# Connect Four, A Disappointing Game

William D. Gibson

Northwest Nazarene University

COMP 4220

Prof. Enoch Levandovsky

02 April 2023

## Abstract

Starting with a minimax algorithm, this agent was developed fairly quickly, and was fixed and ready to turn in on time using Enoch's code as a reference for refactoring. From here, I started attempting to implement alpha-beta pruning, in hopes of getting the algorithm to work much better than before. This journey to implement and get this working was long, frustrating, and, in the end, fruitless. Thus, I come here, empty handed, with an algorithm that works worse now, being turned in a month late.

*Keywords:* First line indented, lorem, ipsum, dolor

# Connect Four, a Disappointing Game

Connect Four is a solved game, meaning that the first player can win 100% of the time no matter what. Knowing this information, it should be obvious that any artificial intelligence agent worth its salt should be able to win 100% of the time while going first. With this idea in mind, I started this project. Building up a basic minimax algorithm was mostly simple, and the minor bugs I had were remedied when the code was refactored using Enoch's minimax algorithm. This AI was good, but still beatable, and so, I started improving, by working on alpha-beta pruning. This took a while to say the least, since I was still working under the assumption that I could hit a 100% win rate with the AI against me. This never ended up happening however, as the AI seems to have blind spots, leading to a human player being able to capitalize on these blind spots. And then, when I went to test accuracy, the problems came to light. But first, I should probably explain this project in its whole.

## Background

### Minimax Algorithm

A minimax algorithm is a fairly simple AI algorithm for turn based games. The algorithm takes a state, it figured out each possible move for one player, then takes all of those states and expands every possible move for the next player, then it takes all of those states and expands every possible move for the next player, and so on. It "rates" each of these positions, and determines from there, what the best move is for each player in any given state, minimizing losses and maximizing gains. This kind of algorithm is what many chess engines use when determining what the rating of a position is, and determining the best move in a given position.

### Improvements to the Minimax

A number of improvements can be made to this algorithm. The clearest drawback is that looking at each move in a position, then each move in those positions, and so on, is the fact that the number of positions it is checking expands exponentially. This means the easiest way to improve this algorithm is to limit the number of moves it looks at. The most optimal way to do this is with alpha-beta pruning. This is a complex topic, but to simplify it as much as I can, this algorithm checks to see if any of the paths are obviously bad for one player or the other, and prunes them before the algorithm can even explore these paths. This improves the number of iterations that the algorithm can search while still being able to find a good move in a reasonable amount of time.

## Methods

### Implementing a Minimax Algorithm

Most of the connect four game was built already, allowing us to jump directly into the development of the minimax algorithm. The implementation of this algorithm began with creating an “expand” function. This function takes a number of parameters. The most important thing that is passed in is the snapshot of the board. This function checks first to see if either player is currently winning, doling out a win or loss appropriately. From here, we use another function to check every single move in a position, see if any of them win, and then recursively do all of this again, until it reaches a maximum depth. This depth is currently set to 5, since any further than this takes too long to produce a move.

### Implementing Alpha-Beta Pruning

The implementation of alpha beta pruning was literally 4 lines of code. While maximizing it sets alpha to the max of the current alpha and the eval of the best move. If this alpha is better than whatever beta is set to it will break out of the loop and quit looking for better moves on that path. It does the same for beta when minimizing. This should theoretically mean that the depth can be expanded deeper while keeping things moving at a relatively quick pace. When this was implemented in my practice, i was able to continue expanding the depth, without care, and I never got into a situation where the game slowed. AI moves were played in under a second every time, even when expanding the depth to 42 (the maximum number of moves that can be played).

## Analysis

### Analysis of the Minimax Algorithm Alone

The game as is with just a minimax algorithm runs well, but can only expand to a depth of 5 before slowing beyond reasonable. It works reasonably well, but seems to work incredibly selfishly, leaving blind spots that can be exploited by anyone who understands the game of connect four. This basically means that the AI will completely miss that the player is 1 move away from winning, and will make some other move to better its game, while losing on the next move. I have managed to pull off situations where I have had multiple sets of 3 in a row lined up, and the AI has missed all 3 of them in favor of giving itself 3 in a row. While this isn’t a major problem, it does mean that the AI can be beaten 100% of the time if the player knows how to play connect 4. This isn’t all bad however, as against the random bot, this minimax algorithm, out of 25 tests playing first and 25 tests playing second, the bot won every single time. So, while the bot seems to have blind spots, they are only against a player when playing in the pygame interface.

**Analysis of adding Alpha-Beta Pruning**

Once alpha-beta pruning is added, it is apparently possible to expand the depth infinitely without slowing at all. This algorithm performs marginally better. I have actually managed to lose to this version of the bot, which is a good measure of how well it works. The biggest disappoint though, was the fact that there were still a number of blind spots. I never managed to get 3 sets of 3 in a row, but I did get 2 sets on a couple of occasions. Against the random bot, once again 25 games going first, 25 games going second, the issue at heart was found. The very first game was a loss for the bot, and it was immediately clear why. Messages of "Error, player 1 code crashed" were being thrown left and right. This is the start of my woes. This was the reason why the depth could be expanded infinitely, because beneath the surface, the code wasn't working at all, so it was defaulting to just a random bot against a random bot. Lightly testing this further showed that the alpha-beta pruning bot against the random bot was about a 50-50 split on which won each game.

I never found a solution to this problem. It was only 4 lines difference, and yet I simply couldn't find the issue. In a serious attempt to find the issue, I ran the code through ChatGPT, asking it if it could find the problem. Sadly, not even AI was able to find the problem. After struggling through this for a while, and then getting busy and dropping the project for a while, I picked this back up over break. I worked to try to find the issue, but once again came up fruitless. After working on this for an extra month, I decided the time had come to call it quits, mildly frustrated, confused, and overall, a bit dejected.

## Conclusion

### Overall Concluding Thoughts

Overall, this project was a roller coaster. Recognizing the blind spots was good, but there was never any time to attempt to fix this. Realizing that the minimax algorithm did in fact work perfectly as expected against the random bot was a huge win, though toned down since it was mostly refactored to match Enoch's code (if you want to run the minimax algorithm alone, use EnochPlayer, since that is my algorithm, pre-implementing any alpha-beta pruning). Once alpha–beta pruning was added, things were looking up, but it still seemed that the blind spots existed. Then, the absolute crash as I realized the code just doesn't work was crushing. Then, having to drop the project for a couple of weeks to make sure I kept up with other project was refreshing, though worrying as the project fell more and more behind. Then, it just felt sad to finally come back to it, work, and give in.

### And Yet There is Hope

As dejecting as it is to turn in an unfinished project, there may still be hope. Let's look into further things we can do to finish out this project and make it work as intended. Firstly, on the alpha-beta pruning, I believe the easiest way to do this is simply to refactor the code around it. Tacking this enhancement on at the end of the project, while it may be the most intuitive and easiest way to do things, may have led to more problems. It is probably a good idea to fix this before anything else. If however, we wanted to look in a different direction, and give ourselves an opportunity to step away from this issue (which I very desperately need), we could look into the blind spots. I have not done any testing on whether this is an issue that only exists when playing against a human using the pygame interface, or if this would continue to happen if we used the terminal interface instead. Or maybe this is a different problem entirely, where moves are getting expanded incorrectly. Either way, digging deeper into the root cause of why this algorithm has blind spots is a good idea, and will give us tangible improvements that we can make to improve the AI.

**Concluding Remarks**

Overall, this was a good project. This would have gone better if I had more time to work on it, but that is all in the past. All the minor complications that came up along the way allowed me a chance to learn something new about this algorithm, and even reading through and refactoring to the "correct" algorithm that was shown off in class, it was a learning experience then. And yes, even when the major program ruining complications came up, it was a good time for me to learn to take a step back, and realize that even if I can't get this small enhancement working, the main algorithm will still absolutely crush any unwitting soul who dares test it. In conclusion, good project, lots of fun, even when suffering a crushing defeat.