NAME:- DIVYANG BAGLA
PANEL:- C PC-33
C-3

PAGE NO.:

AI LAB ASSIGNMENT - 2

AIM:- To solve tictactoe using minmax algorithm.

OBJECTIVE:- To study and impeement min-max algorithm for tic tac toe.

THEORY:- Adversarial search:-
This is the search when there is an enemy changing the state of problem every step in a direction you don't want. Eg. chess etc.
Tic Tac Toe solving steps:-
1.) check if game has reached terminal state & return a value depending on the outcome.
2) generate all available moves
3.) call the minmax fun. on every available step move recursively to reach terminal state.
4.) Evaluate collection of sorted movies.
5) Return optimal move.

MINMAX:- It is a kind of back tracking i.e. used in decision making & game theory to find optimal value of player.

INPUT:- Initial stage
OUTPUT:- Final stage.

FAQ's

Q.1 compare informed search & adversial search?
→ Adversial search is a search where we examine the problem which arises when we try to plan ahead of world and other agent are planning against us.

PAGE NO.:

→ Informed search is more careful for large search spaces. It used concept of heuristics

Q.2 What is Alpha-beta pruning?
= · Modified version of min-max algorithm
· Optimization technique for min-max algorithm.
· Can be applied at any depth of tree sometimes it only prunes entire sub tree.

Q.3 Min Max Algorithm:

```
function MinMax(node, depth, maximizing player):
    if depth=0 or node=terminal node then
        return static evaluation of node.
    if maximizing player then
        return max Eva = ∞;
    for each child node do
        eva = min max(child, depth-1, false)
    max eva = max (max Eva , eva)
    return max Eva
   else
    min Eva = infinity;
    for each child of node)
        eva = minmax(child, depth-1, true)
        min eva= min (mineva, eva)
    return mineva;
```

AI lab 2 Code

```
from math import inf as infinity
from random import choice
import platform
import time
from os import system

HUMAN = -1
COMP = +1
board = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
]


def evaluate(state):
    if wins(state, COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0

    return score


def wins(state, player):
    win_state = [
        [state[0][0], state[0][1], state[0][2]],
        [state[1][0], state[1][1], state[1][2]],
        [state[2][0], state[2][1], state[2][2]],
        [state[0][0], state[1][0], state[2][0]],
        [state[0][1], state[1][1], state[2][1]],
        [state[0][2], state[1][2], state[2][2]],
        [state[0][0], state[1][1], state[2][2]],
        [state[2][0], state[1][1], state[0][2]],
    ]
    if [player, player, player] in win_state:
        return True
    else:
        return False


def game_over(state):
    return wins(state, HUMAN) or wins(state, COMP)


def empty_cells(state):
```

```
    cells = []

    for x, row in enumerate(state):
        for y, cell in enumerate(row):
            if cell == 0:
                cells.append([x, y])

    return cells


def valid_move(x, y):
    if [x, y] in empty_cells(board):
        return True
    else:
        return False


def set_move(x, y, player):
    if valid_move(x, y):
        board[x][y] = player
        return True
    else:
        return False


def minimax(state, depth, player):

    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]

    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]

    for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y

        if player == COMP:
            if score[2] > best[2]:
                best = score  # max value
        else:
            if score[2] < best[2]:
                best = score  # min value
```

```
    return best


def render(state, c_choice, h_choice):
    chars = {
        -1: h_choice,
        +1: c_choice,
        0: ' '
    }
    str_line = '---------------'

    print('\n' + str_line)
    for row in state:
        for cell in row:
            symbol = chars[cell]
            print(f'| {symbol} |', end='')
        print('\n' + str_line)


def ai_turn(c_choice, h_choice):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    print(f'Computer turn [{c_choice}]')
    render(board, c_choice, h_choice)

    if depth == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
    else:
        move = minimax(board, depth, COMP)
        x, y = move[0], move[1]

    set_move(x, y, COMP)
    time.sleep(1)


def human_turn(c_choice, h_choice):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    move = -1
    moves = {
        1: [0, 0], 2: [0, 1], 3: [0, 2],
        4: [1, 0], 5: [1, 1], 6: [1, 2],
        7: [2, 0], 8: [2, 1], 9: [2, 2],
    }
```

```
    print(f'Human turn [{h_choice}]')
    render(board, c_choice, h_choice)

    while move < 1 or move > 9:
        try:
            move = int(input('Use numpad (1..9): '))
            coord = moves[move]
            can_move = set_move(coord[0], coord[1], HUMAN)

            if not can_move:
                print('Bad move')
                move = -1
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')


def main():
    #clean()
    h_choice = ''  # X or O
    c_choice = ''  # X or O
    first = ''  # if human is the first

    while h_choice != 'O' and h_choice != 'X':
        try:
            print('')
            h_choice = input('Choose X or O\nChosen: ').upper()
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')

    if h_choice == 'X':
        c_choice = 'O'
    else:
        c_choice = 'X'

    while first != 'Y' and first != 'N':
        try:
            first = input('First to start?[y/n]: ').upper()
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')
```

```
    while len(empty_cells(board)) > 0 and not game_over(board):
        if first == 'N':
            ai_turn(c_choice, h_choice)
            first = ''

        human_turn(c_choice, h_choice)
        ai_turn(c_choice, h_choice)


    if wins(board, HUMAN):
        print(f'Human turn [{h_choice}]')
        render(board, c_choice, h_choice)
        print('YOU WIN!')
    elif wins(board, COMP):
        print(f'Computer turn [{c_choice}]')
        render(board, c_choice, h_choice)
        print('YOU LOSE!')
    else:
        render(board, c_choice, h_choice)
        print('DRAW!')

    exit()

if __name__ == '__main__':
    main()


'''
OUTPUT :-

Choose X or O
Chosen: X
First to start?[y/n]: y
Human turn [X]

---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): 1
Computer turn [O]

---------------
| X ||   ||   |
---------------
|   ||   ||   |
---------------
```

```
|   ||   ||   |
---------------
Human turn [X]

---------------
| X ||   ||   |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): 3
Computer turn [O]

---------------
| X ||   || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
Human turn [X]

---------------
| X || O || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): 8
Computer turn [O]

---------------
| X || O || X |
---------------
|   || O ||   |
---------------
|   || X ||   |
---------------
Human turn [X]

---------------
| X || O || X |
---------------
| O || O ||   |
---------------
|   || X ||   |
---------------
Use numpad (1..9): 6
```

```
Computer turn [O]

---------------
| X || O || X |
---------------
| O || O || X |
---------------
|   || X ||   |
---------------
Human turn [X]

---------------
| X || O || X |
---------------
| O || O || X |
---------------
|   || X || O |
---------------
Use numpad (1..9): 7

---------------
| X || O || X |
---------------
| O || O || X |
---------------
| X || X || O |
---------------
DRAW!

'''
```