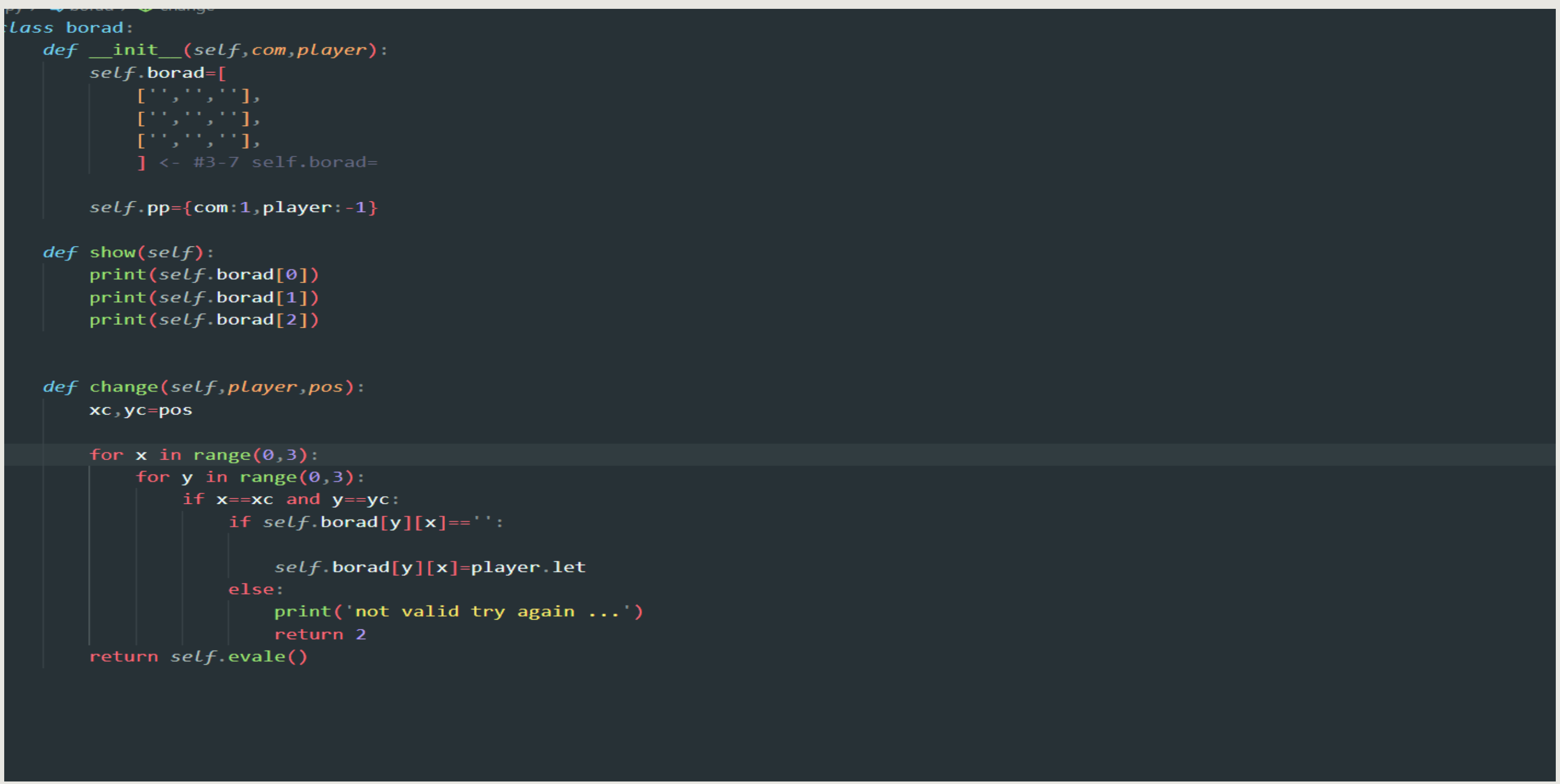
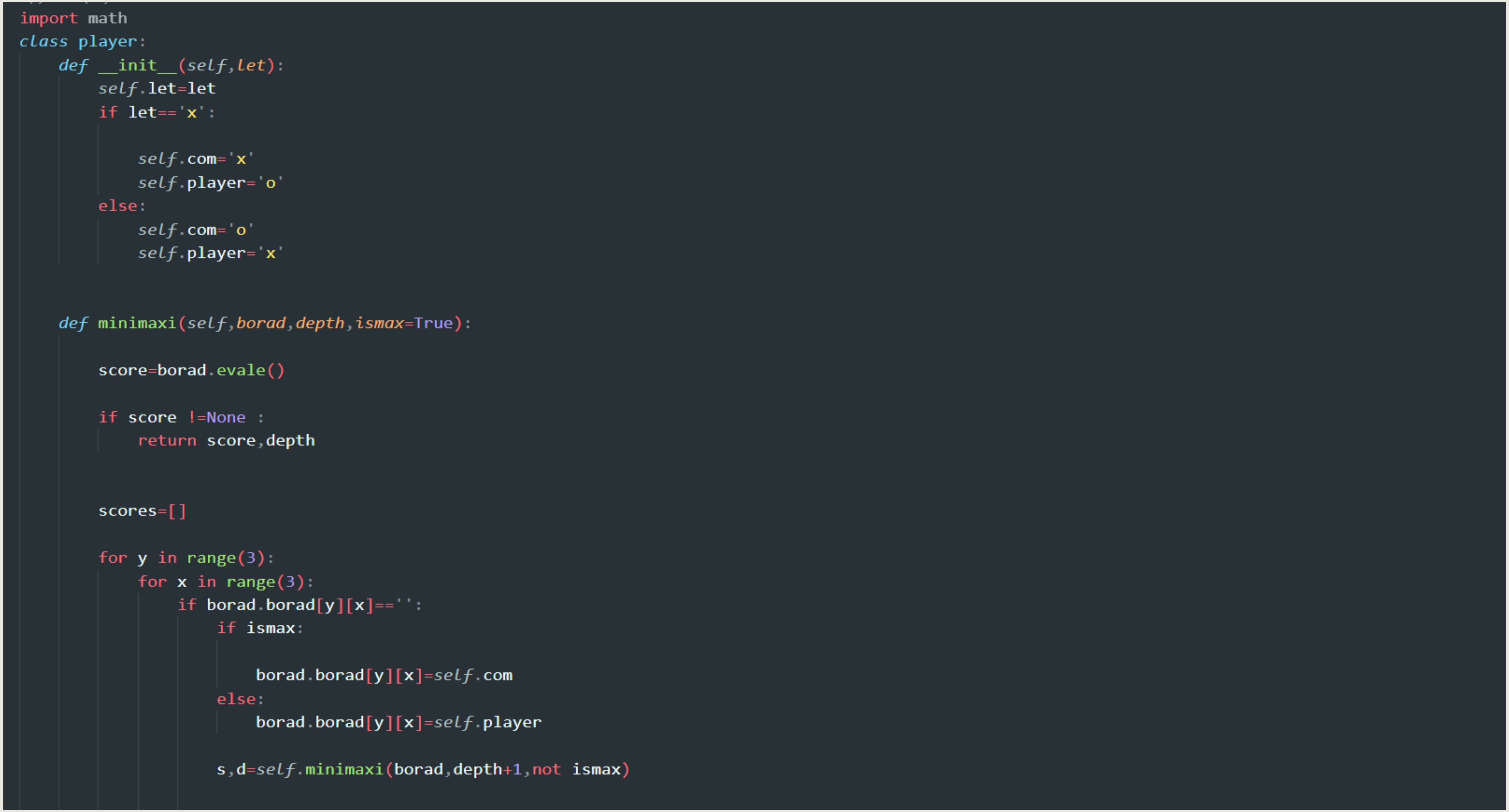
x/o game with minimax

Classes



```python
class borad:
    def __init__(self,com,player):
        self.borad=[
            ['','',''],
            ['','',''],
            ['','',''],
            ] <- #3-7 self.borad=

        self.pp={com:1,player:-1}

    def show(self):
        print(self.borad[0])
        print(self.borad[1])
        print(self.borad[2])


    def change(self,player,pos):
        xc,yc=pos

        for x in range(0,3):
            for y in range(0,3):
                if x==xc and y==yc:
                    if self.borad[y][x]=='':

                        self.borad[y][x]=player.let
                    else:
                        print('not valid try again ...')
                        return 2
        return self.evale()
```



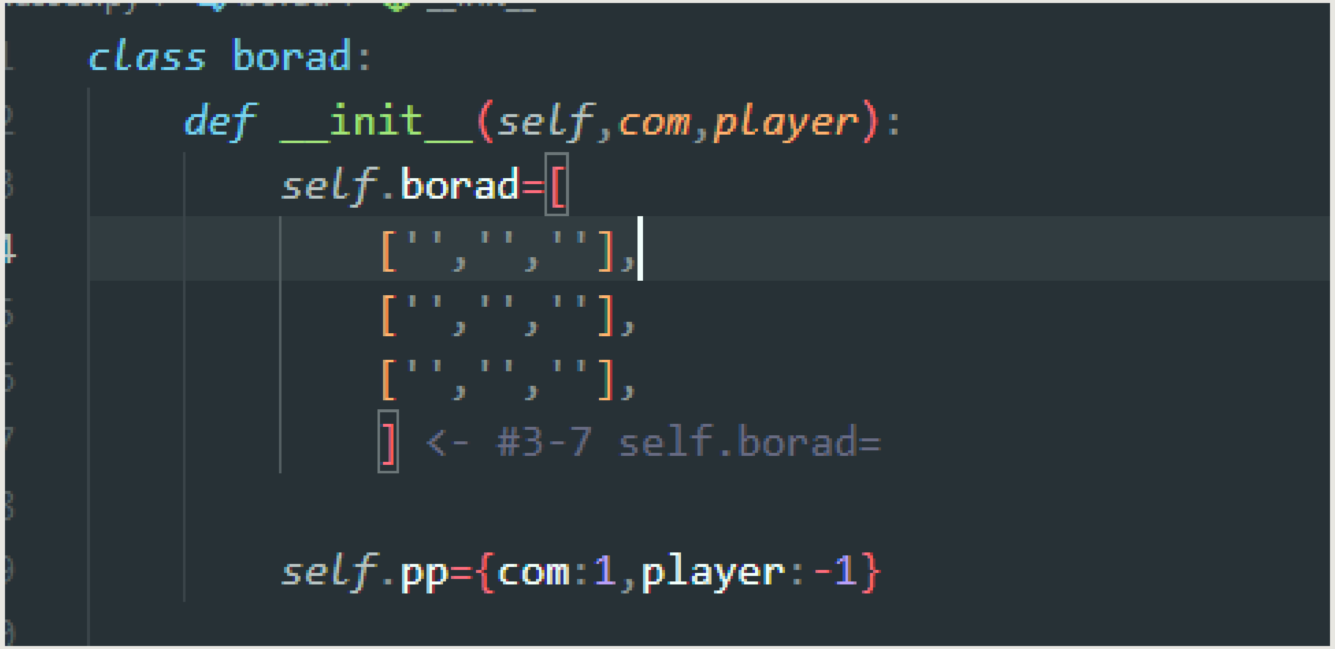
```
import math
class player:
    def __init__(self,let):
        self.let=let
        if let=='x':

            self.com='x'
            self.player='o'
        else:
            self.com='o'
            self.player='x'


    def minimaxi(self,borad,depth,ismax=True):

        score=borad.evale()

        if score !=None :
            return score,depth


        scores=[]

        for y in range(3):
            for x in range(3):
                if borad.borad[y][x]=='':
                    if ismax:

                        borad.borad[y][x]=self.com
                    else:
                        borad.borad[y][x]=self.player

                    s,d=self.minimaxi(borad,depth+1,not ismax)
```

Methods of class Borad

def __init__

Constructor for the class it

initialize the x/o board

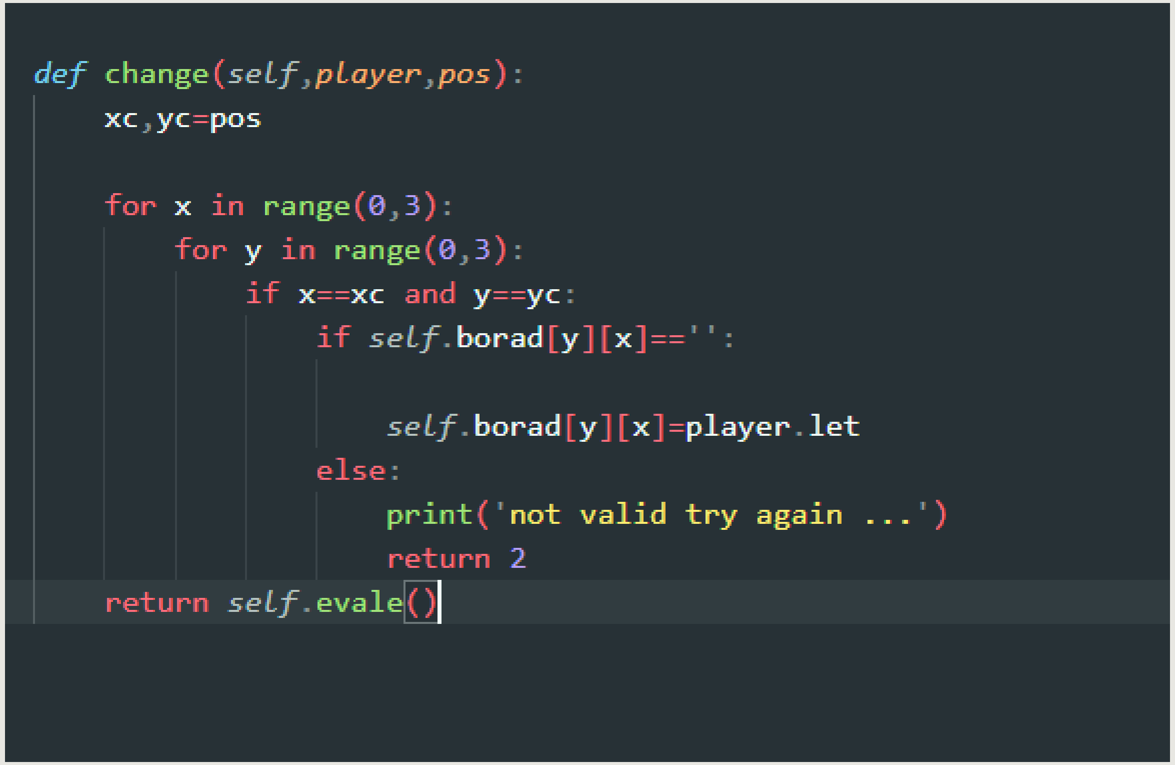


```
class borad:
    def __init__(self,com,player):
        self.borad=[
            ['','',''],
            ['','',''],
            ['','',''],
            ] <- #3-7 self.borad=

        self.pp={com:1,player:-1}
```

Shows the board to the pLayer

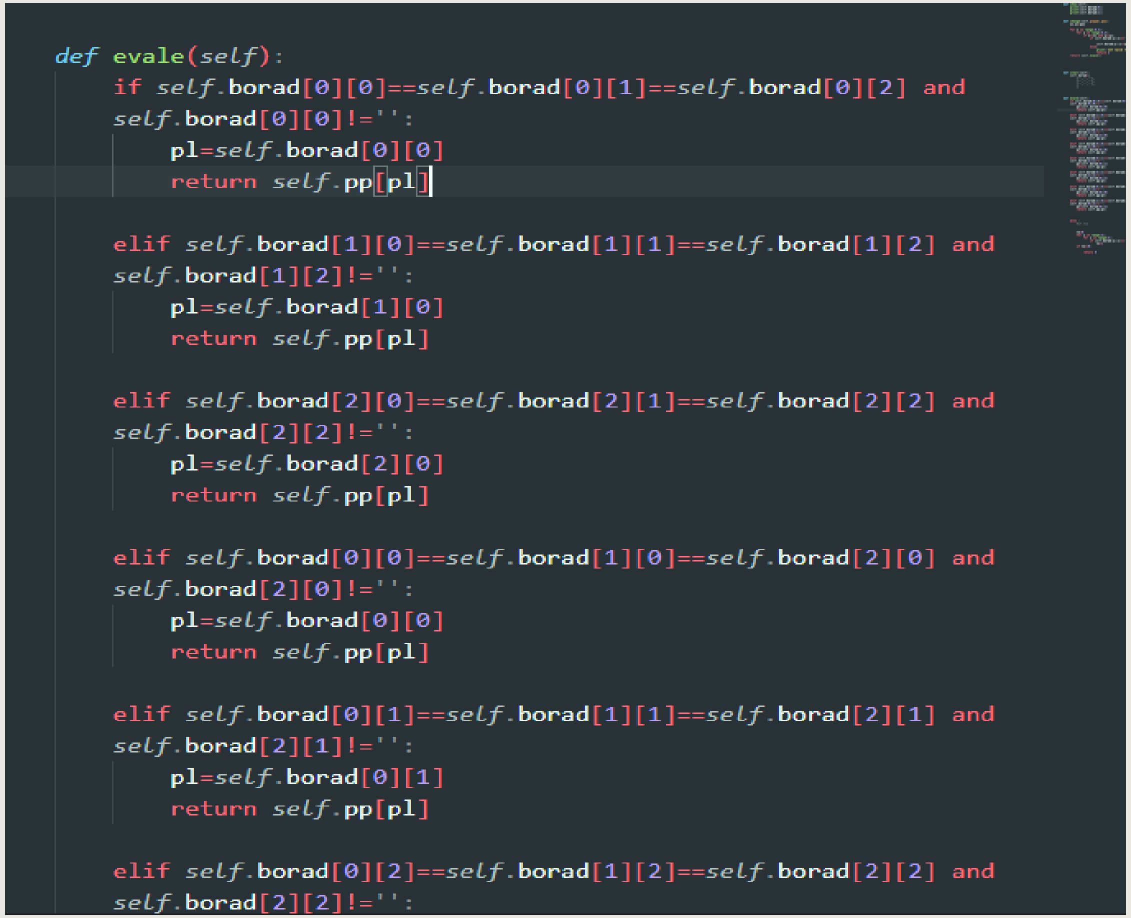
```
def show(self):
        print(self.borad[0])
        print(self.borad[1])
        print(self.borad[2])
```

Changes the player who plays and evaluates if there is a winner after last move or not



```
def change(self,player,pos):
    xc,yc=pos

    for x in range(0,3):
        for y in range(0,3):
            if x==xc and y==yc:
                if self.borad[y][x]=='':

                    self.borad[y][x]=player.let
                else:
                    print('not valid try again ...')
                    return 2
    return self.evale()
```

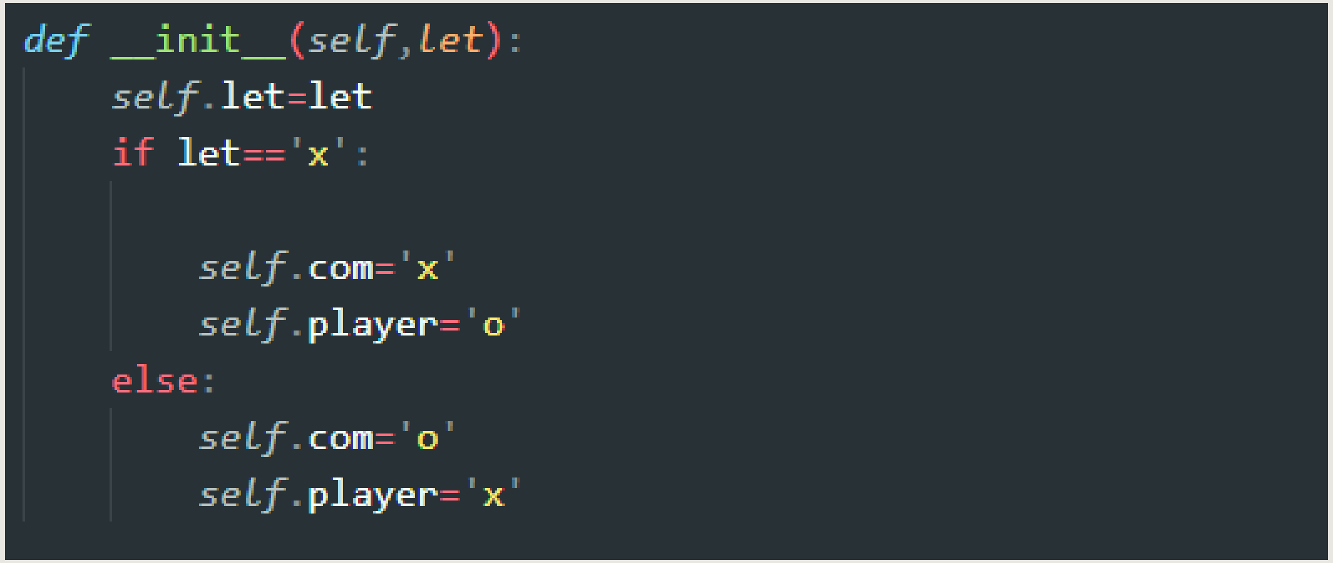
Checks for a winner by checking the board matrix in different dimensions



```
def evale(self):
    if self.borad[0][0]==self.borad[0][1]==self.borad[0][2] and
    self.borad[0][0]!='':
        pl=self.borad[0][0]
        return self.pp[pl]

    elif self.borad[1][0]==self.borad[1][1]==self.borad[1][2] and
    self.borad[1][2]!='':
        pl=self.borad[1][0]
        return self.pp[pl]

    elif self.borad[2][0]==self.borad[2][1]==self.borad[2][2] and
    self.borad[2][2]!='':
        pl=self.borad[2][0]
        return self.pp[pl]

    elif self.borad[0][0]==self.borad[1][0]==self.borad[2][0] and
    self.borad[2][0]!='':
        pl=self.borad[0][0]
        return self.pp[pl]

    elif self.borad[0][1]==self.borad[1][1]==self.borad[2][1] and
    self.borad[2][1]!='':
        pl=self.borad[0][1]
        return self.pp[pl]

    elif self.borad[0][2]==self.borad[1][2]==self.borad[2][2] and
    self.borad[2][2]!='':
```

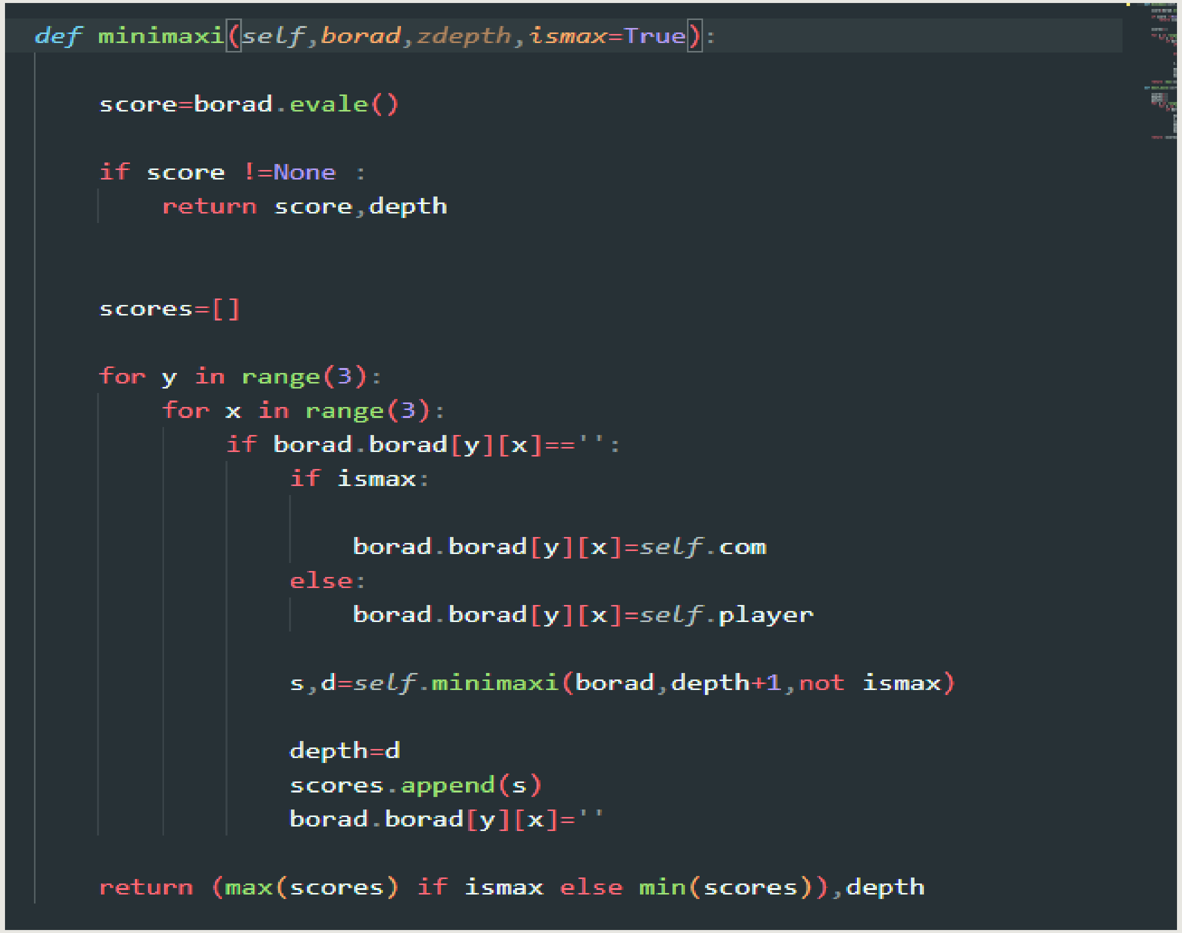
Methods of class Player

Constructor that lets the computer take either X or O



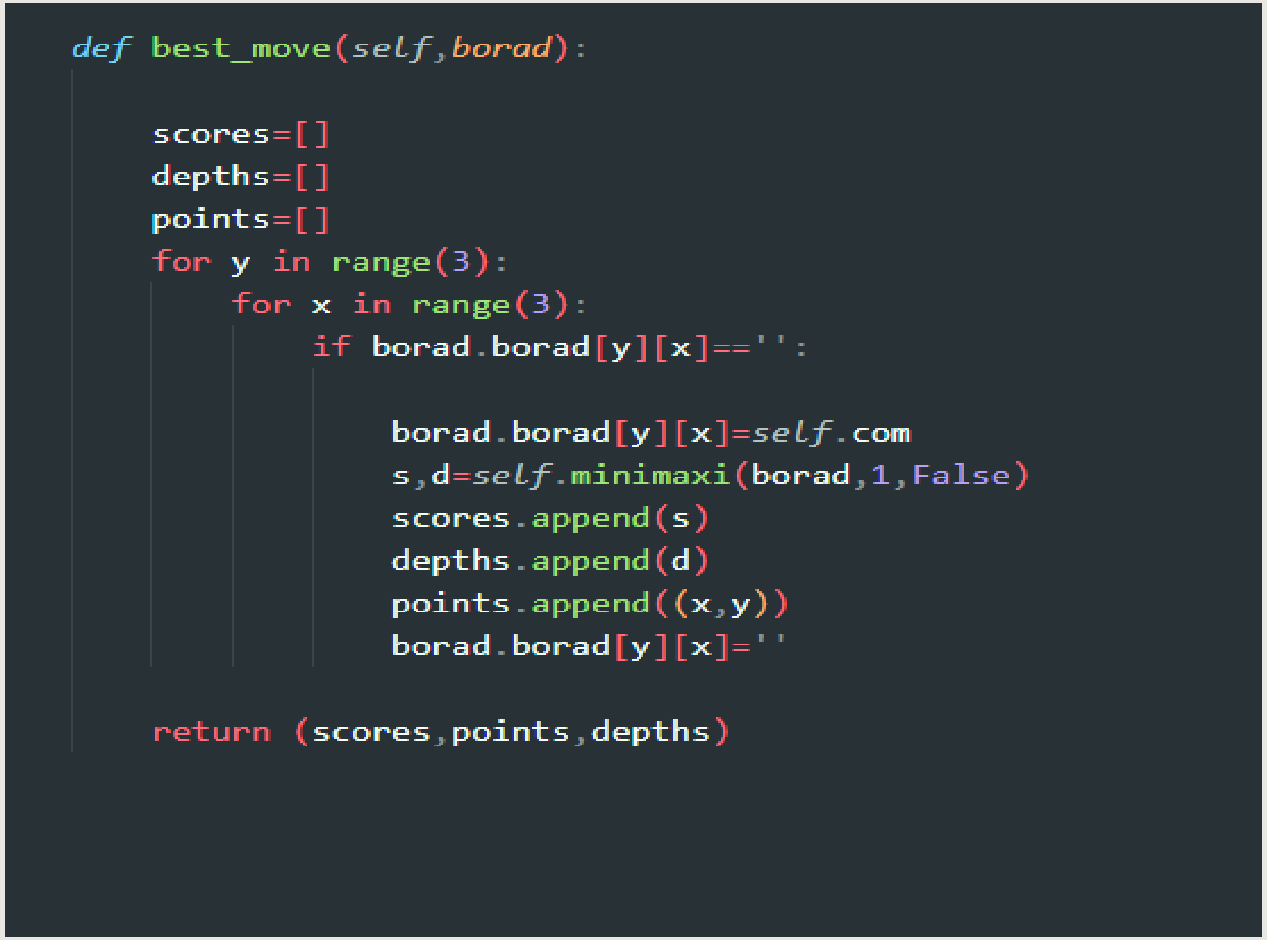
```python
def __init__(self,let):
    self.let=let
    if let=='x':

        self.com='x'
        self.player='o'
    else:
        self.com='o'
        self.player='x'
```

Applys the minmax algorithm to the computer to make the best move



```python
def minimaxi(self,borad,zdepth,ismax=True):

    score=borad.evale()

    if score !=None :
        return score,depth


    scores=[]

    for y in range(3):
        for x in range(3):
            if borad.borad[y][x]=='':
                if ismax:

                    borad.borad[y][x]=self.com
                else:
                    borad.borad[y][x]=self.player

                s,d=self.minimaxi(borad,depth+1,not ismax)

                depth=d
                scores.append(s)
                borad.borad[y][x]=''

    return (max(scores) if ismax else min(scores)),depth
```

Finds the best move and return the score and the depth of layers took to find it



```
def best_move(self,borad):

    scores=[]
    depths=[]
    points=[]
    for y in range(3):
        for x in range(3):
            if borad.borad[y][x]=='':

                borad.borad[y][x]=self.com
                s,d=self.minimaxi(borad,1,False)
                scores.append(s)
                depths.append(d)
                points.append((x,y))
                borad.borad[y][x]=''

    return (scores,points,depths)
```