Reg No : 231801151

Name : Sanjay S P

# Minmax Algorithm

Program:

from math import inf

from random import choice

import platform

import time

from os import system

hum = -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, hum):

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[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,hum) 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,-inf]

else:

best = [-1,-1,+inf]

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

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], hum)

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:

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'

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, hum):

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")

exit()

if \_\_name\_\_ == '\_\_main\_\_':

main()