EXP NO : 5

DATE :

MINIMAX ALGORITHM

AIM :

To Implement MINIMAX Algorithm problem using Python.

PROGRAM :

```
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
            else :
                if score[2] < best[2]:
                    best = score
        return best
    def clean():
        os_name = platform.system().lower()
        if 'windows' in os_name :
            system('cls')
        else :
            system('clear')
    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
    clean()
    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
def main():
    clean()
    h_choice = ''
    c_choice = ''
    first = ''
    while h_choice!='O' and h_choice!='X':
        try:
            print('')
            h_choice = input('Choose X or O\n Chosen:').upper()
        except(EOFError, KeyBoardInterrupt):
            print('Bye')
            exit()
        except(KeyError, ValueError):
            print('Bad choice')
    if h_choice=='X':
        c_choice = 'O'
    else:
        c_choice = 'X'
    clean()
```

```
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):
    clean()
    print(f'Human turn[{h_choice}]')
    render(board, c_choice, h_choice)
    print('You win')
elif wins(boaird, COMP):
    clean()
    print(f'Computer turn[{c_choice}]')
    render(board, c_choice, h_choice)
    print('You lose!')
else:
    clean()
    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): 4
Computer turn [O]
 |  ||  ||  |
 | X||  ||  |
 |  ||  ||  |
Human turn [X]
 |  ||  ||  |
 |  ||  ||  |
 |  ||  ||  |
Use numpad (1..9): 2
Computer turn [O]
 | O|| X||  |
 | X||  ||  |
 |  ||  ||  |
Human turn [X]
 | O|| X||  |
 | X|| O||  |
 |  ||  ||  |
Use numpad (1..9): 3
Computer turn [O]
 | O|| X|| X |
 | X|| O||  |
 |  ||  ||  |
```

RESULT:

Thus the python program has been implemented to solve minimax algorithm