Week 2

Implement tick tack toe in Ai

def ConstBoard(board):

print("Current State Of Board : \n\n");

for i in range (0,9):

if((i>0) and (i%3)==0):

print("\n");

if(board[i]==0):

print("- ",end=" ");

if (board[i]==1):

print("O ",end=" ");

if(board[i]==-1):

print("X ",end=" ");

print("\n\n");

#This function takes the user move as input and make the required changes on the board.

def User1Turn(board):

pos=input("Enter X's position from [1...9]: ");

pos=int(pos);

if(board[pos-1]!=0):

print("Wrong Move!!!");

exit(0) ;

board[pos-1]=-1;

def User2Turn(board):

pos=input("Enter O's position from [1...9]: ");

pos=int(pos);

if(board[pos-1]!=0):

print("Wrong Move!!!");

exit(0);

board[pos-1]=1;

#MinMax function.

def minimax(board,player):

x=analyzeboard(board);

if(x!=0):

return (x\*player);

pos=-1;

value=-2;

for i in range(0,9):

if(board[i]==0):

board[i]=player;

score=-minimax(board,(player\*-1));

if(score>value):

value=score;

pos=i;

board[i]=0;

if(pos==-1):

return 0;

return value;

#This function makes the computer's move using minmax algorithm.

def CompTurn(board):

pos=-1;

value=-2;

for i in range(0,9):

if(board[i]==0):

board[i]=1;

score=-minimax(board, -1);

board[i]=0;

if(score>value):

value=score;

pos=i;

board[pos]=1;

#This function is used to analyze a game.

def analyzeboard(board):

cb=[[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6]];

for i in range(0,8):

if(board[cb[i][0]] != 0 and

board[cb[i][0]] == board[cb[i][1]] and

board[cb[i][0]] == board[cb[i][2]]):

return board[cb[i][2]];

return 0;

#Main Function.

def main():

choice=input("Enter 1 for single player, 2 for multiplayer: ");

choice=int(choice);

#The broad is considered in the form of a single dimentional array.

#One player moves 1 and other move -1.

board=[0,0,0,0,0,0,0,0,0];

if(choice==1):

print("Computer : O Vs. You : X");

player= input("Enter to play 1(st) or 2(nd) :");

player = int(player);

for i in range (0,9):

if(analyzeboard(board)!=0):

break;

if((i+player)%2==0):

CompTurn(board);

else:

ConstBoard(board);

User1Turn(board);

else:

for i in range (0,9):

if(analyzeboard(board)!=0):

break;

if((i)%2==0):

ConstBoard(board);

User1Turn(board);

else:

ConstBoard(board);

User2Turn(board);

x=analyzeboard(board);

if(x==0):

ConstBoard(board);

print("Draw!!!")

if(x==-1):

ConstBoard(board);

print("X Wins!!! Y Loose !!!")

if(x==1):

ConstBoard(board);

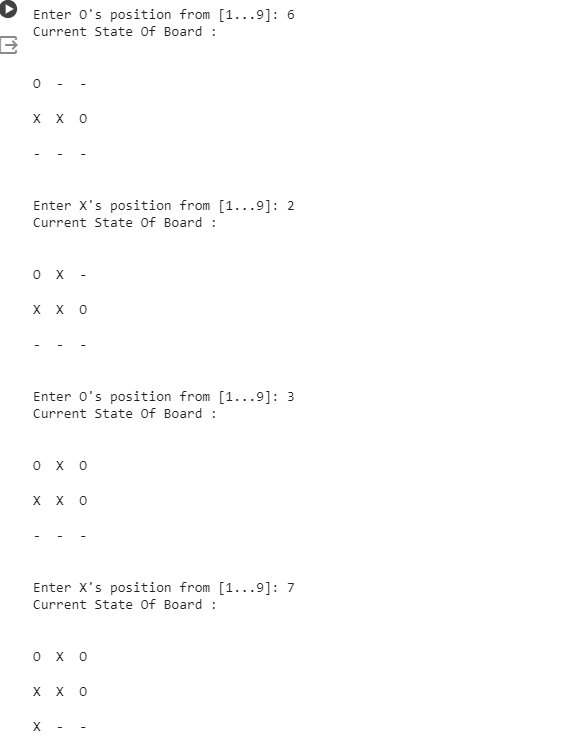
print("X Loose!!! O Wins !!!!")

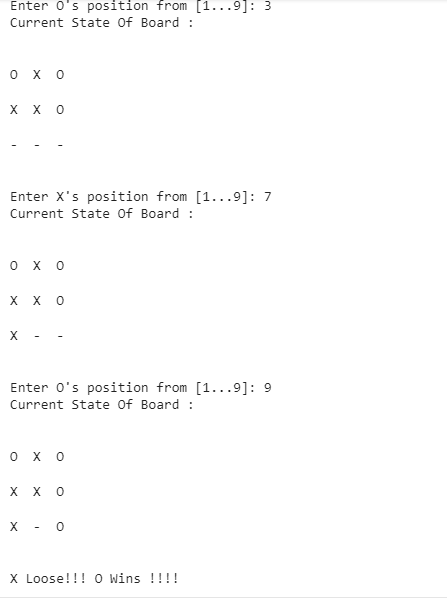
#---------------#

main()

#---------------#

Output:





2. Implement 8 puZzle using bfs

import copy

from heapq import heappush, heappop

n = 3

rows = [ 1, 0, -1, 0 ]

cols = [ 0, -1, 0, 1 ]

class priorityQueue:

def \_\_init\_\_(self):

self.heap = []

def push(self, key):

heappush(self.heap, key)

def pop(self):

return heappop(self.heap)

def empty(self):

if not self.heap:

return True

else:

return False

class nodes:

def \_\_init\_\_(self, parent, mats, empty\_tile\_posi,costs, levels):

self.parent = parent

self.mats = mats

self.empty\_tile\_posi = empty\_tile\_posi

self.costs = costs

self.levels = levels

def \_\_lt\_\_(self, nxt):

return self.costs < nxt.costs

def calculateCosts(mats, final) -> int:

count = 0

for i in range(n):

for j in range(n):

if ((mats[i][j]) and

(mats[i][j] != final[i][j])):

count += 1

return count

def newNodes(mats, empty\_tile\_posi, new\_empty\_tile\_posi,levels, parent, final) -> nodes:

new\_mats = copy.deepcopy(mats)

x1 = empty\_tile\_posi[0]

y1 = empty\_tile\_posi[1]

x2 = new\_empty\_tile\_posi[0]

y2 = new\_empty\_tile\_posi[1]

new\_mats[x1][y1], new\_mats[x2][y2] = new\_mats[x2][y2], new\_mats[x1][y1]

costs = calculateCosts(new\_mats, final)

new\_nodes = nodes(parent, new\_mats, new\_empty\_tile\_posi,costs, levels)

return new\_nodes

def printMatsrix(mats):

for i in range(n):

for j in range(n):

print("%d " % (mats[i][j]), end = " ")

print()

def isSafe(x, y):

return x >= 0 and x < n and y >= 0 and y < n

def printPath(root):

if root == None:

return

printPath(root.parent)

printMatsrix(root.mats)

print()

def solve(initial, empty\_tile\_posi, final):

pq = priorityQueue()

costs = calculateCosts(initial, final)

root = nodes(None, initial,empty\_tile\_posi, costs, 0)

pq.push(root)

while not pq.empty():

minimum = pq.pop()

if minimum.costs == 0:

printPath(minimum)

return

for i in range(n):

new\_tile\_posi = [

minimum.empty\_tile\_posi[0] + rows[i],

minimum.empty\_tile\_posi[1] + cols[i], ]

if isSafe(new\_tile\_posi[0], new\_tile\_posi[1]):

child = newNodes(minimum.mats, minimum.empty\_tile\_posi,new\_tile\_posi, minimum.levels + 1, minimum, final,)

pq.push(child)

initial = [ [ 1, 2, 3 ],

[ 5, 6, 0 ],

[ 7, 8, 4 ] ]

final = [ [ 1, 2, 3 ],

[ 5, 8, 6 ],

[ 0, 7, 4 ] ]

empty\_tile\_posi = [ 1, 2 ]

solve(initial, empty\_tile\_posi, final)

Output:

