Assignment No:- 2

Code:-

```
import heapq

import math


class TicTacToeNode:

    def __init__(self, board, player, move=None):

        self.board = board

        self.player = player

        self.move = move


    def __lt__(self, other):

        return False  # A* algorithm do0es not rely on the comparison of nodes


def print_board(board):

    for row in board:

        print(" ".join(row))

    print()


def is_winner(board, player):

    # Check rows, columns, and diagonals for a win

    for i in range(3):

        if all(cell == player for cell in board[i]) or all(board[j][i] == player for j 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


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


def game_over(board):
    # Check if the game is over
    for player in ['X', 'O']:
        if is_winner(board, player):
            return True, player
    if is_board_full(board):
        return True, 'Tie'
    return False, None


def heuristic(board, player):
    # A simple heuristic: +1 for 'O' in a line, -1 for 'X' in a line
    score = 0
    for i in range(3):
        for j in range(3):
            if board[i][j] == 'O':
                score += 1
            elif board[i][j] == 'X':
                score -= 1
    return score


def a_star_search(initial_node):
    open_set = []
    closed_set = set()

    heapq.heappush(open_set, (heuristic(initial_node.board, initial_node.player), initial_node))

    while open_set:
        _, current_node = heapq.heappop(open_set)

        if game_over(current_node.board)[0]:
            return current_node

        closed_set.add(tuple(map(tuple, current_node.board)))

        for i in range(3):
            for j in range(3):
                if current_node.board[i][j] == ' ':
                    new_board = [row[:] for row in current_node.board]
                    new_board[i][j] = current_node.player
                    new_player = 'X' if current_node.player == 'O' else 'O'
                    child_node = TicTacToeNode(new_board, new_player, move=(i, j))

                    if tuple(map(tuple, child_node.board)) not in closed_set:
                        heapq.heappush(open_set, (heuristic(child_node.board, child_node.player), 
child_node))

    return None
```

```
def play_tic_tac_toe():
    board = [[' ' for _ in range(3)] for _ in range(3)]

    while not game_over(board)[0]:
        print_board(board)

        # Player's move
        row = int(input("Enter row (0, 1, or 2): "))
        col = int(input("Enter column (0, 1, or 2): "))
        if board[row][col] == ' ':
            board[row][col] = 'X'
        else:
            print("Cell already occupied. Try again.")
            continue

        if game_over(board)[0]:
            break

        # AI's move
        print("AI's move:")
        ai_node = a_star_search(TicTacToeNode(board, 'O'))
        ai_row, ai_col = ai_node.move
        board[ai_row][ai_col] = 'O'

    print_board(board)
    result = game_over(board)[1]
    if result == 'Tie':
        print("It's a tie!")
    else:
        print(f"{result} wins!")


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

Output :-

```
PS F:\3.Programming\.vscode\AI Assignments> &
C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.10.exe
"f:/3.Programming/.vscode/AI Assignments/Assignment No.2 Code.py"



Enter row (0, 1, or 2): 0

Enter column (0, 1, or 2): 0

AI's move:

X

O



Enter row (0, 1, or 2): 1

Enter column (0, 1, or 2): 1

AI's move:

X

O X

  O
```

```
Enter row (0, 1, or 2): 0

Enter column (0, 1, or 2): 1

AI's move:

X X

O X

  O O


Enter row (0, 1, or 2): 0

Enter column (0, 1, or 2): 2

X X X

O X

  O O


X wins!
```