<u>Approved Project</u>
Connect four, Playing against an AI

<u>Analysis</u>
<u>2.1</u>

Connect Four being a well known board game, when gathering my objectives i did a research based investigation rather than interviewing people as i found it more appropriate for a game.

During my research my main focus was to make sure I knew how the game works. Eg strategies and methods that will give a better chance of winning and how to meet these win conditions. When playing against another human being we can't always identify the opponent having a three in a row and is one step away from winning. But with a computer it checks how it will affect the gameboard when a piece is placed in each column therefore will always try to prevent you from winning. This means that the only way to beat a structured AI is to formulate a plan that eventually gives you two options to win which results in the AI's inability to prevent the user from winning

Being new to coding an AI I had to research algorithms that can help guide me to writing an AI to play againsts. During my research I found a well known algorithm called " MINI MAX" using this algorithm that allows the AI to be more advanced that allows the AI to look into how many moves ahead depending on the AI's "Depth".

The last part was efficient as the game is already designed. I have to make a board that is sized 7 by 6 as that is how the board is structured. And functions that allow the user to input which column they want to put their peice in and will then print out the whole gameboard with the piece that the user has chosen to place included each time. Between each move there will be a seperate function that constantly checks for four in a row to see if the Player or AI has won.

2.2
The main purpose of the AI is to create a game that requires the user playing to think and structure a good routine for setting up the perfect situation to win. Practice makes perfect, but parents and friends can often be occupied and busy with other material. The game aims to allow people who want to improve in connect four to be able to do that without requiring another person to play against.

Advantages :
- Does not require another player to play against
- Help and improve the players, tactically
- Easy to access and load

Disadvantages :
- AI can be said to run a certain algorithm and once if figured out can be easily beaten.

2.3

Model Of The Project

2.4

No.	Objective	Performance Criteria
1	Create a game board screen for the player to make his/her decision on where to place the next piece	Simple and clear board that allows to clearly visually empty spaces and differentiate between pieces that are placed from the player and the AI
2	A function that asks for the players input to choose where they want to put then next piece	Ensures that the piece is correctly placed on to the gameboard where the user wanted to
3	Function that checks if the Player or the AI has one and outputs a message which player has won.	Constantly checks the whole of the game board after each piece input checks for 4 in a row. Verticles, Horizontals and Diagonals

4	Function to check for if the game is drew	Checks the game board to see if the whole board is filled with no spaces left, output game state as “Draw” and ends programm
5	Function that checks if any of the columns are full.	Will check the gameboard to see if the columns are full. Once the column is full the Player and AI are not allowed to choose that column and an error message will be outputted .
6	AI prioritises the middle column as this generates the highest probability for winning.	AI when all moves are equal in value will always prioritise the middle.
7	AI piotises the next move to make and attempts to make a two in a row and 3 in a row. Looks into all direction	AI moves will be adjacent or on top of the previous placed to make a two in a row. And 3 in a row instead of 2 when available. Looking into horizontal, vertical and also diagonals .
8	AI will place the winning piece as highest priority	When there is a space where creates a 4 in a row it will ignore other options and win the game
9	AI to know when a column is full	When a column is full the AI will not consider the column to be an option will only look into the other available columns
10	AI to be able to defend against the player as 2nd highest priority	When the opponent has a three in a row the AI will know to place its next piece to prevent the player from winning.
11	Implement MiniMax Algorithm	Implementing mini max allows the AI to look into moves ahead. This increases the difficulty of the AI .
12	Implement Alpha - Beta Pruning	Saves time when running mini max with a big depth. Going through the tree eliminating Nodes that cannot be

2.5

<u>Consideration Of The Different Types Of Solution</u>

For my project I have decided to use python. Although it is the language that we are learning, this means this programming language will be the most comfortable one to use. It allows me to use functions to structure my code to be easier to read. Furthermore, Being more familiar with the language will save time for searching up syntax and how the language properly works, making my code easier to debug. Python also allows other modules to be implemented , for example if i wanted to add a graphic interface using Python's module " Pygame" can be changed into a more eye catching and colourful game to play.

3.1

My Critical Path

1. The game allows the player to play against the AI in a game of Connect Four
2. When the game starts there will be an output printed out asking for the column number where the player wants to insert their peice.
3. Program must have a visible gameboard that is 7 By 6 in size.
4. Program must allow users to differentiate between the AI pieces and their own piece.
5. AI will respond with the best move subsequently after the player has made their move.
6. Program must constantly check if either the AI or the Player has won after each new piece is placed
7. Program to output a message when the game is a draw with no winners
8. Program must repeat asking for player input and inputting A piece until there is a winning or draw
9. Program must stop and output the correct winner, either the player or the AI when they achieved a four in a row in any direction.

Prototyping

Sections	**Prototype 1**	**Prototype 2**	**Final Product**
Analysis	Having a working connect four game that functions correctly.	Using the connect four game, replace player 2 with a an AI to play against	Implement the Mini Max Algorithm. Increases the difficulty of the Bot. Also Add Alpha -Beta

			Pruning, Saving time when depth increases
Design	-Gameboard created -2 user input -Player 1 and Player 2 Pieces -Win Condition Checker	-Remove Player 2 inputs -AI able to look into all 7 possible moves and output the best move	Using the existing pseudocode, apply the minimax algorithm to existing AI
Technical Solution	Game will ask for player “input()” and add it to the correct places on the gameboard. Function “Game_Condition_Checker” to check if the board has a 4 in a row.	Points are assigned to different situations. For example a 4 in a row will be worth 100 points and 3 in a row will be 50. In each 7 move the one that has the most points will be chosen to be the AI ‘s move.	The AI will look into future moves depending on the depth placed on the mini max algorithm. Which increases the difficulty of the bot. Eg. If the player is 2 moves away from winning
Testing	I played a game of connect four with a friend. Checking if the column we chose is where the piece is dropped. Intentionally made 4 in a row in all directions to check if “Game_Condition_Checker” will be able to detect a 4 in a row.	During testing i will for example make a 3 in a row and see if the AI move will Block it. Ai should prioritise middle on its first move. And if the AI is always attempting to make 2 and 3 in a row around the map in all directions.	Setting up a 2 in a row. If the minimax algorithm is working. Its next move should be to prevent me to making a 3 in a row instead of a attacking move as the bot is set to prioritize defensive moves
Evaluation	Success, game runs until either player 1 or player 2 has won, constantly asking for inputs from both players with columns they chose correctly shown on the game board. Correctly output which player has won	Success, when the player has a 3 in a row on the board. The AI will place its next move to block the win. And likewise prioritises the center for its first move. Constantly making 2 and 3 in a row and even winning at times.	Failure, when tried the program does not work properly as even when turned on it does not look into future moves. Problem with the implementation while using pseudo code, having trouble understanding how to implement.

4 Documenting Design Section

4.1

Entity Relationship Model

```
Player 1 ------------ <-Column Choice->-----------AI
First Name              Row 1
Surname                 Row 2
Age                     Row 3
                        Row 4  ---------------------- Gameboard
                        Row 5                          Column Number
                        Row 6                          Row Number
```

Row 7

tblPlayer (<u>FirstName</u>, Surname, Age)

tblColumnNumber(<u>Column 1</u>, Column 2 , Column 3 , Column 4 , Column 5 , Column 6,Column 7)

tblGameboard(<u>ColumnNumber , Row Number)</u>

Normalisation.

tblPlayer (<u>FirstName</u>, Surname, Age)

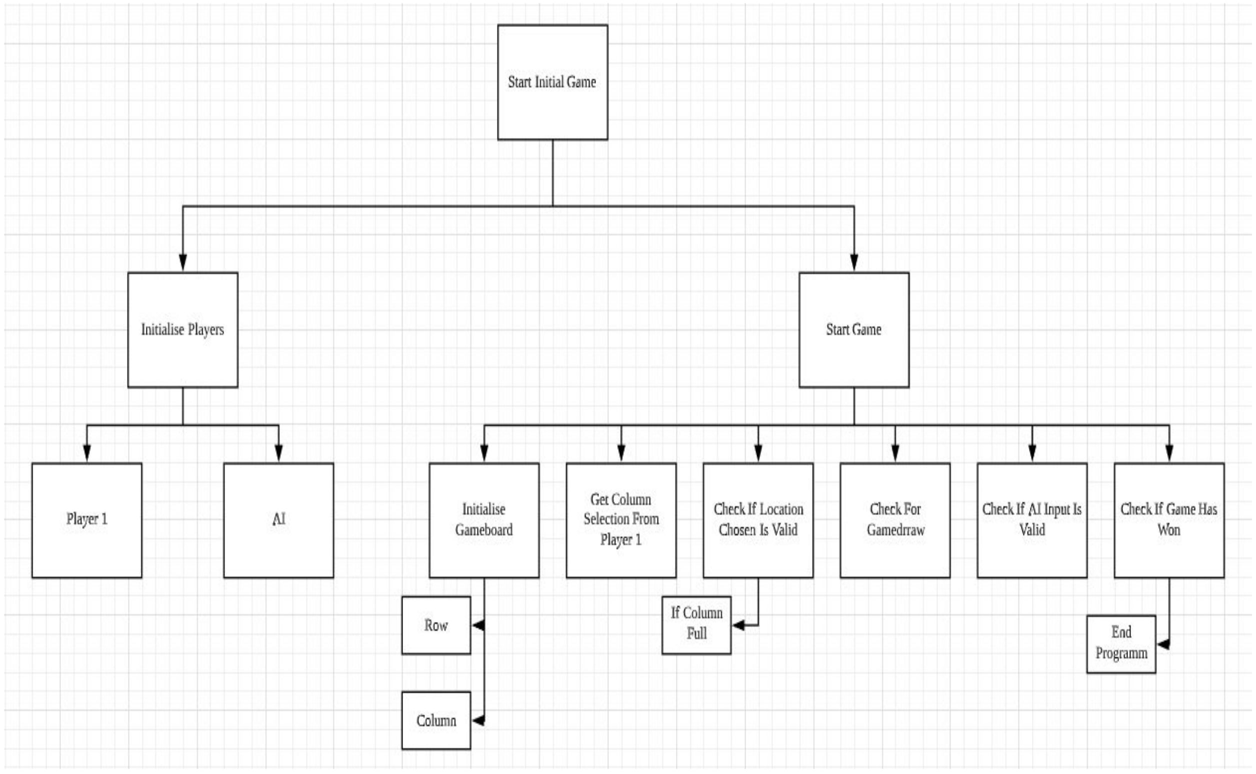
tblGameboard(<u>ColumnNumber , Row Number)</u>

tblColumnNumber(<u>Column 1</u>, Column 2 , Column 3 , Column 4 , Column 5 , Column 6, Column 7)

tblRowNumber(<u>Row 1</u>, Row 2 ,Row 3 , Row 4, Row 5, Row 6)

<u>System Design</u>

Top Down Diagram



Data Structure

Data Item	**Data Type**	**Validation**	**Sample Data**
Player First Name	**String**		**“Owen”**
Player Surname	**String**		**“Chan”**
Player Age	**Integer**		**“18”**
Gameboard	**List**		**“- - - - 1 - 2”**
Gameboard X Cord	**Integer**	**1 <= Xcord <= 7**	**4**
Gameboard Y Cord	**Integer**	**1 <= Ycord <= 6**	**3**
AI Piece	**Integer**	**AI = 2**	**2**
Player Piece	**Integer**	**Player = 1**	**1**
AI Best Move	**List , Integer**	**Biggest Integer From List**	**100**

Interface Design

A simple interface will be created minimising time spent on it.

The Gameboard.

- Contains Lists of “ - - - - - - - “
- 6 of the above list will be combined into one big list
- Easy to manipulate eg using [5][1]
- Changing the ‘-’ to ‘1’ or ‘2’ depending on the piece

Piece

- Using the number 1 will be assigned to Player Piece
- Using the number 2 will be assigned to Player Piece

User Input

- A clear grammatically correct sentence will be formed at the start of the programm to ask for which column Player 1 chooses to insert their piece

- Correct Error output will be output.
    - When the column is full
    - Column Number is too big or too small

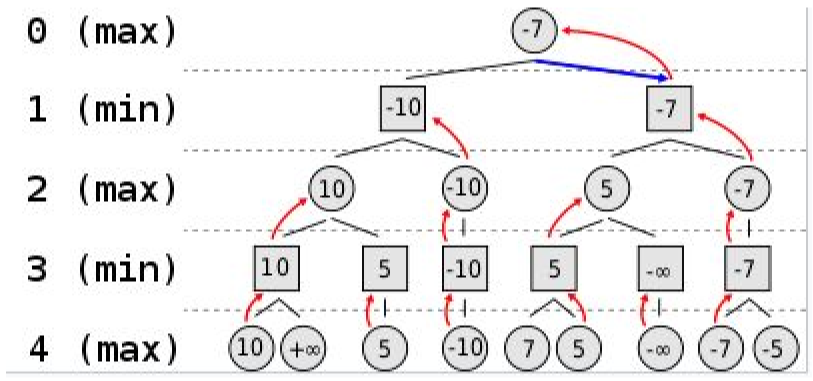
  Error Output will repeat until receiving a valid answer
- Clear Output when either the game is drawn. Or either player 1 or 2 has won the game.
    - “ Player 2 won ! “
    - “Player 1 won ! “
    - “Draw”

<u>Algorithms</u>

Minimax Algorithm

PseudoCode From https://en.wikipedia.org/wiki/Minimax

The minimax algorithm allows the AI to look into future moves to decide the best most that it can make. It looks in terms of the worst and best case scenario. The minimizing player for the AI will be the player. And the maximising player will be the AI that always aims to get the best move. The minimax algorithm can be visualized using a Tree Diagram



Tree diagram is from https://en.wikipedia.org/wiki/Minimax

We can see there is value assigned to the nodes. The depth of the tree shows how many moves ahead the AI is looking at. Each node represents each move's independent number of how impactful it will be in the game. Alternating the minimum value and the maximum value increases the difficulty of the AI as it is making the best move both offensively and defensively. In most cases the AI will place its piece where it has the highest chance of producing a 4 in a row . And also the move denies the player the potential of winning.

5 Technical Solution

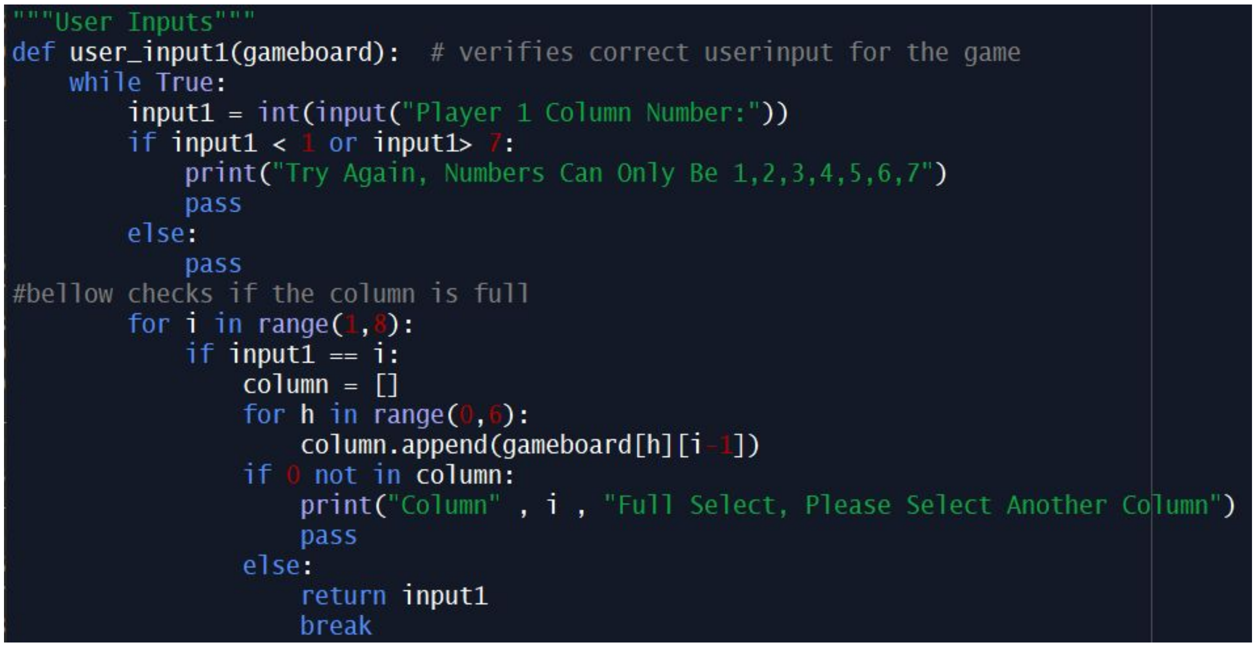
Procedures And Variables

Procedures	**Purpose**
board()	Creates gameboard in correct size
printboard()	Prints the game board
user_input1()	Asks for which column Player 1 Chooses
Board_updater()	Implements the Pieces from AI and Player into the gameboard in the correct spaces
win_condition_checker()	Check the game board for a 4 in a row.

horizontal()	Checks for horizontal 4 in a row
vertical ()	Checks for Verticle 4 in a row
diaganol_ne()	Checks for 4 in a row in diagonally
diaganol_nw()	Checks for 4 in a row in diagonally in opposite direction
game_draw()	Checks If the game board is full for Draw
freespace()	For AI to know the available Positions on a game board.
scoring_position()	Scores each 7 position for AI
best_position()	Gets the best score and outputs the coordinates of the position with the highest score
Minimax ()	The Minimax Algorithm
Main()	Main game function

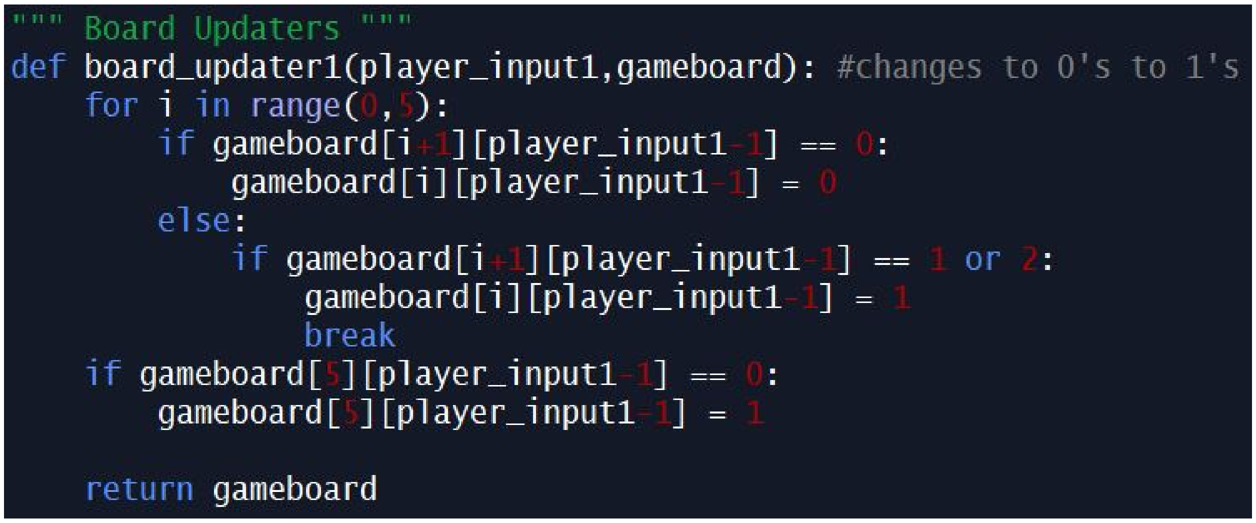
Code Annotation

The code above sets up a list of 7, 0’s to represent an empty space. 6 of the same list will be produced and appended to one big list that makes the gameboard. This allows easy manipulation when pieces are added as I can simply change the variables in the list by using game board[2][1] to change to 0 to correct the user piece. Printboard function just simply makes it neat and easier to comprehend as it will look like a connected four board.



```
"""User Inputs"""
def user_input1(gameboard):  # verifies correct userinput for the game
    while True:
        input1 = int(input("Player 1 Column Number:"))
        if input1 < 1 or input1> 7:
            print("Try Again, Numbers Can Only Be 1,2,3,4,5,6,7")
            pass
        else:
            pass
#bellow checks if the column is full
        for i in range(1,8):
            if input1 == i:
                column = []
                for h in range(0,6):
                    column.append(gameboard[h][i-1])
                if 0 not in column:
                    print("Column" , i , "Full Select, Please Select Another Column")
                    pass
                else:
                    return input1
                    break
```

The code above shows how the program asks the player for where it wants to place its piece that must be between 1 and 7. If the player inputs a number that is bigger or smaller the program will ask for another input, asking them to try again. The Second part is a check to see if the column is already full and will then output asking to choose another column as there is no available space.



```
""" Board Updaters """
def board_updater1(player_input1,gameboard): #changes to 0's to 1's
    for i in range(0,5):
        if gameboard[i+1][player_input1-1] == 0:
            gameboard[i][player_input1-1] = 0
        else:
            if gameboard[i+1][player_input1-1] == 1 or 2:
                gameboard[i][player_input1-1] = 1
                break
    if gameboard[5][player_input1-1] == 0:
        gameboard[5][player_input1-1] = 1

    return gameboard
```

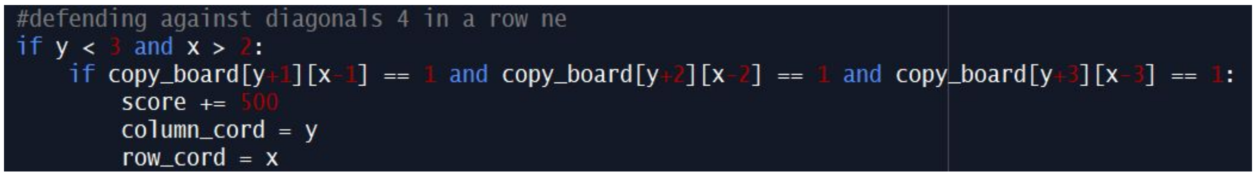
The function aboves helps update the player 1 input into the gameboard. The function detects and sees if there is already an existing piece either an ai peice or a player piece. If the column already has one it will simply place it above the piece according to whose turn it is.

The above code is to check the gameboard to see if either Player 1 or the AI has won the game. This is done by having a function each that iterates through the whole gameboard to see if there is a 4 in a row. I have only taken a screenshot of horizontal and vertical but in the actual code the two diagonal ways to win the game exist as well. The function basically compares the next 3 places to see if they are all 1's or 2's. Eg for horizontal it will look at the next 3 locations on its right.

The Code above is to check if the game is at a draw state and will end the program while outputting to the user saying the game is drawn.   This is done by iterating through every position of the gameboard to see how many of the spaces are filled with the players and AI piece. Having a 6 * 7 game boards have a total of 42 spaces. Therefore it returns the amount of space that is occupied and if the value is 42 the game must be at a draw state.

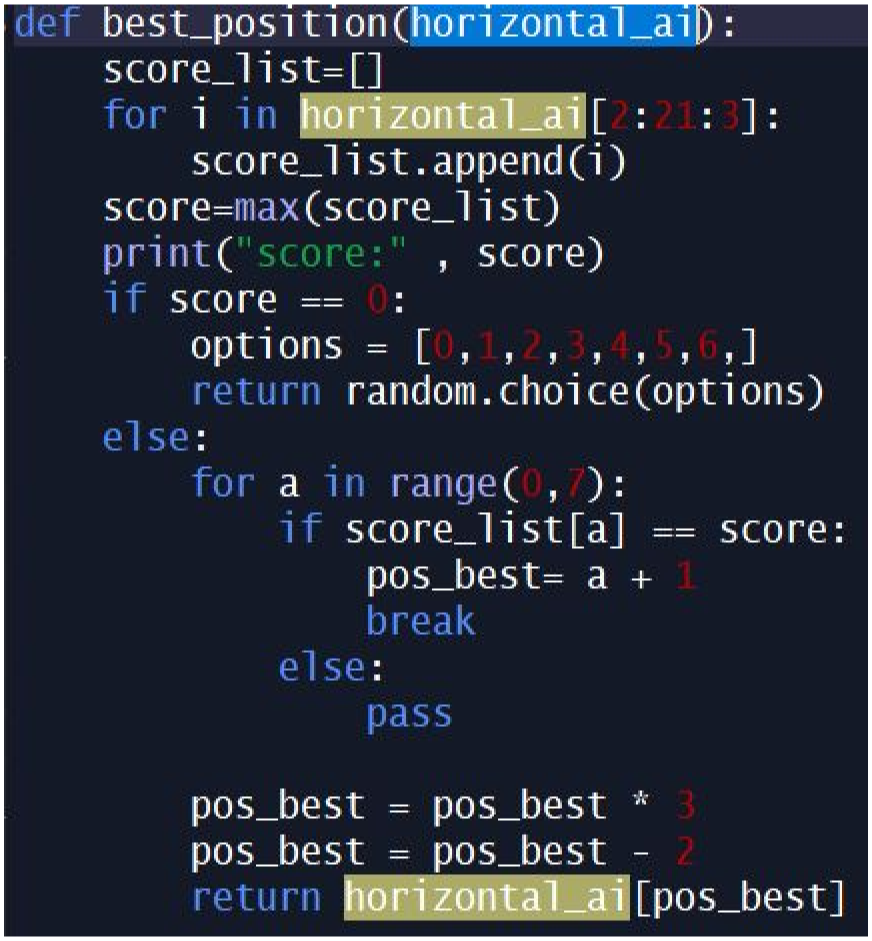
The above code is to help the Ai to visualise the available spaces that it can put its piece on its move. The function takes and goes down each column to see where it has the available space and for each column it takes it all into a list. Enumerate allows that freespace to be assigned according to which column it represents so that there will be a x and y output assigned on the freespace the AI can look into for the highest score move. An x and y output is needed to help manipulate the game board “gameboard[x][y]”

From the list above we see that column 3 has a value of 5 as the Ai prioritises the centre column as the centre has a high priority as previously mentioned increases chance of winning.



```
#defending against diagonals 4 in a row ne
if y < 3 and x > 2:
    if copy_board[y+1][x-1] == 1 and copy_board[y+2][x-2] == 1 and copy_board[y+3][x-3] == 1:
        score += 500
        column_cord = y
        row_cord = x
```

The code above determines the best move for the AI. From before the enumerated values allows us to get the x and y cord of the first available free space in each column. Using the freespace we can score its next move by copying the existing game board and mimicking dropping an AI piece in each column. As you can see "copy_board" we are manipulating it by adding a peice. The two screenshots from above shows defensive and offensive scores that are added up together to get the best score. As you can see the program looks at the copy_board to see if there is 2 in a row, 3 in a row etc and if the pieces generate a larger sequence it is given a higher score. Eg 2 in a row only adds 5 points while 3 in a row adds 25 points. The second screenshot also shows that points are also given defensively as preventing the opponent winning and denying their set up for a four in a row is crucial and therefore has high scores for defensive moves.



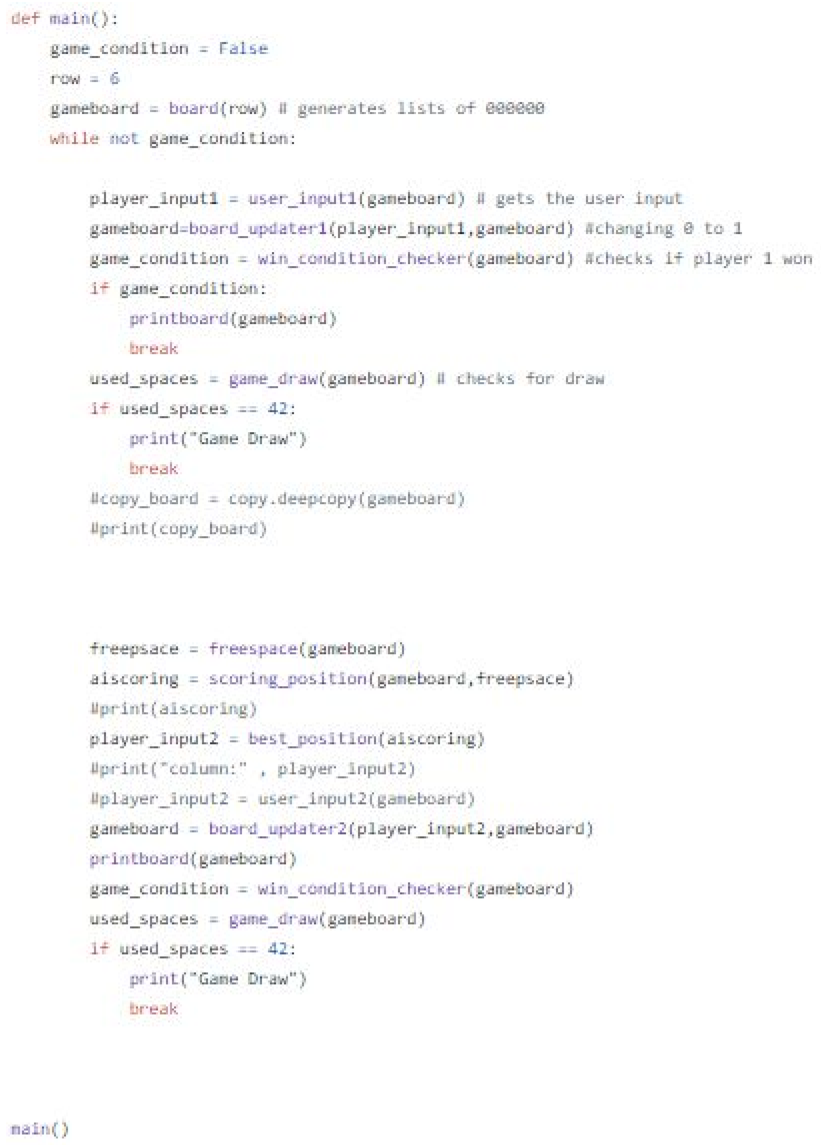
```python
def best_position(horizontal_ai):
    score_list=[]
    for i in horizontal_ai[2:21:3]:
        score_list.append(i)
    score=max(score_list)
    print("score:" , score)
    if score == 0:
        options = [0,1,2,3,4,5,6,]
        return random.choice(options)
    else:
        for a in range(0,7):
            if score_list[a] == score:
                pos_best= a + 1
                break
            else:
                pass

        pos_best = pos_best * 3
        pos_best = pos_best - 2
        return horizontal_ai[pos_best]
```

After getting a score for every possible move for the AI the code above sorts all of the score with its x and y cord. This function makes a list that enters “ x cord, y cord, score” therefore there will be 21 as there are 7 possible moves and have 3 values set to each. It looks at every third value of the list to see which has the best score. Pos_best is used to manipulate the list by using the highest score and returning the value of its x coordinate as that represents its column number. As the board updated only takes in one integer value, of the column the AI will want to insert its piece. Finalising the AI part in the programm outputting the column number with the highest score assigned to it

Function ai_win () and player_win() were functions to help implement minimax.

Above is the code used for the minimax algorithm. Referring back to the pseudocode that was given. This was my attempt to put the right variables in hope of the AI looking into future moves. I had problems implementing some of the variables from the pseudocode as I did not know and understand what some of them referred to my connect four game. As the pseudocode was one that was generally used for most games that allows AI to look into future moves eg like chess. Sadly when tested the programm ran the same as before and did not look into future moves



```
def main():
    game_condition = False
    row = 6
    gameboard = board(row) # generates lists of 000000
    while not game_condition:

        player_input1 = user_input1(gameboard) # gets the user input
        gameboard=board_updater1(player_input1,gameboard) #changing 0 to 1
        game_condition = win_condition_checker(gameboard) #checks if player 1 won
        if game_condition:
            printboard(gameboard)
            break
        used_spaces = game_draw(gameboard) # checks for draw
        if used_spaces == 42:
            print("Game Draw")
            break
        #copy_board = copy.deepcopy(gameboard)
        #print(copy_board)



        freepsace = freespace(gameboard)
        aiscoring = scoring_position(gameboard,freepsace)
        #print(aiscoring)
        player_input2 = best_position(aiscoring)
        #print("column:" , player_input2)
        #player_input2 = user_input2(gameboard)
        gameboard = board_updater2(player_input2,gameboard)
        printboard(gameboard)
        game_condition = win_condition_checker(gameboard)
        used_spaces = game_draw(gameboard)
        if used_spaces == 42:
            print("Game Draw")
            break


main()
```

Main function uses a while loop. The only two conditions is that either player 1 or the AI has won and will then end the game or when its game is at a draw. Functions are put in the main () in a sequence that it repeatedly asks for user inputs and checks if anyone has won the game. Asking for the input of the player then the AI will calculate the best move it can do and the gameboard will print out showing the board after the AI has made a move.

Testing

Things That Need Testing

- Accurate Player Input that will correspond to where the player wants to put its piece
- Board Creation to look as planned
- The Game will return player 1 win when there is a 4 in a row in the gameboard and end.
- Player input to be asked again when column full
- Player input to be asked again when invalid column number
- New Piece input to be accurate
- Ai testing to work according to its game plan

Testing Strategy

For my testing, I will be using Module testing. I find this the most appropriate as my code is broken down to a lot of functions that carry their own purpose. Most functions are only put together in the main function to produce how the game is supposed to function. Therefore individually testing each function will be more efficient and if a problem is encountered can be addressed immediately.

Testing Plan And Evidence

Game Testing

1.

Function	Board () , printboard()
Purpose of test	To test whether the two functions together generate a game board that matches a connect four board
Description of test	Called at the start of the game, a list that consists of 6 lists put together and to be printed out vertically in order.
Expected result	“0000000” “0000000” “0000000” “0000000” “0000000” “0000000”

Actual result	Prints Connect Four Gameboard -- Works

2.

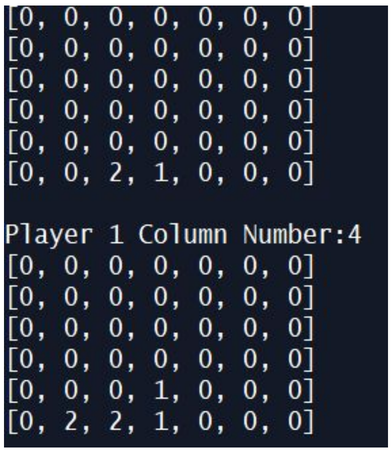
Function	User_inputs1()
Purpose of test	To test to see if the program will ask for a user input until it receives one that is valid
Description of test	Correct Error Outputs and will recursively ask for a user input
Expected result	"Try Again, Numbers Can Only Be 1,2,3,4,5,6,7" "Column" , 4 , "Full Select, Please Select Another Column" "Player 1 Column Number:"
Actual result	User Inputs -- Works

When entered a input of 8 it accurately displays the error message of the number being too big. And repeats until received a number between 1-7

Middle Column full therefore the programme repeats and tells you which column is full -- works

3.

Function	board_updater()
Purpose of test	To test to see if the program will place pieces in the correct spaces
Description of test	When either player inputs a piece it should go on top of an existing pieces unless empty
Expected result	Pieces to go on top of existing pieces
Actual result	Board Updated, pieces goes to correct spaces -- Works



```
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 2, 1, 0, 0, 0]

Player 1 Column Number:4
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0]
[0, 2, 2, 1, 0, 0, 0]
```

As shown when inserting a piece in column 4 it goes above the existing piece instead of replacing it.

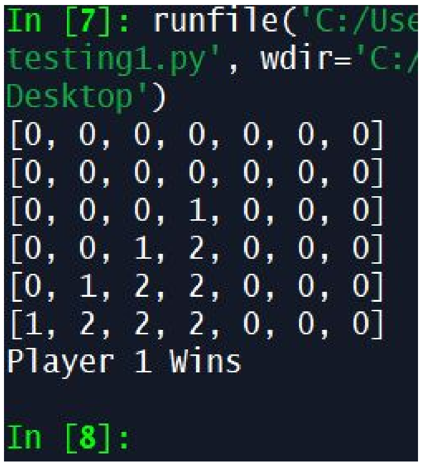
4.

Function	win_condition_checker
Purpose of test	To test to see if the program detects on the gameboard for a 4 in a row
Description of test	To make situations of 4 in a row in each 4 directions to win. Vertical, horizontal and the 2 diagonals.
Expected result	When there is a 4 in a row, the game stops and output player 1 to win.
Actual result	Win function works when the program stops when there is a 4 in a row. -- works

Horizontal :

Vertical :

Diagonals :



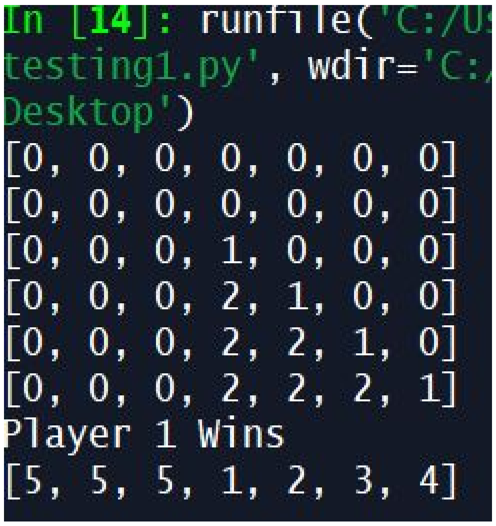
```
In [7]: runfile('C:/Use
testing1.py', wdir='C:/
Desktop')
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 2, 0, 0, 0]
[0, 1, 2, 2, 0, 0, 0]
[1, 2, 2, 2, 0, 0, 0]
Player 1 Wins

In [8]:
```

Pictures above show that in all 4 situations when there is a 4 in a row on the gameboard the program terminates as shown above and returns “ player 1 wins “ therefore the function works.

<u>AI Testing</u>

Function	Freespace ()
Purpose of test	To test to see if the program detects on the gameboard for all the free space for the next move
Description of test	Having pieces already in game board and run function
Expected result	A list of all of the available space to return.
Actual result	List of all the freespace returns-- works



```
In [14]: runfile('C:/U
testing1.py', wdir='C:/
Desktop')
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 2, 1, 0, 0]
[0, 0, 0, 2, 2, 1, 0]
[0, 0, 0, 2, 2, 2, 1]
Player 1 Wins
[5, 5, 5, 1, 2, 3, 4]
```

From this picture the bottom list is the function that produced a list that shows all of the available spaces in each column. As the first list starts from 0 we can see that the list shows an accurate value of where the free spaces are. 5 being the bottom row and 0 being the top row. Therefore the function works.

Function	scoring_position()
Purpose of test	To test to see if the to use the freespace and give a score to each possible column
Description of test	Placing AI pieces around the board manually
Expected result	A list of all of the coordinate then leading by the score for that position
Actual result	List returns accurately the coordinates and the score for each coordinate -- works

Above shows the existing game board followed by a list that was made by the function scoring_position() . The list is structured by [row number , column number , score] the score is in relation to the position. The list shows the first 3 to have a score of 5 as they all generate a 2 in a row. For column 0 and 2 it is a horizontal 2 in a row and column 1 has a vertical 2 in a row.

Another offensive scenario, we see that the best move here is in column 2 as it returns the highest score of 35. As that position gives a 2 in a row both left and right but also creates a 3 in a row as it is between two other AI pieces . Giving a score of 35 as a 3 in a row is 25 points and 5 for each 2 in a row. And is definitely also the best move in this scenario .

Next a defensive scenario.

As we can see even though there is a good move that creates a 3 in a row. The AI always also checks if the board for situations where the player is close to winning. Here we see column 7 having a three in a row already and must be blocked or else player 1 will win. Therefore from the scoring list we see a value of 500 for column 6 row 2. Which is the coordinate to stop the 4 in a row. Showing that the bot will deny and stop you from winning. Making it essential to create a scenario where the player can only win when a 2 way win condition is created.

From This picture we can also see that it is the bot's turn to move. If there is a win condition it will take it over any other decision

Function	Best Position
Purpose of test	To test to see if the function manipulates the list and return which column has the highest score
Description of test	Running the program to identify which column is the best move with the highest score.
Expected result	The column number of the best move .
Actual result	The column number of the best move is returned - works.

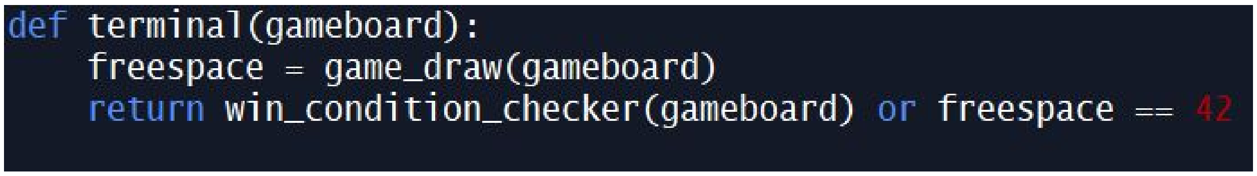
From the picture above we see that the list created shows us that 1060 is the highest score and the coordinate for it is row 5 in column 2. The function we can see working as we see a 2 being returned. The list is manipulated to see where their highest number in the list is and to return the number of the position before it. Therefore we see the column number 2 being returned as it contains the best move for the AI to place its next piece in column 2

<u>Failed Tests</u>

When testing if the Mini Max algorithm was working it was really easy to identify if it was working .The Program was simply still running the same way and was not looking into future moves. This is identified by playing a game with the AI it was making the exact decision from before and when increasing the depth which is supposed to increase the time complexity of the programm it ran at the same speed.

Some changes to the code that I tried to make the mini max algorithm work. The mini max algorithm compares the highest score for the AI pieces and the lowest score for the player.

When trying to define a terminal node. In my game there were only 3 conditions either a Draw or a win by AI or the player. The way I checked if the player would win was from my function win_condition_checker() which you can see is used in my terminal function for mini max .



```
def terminal(gameboard):
    freespace = game_draw(gameboard)
    return win_condition_checker(gameboard) or freespace == 42
```

Currently the function checks for both the player and the AI but in the algorithm the scoring system is treated separately between the AI and the player. The player's best score being as negative as possible and the AI score being as positive as possible. Therefore i copied out and tried separating the functions

The next problem i had was that

The child nodes were assigned to each column to slowly see which move is the best. But as you can see the program will keep traversing until depth reaches 0 which means that using my method of scoring the list of scores created was if depth is at 0. And I did not understand how to use the scores from that existing list and compare against the new list created to minimize the player moves and compare it again to another list that was determined by the minimized list for the best future move.

What the solution might be :

Logically I needed another function to see after the AI has pretended to drop its piece on all 7 columns. Once that is done I need a function to then drop a player piece on all 7 again on all of the previous 7 moves the AI made. Meaning in total there will be a list of 49 scores. But the difference of the new function for the Player is that instead of taking the maximum score it will take the smallest score. And how the scoring would work for the players scoring system will be all of the scores assigned to eg 3 in a row will be negative 25 instead of positive.

Above is a solution that i think might work but due to lack of time i couldn't finish or test if it actually worked out and during the time i was also flustered and could not calmly figure out what was going wrong.

Evaluation

No.	Objective	Performance Criteria	Evaluation
1	Create a game board screen for the player to make his/her decision on where to place the next piece	Simple and clear board that allows to clearly visually empty spaces and differentiate between pieces that are placed from the player and the AI	The 2 functions for creating the board worked well together creating a structure that looks like a connect four game board.
2	A function that asks for the players input to choose where they want to put then next piece	Ensures that the piece is correctly placed on to the gameboard where the user wanted to	The function worked well ensured no piece and be overlapped and taken over and new pieces always appear above the any existing pieces

3	Function that checks if the Player or the AI has one and outputs a message which player has won.	Constantly checks the whole of the game board after each piece input checks for 4 in a row. Vertices, Horizontals and Diagonals	Works perfectly always ending the program when there is a 4 in a row present and outputs a message of who has won the game.
4	Function to check for if the game is drew	Checks the game board to see if the whole board is filled with no spaces left, output game state as “Draw” and ends programm	Works well by identifying how many pieces are on the board and will stop the game and return the game as a draw. No one won.
5	Function that checks if any of the columns are full.	Will check the gameboard to see if the columns are full. Once the column is full the Player and AI are not allowed to choose that column and an error message will be outputted .	Works perfectly , when a column is filled up the user and AI is unable to further add pieces to that column and will choose the 2nd highest number which is how the AI should work.
6	AI prioritises the middle column as this generates the highest probability for winning.	AI when all moves are equal in value will always prioritise the middle.	Works perfectly, plenty of evidence above showing how the bot functions and giving column 4 priority when there is no other clear move.
7	AI piotises the next move to make and attempts to make a two in a row and 3 in a row. Looks into all direction	AI moves will be adjacent or on top of the previous placed to make a two in a row. And 3 in a row instead of 2 when available. Looking into horizontal, vertical and also diagonals .	Works perfectly as the bot always is looking to create links between pieces and once one direction is blocked by an opponent piece it looks in the other 3 directions.
8	AI will place the winning piece as highest priority	When there is a space where creates a 4 in a row it will ignore other options and win the game	Always the bot will place the piece that gives it a four in a row therefore works perfectly
9	AI to know when a column is full	When a column is full the AI will not consider the column to be an option will only look into the other available columns	Works well as when a column is full even if that column gives the highest score that column will be ignored by the AI
10	AI to be able to defend against the player as 2nd highest priority	When the opponent has a three in a row the AI will know to place its next piece	Showed from evidence defending opponent wins is 2nd highest priority. Works

		to prevent the player from winning.	
11	Implement MiniMax Algorithm	Implementing mini max allows the AI to look into moves ahead. This increases the difficulty of the AI .	Does not work. Above already explained what I think needs solving. But overall I attempted to implement but failed for it to work in my programm.
12	Implement Alpha - Beta Pruning	Saves time when running mini max with a big depth. Going through the tree eliminating Nodes that cannot be	Did not attempt as mini max was not working and it is reliant on it to work. Not enough time to fix mini max.

<u>**Feedback on programm**</u>

After asking a few of my friends to play on my programm these are the main points of suggestion and feedback I got from them.

- At times the board can be confusing to look at. Maybe adding a user interface and having the AI peice and Players piece to have a distinct colour can make it easier to play.
- The bot can be quite predictable after playing against it a few times. But despite being predictable it is hard to win against it as it always knows you are one step from winning and will always block it. Notice the pattern of how bot works but still find it hard to beat.
- There is no save function which can be useful so people can continue their game when they have more freetime instead of restarting a game every time
- Overall meets a minimal requirement of having a bot to play against, the bot gets you to think even without looking into the future steps. Definitely feels like playing connect four with an opponent
- Maybe even a highscore system or a counter to see how quickly you can beat the bot and to compare your score with other competitors to see how good you are compared to other players.

<u>**How Can The Outcomes Be Improved**</u>

- To improve the difficulty simply to add Mini Max algorithm and alpha beta pruning to make it harder and the programm to run smoother.
- Adding a graphic interface can certainly help the eye to make quicker decision as the pieces are more distinctively separable
- Adding a ranking system or a highscore table from numerous players can increase drive for playing the game and can help increase motivation to improve.