Michelle Prior (map398) and Sheba Sow (ss2956)
Professor Selman
CS 4701 - AI Practicum
16 December 2022

**Connect 4 AI Bot**

## 1. Introduction

Game artificial intelligence (AI) refers to the development of computer programs that are able to play games against human or computer opponents (Reed 2021). Game AI allows us to study strategic gameplay/game theory by allowing us to look at different algorithms that can be used to find an optimal solution to a game. In addition, it helps us to understand human thinking, as it allows us to study how people make decisions and why people make the decisions that they do. Finally, they can be performed in a way that is lower risk, compared to more dangerous fields such as autonomous vehicles, opening up more opportunities for studies. There are many different techniques and algorithms that can be used to develop game AI, including minimax and alpha-beta pruning. Game AI is a widely studied area of AI from Chess AI on Chess.com to the CPUs developed in video games like Pac-Man.

One great example of a game with several strategies and heuristics is Connect 4. It is played on a 6 by 7 grid and consists of two players. Each player has chips of a designated color (red and black) which they will use to mark their spot. For their turn, the player selects one column slot to drop their chip down. The next player will do the same and if they choose the same column slot, their chip will fall right above the other players. The goal of the game is to get 4 of your chips next to each other either vertically, horizontally, or diagonally. There is already a solution for a perfect AI for Connect 4 on traditionally sized boards. However, to add to its complexity, we hope to expand upon the traditionally “perfect” AI for Connect 4 (Pons 2016) by modifying it to work with boards of varying dimensions, requiring more optimizations and decisions to be made by the AI.

## 2. Prior Work

Minimax is a well-known algorithm that is often used in game artificial intelligence (AI) to determine the optimal move for a player in a two-player, zero-sum game such as Connect Four (Javatpoint). The algorithm works by considering all possible moves for both players, and then determining the best move for the current player based on the minimum value that the opponent could achieve in their next move. To implement minimax in Connect Four, you would start by creating a game

tree that represents all possible moves and their outcomes for the current game state. Each node in the tree represents a game state, and the children of a node represent the possible moves and resulting game states that can be reached from that state. To evaluate each game state, you can use a heuristic function that assigns a score to each state based on certain criteria. In our game in particular, the scoring is determined by checking how many future wins this move could be tied to. For example if we had a 6x6 connect 4 grid where the current state was:

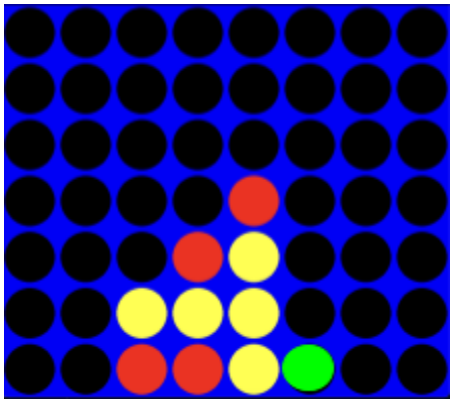

**Figure 1: Gameboard UI With Example Move**

The AI would evaluate each open spot based on the potential wins this move could make. So for coordinate (7,6) (in green), this spot could bring the game closer to winning at row 7, column 6 and a less direct win for row 6 and positive slope diagonal intersecting at (6,6). The AI also considers the number of winning combinations that are available to the opponent, and the position of the pieces on the board.
Once the game tree has been created, the minimax algorithm can be used to traverse the tree and determine the optimal move for the current player. It does this by starting at the root of the tree and recursively evaluating the children of each node, alternating between maximizing the value for the current player and minimizing the value for the opponent. Once the algorithm reaches the leaf nodes of the tree (the final game states), it can backpropagate the scores up the tree and determine the optimal move for the current player based on the minimum value that the opponent could achieve in their next move.

Lots of research has already been done in this area for the minimax function in a lot of different game settings from Tic-Tac-Toe, to Checkers, to even Connect 4. There is even a perfect AI solution using Minimax for a traditionally sized Connect 4 board, that can be seen in many different language implementations from C++ to Python. However, the minimax algorithm alone is not optimal enough in

order to solve a Connect 4 game where there are increased dimensions. We will explore in the method section on why and how we are going to take it a step further by implementing a modified version of Alpha-Beta pruning built on top of the Minimax algorithm in order to allow for ordering heuristics and removal of unnecessary branches.

## 3. Methods

Minimax provides the optimal strategy for playing connect 4 on a traditional 6x7 board, but if we expand the game to larger dimensions, there will be more possible moves to play, each with their unique score and responses from the opponent. So we needed an algorithm to minimize the branches we transverse through. We decided to use a modified version of alpha-beta pruning since it is often used in conjunction with the minimax algorithm to improve its efficiency in two-player, zero-sum games such as Connect Four. The idea behind alpha-beta pruning is to skip large portions of the game tree that are not relevant to the current search, which reduces the amount of work that needs to be done by the minimax algorithm.

To implement alpha-beta pruning in Connect Four, it creates a game tree that represents all possible moves and their outcomes for the current game state (GeeksforGeeks). Then, as you traverse the tree using the minimax algorithm, you can use the alpha and beta values to keep track of the best scores that have been achieved so far by the current player and the opponent. The alpha value represents the maximum score that the current player is guaranteed to achieve, given the current game state and the moves that have been made so far. The beta value represents the minimum score that the opponent is guaranteed to achieve, given the current game state and the moves that have been made so far. As you traverse the tree, you can use the alpha and beta values to eliminate branches of the tree that are not relevant to the current search. For example, if the current player's score is greater than or equal to the opponent's beta value, then it is not necessary to search the children of the current node, because the opponent cannot achieve a score that is better than the current player's score. If the opponent's score is less than or equal to the current player's alpha value, it’s not necessary to search the children of the current node because the current player cannot achieve a score that is worse than the opponent's score.
In our code we divided the algorithm into multiple functions.

We hope to then optimize the traditional alpha-beta pruning by introducing an ordering heuristic as well as exploring the impact of search depth on the performance and speed of the bot as well. The great thing about ordering heuristics is that they do not impact the accuracy of the program, and also offer extra optimization to searching the tree. By using ordering heuristics, we are able to search the earlier parts of

the tree, and make earlier cutoffs for suboptimal moves in the beginning, rather than having to search these branches and wasting extra efficiency. In addition, we want to explore the impact of changing search depth on the tradeoffs between efficiency and performance, so that we will be able to use the one that is . There is a chance that if we were to increase our depth by too much that it would not be able to run on our processors, but if it's too small that we would not be able to look at enough potential branches.

The major functions of our bot are Minimax, get_valid _location and drop piece. Minimax function checks if there is a winner/tie/no open space left. If so, it ends the game, else it determines the next best move using scoring and alpha beta pruning. In this algorithm there is a helper function Get_valid_location which gets coordinates of where both can move and Evaluate_window which calculates a score for these moves by determining how much this chip location will increase bot's chance of winning as described previously. Then there is the function drop_piece which updates the board graph so that it can be displayed. In addition to the functions for the algorithm, we created a function draw_board that uses pygame to draw rectangles and circles for the connect 4 frame and new chips.

In order to make sure that our algorithm is in fact the most optimal, we can compare it to two other methods by having the bot face off against these other known algorithms. One baseline algorithm that we would have for the sake of a control group would be a random algorithm that would pick a random spot to place a piece each time. The other algorithm that our modified Alpha-Beta Pruning bot would go against would be the standard Minimax algorithm to make sure that the modifications to our heuristics and search depth will still make it more optimal than that algorithm that can be perfect with normal size boards. To quantitatively compare all these algorithms, we will calculate the winning percentage of these algorithms when paired against each other 100 times, and compare them to determine which algorithm is best against both of the other two respective algorithms. For testing against human players, we needed an efficient way to run and collect information on games. We used the pandas library to keep track of the user name, time, number of moves, skill level, and outcome of each game, and then can later analyze the totals and averages of these results and how that relates based on different size boards.

## 4. Results

For analyzing and demonstrating that our algorithm is superior to the Minimax and Baseline Random algorithm, we created a heatmap with the percentage of wins for each algorithm vs. the other ones. For standardization, we ran the test through 100 times for each pairing and used a larger board for testing (8x10), since we are trying to see how the Alpha-Beta pruning is able to cut down decisions on this board. Figure 2 shows the results of this analysis, with green representing higher winning percentages and red demonstrating lower win percentages, and the winners being the horizontal choices and the losers being the vertical choices.

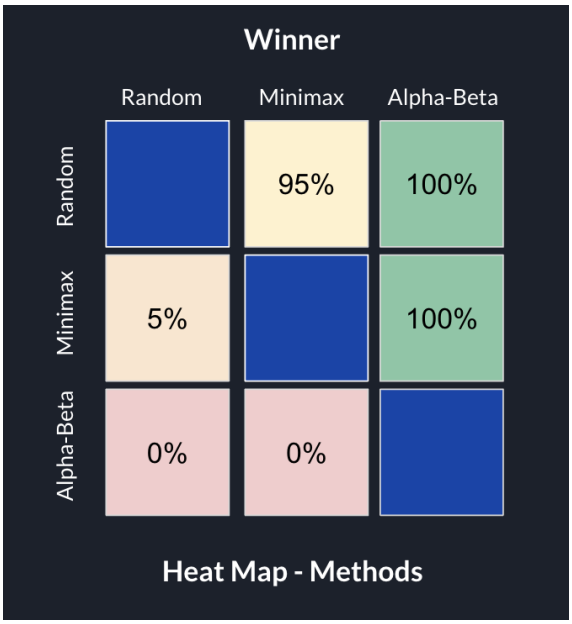


**Figure 2: Heat Map for the Three Measured Algorithms**

In terms of the human testing, the image below showcases the finished result of our GUI that was built in Pygame, with inspiration from Keith Galli's Connect 4 GUI (Galli 2019) and Msaveski's Connect 4 Player (Msaveski). Figure 3, which can seen on the next page, demonstrates the winner banner for when the AI (yellow pieces) wins the game by getting four in a row.

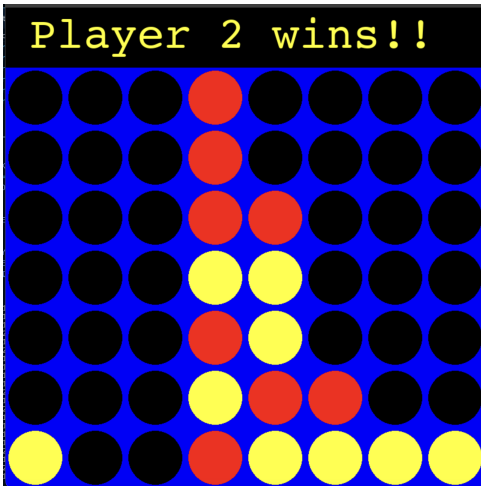


**Figure 3: GUI Visualization with AI Win**

For human testing we tested how our game works on multiple board dimensions, by running 200 different games with 25 different participants. We then kept track of the stats for each game through our Pandas dataframe. Table 1 demonstrates the overall results of the testing based on all the data collected in the data frame, regarding average time, moves, and player and AI wins for each grouping of board dimensions. These were all calculated from the data in our CSV file that was transformed from a Pandas dataframe, which demonstrates the data we collect on our participants during our testing phases, which include the name, skill level, board dimensions, time, moves, and outcome for each game that they are playing.

n = 200 Games

Avg Skill Level = 3.04/5

Search Depth = 5

| Board Dimensions | Avg Time (s) | Avg Moves | Player Wins | AI Wins |
|---|---|---|---|---|
| 6x7 (Baseline) | 24.95 s | 19.06 Moves | 0 Wins | 50 Wins |
| 7x8 (Larger) | 27.17 s | 20.82 Moves | 0 Wins | 50 Wins |
| 5x6 (Smaller | 24.11 s | 18.90 Moves | 0 Wins | 50 Wins |
| 6x6 (Square) | 24.04 s | 21.64 Moves | 0 Wins | 50 Wins |
| Total/Avg | 25.07 s | 20.11 Moves | 0 Wins | 200 Wins |

**Table 1: Human Test Results - Broken Down By Board Dimensions**

## 5. Discussion

In terms of comparing our modified Alpha Beta algorithm for our bot to other known algorithms, our bot performed with a 100% win rate when we ran each algorithm against each other 100 times (n=100 times). Coming in next, minimax would have the next highest percentage of wins with 95% against the random algorithm and 0% against the modified Alpha Beta algorithm, with random coming in last with . This lines up with our initial hypothesis and thinking about this, since the pruning should eliminate suboptimal paths and ordering heuristics should impact how early these paths are removed, and makes a strong case for our modified Alpha Beta being the strongest algorithm of the three. However, one thing we aren't completely sure about is if we increase the number of iterations we run significantly (n=1000 times), could there be an increased chance for an anomaly in terms of our "perfect" AI? As we increase the number of runs, there's a higher likelihood that our algorithm/bot may lose some games by chance or because there is a flaw in one aspect of our strategy, in this case no longer making our algorithm "perfect". Unfortunately, there are physical limitations in testing this, as running Pygame that many times is computationally expensive and requires more computing power than our computers can provide. However, if we were able to run this on more powerful processors, it would be interesting to see if this "perfection" holds to more tests against the other algorithms.

In terms of comparing our modified Alpha Beta algorithm for our bot to other competitors, our bot performed with a 100% win rate against collectively intermediate players, with the factor that n=200 games in this situation. The fact that no player of beginner or intermediate skill level was able to beat our bot in 200 games provides strong evidence that our AI bot meets our requirements for players up to the intermediate skill level. However, there are some things that should be considered further in the human phase of testing. Since our average skill level was 3.04, this means that our players tended to fall more toward the middle in terms of skill. This leads us to wonder what would happen if we increased our sampling size (i.e. n = 1000 games) and include more higher skilled players in the advanced and expert ranges. Would our Bot AI still be perfect or would a skilled player be able to determine a heuristic that causes our bot to lose or tie with it? Theoretically and considering probability, this would make sense and this situation also considers factors we are not able to test in our 200 person sample. However, due to time constraints and the audience of the people taking the test all being students meant that we were not able to run that many tests and necessarily find people that meet the criteria of being that skilled at Connect 4. We hope that in the future, in a different setting and with more time we would be able to observe how our AI would do against some top Connect 4 players beyond people who just play for leisure. Another assumption we have to consider with this is that the skill level of our participants is truly representative of their actual skill levels. Since terms like "beginner" and "average" were thrown around loosely, it was up

to each participant's interpretation to determine. This could lead to some players who consider themselves higher or lower skill to not demonstrate that same level of skill relative to other players who went up against the bot, which could lead to inconsistencies. In the future, we hope to make it clearer what we mean by skill level and use it in terms of relativity to the average player.

One main consideration that had to be taken into account for our bot/algorithm that had an impact was a tradeoff between speed and performance with search depth. This is something that we continuously kept experimenting with while we were testing. With the increasing search depth, it allows the search to go down to further branches in order to consider more potential branches it could take in terms of decisions. This could mean that the AI could make even better decisions. However, at the same time, this can significantly lower the speed and efficiency of the program, and especially in the case where we were running the different algorithm tests against one another 100 times, this would not be optimal. This is why we ended up choosing our search depth to be 5, as it allows the program to observe more board states than with 4, but also making sure that it doesn't slow down our program too much as it does with 6.

Finally there are three main interesting findings that we discovered through simply observing our human testing. One thing that we noticed through wanting to try to test people on larger boards (i.e. 8x10) is that they found them to be disorienting and difficult to process all at once, prompting them to not want to strategize or basically give up against the bot. This physical limitation held us back a bit, as we were required to remove boards of 8x10 or larger from our human testing because it was not showing accurate statistics of each match. In addition, another similar issue occurred in terms of us telling people that we are trying to test our "perfect" Connect 4 AI. People tended to either get psyched out or give up more easily after hearing us say that, which prompted us to not mention that for future tests, to try to make the testing environment more controlled. Finally, one last finding was that there were situations that we observed where even though the bot eventually wins, they had a chance of winning earlier on. Even though having each move be "perfect" was not a part of our requirements, it makes us want to further reconsider the impact of the search depth for our Alpha Beta pruning, allowing for the pruning to consider more decisions that may be deeper down the tree. We could also take a look at our ordering heuristic again to see if there is another way we can order it to find the optimal move faster in the future, while still maintaining the requirement of always being able to win.

## 6. Conclusion

There are three main takeaways from the Connect 4 AI Bot project. One main takeaway is that we were successful in creating an algorithm that is effective in winning against other known algorithms, such as a random baseline algorithm and Minimax algorithm. This claim could be further strengthened by increasing the amount of iterations in which the algorithms could be tested against one another. In addition, our bot proved to be an effective competitor in human trials with players up to intermediate skill levels. We hope in a different setting to be able to find stronger competitors to have our bot compete against us, to further validate our claim. Finally, one last main takeaway from our observations is that even though our bot was able to win in all the games it played, there were still situations where the AI missed a chance at winning or an optimal sub move that could have set itself up for quicker win. Reflecting on this still apparent issue, we believe that modifying search depth in the future could help to mitigate some of these issues, as it would be able to find solutions that might be farther down the tree.

While we were able to find some fascinating findings through our research, there is a lot more to be explored in terms of the Connect 4 AI Bot. One potential area of growth for the future would be in gathering a larger sample size of human test data. By being able to sample a larger group of people, we will hopefully be able to have a larger amount of highly skilled players (players with skill level 4 and 5), which would allow us to see if our bot can be just as effective against players with more robust strategies. In addition, another interesting area to explore would be to make the AI bot work for Connect 4 if we were to add an additional dimension. Rather than just having the simpler X by Y board that we see in our experiments, we would implement a Z coordinate, which would result in more strategies and possible moves to consider for getting four in a row. Finally, one last area of expansion from our project would be to attempt another implementation of our Connect 4 AI Bot would be to use reinforcement learning. Rather than just repeating the same algorithms and heuristics, through reinforcement learning, we could see how the AI generativity improves over time through reward based training systems. We can compare this method to see if the strategies developed by the AI are even stronger than that of which we have developed.

Works Cited


"Artificial Intelligence: Mini-Max Algorithm ." Www.javatpoint.com, Javatpoint, www.javatpoint.com/mini-max-algorithm-in-ai.

Galli, Keith. "Develop an AI to Play Connect Four - Python Tutorial." YouTube, YouTube, 29 Mar. 2019, www.youtube.com/watch?v=8392NJjj8s0.

"Minimax Algorithm in Game Theory: Set 4 (Alpha-Beta Pruning)." GeeksforGeeks, 14 Nov. 2022, www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/.

Msaveski. "MSAVESKI/Connect-Four: Ai Connect Four Player." GitHub, www.github.com/msaveski/connect-four.

Pons, Pascal. "How to Build a Perfect AI." Solving Connect 4, 1 May 2016, www.blog.gamesolver.org/solving-connect-four/01-introduction/.

Reed, Niamh. "Game Ai: Why Do We Teach AI to Play Games?" ThinkAutomation, 11 Feb. 2021, www.thinkautomation.com/bots-and-ai/game-ai-why-do-we-teach-ai-to-play-games/#:~:text=In%20short%2C%20gameplay%20helps%20researchers,natural%20language%20processing%20(NLP).