**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 managing to be considerably faster using a technique called “Alpha-Beta pruning”. It works just like the standard mini-max algorithm, using recursion to analyse out of all the possible moves each player could take (up to a set stage in the tree of moves known as the ply), but is able to skip analysing certain sections of the tree if it knows (if both players play optimally) that the moves leading to that part 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 it found was 4, it would then find out what the score would be if 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 it knows that blue (if they were playing optimally) would choose the move to lead to the lowest possible score, it knows that red choosing the move on the right would lead to at most a score-3, and therefore, without needing to calculate the final board state if blue chose the move on the right, the program could pick the best possible move to make for red (the move on the left). 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 difference between the player’s scores the program has found if that value is higher (for alpha) or lower (for beta) than what it was previously and it is the appropriate player’s turn (if blue was trying to minimize the score difference, and it found that as a result of blue’s move, the score difference was lower than beta, beta would be set to that value) (at the start of the code alpha is set to -99999999 and beta is set to 99999999 so they can be updated to the first values the program finds.) 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 when both players play optimally.

**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 to get the time the AI took to play a turn in seconds. I then used an app called “Throttlestop” to change the CPU speed to simulate how different computers would handle it. For each value, I then took an average from 3 measurements to avoid large anomalies.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ply and test number | Minimax (3.5 GHz) | Minimax (2.8 GHz) | Minimax (0.8 GHz) | Alpha-beta (3.5 GHz) | Alpha-beta (2.8 GHz) | Alpha-beta (0.8 GHz) |
| 1 (#1) | 0.016 | 0.020 | 0.052 | 0.019 | 0.018 | 0.059 |
| 1 (#2) | 0.016 | 0.028 | 0.064 | 0.016 | 0.018 | 0.058 |
| 1 (#3) | 0.029 | 0.020 | 0.099 | 0.017 | 0.028 | 0.049 |
| **1 (Avg)** | **0.020** | **0.023** | **0.072** | **0.017** | **0.018** | **0.055** |
| 2 (#1) | 0.174 | 0.180 | 0.608 | 0.102 | 0.124 | 0.341 |
| 2 (#2) | 0.161 | 0.164 | 0.603 | 0.090 | 0.135 | 0.365 |
| 2 (#3) | 0.148 | 0.165 | 0.589 | 0.110 | 0.119 | 0.348 |
| **2 (Avg)** | **0.161** | **0.170** | **0.600** | **0.101** | **0.126** | **0.351** |
| 3 (#1) | 1.248 | 1.492 | 4.293 | 0.696 | 0.832 | 1.718 |
| 3 (#2) | 1.266 | 1.535 | 3.991 | 0.646 | 0.860 | 2.208 |
| 3 (#3) | 1.287 | 1.520 | 4.390 | 0.480 | 0.785 | 1.381 |
| **3 (Avg)** | **1.267** | **1.516** | **4.225** | **0.607** | **0.826** | **1.769** |
| 4 (#1) | 11.381 | 13.390 | 32.658 | 3.363 | 4.016 | 9.636 |
| 4 (#2) | 11.960 | 13.354 | 32.828 | 3.446 | 3.984 | 9.028 |
| 4 (#3) | 11.292 | 13.346 | 31.558 | 3.316 | 3.947 | 9.513 |
| **4 (Avg)** | **11.544** | **13.363** | **32.348** | **3.375** | **3.982** | **9.392** |

I then tested the speed alpha-beta AI without randomness to see if there was any change in the time it took to make each turn and tested its score ratio (alpha-beta score/minimax score – not win ration because of how the player going first almost always wins) against minimax (with randomness to make it so the game didn’t happen the same way again and again) (all tested at 3.5 GHz)

|  |  |  |
| --- | --- | --- |
| Starting player | With randomness  (Avg from 5 games) | Without randomness  (Avg from 5 games) |
| Alpha-beta | 1.289 | 1.231 |
| Minimax | 0.750 | 0.791 |

|  |  |  |
| --- | --- | --- |
| Ply and test number | Alpha-beta  (with randomness) | Alpha-beta  (without randomness) |
| 1 (#1) | 0.019 | 0.016 |
| 1 (#2) | 0.016 | 0.017 |
| 1 (#3) | 0.017 | 0.020 |
| **1 (Avg)** | **0.017** | **0.018** |
| 2 (#1) | 0.102 | 0.103 |
| 2 (#2) | 0.090 | 0.101 |
| 2 (#3) | 0.110 | 0.096 |
| **2 (Avg)** | **0.101** | **0.100** |
| 3 (#1) | 0.696 | 0.695 |
| 3 (#2) | 0.646 | 0.692 |
| 3 (#3) | 0.480 | 0.677 |
| **3 (Avg)** | **0.607** | **0.688** |
| 4 (#1) | 3.363 | 3.040 |
| 4 (#2) | 3.446 | 3.059 |
| 4 (#3) | 3.316 | 3.154 |
| **4 (Avg)** | **3.375** | **3.084** |

Average time taken for the first move (seconds)

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

The AI with Alpha-Beta pruning implemented performed exactly the same moves as the standard minimax AI when running at the same ply with no randomness added (the randomness in the final code was added to simply make games more interesting with a ± up to 0.4 ensuring that the AI still performed the best possible moves as the state score is always an integer and 0.4\*2 is still less than 1, however, due to limited ply, the randomness could sometimes lead to anomalies in which AI won.) Despite no reduction in ability, alpha-beta AI managed to be much faster than standard minimax, especially in the higher plies where it had more opportunity to prune large amounts of data.

The effect of randomness in the alpha-beta AI

The effect of implementing randomness into the alpha-beta AI from a speed standpoint is almost negligible with the largest difference between results at 3 Ply being almost entirely due to the anomalous result of 0.48 for the AI with randomness implemented.

The performance of the AI is also almost exactly the same with the negligible differences also likely due to anomalies instead of any sort of pattern (both score-ratios are within 0.06). Therefore, it can be concluded that the implementation of randomness to the alpha-beta AI has entirely negligible effects other than it making different decisions when in the exact same situation making every game unique.

Average time taken for the first move (seconds)

**Results:**

Especially at lower CPU speeds, alpha-beta pruning allowed the AI to reach much faster speeds (at 4 ply it managed to overtake minimax at 3.5GHz despite being run at lower than a quarter of the speed.) However, the increase in time with each jump in ply is still exponential and soon reaches unreasonable speeds at higher plies (the first turn for the alpha-beta AI at 3.5 GHz is still over 13 seconds.)

**Conclusion:**

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. Randomness being implemented would also be beneficial as it would stop the other player being able to learn anticipate exactly what the AI will do next from a given situation with a negligible impact on turn-making speed. 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 with the CPU at 3.5 GHz takes 13 seconds.) The decision between the ply being at 3 or 4 would definitely depend a huge amount on the computer being used however, as at 0.8 GHz, a ply of 4 takes almost 10 seconds, much slower than the 3 and half seconds when at 3.5 GHz. Therefore, if the maximum amount of time that could be taken for a move was 4 seconds, then if computer’s CPU was running at more than 2.8 GHz it should use a ply of 4, but if the computer was any slower, then it should use the alpha-beta AI with a ply of 3.