220701077

classmate

Date

Page

EXP NO - 5 MINIMAX ALGORITHM

AIM:

To implement minimax algorithm using python

ALGORITHM:

1) create a 3x3 grid for the board filled with zeros for empty cells.

2) ASSIGN HUMAN as -1 and comp as +1, and let the human choose X or O.

3) Decide if human or computer goes first.

4) Display the board, prompt the human for a move, validate it and update board.

5) use the minimax algorithm to find the best move and update board

6) If the game is won, lost or drawn return score (+1,-1

7) simulate moves for each empty cell, recursively call minimax and pick best score.

8) check for a winner or full board do declare win, l or draw.

9) show the final board and print "YOU WIN", "YOU LOSE" "DRAW" based on outcome.

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
```

220701077

classmate

Date

Page

```
def minimax (state, depth, player):
    if player == comp:
        best = [-1, -1, -infinity]
    else:
        best=[-1, -1, +infinity]

    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 clear():
    os_name = platform.system().lower()
    if 'windows' in os_name:
        system('cls')
    else:
        system('clear')

def render(state, c_choic, 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 == 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
    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 = ''
 c_choice = ''
 first = ''
 while h_choice != 'o' and h_choice != 'x':
  try:
   h_choice = input('choose x or o\nchoice:').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 wins(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('Draw!')
if __name__ == '__main__':
    main()
```

220701077

classmate

Date

Page

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]

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

---------------
| O ||   || X |
---------------
| X || X || O |
---------------
| O ||   ||   |
---------------
Use numpad (1..9): 2
Computer turn [O]

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

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

```
Human turn [X]

---------------
| O ||   || X |
---------------
| X || X || O |
---------------
| O ||   ||   |
---------------
Use numpad (1..9): 2
Computer turn [O]

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

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

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

RESULT:

Thus the program to implement minimax algorithm is successfully executed and verified.