**Aim: Write a program to implement Tic-Tac-Toe game using alpha beta search.**

**Code:**

from random import choice

from math import inf

XPLAYER = +1

OPLAYER = -1

EMPTY = 0

board = [[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY]]

def printBoard(brd):

chars = {XPLAYER: 'X', OPLAYER: 'O', EMPTY: ' '}

for x in brd:

for y in x:

ch = chars[y]

print(f'| {ch} |', end='')

print('\n' + '---------------')

print('===============')

def clearBoard(brd):

for x, row in enumerate(brd):

for y, col in enumerate(row):

brd[x][y] = EMPTY

def winningPlayer(brd, player):

winningStates = [[brd[0][0], brd[0][1], brd[0][2]],

[brd[1][0], brd[1][1], brd[1][2]],

[brd[2][0], brd[2][1], brd[2][2]],

[brd[0][0], brd[1][0], brd[2][0]],

[brd[0][1], brd[1][1], brd[2][1]],

[brd[0][2], brd[1][2], brd[2][2]],

[brd[0][0], brd[1][1], brd[2][2]],

[brd[0][2], brd[1][1], brd[2][0]]]

if [player, player, player] in winningStates:

return True

return False

def gameWon(brd):

return winningPlayer(brd, XPLAYER) or winningPlayer(brd, OPLAYER)

def printResult(brd):

if winningPlayer(brd, XPLAYER):

print('X has won! ' + '\n')

elif winningPlayer(brd, OPLAYER):

print('O\'s have won! ' + '\n')

else:

print('Draw' + '\n')

def emptyCells(brd):

emptyC = []

for x, row in enumerate(brd):

for y, col in enumerate(row):

if brd[x][y] == EMPTY:

emptyC.append([x, y])

return emptyC

def boardFull(brd):

if len(emptyCells(brd)) == 0:

return True

return False

def setMove(brd, x, y, player):

brd[x][y] = player

def playerMove(brd):

e = True

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]}

while e:

try:

move = int(input('Pick a position(1-9)'))

if move < 1 or move > 9:

print('Invalid location! ')

elif not (moves[move] in emptyCells(brd)):

print('Location filled')

else:

setMove(brd, moves[move][0], moves[move][1], XPLAYER)

printBoard(brd)

e = False

except(KeyError, ValueError):

print('Please pick a number!')

def getScore(brd):

if winningPlayer(brd, XPLAYER):

return 10

elif winningPlayer(brd, OPLAYER):

return -10

else:

return 0

def MiniMaxAB(brd, depth, alpha, beta, player):

row = -1

col = -1

if depth == 0 or gameWon(brd):

return [row, col, getScore(brd)]

else:

for cell in emptyCells(brd):

setMove(brd, cell[0], cell[1], player)

score = MiniMaxAB(brd, depth - 1, alpha, beta, -player)

if player == XPLAYER:

# X is always the max player

if score[2] > alpha:

alpha = score[2]

row = cell[0]

col = cell[1]

else:

if score[2] < beta:

beta = score[2]

row = cell[0]

col = cell[1]

setMove(brd, cell[0], cell[1], EMPTY)

if alpha >= beta:

break

if player == XPLAYER:

return [row, col, alpha]

else:

return [row, col, beta]

def AIMove(brd):

if len(emptyCells(brd)) == 9:

x = choice([0, 1, 2])

y = choice([0, 1, 2])

setMove(brd, x, y, OPLAYER)

printBoard(brd)

else:

result = MiniMaxAB(brd, len(emptyCells(brd)), -inf, inf, OPLAYER)

setMove(brd, result[0], result[1], OPLAYER)

printBoard(brd)

def makeMove(brd, player, mode):

if mode == 1:

if player == XPLAYER:

playerMove(brd)

else:

AIMove(brd)

else:

if player == XPLAYER:

AIMove(brd)

else:

AI2Move(brd)

def playerVSai():

while True:

try:

order = int(input('Would you like to go first or second? (1/2)? '))

if not (order == 1 or order == 2):

print('Please pick 1 or 2')

else:

break

except(KeyError, ValueError):

print('Enter a number')

clearBoard(board)

if order == 2:

currentPlayer = OPLAYER

else:

currentPlayer = XPLAYER

while not (boardFull(board) or gameWon(board)):

makeMove(board, currentPlayer, 1)

currentPlayer \*= -1

printResult(board)

def main():

while True:

user = input('Play?(Y/N) ')

if user.lower() == 'y':

playerVSai()

else:

print('Bye!')

exit()

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

main()

**Output:**



