**TASK 1:**

Implement Alpha beta pruning on Tic Tac Toe game decision making.

**CODE:**

import time

class Game:

    def \_\_init\_\_(self):

        self.initialize\_game()

    def initialize\_game(self):

        self.current\_state = [['.','.','.'],

                              ['.','.','.'],

                              ['.','.','.']]

        self.player\_turn = 'X'

    def draw\_board(self):

        for i in range(0, 3):

            for j in range(0, 3):

                print('{}|'.format(self.current\_state[i][j]), end=" ")

            print()

        print()

    def is\_valid(self, px, py):

        if px < 0 or px > 2 or py < 0 or py > 2:

            return False

        elif self.current\_state[px][py] != '.':

            return False

        else:

            return True

    def is\_end(self):

        for i in range(0, 3):

            if (self.current\_state[0][i] != '.' and

                self.current\_state[0][i] == self.current\_state[1][i] and

                self.current\_state[1][i] == self.current\_state[2][i]):

                return self.current\_state[0][i]

        for i in range(0, 3):

            if (self.current\_state[i] == ['X', 'X', 'X']):

                return 'X'

            elif (self.current\_state[i] == ['O', 'O', 'O']):

                return 'O'

        if (self.current\_state[0][0] != '.' and

            self.current\_state[0][0] == self.current\_state[1][1] and

            self.current\_state[0][0] == self.current\_state[2][2]):

            return self.current\_state[0][0]

        if (self.current\_state[0][2] != '.' and

            self.current\_state[0][2] == self.current\_state[1][1] and

            self.current\_state[0][2] == self.current\_state[2][0]):

            return self.current\_state[0][2]

        for i in range(0, 3):

            for j in range(0, 3):

                if (self.current\_state[i][j] == '.'):

                    return None

        return '.'

    def max(self):

        maxv = -2

        px = None

        py = None

        result = self.is\_end()

        if result == 'X':

            return (-1, 0, 0)

        elif result == 'O':

            return (1, 0, 0)

        elif result == '.':

            return (0, 0, 0)

        for i in range(0, 3):

            for j in range(0, 3):

                if self.current\_state[i][j] == '.':

                    self.current\_state[i][j] = 'O'

                    (m, min\_i, min\_j) = self.min()

                    if m > maxv:

                        maxv = m

                        px = i

                        py = j

                    self.current\_state[i][j] = '.'

        return (maxv, px, py)

    def min(self):

        minv = 2

        qx = None

        qy = None

        result = self.is\_end()

        if result == 'X':

            return (-1, 0, 0)

        elif result == 'O':

            return (1, 0, 0)

        elif result == '.':

            return (0, 0, 0)

        for i in range(0, 3):

            for j in range(0, 3):

                if self.current\_state[i][j] == '.':

                    self.current\_state[i][j] = 'X'

                    (m, max\_i, max\_j) = self.max()

                    if m < minv:

                        minv = m

                        qx = i

                        qy = j

                    self.current\_state[i][j] = '.'

        return (minv, qx, qy)

    def play(self):

        while True:

            self.draw\_board()

            self.result = self.is\_end()

            if self.result != None:

                if self.result == 'X':

                    print('The winner is X!')

                elif self.result == 'O':

                    print('The winner is O!')

                elif self.result == '.':

                    print("It's a tie!")

                self.initialize\_game()

                return

            if self.player\_turn == 'X':

                while True:

                    start = time.time()

                    (m, qx, qy) = self.min()

                    end = time.time()

                    print('Evaluation time: {}s'.format(round(end - start, 7)))

                    print('Recommended move: X = {}, Y = {}'.format(qx, qy))

                    px = int(input('Insert the X coordinate: '))

                    py = int(input('Insert the Y coordinate: '))

                    (qx, qy) = (px, py)

                    if self.is\_valid(px, py):

                        self.current\_state[px][py] = 'X'

                        self.player\_turn = 'O'

                        break

                    else:

                        print('The move is not valid! Try again.')

            else:

                (m, px, py) = self.max()

                self.current\_state[px][py] = 'O'

                self.player\_turn = 'X'

def main():

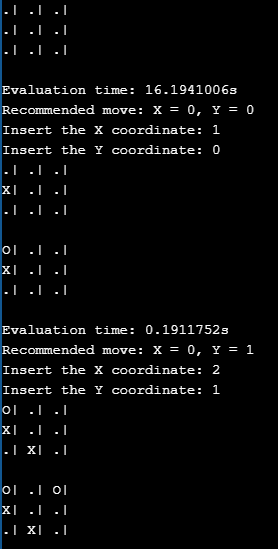
    g = Game()

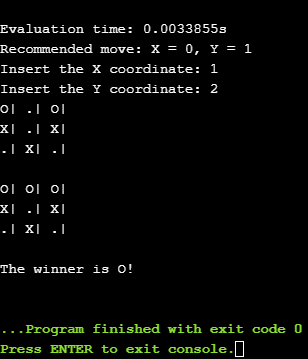
    g.play()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**OUTPUT:**

****

****