NAME: - Rushikesh Sunil Mangalkar **ROLL NO-** 3202007

ASSIGNMENT 2

CODE:

```
import heapq

class Node:
    def __init__(self, x, y, board, player):
        self.x = x
        self.y = y
        self.board = board
        self.player = player
        self.g_cost = float('inf')
        self.h_cost = float('inf')
        self.f_cost = float('inf')
        self.parent = None

    def __lt__(self, other):
        return self.f_cost < other.f_cost

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y and self.board == other.board

    def __hash__(self):
        return hash((self.x, self.y, str(self.board)))

def heuristic(a, b):
    return 0

def get_neighbors(node, player):
    neighbors = []
    for i in range(3):
        for j in range(3):
            if node.board[i][j] == ' ':
                new_board = [row[:] for row in node.board]
                new_board[i][j] = player
                neighbors.append(Node(i, j, new_board, player))
    return neighbors

def check_winner(board, player):
    for row in range(3):
        if all([board[row][col] == player for col in range(3)]):
            return True
    for col in range(3):
        if all([board[row][col] == player for row in range(3)]):
            return True
    if board[0][0] == player and board[1][1] == player and board[2][2] == player:
        return True
    if board[0][2] == player and board[1][1] == player and board[2][0] == player:
        return True
    return False
```

```
def find_block_move(board, player):
    opponent = 'O' if player == 'X' else 'X'
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                new_board = [row[:] for row in board]
                new_board[i][j] = opponent
                if check_winner(new_board, opponent):
                    return i, j
    return None


def display_board(board):
    print("\nBoard Layout:")
    print("--------------")
    for i in range(3):
        row = "| "
        for j in range(3):
            row += board[i][j] + " | "
        print(row)
        print("--------------")


def a_star(start, player):
    open_set = []
    closed_set = set()

    start.g_cost = 0
    start.h_cost = heuristic(start, None)
    start.f_cost = start.g_cost + start.h_cost

    heapq.heappush(open_set, start)

    while open_set:
        current = heapq.heappop(open_set)

        if check_winner(current.board, player):
            return [(current.x, current.y)]

        closed_set.add(current)

        for neighbor in get_neighbors(current, player):
            if neighbor in closed_set:
                continue

            tentative_g_cost = current.g_cost + 1
            if tentative_g_cost < neighbor.g_cost:
                neighbor.g_cost = tentative_g_cost
                neighbor.h_cost = heuristic(neighbor, None)
                neighbor.f_cost = neighbor.g_cost + neighbor.h_cost
                neighbor.parent = current

                if neighbor not in open_set:
                    heapq.heappush(open_set, neighbor)

    return None
```

```
def is_tie(board):
    for row in board:
        if ' ' in row:
            return False
    return True


def tic_tac_toe():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    turn = 'X'
    moves_left = 9

    while moves_left > 0:
        display_board(board)

        if turn == 'X':
            print(f"Player {turn}'s turn:")
            while True:
                try:
                    cell = int(input(f"Enter a cell number (1-9): ")) - 1
                    if cell < 0 or cell > 8:
                        print("Invalid cell number! Please enter a number between 1 and 9.")
                        continue

                    row, col = divmod(cell, 3)

                    if board[row][col] != ' ':
                        print("This cell is already taken. Please choose another cell.")
                        continue

                    board[row][col] = turn
                    moves_left -= 1
                    break
                except ValueError:
                    print("Invalid input. Please enter an integer between 1 and 9.")
        else:
            print(f"AI (Player {turn})'s turn:")
            block_move = find_block_move(board, 'O')
            if block_move:
                print(f"AI blocks Player X's winning move at cell {block_move[0]*3 + block_move[1] + 1}")
                board[block_move[0]][block_move[1]] = 'O'
            else:
                start_node = Node(0, 0, board, 'O')
                move = a_star(start_node, 'O')

                if move:
                    best_move = move[-1]
                    row, col = best_move
                    board[row][col] = turn
                    moves_left -= 1
                    print(f"AI places 'O' in cell {row*3 + col + 1}")

        if check_winner(board, turn):
            display_board(board)
            print(f"Player {turn} wins!")
            return
```

```
        if is_tie(board):
            display_board(board)
            print("It's a tie!")
            return

        turn = 'O' if turn == 'X' else 'X'

tic_tac_toe()
```

OUTPUT:

```
Board Layout:
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Player X's turn:
Enter a cell number (1-9): 1

Board Layout:
-------------
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
AI (Player O)'s turn:
AI places 'O' in cell 7

Board Layout:
-------------
| X |   |   |
-------------
|   |   |   |
-------------
| O |   |   |
-------------
Player X's turn:
Enter a cell number (1-9): 5

Board Layout:
-------------
| X |   |   |
-------------
|   | X |   |
-------------
| O |   |   |
-------------
AI (Player O)'s turn:
AI blocks Player X's winning move at cell 9
```

Board Layout:

| X | | |

| | X | |

| O | | O |

Player X's turn:
Enter a cell number (1-9): 8

Board Layout:

| X | | |

| | X | |

| O | X | O |

AI (Player O)'s turn:
AI blocks Player X's winning move at cell 2

Board Layout:

| X | O | |

| | X | |

| O | X | O |

Player X's turn:
Enter a cell number (1-9): 3

Board Layout:

| X | O | X |

| | X | |

| O | X | O |

AI (Player O)'s turn:

Board Layout:

| X | O | X |

| | X | |

| O | X | O |

Player X's turn:
Enter a cell number (1-9): 4

Board Layout:

| X | O | X |

| X | X | |

| O | X | O |

AI (Player O)'s turn:
AI blocks Player X's winning move at cell 6

Board Layout:

| X | O | X |

| X | X | O |

| O | X | O |

It's a tie!