Connect 4 AI

By:

Brian Guiterrez

Konrad Phillips-Lenz

Pedro Rodriguez

Presented for:

CSCI Senior Design

University of Texas Rio Grande Valley Computer Science Department

Under supervision of:

Dr. Zhixiang Chen

May 10, 2023

Table of Contents

Abstract

In a project that took place over the course of a semester, The goal is to create an artificial intelligence that could simulate games of Connect 4 to consistently win against human opponents. The program has a functional GUI, contains quality of life features, and ultimately be ported to a web server. While still being able to run on low-performance computers, the program delivers on beating human players while running quickly using a Minimax algorithm with alpha-beta pruning. The future of improving this project will involve making use of Shannon-C theorem to take advantage of predetermined rules and strategies, rather than strictly brute force computation methods.

Introduction

The aspirations of this project began in an attempt to create an AI that can sufficiently win against human opponents. This was inspired by the massive achievements conducted in the chess and gomoku gaming spaces. There, AI like Stockfish and AlphaGo have been sweeping competitions using vast neural networks. We decided to take on the challenge of making an unbeatable AI, but instead went with the more casually played Connect 4.

Connect 4 was published in 1974 by Milton Bradley and the Hasbro company. It was based on a long tradition of four-in-a-row games that went by many names and spanned many cultures, becoming a household name to this day. It is a 2-player game, usually spanning no more than 10 minutes, where players take turns dropping discs into an upright, 7x6 grid. The first to make a straight line on 4 pieces wins the match. The game definitely has a lower skill ceiling than something like chess, but still has a great deal of strategy lying underneath the surface.

The game was individually solved by both James D. Allen and Victor Allis in 1988 and has since had an entire dataset developed that can guarantee a win from the first player as soon as the game starts (Allen, 2010). However, this is a very large database and not something feasible for a quick consumer-level program. With 42 cells and 2 players, there would be over 4 trillion possibilities to account for. However, as will be explained later in the report, there are many tricks to get an unbeatable AI working at low cost without having to reference such a massive dataset.

This project came about from multiple motivations. For one, it would be using algorithms to solve a real-world problem. With how easy Connect 4 is to pick up, you would be able to see results in real time with quick matches against fellow humans. Through this project, there is also intention to explore the hidden depth of the game and discover and identify the key strategies to winning the game. It is hoped that this project can be used by fans of Connect 4 to train themselves and get better by playing against a computer that always challenges them to the fullest as well as develop Shannon-C theory to be applicable for board game computations that traditionally prosper through brute force computations.

Background

Connect Four is a two-player abstract strategy game played on a vertical grid with seven columns and six rows. The game starts with an empty grid, and each player takes turns dropping one of their colored discs into a column until one player achieves four discs in a row, either vertically, horizontally, or diagonally. Connect Four was invented by Howard Wexler and Ned Strongin in 1974 and became a popular game that has been studied extensively by researchers and game theorists.

In Connect Four, the minimax algorithm with alpha-beta pruning is a common method used to create an artificial intelligence (AI) opponent. The minimax algorithm is a recursive search algorithm that explores the game tree to find the optimal move for the current player assuming that the opponent plays optimally as well. Alpha-beta pruning is a technique used to reduce the number of nodes evaluated by the minimax algorithm by eliminating branches that will not change the final result. This combination of algorithms can create a strong Connect Four AI that can compete with human players.

Furthermore, researchers have explored other methods to improve Connect Four AI, such as the Shannon C theory. Shannon C theory proposes that the value of a game can be determined by considering the entire tree of possible moves and outcomes, rather than just the immediate next move. This approach can lead to more optimal moves and strategies in Connect Four.

Pygame, a Python module designed for game development, was used to create a user-friendly interface for the game and for AI development. Pygame_menu, a library built on top of Pygame, was used to create a menu and control the game flow. These libraries made it possible to develop and test the Connect Four AI in a practical and accessible manner.

With this background information, we can explore the development and implementation of a Connect Four AI using the minimax algorithm with alpha-beta pruning, as well as the improvements made to the AI using the Shannon C theory and other information acquired through the solutions of the game.

Goals

The development of this Connect 4 AI project is driven by three main goals. Firstly, we aim to create a functional player game of Connect 4. This requires the development of a graphical user interface (GUI) that can display the Connect 4 game board and allow players to make moves by clicking on the appropriate column. This will also involve implementing the rules of Connect 4, such as checking for winning or draw conditions after each move.

Secondly, we strive to create a computer player that can compete against human players with a high degree of success. To achieve this, we will use common techniques in game-playing AI, such as the minimax algorithm and alpha-beta pruning. We will also incorporate the findings from the Connect 4 solution paper and explore the potential of the Shannon-C theory as a hybrid

approach to determining the best possible move without a database of brute force searched positions. Through iteration and testing, we aim to develop an AI that can play significantly more times than it loses against human players.

Finally, we aim to optimize the performance of our Connect 4 AI on low-performing computers. While many of the world's best game bots rely on supercomputers, we aim to create an AI that is efficient on a regular, consumer laptop. To achieve this, we will use techniques such as optimizing code and reducing memory usage.

In addition to these main goals, our project aims to propose new ways of thinking and strategizing against bot vs bot strategies. By experimenting with different techniques and theories, we aim to contribute to the field of game-playing AI and push the boundaries of what is possible with Connect 4 AI.

Furthermore, in the process of developing this Connect 4 AI, we explored different platforms and programming languages. After initial development in Python, we successfully converted the base code to Dart and utilized Flutter to create a functional Connect 4 game that can be deployed across multiple environments, including iOS, Android, and web. This allowed us to expand the reach of our game and make it accessible to a wider audience. We used a game template compatible with Flutter that gave us a screen for start-up and game over situations. This template was also a useful tool in allowing us to create a functional Connect 4 app with no ads that helped us test and read the paper. We labeled the grid coordinates and allowed for

player-vs-player gameplay to quickly input information of board states before integrating the AI component.

Finally, we used Pygbag to deploy straight from Pygame to the web without transforming the bulk of the code base. This allowed us to create a web version of the game that could be played on different platforms, expanding the accessibility of our Connect 4 AI beyond a single programming language and operating system.

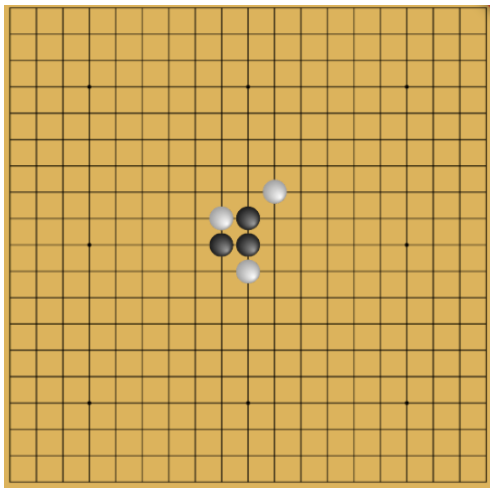
Early Development

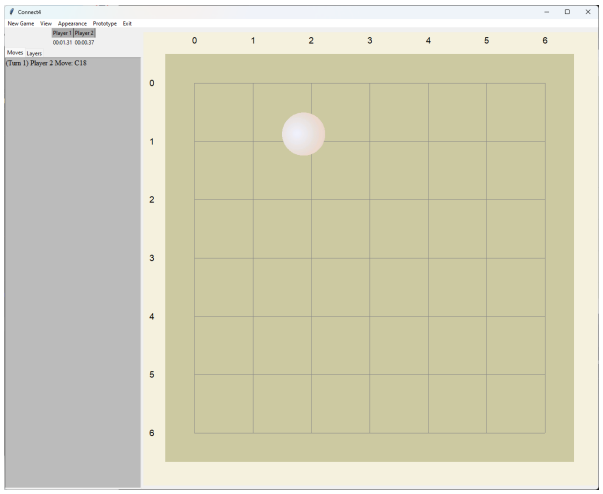
At the start of this project, we were given code from a previous project that was advised under Professor Chen. This code was a template for Gomoku, a similar game to Connect 4 in that they are both strategy games played on a grid. However, there were also significant differences between the two games, such as Connect 4's gravity mechanic and the difference in board dimensions.

We started by diving into the code and looking for similarities in how the game was computed, such as checking for winning conditions and keeping track of the game state. While there were certainly some similarities, there were also some key differences that needed to be addressed. For example, the code for computing risk vs reward moves in Gomoku was heavily skewed and needed to be adjusted for Connect 4. Since the board was also assessing its scoring evaluation based on the dimensions of the board, any alteration to the dimensions would crash the program immediately as factors became undetermined. The task at hand was tweaking a

previous function to allow a playable game of gomoku on the board dimensions of Connect 4 with the win conditions of Connect 4 as well.

Another major difference was Connect 4's gravity mechanic. In Gomoku, pieces could be placed anywhere on the board, whereas in Connect 4, pieces would fall to the bottom of the column due to gravity. This required significant changes to the code, including a complete overhaul of the game board and functions to be properly formatted for Connect 4's dimensions and rules.





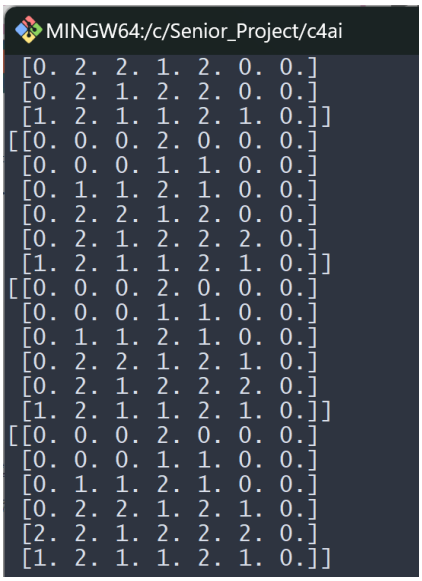
After several iterations of tweaking and adjusting the code, we were able to get a functional Connect 4 game up and running without crashing. However, we soon realized that the Gomoku code was too heavily intertwined with its UI and other components, and that a full rewrite was necessary to create a Connect 4-specific UI that could fully showcase the game's mechanics and features.

This required us to strip the code down to its bare essentials and start from scratch with a UI solely focused and coded for the purposes of Connect 4. By doing so, we were able to fully highlight and optimize Connect 4's unique mechanics, such as the gravity-based piece movement, and create an intuitive and user-friendly UI that would appeal to both novice and experienced players alike.

The first consensus we came to for the project at this stage was that we would be using the language Python again in terms of resources available and to gain familiarity with a popular programming language. There were several benefits to this decision. The access to many useful libraries such as pygame and pygbag are most notable. Pygame would be immediately useful for getting the graphical elements of the game up and running, such as the menu. Pygbag would prove itself useful later when porting the game to a web server.

Github was used to store code and gradually add features through branches. Discord and Jira were also used for communication and organization.

The first version of our program was a simple terminal based game with no graphical elements, to fulfill the first goal of having a functional Connect 4 simulator. In this terminal version, 0 would be a space with no piece, 1 would contain the player’s piece, and 2 the computer’s. The functions below were used for the winning conditions. It checked for all possibilities, be it vertical, horizontal, or diagonal.




```
# Check positively sloped diaganols
for c in range(COLUMN_COUNT - 3):
    for r in range(ROW_COUNT - 3):
        if (
            board[r][c] == piece
            and board[r + 1][c + 1] == piece
            and board[r + 2][c + 2] == piece
            and board[r + 3][c + 3] == piece
        ):
            return True
```

```
# Check vertical locations for win
for c in range(COLUMN_COUNT):
    for r in range(ROW_COUNT - 3):
        if (
            board[r][c] == piece
            and board[r + 1][c] == piece
            and board[r + 2][c] == piece
            and board[r + 3][c] == piece
        ):
            return True
```

```
def winning_move(board, piece):
    # Check horizontal locations for win
    for c in range(COLUMN_COUNT - 3):
        for r in range(ROW_COUNT):
            if (
                board[r][c] == piece
                and board[r][c + 1] == piece
                and board[r][c + 2] == piece
                and board[r][c + 3] == piece
            ):
                return True
```

```
# Check negatively sloped diaganols
for c in range(COLUMN_COUNT - 3):
    for r in range(3, ROW_COUNT):
        if (
            board[r][c] == piece
            and board[r - 1][c + 1] == piece
            and board[r - 2][c + 2] == piece
            and board[r - 3][c + 3] == piece
        ):
            return True
```

As the project continued development on the side of the algorithm, parallelly added was the graphical presentation of the game. Using the pygame derivative library, *pygame_menu*, we got a user-friendly menu going and a loop going that returned players to the main menu after completing a game.

During the development of this Connect 4 AI program, we also aimed to add quality of life features that would enhance the user's experience and make it a more useful training tool for Connect 4 enthusiasts. One of the significant additions we implemented was the undo button, which allowed players to go back and explore various possibilities even after the game has been completed. This feature enabled players to learn from their mistakes and analyze different strategies and moves to better their gameplay.

In order to provide an effective training tool for Connect 4 players, we added an undo button that works even after the game has already been completed. This feature allows players to go back and explore as many possibilities as they wish in order to gain a deeper understanding of the game. To implement this feature, we realized that we could store board states simply by pushing the column number of where moves were made. This is because if a number repeated, it meant that a piece was on top of it in the array. This approach leads to a max array of 42 with numbers 0-6, which allowed us to easily track the history of moves made in the game. When the undo button is clicked, we simply pop the last element off the stack and update the board accordingly, providing players with greater flexibility and control in their gameplay.

We also added animations for the pieces to make the gameplay more engaging and visually appealing. Instead of the pieces just appearing on the board, we coded how the pixels would traverse the graphical interface to create a physics-based animation, explicitly showcasing the gravity element difference to other games. To achieve this effect, we applied a sin function to the piece's final position to give them movement.

As the development phase of the Connect 4 program progressed, we realized the need to enhance the user experience by adding a set of quality-of-life features. One such feature was the inclusion of an undo button that allowed players to backtrack on their moves even after completing the game, facilitating a better understanding of the game's various possibilities. Additionally, we implemented animations for the game pieces to make the gameplay more engaging.

With the project nearing completion, we added a more technical scoreboard under the game board to provide users with a better understanding of the game's progress. The scoreboard shows the number of moves made, the AI's prediction level, and the scoring prediction based on the depth of the search algorithm. This feature provides a deeper insight into the computer's optimal move choice, allowing players to assess the score and refine it accordingly.

To further enhance the gameplay experience, we included an Algo2 vs. AI mode towards the end of development. This mode is unique in that it follows a predefined set of rules, allowing the algorithm to deviate from the theoretical best move and capitalize on opportunities that may not be immediately apparent in the scoring possibilities for the next five moves. This mode is highlighted in blue in the scoring column, while the recommended move is indicated in red.

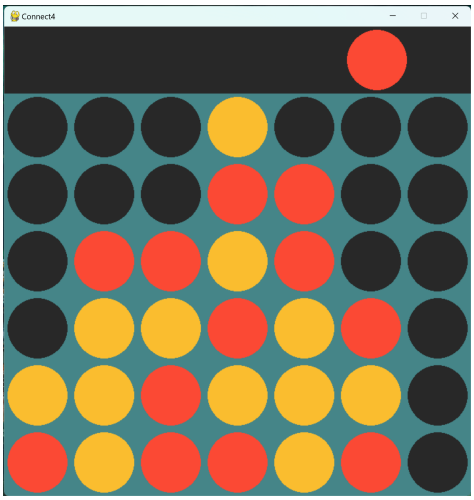
For the introduction of the UI the first thought considered was about the counter. In board games that involve game AI, there's often a need for a counter to keep track of whose turn it is - whether it's the user's turn or the AI's turn. In our project, we implemented a counter that would trigger the AI move once the user makes a move and then reset once the AI has made its move. Initially, we had set the game to randomly determine who would make the first move. However,

with the introduction of Pygame_menu, we were able to provide the user with the option to

choose who would make the first move. This meant we had to tweak the counter to be able to enter the state of back and forth triggers at any point that was a pretty simple refactor.

In addition to the turn counter, Pygame_menu also provided other features that we were able to incorporate into the game. For example, we added a quit button and a play button to start and exit the game from the menu. Consequently, we had disabled the window buttons in the process such as minimize and exit in the corner of the game window during startup of the program.

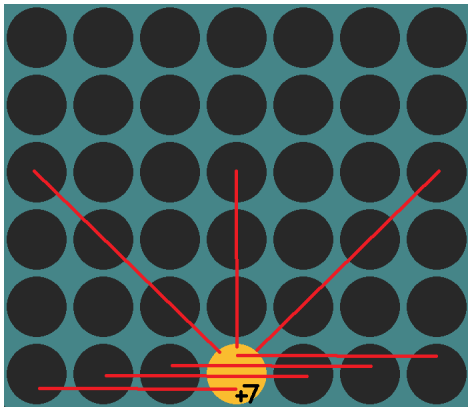
Through refactoring, we were able to reintroduce the disabled window buttons, and ultimately, we were able to create a light-themed lofi UI and game screen that was both functional and visually appealing. Overall, these additional features and improvements made the game more user-friendly and enhanced the overall gaming experience. It also helped make the program a more useful training tool for Connect 4 players, allowing them to explore different strategies, learn from their mistakes, and improve their gameplay.



Scoring and Strategy

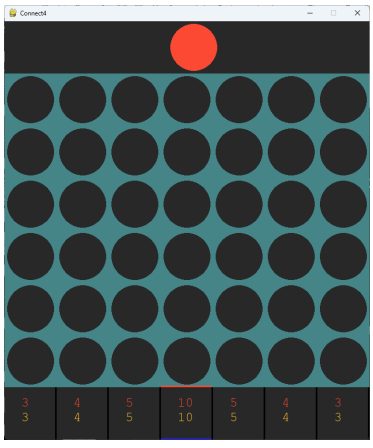
While Connect 4 was fully solved through a brute force algorithm, this was not a feasible approach as discussed previously in *Introduction*. By exploiting symmetries and rotations so that it does not have to redo work on repetitive calculations, the most effective technique we utilized was the minimax algorithm with alpha-beta pruning to accompany the scoring system.

A benefit of the player having only 7 or less possible moves a turn meant it wasn't too difficult or intensive to implement a scoring evaluation for the optimal move to do. Here, every column is assigned based on its attacking and defensive ability combined. The following is the move hierarchy:

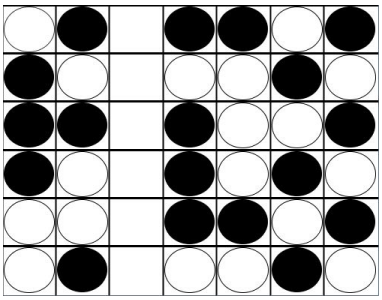


- If this move is an instant win, do it
- If the opponent has an instant win next turn, move there
- A 3 in a row where an empty space is available
- A 2 in a row with 2 empty spaces available
- Singular moves with the highest potential for wins in multiple directions

Through setting up the program and learning from others, multiple winning strategies made themselves apparent to us. Most apparent of all was the specialty of the central column. Placing a piece in the central column is the most optimal move to start, and the central column remains important throughout the game because it holds the most potential to get a 4 in a row, with 7 possibilities in total. A placement to the right or left of the center



removes 1 possibility making the best strategy to prioritize the center. An optimal game will always start with the central column getting immediately filled up by both players. This can be seen in the scoreboard visualization on the bottom of the board, which was added towards the end of development and can be seen in the following figure.



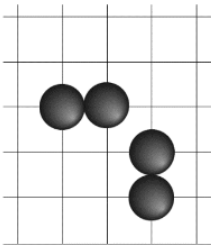
Another part of the strategy is to get traps going for the other player to fall into. Traps naturally occur by maximizing moves that have the potential to be in a winning combination. Eventually, players will be stuck in Zugzwang, meaning they will be put in a position where the best move would hypothetically be not to play but they are forced into making moves that benefit the opponent. An example of a Zugzwang scenario can be seen to the left. Zugzwang is a powerful strategy that can be used to gain an advantage in Connect 4. To utilize zugzwang, you must first create a situation where your opponent is forced to make a move that will weaken their position. This can be accomplished by creating a trap in the game board that will force your opponent to make a move that will ultimately benefit you. For example, you can create a situation where your opponent has no choice but to block one of your potential winning moves, but in doing so, they create an opportunity for you to win the game on your next turn.

Another way to utilize zugzwang in Connect 4 is to create a situation where your opponent is forced to make a move that will allow you to set up a winning move on your next turn. For instance, if you have two potential winning moves on the board, you can force your opponent to block one of them, which will allow you to set up the other and win the game. The overall strategy is to create the most amount of winning potential while minimizing threats. This

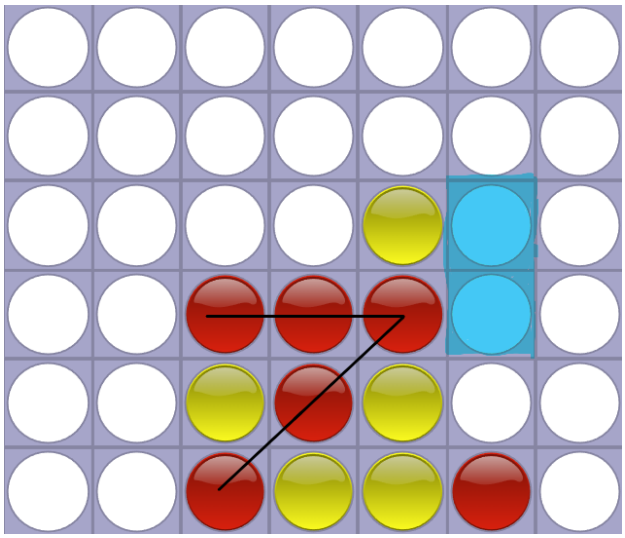
strategy is partially achieved by implementing a minimax algorithm given a terminal node is found within the specified depth of moves into the future.

Closing rows is also a crucial strategy to secure a win. This can be achieved by connecting three pieces in a row, leaving the fourth spot open for your opponent to fill. However, this is only effective if the fourth spot will result in a win for you on the immediate next move if your opponent makes that move. This is known as utilizing zugzwang to your advantage, forcing your opponent to make a move that will ultimately result in their defeat. By creating this scenario, you are putting pressure on your opponent and limiting their options, increasing your chances of winning the game. Therefore, paying close attention to your opponent's moves and planning ahead to close rows can be the key to success in Connect 4.

Similar to Gomoku, Connect 4 also has a strategic pattern that can lead to an imminent



win. The formation of a "7" shape by connecting four pieces in a particular way provides two potential winning moves. This pattern involves placing three pieces in a vertical line and one piece in a horizontal position adjacent to the bottom piece of the vertical line. This creates an L shape and essentially gives you two winning opportunities, one on top of the other. The importance of having two winning opportunities is that even if your opponent blocks the first winning move, you can start playing moves in the row above it, and the immediate move above it should still result in a win. This formation of pieces allows you to control the game and put your



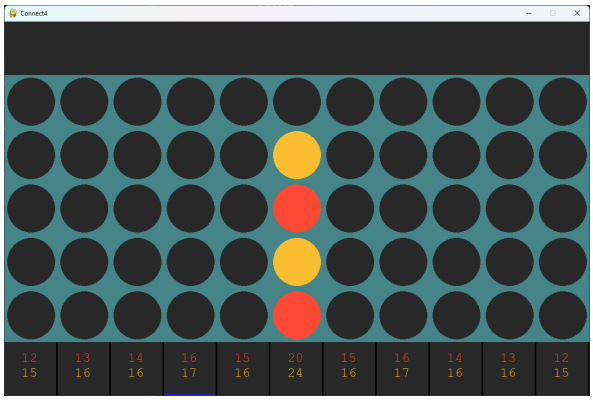
opponent in a difficult position where they have to constantly block both winning opportunities to avoid losing the game.

For explicitly scoring based off of the move hierarchy above, we assigned scores to different board positions to determine the best immediate move to make. The highest score was given to a terminal node win for a connection of 4. If there were three in a row with an empty space in the direction of the connection on either side, it was given a score of +5. For two in a row with two empty spaces in the direction of the connection on either side, it was given a score of +2. Potential starts of a connect four a single piece with three empty spaces in all possible directions available were scored +1. On the other hand, if the opponent had three in a row with empty spaces, it was scored -4, and two in a row with two empty spaces were scored -1. A loss was given the lowest score as a terminal node. This scoring system was recursive and iterated 5 moves into the future for every column available to play.

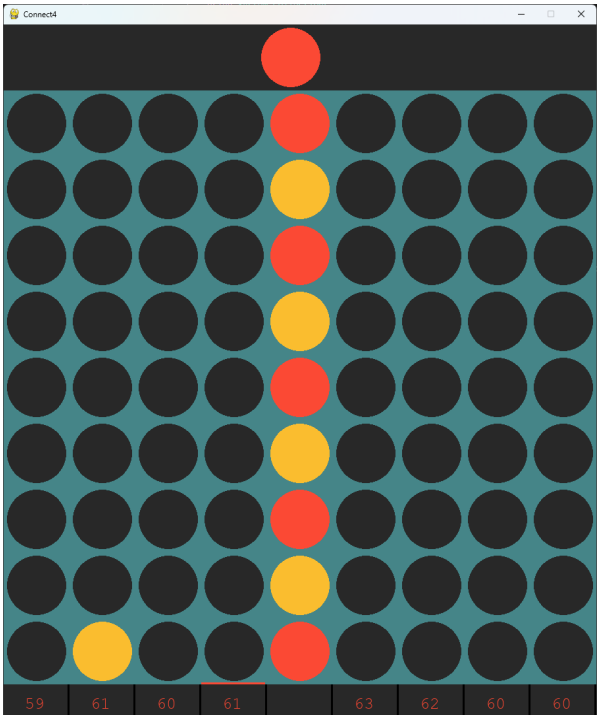
The scoring system was fine-tuned against a perfect Connect 4 bot to mimic its optimal play. However, further tweaking may still be necessary to further proportionate the scoring system with the addition of more rules and patterns to find. Additionally, other scoring mechanics such as forks and closing rows when you have control of the zugzwang were implemented but not yet fine-tuned. These changes have the potential to significantly strengthen the bot's scoring function and improve its overall gameplay. In addition to the scoring mechanics mentioned to provide a concrete value of measurement, the algorithm also prioritized connections in the center of the board. The scoring factor for a move that created a connection in the center was multiplied by 3, as having control of the center can be crucial for future moves and potential connections. Even if there were more connections available elsewhere on the

board, the algorithm recognized the importance of prioritizing the center. This strategy was based on the idea that controlling the center allows for more flexibility and opportunities for future moves, increasing the chances of winning the game.

To clarify, controlling the middle of the board is generally more advantageous for attacking purposes than controlling an edge, but controlling the edges can be more important for blocking purposes to prevent a loss as determined by the terminal node heuristic value. Therefore, blocking strategies take precedence when necessary, even over prioritizing the scoring of connections in the center.

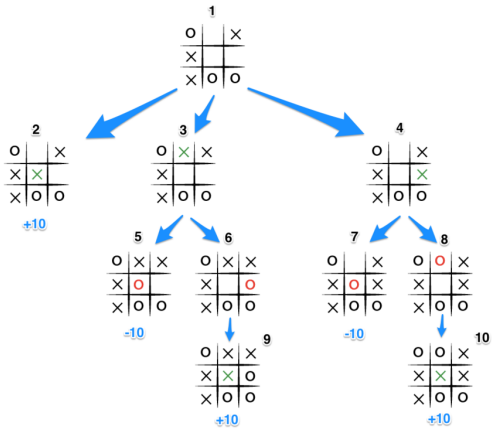


To eliminate magic numbers and ambiguity, the scoring function was designed to prioritize the center by calculating a central point based on the size of the grid. The central point was calculated using the inverse of the base value minus the average of the row and column counts, divided by two. This approach makes the center more or less advantageous depending on the size of the grid, with larger grids giving the center less importance compared to smaller ones. However, in more constricted areas of the grid, the center still remains important. This makes



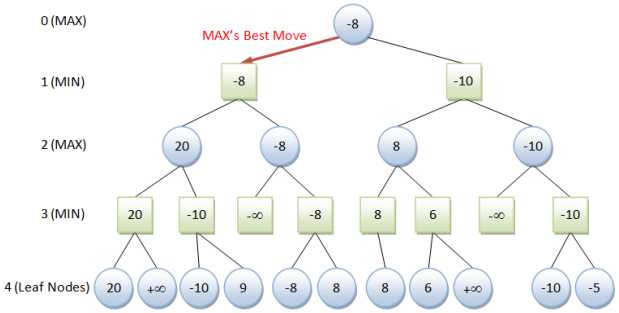
the game scalable to any grid size, as the row and column counts can be adjusted to play with different dimensions. Overall, the scoring function and central point calculation provide a more nuanced approach to decision-making in the game, allowing the algorithm to consider various factors and make strategic moves accordingly.

Minimax Algorithm



The minimax method is a recursive algorithm focused on the goal of minimizing chances of loss and maximizing chances of wins. A way to visualize the algorithm presented is like a tree. Each node in the tree above represents a different option that can be taken by the agent with depth 0 being the current state of the game. Firstly, the agent acts as the maximizing player, that acts as the agent wants to: maximize the score which maximizes the odds of winning. However, on the following depth, the computer simulates what the opponent would likely do to heighten their chances of winning. In this state, the agent acts as a minimizing player, trying to minimize the score as much as possible. The computer will continue to simulate, creating a recursive tree until a terminal node is reached. This terminal node is the end of the game where either the maximizing player or minimizing player wins.

Using this method, the computer simulates the act of looking ahead in the game through predictions and counterplay. And with that we can determine how far into the future the computer simulates possibilities by controlling the depth in which the tree searches. The following figures show the minimax pseudocode, further visualization with numerical values for the scoring, and our code for the terminal nodes.




```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

```
def minimax(board, depth, alpha, beta, maximizingPlayer):
    valid_locations = get_valid_locations(board)
    is_terminal = is_terminal_node(board)
    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 10000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -10000)
            else:  # Game is over, no more valid moves
                return (None, 0)
        else:  # Depth is zero
            return (None, score_position(board, AI_PIECE))
```

Alpha-Beta Pruning

This method is one that is very effective for a game like Connect 4, Chess, or Go that have many more possible moves on every turn. The small amount of choice means that tree searches are very efficient but they can be further pruned to avoid wasting time searching through branches that seem weak because they end in terminal nodes assuming glaring mistakes

are played which in turn would produce nonviable results given the goal is to perform optimal on all accounts.

Alpha-beta pruning adds bounds where the maximizing and minimizing player can reasonably guarantee that going down a certain branch will guarantee a terminal mode in the AI's favor. The alpha is the upper bound, or the best value for the maximizing player, and the beta is the lower bound, or the best value for the minimizing player. After a value has exceeded the bounds set, the implementation is considered fail-soft because it returns the best move found so far, even if it hasn't gone through exploring every possible move to its finality.

Experimentation and Results

The implementation of the minimax algorithm with alpha-beta pruning was successful in winning 96% of games against human contestants with a wide variety of experience levels. The algorithm was able to make intelligent decisions based on a given game state and make moves that minimized the maximum loss, hence the name "minimax." Of the 50 games played over the server for anonymity, 48 were won by the AI, 1 resulted in a tie, and 1 resulted in a loss.

Given the negative results through experimentation and our own trial and error, production of a stronger Connect 4 AI can now begin development for the remainder of the project.

Research and Improvements

Although the minimax algorithm was highly successful, one loss could be seen as an anomaly or a result of chance. The opponent in this game was intentionally mirroring the algorithm's play "just for fun," and according to the Victor paper, this method of playing according to predefined rules can do well against bots that are very strong but still make an occasional mistake (as little as one depending on its point in the game) but is not useful against humans.

Furthermore, the Shannon C theory seems to be a promising area to explore for Connect 4 AI. It suggests that a game's value can be determined by considering its entire tree of possible moves and outcomes, rather than just the immediate next move. This could be a useful addition to further explore the capabilities of Connect 4 AIs.

Another modification would be to introduce some randomness into the algorithm to prevent it from becoming too predictable and easy to defeat. For example, the algorithm could sometimes make a suboptimal move to intentionally throw off the opponent or force them to make a mistake. Of course, this would only be useful given there are strategies of non optimalism against non optimalism as defined by Fernau, H. (2006) in “The game of Connect Four has been solved.” One implementation introduced can be seen here that deviates from the proposed solution so compute a different set of moves based on rules rather than scoring to mirror and adapt before returning to its control of the board in its mostly optimal play.

In terms of the technique of mirroring moves after nonoptimal play, the algorithm could be modified to recognize and adapt to such patterns. For example, it could track the opponent's moves and use that information to anticipate and counter any attempts to mirror the algorithm's moves. It could also adjust its evaluation function or search depth based on the opponent's play style and skill level, allowing it to more effectively exploit weaknesses or avoid traps through allowing minimax to search further without computing so much more given there are less moves left in the game (iterative deepening search).

The specific approach taken will depend on the desired level of complexity, randomness, and adaptability. Aside from being functional, it maintained winning streaks against human opponents, and its capability was tested with multiple college students and professors over the course of the semester. Since its loss, the algorithm has been further optimized to be even stronger through the criteria and alternative strategies above. We've tracked our computer's proficiency through playing its moves against other Connect 4 AI's such as

Functionality updates organized by progression of project

Animate the piece drop.

One of the first things worked on was when a player drops a piece you can see it appear at the top of the column, pauses so the player can see the piece then it moves down another row in the column and repeats until it reaches its final spot

at the bottom of that column.

```
def animate_drop(row, col, piece, screen):
    color = RED
    if piece == AI_PIECE:
        color = YELLOW
    for r in range(ROW_COUNT - row):
        pygame.draw.circle(
            screen,
            color,
            (
                int(col * SQUARESIZE + SQUARESIZE / 2),
                height - (height - int(r * SQUARESIZE + SQUARESIZE / 2)),
            ),
            RADIUS,
        )
        pygame.display.update()
        pygame.time.wait(90)
        pygame.draw.circle(
            screen,
            BLACK,
            (
                int(col * SQUARESIZE + SQUARESIZE / 2),
                height - (height - int(r * SQUARESIZE + SQUARESIZE / 2)),
            ),
            RADIUS,
        )
```

Game Menu

The goal of this menu was to allow the player to choose if they wanted to play first or second. When the game ends the player sees who won and instead of exiting the program it will go back to this menu to allow the player a chance to play again or quit.

The python package called pygame_menu was utilized. This package wasn't built into pygame, so the necessity to install it using pip install pygame_menu was required. Once installed it was able to be imported as a package at the top with import pygame_menu.

Moving on to the pygame_menu documentation which had some examples that were used to learn how to implement the menu system that worked for our connect4 game,

a few challenges that we came across was that we had to restructure the code a bit and figure out how to get the play button to call our main game loop with the correct variables set, like if the player wanted to play first or second. Another challenge was that instead of exiting the program, the game would pause for a bit then return to the main menu.

Source: https://pygame-menu.readthedocs.io/en/latest/

Game State History with undo move on right click

The goal with this feature was to allow the player to go back as many moves as they like. As an example, if the computer won the match or the player wanted to play in a different column, the player could right click to undo that move and play a different spot.

To get this feature working, use a list and a copy of the current board state appended to this list. It was important to create a copy of the board state, since we learned that in python (unlike c++) it will store a pointer to the board state and if it isn't a copy you end up with a list full of the same board state. Since our board state is a matrix I ended up importing deepcopy from the copy python library.

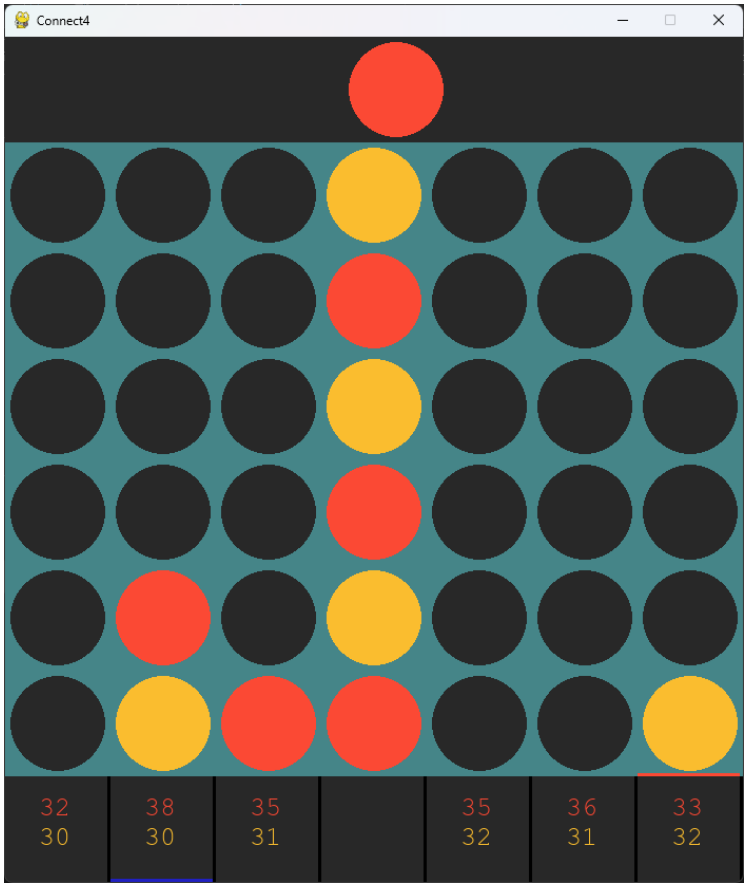
Another challenge was getting the right click to work correctly.We had to add another condition within the main game loop and also instead of pausing for a set amount of time on a win or loss,

we changed it to left click to exit the game or right click to undo their move. This allowed the player to right click and undo the previous moves they made and try again.

Displaying the scoring system to the player with column recommendations

This was an interesting feature, since it allows the player to see what the computer was calculating based on its scoring system and minimax function for move decisions.

It ended up making the game window size a bit larger to make room for the scoring bar at the bottom of the game.



The print_score() function first would run the minimax() function to get the next best column move for the player. It would show what the next best column the computer would play if it was its turn.We decided to implement a small red tab above the score bar to show this to the player since there is some convolution of the scoring from the tree as given the risk function for the algorithm it is not a simple deduction of the difference between if you were to move there minus if the opponent were to move there. Next the print_score() function will loop over each of the columns and score each player's move and show those scores on the bottom of the screen for both players. This gives the player insights to what that column is worth as a player and what the column is worth to the computer if it were to play that column. Some of the challenges were to get the layers correct on the score bar and adding it to the bottom of the game.

Alternative approach through Shannon-C (“algo2”)

During our weekly meetings with our advisor Dr. Zhixiang Chen, in one particularly interesting one was information a group member shared where through utilizing rules other than conventional scoring we can still achieve success against calculated algorithms because with just one mistake, there exists board positions which can force and win. This is testament to reading and

implementing a more refined set of rules based on patterns alongside a mathematically calculated one through a smaller exhaustive search. One example that Brian showed us was a forced win after white played 5 turns that he learned about while reading the master thesis by Victor Allis.

```
#algo2 setup and variables
key_positions = list([3,3,3,2,0])
```

With a general ai model, its performance is dependent on the little information that you give it. So to improve our connect4 ai model, we wanted to see if it would block a scenario like this from happening. Another advantage would be if it knows about a board position like this, it could try and use it against the player. Once a player has acquired these key board positions they just mirror the opponent's moves which forces a win. If the even and odd moves are in your favor(like if the enemy player will have to eventually play in the column that gives you a win), mirroring moves should block any threats as long as the pieces in the neighboring columns are symmetrical.

```
def algo2(key_positions,mirror_move,board,screen):
    ##
    # Recommends next move using a different algorithm
    # The key positions are based off the master thesis by Victor Allis Diagram 3.14
    move = mirror_move
    if(board[2][1] == 0 and board[0][1] != 0):
        move = 1
    elif(len(key_positions) > 0):
        move = key_positions.pop(0)
    elif(is_valid_location(board,mirror_move) == False):
        move = 6
        if(is_valid_location(board,move) == False):
            move = get_next_open_row(board,6)

    if(move != None):
        pygame.draw.rect(screen,DARKBLUE,
            ((move*SQUARESIZE,height-SQUARESIZE+(SQUARESIZE-3),SQUARESIZE-3,SQUARESIZE))
        )
    return move
```

The parity of the threats is an essential consideration for both players. In a closed game, if the white player has an odd threat and the black player has an even threat, then the white player

will win. On the other hand, if both players have an even threat, both will get their normal squares since there are no columns where only an odd number of pieces can be played. Black can then refute white's threat and win afterward on their own threat. If white has an even threat and black has an odd threat, then both players have threats that cannot be easily utilized, and this would result in a draw if black gets even squares and white gets odd squares. However, if black chooses to preserve their odd threat, they would have to give it up before being able to play in a column where white has their even threat, causing the shift of odd and even squares belonging to a color to occur and giving white the win. Lastly, if both players have an odd threat, the moves remaining would be even, and white would have to sacrifice their threat to avoid loss by zugzwang, resulting in a draw. However, there is a possibility of setting up a new threat in the creation of a draw, which would prevent the termination of both threats in forced moves.

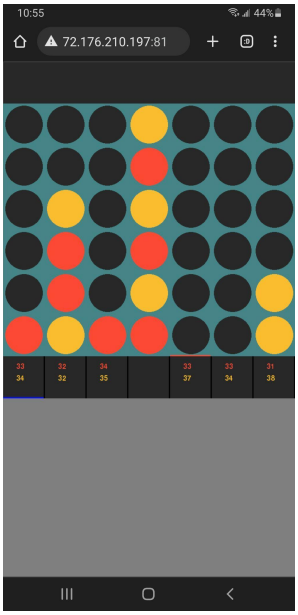
The threat combinations for even and odd threats in Connect 4 can be traced to a version of Shannon C theory through this approach. This theory suggests that a game's value can be determined by considering its entire tree of possible moves and outcomes, rather than just the immediate next move. By considering the possible threats and their combinations at each level of the tree, the players can anticipate potential outcomes and make more strategic moves. This can help them to identify even and odd threats and plan their gameplay accordingly, with the goal of creating advantageous positions and avoiding zugzwang. So, the concept of even and odd threats in Connect 4 can be seen as a practical application of Shannon C theory, albeit in a simplified form. Introductions of certain patterns can be introduced to change results in some instances based on how they affect an area and their interactions with one another. Although not utilized

explicitly, they may arise in the mathematical calculation approach and are label as:

Baseclaim	Baseinvese
Claimeven	Aftereven
Lowerinverse	Highinverse
Before	Specialbefore

Implementation based on adding to the score given the conditions for the positions are satisfied would be an area for *Further Improvements*.

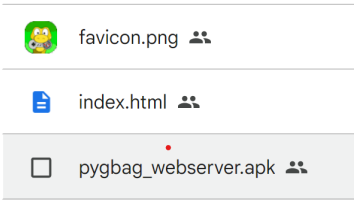
Web Deployment

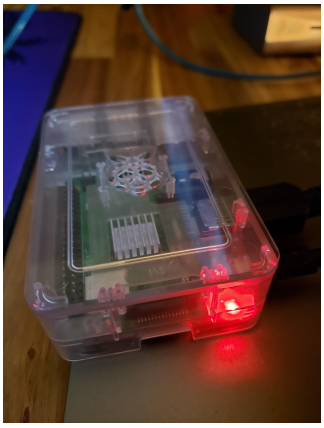


Pygbag is a python package that was created in 2022. Unfortunately it doesn't work with pygame_menu so the home screen was cut out for the web development, which means the player has to play first. Another change to the code made was using the function asyncio.run(main()) to run our main function. In order for this function to work just import asyncio. Once you download pygbag with the command pip install pygbag. You can run the pygbag command from the terminal with the folder your python code is in, named main.py.

pygbag folder.with.code

When you run the command, it creates an app (pygbag_webserver.apk) that runs in your web browser on localhost. You can go to http://localhost:8000/#debug and watch what the output is when the app is being built and booting up. It was very helpful when trying to figure out what changes needed to be made to get our code working with pygbag.





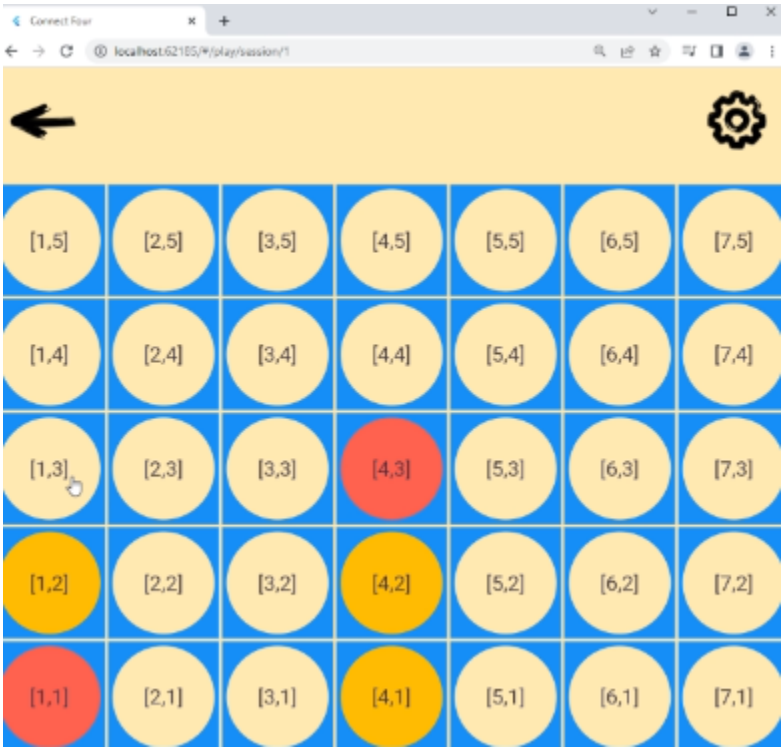
Once everything was up and running, Konrad decided to setup a raspberry pi with a webserver to host the app pygbag created with our code. He used the terminal to install nginx, a light weight web server. Copying the app folder pygbag created to the pi and configured the webserver to host on port 81. Forwarding the port on his router to the pi to allow people to connect with the correct web address, players can play the game from a computer and even their phone.

Source: https://github.com/pygame-web/pygbag

Source: https://docs.nginx.com/nginx/admin-guide/web-server/web-server/

Flutter for Mobile (IOS/Android) and Extension to Web

In the latter part of the semester, the team utilized the Flutter environment and a template to convert the basic elements of the Python code to Dart for mobile development. While implementing some of the easier functionalities of buttons in the workspace, the AI occasionally suffered from certain logic issues that resulted in illegal floating moves. Nevertheless, mobile development proved to be a helpful tool for player versus player interaction in setting up board positions, especially given that the grid locations were labeled.



Project Management

Our team used several tools and processes to manage our project. To decide what to work on each week, we held weekly meetings with our advisor to discuss improvements and pace ourselves to stay on track with feasible tasks. Assignments were typically made by talking about implementing a feature of functionality while also discussing how to improve our main method.

To stay on the same page, we used Jira as a professional standard in the software engineering space to maintain a record of tasks, delegation, and review. In addition, Discord and in-person meetings were used for open lines of communication. Our advisor was also a great help in delegating tasks and pushing for improvements in several areas.

For minimizing effort, redundancy, and inefficiency in combining individual work, tasks were outlined and given points in certain instances on Jira for tracking and pacing. Github was used for version control with push commits mainly done through Konrad as he implemented many features. Brian shared much of the AI and scoring functionality for Konrad to also add. Pedro helped write and organize as well as assist in more localized areas that Brian and Konrad worked on.

Our team regularly checked in with each other and with our advisor to organize and formulate plans. Konrad largely led the design and functionality and was responsible for the web development that ran well during the demonstration. He kept on top of knowing and informing all parties about any changes and how he was going about them in order to halt any potential coding conflicts by changing anything on the UI that may interrupt how the algorithm was producing moves or deciding order.

Summary

Our team successfully developed a Connect 4 game with an AI player that is able to play against a human player. The game has a graphical user interface built using tools within the Python environment as well as scaling to a web version with mobile implementation also in the works. We used Jira as our project management tool and Github for version control. Our team was able to overcome several obstacles during the development process, including issues with implementing the AI player, integrating the UI with the game logic, and managing conflicts in the codebase.

Lessons Learned

Throughout this project, our team learned several important lessons. Firstly, we learned that project management is crucial for the success of a software development project. Using tools like Jira and maintaining open communication channels among team members and the advisor helps in organizing and prioritizing tasks and keeping everyone on the same page.

Secondly, we learned that collaboration is key in software development. Dividing the work among team members and assigning responsibilities based on each member's strengths and expertise allowed us to work efficiently and minimize redundancy and inefficiency.

Lastly, we learned that testing and debugging are crucial in ensuring the quality of the final product. We encountered several issues with the AI player during the development process, but we were able to overcome them through thorough testing and debugging. Overall, we are proud of what we accomplished as a team even through compromise on some decisions and are grateful for the experience and knowledge gained through this project**.**

References

Allen, J. D. (2010). *The Complete Book of Connect 4: History, strategy, puzzles*. Puzzle Wright Press.

Lazar, D. (2021, May 14). *Understanding the minimax algorithm*. Medium. https://towardsdatascience.com/understanding-the-minimax-algorithm-726582e4f2c6

Fernau, H. (2006). The game of Connect Four has been solved: The first player can always win by force. Retrieved from http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf

Shannon, C. E. (1979). An introduction to the game of connect-four. Retrieved from https://webpages.ciencias.ulisboa.pt/~ipsilva/pdf/preprints/Shannonrevisto.pdf

Silva, I. P. (2002). Solving Connect-Four. Retrieved from https://www.maths.ed.ac.uk/~jop/teaching/honours_project/BEL-KimberlyWood.pdf

Pygame. (n.d.). Pygame documentation. Retrieved from https://pygame.org/docs/

Pygame_menu. (2021). Pygame_menu 4.4.2 documentation. Retrieved from https://pygame-menu.readthedocs.io/en/4.4.2/