

Connect 4 game

Ai project

Aya Elneanaei Fouda 202002609

Nada Hussain Mokhtar 211002028

Dr: Hala Zayed

**Table of Contents**

[**1****.1Abstract**](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758516) 3

[**1.2Introduction**](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758517) 3

[**1.3 literature survey**](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758519) 4

[***1.3.1 Minimax Algorithm***](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758520) 4

[***1.3.2 Alpha Beta pruning***](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758521)

[**1.4 Methodology**](file:///D:\sophomore%20semester%202\Writing%20Skills\engl%20paper\final%20draft.docx#_Toc132758523)

**1.5 Results…………………………………………………………………………………………**

**1.6 Conclusion…………………………………………………………………………………**

**1.7 Future work………………………………………………………………….**

**1.8 References…………………………………………………………………….**

* 1. **Abstract**

This paper presents the development of a graphical user interface (GUI) for playing Connect Four, a popular two-player board game. The GUI incorporates three levels of difficulty: easy, medium, and hard. Each level utilizes different algorithms and strategies to provide varying gameplay experiences. The implementation of the GUI aims to provide an interactive platform for users to play Connect Four against computer opponents of different skill levels.

* 1. **Introduction**

Connect Four is a strategic board game that requires players to strategically drop colored tokens into a grid to connect a line of four tokens horizontally, vertically, or diagonally. With its inherent complexity, Connect Four has been a subject of interest in the field of artificial intelligence, offering a challenging environment for developing intelligent gameplay algorithms. In this paper, we present the development of a graphical user interface (GUI) for playing Connect Four. The GUI incorporates three levels of difficulty: easy, medium, and hard. Each level utilizes different algorithms and strategies to provide a distinct gameplay experience. The easy level serves as an introductory level, where the computer opponent makes random moves. This level allows novice players to familiarize themselves with the game and its dynamics, without facing a strong strategic opponent. The medium level employs the Minimax algorithm with alpha-beta pruning. This algorithm explores the game tree to determine the optimal move by considering all possible future moves and their outcomes. By utilizing alpha-beta pruning, the search space is significantly reduced, leading to improved computational efficiency and more challenging gameplay. The hard level employs a combination of move evaluation, heuristics, and randomization to enhance gameplay. Move evaluation involves assigning values to potential moves based on their desirability, while heuristics guide decision-making by incorporating domain-specific knowledge. Randomization adds an element of unpredictability, making the computer opponent's moves more varied and challenging. Through the development of this GUI, players can now engage in Connect Four matches against computer opponents of different skill levels. The GUI not only provides an interactive platform for gameplay but also offers an opportunity to experience the varying degrees of difficulty and strategic decision-making involved in Connect Four.

* 1. **literature survey**

In this section, we will provide an overview of the algorithms that can be used to play Connect Four and explain how each of these algorithms works.

*1.3.1Minimax Algorithm*

To create game states, the mini-max algorithm is employed. The category of backtracking algorithms includes Minimax. This method is used in game theory and decision-making to determine the best course of action for a provided that the adversary plays effectively as well. It is frequently employed in two-player games like chess and tic tac toe where players take turns making their moves. Mini max has two participants, known as the maximizer and minimizer, as is clear. While the minimizer attempts to obtain the lowest score possible, the maximizer strives to achieve the best score attainable. With each board state, there is a value connected. If the maximizer has the upper hand in a particular state, the board score will typically be positive. In that board condition, if the minimizer has the advantage, it will typically be some negative number. The values of the board are calculated using a heuristic function. Every sort of game utilizes a different heuristic function. Think about the ideal binary tree in

Fig. 1. It has four final states, and there are numerous ways to get from the root to each of its four leaves [1].

A picture containing diagram, sketch, line, circle

Description automatically generated

**Fig 1**: Initial Game Tree in case of Minimax algorithm [1].

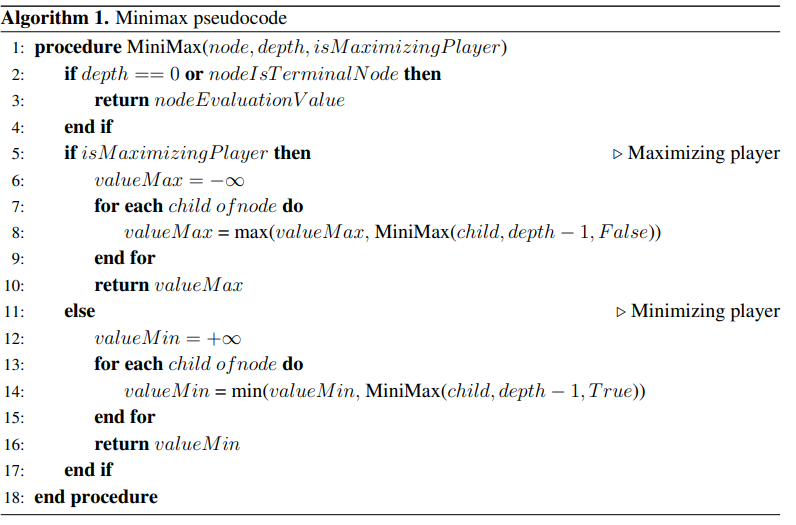
The decision is reached by backtracking after attempting all feasible steps because it is a backtracking algorithm. The maximizer has the choice to move left or right at first. Assuming it turns left, the minimizer will then take the wheel. Between 2 and 4, the minimizer must decide. Since the goal of a minimizer is to minimize its value, it selects the value that is least of the two, which is equal to 2. After the following maximizer, it is the minimizer's turn. A value between 1 and 8 must be selected by the minimizer. It will undoubtedly select 1 over 8. The maximizer will now get to select between 2 and 1 in this round. The goal of a maximizer is to obtain the highest value possible, thus it selects the greater value of the two, which is 2. The maximizer ultimately obtains the ideal value of 2, and its best course of action is to turn left. In the example, a relatively little game tree is utilized to illustrate the principles simply and readily. However, in the Connect-4 game, a much larger game tree with many more game states is produced. It is practically hard to locate and calculate each game state. The tree may have a six-depth and a seven-branch structure. As a result, many different game states are computed [1].

*A picture containing diagram, sketch, line, circle

Description automatically generated*

**Fig 2:** Final Game Tree in case of Mini-max [1].

According to the maximizer's left and right moves, there are two potential scores, as shown by the tree above. Since it is assumed that the adversary always plays optimally, the minimizer will never choose the value 8 from the appropriate sub tree. In the scenario above, a player has only two options, however there are typically lots more options. In that situation, repeating all of the potential motions yields the maximum/minimum. For the example given, the scores kept in the leaves of the game tree were generated at random, but these values are obtained for a regular game. According to the study, the AI generates the game state after 2799 iterations and 33.00 milliseconds of computation time for a depth 4 difficulty when the user plays the game alone and selects the minimax operating method[1].



Algorithm 1 contains the Minimax algorithm's pseudocode. The algorithm's initial state would resemble this: MiniMax(originNode, depth, True). Depth here denotes how many levels the game tree will have. The game-state is represented by OriginNode and node variables, whereas isA Boolean variable called MaximizingPlayer determines whether the node should be treated as a maximizer or a minimizer. If a node is a terminal node, either through reaching a certain depth or one of the players achieving the win condition, the first if statement yields the node's score. By repeatedly using the MiniMax function, the code in lines 5 to 17 attempts to obtain the greatest value for the maximizer and the least for the minimizer. A game state that occurred after the game state in the node variable is represented by a child variable [1].

*1.3.2 Alpha Beta pruning*

The term "pruning" refers to a tree's selective removal of branches and leaves. The goal of alpha-beta pruning is to make a search tree smaller. The advantage of alpha-beta pruning is that it is not necessary to fully analyze all potential game pathways to determine the position's score. Therefore, it is crucial to give the minimax algorithm certain stopping conditions so that it can stop searching through a region of the tree once it discovers the guaranteed minimum or maximum at that level. The original minimax algorithm uses a depth-first search approach and traverses the tree from left to right while also aiming for the deepest possible level in the tree. It then determines the values that must be directly allocated to nodes above it, without ever consulting other tree branches. Additionally, the halting condition option forces minimax to behave as it did in the past while making judgments, which improves the algorithm's performance. In this sense, the minimax technique's execution time can be drastically decreased by using Alpha Beta Pruning as an optimization tool. This makes it possible to look through the nodes much more quickly and even to access deeper levels of the game tree. It eliminates the game tree branches that are no longer necessary because a better option is already available. The improvement is crucial since the AI doesn't need to take too long to make a move, but with alpha-beta pruning, this can be done much more quickly. By keeping an eye on them and short-circuiting both max and min in the game tree, two more parameters, alpha, and beta, are provided to minimax to carry out the algorithm. The maximizer's best move in the current state or before it during the maximizer turn is alpha. Beta is the highest value that the minimizer can guarantee in either the prior state or the present state. α> =ß is the prerequisite for alpha-beta pruning.

Each node must keep track of its alpha and beta values. The general rule is that, given a state's' anywhere in the tree, if a player decides to perform an activity 'm,' which is perceived as 's,' and has been discovered, at that point's,' will never be reached [2]

A picture containing text, screenshot, font

Description automatically generated

**Fig 3**: The pseudo-code for Alpha-Beta Pruning [2]

A few trees are shown in Figure 4 with varying scores given to each node. There are certain nodes that are dotted, which means there isn’t a good reason to audit them. Beta is estimated to be + and Alpha is estimated to be -. Beta can only be modified while it is in the Minimizer turn, which will then be passed on to the child/state, while Alpha can only be updated when it is in the Maximizer turn. The potential activity of the initial state A is increased, resulting in the creation of the states B, C, and D starting with the primary index B. The turn of minimizer B follows, during which the potential activity of B is expanded and its score in the child node is evaluated. While Alpha is still, Beta has been modified with a fanciful value of Beta=3=min (3, 12, 8,). Assuming a depth of 2, an A is given for a minimum score of 3. The Maximizer turn A follows, which is updated with Alpha=3=(-, 3), before going on to the following child, the C state, which has the Minimizer turn C with passed Alpha parameters (3). C determines that the primary child received a score of 2, which is updated as Beta=2=min (, 2). The maximizer eventually finds a better move with a score of 3, which can still be checked for another potential action in C. The term “pruning process” refers to the entire technique. The best score, 2, is therefore returned. In a basic sense, pruning is required if beta=alpha. Max (alpha, 2) = alpha is computed to maximize turn A, after which the procedure proceeds with last child C. To minimize turn C, min (, 14) is calculated, beta is updated to 14, and then it is determined whether beta equals alpha (14=3), which has not been trimmed. In essence, the kid beta == alpha (5 == 3) is not pruned. The process checks for beta =alpha (2 =3) that needs to be trimmed at the end. No more child nodes are present; hence no other nodes cannot be pruned. If the child that returned 2 is discovered at first, pruning can proceed. As a result, score 2 is changed to A. After the A turn that maximizes, max (alpha, 2) =alpha is calculated. The process concludes at this point. The highest score ultimately discovered in the tree is 3, which is returned. The possible advantages of this technique over the Random AI and Minimax AI strategies are currently being investigated with the understanding of how the Alpha-Beta Pruning algorithm operates [2]

*A picture containing sketch, drawing, white

Description automatically generated*

**Fig 4**: Alpha-Beta Pruning Game Tree [2]

* 1. **Methodology**

We implement a Connect Four game with a graphical user interface (GUI) implemented using the tkinter library. The game has three levels (easy, medium, hard), and there is also an option to play a typical game against an AI opponent or with another player or play Ai against Ai. In 201.file we use necessary libraries such as tkinter for GUI, sys for system-related operations, and ImageTk and Image from PIL (Python Imaging Library) for image handling.It defines different functions for each game level (easy\_mode(), medium\_mode(), default\_mode()), a function for displaying game rules (RULE\_mode()), and a function for playing a sample game against an AI opponent (AI()).Each game level function creates a new window for the menu selection using tk.Toplevel(root) and sets the window size and position on the screen. It loads a background image and creates a background label to display it.

It creates labels and buttons for the menu options (two players or AI opponents) and associates them with corresponding functions using the command parameter.

When a menu option is selected, a message box is displayed to show the selected option, and then the menu window is destroyed. Depending on the option, the corresponding game or AI function is called. The RULE\_mode() function creates a window to display the game rules and includes a button to close the window. The main part of the script sets up the main window for the Connect Four game. It loads a background image, creates a background label, and creates a title label. It creates buttons for different game levels (easy, medium, hard) and associates them with their respective functions. It creates a button for playing a sample game against an AI opponent and a button for displaying the game rules. The main window is displayed using root.mainloop().

In the Connect\_4.py we import numpy as np: This imports the NumPy library, which is used for numerical operations. It provides a multidimensional array object and various functions for working with arrays.import pygame: This imports the Pygame library, which is used for creating games and multimedia applications.import sys: This imports the sys module, which provides access to some variables used or maintained by the interpreter and functions that interact with the interpreter.import math: This imports the math module, which provides mathematical functions and constants.

Defining the Game () function: This function contains the main logic of the Connect Four game.

Defining constants: BLUE, BLACK, RED, and YELLOW: These variables represent RGB color values used for drawing the game board and game pieces.

ROW\_COUNT and COLUMN\_COUNT: These variables represent the number of rows and columns in the game board.

Defining helper functions:

create\_board(): This function creates an empty game board using NumPy's zeros() function and returns it. drop\_piece(board, row, col, piece): This function places a game piece (1 or 2) at the specified row and column on the game board. is\_valid\_location(board, col): This function checks if a column is a valid move by verifying if the bottom row of that column is empty.

get\_next\_open\_row(board, col): This function returns the next available row in a given column.

print\_board(board): This function prints the game board (flipped vertically) using NumPy's flip() function.

winning\_move(board, piece): This function checks if a player has won the game by checking for four consecutive game pieces in a row, column, or diagonal.

Defining the draw\_board(board) function: This function is responsible for drawing the game board and the game pieces on the Pygame window. It uses nested loops to iterate through the rows and columns of the game board and uses Pygame's drawing functions to draw rectangles and circles.

Initializing the game:

Creating the game board using create\_board().

Printing the initial game board using print\_board(board).

Initializing variables for the game state (game\_over and turn).

Initializing Pygame and setting up the game window.

Game loop:

The loop runs until the game is over.It handles various events, such as quitting the game, mouse motion, and mouse button clicks.On mouse motion, it updates the position of the game piece indicator on the screen.On mouse button click, it performs the following actions:Checks if the selected column is a valid move.Finds the next available row in the selected column.Drops the player's game piece on the board.Checks if the player has won the game.Updates the game board and draws the updated board.Switches the turn to the next player.If the game is over, it waits for 3 seconds before exiting.The Game() function is currently commented out at the end. To run the game, you can uncomment the line #Game().

In the file named easy.py we use randomly selects a column for the AI player's move to use it as an easy level. We defining the easy() function: This function contains the main logic of the Connect Four game with an AI opponent.

Defining helper functions:

create\_board(): This function creates an empty game board using NumPy's zeros() function and returns it. drop\_piece(board, row, col, piece): This function places a game piece (1 or 2) at the specified row and column on the game board. is\_valid\_location(board, col): This function checks if a column is a valid move by verifying if the bottom row of that column is empty.

get\_next\_open\_row(board, col): This function returns the next available row in a given column.

winning\_move(board, piece): This function checks if a player has won the game by checking for four consecutive game pieces in a row, column, or diagonal. Defining the draw\_board(board) function: This function is responsible for drawing the game board and the game pieces on the Pygame window. It uses nested loops to iterate through the rows and columns of the game board and uses Pygame's drawing functions to draw rectangles and circles.

Initializing the game: Initializing variables for the game state (ROW\_COUNT, COLUMN\_COUNT, BLUE, BLACK, RED, YELLOW, board, game\_over, and turn).

Initializing Pygame and setting up the game window.

Game loop:

The loop runs until the game is over.

It handles various events, such as quitting the game, mouse motion, and mouse button clicks.

On mouse motion, it updates the position of the game piece indicator on the screen.

On mouse button click, it performs the following actions: Checks if the selected column is a valid move.Finds the next available row in the selected column.Drops the player's game piece on the board.Checks if the player has won the game.Updates the game board and draws the updated board.Switches the turn to the next player.If it's the AI player's turn (turn == 1), it selects a column randomly and places its game piece.Checks if the AI player has won the game.

Updates the game board and draws the updated board. Switches the turn to the next player.

If the game is over, it waits for 3 seconds before exiting.

In medium\_Ai.py file we are using the Pygame library. It allows two players to take turns placing their respective pieces on a grid. The objective of the game is to connect four pieces in a row, either horizontally, vertically, or diagonally. we use two main algorithms: the minimax algorithm and the alpha-beta pruning technique. The medium function is the main function that runs the game. It initializes the game variables and sets up the Pygame window. It contains the game loop where players take turns making moves until a winner is determined or the game ends in a draw. create\_board: Creates and returns an empty game board represented as a NumPy array.

drop\_piece: Places a game piece on the board at the specified row and column.

is\_valid\_location: Checks if a column is a valid move (i.e., the top row of that column is empty).

get\_next\_open\_row: Finds the next available row in a column for placing a game piece.

print\_board: Prints the game board to the console. winning\_move: Checks if a player has achieved a winning move by connecting four pieces. evaluate\_window: Evaluates a window of four consecutive pieces and assigns a score based on the number of player and empty pieces.

score\_position: Scores the current position on the board for the specified player by evaluating all possible windows. is\_terminal\_node: Checks if the current game state is a terminal node (i.e., the game has ended). minimax: Implements the minimax algorithm with alpha-beta pruning to determine the best move for the AI player. get\_valid\_locations: Returns a list of valid column locations for making a move. pick\_best\_move: Chooses the best column for the AI player based on the maximum score obtained from evaluating all possible moves. draw\_board: Draws the game board and the pieces on the Pygame window.

In hard.py file we use strategies commonly used in game playing algorithms, such as:

Move Evaluation: The AI\_move() function evaluates possible moves to determine the best move for the AI player. It checks if the AI can win in the next move and makes that move if possible. It also checks if the player can win in the next move and blocks them. If neither condition is met, it chooses a random valid move. Although this implementation does not use advanced algorithms like minimax or alpha-beta pruning, it applies simple heuristics to make strategic decisions.

Heuristics: The AI player uses simple heuristics to make decisions. It prioritizes winning by checking if it can win in the next move. If not, it looks for opportunities to block the opponent from winning. Finally, it chooses a random valid move as a fallback strategy. While this approach is not optimal, it provides basic gameplay for the AI opponent.

Randomization: The AI\_move() function uses numpy.random.choice() to select a random valid move when no winning or blocking moves are available. This introduces an element of randomness and prevents the AI from making the same move repeatedly.

* 1. **Results**

Our GUI:

*A screenshot of a game

Description automatically generated with medium confidence*

*A screenshot of a computer

Description automatically generated A screenshot of a game

Description automatically generated with medium confidence*

You can play with ai or other player for each level.

**1.6 Conclusion**

In this paper, we presented the development of a graphical user interface (GUI) for playing Connect Four. The GUI incorporates three levels of difficulty: easy, medium, and hard. Each level utilizes different algorithms and strategies to provide a unique gameplay experience.

The easy level allows novice players to learn and enjoy the game without facing a strong opponent. The computer opponent makes random moves, providing a simple and introductory gameplay experience. The medium level employs the Minimax algorithm with alpha-beta pruning, offering a more challenging gameplay experience. The computer opponent considers future moves and optimizes decision-making, creating a stronger strategic opponent. The hard level combines move evaluation, heuristics, and randomization to enhance gameplay. This level offers the most advanced and unpredictable gameplay experience, incorporating strategic decision-making, domain-specific knowledge, and varied moves from the computer opponent.

The development of this GUI provides players with the opportunity to play Connect Four against computer opponents of different skill levels, offering a range of challenging gameplay experiences. The GUI serves as a platform for users to engage with the game and explore the strategies and decision-making involved in Connect Four. Overall, the development of this GUI enhances the accessibility and enjoyment of Connect Four, while also showcasing the potential of different algorithms and strategies in the field of artificial intelligence and game theory. The GUI encourages further exploration and innovation in developing intelligent gameplay systems for board games like Connect Four.

**1.8 References**

[1]Alpha-Beta Pruning in Mini-Max Algorithm –An Optimized Approach for a Connect-4 Game. (2018, April). *International Research Journal of Engineering and Technology (IRJET)*.

[2] *الباحث العلمي من Google*. (n.d.). https://scholar.google.com/scholar?hl=ar&as\_sdt=0%2C5&q=Board+Game+AI+Development+Using+Alpha-Beta+Pruning&btnG=