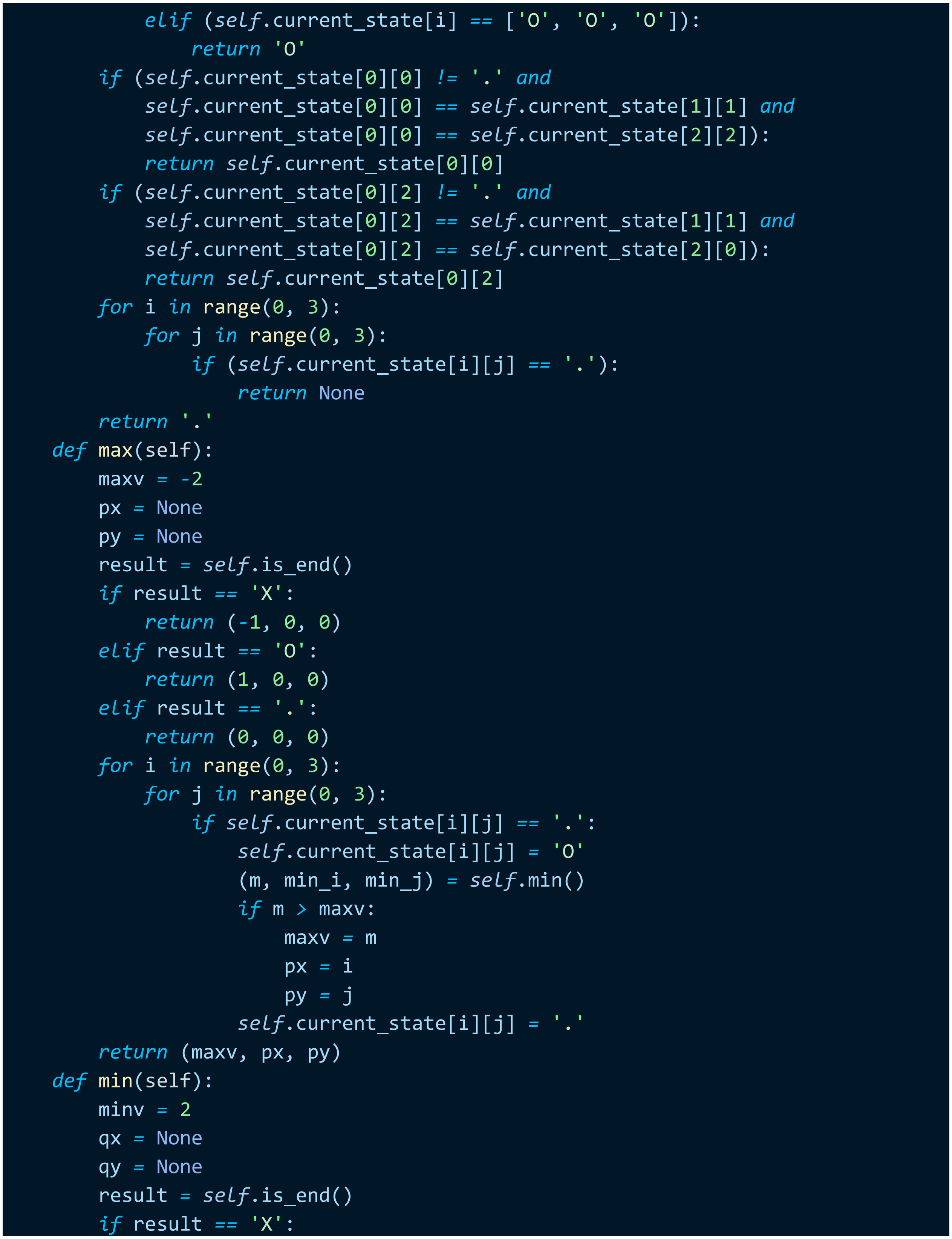
TASK 1:

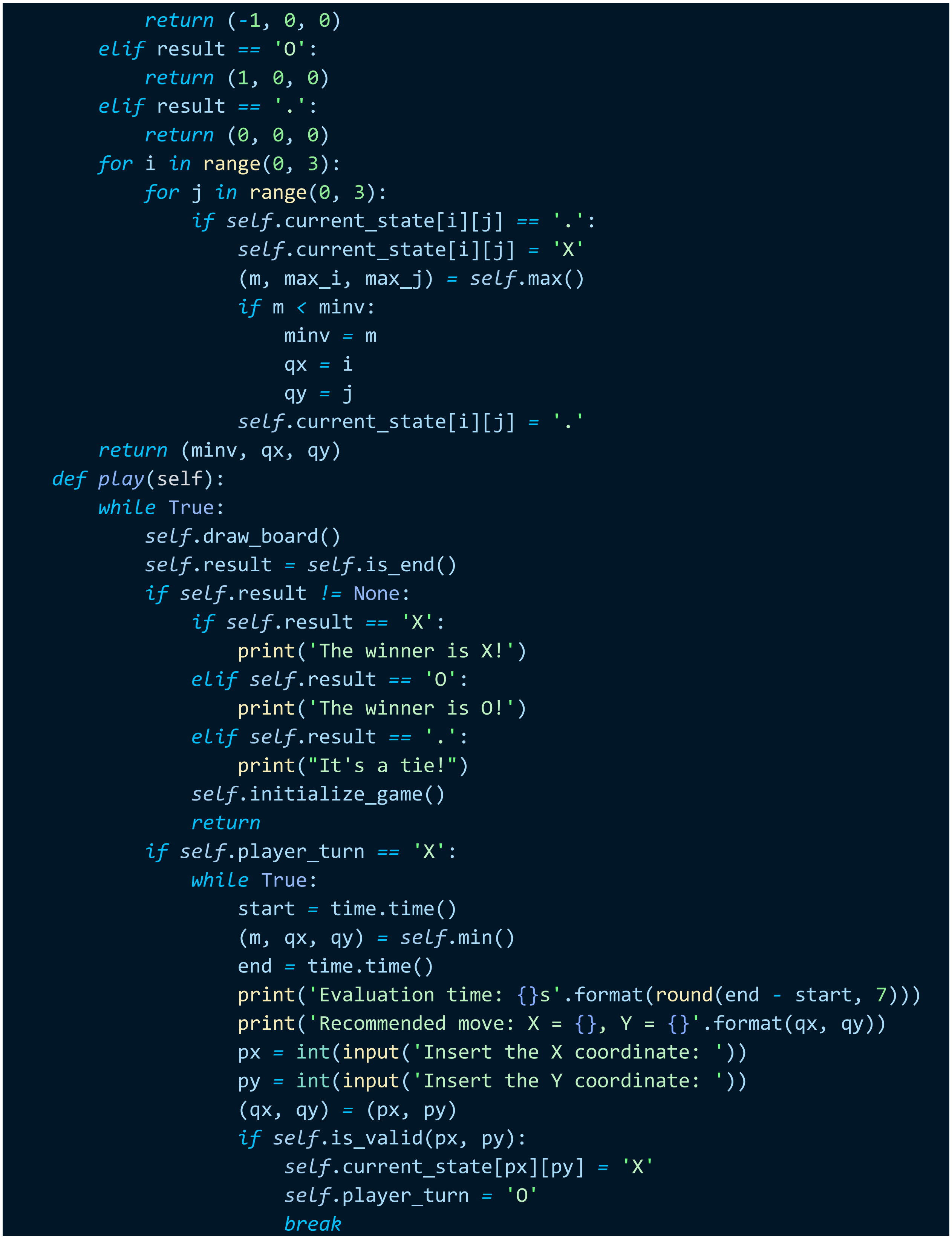
Implement Alpha beta pruning on Tic Tac Toe game decision making.

CODE:

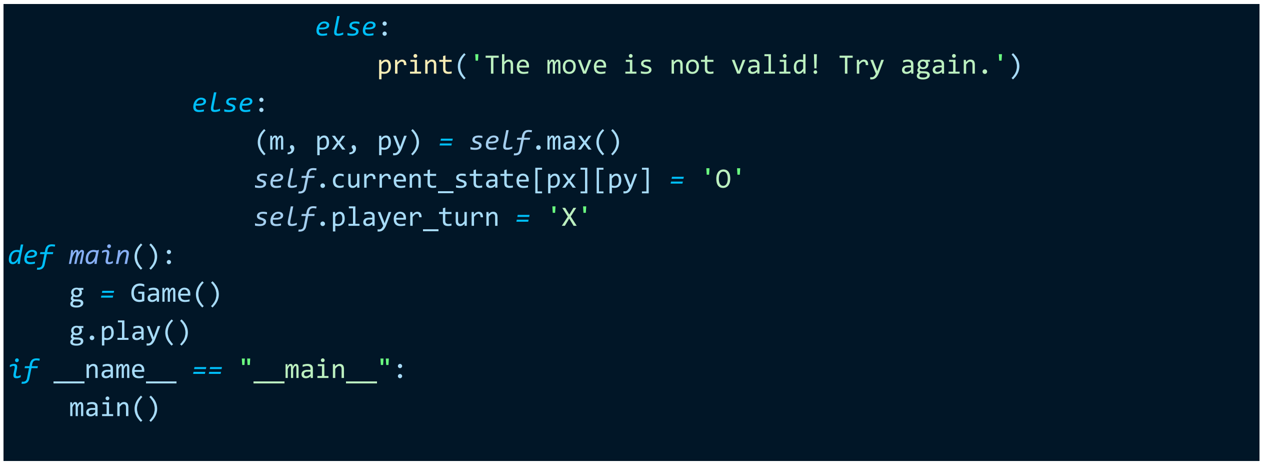
```
# Owned
__author__ = "Qaiser Abbas"
__copyright__ = "Copyright 2020, Artificial Intelligence lab-12"
__email__ = "qaiserabbas889@yahoo.com"
#=================================================================
# {code}
import time
class Game:
    def __init__(self):
        self.initialize_game()

    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':
```

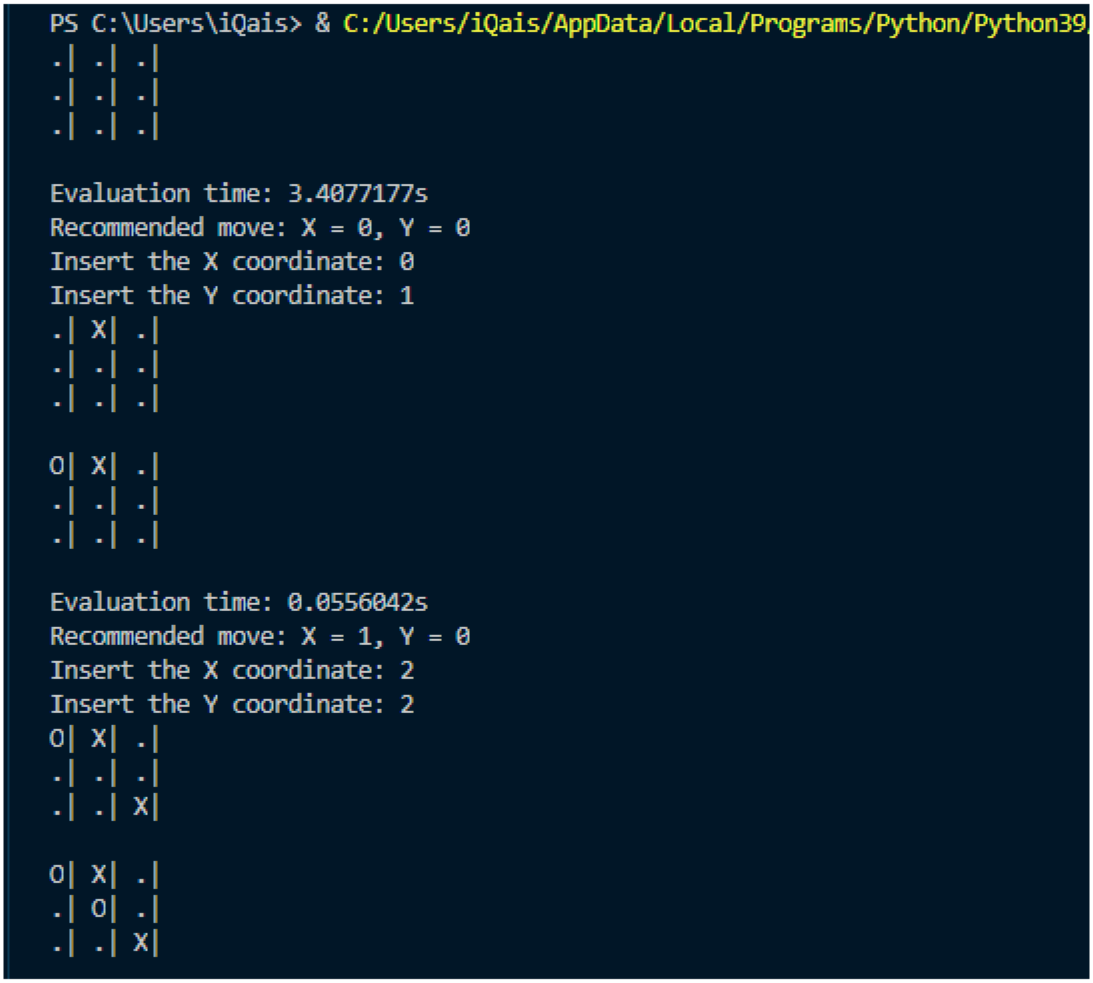


```
            return (-1, 0, 0)
        elif result == 'O':
            return (1, 0, 0)
        elif result == '.':
            return (0, 0, 0)
        for i in range(0, 3):
            for j in range(0, 3):
                if self.current_state[i][j] == '.':
                    self.current_state[i][j] = '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
```

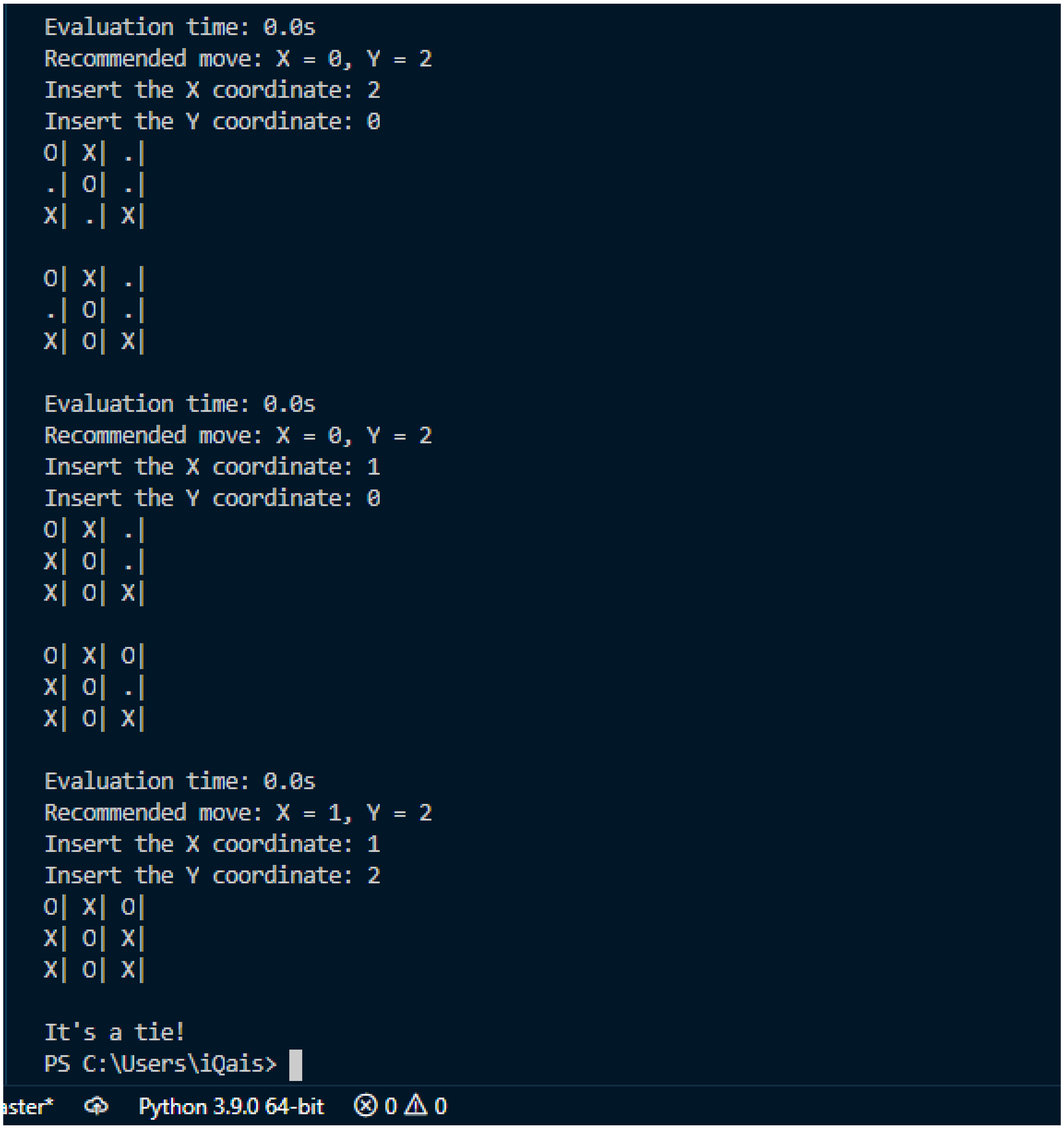


```
                    else:
                        print('The move is not valid! Try again.')
            else:
                (m, px, py) = self.max()
                self.current_state[px][py] = 'O'
                self.player_turn = 'X'
def main():
    g = Game()
    g.play()
if __name__ == "__main__":
    main()
```

OUTPUT:



```
PS C:\Users\iQais> & C:/Users/iQais/AppData/Local/Programs/Python/Python39
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 3.4077177s
Recommended move: X = 0, Y = 0
Insert the X coordinate: 0
Insert the Y coordinate: 1
.| X| .|
.| .| .|
.| .| .|

O| X| .|
.| .| .|
.| .| .|

Evaluation time: 0.0556042s
Recommended move: X = 1, Y = 0
Insert the X coordinate: 2
Insert the Y coordinate: 2
O| X| .|
.| .| .|
.| .| X|

O| X| .|
.| O| .|
.| .| X|
```



```
Evaluation time: 0.0s
Recommended move: X = 0, Y = 2
Insert the X coordinate: 2
Insert the Y coordinate: 0
O| X| .|
.| O| .|
X| .| X|

O| X| .|
.| O| .|
X| O| X|

Evaluation time: 0.0s
Recommended move: X = 0, Y = 2
Insert the X coordinate: 1
Insert the Y coordinate: 0
O| X| .|
X| O| .|
X| O| X|

O| X| O|
X| O| .|
X| O| X|

Evaluation time: 0.0s
Recommended move: X = 1, Y = 2
Insert the X coordinate: 1
Insert the Y coordinate: 2
O| X| O|
X| O| X|
X| O| X|

It's a tie!
PS C:\Users\iQais>
```