Project 2 Connect Four with AI

David Bush

CS 470 Soule

June 1st, 2021

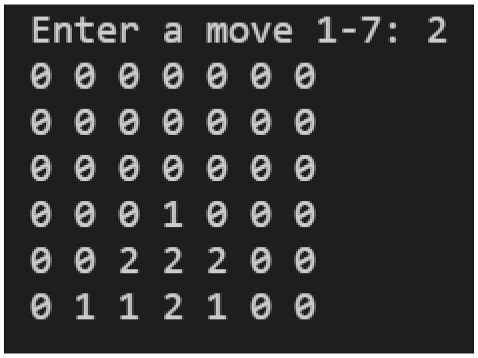
Abstract: In this project I built a simple Connect Four game that takes turns between the computer player (AI) and the human player. The AI determines each move using an adversarial search that implements a min-max algorithm and uses an evaluation function to determine the value/utility of each state of the board. For example, when the next move would result in a win for the human player, that move would have a utility of -1 and when the move would result in a win for the AI player the utility would be +1. The AI would always take the path with the higher utility and avoid the paths that would result in a loss since that was how the minmax function was implemented. After testing various board states, I found that the AI would always take the move that resulted in a win, and the AI would always take the move that prevented the player from winning. The more challenging part was determining the in-between states that did not directly result in a win, loss, or draw. For this I had to find a way to score each board state with the goal of showcasing how far ahead the AI was.

Algorithm: The minmax algorithm works by searching each possible move for the AI which results in new board states and then it searches each possible move for the player which results in more new board states, this continues switching back and forth from the AI and player until it there is a win, loss, draw or the max depth is reached. The value of each node in the tree is a utility value which is determined by the evaluation function that scores each board state. The maximizing player (AI) returns the higher utility value for each node, and the minimizing player (human) returns the lower utility value. If working properly this results in the AI selecting the ‘best’ move that should lead to the ‘best’ outcome for the AI.

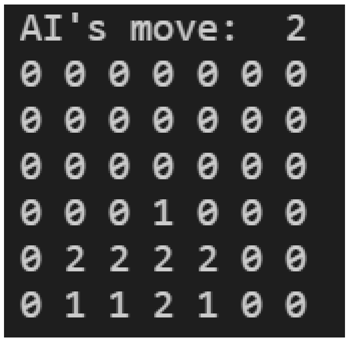
I first started using a max depth of 6 for the search and found that it was taking anywhere from 30-60 seconds to make a move which was too long. I changed the max depth to 5 and found it to be right around 10 seconds per move or less. Most of this time was probably being used in the evaluation function which had to be called at every node. The evaluation function is used to detect how favorable a particular board state is for the AI. The evaluation function first checks if the game state is a win, loss, or draw, returning the associated utility if true. Next the evaluation function would count the number of pieces in a line, checking all the horizontal, vertical, and diagonal parts, giving each a score. For example, if there was two of the same pieces in a line it might return 20, and if there were three of the same pieces in a line it might return 65. I also found that the center column usually led to favorable positions and made each piece in the center worth 10. This score would be a combination of the different sections added together and then it would be returned and used as the utility value in the min and max functions.

Example Screenshots:

AI takes win:

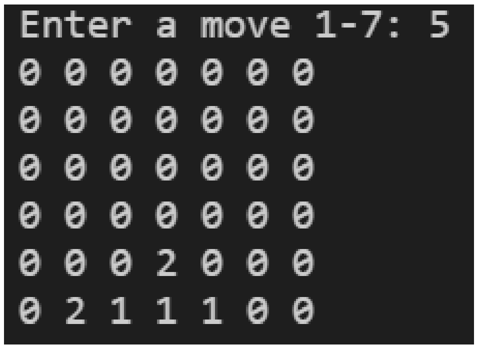


```
Enter a move 1-7: 2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 2 2 2 0 0
0 1 1 2 1 0 0
```

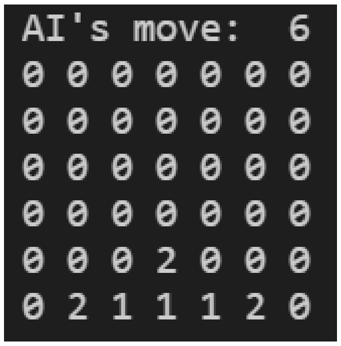


```
AI's move:  2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 2 2 2 2 0 0
0 1 1 2 1 0 0
```

AI prevents win:

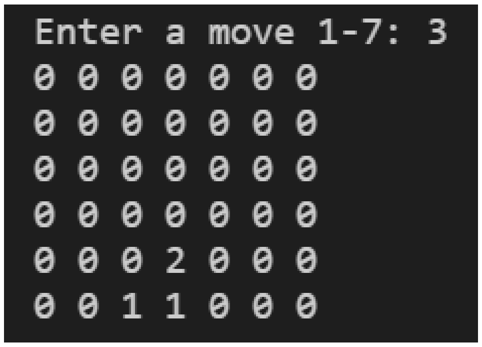


```
Enter a move 1-7: 5
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 2 0 0 0
0 2 1 1 1 0 0
```

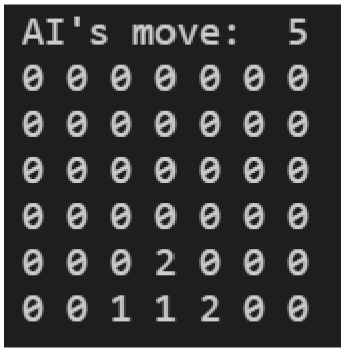


```
AI's move:  6
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 2 0 0 0
0 2 1 1 1 2 0
```

AI does not allow player to set up trap where all moves lead to win:

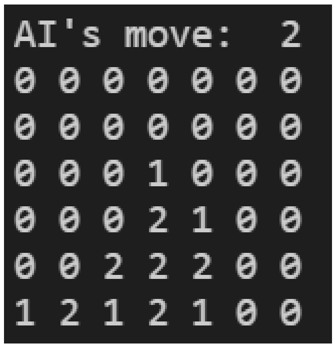


```
Enter a move 1-7: 3
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 2 0 0 0
0 0 1 1 0 0 0
```

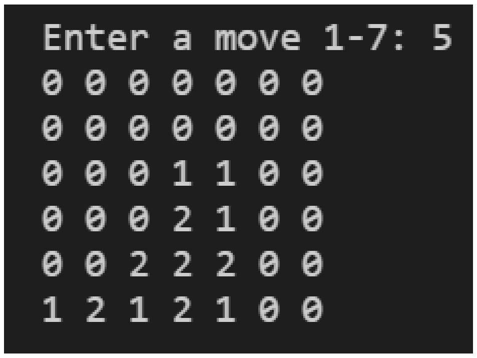


```
AI's move:  5
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 2 0 0 0
0 0 1 1 2 0 0
```

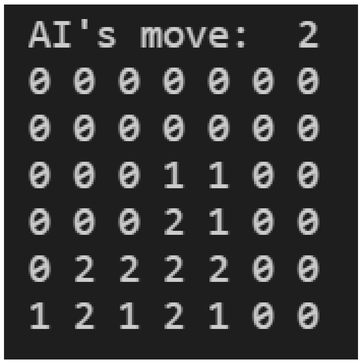
AI sets up a trap and wins:



```
AI's move:  2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 2 1 0 0
0 0 2 2 2 0 0
1 2 1 2 1 0 0
```

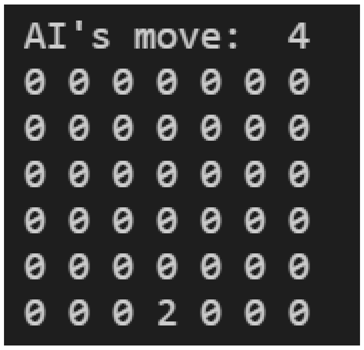


```
Enter a move 1-7: 5
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 1 0 0
0 0 0 2 1 0 0
0 0 2 2 2 0 0
1 2 1 2 1 0 0
```



```
AI's move:  2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 1 1 0 0
0 0 0 2 1 0 0
0 2 2 2 2 0 0
1 2 1 2 1 0 0
```

AI takes center:


```
AI's move:  4
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 2 0 0 0
```

Conclusion: For this project I implemented a Connect Four AI using a minmax algorithm and evaluation function. The AI was consistent at selecting the winning move if one existed and at blocking the players winning move. This made it somewhat challenging to play against since the AI was not making simple mistakes. After playing more and more games I found that the AI was also good at setting up traps that could ensure a win multiple moves ahead and it appeared to be just as good at reading opponent traps. The evaluation function did appear to be working well in the games I played. I think the AI’s weakness was the time it was taking to make a move. At the beginning of a game, it would take longer 10-15 seconds but after more moves had been executed, it would start making decision faster around 5 seconds per move (max depth 5). The time it was using was based on how deep the branches it was looking down, which is why the time was less towards the end of a game when the branches were shorter. This also can be changed by lowering the max depth but will ‘weaken’ the AI since it is not looking as many moves ahead. I found it was difficult to measure exact examples like whether the AI was taking the longest or shortest path and I was unable to determine whether it was. I also ran into the most problems with setting up the minmax and evaluation functions. I was getting quite a few errors from not checking first to see if there were available moves, and with adjusting the values in the evaluation function. Early on there was one point it appeared that the AI wanted the player to win but it was because it was returning the wrong value for the players win. The improvements I would focus on would be figuring out how to use less time in the evaluation function, adding alpha-beta pruning, and playing around with the evaluation more to optimize win percentage. Overall, I believe the AI was performing well at choosing moves and it was using the minmax algorithm correctly.