The Connect Four Game

Table of Contents:

The Connect Four Game

Youssef Abdeltawab
Texas state University
Department of Computer Science
San Marcos, TX, 78666, USA
October 19, 2023

1. Problem Description

Connect Four is a popular board game. Players take turns dropping their discs into a grid, with the goal of occupying four slots in a row. This project aims to explore various AI techniques to play Connect Four.

Almost all game players use a game tree to represent positions and moves. However, unfortunately the whole game tree size is tremendously huge for almost all interesting games. For example, checkers is 10^20 and chess is 10^40. The total number of nodes generated in game tree is roughly W^D, where W stands for number of possible moves on average for each node, and D is the typical game length. For many games such as chess, there are no practical algorithms that can manage such a full tree due to lack of time. Connect Four Game has about 10^13 possible board positions in a standard 6*7 board, making it infeasible to store a move tree in memory.

The proposed solution is to stop generating the tree at fixed depth, d, and use an evaluation function to estimate the positions d ahead of the root. The use of depth limited search and evaluation function is a pragmatic solution for dealing with immense complexity of the game trees in challenging games like chess. Once we have, we can optimize the searching process we are interested in using AI to create a gameplay agent (in other words, a difficult opponent to play against).

The challenge of this project is limited computer power, so we don’t have the ability to explore most of the game tree as mentioned above. It’s difficult to measure to evaluate the effectiveness of AI agent other than making our evaluations functions compete against each other at different depth levels.

2. The Domain

2.1 Project Objective

The objective of this assignment is to create AI playing algorithms, implement Min Max and alpha beta-pruning and design an evaluation function for measuring game states. The problem of creating AI agents using Min-Max-A-B has been a classic AI challenge and has applications in game development. This project uses the Connect Four game which is a very well-sourced game that many people have experienced before. Here's are a list containing project objectives:

- The primary goal of this project is to design two AI agents capable of playing and winning games.
- Develop Min-Max-A-B algorithm which aims to find the optimal move for the player by recursively exploring the game tree and the algorithm seeks to find the best move for the X (maximizing player) and O (minimizing player). The algorithm will be explained in greater detail in the methodology section.
- Develop Alpha Beta Pruning Algorithm which is intended to optimize the searching process by reducing the number of nodes explored in the game tree.
- Design evaluation function for measuring game states and provide a numerical value that represents the desirability of that state for the AI player.
- Create a statistical table that will analyze the performance of each evaluation function with different cuts of depth in terms of number of node generated, execution time, the size of memory used by the program and winning/losing statistics for each player.

2.2 Connect Four Game Board

Connect Four is a two-player connection game played on a board with 7 vertical columns and 6 horizontal rows. The game begins with an empty board, providing 42 available squares. Player 1 is represented by X, while Player 2 is represented by O. In the standard form of the game, one player is yellow, and the other side is red. If a player puts X in one of the columns, it will fall to the lowest unoccupied square in the column. As soon as 6 squares in one column are filled, no other player can make a move in this column.

2.3 Connect Four Game Rules

The players make their moves in turn. Both players will try to get four connected squares, either vertically, horizontally, or diagonally. The first player who achieves one such group of four connected squares, wins the game. If all 42 squares are played and no player has achieved this goal, the game is drawn.

Diagrams 1.1 and 1.2 show positions in which X won the game:

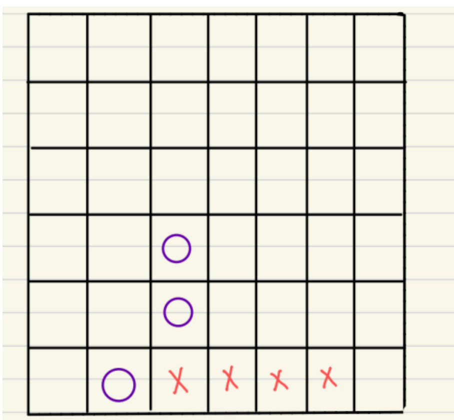

Diagram 1.1 X has won.

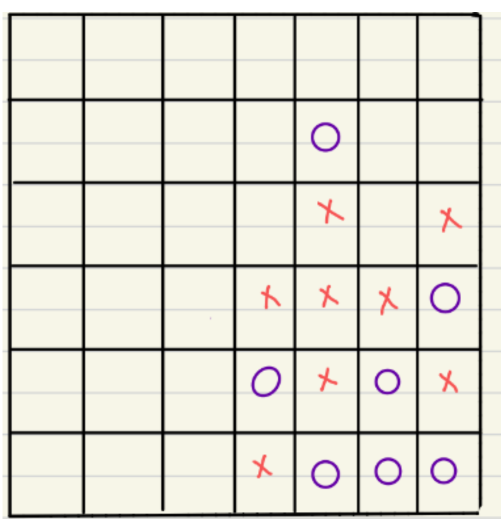

Diagram 1.2 X has won.

3. Methodologies

3.1 The Min-Max Algorithm

The Minimax algorithm is a fundamental technique employed in the realm of sequential two-player games. It serves as a decision-making rule that guides a player in choosing the optimal move, given the condition that the opponent is also making the best possible choices. In other words, Minimax operates under the assumption that both players are striving for an optimal outcome with each move they make. There are two actors in the Min-Max, the maximizer (X) and minimizer (O). The player is the maximizer, and his corresponding rival is the minimizer. Using the evaluation function numerical values, the maximizer will seek to maximize the evaluation score and the minimizer will look through the least score as much as possible. Likewise, the objective of the player's opponent at the node representing its move called min node is to limit the value at that node. Min Max generates a game tree which serves as a graphical representation that encapsulates all the potential game states and moves within a sequential two-player game. Each node in the tree represents a specific game state, which includes the position of the pieces, the current player turns i.e., Min or Max.

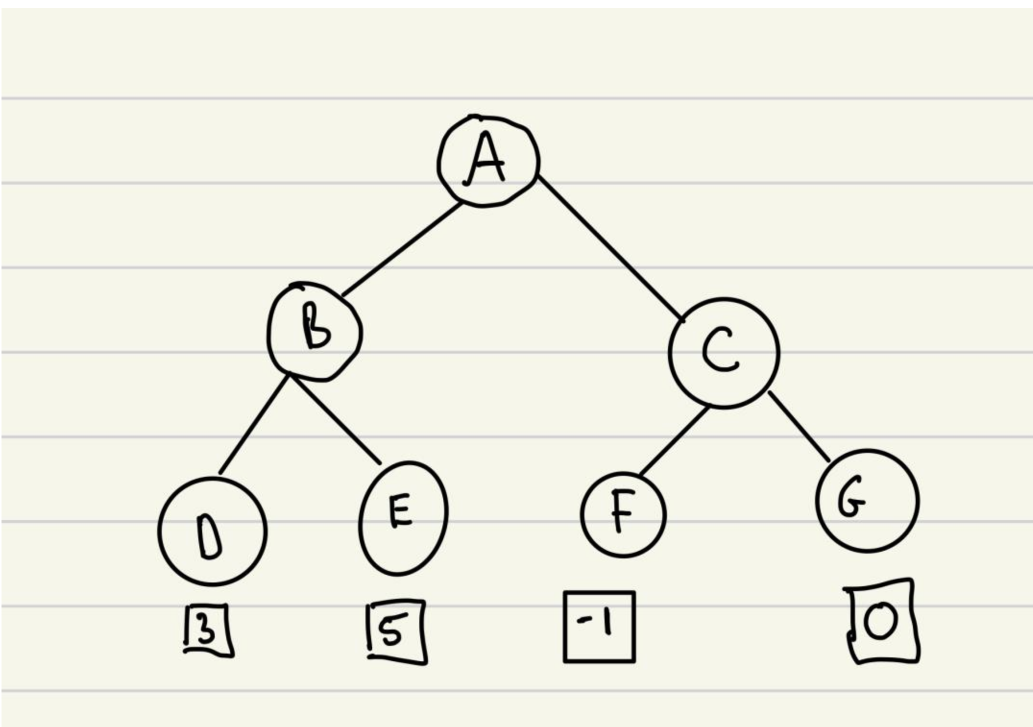

Figure 3: A simple example of minmax algorithm.

Minmax can be represented by a game tree where each node represents the value of game state starting from the root node (level 0) which is the initial game state and branching into child nodes at level one. Then again it will branch into grandchild nodes level 2 and so on until it reaches the terminal state. Connect Four Game can have up to 4.5 trillion combinations, branching all the nodes in Minmax search would be an impossible task for machines to compute in reasonable time. For this reason, we must limit our search depth level. Theoretically, the greater the search depth, the more knowledge the Minmax search algorithm must find the best move. So, we need to have search depth that's not so large that it takes too long to compute but, not very small that the search algorithm is provided with very little information.

3.2 Depth First Search

Depth First Search the minmax agent performs a depth-first search with a recursive search of the game tree. As a rule, the searching is represented as a tree data structure. The search tree is created by recursively expanding all nodes from the root node in a depth-first search manner. It primarily moves vertically down the whole length of the tree until it reaches the terminal nodes and then backtracks. It's important to note that the algorithm explores one branch of the search tree as deeply as possible before backtracking and moving to other branches. The algorithm can move horizontally or, among other sibling nodes. Depth First search is stark contrast to a breadth-first search, which does opposite for example it fundamentally moves horizontally across nodes and search one whole level at a time and work its way down the tree.

Search Depth (n)	Number of legal positions
2	644
4	8383
8	151039

Table 1: Number of Connect Four positions after n depth levels.

As you move deeper into the search tree, the number of legal positions grows significantly. This demonstrates the quadratic growth of the game tree Connect Four Game.

3.3 Evaluation Function

The Evaluation Function plays a pivotal role in guiding the Minimax algorithm to make optimal decisions in the game. The effectiveness of Minimax largely depends on the quality of the evaluation function used. In our project, we have developed three distinct evaluation functions tailored for the Connect Four game. These evaluation functions analyze the current state of the game by examining different squares on the board and assigning them appropriate values. These values reflect the desirability of each square, and the evaluation function ultimately returns a numerical score that informs the Minimax algorithm's decision-making process. The three evaluation functions we've designed will compete against each other, allowing us to assess their performance and determine which one is most effective in guiding the AI to make strategic moves.

3.4 Alpha-Beta pruning

Alpha-Beta pruning AI is an addition to Min Max algorithm which further optimizes the searching process. The word pruning means selectively removing branches and leaves from a tree. Alpha-Beta Pruning is primarily concerned with making the search tree smaller. It removes branches in the game tree which don’t need to be investigated since there is now a superior move accessible. Alpha-Beta pruning leverages the fact that it is not required to completely expand the search tree until terminal nodes to figure out the score of the position. Alpha Beta Pruning is thought of as an optimization strategy for the Min Max algorithm where the running is reduced by a huge factor. This enables looking through the nodes much faster and allows the algorithm to look deeper into the tree. For the algorithm to work two arguments are required alpha and beta by monitoring them and short-circuiting both max and min in the game tree. Alpha is the best move for the maximizer X and Beta is the best value for the minimizer (O).

When applied the Alpha-Beta Pruning algorithm on a standard minmax search tree, it returns a result of the same move as the minmax algorithm would return but a new trimmed tree will be considered that has some branches, which are impossible to affect the final decision of the minmax algorithm. As a result, more traceable problems will be considered rather than minmax computationally expensive ones. Figure 4 shows [4] an example of an application for the Alpha-Beta Pruning algorithm.

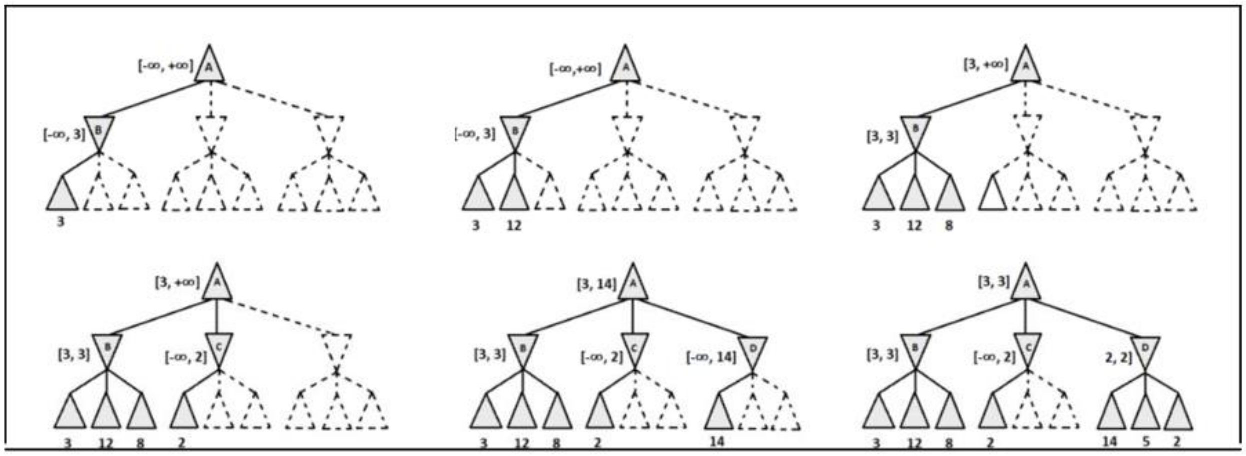

Figure 4: Alpha-Beta Prunning Game Tree [4]

In figure 4, we have several trees with different assigned to each node. Some nodes are marked with dots, indicating that there's no need to examine them further. We use the concepts of "Alpha" and "Beta" to track the best scores for the maximizing and minimizing players.

Let’s break down the process in greater detail:

- The initial state A is evaluated, and it generates child states B, C, and D. Starting with B, which is a Maximizer's turn:
    - B explores its children and finds a potential score of 3. This score updates Beta to 3, as Alpha remains at negative infinity.
- Moving up to A, a Maximizer's turn:
    - A now has a score of 3 to consider, as it's the best score from its children so far.
    - The process continues with C, which is a Minimizer's turn. C evaluates its children and finds a score of 2. This score updates Beta to 2.
    - However, A, the Maximizer, has a better score of 3 than its other child. This means C's branch can be pruned.
- Back to A's turn:
    - A has considered B and C, and its best score remains 3.

This process continues with other branches. The key pruning conditions are: if Benta is less than equal to alpha, then pruning should be done. In this case, pruning is applied at node C when Beta (2) is less than Alpha (3). In this explanation, the best score found is 3 [4].

4. Source Code Implementation

4.1 Development Environment and Programming Language

We chose the Python programming language due to its extensive collection of libraries that have been utilized in this project. For instance, one of our objectives is to measure the performance of the program by tabulating the memory usage of the Connect Four Game. Python provides a library called 'memory profiler,' which offers tools and functions used to track memory allocation and detect memory leaks.

In comparison to lower-level languages such as C++, Python stands out for its accessibility in memory profiling. In contrast, C++ lacks a built-in memory profiling library, and to measure memory in C++ often necessitates running the program on a different operating system, further complicating the process. Therefore, Python is undoubtedly a favorable choice for this project, making programmers' lives easier.

We run python on Visual Studio 2019, and I will further elaborate why Visual Studio is a good editor to use. Visual Studio Code is not only visually appealing to its users but also offers robust debugging tools to ensure that programs run as intended. The Visual Studio debugging feature is particularly valuable, enabling developers to set breakpoints and pause the program's execution at specific points. This allows the programmer to stop the flow of the program at specific locations whenever there’s problems in the code that need to be fixed. Furthermore, VS Code Debugging tool allows for testing different functions by inputting different values to variables, ensuring the output will always be consistent with the input. Furthermore, Visual Studio Code is supported by GitHub Desktop, simplifying the process of pulling and pushing code when collaborating with my groupmates. This integration streamlines version control and enhances team collaboration.

GitHub is the tool that has been used by all my group members for collaboration. We decided to use GitHub as it’s the main tool used by many professional software developers, and we need to get firsthand experience of sharing code using GitHub to further enhance our experience with version control. One feature of GitHub that I really liked is that it allows us to see the history of how the program has changed after each team member has pushed their changes.

4.2 Algorithm implementation

<u>Minmax-A-B pseudocode:</u>

```
function minimax_alpha_beta(node, depth, alpha, beta, maximizingPlayer)
    if depth is 0 or node is a terminal node
        return the heuristic value of node

    if maximizingPlayer
        bestValue = negative_infinity
        for each child in node
            value = minimax_alpha_beta(child, depth - 1, alpha, beta, False)
            bestValue = max(bestValue, value)
```

```
            alpha = max(alpha, bestValue)
            if beta <= alpha
                break
        return bestValue
    else
        bestValue = positive_infinity
        for each child in node
            value = minimax_alpha_beta(child, depth - 1, alpha, beta, True)
            bestValue = min(bestValue, value)
            beta = min(beta, bestValue)
            if beta <= alpha
                break

        return bestValue
```

<u>Algorithm Implementation in Detail:</u>

In straightforward terms, the AI player's goal is to find the optimal move for maximizing their chances of winning. To achieve this, we require a function that can anticipate future game states. This is where recursion comes into play, creating a tree encompassing all possible Connect Four Game states.

Now, the challenge arises: which tree node is the best choice? Randomly picking a node from the search tree would make our AI predictable and easy to beat. Hence, we must assign numerical values to evaluate these tree nodes, helping our AI player predict the best move based on the evaluation function.

However, we encounter some challenges. The search tree is extensive and navigating it through recursion can be time-consuming. To overcome this, we employ depth-limited search, which allows us to focus on specific levels of node expansion and so our search doesn't need to iterate to terminal nodes.

Moreover, the algorithm often investigates tree branches that don't impact the AI player strategic decisions and it doesn't make sense to use computational power to investigate these tree nodes.
To overcome these challenges, we implement Alpha-Beta Pruning. This optimization technique trims off search tree branches when a better move is already available. Alpha and beta values, representing the best options for the maximizer and minimizer, are continually updated based on available information. This gives our computer the advantage of using this extra memory space to look further into the search tree more optimally.

In conclusion, I hope I conveyed the message properly how this algorithm works and why it's the most important part of this project. As described above, the algorithm uses recursion, depth limited search, evaluation of game states and pruning, all these together give us the Min Max-AB algorithm.

4.3 Important features and highlights

Connect Four Game Logic

The program implements the game logic for Connect Four where the players make their moves in turn. Both players will try to get four connected squares, either vertically, horizontally, or diagonally. The first player who achieves one such group of four connected squares, wins the game. If all 42 squares are played and no player has achieved this goal, the game is drawn.

Evaluation Function

My Evaluation Function algorithm assesses game states for potential wins for the Min and Max players separately. The algorithm works by assigning a numerical value for the investigated game states. The Min Max algorithm then uses the information given by the Evaluation Function to decide the next move for the AI player.

5. Source Code

```
import os
import glob
import sys
import copy
import time as t
import psutil


ROWS = 6
COLS = 7

PLAYER_O = 'O'
PLAYER_X = 'X'
EMPTY_SPACE = '-'

debug = False
round = 0
winner = None
show_tree = False
counter = 0

# Evaluation Functions
# Cameron
def eval_function_1(board):
    """
    Calculate the difference between the number of PLAYER_X pieces
    and PLAYER_O pieces on the board.

    Positive value indicates an advantage for PLAYER_X
    Negative value indicates an advatange for PLAYER_O

    Args:
    - board (Board): The current state of the game board.

    Returns:
    - int: The difference between the number of PLAYER_X and PLAYER_O pieces
    """
    count_x = sum(row.count(PLAYER_X) for row in board.board)
    count_o = sum(row.count(PLAYER_O) for row in board.board)
    return count_x - count_o

# Dhruve
```

```
def eval_function_2(board):
    """
    Computes the difference between the number of the three-in-a-row
    pieces for PLAYER_X and PLAYER_O. It checks both horizontally and vertically.

    Args:
    - board (object): The current game board.

    Returns:
    - int: The difference in the number of three-in-a-row sequences between PLAYER_X
and PLAYER_O.
    """
    def count_threes(player):
        """
        Helper function to count the number of three-in-a-row sequences for a given
player.

        Args:
        - player (str): Either PLAYER_X or PLAYER_O.

        Returns:
        - int: The number of three-in-a-row sequences for the given player.
        """
        count = 0
        for row in range(ROWS):
            for col in range(COLS - 2):
                if board.board[row][col:col+3] == [player]*3:
                    count += 1
        for col in range(COLS):
            for row in range(ROWS - 2):
                if board.board[row][col] == board.board[row+1][col] ==
board.board[row+2][col] == player:
                    count += 1
        return count

    return count_threes(PLAYER_X) - count_threes(PLAYER_O)

# Youssef
def eval_function_3(board):
    board_array = board.board  # Convert the board to a 2D array

    def count_groups(board_array, player):
        count = 0
        # Check horizontally
```

```
        for row in range(ROWS):
            for col in range(COLS - 3):  # Need at least 4 consecutive spaces for a
win
                group = [board_array[row][col + i] for i in range(4)]
                count += group.count(player)

        # Check vertically
        for col in range(COLS):
            for row in range(ROWS - 3):  # Need at least 4 consecutive spaces for a
win
                group = [board_array[row + i][col] for i in range(4)]
                count += group.count(player)

        # Check diagonally (bottom-left to top-right)
        for row in range(3, ROWS):
            for col in range(COLS - 3):
                group = [board_array[row - i][col + i] for i in range(4)]
                count += group.count(player)

        # Check diagonally (top-left to bottom-right)
        for row in range(ROWS - 3):
            for col in range(COLS - 3):
                group = [board_array[row + i][col + i] for i in range(4)]
                count += group.count(player)

        return count

    # Calculate the total number of winning lines for MAX (Player X)
    max_player_groups = count_groups(board_array, PLAYER_X)

    # Calculate the total number of winning lines for the opponent (MIN, Player O)
    min_player_groups = count_groups(board_array, PLAYER_O)

    # Calculate the evaluation using the formula E(n) = M(n) - O(n)
    evaluation = max_player_groups - min_player_groups
    return evaluation


class Metrics:
    def __init__(self):
        self.nodes_generated = 0
        self.nodes_expanded = 0
        self.player_x_time = 0
        self.player_o_time = 0
```

```python
    def start_timer(self, player):
        if player == PLAYER_X:
            self.player_x_start_time = t.time()
        else:
            self.player_o_start_time = t.time()


    def stop_timer(self, player):
        if player == PLAYER_X:
            self.player_x_time += t.time() - self.player_x_start_time
        else:
            self.player_o_time += t.time() - self.player_o_start_time

    # def elasped_time(self, player):
    #     if player == PLAYER_X:
    #         return self.player_x_time
    #     else:
    #         return self.player_o_time


    def increment_nodes_generated(self):
        self.nodes_generated += 1

    def increment_nodes_expanded(self):
        self.nodes_expanded += 1


class Board:
    def __init__(self):
        self.rows = ROWS
        self.columns = COLS
        self.board = [[EMPTY_SPACE for _ in range(COLS)] for _ in range(ROWS)]
        self.turn = PLAYER_X

    def winner(self):
        """ Return the winner of the game. """
        bool = self.check_win()
        if self.turn == PLAYER_O:
            return PLAYER_X if bool else None
        if self.turn == PLAYER_X:
            return PLAYER_O if bool else None

        return None
```

```
    def display(self, file=None):
        output_stream = file if file is not None else sys.stdout
        global round, winner
        print("\n" + f"Round: {round + 1}", file=output_stream)
        print(f"Player {PLAYER_X if round % 2 == 0 else PLAYER_O} turn",
file=output_stream)
        print("-" * (COLS * 4 + 1), file=output_stream)

        winner = self.winner()

        for row in self.board:
            print("|", end=" ", file=output_stream)
            for cell in row:
                print(cell, end=" | ", file=output_stream)
            print("\n" + "-" * (COLS * 4 + 1), file=output_stream)
        if winner == PLAYER_O or winner == PLAYER_X:
            print(f"We have a winner! Player {winner} won!", file=output_stream)
        else:
            print("No player wins. It was a tie.", file=output_stream)

    def place_token(self, column):
        for i in range(self.rows - 1, -1, -1):
            if self.board[i][column] == EMPTY_SPACE:
                self.board[i][column] = self.turn
                self.switch_turn()
                return True
        return False

    def switch_turn(self):
        self.turn = PLAYER_O if self.turn == PLAYER_X else PLAYER_X

    def is_terminal(self):
        return self.check_win() or all(cell != EMPTY_SPACE for row in self.board for
cell in row)

    def check_win(self):
        """
        Check if there's a winning move on the board.
        A win is defined by four consecutive pieces of the same type a row, column,
or diagonal.

        Returns:
        - bool: True if a winning move exists, otherwise False.
        """
```

```
        # Check the rows for 4 consecutive pieces of the same type
        for row in range(ROWS):
            for col in range(COLS - 3): # Subtract 3 to prevent index out of the
bound error
                if self.board[row][col] == self.board[row][col + 1] ==
self.board[row][col + 2] == self.board[row][col + 3] and self.board[row][col] !=
EMPTY_SPACE:
                    return True

        # Check the columns for 4 consecutive pieces of the same type
        for col in range(COLS):
            for row in range(ROWS - 3): # Subtract 3 to prevent index out of the
bound error
                if self.board[row][col] == self.board[row + 1][col] == self.board[row
+ 2][col] == self.board[row + 3][col] and self.board[row][col] != EMPTY_SPACE:
                    return True

        # Check the diagonals for 4 consecutive pieces of the same type
        for row in range(ROWS - 3): # Subtract 3 to prevent index out of the bound
error
            for col in range(COLS - 3): # Subtract 3 to prevent index out of the
bound error
                # Check top-left to bottom-right diagonal
                if self.board[row][col] == self.board[row + 1][col + 1] ==
self.board[row + 2][col + 2] == self.board[row + 3][col + 3] and self.board[row][col]
!= EMPTY_SPACE:
                    return True
                # Check bottom-left to top-right diagonal
                if self.board[row + 3][col] == self.board[row + 2][col + 1] ==
self.board[row + 1][col + 2] == self.board[row][col + 3] and self.board[row + 3][col]
!= EMPTY_SPACE:
                    return True

        # No winning move found
        return False


def format_board_for_logging(board):
    """ Format the board state into a string representation for logging. """
    board_str = ""
    for row in board:
        board_str += ' '.join(row) + '\n'
    return board_str

def show_algorithm_tree(board, current_depth, scenario_counter):
```

```
    global round, counter
    if show_tree:
        with open(os.path.join("Project2_AI", "SearchTree",
f"scenario_{scenario_counter + 1}",f"round_{round + 1}.txt"), "a") as file:
            if current_depth == 1:
                file.write(f"\n=== New Board. Player {board.turn} ===\n")
            file.write(f"Depth: {current_depth}\n")
            file.write(format_board_for_logging(board.board))
            file.write("\n" + "-"*20 + "\n")  # Separator


def minimax_alpha_beta(board, current_depth, max_depth, current_player,
maximizing_player, alpha, beta, eval_function, metrics, scenario):
    """
    Implement the Minimax algorithm with alpha-beta pruning to find the best move for
a given player.

    Parameters:
    - board: The current game state.
    - depth: The current depth in the search tree.
    - current_player: The player whose turn it currently is.
    - maximizing_player: A flag indicating if the current move is a maximizing move.
    - alpha: The best value achieved so far by any choice the maximizer has made at
any choice point along the path.
    - beta: The smallest value achieved so far by any choice the minimizer has made
at any choice point along the path.
    - eval_function: The evaluation function used to evaluate the board state.

    Returns:
    - The best move's value from the current state.

    """
    global round, counter, show_tree
    counter += 1
    show_algorithm_tree(board, current_depth, scenario)
    # print(f"{current_depth}   ", end='')
    metrics.increment_nodes_expanded()
    # Best case: If we've reached the maximum depth or the board state is terminal
    if current_depth >= max_depth or board.is_terminal():
        # print(f"Board returned at Round: {round}")
        # Return the evaluation for the current board. Negate for 0 since it's the
minimizing player
        return eval_function(board) if current_player == PLAYER_X else -
eval_function(board)
```

```
    # Maximizing player's logic
    if maximizing_player:
        max_eval = float('-inf') # Initialiaze to negative infinity
        for col in range(board.columns):
            # Create a temporary copy of the board to simulate the move
            temp_board = copy.deepcopy(board)
            temp_board.place_token(col)

            # Recursive call to continute the search tree
            metrics.increment_nodes_generated()
            eval = minimax_alpha_beta(temp_board, current_depth + 1, max_depth,
board.turn, False, alpha, beta, eval_function, metrics, scenario)
            max_eval = max(max_eval, eval)
            # Alpha-beta pruning logic
            alpha = max(alpha, eval)
            if beta <= alpha: # If beta is less than or equal to alpha, prune the
branch
                break
        return max_eval

    # Minimizing player's logic
    else:
        min_eval = float('inf') # Initialiaze to positive infinity
        for col in range(board.columns):
            # Create a temporary copy of the board to simulate the move
            temp_board = copy.deepcopy(board)
            temp_board.place_token(col)

            # Recursive call to continue the search tree
            metrics.increment_nodes_generated()
            eval = minimax_alpha_beta(temp_board, current_depth + 1, max_depth,
board.turn, True, alpha, beta, eval_function, metrics, scenario)
            min_eval = min(min_eval, eval)

            # Alpha-beta pruning logic
            beta = min(beta, eval)
            if beta <= alpha: # If beta is less than or equal to alpha, prune the
branch
                break
        return min_eval

def get_best_move(board, max_depth, eval_function, metrics, scenario):
    best_move = -1
```

```
    best_value = float('-inf') if board.turn == PLAYER_X else float('inf')
    valid_moves = [col for col in range(board.columns) if board.board[0][col] ==
EMPTY_SPACE]  # Only consider columns that aren't full

    for col in valid_moves:
        temp_board = copy.deepcopy(board)
        temp_board.place_token(col)

        move_value = minimax_alpha_beta(temp_board, 1, max_depth, board.turn,
board.turn == PLAYER_O, float('-inf'), float('inf'), eval_function, metrics,
scenario)

        if board.turn == PLAYER_X and move_value > best_value:
            best_value = move_value
            best_move = col
        elif board.turn == PLAYER_O and move_value < best_value:
            best_value = move_value
            best_move = col

    return best_move


def print_metrics_to_file(metrics1, metrics2, filename, eval1, eval2, total_time,
mem_used, scenario_counter, board):
    """
    Write a comparison of metrics for two avaluation functions to its respective
file.
    Uses a table style format.

    Parameters:
    - metrics1: The metrics gathered from the first eval function.
    - metrics2: The metrics gathered from the second eval function.
    - filename: The name of the file where the comparison table will be appended on.
    """

    global counter
    header = "{:<25} {:<20} {:<20} {:<20} {:<20}\n".format("Metric Used", eval1,
eval2, "Difference", "Who did better?")
    divider = "-" * 105 + "\n"

    nodes_gen_better = eval1 if metrics1.nodes_generated < metrics2.nodes_generated
else eval2
    nodes_exp_better = eval1 if metrics1.nodes_expanded < metrics2.nodes_expanded
else eval2
    # print(f"Nodes Better: {nodes_exp_better} Super Meh: {nodes_gen_better}")
```

```
    if metrics1.player_x_time > metrics2.player_o_time:
        time_better = eval2
    elif metrics1.player_x_time < metrics2.player_o_time:
        time_better = eval1
    else:
        time_better = "No one."

    with open(filename, "a") as file:
        file.write(header)
        file.write(divider)
        file.write("{:<25} {:<20} {:<20} {:<20} {:<20}\n".format("Nodes Generated",
metrics1.nodes_generated, metrics2.nodes_generated, metrics1.nodes_generated -
metrics2.nodes_generated, nodes_gen_better))
        file.write("{:<25} {:<20} {:<20} {:<20} {:<20}\n".format("Nodes Expanded",
metrics1.nodes_expanded, metrics2.nodes_expanded, metrics1.nodes_expanded -
metrics2.nodes_expanded, nodes_exp_better))
        file.write("{:<25} {:<20.4f} {:<20.4f} {:<20.4f} {:<20}\n".format("Elapsed
Time", metrics1.player_x_time, metrics2.player_o_time, metrics1.player_x_time -
metrics2.player_o_time, time_better))

    # Write the tabulated results at the end
    with open(os.path.join("Project2_AI", "tabulation", "analysis.txt"), "a") as
file:
        file.write(f"Results for Scenario: {scenario_counter + 1}\n")
        file.write(f"Winner for this case: {board.winner()}\n")
        file.write(header)
        file.write(divider)
        file.write("{:<25} {:<20} {:<20} {:<20} {:<20}\n".format("Nodes Generated",
metrics1.nodes_generated, metrics2.nodes_generated, metrics1.nodes_generated -
metrics2.nodes_generated, nodes_gen_better))
        file.write("{:<25} {:<20} {:<20} {:<20} {:<20}\n".format("Nodes Expanded",
metrics1.nodes_expanded, metrics2.nodes_expanded, metrics1.nodes_expanded -
metrics2.nodes_expanded, nodes_exp_better))
        file.write("{:<25} {:<20.4f} {:<20.4f} {:<20.4f} {:<20}\n".format("Elapsed
Time", metrics1.player_x_time, metrics2.player_o_time, metrics1.player_x_time -
metrics2.player_o_time, time_better))
        file.write(f"Total Time Elasped {total_time} seconds.\n")
        file.write(f"Total Memory Used {mem_used} MB.\n")
        file.write(f"Total Boards Evaluated: {counter}\n")
        file.write('=' * 105 + '\n\n')

# Function removes all the contents in the directory
def remove_txt_files(directory):
    txt_files = os.path.join(directory, '**', '*.txt')
```

```
    txt_files = glob.glob(txt_files, recursive=True)
    for txt_file in txt_files:
        os.remove(txt_file)

def main():
    # Remove, only keep when using visual studio
    sys.argv.insert(1,'--nogui')
    sys.argv.insert(2,'--notree')
      #sys.argv.insert(2,'--')

    # Define scenarios
    scenarios = [
        {"max_func": eval_function_1, "max_depth": 2, "min_func": eval_function_2,
"min_depth": 2}, # Eval 1 v 2
        {"max_func": eval_function_1, "max_depth": 2, "min_func": eval_function_3,
"min_depth": 4}, # Eval 1 v 3
        {"max_func": eval_function_2, "max_depth": 2, "min_func": eval_function_3,
"min_depth": 8}, # Eval 2 v 3

        {"max_func": eval_function_1, "max_depth": 4, "min_func": eval_function_2,
"min_depth": 2}, # Eval 1 v 2
        {"max_func": eval_function_1, "max_depth": 4, "min_func": eval_function_3,
"min_depth": 4}, # Eval 1 v 3
        {"max_func": eval_function_2, "max_depth": 4, "min_func": eval_function_3,
"min_depth": 8}, # Eval 2 v 3

        {"max_func": eval_function_1, "max_depth": 8, "min_func": eval_function_2,
"min_depth": 2}, # Eval 1 v 2
        {"max_func": eval_function_1, "max_depth": 8, "min_func": eval_function_3,
"min_depth": 4}, # Eval 1 v 3
        {"max_func": eval_function_2, "max_depth": 8, "min_func": eval_function_3,
"min_depth": 8}, # Eval 2 v 3

    ]
    output_dir = os.path.join("Project2_AI", "Outputs")
    remove_txt_files(output_dir)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    output_tree_dir = os.path.join("Project2_AI", "SearchTree")
    remove_txt_files(output_tree_dir)
    if not os.path.exists(output_tree_dir):
        os.makedirs(output_tree_dir)
```

```
    tabulation_dir = os.path.join("Project2_AI", "tabulation")
    remove_txt_files(tabulation_dir)
    if not os.path.exists(tabulation_dir):
        os.makedirs(tabulation_dir)

    global round, counter, show_tree
    if '--notree' in sys.argv[2]:
        print("No tree will be generated")
        show_tree = False
    else:
        print("Tree will be generated. Please check the \"SearchTree\" folder")
        show_tree = True


    for i, scenario in enumerate(scenarios):
        counter = 0
        start_time = t.time()
        metrics_max = Metrics()
        metrics_min = Metrics()
        board = Board()
        scenarios_tree = os.path.join(output_tree_dir, f"scenario_{i + 1}")
        if not os.path.exists(scenarios_tree):
            os.makedirs(scenarios_tree)

        # Continue the loop until winner or board is full
        while not board.is_terminal():
            if board.turn == PLAYER_X:
                metrics_max.start_timer(PLAYER_X)
                move = get_best_move(board, scenario["max_depth"],
scenario["max_func"], metrics_max, i)
                metrics_max.stop_timer(PLAYER_X)
            else:
                metrics_min.start_timer(PLAYER_O)
                move = get_best_move(board, scenario["min_depth"],
scenario["min_func"], metrics_min, i)
                metrics_min.stop_timer(PLAYER_O)
            move_made = board.place_token(move)
            if not move_made:
                print("Invalid move attempted")
                break
            output_file_name = os.path.join(output_dir, f"output_scenario{i +
1}.txt")
            with open(output_file_name, "a") as output_file:
                board.display(output_file)
```

```
            round += 1

        memory_used = psutil.Process(os.getpid()).memory_info().rss / (1024 ** 2)
        print(f"Evaluated {counter} boards.")
        end_time = t.time()
        start_to_end = '%.2f' % (end_time - start_time)
        print(f"Scenario {i + 1} took {start_to_end} seconds.")
        print(f"Memory Used: {memory_used} MB")

        print_metrics_to_file(metrics_max, metrics_min, output_file_name,
scenario["max_func"].__name__, scenario["min_func"].__name__, start_to_end,
memory_used, i, board)
        counter = 0
        round = 0

    # Check if user would like GUI at the end
    if '--nogui' in sys.argv[1]:
        print("No GUI, please check analysis.txt to view the results instead.")
    else:
        print("Recongized user wants GUI. Although no function therefore just check
analysis.txt")


if __name__ == "__main__":
    main()
```

6. Copy of the program Run

Output Terminal

```
OUTPUT
[Running] python -u "c:\Users\youss\OneDrive\Documents\GitHub\CS4346_Project2\connect_4.py"
No tree will be generated
Evaluated 1332 boards.
Scenario 1 took 0.06 seconds.
Memory Used: 15.92578125 MB
Evaluated 3792 boards.
Scenario 2 took 0.29 seconds.
Memory Used: 16.0234375 MB
Evaluated 1895423 boards.
Scenario 3 took 138.62 seconds.
Memory Used: 15.9921875 MB
Evaluated 8575 boards.
Scenario 4 took 0.22 seconds.
Memory Used: 15.9921875 MB
Evaluated 5329 boards.
Scenario 5 took 0.35 seconds.
Memory Used: 15.9921875 MB
Evaluated 1903102 boards.
```

```
Scenario 6 took 135.04 seconds.
Memory Used: 15.9921875 MB
Evaluated 444529 boards.
Scenario 7 took 10.33 seconds.
Memory Used: 15.9921875 MB
Evaluated 76664 boards.
Scenario 8 took 2.08 seconds.
Memory Used: 15.9921875 MB
Evaluated 988330 boards.
Scenario 9 took 64.72 seconds.
Memory Used: 16.03515625 MB
No GUI, please check analysis.txt to view the results instead.

[Done] exited with code=0 in 352.238 seconds
```

Tables Produced

Output Scenario 1:

```
No player wins. It was a tie.
Metric Used              eval_function_1     eval_function_2     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          602                 560                 42                  eval_function_2
Nodes Expanded           688                 644                 44                  eval_function_2
Elapsed Time             0.0169              0.0270              -0.0101             eval_function_1
```

Output Scenario 2:

```
We have a winner! Player X won!
Metric Used              eval_function_1     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          189                 3554                -3365               eval_function_1
Nodes Expanded           217                 3575                -3358               eval_function_1
Elapsed Time             0.0062              0.2788              -0.2726             eval_function_1
```

Output Scenario 3:

```
We have a winner! Player O won!
Metric Used              eval_function_2     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          616                 1894632             -1894016            eval_function_2
Nodes Expanded           704                 1894719             -1894015            eval_function_2
Elapsed Time             0.0430              138.5450            -138.5021           eval_function_2
```

Output Scenario 4:

```
No player wins. It was a tie.
Metric Used              eval_function_1     eval_function_2     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          7845                560                 7285                eval_function_2
Nodes Expanded           7931                644                 7287                eval_function_2
Elapsed Time             0.1823              0.0261              0.1562              eval_function_2
```

Output Scenario 5:

```
We have a winner! Player X won!
Metric Used              eval_function_1     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          1726                3554                -1828               eval_function_1
Nodes Expanded           1754                3575                -1821               eval_function_1
Elapsed Time             0.0470              0.2953              -0.2483             eval_function_1
```

Output Scenario 6:

```
We have a winner! Player O won!
Metric Used              eval_function_2     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          8295                1894632             -1886337            eval_function_2
Nodes Expanded           8383                1894719             -1886336            eval_function_2
Elapsed Time             0.3140              134.6479            -134.3339           eval_function_2
```

Output Scenario 7:

```
No player wins. It was a tie.
Metric Used              eval_function_1     eval_function_2     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          443799              560                 443239              eval_function_2
Nodes Expanded           443885              644                 443241              eval_function_2
Elapsed Time             10.2821             0.0289              10.2532             eval_function_2
```

Output Scenario 8:

```
We have a winner! Player X won!
Metric Used              eval_function_1     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          73061               3554                69507               eval_function_3
Nodes Expanded           73089               3575                69514               eval_function_3
Elapsed Time             1.7782              0.2838              1.4944              eval_function_3
```

Output Scenario 9:

```
We have a winner! Player X won!
Metric Used              eval_function_2     eval_function_3     Difference          Who did better?
----------------------------------------------------------------------------------------------------
Nodes Generated          151039              837242              -686203             eval_function_2
Nodes Expanded           151067              837263              -686196             eval_function_2
Elapsed Time             5.4548              59.2528             -53.7980            eval_function_2
```

<u>Analysis Text</u>

```
Results for Scenario: 1
Winner for this case: None
Metric Used                eval_function_1     eval_function_2     Difference          Who did better?
-----------------------------------------------------------------------------------------------------------
Nodes Generated            602                 560                 42                  eval_function_2
Nodes Expanded             688                 644                 44                  eval_function_2
Elapsed Time               0.0169              0.0270              -0.0101             eval_function_1
Total Time Elasped 0.06 seconds.
Total Memory Used 15.92578125 MB.
Total Boards Evaluated: 1332
===========================================================================================================

Results for Scenario: 2
Winner for this case: X
Metric Used                eval_function_1     eval_function_3     Difference          Who did better?
-----------------------------------------------------------------------------------------------------------
Nodes Generated            189                 3554                -3365               eval_function_1
Nodes Expanded             217                 3575                -3358               eval_function_1
Elapsed Time               0.0062              0.2788              -0.2726             eval_function_1
Total Time Elasped 0.29 seconds.
Total Memory Used 16.0234375 MB.
Total Boards Evaluated: 3792
===========================================================================================================
```

```
Results for Scenario: 3
Winner for this case: O
Metric Used                eval_function_2     eval_function_3     Difference          Who did better?
-----------------------------------------------------------------------------------------------------------
Nodes Generated            616                 1894632             -1894016            eval_function_2
Nodes Expanded             704                 1894719             -1894015            eval_function_2
Elapsed Time               0.0430              138.5450            -138.5021           eval_function_2
Total Time Elasped 138.62 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 1895423
===========================================================================================================

Results for Scenario: 4
Winner for this case: None
Metric Used                eval_function_1     eval_function_2     Difference          Who did better?
-----------------------------------------------------------------------------------------------------------
Nodes Generated            7845                560                 7285                eval_function_2
Nodes Expanded             7931                644                 7287                eval_function_2
Elapsed Time               0.1823              0.0261              0.1562              eval_function_2
Total Time Elasped 0.22 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 8575
===========================================================================================================
```

```
Results for Scenario: 5
Winner for this case: X
Metric Used              eval_function_1      eval_function_3      Difference           Who did better?
----------------------------------------------------------------------------------------------------------
Nodes Generated          1726                 3554                 -1828                eval_function_1
Nodes Expanded           1754                 3575                 -1821                eval_function_1
Elapsed Time             0.0470               0.2953               -0.2483              eval_function_1
Total Time Elasped 0.35 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 5329
==========================================================================================================

Results for Scenario: 6
Winner for this case: O
Metric Used              eval_function_2      eval_function_3      Difference           Who did better?
----------------------------------------------------------------------------------------------------------
Nodes Generated          8295                 1894632              -1886337             eval_function_2
Nodes Expanded           8383                 1894719              -1886336             eval_function_2
Elapsed Time             0.3140               134.6479             -134.3339            eval_function_2
Total Time Elasped 135.04 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 1903102
==========================================================================================================
```

```
Results for Scenario: 7
Winner for this case: None
Metric Used              eval_function_1      eval_function_2      Difference           Who did better?
----------------------------------------------------------------------------------------------------------
Nodes Generated          443799               560                  443239               eval_function_2
Nodes Expanded           443885               644                  443241               eval_function_2
Elapsed Time             10.2821              0.0289               10.2532              eval_function_2
Total Time Elasped 10.33 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 444529
==========================================================================================================

Results for Scenario: 8
Winner for this case: X
Metric Used              eval_function_1      eval_function_3      Difference           Who did better?
----------------------------------------------------------------------------------------------------------
Nodes Generated          73061                3554                 69507                eval_function_3
Nodes Expanded           73089                3575                 69514                eval_function_3
Elapsed Time             1.7782               0.2838               1.4944               eval_function_3
Total Time Elasped 2.08 seconds.
Total Memory Used 15.9921875 MB.
Total Boards Evaluated: 76664
==========================================================================================================
```

```
Results for Scenario: 9
Winner for this case: X
Metric Used              eval_function_2      eval_function_3      Difference           Who did better?
----------------------------------------------------------------------------------------------------------
Nodes Generated          151039               837242               -686203              eval_function_2
Nodes Expanded           151067               837263               -686196              eval_function_2
Elapsed Time             5.4548               59.2528              -53.7980             eval_function_2
Total Time Elasped 64.72 seconds.
Total Memory Used 16.03515625 MB.
Total Boards Evaluated: 988330
==========================================================================================================
```

Search Tree Example Output

```
--------------------
Depth: 2
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - - - -
X - O - X - -

--------------------
Depth: 2
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - - - -
X - - O X - -

--------------------
Depth: 2
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - O - -
X - - - X - -

--------------------
```

```
Depth: 2
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - - - -
X - - - X O -

-------------------
Depth: 2
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - - - -
X - - - X - O

-------------------

=== New Board. Player O ===
Depth: 1
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - - -
O - - - - - -
X - - - - X -

-------------------
```

Example of the Board Class Display Function

```
Round: 6
Player O turn
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
No player wins. It was a tie.

Round: 7
Player X turn
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| - | - | - | - | - | - | - |
-----------------------------
| X | - | - | - | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
| X | - | - | O | - | - | - |
-----------------------------
```

7. Analysis of Program

<u>Metric Class:</u>

After my group mates developed each their own evaluation function, we needed to compare their performance by recording execution time, number of nodes generated. Initially this was a challenging task because we needed to create many variables that will hold this information, which will make our program more complicated to read and harder to change. So instead of using Procedural Programming, we decided to focus on OOP principles to organize our code to use objects and classes. The Metric Class in our program will be solely responsible for measuring the performance of our evaluation functions. It contains attributes which will hold information about our Evaluation function and methods that will create the data to display them to the terminal. This dataset will successfully allow us to decide which Evaluation function algorithm performs best, to be used by our AI player in the future.

<u>The Minmax Algorithm:</u>

At the start of this project, there were many discussions about how the Min Max algorithm should operate. There were mistakes that I have corrected for the function to work properly. Initially, the first move of AI player 1 was made randomly. This could make our AI significantly weaker since it's supposed to work on finding the best path to victory using the Min Max algorithm. I corrected this mistake by having the first player make its first move using the Min Max algorithm and evaluation function values to decide which square on the Connect Four board is the best. Furthermore, the Alpha Beta Pruning algorithm, the values of alpha and beta need to be continuously updated in the care if there is better path found. I have helped implement this feature because if Alpha and Beta values were not changed then the optimization technique will not work as intended.

<u>The Board Class:</u>

As discussed before, the program uses various OOP principles to make the usability of the code easier without the need for creating many different variables. The Board class that I have helped implement is responsible for outputting board states, responsible for checking wins diagonally, vertically, and horizontally and declaring the ‘winner’ player if 4 squares connect otherwise it declares ‘draw’. Furthermore, the class switches turn between AI players and places the disc in the selected column by dropping it in the lowest unoccupied row.

8. Tabulation of Results

Scenario 1: Max (EV#1) with cutoff depth 2 verses Min (EV #2), with cutoff depth 2

Metric Used	Evaluation function 1	Evaluation function 2	Difference	Performed Best
Node Generated	602	644	42	Evaluation Function 2
Node Expanded	688	644	44	Evaluation Function 2
Elapsed Time	0.0196	0.0167	0.0029	Evaluation Function 2
No Player wins. It was a tie.				

Scenario 2: Max (EV#1) with cutoff depth 2 verses Min (EV #3), with cutoff depth 4

Metric Used	Evaluation function 1	Evaluation function 3	Difference	Performed Best
Node Generated	189	3554	-3365	Evaluation Function 1
Node Expanded	217	3575	-3358	Evaluation Function 1
Elapsed Time	0.0068	0.2692	-0.2624	Evaluation Function 1
Player X Won!				

Scenario 3: Max (EV#2) with cutoff depth2 verses Min (EV #3), with cutoff depth 8

Metric Used	Evaluation function 2	Evaluation function 3	Difference	Performed Best
Node Generated	616	1894632	-1894016	Evaluation Function 2
Node Expanded	704	1894719	-1894015	Evaluation Function 2
Elapsed Time	0.0190	137.0434	-137.0243	Evaluation Function 2
Player O Won!				

Scenario 4: Max (EV#1) with cutoff depth 4 verses Min (EV #2), with cutoff depth2

Metric Used	Evaluation function 1	Evaluation function 2	Difference	Performed Best
Node Generated	7845	560	7289	Evaluation Function 2
Node Expanded	7931	644	7287	Evaluation Function 2
Elapsed Time	0.1811	0.0265	0.1546	Evaluation Function 2
No player wins. It was a tie.				

Scenario 5: Max (EV#1) with cutoff depth 4 verses Min (EV #3) with cutoff depth 4

Metric Used	Evaluation function 1	Evaluation function 3	Difference	Performed Best
Node Generated	1726	3554	-1828	Evaluation Function 1
Node Expanded	1754	3575	-1821	Evaluation Function 1
Elapsed Time	0.0430	0.2941	-0.2511	Evaluation Function 1
Player X Won!				

Scenario 6: Max (EV#2) with cutoff depth 4 verses Min (EV #3), with cutoff depth 8

Metric Used	Evaluation function 2	Evaluation function 3	Difference	Performed Best
Node Generated	8295	1894632	-1886337	Evaluation Function 2
Node Expanded	8383	1894719	-1886336	Evaluation Function 2
Elapsed Time	0.3173	136.4283	-136.4283	Evaluation Function 2
Player O Won!				

Scenario 7: Max (EV#1) with cutoff depth 8 verses Min (EV #2) with cutoff depth 2

Metric Used	Evaluation function 1	Evaluation function 2	Difference	Performed Best
Node Generated	443799	560	443239	Evaluation Function 2
Node Expanded	443885	644	443241	Evaluation Function 2
Elapsed Time	10.8089	0.0190	10.7899	Evaluation Function 2
No player wins. It was a tie				

Scenario 8: Max (EV#1) with cutoff depth 8 verses Min (EV #3) with cutoff depth 4

Metric Used	Evaluation function 1	Evaluation function 3	Difference	Performed Best
Node Generated	73061	3554	69507	Evaluation Function 3
Node Expanded	73089	3575	69514	Evaluation Function 3
Elapsed Time	2.0411	0.3003	1.7408	Evaluation Function 3
Player X Won!				

Scenario 9: Max (EV#2) with cutoff depth 8 verses Min (EV #3) with cutoff depth 8

Metric Used	Evaluation function 2	Evaluation function 3	Difference	Performed Best
Node Generated	151039	837242	-686203	Evaluation Function 2
Node Expanded	151067	837263	-686196	Evaluation Function 2
Elapsed Time	5.7369	61.7622	-56.0252	Evaluation Function 2
Player X Won!				

9. Analysis of Results

9.1 Memory Usage

The memory usage is consistent throughout the program which is 16 MB. Considering that the Min Max produces large search trees with extensive nodes and uses three different Evaluation Function algorithms to evaluate these nodes, 15 mb is a reasonable memory usage for the Connect Four Game.

9.2 Results of the nine tables

By looking through the table I was able to derive important information about the performance of Evaluation Functions. To start simply, I'm going to compare the evaluation functions performance at the same depth.

<u>How does Evaluation Function 3 perform against Evaluation Function 2?</u>

Depth of 8: Evaluation Function 3 generates and expands more nodes. The execution time for Evaluation Function 3 is higher by drastic amount taking more than 60 seconds to fully execute. Furthermore, Evaluation Function 3 fails and loses the game.

<u>How does Evaluation Function 3 perform against Evaluation Function 1?</u>

Depth of 4: Evaluation Function 3 generates and expands more nodes. The execution time for Evaluation Function 3 is higher by more than 10 seconds. Evaluation Function 3 has failed to execute more efficiently and has also failed to win the game at depth of 4 against Evaluation Function 1.

<u>Does the Evaluation Function 3 ever win?</u>

Evaluation Function 3 has lost the fight against the other functions at the same depth in terms of optimality and winning the game. However, Evaluation Function 3 does win sometimes if it has the depth advantage. So, let's analyze those results:

EV #2 depth 4 vs EV #3 depth 8: Evaluation Function 3 looks deeper into the search and subsequently wins the game. However, it's important to note that Evaluation Function doesn't seem to operate effectively because it took 136 seconds to execute.

EV #2 depth 2 vs EV #3 depth 4: Evaluation Function 3 looks deeper into the search tree and wins the game. However, doesn't work optimally as it generated massive amounts of nodes in the search tree.

Which Evaluation Function is the best and why?

By looking through the 9 tables I have concluded that Evaluation Function 3 is the worst in terms of space and time complexity. So, let's dive deeper to compare Evaluation Function 2 versus Evaluation Function 1:

By analyzing all the different depths, I have concluded It's a draw, neither Eval #1 nor Eval #2 win at any depth and Eval #2 expands less nodes and has lower execution time compared to Eval #1.

However, my goal is to find the best Evaluation Function, so I'm going to create a table for winning/losing statistics to give us a better picture.

Evaluation Functions	Games won at any depth	Winning percentage
1	3	50%
2	1	16%
3	2	30%

9.3 Evaluation Functions ranking winning percentage

1. Evaluation Function 1
2. Evaluation Function 3
3. Evaluation Function 2

9.4 Evaluation Functions optimality ranking

1. Evaluation 2
2. Evaluation 1
3. Evaluation 3

9.5 Further analysis of Evaluation Function 2

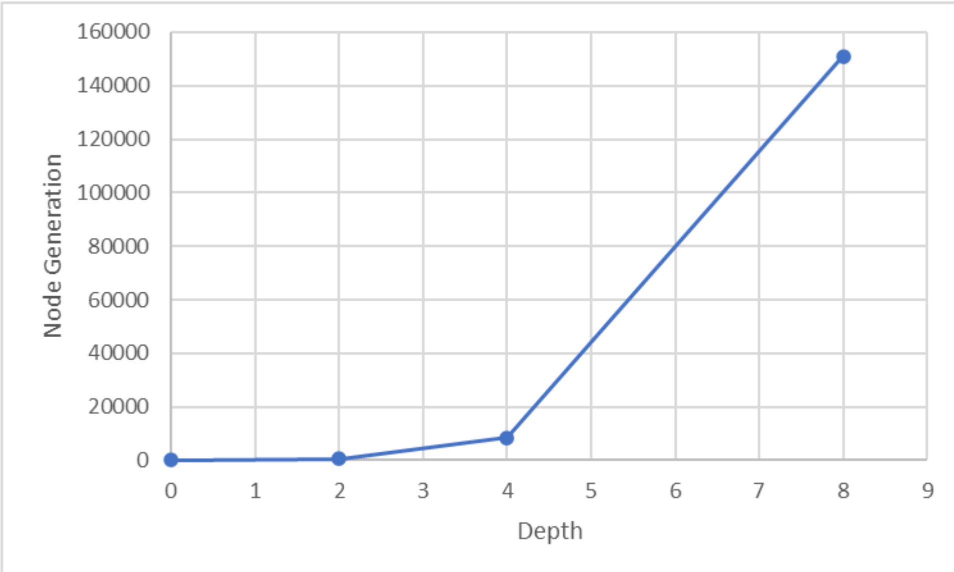

Figure 5 the results of evaluation function 2 node generation in a graph

Evaluation Function 2, as shown in Figure 5, illustrates the relationship between depth and node generation. The best-fit line equation, in the form of *Y= mx + b*, is expressed as Y = 5299X^2 – 27925x + 35298. Surprisingly, this represents a quadratic relationship. Initially, I predicted the relationship would be exponential based on my knowledge of the Minimax algorithm; however, Figure 5 contradicts my prediction. Further analysis is needed to better understand the relationship with greater depth in the search tree to make more informative decisions.

9.6 Implementation and modified features

Evaluation Function 3 optimally performs worst, let's deep dive into the algorithmic perspective. Evaluation Function 3's biggest mistake was that it checks for four connecting squares instead of three. In perspective, the Evaluation Function doesn't look for potential wins, it only checks if there is a win. So, all three connecting squares are simply ignored by the function which is a fatal mistake and doesn't allow the function to work as intended.

I have decided to keep the Evaluation Function 3 the same and not optimize to greater emphasize the impact of the function on the minmax algorithm.

10. Conclusion

In this paper, a two-player sequential game called Connect Four has been formed and a self-learning player is acquainted with it. The AI player is an intelligent agent that performs based on thinking and acting. The AI player improves the game playing by foreseeing the moves of the rival ahead of time. The AI agent is trained using two algorithms Minmax and Alpha Beta Pruning, and a relative analysis we made using them. The analysis revealed that AI agent has greater difficulty and higher winning chance with more depth.

This project was a great experience for me and took me all the time throughout the semester learning deeply how the Min Max algorithm and how to implement it. This project stands out to me, because for the first time I was able to create a self-playing AI player. It was a great challenge that seemed impossible, however, with great hard work collaborating with my group mates we successfully created the project. Throughout the project I have been exposed to new python libraries with extensive comprehensive documentation. I was also surprised, at first, to find this game specifically explored by the research community even until very recently. Which shows that having a game solved doesn't mean there is no usefulness in exercising to create an AI player for it.

11. Team Members Contributions

<u>Dhruv Mistry:</u>

- I had great discussions with him about the Min Max algorithm which helped further my understanding of how to do this project.
- Extensive experience with coding skills which has effectively helped to convert our ideas to code.
- Great communicator and successfully set up GitHub repository to share our code live.
- Highly available to answer questions and great at explaining the parts of code that he worked on.

<u>Cameron Salisbury:</u>

- The creator of the best evaluation function in the program.
- Highly available, great discussions about the minmax algorithm which helped further understanding of this project.

12. References

[1] ALPHA-BETA PRUNING-A STREAMLINE APPROACH FOR PERCEPTIVE GAME PLAYING

[2] Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game

[3] Evaluation of the Use of Minimax Search in Connect-4 —How Does the Minimax Search Algorithm Perform in Connect-4 with Increasing Grid Sizes?

[4] Artificial intelligence a modern approach third edition