Experiment No.: 5 MinMax Algorithm

Aim:

To implement Minmax Algorithm problem using Python

Source 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 minmax(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 = minmax(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's turn [{c_choice}]')
    render(board, c_choice, h_choice)
    if depth == 9:
        x = choice([0,1,2])
        y = choice([0,1,2])
    else:
        move = minmax(board, depth, COMP)
        x, y = move[0], move[1]
    set_move(x,y,COMP)
    time.sleep(1)

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

Output:

```
Choose : X or O
Choosen: X
First to start? [Y/n]: Y
Human turn [X]

 O   X   X
 X   O
         O

You lose!
```

Result:

Thus, the code of Min Max Algorithm is successfully executed.