# **ass 2 tic tac toe**

# Implement A star Algorithm for any game search problem.

```
import numpy as np


class Node:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent
        self.g_score = 0 if parent is None else parent.g_score + 1      #r
        self.h_score = self.heuristic()


    def f_score(self):
        return self.g_score + self.h_score


    def path(self):
        path = [self.state]
        node = self.parent
        while node is not None:
            path.append(node.state)
            node = node.parent
        return path[::-1]


    def heuristic(self):
        winner = check_winner(self.state)
        if winner is not None:
            #3
            if winner == 1:
                return 100 - self.g_score
            else:
                return -100 + self.g_score
        else:
            return self.get_empty_spaces() - self.get_opponent_empty_spaces()

    def get_empty_spaces(self):
        return np.sum(self.state == -1)

    def get_opponent_empty_spaces(self):
        return np.sum(self.state == 1)          #x

    def __eq__(self, other):
        return np.array_equal(self.state, other.state)

    def __hash__(self):
        return hash(self.state.tostring())

def check_winner(state):
    # Check rows
    for i in range(3):
        #o
        if np.all(state[i, :] == 1):
            return 1
        elif np.all(state[i, :] == 0):
            return 0

    # Check columns
    for i in range(3):
        if np.all(state[:, i] == 1):
            return 1
        elif np.all(state[:, i] == 0):
            return 0

    # Check diagonals
    if np.all(np.diag(state) == 1) or np.all(np.diag(np.fliplr(state)) == 1):
        return 1
    elif np.all(np.diag(state) == 0) or np.all(np.diag(np.fliplr(state)) == 0):
        return 0

    # Check for tie
    if np.sum(state == -1) == 0:
        return -1

    # No winner yet     #r
    return None

def get_possible_moves(state, player):
    moves = []
    for i in range(3):
        for j in range(3):
            if state[i, j] == -1:
                new_state = state.copy()
                new_state[i, j] = player
                moves.append(new_state)
    return moves

def a_star(start_state, player):
    open_list = [Node(start_state)]
    closed_list = []

    while open_list:
        current = min(open_list, key=lambda x: x.f_score())
        open_list.remove(current)
        closed_list.append(current)

        if check_winner(current.state) is not None:
            # If the current state is a win for the AI player, return the path
            return current.path()

        for child_state in get_possible_moves(current.state, player):
            child = Node(child_state, current)
            if child in closed_list:
                continue
            if child not in open_list:
                open_list.append(child)
            else:
                # Update the existing node if this path is better
                existing_child = open_list[open_list.index(child)]
                if child.g_score < existing_child.g_score:
                    existing_child.parent = current

    # If no path is found, return None
    return None

def print_board(state):
    """
    Prints the Tic Tac Toe board in a human-readable format.
    """
    symbols = {-1: " ", 0: "O", 1: "X"}  # Map player numbers to symbols
    for i in range(3):
        print("-------------")
        row = "|"
        for j in range(3):
            row += " " + symbols[state[i, j]] + " |"
        print(row)
    print("-------------")



def main():
    # Initialize the game board
    board = np.full((3, 3), -1)
    print_board(board)

    # Game loop
    while True:
        # Player 1 (human) turn
        print("Player 1 (X) turn.")
        row = int(input("Enter row number (0-2): "))
        col = int(input("Enter column number (0-2): "))
        if board[row, col] != -1:
            print("Invalid move. Try again.")
            continue
        board[row, col] = 1
        print_board(board)
        winner = check_winner(board)
        if winner is not None:
            break

        # AI player (player 2) turn
        print("Player 2 (O) turn.")
        path = a_star(board, 0)
        if path is None:
            print("Error: AI failed to find a valid move.")
            continue
        board = path[1]
        print_board(board)
        winner = check_winner(board)
        if winner is not None:
            break

    # Print the result
    if winner == 1:
        print("Player 1 (X) wins!")
    elif winner == 0:
        print("Player 2 (O) wins!")
    else:
        print("It's a tie!")

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

<u># ass 2 done</u>