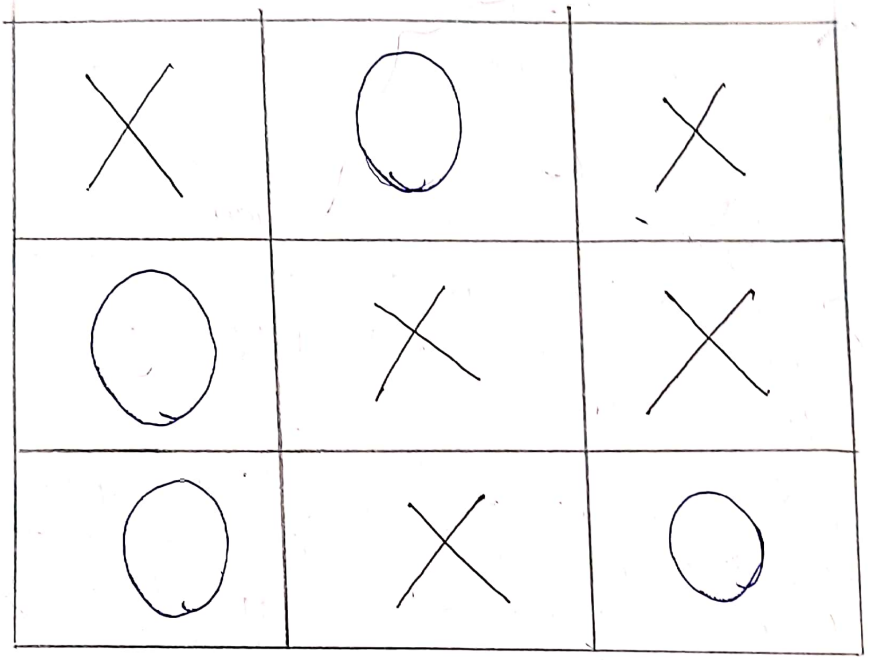
III / I Semester

Subject : AI

Reg no : 23B01A05523 , 23B01A05539

St. name : D. Navya
T. Hemanjali



Part A : understanding TIC-TAC-TOE

1) It is a simple two player zero sum game played on 3×3 grid where player take turns making a cell with 'X' & 'O'

winning conditions are

Either of the player to place 3 of their marks in a row & a column. Either horizontally, Vertically & diagonally.

2, In this particular game first turn of the player with mark "X".
There the possible moves for the player with first turn are 5 moves.

3, Importance of AI In TIC-TAC-TOE
It helps avoid making moves that could lead to a loss in future turns.
It helps in evaluating the concequences of each move, choosing the best one to acheive victory & avoid defeat.

Part B: AI concepts

4, Strategy to never loose the game.
mini-max algorithm is used as a strategy with AI which ensures the player always plays optimally.
Strategy:
→ simulates all possible moves for both players.
→ chooses the move that maximises its chances of wining and minimises the opponent chances.
This strategy quarentees an outcome.

5, mini-max Algorithm
It is a recursive strategy used in decision-making and game theory to determine the optimal move for a player, assuming that the opponent also plays optimally.
In this player B select the strategy that minimises the player A more gains.

In this algorithm - worst guarentee of loss for player B.

Eg, The AI will simulate all moves and pick the one with the highest guaranteed value, assuming the human also plays optimally.

6, Can AI make mistakes?

Of correctly implemented (Algorithm) AI will not make mistakes because the game has finite number of possible moves and can be completely solve using mini-max algorithm.

Mistakes can occur if:

- => Algorithm has errors
- => Evaluation functions are flawed
- => AI has depth limitations (for larger game)

Part C : coding and debugging

7, Inputs to the program

- => Current state of the board (2D array & list)
- => The player symbol it is playing as ('X' or 'O')
- => who's turn it is

8, outputs AI produce

- => coordinates & Index of the best next move
- => Outputs the updated board state after the move is made.

9, Bug encounter - fixing it

=> Bug : AI sometimes pick's an already occupied cell

Cause : Algorithm did not properly check whether the cell was empty or not before selecting.

=> fix :

If board[row][col] == ' ' :

added a condition to check for only empty cells while generating possible moves.

Part D: Heuristic Evaluation

10, A Heuristic evaluation function is an intelligent guess about how good or bad a game position is helping AI make smarter, faster decisions.

Heuristic Score Rules:

=> +10 X wins

=> +1 X has two in a row with one empty

=> -1 O has two in a row with one empty

=> -10 O wins

11,

Python Code :

```
def check_win(board, player):
    win_lines = [([board[0][0], board[1][0], board[2][0]], "column 1"),
                 ([board[0][1], board[1][1], board[2][1]], "column 2"),
                 ([board[0][2], board[1][2], board[2][2]], "column 3"),
                 ([board[0][0], board[0][1], board[0][2]], "Row 1"),
                 ([board[1][0], board[1][1], board[1][2]], "Row 2"),
                 ([board[2][0], board[2][1], board[2][2]], "Row 3"),
                 ([board[0][0], board[1][1], board[2][2]], "Main Diagonal"),
                 ([board[0][2], board[1][1], board[2][0]], "Anti Diagonal"),
                 ]
    for line, name in win_lines:
        if line == [player, player, player]:
            return True, name
    return False, None.
```

```
def is_draw(board):
    return all(cell != '' for row in board for cell in row)

def check_final_result(board):
    x_wins, x_line = check_win(board, 'X')
    o_wins, o_line = check_win(board, 'O')

    if x_wins:
        print("X wins by:", x_line)
    elif o_wins:
        print("O wins by:", o_line)
    elif is_draw(board):
        print("Result: Draw")
    else:
        print("Game is still ongoing.")

    print("Final Board:")
    for row in board:
        print(row)

final_board = [
        ['X', 'O', 'X'],
        ['O', 'X', 'X'],
        ['O', 'X', 'O']]

check_final_result(final_board)
```

Output:

```
Result: Draw
Final Board:
['X', 'O', 'X']
['O', 'X', 'X']
['O', 'X', 'O']
```

Sample Questions:

- How does the AI decide which move to make

It simulates all possible future moves, evaluate outcome and choose the move that maximises the chance of winning and minimises the opponent chances.

- What makes Tic-Toc-Toe a good Example to learn about AI?

It's simple with a small state space. Easy to visualize and understand. Demonstrates core AI concepts like: Game trees, Decision making, Recursion.

- Can the AI always guarantee a win? why & why not?

=> No, AI cannot always guarantee a win.
but it can always avoid a loss.
AI can only win if the opponent makes a mistake

- How would the AI change if the board was bigger?

-> A bigger board increases the complexity:

- more possible moves - larger game tree.
- minimax will take longer due to more combinations.

- What other games could you apply this AI technique to?

=> minimax and similar AI techniques can be applied to: Connect Four, checkers, Chess, Gomoku