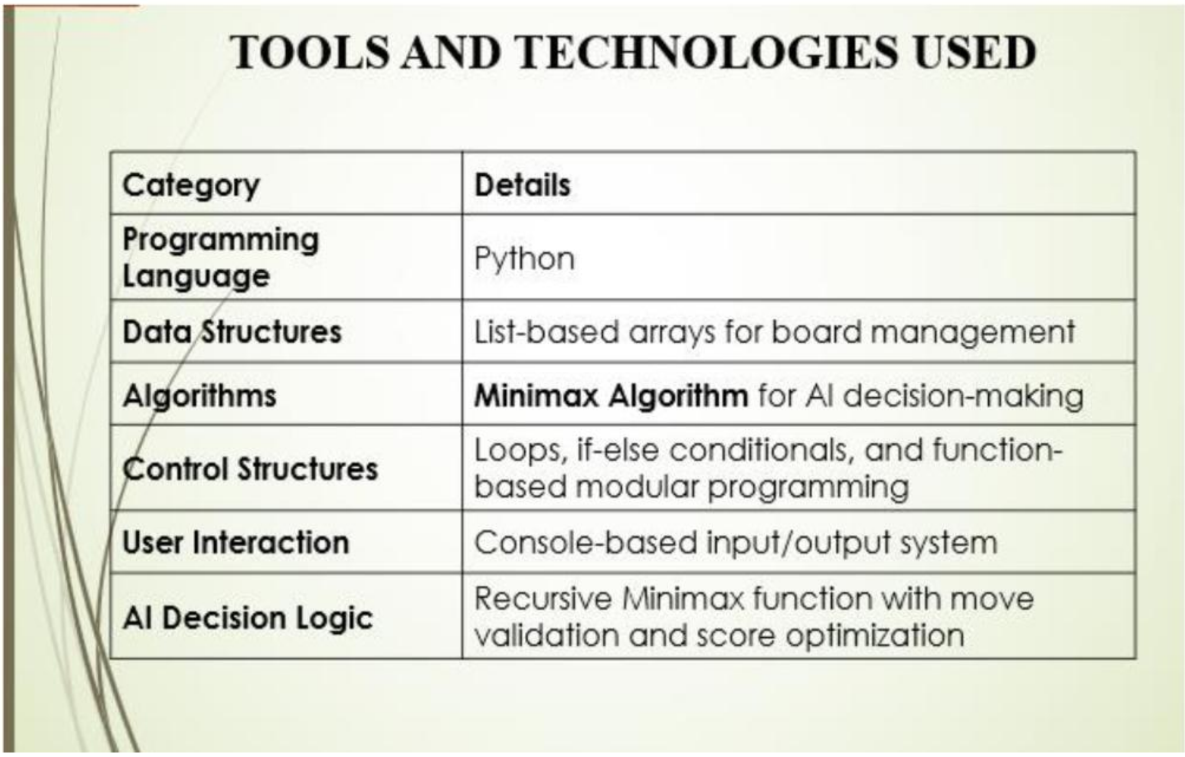
TASK-2

Tic-Tac -Toe AI Game



TOOLS AND TECHNOLOGIES USED

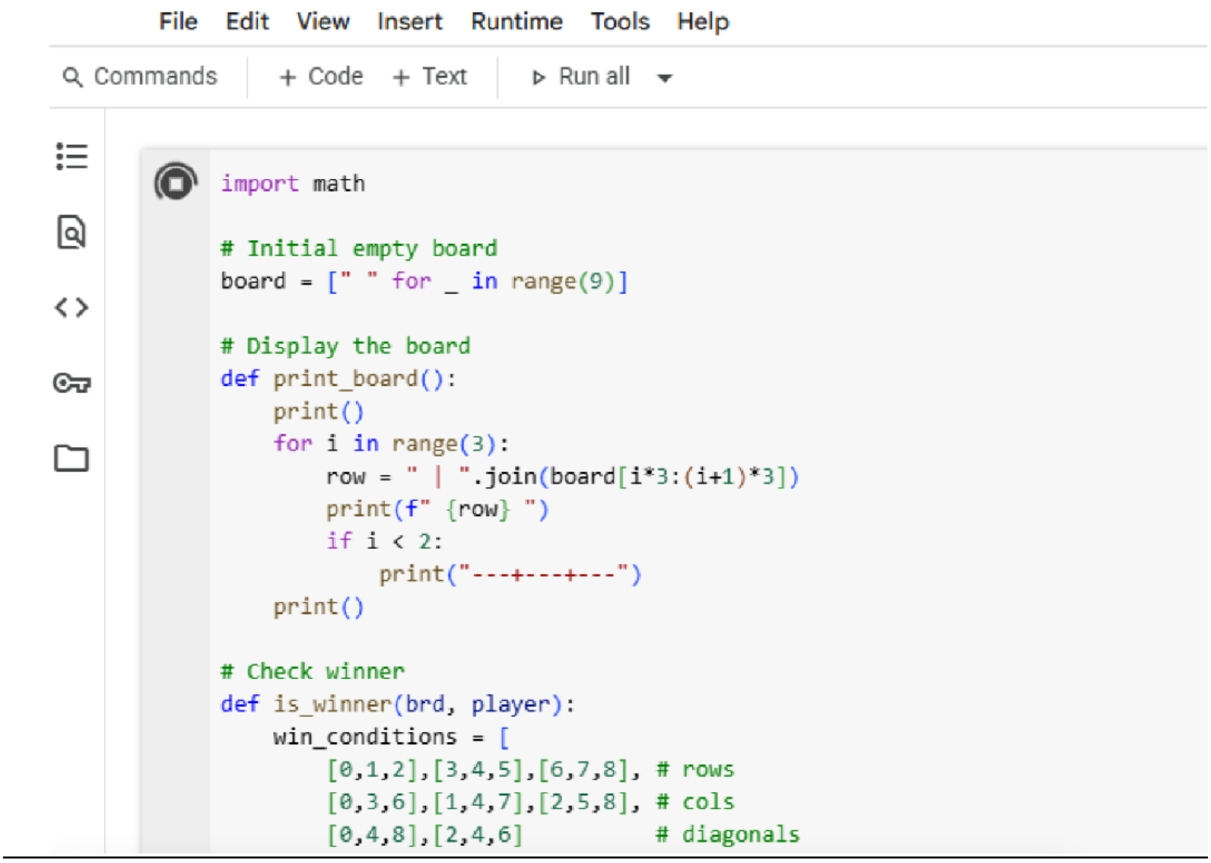
Category	Details
Programming Language	Python
Data Structures	List-based arrays for board management
Algorithms	**Minimax Algorithm** for AI decision-making
Control Structures	Loops, if-else conditionals, and function-based modular programming
User Interaction	Console-based input/output system
AI Decision Logic	Recursive Minimax function with move validation and score optimization

CODE :

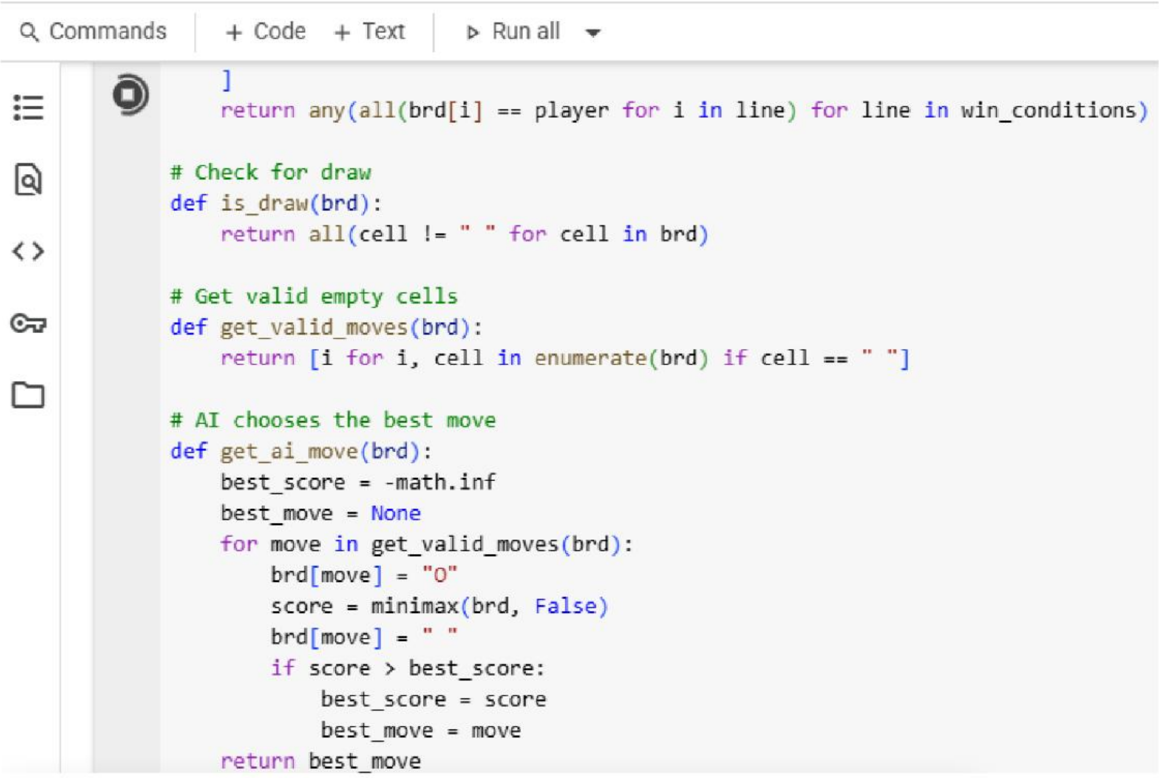
Output

```
# Tic-Tac-Toe with Minimax AI


def print board(board):
    for row in board:
        print("|".join(row))
        print("-" * 5)


def check winner(board):
    # Check rows, columns, and diagonals
    lines = board + [list(col) for col in zip(*board)]
    lines. append([board[i][i] for i in range(3)])
    lines. append([board[i][2 - i] for i in range(3)])

    for line in lines:
        if line == ["X"] * 3:
            return "X"
        if line == ["O"] * 3:
            return "O"
    return None

def is full(board):
    return all(cell != " " for row in board for cell in row)

def minimax(board, is maximizing):
    winner = check winner(board)
    if winner == "X":
        return 1
    elif winner == "O":
        return -1
    elif is full(board):
        return 0

    if is maximizing:
        best score = -float("inf")
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    score = minimax(board, False)
                    board[i][j] = " "
                    best score = max(score, best score)
        return best score
    else:
        best score = float("inf")
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    score = minimax(board, True)
                    board[i][j] = " "
                    best score = min(score, best score)
        return best score

def best move(board):
    best score = -float("inf")
    move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "X"
                score = minimax(board, False)
                board[i][j] = " "
                if score > best score:
                    best score = score
                    move = (i, j)
    return move


def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic-Tac-Toe!")
    print("You are 'O', AI is 'X'. Enter moves as row and column (e.g., '1 2')\n")
    print board(board)


    while True:
        # Player move
        try:
            row, col = map(int, input("Enter your move (row and column from 0 to 2): ").split())
            if not (0 <= row <= 2 and 0 <= col <= 2):
                print("Invalid input. Please enter values between 0 and 2.")
                continue
            if board[row][col] != " ":
                print("That spot is already taken. Try again.")
                continue
        except Value Error:
            print("Invalid input format. Enter two numbers separated by space.")
            continue
        board[row][col] = "O"
        print("\n Your move:")
        print board(board)

        if check winner(board) or is full(board):
            break

        # AI move
        Ai row, ai col = best move(board)
        board[ai row][ai col] = "X"
        print("\n AI's move:")
        print board(board)

        if check winner(board) or is full(board):
            break

    winner = check winner(board)
    print("\n Game Over!")
    if winner:
        print(f"{winner} wins!")
    else:
        print("It's a draw!")

if _name_ == "_main_":
    main()
```

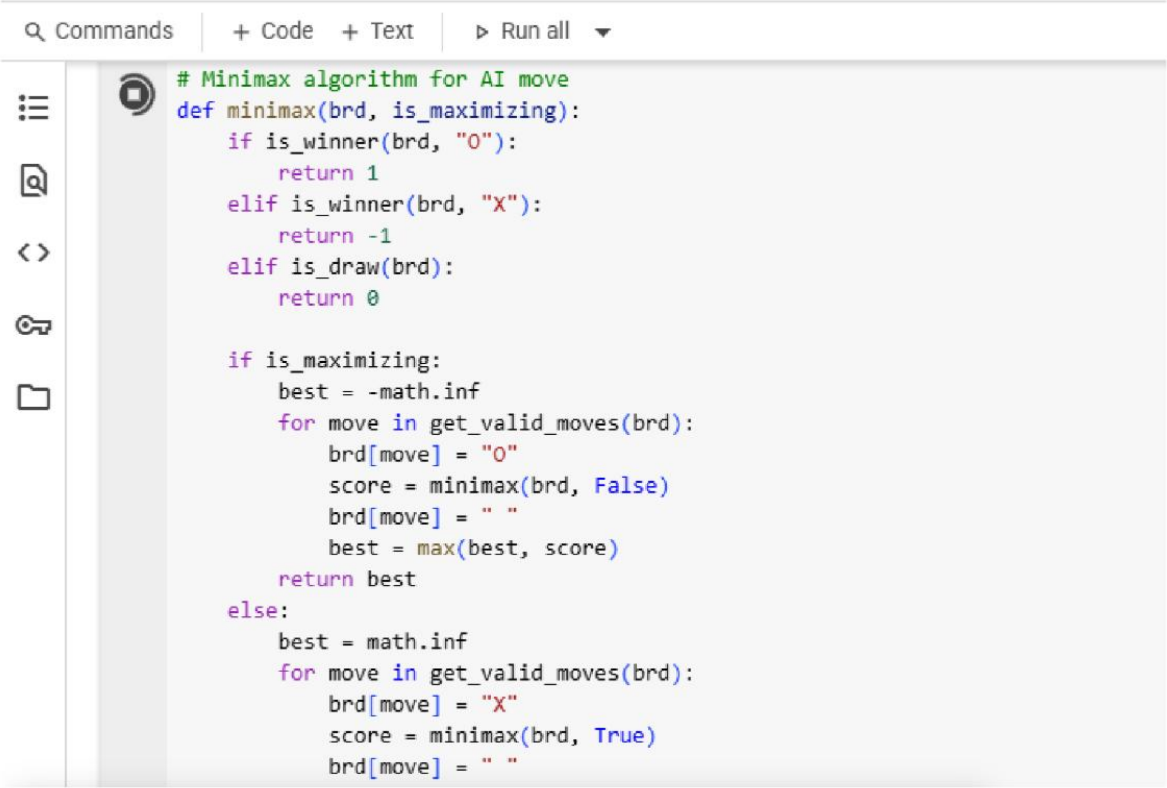
Input:



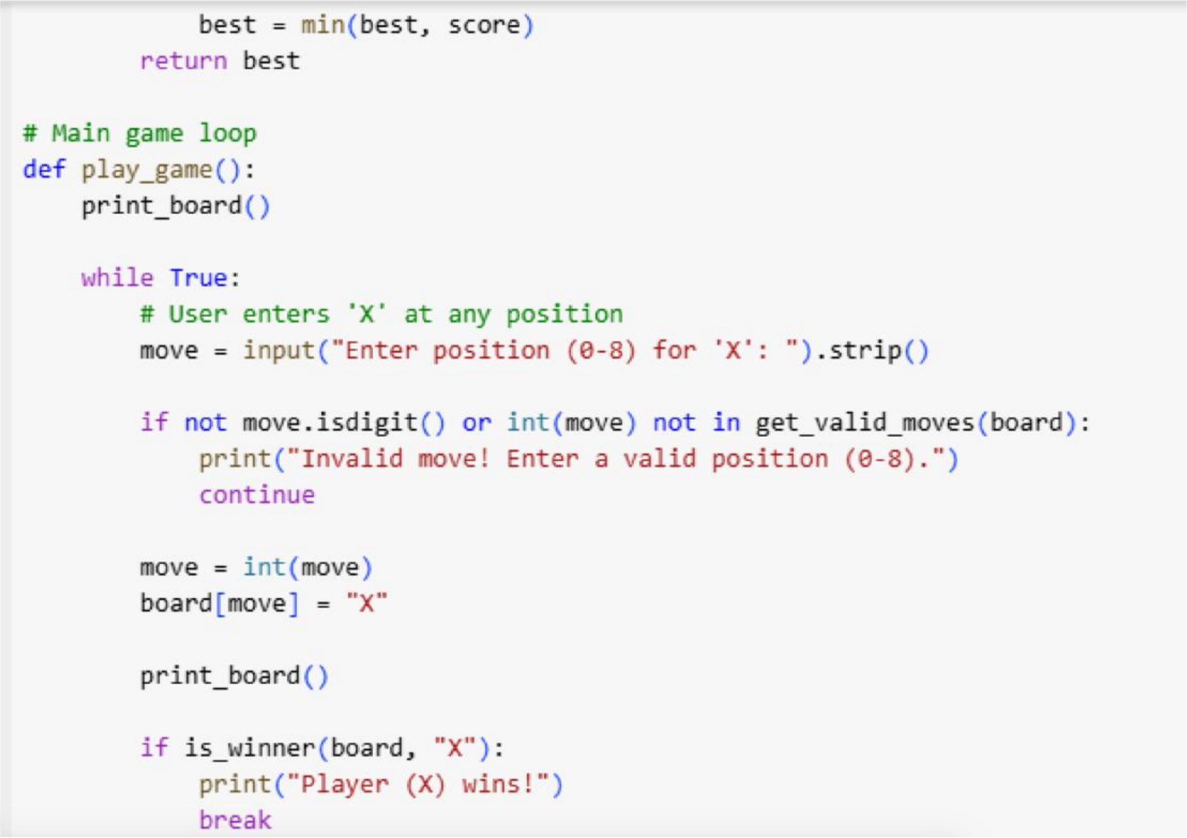
```
import math

# Initial empty board
board = [" " for _ in range(9)]

# Display the board
def print_board():
    print()
    for i in range(3):
        row = " | ".join(board[i*3:(i+1)*3])
        print(f" {row} ")
        if i < 2:
            print("---+---+---")
    print()

# Check winner
def is_winner(brd, player):
    win_conditions = [
        [0,1,2],[3,4,5],[6,7,8], # rows
        [0,3,6],[1,4,7],[2,5,8], # cols
        [0,4,8],[2,4,6]          # diagonals
```



```
    ]
    return any(all(brd[i] == player for i in line) for line in win_conditions)

# Check for draw
def is_draw(brd):
    return all(cell != " " for cell in brd)

# Get valid empty cells
def get_valid_moves(brd):
    return [i for i, cell in enumerate(brd) if cell == " "]

# AI chooses the best move
def get_ai_move(brd):
    best_score = -math.inf
    best_move = None
    for move in get_valid_moves(brd):
        brd[move] = "O"
        score = minimax(brd, False)
        brd[move] = " "
        if score > best_score:
            best_score = score
            best_move = move
    return best_move
```



```
# Minimax algorithm for AI move
def minimax(brd, is_maximizing):
    if is_winner(brd, "O"):
        return 1
    elif is_winner(brd, "X"):
        return -1
    elif is_draw(brd):
        return 0

    if is_maximizing:
        best = -math.inf
        for move in get_valid_moves(brd):
            brd[move] = "O"
            score = minimax(brd, False)
            brd[move] = " "
            best = max(best, score)
        return best
    else:
        best = math.inf
        for move in get_valid_moves(brd):
            brd[move] = "X"
            score = minimax(brd, True)
            brd[move] = " "
```



```
            best = min(best, score)
        return best

# Main game loop
def play_game():
    print_board()

    while True:
        # User enters 'X' at any position
        move = input("Enter position (0-8) for 'X': ").strip()

        if not move.isdigit() or int(move) not in get_valid_moves(board):
            print("Invalid move! Enter a valid position (0-8).")
            continue

        move = int(move)
        board[move] = "X"

        print_board()

        if is_winner(board, "X"):
            print("Player (X) wins!")
            break
```

Code explanation:

1. **Print board(board)**

python

Copy Edit

```
def print board(board):
    for row in board:
        print("|".join(row))
        print("-" * 5)
```

- Displays the 3x3 board in a human-readable format.
- Each row is joined by | symbols.
- A line (-----) is printed after each row for separation.

2. check winner(board)

python

Copy Edit

```
def check winner(board):
    lines = board + [list(col) for col in zip(*board)]
    lines. Append  ([board[i][i] for i in range(3)])
    lines. Append ([board[i][2 - i] for i in range(3)])

    for line in lines:
        if line == ["X"] * 3:
            return "X"
        if line == ["O"] * 3:
            return "O"
    return None
```

- Checks all rows, columns, and diagonals for 3 same marks.
- If "X" or "O" has a full line, it returns the winner.
- Returns None if no winner yet.

3. Is full(board)

python

Copy Edit

```
def is full(board):
    return all(cell != " " for row in board for cell in row)
```

- Returns True if the board is full (no empty cells).
- Useful to check for a draw.

4. minimax(board, is maximizing)

This is the core AI algorithm. It simulates all possible moves to choose the best one.

python

Copy Edit

```
def minimax(board, is maximizing):
    winner = check winner(board)
    if winner == "X":
        return 1
    elif winner == "O":
        return -1
    elif is full(board):
        return 0
```

- Base cases:
  - If AI wins ("X"), return +1.
  - If player wins ("O"), return -1.
  - If draw, return 0.

Then the recursion begins:

- AI's turn (Maximizing Player):

python

Copy Edit

```
if is maximizing:
    best score = -float("inf")
    for all empty spots:
        simulate placing "X"
        recursively call minimax for opponent
        undo move
        choose max score
```

- Player's turn (Minimizing Player):

python

Copy Edit

```
else:
    best score = float("inf")
    for all empty spots:
        simulate placing "O"
        recursively call minimax for AI
        undo move
        choose min score
```

This ensures that the AI always picks the move that leads to a win or draw (never a loss).

5. best move(board)

python

Copy Edit

```
def best move(board):
    best score = -float("inf")
    move = None
    for all empty cells:
        simulate "X" move
        get score from minimax
        undo move
        choose move with highest score
    return move
```

- Iterates over the board to find the move that gives the highest minimax score.
- Returns the best move for the AI.

6. main () Function

Handles the game loop and interactions:

python

Copy Edit

```
def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    ...
```

- Initializes an empty 3x3 board.
- Asks the user to input moves as row and column.
- After each move:
    - Displays the board.
    - Checks for win/draw.
    - Lets the AI make its best move.

7. Ending the Game

python

Copy Edit

```
if check winner(board) or is full(board):
    break
```

- After each turn, it checks if someone won or the board is full.
- Declares the winner or a draw.

8. Typo in Execution Check

python

Copy Edit

```
if _name_ == "_main_":
    main()
```

❌ This line has a mistake.

✅ It should be:

python

Copy Edit

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

This ensures the main () function runs only when the script is executed directly, not when imported.