Artificial Intelligence

CSL 411

Lab Journal 09

Tauheed Butt
01-134191-068
BSCS-6B

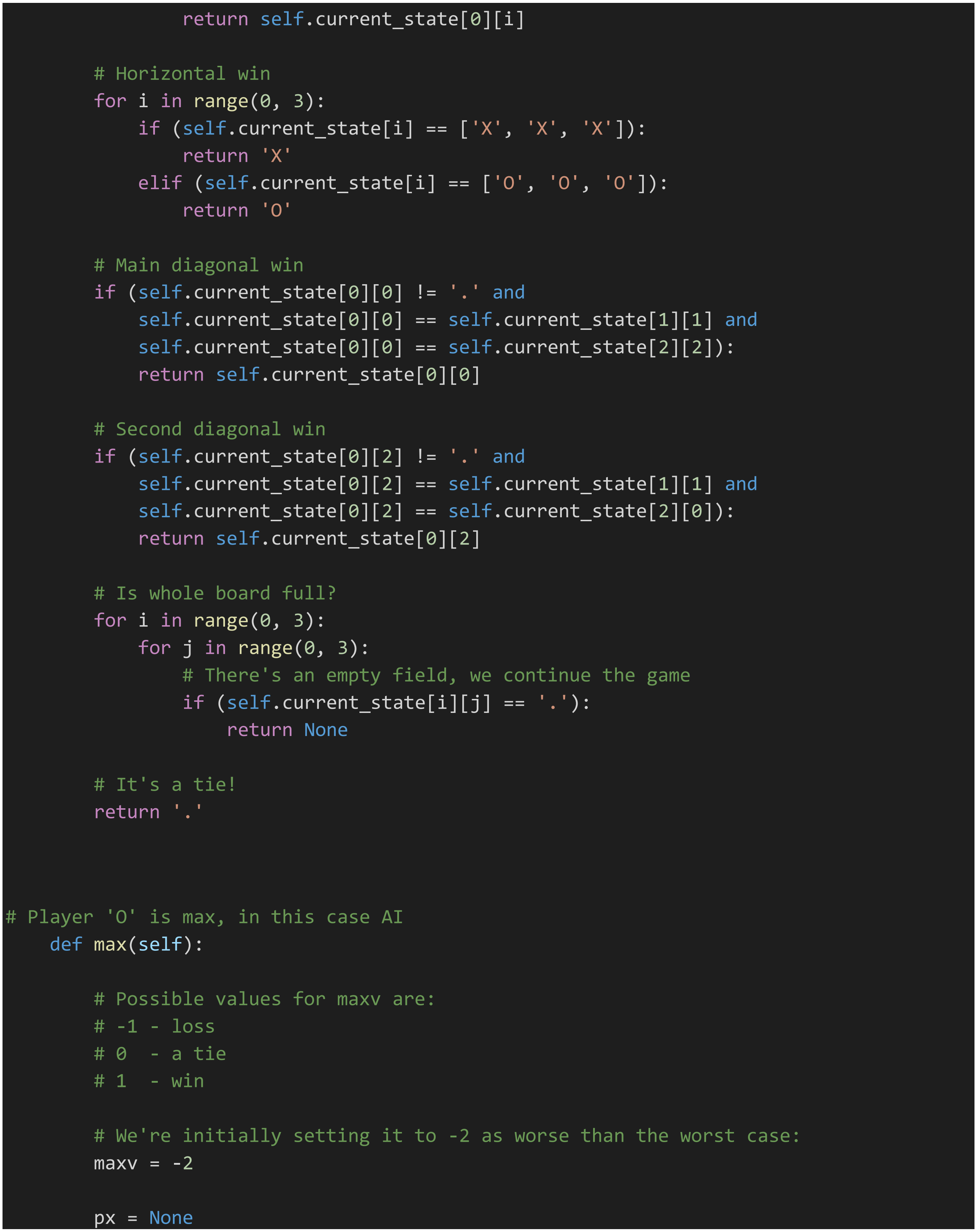
Department of Computer Science
BAHRIA UNIVERSITY, ISLAMABAD

Task #1:

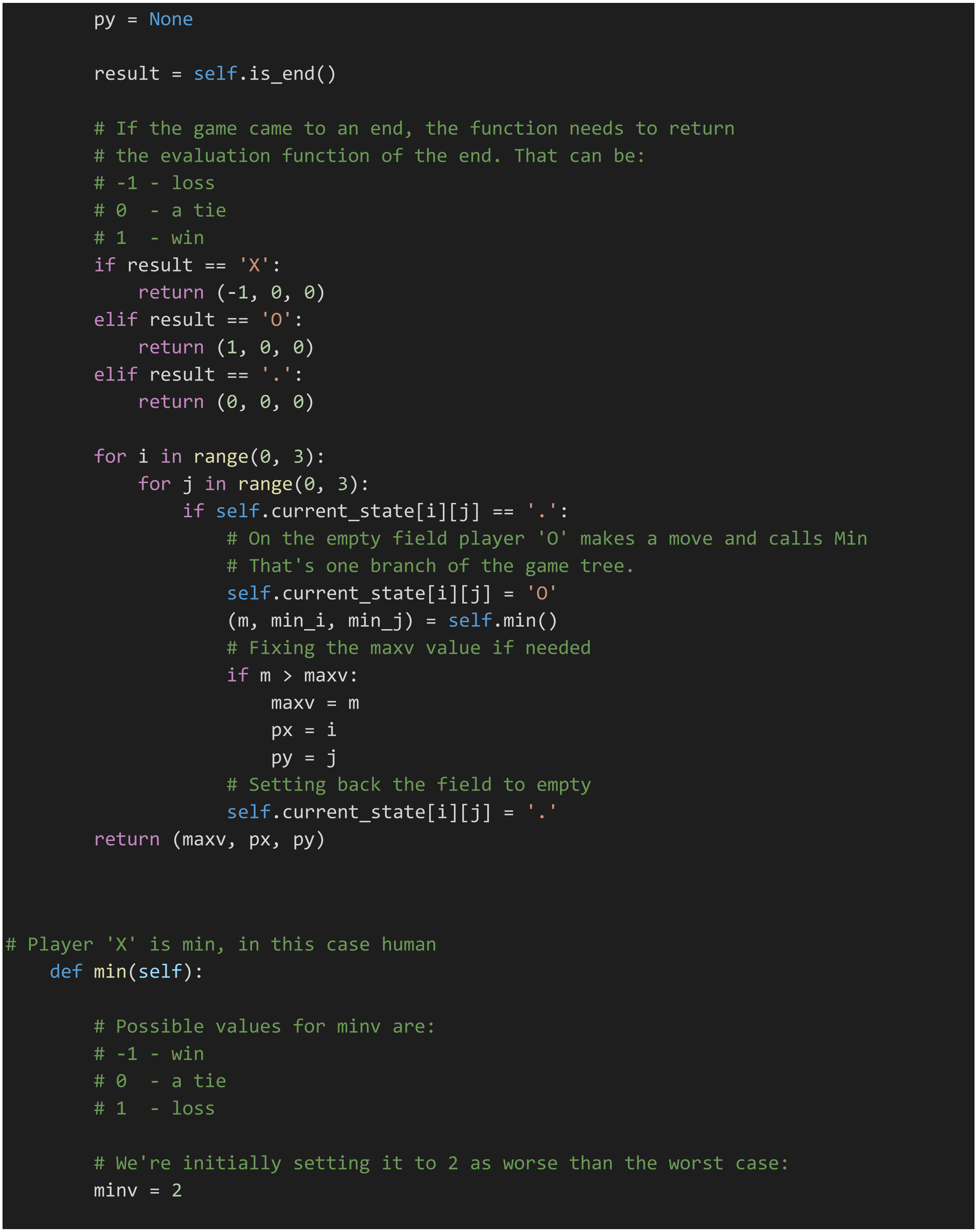
Implement Minmax and alpha-beta pruning algorithm.

Procedure/Program:

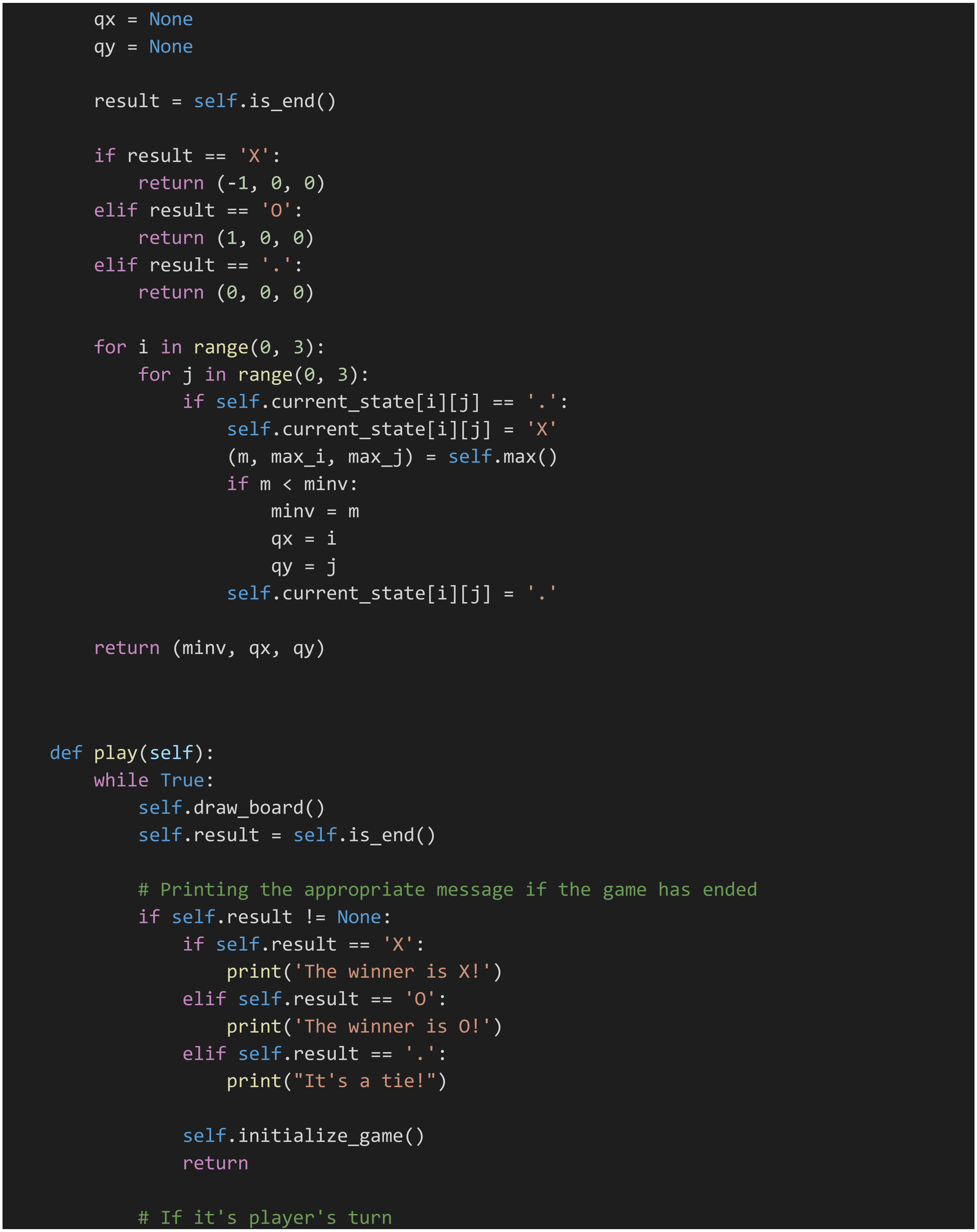
```
import time

class Game:
    def __init__(self):
        self.initialize_game()

    def initialize_game(self):
        self.current_state = [['.','.','.'],
                              ['.','.','.'],
                              ['.','.','.']]

        # Player X always plays first
        self.player_turn = 'X'

    def draw_board(self):
        for i in range(0, 3):
            for j in range(0, 3):
                print(f'{self.current_state[i][j]}|', end=' ')
            print()
        print()

    # Determines if the made move is a legal move
    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


# Checks if the game has ended and returns the winner in each case
    def is_end(self):
        # Vertical win
        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]):
```



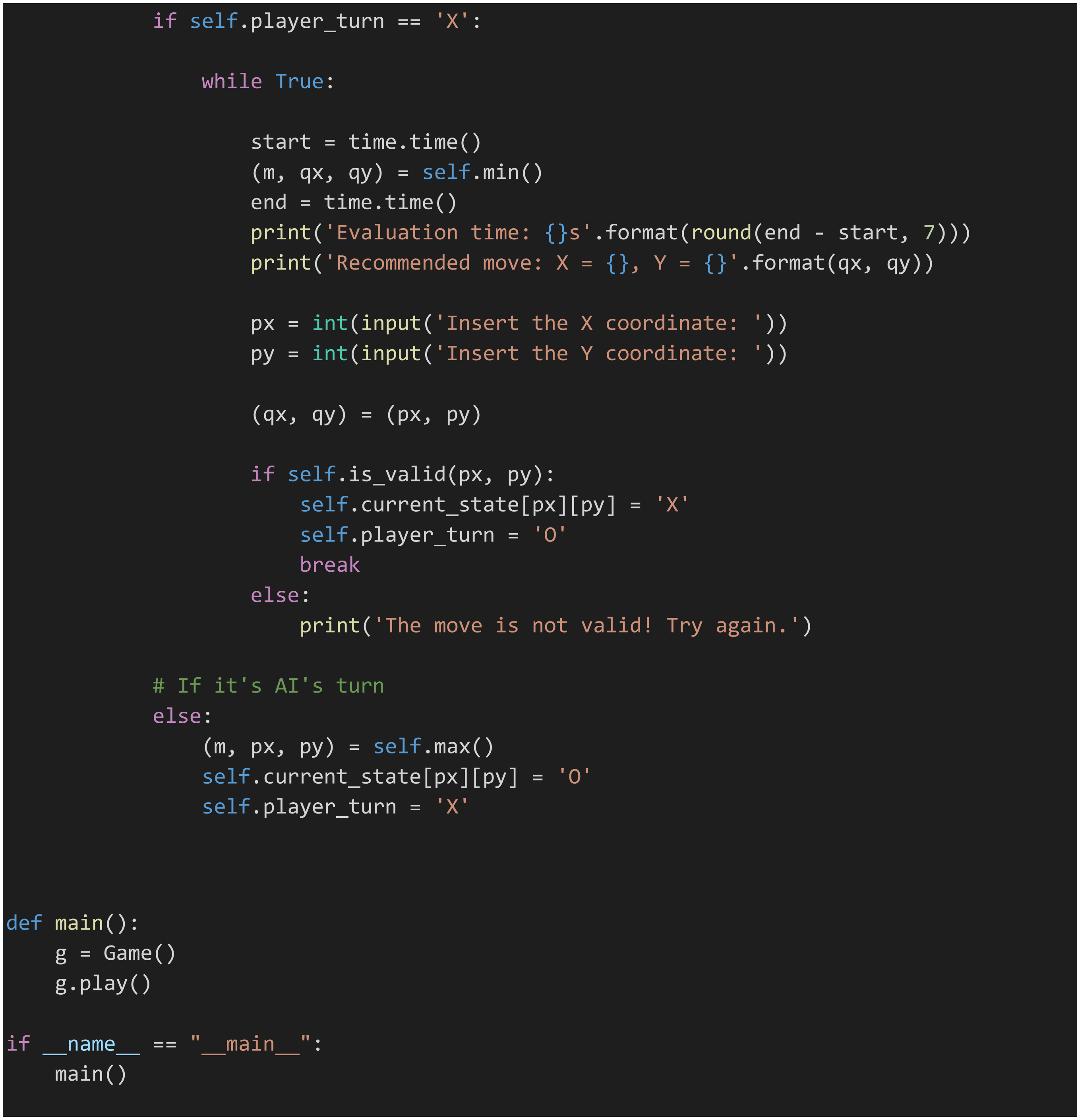
```
                return self.current_state[0][i]

        # Horizontal win
        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'

        # Main diagonal win
        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]

        # Second diagonal win
        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]

        # Is whole board full?
        for i in range(0, 3):
            for j in range(0, 3):
                # There's an empty field, we continue the game
                if (self.current_state[i][j] == '.'):
                    return None

        # It's a tie!
        return '.'



# Player 'O' is max, in this case AI
    def max(self):

        # Possible values for maxv are:
        # -1 - loss
        # 0  - a tie
        # 1  - win

        # We're initially setting it to -2 as worse than the worst case:
        maxv = -2

        px = None
```



```
        py = None

        result = self.is_end()

        # If the game came to an end, the function needs to return
        # the evaluation function of the end. That can be:
        # -1 - loss
        # 0  - a tie
        # 1  - win
        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] == '.':
                    # On the empty field player 'O' makes a move and calls Min
                    # That's one branch of the game tree.
                    self.current_state[i][j] = 'O'
                    (m, min_i, min_j) = self.min()
                    # Fixing the maxv value if needed
                    if m > maxv:
                        maxv = m
                        px = i
                        py = j
                    # Setting back the field to empty
                    self.current_state[i][j] = '.'
        return (maxv, px, py)



# Player 'X' is min, in this case human
    def min(self):

        # Possible values for minv are:
        # -1 - win
        # 0  - a tie
        # 1  - loss

        # We're initially setting it to 2 as worse than the worst case:
        minv = 2
```

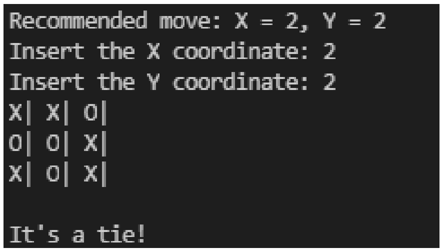


```
        qx = None
        qy = None

        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()

            # Printing the appropriate message if the game has ended
            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 it's player's turn
```



```
            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.')

            # If it's AI's turn
            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()
```

Result/Output:



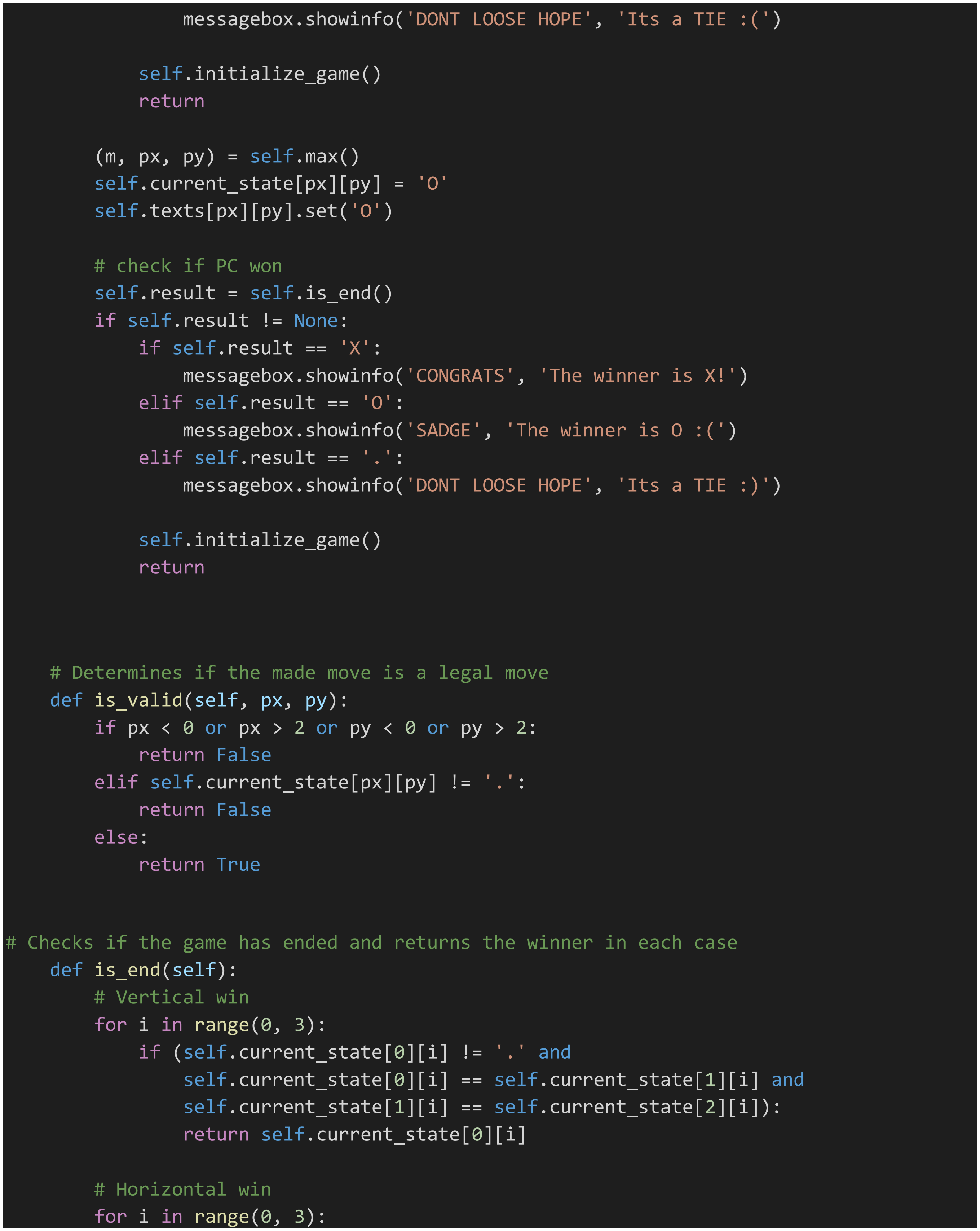
```
Recommended move: X = 2, Y = 2
Insert the X coordinate: 2
Insert the Y coordinate: 2
X| X| O|
O| O| X|
X| O| X|

It's a tie!
```

Task # 2:

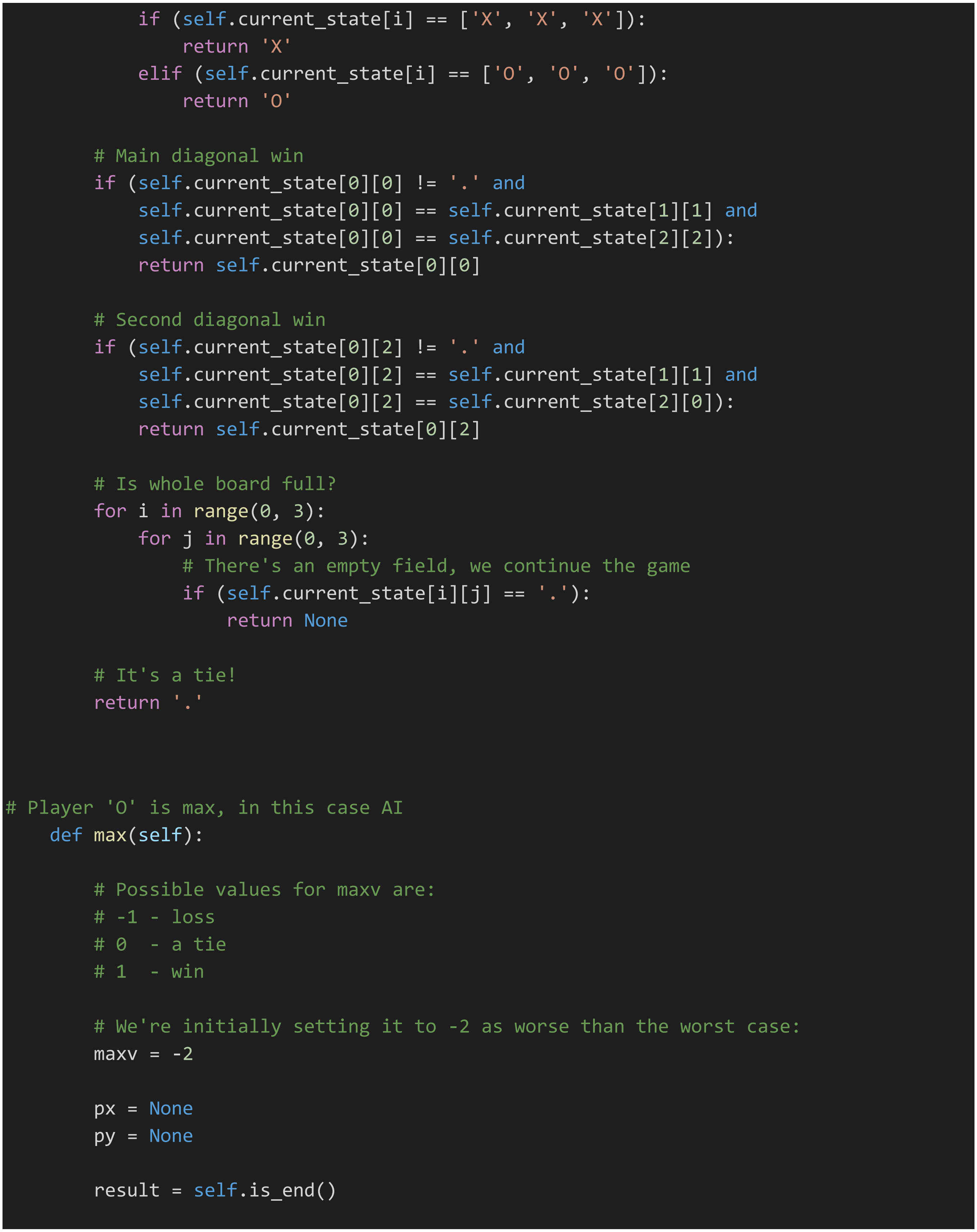
Implementation of Tic Tac Toe game in GUI using TKinter by using MiniMax and Alpha-Beta pruning algorithms.

Procedure/Program:

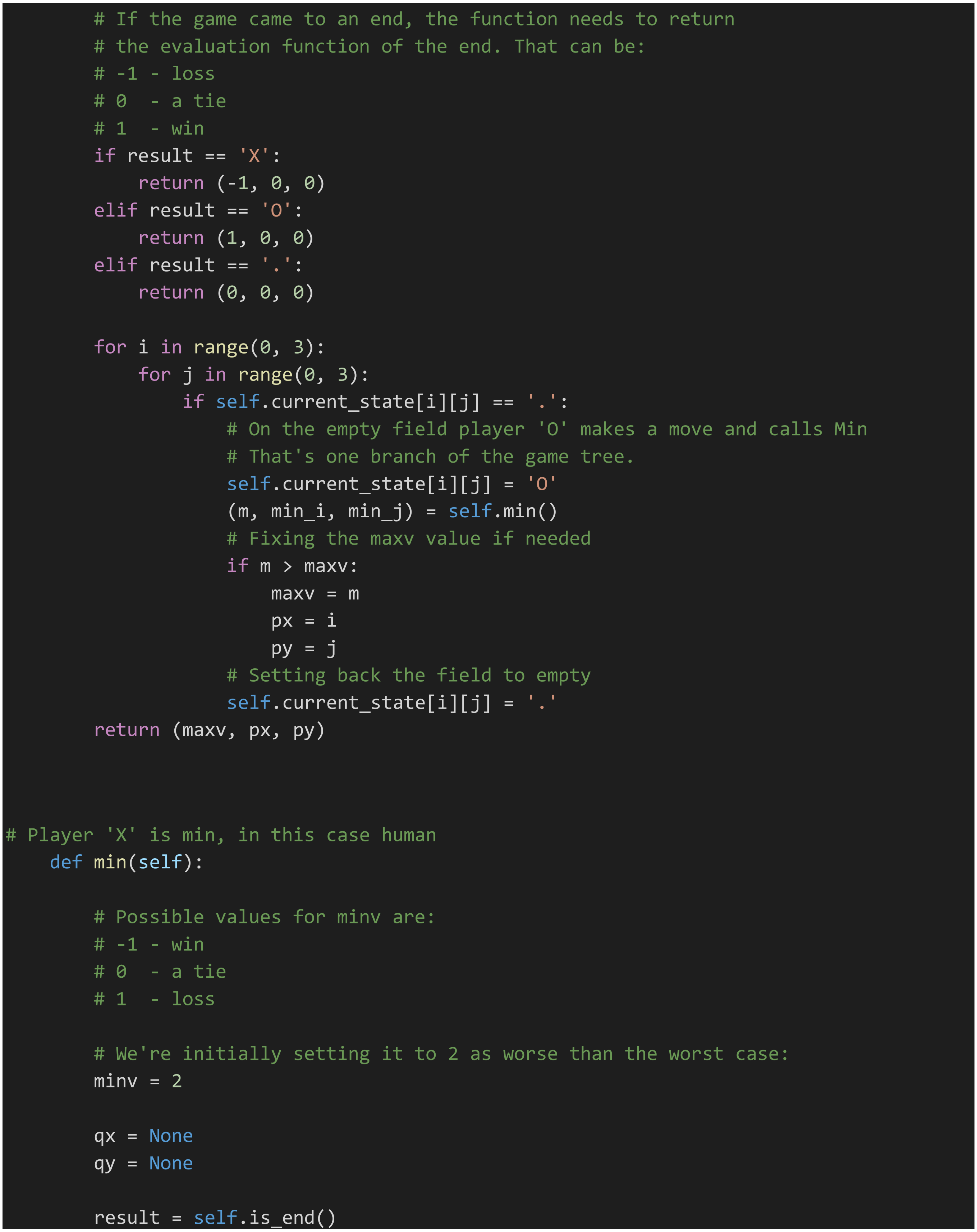
```
import time
from tkinter import *
from tkinter import messagebox

class Game:
    def __init__(self):

        self.root = Tk()
        self.root.title("TicTacToe")
        self.texts = [
            [StringVar(), StringVar(), StringVar()],
            [StringVar(), StringVar(), StringVar()],
            [StringVar(), StringVar(), StringVar()]
        ]

        self.initialize_game()

    def initialize_game(self):
        self.current_state = [['.','.','.'],
                              ['.','.','.'],
                              ['.','.','.']]

        # initialize button texts
        for i in range(len(self.texts)):
            for j in range(len(self.texts[i])):
                self.texts[i][j].set(f' ')

        # initialize input buttons
        self.buttons = [
            [
                Button(self.root, textvariable=self.texts[0][0], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(0,0)),
                Button(self.root, textvariable=self.texts[0][1], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(0,1)),
                Button(self.root, textvariable=self.texts[0][2], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(0,2))
            ],
            [
                Button(self.root, textvariable=self.texts[1][0], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(1,0)),
                Button(self.root, textvariable=self.texts[1][1], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(1,1)),
                Button(self.root, textvariable=self.texts[1][2], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(1,2))
            ],
            [
                Button(self.root, textvariable=self.texts[2][0], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(2,0)),
                Button(self.root, textvariable=self.texts[2][1], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(2,1)),
                Button(self.root, textvariable=self.texts[2][2], font = ("Helvetica",
30),height=5, width=10, command = lambda: self.btnClick(2,2))
            ]
        ]

    def draw_board(self):
        for i in range(3):
            for j in range(3):
                self.buttons[i][j].grid(row=i, column=j)

        #create resposnive grid
        for i in range(3):
            self.root.grid_rowconfigure(i,  weight =1)
            self.root.grid_columnconfigure(i,  weight =1)

        self.root.mainloop()

    def btnClick(self, i, j):
        if not self.is_valid(i, j):
            messagebox.showinfo('ERROR', 'Cant Place X, Already Filled')
            return

        self.current_state[i][j] = 'X'
        self.texts[i][j].set('X')

        # check if user won
        self.result = self.is_end()
        if self.result != None:
            if self.result == 'X':
                messagebox.showinfo('CONGRATS', 'The winner is X!')
            elif self.result == 'O':
                messagebox.showinfo('SADGE', 'The winner is O!')
            elif self.result == '.':
```



```
                messagebox.showinfo('DONT LOOSE HOPE', 'Its a TIE :(')

            self.initialize_game()
            return

        (m, px, py) = self.max()
        self.current_state[px][py] = 'O'
        self.texts[px][py].set('O')

        # check if PC won
        self.result = self.is_end()
        if self.result != None:
            if self.result == 'X':
                messagebox.showinfo('CONGRATS', 'The winner is X!')
            elif self.result == 'O':
                messagebox.showinfo('SADGE', 'The winner is O :(')
            elif self.result == '.':
                messagebox.showinfo('DONT LOOSE HOPE', 'Its a TIE :)')

            self.initialize_game()
            return



    # Determines if the made move is a legal move
    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


# Checks if the game has ended and returns the winner in each case
    def is_end(self):
        # Vertical win
        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]

        # Horizontal win
        for i in range(0, 3):
```

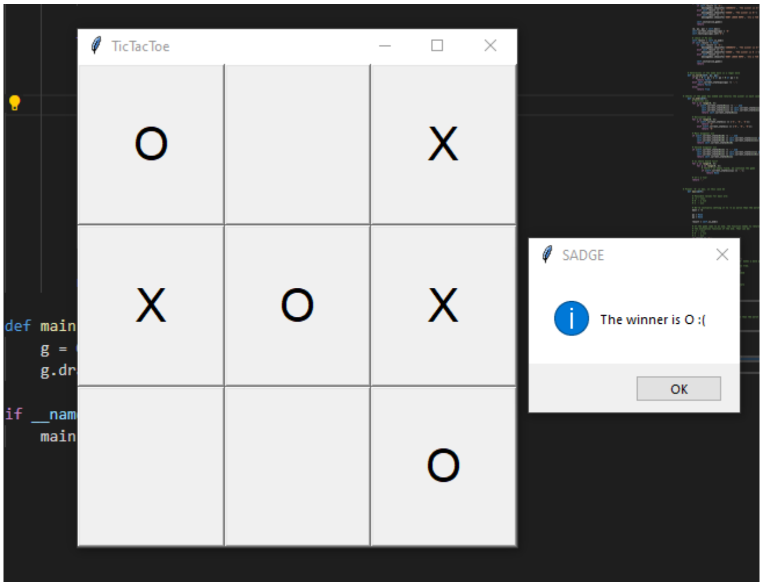


```
            if (self.current_state[i] == ['X', 'X', 'X']):
                return 'X'
            elif (self.current_state[i] == ['O', 'O', 'O']):
                return 'O'

        # Main diagonal win
        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]

        # Second diagonal win
        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]

        # Is whole board full?
        for i in range(0, 3):
            for j in range(0, 3):
                # There's an empty field, we continue the game
                if (self.current_state[i][j] == '.'):
                    return None

        # It's a tie!
        return '.'



# Player 'O' is max, in this case AI
    def max(self):

        # Possible values for maxv are:
        # -1 - loss
        # 0  - a tie
        # 1  - win

        # We're initially setting it to -2 as worse than the worst case:
        maxv = -2

        px = None
        py = None

        result = self.is_end()
```



```
        # If the game came to an end, the function needs to return
        # the evaluation function of the end. That can be:
        # -1 - loss
        # 0  - a tie
        # 1  - win
        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] == '.':
                    # On the empty field player 'O' makes a move and calls Min
                    # That's one branch of the game tree.
                    self.current_state[i][j] = 'O'
                    (m, min_i, min_j) = self.min()
                    # Fixing the maxv value if needed
                    if m > maxv:
                        maxv = m
                        px = i
                        py = j
                    # Setting back the field to empty
                    self.current_state[i][j] = '.'
        return (maxv, px, py)



# Player 'X' is min, in this case human
    def min(self):

        # Possible values for minv are:
        # -1 - win
        # 0  - a tie
        # 1  - loss

        # We're initially setting it to 2 as worse than the worst case:
        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 main():
    g = Game()
    g.draw_board()

if __name__ == "__main__":
    main()
```

Result/Output:



O		X
X	O	X
		O

The winner is O :(