LAB 1

Tic Tac Toe

Code:-

```
import random

def initialize_board():
    return [[' ' for _ in range(3)] for _ in range(3)]

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

def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return row[0]

    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return board[0][col]

    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]

    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]

    return None

def available_moves(board):
```

```
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

def check_two_in_a_row(board, player):
    for row in range(3):
        if board[row].count(player) == 2 and board[row].count(' ') == 1:
            return row, board[row].index(' ')

    for col in range(3):
        if [board[row][col] for row in range(3)].count(player) == 2:
            empty_index = [row for row in range(3) if board[row][col] == ' ']
            if empty_index:
                return empty_index[0], col



    if [board[i][i] for i in range(3)].count(player) == 2:
        empty_index = [i for i in range(3) if board[i][i] == ' ']
        if empty_index:
            return empty_index[0], empty_index[0]

    if [board[i][2 - i] for i in range(3)].count(player) == 2:
        empty_index = [i for i in range(3) if board[i][2 - i] == ' ']
        if empty_index:
            return empty_index[0], 2 - empty_index[0]

    return None

def make_move(board, player, move):
    board[move[0]][move[1]] = player

def computer_move(board):
```

```
    move = check_two_in_a_row(board, 'O')
    if move:
        make_move(board, 'O', move)
        return


    move = check_two_in_a_row(board, 'X')
    if move:
        make_move(board, 'O', move)
        return


    moves = available_moves(board)
    if moves:
        move = random.choice(moves)
        make_move(board, 'O', move)

def user_move(board):
    while True:
        try:
            row = int(input("Enter row (0-2): "))
            col = int(input("Enter column (0-2): "))
            if board[row][col] == ' ':
                make_move(board, 'X', (row, col))
                return
            else:
                print("That spot is already taken. Try again.")
        except (ValueError, IndexError):
            print("Invalid input. Please enter numbers between 0 and 2.")

def play_game():
```

```
    board = initialize_board()
    players = ['X', 'O']
    current_player = 0

    for _ in range(9):
        display_board(board)
        if current_player == 0:
            user_move(board)
        else:
            computer_move(board)

        winner = check_winner(board)
        if winner:
            display_board(board)
            print(f"Player {winner} wins!")
            return

        current_player = 1 - current_player

    display_board(board)
    print("It's a draw!")


play_game()
```

Output:-

```
>>>
=== RESTART: C:/Users/User/AppData/Local/Programs/Python/Python311/AI labl.py =
 | |
-----
 | |
-----
 | |
-----
Enter row (0-2): 1
Enter column (0-2): 1
 | |
-----
 |X|
-----
 | |
-----
 | |
-----
O|X|
-----
 | |
-----
Enter row (0-2): 1
Enter column (0-2): 2
 | |
-----
O|X|X
-----
 | |
-----
 | |
-----
O|X|X
-----
O| |
-----
Enter row (0-2): 1
Enter column (0-2): 0
That spot is already taken. Try again.
Enter column (0-2): 1
That spot is already taken. Try again.
Enter row (0-2): 2
Enter column (0-2): 1
 | |
-----
O|X|X
-----
O|X|
-----
O| |
-----
O|X|X
-----
O|X|
-----
Player O wins!
```

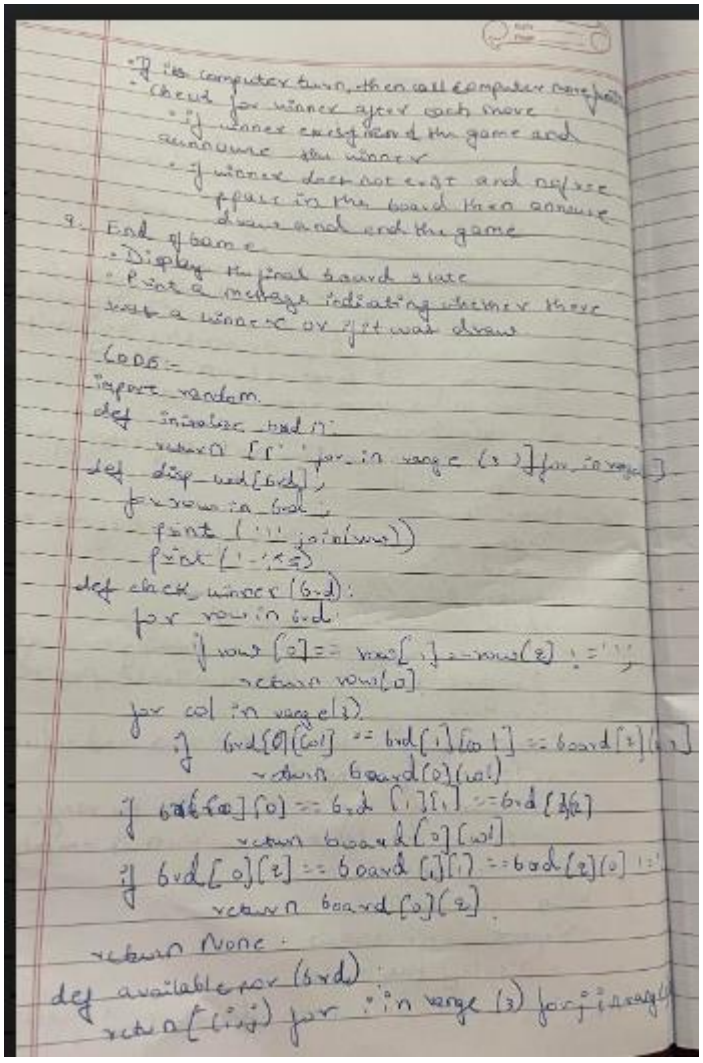
Observation book:

LAB-1

Tic Tac Toe

Algorithm.

1. Initialize the Game
   - Create an empty 3x3 board.
   - Define players : User as 'X' and computer as 'O'
2. Display the Board
   - Print the current state of the board.
3. Check for Winner.
   - Define a function that checks
     - All rows for three matching symbols
     - All columns for three matching symbols
     - All diagonals for the matching 3 Symbols
   - If match found then return winner
   - If not found then return None
4. Available Moves
   - Define function to return a list of empty positions on the board.
5. Make Move
   - Define a function to place a player's symbol on the board.
6. Computer Move Logic
   - Check for a winning move 'o'.
   - If no winning move check for a blocking move against 'X'
   - If neither, select a random available position.
7. User Move Logic.
   - Prompt the user to enter a row and column.
   - Validate the input.
     - Ensure the input is within the range (0-2)
     - Ensure the selected position is empty.
   - If valid, make the move
8. Main Game loop
   - Repeat until draw
     - Display the board
     - If its the user's turn, then call the user function move



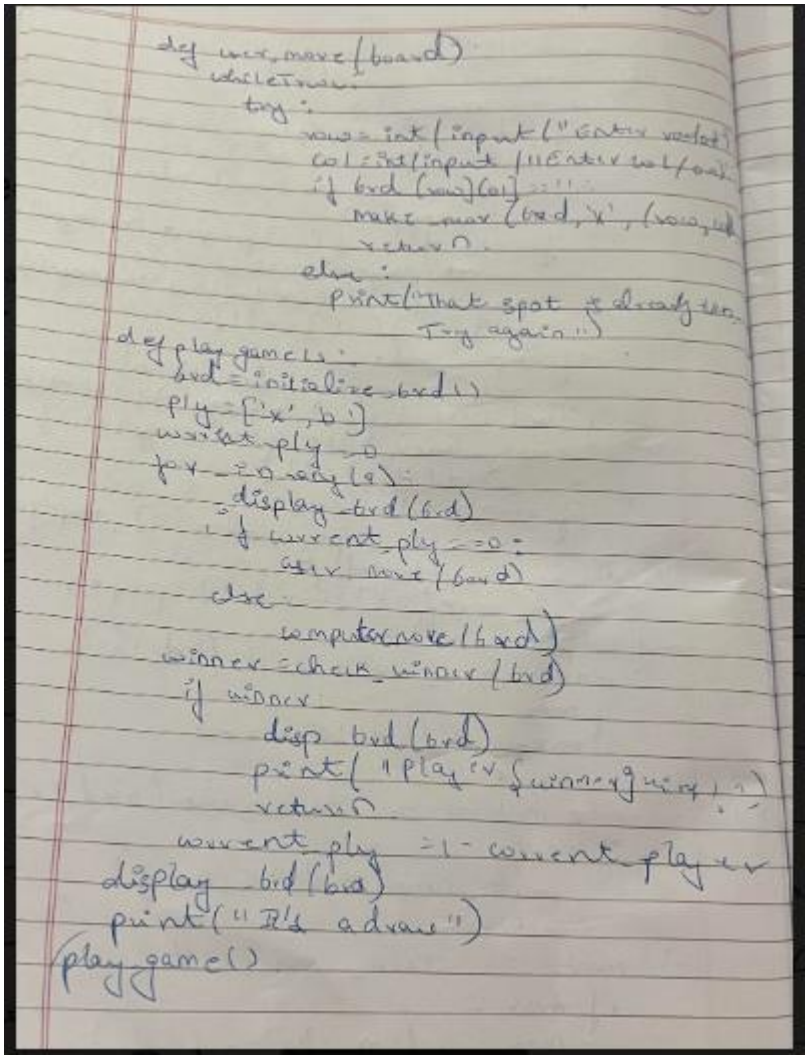
- If its computer turn, then call computer move function
- Check for winner after each move
- If winner exists end the game and announce the winner
- If winner does not exist and no more space in the board then announce draw and end the game

9. End of game
- Display the final board state
- Print a message indicating whether there was a winner or if it was a draw

CODE:

```
import random
def initialize_board():
    return [[' ' for _ in range(3)] for _ in range(3)]
def display_board(board):
    for row in board:
        print('| ' + ' | '.join(row) + ' |')
        print('-------------')
def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != ' ':
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return board[0][2]
    return None
def available_moves(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']
```

```
if board[i][j] == ' ']
def check two in a row (brd, ply):
    for row in range(3):
        if brd[row].count(player) == 2 and
            brd[row].count(' ') == 1:
        return row, brd[row].index(' ')
    for col in range(3):
        if [board[row][col] for row in range(3)
            count(player) == 2:
        empty index = [row for row in range(3)
          if brd[row][col] == ' ']
        if empty index
            return empty index[0], col
    if [brd[i][i] for i in range(3)] count(ply) =
        empty index = [i for i in range(3) if
            board[i][i] = ' ']
        if empty index:
            return empty index[0], empty index
def make move (board, ply, move):
        brd[move[0]][mov[1]] = player
def comp mov (brd):
    mov = check two in a row (brd, 'o')
    if mov:
    make mov (brd, 'o', move)
    return
!= ' ':    mov = check two in a row (brd, 'x')
    if mov:
        make mov [brd, 'o', move)
        return
    move = available mov (brd
    if mov:
        mov = random choice (mov)
        make mov {brd, 'o', mov)
```



```
def user_move(board):
    while True:
        try:
            row = int(input("Enter row (0-2): "))
            col = int(input("Enter col (0-2): "))
            if board[row][col] == ' ':
                make_move(board, 'X', (row, col))
                return
            else:
                print("That spot is already taken. Try again.")
def play_game():
    board = initialize_board()
    players = ['X', 'O']
    current_player = 0
    for _ in range(9):
        display_board(board)
        if current_player == 0:
            user_move(board)
        else:
            computer_move(board)
        winner = check_winner(board)
        if winner:
            display_board(board)
            print(f"Player {winner} wins!")
            return
        current_player = 1 - current_player
    display_board(board)
    print("It's a draw!")
play_game()
```