Practical No. 3B

Tic - Tac - Toe Game Implementation (MinMax)

**Q.1]** Write a Python program for implementing a Tic-Tac-Toe game using MinMax algorithm.

**Code:**

import copy

import random

import time

class TickTacToe:

mat= []

turnNum = 0

calls = 0

def printAfterTurn(self):

print("Game after you turn:")

self.printKeymat()

def printKeymat(self):

for i in range(3):

for j in range(3):

print(self.mat[i][j],"|",end="")

print("\n")

def initMat(self):

self.mat = [[" " for i in range(3)] for j in range(3)]

def startGame(self):

whichTurn = 0

isWon = False

while (self.turnNum < 9):

if whichTurn == 0:

whichTurn = 1

self.turnNum += 1

self.player1Input()

if (self.checkWin("O")):

isWon = True

print("Player 1 win")

break

else:

whichTurn = 0

self.turnNum += 1

self.player2Play()

if (self.checkWin("X")):

isWon = True

print("Player Bot win")

break

if isWon == False:

print("Game draw")

def player1Input(self):

print("Player 1 Turn:")

pos = self.takeInput()

self.fillMat(pos, 0)

self.printAfterTurn()

def player2Play(self):

print("Bot turn:")

*#pos = self.getEmptyPos()*

max = self.minMax(copy.deepcopy(self.mat), (9 - self.turnNum), True, -1, -1, 0)

pos = [max[1], max[2]]

self.fillMat(pos, 1)

self.printAfterTurn()

def getEmptyPos(self):

rnList = []

for i in range(3):

for j in range(3):

if (self.mat[i][j] == " "):

rnList.append((i, j))

random.shuffle(rnList)

return rnList[0]

def getMatStates(self, mat, whichTurn):

rnList = []

states = []

for i in range(3):

for j in range(3):

if (mat[i][j] == " "):

rnList.append((i, j))

newStateMat = copy.deepcopy(mat)

if (whichTurn == 0):

newStateMat[i][j] = "O"

else:

newStateMat[i][j] = "X"

states.append([newStateMat, i, j])

return states

def minMax(self, node, depth, isMaxPlayer, i, j, deapthSearch):

""" if (self.calls > 900):

return

else:

self.calls += 1 """

self.calls += 1

print("Searching for best possible move... calls:",self.calls,"/ 59705")

*#print("DepathSearcH:",deapthSearch)*

if (depth == -1):

*#print("called till here",node, depth, isMaxPlayer, i ,j, "\n")*

if self.checkIsMatWin(node, "X"):

*#time.sleep(50)*

return [100, i, j]

elif self.checkIsMatWin(node, "O"):

return [-100, i, j]

else:

return [0, i, j]

elif self.checkIsMatWin(node, "X"):

*#print("x wing",node,depth,isMaxPlayer)*

*#time.sleep(10)*

return [100, i, j]

elif self.checkIsMatWin(node, "O"):

*#print("o wing",node,depth,isMaxPlayer)*

*#time.sleep(10)*

return [-100, i, j]

if isMaxPlayer:

bestVal = [-99999999, 0, 0]

nextStates = self.getMatStates(node, 1)

highVal = []

*#print(nextStates, node, depth, isMaxPlayer, "\n")*

for childNode in nextStates:

v = [-99999999, 0, 0]

v = self.minMax(childNode[0], depth-1, False, childNode[1], childNode[2], deapthSearch+1)

*#bestVal = max(bestVal, v)*

highVal.append(v)

highVal.sort(key=lambda x:x[0], reverse=True)

*#print("HIgles: ",highVal, depth, isMaxPlayer)*

return highVal[0]

else:

bestVal = +99999999

nextStates = self.getMatStates(node, 0)

highVal = []

*#print(nextStates, node, depth, isMaxPlayer, "\n")*

for childNode in nextStates:

v = self.minMax(childNode[0], depth-1, True, childNode[1], childNode[2], deapthSearch+1)

*#bestVal = min(bestVal, v)*

highVal.append(v)

highVal.sort(key=lambda x:x[0])

*#print("HIgles: ",highVal, depth, isMaxPlayer)*

return highVal[0]

def fillMat(self, pos, whichTurn):

row, col = pos

if (whichTurn == 0):

self.mat[row][col] = "O"

else:

self.mat[row][col] = "X"

def takeInput(self):

valid = False

numRow, numCol = 0, 0

while (valid == False):

numRow = int(input("Enter the row (0-2): "))

numCol = int(input("Enter the col (0-2): "))

if (self.mat[numRow][numCol] == " "):

valid = True

else:

print("Invalid input, try again!")

return (numRow, numCol)

def checkIsMatWin(self, mat, whichTurn):

isWin = False

if (mat[0][0] == whichTurn and mat[0][1] == whichTurn and mat[0][2] == whichTurn):

isWin = True

elif (mat[1][0] == whichTurn and mat[1][1] == whichTurn and mat[1][2] == whichTurn):

isWin = True

elif (mat[2][0] == whichTurn and mat[2][1] == whichTurn and mat[2][2] == whichTurn):

isWin = True

elif (mat[0][0] == whichTurn and mat[1][0] == whichTurn and mat[2][0] == whichTurn):

isWin = True

elif (mat[0][1] == whichTurn and mat[1][1] == whichTurn and mat[2][1] == whichTurn):

isWin = True

elif (mat[0][2] == whichTurn and mat[1][2] == whichTurn and mat[2][2] == whichTurn):

isWin = True

elif (mat[0][0] == whichTurn and mat[1][1] == whichTurn and mat[2][2] == whichTurn):

isWin = True

elif (mat[0][2] == whichTurn and mat[1][1] == whichTurn and mat[2][0] == whichTurn):

isWin = True

return isWin

Game = TickTacToe()

Game.initMat()

Game.startGame()

**Output:**







