```
class Counter:
    def __init__(self, x, y, playerID, temp=False):
        self.x = x
        self.y = y
        if not temp:
            board[y][x] = self
        self.player = playerID

    def setPlayer(self, player):
        self.player = player

    def search(self, direction, player, line):
        c = None

        if direction == "N":
            if self.y > 0:
                c = board[self.y - 1][self.x]
        elif direction == "NE":
            if self.y > 0 and self.x < (boardSize - 1):
                c = board[self.y - 1][self.x + 1]
        elif direction == "E":
            if self.x < (boardSize - 1):
                c = board[self.y][self.x + 1]
        elif direction == "SE":
            if self.y < (boardSize - 1) and self.x < (boardSize - 1):
                c = board[self.y + 1][self.x + 1]
        elif direction == "S":
            if self.y < (boardSize - 1):
                c = board[self.y + 1][self.x]
        elif direction == "SW":
            if self.y < (boardSize - 1) and self.x > 0:
                c = board[self.y + 1][self.x - 1]
        elif direction == "W":
            if self.x > 0:
                c = board[self.y][self.x - 1]
        elif direction == "NW":
            if self.y > 0 and self.x > 0:
                c = board[self.y - 1][self.x - 1]

        if type(c) is Counter:
            if c.player != player:
                line.append(c)
                line = c.search(direction, player, line)
                return line
            else:
                return line
        else:
            return []


from tkinter import *




directions = ["N","NE","E","SE","S","SW","W","NW"]
noOfPlayers = 2
playerColours = {1:"black", 2:"white",3:"green",4:"blue"}
scores = {1:0,2:0,3:0,4:0}
```

```
showValid = {1:True,2:True,3:True,4:True}
currentPlayer = 1
boardSize = 8


buttons = []
vars = []

def setupBoard():
    global board, validMoves
    board = []
    validMoves = []
    for y in range(boardSize):
        temp = []
        temp2 = []
        for x in range(boardSize):
            temp.append(None)
            temp2.append(0)
        board.append(temp)
        validMoves.append(temp2)

setupBoard()

def boardDisplay():
    for y in range(boardSize):
        for x in range(boardSize):
            c = board[y][x]
            if type(c) is Counter:
                buttons[y][x].config(bg=playerColours[c.player])
            else:
                if showValid[currentPlayer]:
                    if validMoves[y][x] == 1:
                        buttons[y][x].config(bg="red")
                    else:
                        buttons[y][x].config(bg="grey")
                else:
                    buttons[y][x].config(bg="grey")


def anyMoves():
    return any(1 in s for s in validMoves)


def findValidMoves():
    global currentPlayer
    for y in range(boardSize):
        for x in range(boardSize):
            if type(board[y][x]) is not Counter:
                # board[y][x] = None
                validMoves[y][x] = 0
                for direction in directions:
                    if len(Counter(x, y, currentPlayer, True).search(direction,
currentPlayer, [])) > 0:
                        # board[y][x] = "red"
                        validMoves[y][x] = 1
                        break
            else:
                validMoves[y][x] = 0


```

```
def findValidMoves_weighted():
    global currentPlayer
    for y in range(boardSize):
        for x in range(boardSize):
            total = 0
            if type(board[y][x]) is not Counter:
                for direction in directions:
                    total += len(Counter(x, y, currentPlayer,
True).search(direction, currentPlayer, []))
            validMoves[y][x] = total

def findIndex(item):
    for y in range(boardSize):
        for x in range(boardSize):
            if buttons[y][x] == item:
                return (x, y)


import random

def turn(event):
    global currentPlayer, noOfPlayers, board, win, scores
    pos = findIndex(event.widget)
    x = pos[0]
    y = pos[1]
    if anyMoves():
        if validMoves[y][x] == 1:
            c = Counter(x, y, currentPlayer)
            for direction in directions:
                line = c.search(direction, currentPlayer, [])
                if len(line) > 0:
                    for a in line:
                        a.setPlayer(currentPlayer)

            if noOfPlayers == 1:
                currentPlayer = 2
                findValidMoves_weighted()
                #ai turn
                #print(validMoves)
                bestMove_weight = 0
                bestMoves = []
                for y in range(boardSize):
                    for x in range(boardSize):
                        if validMoves[y][x] == bestMove_weight:
                            bestMoves.append((x,y))
                        elif validMoves[y][x] > bestMove_weight:
                            bestMove_weight = validMoves[y][x]
                            bestMoves = [(x,y)]
                #print(bestMoves)
                movePos = random.choice(bestMoves)
                c = Counter(movePos[0],movePos[1],currentPlayer)
                for direction in directions:
                    line = c.search(direction, currentPlayer, [])
                    if len(line) > 0:
                        for a in line:
                            a.setPlayer(currentPlayer)
                currentPlayer = 1
                findValidMoves()
                boardDisplay()
                buttons[movePos[1]][movePos[0]].config(bg=playerColours[3])
```

```
            else:
                currentPlayer += 1
                if currentPlayer > noOfPlayers:
                    currentPlayer = 1
                win.title("Othello: Player " + str(currentPlayer) + " Turn")
                findValidMoves()
                boardDisplay()
            if not anyMoves():
                print("player "+str(currentPlayer)+" no possible moves")
                moves = False
                for i in range(noOfPlayers):
                    currentPlayer += 1
                    if currentPlayer > noOfPlayers:
                        currentPlayer = 1
                    win.title("Othello: Player " + str(currentPlayer) + " Turn")
                    findValidMoves()
                    boardDisplay()
                    if anyMoves():
                        moves = True
                        break
                if not moves:
                    scores = {1: 0, 2: 0, 3: 0, 4: 0}
                    for y in range(boardSize):
                        for x in range(boardSize):
                            c = board[y][x]
                            if type(c) is Counter:
                                scores[c.player] += 1
                    scoreStr = ""
                    print(scores)
                    if noOfPlayers == 1:
                        noOfPlayers = 2
                    for i in range(noOfPlayers):
                        scoreStr += "Player "+str(i+1)+": " + str(scores[i+1]) + "
"

                    win.title(scoreStr)
    else:
        print("player " + str(currentPlayer) + " no possible moves")
        currentPlayer += 1
        if currentPlayer > noOfPlayers:
            currentPlayer = 1
        win.title("Othello: Player " + str(currentPlayer) + " Turn")
        findValidMoves()
        boardDisplay()
    '''
    for y in range(boardSize):
        for x in range(boardSize):
            if type(board[y][x]) is Counter:
                print(board[y][x].player)
            else:
                print(0)
    '''

def hint(event):
    boardDisplay()
    pos = findIndex(event.widget)
    x = pos[0]
    y = pos[1]
    if validMoves[y][x] == 1:
        for direction in directions:
```

```
            for c in Counter(x, y, currentPlayer, True).search(direction, currentPlayer, []):
                buttons[c.y][c.x].config(bg="yellow")

def game():
    global win, buttons, boardSize
    win.destroy()
    win = Tk()
    win.title("Othello: Player 1 Turn")
    win.geometry("600x600")

    if noOfPlayers == 4:
        boardSize = 2*boardSize
        setupBoard()
        mid = boardSize // 2
        Counter(mid, mid, 1)
        Counter(mid + 1, mid, 1)
        Counter(mid + 1, mid + 1, 1)
        Counter(mid, mid + 1, 1)
        Counter(mid - 1, mid - 1, 2)
        Counter(mid - 2, mid - 1, 2)
        Counter(mid - 2, mid - 2, 2)
        Counter(mid - 1, mid - 2, 2)
        Counter(mid - 1, mid, 3)
        Counter(mid - 2, mid, 3)
        Counter(mid - 2, mid + 1, 3)
        Counter(mid - 1, mid + 1, 3)
        Counter(mid, mid - 1, 4)
        Counter(mid + 1, mid - 1, 4)
        Counter(mid + 1, mid - 2, 4)
        Counter(mid, mid - 2, 4)
    else:
        mid = boardSize // 2
        Counter(mid, mid, 1)
        Counter(mid - 1, mid - 1, 1)
        Counter(mid - 1, mid, 2)
        Counter(mid, mid - 1, 2)

    buttons = []
    for y in range(boardSize):
        temp = []
        for x in range(boardSize):
            b = Button(master=win)
            b.bind("<Button-1>", turn)
            b.bind("<Button-3>", hint)
            temp.append(b)
            b.grid(row=y, column=x, sticky=N+S+E+W)
        buttons.append(temp)

    for i in range(boardSize):
        Grid.rowconfigure(win, i, weight=1)
        Grid.columnconfigure(win, i, weight=1)



    findValidMoves()
    boardDisplay()
    win.mainloop()

win = Tk()
```

```
win.geometry("300x100")

def resetArrays():
    global vars
    #labels = [None for i in range(noOfPlayers)]
    vars = [None for i in range(noOfPlayers)]
    #dropdowns = [None for i in range(noOfPlayers)]

resetArrays()

def var_callback(var,player):
    for i in range(noOfPlayers):
        if vars[i] != None:
            if vars[i] != var:
                if vars[i].get() == var.get():
                    vars[i].set(playerColours[player])

    for i in range(noOfPlayers):
        if vars[i] != None:
            playerColours[i+1] = vars[i].get()

def setup():
    global noOfPlayers, noOfPlayers_var, boardSize, boardSize_var, frame

    try:
        noOfPlayers = noOfPlayers_var.get()
        boardSize = boardSize_var.get()
        frame.destroy()
    except:
        pass

    resetArrays()
    setupBoard()
    win.geometry(str(75*noOfPlayers+310)+"x90")

    frame = Frame(master=win)
    frame.grid()

    #print(noOfPlayers)
    for i in range(noOfPlayers):
        label = Label(master=frame,text="Player "+str(i+1))
        label.grid(row=0,column=i,sticky=N+S+E+W)
        vars[i] = StringVar(win)
        vars[i].trace_variable("w",lambda *_, var=vars[i], player=i+1:
var_callback(var,player))
        vars[i].set(playerColours[i+1])
        dropdown = OptionMenu(frame,vars[i],"black","white","green","blue")
        dropdown.grid(row=1,column=i,sticky=N+S+E+W)

    noOfPlayers_label = Label(master=frame,text="Number Of Players")
    noOfPlayers_label.grid(row=0, column=noOfPlayers, sticky=N + S + E + W)

    noOfPlayers_var = IntVar(value=noOfPlayers)

    noOfPlayers_select = Spinbox(master=frame,values=(1, 2,
4),textvariable=noOfPlayers_var,command=setup)
    noOfPlayers_select.delete(0, 'end')
    noOfPlayers_select.insert(0,noOfPlayers)
    noOfPlayers_select.grid(row=1, column=noOfPlayers, sticky=N + S + E + W)

```

```
    boardSize_label = Label(master=frame, text="Board Size")
    boardSize_label.grid(row=0, column=noOfPlayers+1, sticky=N + S + E + W)

    boardSize_var = IntVar(value=boardSize)

    boardSize_select = Spinbox(master=frame, values=(4, 8, 16),
textvariable=boardSize_var, command=setup)
    boardSize_select.delete(0, 'end')
    boardSize_select.insert(0, boardSize)
    boardSize_select.grid(row=1, column=noOfPlayers+1, sticky=N + S + E + W)

    playButton = Button(master=frame,text="Play",command=game)
    playButton.grid(row=2,column=noOfPlayers+2,sticky=N+S+E+W)

    for i in range(boardSize):
        Grid.rowconfigure(win, i, weight=1)
        Grid.columnconfigure(win, i, weight=1)

setup()
win.mainloop()
```