**AI Assignment 1**

Sharvari Deshmukh

TY CSA 73

**Problem Statement**- Implementation of AI and Non-AI technique by implementing any two player game

**Theory-**

**Non AI Techniques-** Non-AI techniques involve a series of instructions that are followed when a trigger is encountered. An information system that is built using decision-making techniques other than AI that include procedural or declarative knowledge. Examples of techniques that form a basis of procedural knowledge include those focused on algorithmic analytical models, clinical cos-, or decision trees.

**AI techniques-** Artificial Intelligence techniques refer to a set of methods and algorithms used to develop intelligent systems that can perform tasks requiring human-like intelligence. These are models built from sophisticated elements of the computational and mathematical models. Such models allow a computer or machine to measure tasks that are supposed to be performed by humans. E.g., Heuristics, Support Vector Machines, Artificial Neural Networks, Markov Decision Process, Natural Language Processing etc.

**Difference between AI and Non AI-**

Non-AI technique is when the system keeps on following the defined set of rules to reach the solution and AI is when the system learns from its past, overcomes its mistakes and gives more optimal solutions to the problems.

|  |  |
| --- | --- |
| **Non AI** | **AI** |
| Smart machines which are not AI, do not require training data, they work on algorithms only. | AI machines are trained with data and algorithm. |
| Smart machines work on fixed algorithms and they always work with the same level of efficiency, which is programmed into them. | AI machines learn from mistakes and experience. They try to improvise on their next iterations. |
| Machines which are not AI cannot take decisions on their own. | AI machines can analyses the situation and can take decisions accordingly. |
| An automatic door in a shopping mall, seems to be AI-enabled, but it is built with only sensor technology | AI based drones capture the real-time data during the flight, processes it in real-time, and makes a humanin dependent decision based on the processed data. |

**Implementation of Tic Tac Toe using Non AI Technique-**

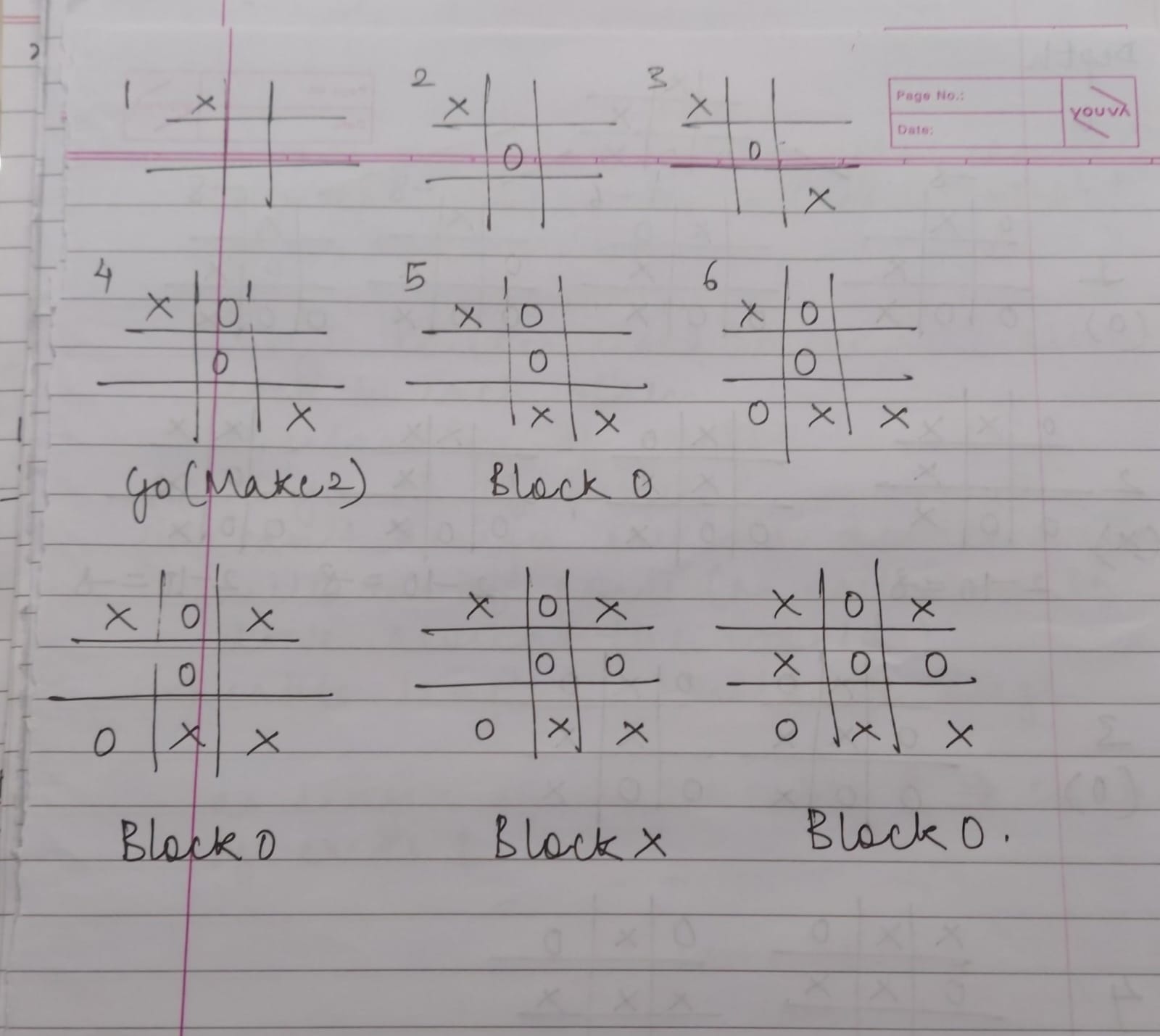
**Algorithm-**

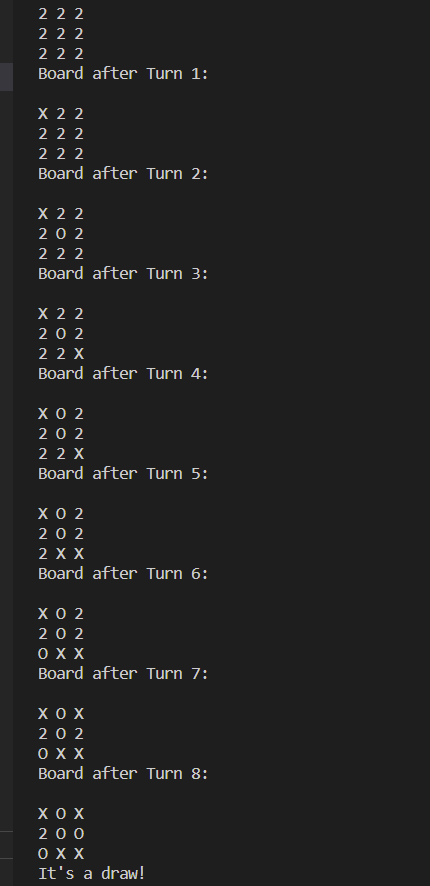
|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

1. The data structure for the board represented using a 1D array, where empty squares are denoted with the number 2, X with 3, and O with 5.
2. The variable TURN tracks the current move number, starting from 1 for the first move and ending at 9 for the last move.
3. MAKE2 function returns the center square (5) if it's blank, otherwise returns any blank non-corner square (2, 4, 6, or 8).
4. POSSWIN(p) function determines if player p can win on the next move by checking rows, columns, and diagonals. It returns the square number that leads to a winning move or 0 if there's no potential win.
5. GO(n) function makes a move to square n, setting it to 3 for X if TURN is odd, or 5 for O if TURN is even. It also increments TURN by 1.
6. The algorithm has specific strategies for each turn:

* For TURN 1, it always chooses the upper left corner (square 1).
* For TURN 2, it chooses square 5 if it's empty, otherwise square 1.
* For TURN 3, it chooses square 9 if it's empty, otherwise square 3.
* For TURN 4, it blocks opponent's win if possible, otherwise uses MAKE2.
* For TURN 5, it aims to win if possible, otherwise blocks opponent's win or creates a fork if possible.
* For TURN 6, it blocks opponent's win if possible, otherwise aims to win or uses MAKE2.
* For TURN 7, it aims to win if possible, otherwise blocks opponent's win, or goes to any empty square.
* For TURN 8, it blocks opponent's win if possible, otherwise aims to win or goes to any empty square.
* TURN 9 follows the same strategy as TURN 7.

The algorithm's approach requires thorough condition checking for each move, resulting in longer computation time but efficient storage usage. However, it's tailored specifically for Tic-Tac-Toe and isn't easily generalized to other games like 3-D Tic-Tac-Toe.





**Implementation of Tic Tac Toe using AI Technique-**

**Heuristic-** MiniMax Algorithm-

Minimax algorithm is a decision-making algorithm and what makes it so powerful as its ability to think ahead. Minimax is a recursive algorithm meaning that it doesn't just look at the current state of the game, it anticipates feature moves and imagines the entire sequence of moves that follow. This recursive nature allows minimax to consider all possible moves and outcomes creating a branching tree of possibilities called the game tree. Minimax's ultimate goal is to find the optimal move for a player maximizing their score while minimizing the opponent's score.

Key components of the minimax algorithm-

1. Evaluation function - responsible for assigning a score to each possible board state. This score is the algorithm's way of quantifying how favorable or unfavorable a given state is. A positive score implies an advantage for the maximizing player and the algorithm sees potential victory. Conversely a negative score signals an advantage for the opponent, a path that might lead to victory for the opponent. A neutral score of 0 is given when neither player wins. The evaluation function acts as a guide for the algorithm to choose the most favorable outcome for the player it represents.

2. Maximizing and minimizing player

A description for the algorithm, assuming X is the "turn taking player," would look something like:

* If the game is over, return the score from X's perspective.
* Otherwise get a list of new game states for every possible move
* Create a scores list
* For each of these states add the minimax result of that state to the scores list
* If it's X's turn, return the maximum score from the scores list
* If it's O's turn, return the minimum score from the scores list

function minimax(board, depth, isMaximizingPlayer):

if current board state is a terminal state :

return value of the board

if isMaximizingPlayer :

bestVal = -INFINITY

for each move in board :

value = minimax(board, depth+1, false)

bestVal = max( bestVal, value)

return bestVal

else :

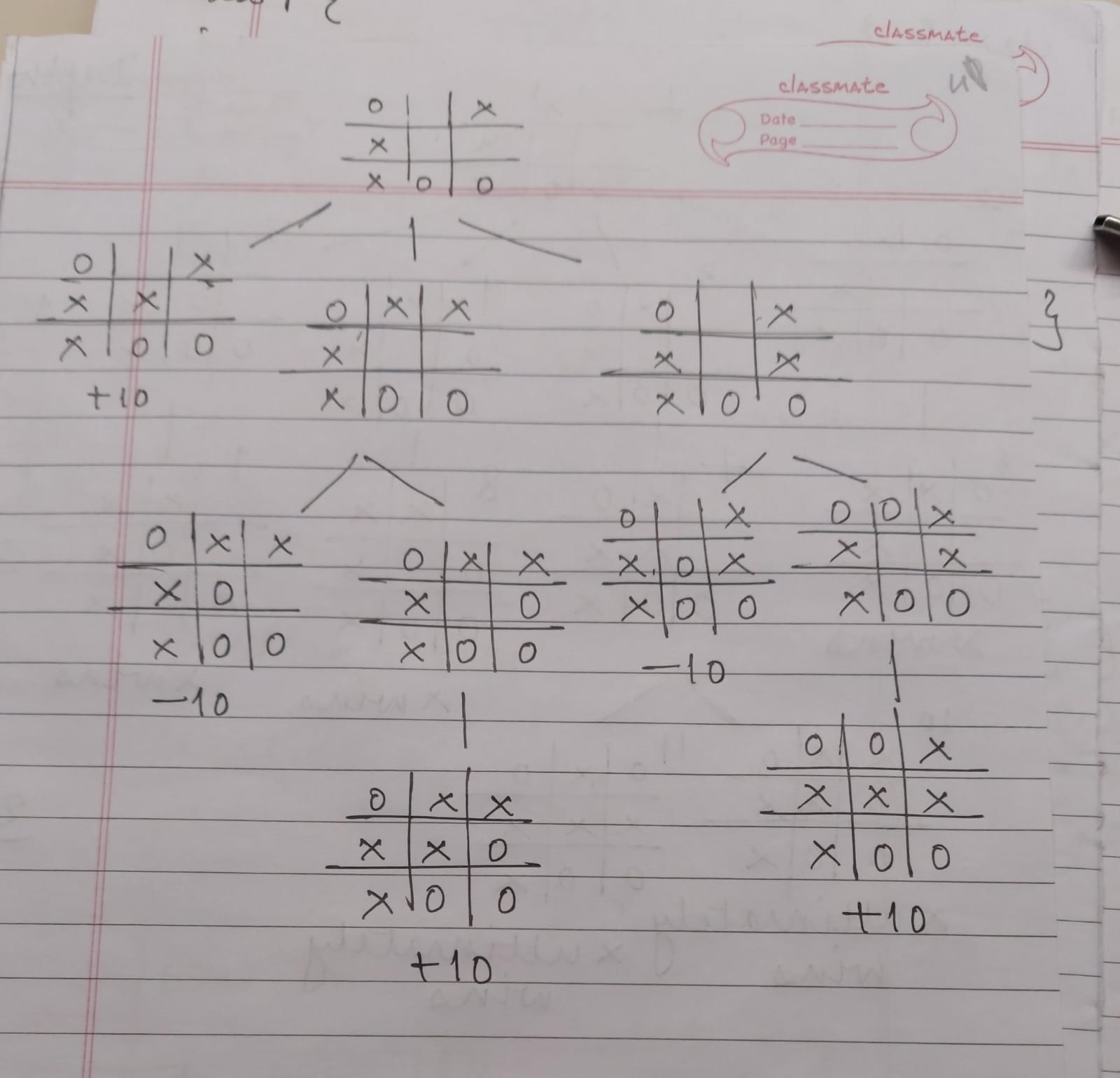
bestVal = +INFINITY

for each move in board :

value = minimax(board, depth+1, true)

bestVal = min( bestVal, value)

return bestVal



**Structure of the algorithm-**

1. Structures and Constants:

- Define structure `Move` to represent a move on the tic-tac-toe board, with `row` and `col` attributes

- Define constants `human` and `computer` to represent the players.

2. Functions:

- isMovesLeft: Determine if there are any remaining moves on the board by iterating through each cell.

- evaluate: Evaluate the current state of the board to determine if there is a winner or if the game is still ongoing.

- minimax: Implement the minimax algorithm to recursively search for the best move for the computer player. This function alternates between maximizing the score for the computer player and minimizing the score for the human player.

- findBestMove: Utilize the minimax function to find the best move for the computer player among the available empty cells on the board.

- printBoard: Display the current state of the tic-tac-toe board.

3. Main Function:

- Initialize the tic-tac-toe board.

- Print the initial state of the board.

- Enter a loop to alternate between the human player and the computer player until there are no more moves left.

- If it's the computer player's turn, calculate and make the best move using the minimax algorithm (here, computer is minimizing player)

- If it's the human player's turn, prompt the user to enter their move and validate it (here, human is the maximizing player)

- Continue this loop until the game ends.

4. Game Flow:

- The game alternates between the human player and the computer player.

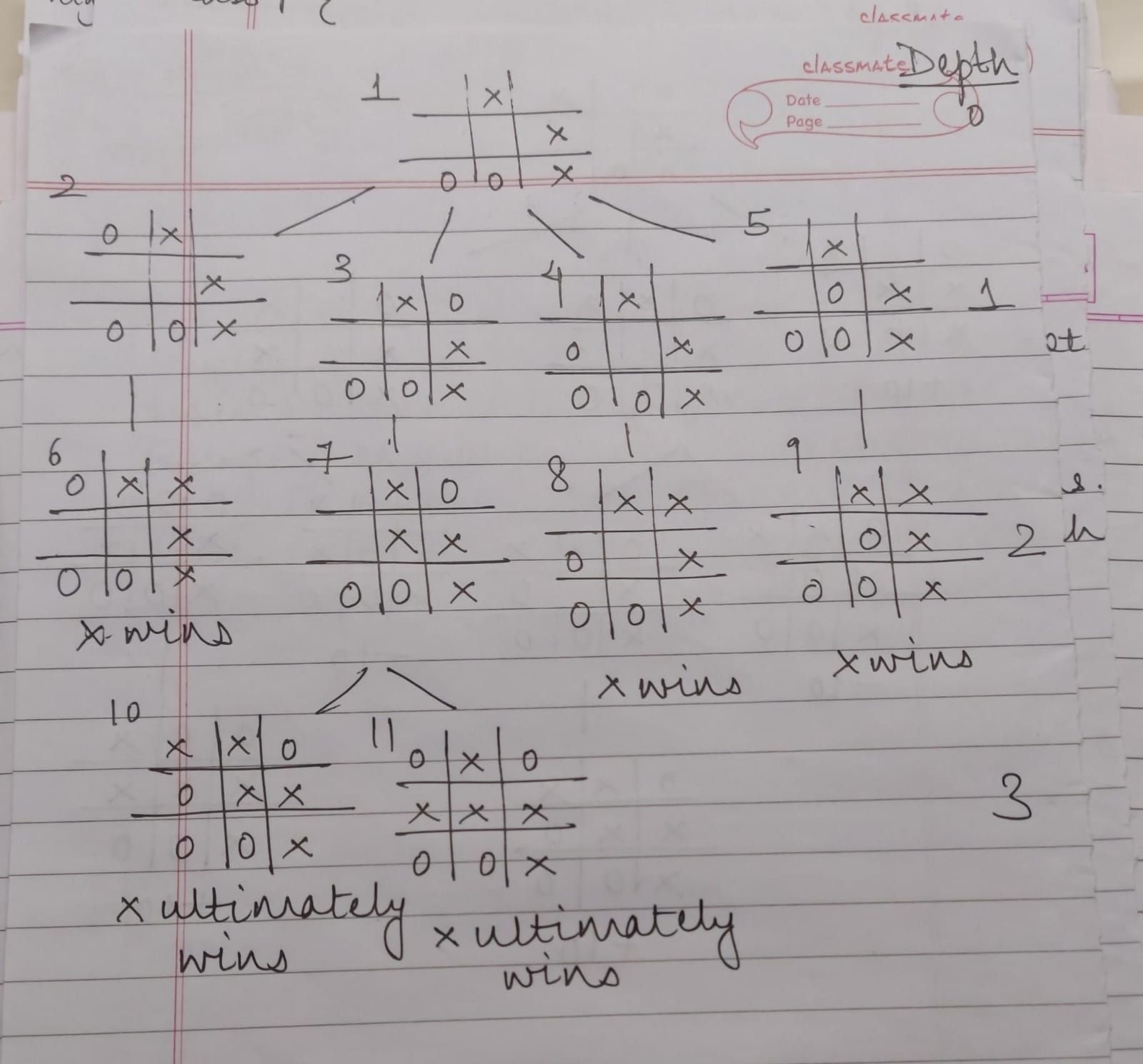
- The computer player uses the minimax algorithm to make intelligent moves.

- The human player enters their moves via standard input.

- The game state is continuously checked for a winner or a draw.

- The game ends when there are no more moves left or when a player wins.

**Importance of depth/turns in this algorithm-**



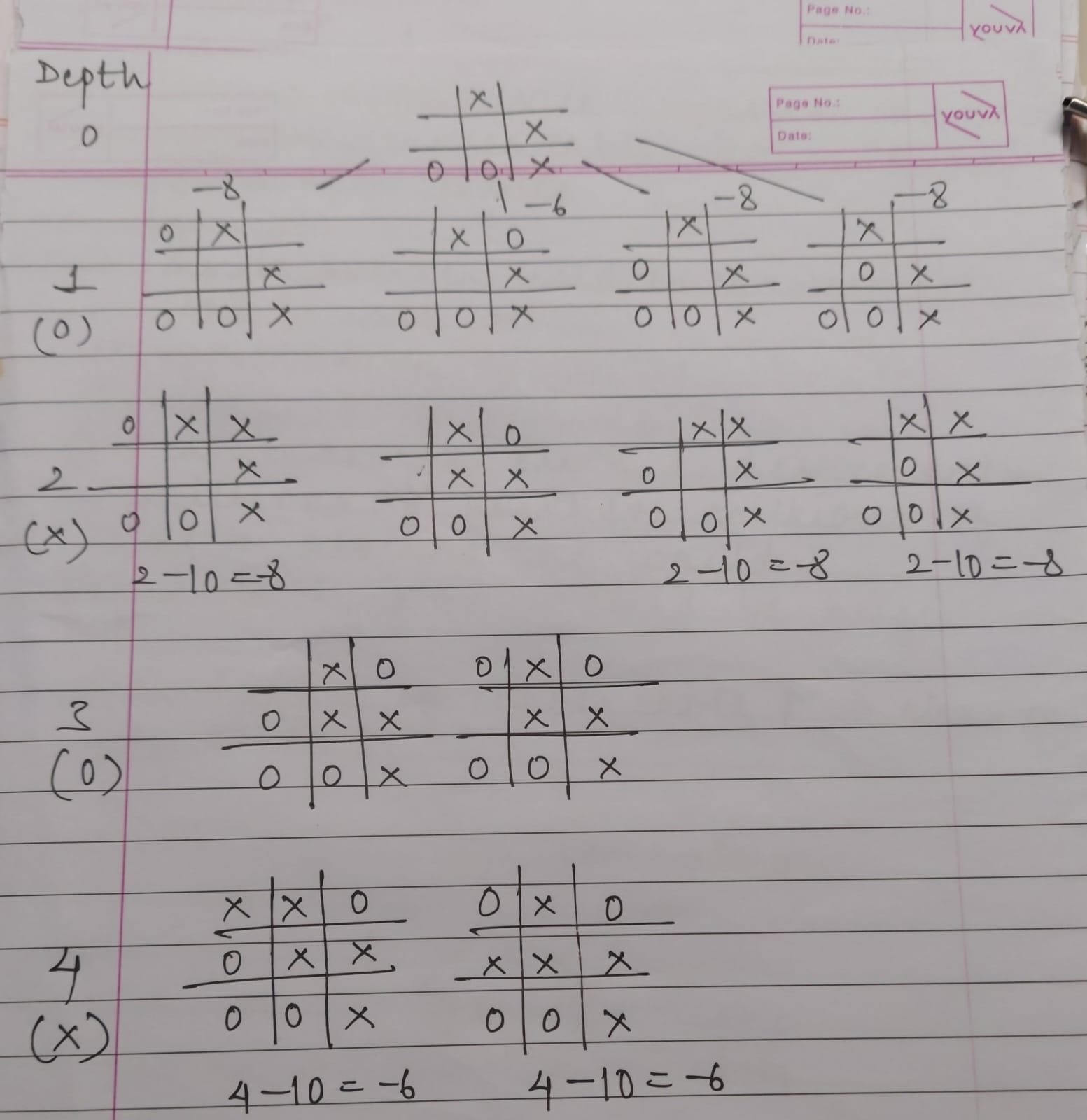
A perfect player is expected to at least put up a fight and block opponent’s immediate win

Given the board state 1 where both players are playing perfectly, and O is the computer player. O choses the move in state 5 and then immediately loses when X wins in state 9.

But if O blocks X's win as in state 3, X will obviously block O's potential win as shown in state 7. This puts two certain wins for X as shown in state 10 and 11, so no matter which move O picks in state 7, X will ultimately win.

As a result of these scenarios, and the fact that we are iterating through each blank space, from left to right, top to bottom, all moves being equal, that is, resulting in a lose for O, the last move will be chosen as shown in state 5, as it is the last of the available moves in state 1. The array of moves being: [top-left, top-right, middle-left, middle-center].

No matter the board arrangement, the perfect player will play perfectly unto its demise, is to take the "depth" or number of turns till the end of the game into account. Basically the perfect player should play perfectly, but prolong the game as much as possible. To achieve that, the depth is subtracted from the score, that is the number of turns, or recursions, from the end game score, the more turns the lower the score, the fewer turns the higher the score.



Depth causes the score to differ for each end state, and because the level 0 part of minimax will try to maximize the available scores (because O is the turn taking player), the -6 score will be chosen as it is greater than the other states with a score of -8. And so even faced with certain death, our trusty, perfect player now will chose the blocking move, rather than commit honor death.

