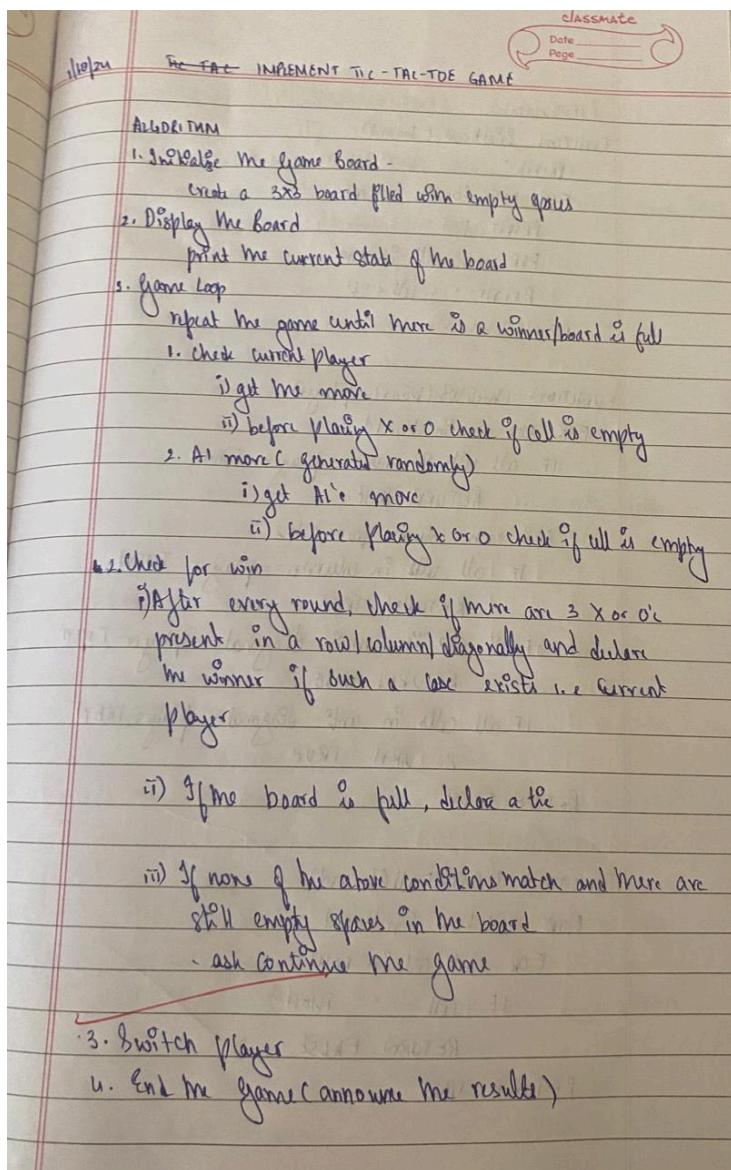


Date: 1/10/24

Program Title: Implement Tic Tac Toe Game

Algorithm -



```

PSEUDOCODE
FUNCTION PrintBoard (board)
    PRINT "-----"
    FOR each row in board
        PRINT "|"
        FOR each cell in row
            PRINT " " + cell + " "
        PRINT "|-----"
    ENDFOR
ENDFUNCTION

FUNCTION CheckWin(board, player)
    FOR each row in board
        IF all cells in row == player THEN
            RETURN TRUE
        FOR each column from 0 to 2
            IF all cells in column == player THEN
                RETURN TRUE
            IF all cells in main diagonal == player THEN
                RETURN TRUE
            IF all cells in anti-diagonal == player THEN
                RETURN TRUE
        RETURN FALSE
    ENDFUNCTION

```

```

FUNCTION IsBoardFull(board)
    FOR each row in board
        FOR each cell in row
            IF cell == "" THEN
                RETURN FALSE
        RETURN TRUE
    ENDFUNCTION

```

classmate
Date _____
Page _____

```

FUNCTION GetPlayerMove(board)
WHILE TRUE
    PROMPT USER FOR ROW AND COLUMN
    IF ROW AND COLUMN ARE VALID AND CELL = EMPTY THEN
        RETURN (ROW, COLUMN)

FUNCTION GetAIMove(board)
    emptyCells = []
    FOR EACH CELL IN BOARD
        IF CELL IS EMPTY THEN
            ADD CELL TO emptyCells
    IF emptyCells IS NOT EMPTY THEN
        RETURN RANDOM CELL FROM emptyCells
    RETURN None

FUNCTION Main()
    board = Initialize A 3x3 Board With Empty Cells
    currentPlayer = "X"
    WHILE TRUE
        CALL PrintBoard(board)
        IF currentPlayer IS "X" THEN
            (row, col) = CALL GetPlayerMove(board)
        ELSE
            (row, col) = CALL GetAIMove(board)
        SET board[row][col] = currentPlayer
        IF CheckWin(board, currentPlayer) THEN
            CALL PrintBoard(board)
            RETURN currentPlayer + " Wins!"
        ELSE IF IsBoardFull(board) THEN
    
```

Data
Page

```

CALL PrintBoard (board)
RETURN current-player + "wins!"
ELSE IF IsBoardFull (board) THEN
    CALL PrintBoard (board)
    RETURN "It's a tie!"
SWITCH current-player
CALL Main()

```

OUTPUT

Enter row (1-3):1

Enter column (1-3):1

X		

A1's turn--

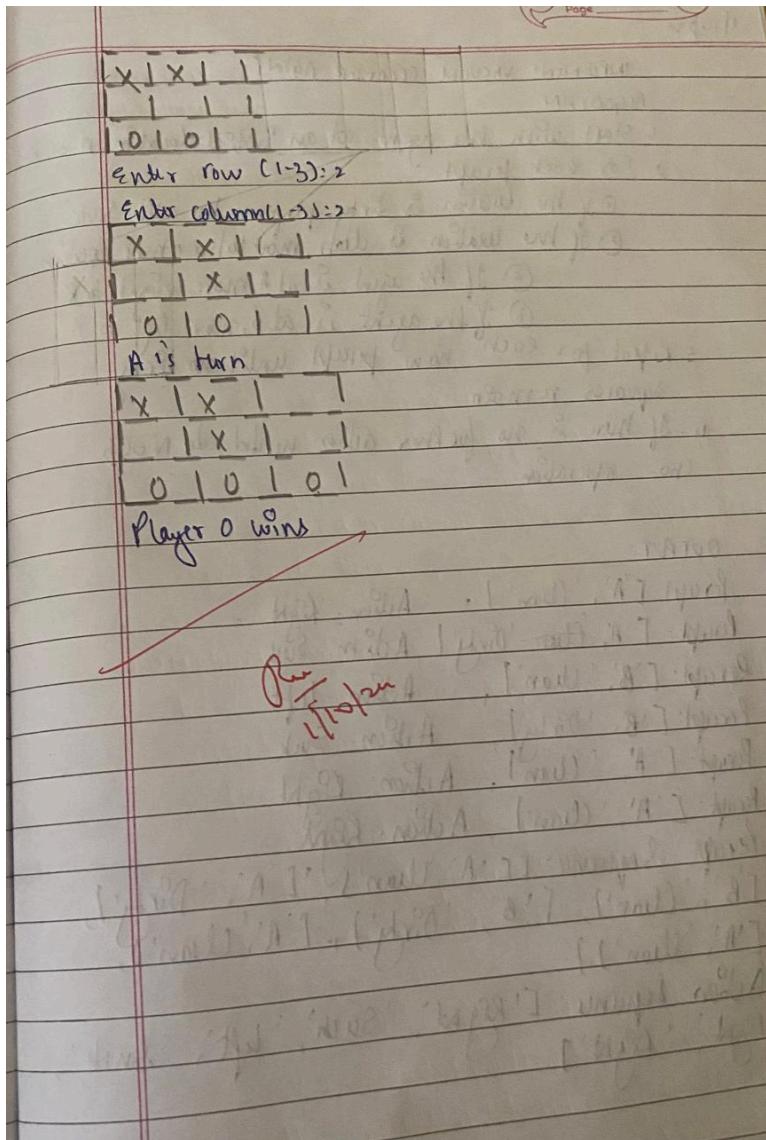
X		

Enter row (1-3):1

Enter column (1-3):2

X	X	

A1's turn--



Code:

```
[5] import random
def print_board(board):
    print("-----")
    for row in board:
        print("|", end="")
        for cell in row:
            print(" " + cell + " |", end="")
        print("\n-----")
def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)):
        return True
    if all(board[i][2 - i] == player for i in range(3)):
        return True
    return False
def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == " ":
                return False
    return True
def get_player_move(board):
    while True:
        try:
            row = int(input("Enter row (1-3): ")) - 1
            col = int(input("Enter column (1-3): ")) - 1
            if 0 <= row <= 2 and 0 <= col <= 2 and board[row][col] == " ":
                return row, col
            else:
                print("Invalid move. Try again.")
        except ValueError:
            print("Invalid input. Enter numbers only.")
def get_ai_move(board):
    empty_cells = []
    for row in range(3):
        for col in range(3):
            if board[row][col] == " ":
                empty_cells.append((row, col))
    if empty_cells:
        return random.choice(empty_cells)
    return None
def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"
    while True:
        print_board(board)
        if current_player == "X":
            row, col = get_player_move(board)
        else:
            print("AI's turn...")
            row, col = get_ai_move(board)
        board[row][col] = current_player
        if check_win(board, current_player):
            print(f"Player {current_player} wins!")
            break
        current_player = "O" if current_player == "X" else "X"
```

```

        print("AI's turn...")
        row, col = get_ai_move(board)
        board[row][col] = current_player
        if check_win(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
            break
        current_player = "O" if current_player == "X" else "X"
    if __name__ == "__main__":
        main()

```

```

import random
def print_board(board):
    print("-----")
    for row in board:
        print("|", end="")
        for cell in row:
            print(" " + cell + " |", end="")
        print("\n-----")
def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)):
        return True
    if all(board[i][2 - i] == player for i in range(3)):
        return True
    return False
def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == " ":
                return False
    return True
def get_player_move(board):
    while True:
        try:
            row = int(input("Enter row (1-3): ")) - 1
            col = int(input("Enter column (1-3): ")) - 1
            if 0 <= row <= 2 and 0 <= col <= 2 and board[row][col] == " ":

```

```

        return row, col
    else:
        print("Invalid move. Try again.")
    except ValueError:
        print("Invalid input. Enter numbers only.")
def get_ai_move(board):
    empty_cells = []
    for row in range(3):
        for col in range(3):
            if board[row][col] == " ":
                empty_cells.append((row, col))
    if empty_cells:
        return random.choice(empty_cells)
    return None
def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"
    while True:
        print_board(board)
        if current_player == "X":
            row, col = get_player_move(board)
        else:
            print("AI's turn...")
            row, col = get_ai_move(board)
            board[row][col] = current_player
        if check_win(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break
        elif is_board_full(board):
            print_board(board)
            print("It's a tie!")
            break
        current_player = "O" if current_player == "X" else "X"
if __name__ == "__main__":
    main()

```

Snapshot of the output: (NEXT PAGE)

```
| | |
| | |
| | |
-----
Enter row (1-3): 1
Enter column (1-3): 1
-----
| X | | |
| | |
| | |
-----
AI's turn...
-----
| X | | |
| | |
| | |
| O | | |
-----
Enter row (1-3): 1
Enter column (1-3): 2
-----
| X | X | |
| | |
| O | | |
-----
AI's turn...
-----
| X | X | |
| | |
| O | O | |
-----
Enter row (1-3): 2
Enter column (1-3): 2
-----
| X | X | |
| | X | |
| O | O | |
-----
AI's turn...
-----
| X | X | |
| | X | |
| O | O | O |
-----
Player O wins!
```

OUTPUT

X		
	X	
		X

Enter row (1-3): 1
Enter column (1-3): 1

X		
	X	
		X

AI's turn...
X | | |
| | |
O | | |

Enter row (1-3): 1
Enter column (1-3): 2
X | X | |
| | |
O | | |

AI's turn...
X | X | |
| | |
O | | |

A red arrow points from the AI's turn entry to the second row of the board.

x	x	
0	0	
Enter row (1-3): 2		
Enter column (1-3): 2		
X	x	
0	0	
A's turn		
x	x	
x		
0	0	0
 Player 0 wins
Player 1 loses

STATE SPACE TREE:

