Date:

MINIMAX ALGORITHM

EX: 4

Aim:

To implement Minimax 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[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 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],
            }
    clean()
    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' # X or O
    c_choice = 'O' # 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\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 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):
    clean()
    print(f'Human turn [{h_choice}]')
    render(board, c_choice, h_choice)
    print('you win!')
elif win(board, 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('you lose!')
    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
```

Page
Date

Result:

Thus, the program for minimax is successfully executed & the output is verified.

EX : 5

Date:

Aim:

To implem

using python.

source code:

from collectio

class graph:

def __init__

self.

def get

retu

def hi(se

H = {

return

def a

open

close

poo

poo

par

par

wh

self.hC