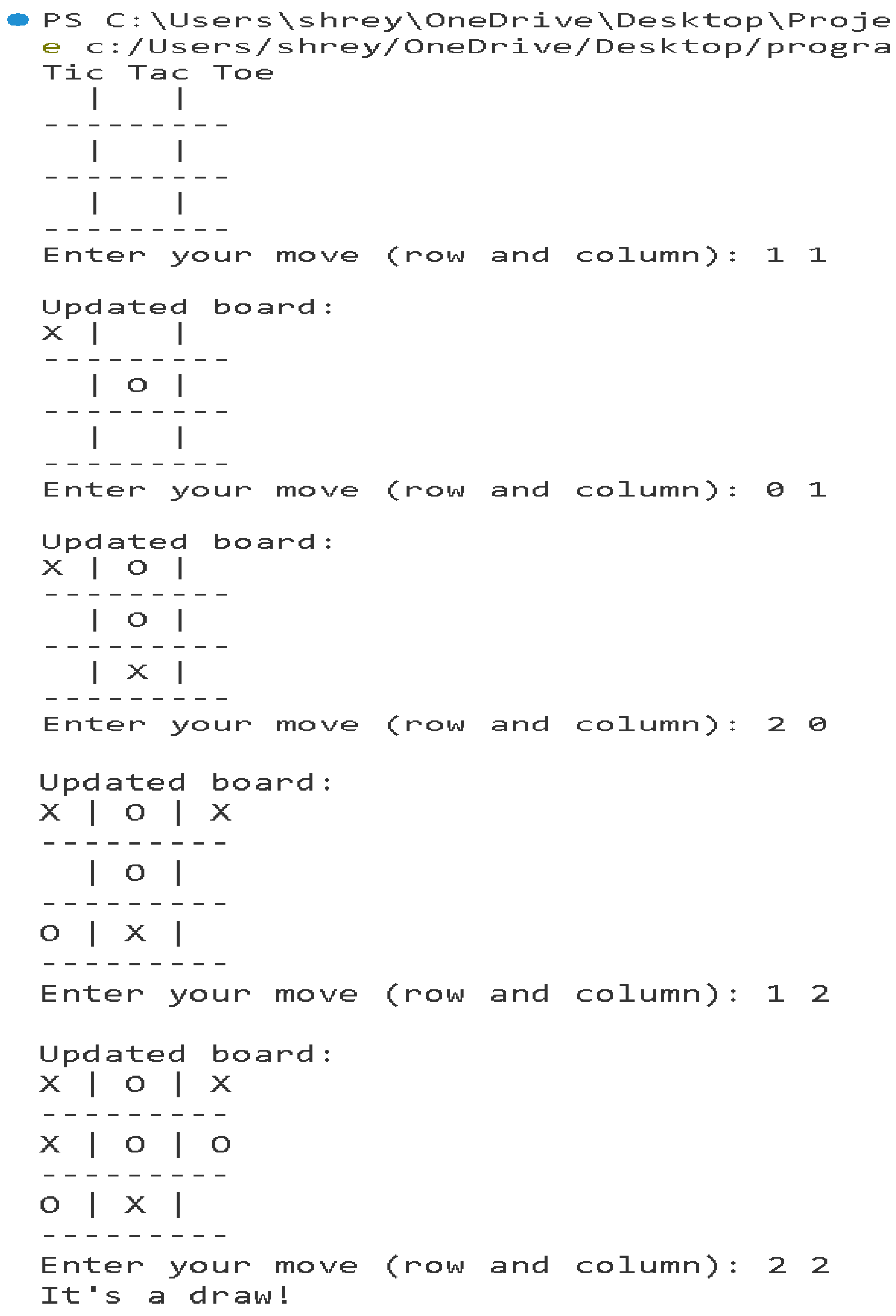
CODE:

```
# 'O' represents the player, 'X' represents the opponent, and ' ' represents an empty cell
initial_board = [
    [' ', ' ', ' '],
    [' ', ' ', ' '],
    [' ', ' ', ' ']
]

def print_board(board):
    for row in board:
        print(' | '.join(row))
        print('-' * 9)

# Function to check if the board is full
def is_full(board):
    return all(cell != ' ' for row in board for cell in row)

# Function to check if a player has won
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)) or all(board[i][2 - i] == player for i in
range(3)):
        return True

    return False

# Function to evaluate the board state
def evaluate(board):
    if check_win(board, 'X'):
        return 1
    elif check_win(board, 'O'):
        return -1
    else:
        return 0

# Min-Max algorithm with alpha-beta pruning
def min_max(board, depth, is_maximizing, alpha, beta):
    if check_win(board, 'X'):
        return 1
    elif check_win(board, 'O'):
        return -1
    elif is_full(board):
        return 0

    if is_maximizing:
        max_eval = float('-inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    eval = min_max(board, depth + 1, False, alpha, beta)
                    board[i][j] = ' '
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    eval = min_max(board, depth + 1, True, alpha, beta)
                    board[i][j] = ' '
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval

# Function to make the best move using Min-Max
def best_move(board):
    best_eval = float('-inf')
    best_move = None

    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'X'
                eval = min_max(board, 0, False, float('-inf'), float('inf'))
                board[i][j] = ' '
                if eval > best_eval:
                    best_eval = eval
                    best_move = (i, j)
    return best_move

# Main game loop
current_board = initial_board
print("Tic Tac Toe")
print_board(current_board)

while True:
    x, y = map(int, input("Enter your move (row and column): ").split())
    if current_board[x][y] == ' ':
        current_board[x][y] = 'O'
    else:
        print("Invalid move. Try again.")
        continue

    if check_win(current_board, 'O'):
        print("You win!")
        break

    if is_full(current_board):
        print("It's a draw!")
        break

    best_x, best_y = best_move(current_board)
    current_board[best_x][best_y] = 'X'

    print("\nUpdated board:")
    print_board(current_board)

    if check_win(current_board, 'X'):
        print("Computer wins!")
        break

    if is_full(current_board):
        print("It's a draw!")
        break
```

OUTPUT:



```
PS C:\Users\shrey\OneDrive\Desktop\Proje
e c:/Users/shrey/OneDrive/Desktop/progra
Tic Tac Toe
  |   |
----------
  |   |
----------
  |   |
----------
Enter your move (row and column): 1 1

Updated board:
X |   |
----------
  | O |
----------
  |   |
----------
Enter your move (row and column): 0 1

Updated board:
X | O |
----------
  | O |
----------
  | X |
----------
Enter your move (row and column): 2 0

Updated board:
X | O | X
----------
  | O |
----------
O | X |
----------
Enter your move (row and column): 1 2

Updated board:
X | O | X
----------
X | O | O
----------
O | X |
----------
Enter your move (row and column): 2 2
It's a draw!
```