

```

1 class Counter:
2     def __init__(self, x, y, playerID, temp=False):
3         self.x = x
4         self.y = y
5         if not temp:
6             board[y][x] = self
7         self.player = playerID
8
9     def setPlayer(self, player):
10        self.player = player
11
12    def search(self, direction, player, line):
13        c = None
14
15        if direction == "N":
16            if self.y > 0:
17                c = board[self.y - 1][self.x]
18        elif direction == "NE":
19            if self.y > 0 and self.x < (boardSize - 1):
20                c = board[self.y - 1][self.x + 1]
21        elif direction == "E":
22            if self.x < (boardSize - 1):
23                c = board[self.y][self.x + 1]
24        elif direction == "SE":
25            if self.y < (boardSize - 1) and self.x < (boardSize - 1):
26                c = board[self.y + 1][self.x + 1]
27        elif direction == "S":
28            if self.y < (boardSize - 1):
29                c = board[self.y + 1][self.x]
30        elif direction == "SW":
31            if self.y < (boardSize - 1) and self.x > 0:
32                c = board[self.y + 1][self.x - 1]
33        elif direction == "W":
34            if self.x > 0:
35                c = board[self.y][self.x - 1]
36        elif direction == "NW":
37            if self.y > 0 and self.x > 0:
38                c = board[self.y - 1][self.x - 1]
39
40        if type(c) is Counter:
41            if c.player != player:
42                line.append(c)
43                line = c.search(direction, player, line)
44            return line
45        else:
46            return line
47    else:
48        return []
49
50
51 from tkinter import *
52
53
54
55
56 directions = ["N", "NE", "E", "SE", "S", "SW", "W", "NW"]
57 noOfPlayers = 2
58 playerColours = {1:"black", 2:"white", 3:"green", 4:"blue"}
59 scores = {1:0, 2:0, 3:0, 4:0}

```

```

60 showValid = {1:True,2:True,3:True,4:True}
61 currentPlayer = 1
62 boardSize = 8
63
64
65 buttons = []
66 vars = []
67
68 def setupBoard():
69     global board, validMoves
70     board = []
71     validMoves = []
72     for y in range(boardSize):
73         temp = []
74         temp2 = []
75         for x in range(boardSize):
76             temp.append(None)
77             temp2.append(0)
78         board.append(temp)
79         validMoves.append(temp2)
80
81 setupBoard()
82
83 def boardDisplay():
84     for y in range(boardSize):
85         for x in range(boardSize):
86             c = board[y][x]
87             if type(c) is Counter:
88                 buttons[y][x].config(bg=playerColours[c.player])
89             else:
90                 if showValid[currentPlayer]:
91                     if validMoves[y][x] == 1:
92                         buttons[y][x].config(bg="red")
93                     else:
94                         buttons[y][x].config(bg="grey")
95                 else:
96                     buttons[y][x].config(bg="grey")
97
98
99 def anyMoves():
100     return any(1 in s for s in validMoves)
101
102
103 def findValidMoves():
104     global currentPlayer
105     for y in range(boardSize):
106         for x in range(boardSize):
107             if type(board[y][x]) is not Counter:
108                 # board[y][x] = None
109                 validMoves[y][x] = 0
110                 for direction in directions:
111                     if len(Counter(x, y, currentPlayer, True).search(direction,
currentPlayer, [])) > 0:
112                         # board[y][x] = "red"
113                         validMoves[y][x] = 1
114                         break
115                 else:
116                     validMoves[y][x] = 0
117
118

```

```

119 def findValidMoves_weighted():
120     global currentPlayer
121     for y in range(boardSize):
122         for x in range(boardSize):
123             total = 0
124             if type(board[y][x]) is not Counter:
125                 for direction in directions:
126                     total += len(Counter(x, y, currentPlayer,
True).search(direction, currentPlayer, []))
127             validMoves[y][x] = total
128
129 def findIndex(item):
130     for y in range(boardSize):
131         for x in range(boardSize):
132             if buttons[y][x] == item:
133                 return (x, y)
134
135
136 import random
137
138 def turn(event):
139     global currentPlayer, noOfPlayers, board, win, scores
140     pos = findIndex(event.widget)
141     x = pos[0]
142     y = pos[1]
143     if anyMoves():
144         if validMoves[y][x] == 1:
145             c = Counter(x, y, currentPlayer)
146             for direction in directions:
147                 line = c.search(direction, currentPlayer, [])
148                 if len(line) > 0:
149                     for a in line:
150                         a.setPlayer(currentPlayer)
151
152             if noOfPlayers == 1:
153                 currentPlayer = 2
154                 findValidMoves_weighted()
155                 #ai turn
156                 #print(validMoves)
157                 bestMove_weight = 0
158                 bestMoves = []
159                 for y in range(boardSize):
160                     for x in range(boardSize):
161                         if validMoves[y][x] == bestMove_weight:
162                             bestMoves.append((x,y))
163                         elif validMoves[y][x] > bestMove_weight:
164                             bestMove_weight = validMoves[y][x]
165                             bestMoves = [(x,y)]
166                 #print(bestMoves)
167                 movePos = random.choice(bestMoves)
168                 c = Counter(movePos[0],movePos[1],currentPlayer)
169                 for direction in directions:
170                     line = c.search(direction, currentPlayer, [])
171                     if len(line) > 0:
172                         for a in line:
173                             a.setPlayer(currentPlayer)
174                 currentPlayer = 1
175                 findValidMoves()
176                 boardDisplay()
177                 buttons[movePos[1]][movePos[0]].config(bg=playerColours[3])

```

```

178         else:
179             currentPlayer += 1
180             if currentPlayer > noOfPlayers:
181                 currentPlayer = 1
182                 win.title("Othello: Player " + str(currentPlayer) + " Turn")
183                 findValidMoves()
184                 boardDisplay()
185             if not anyMoves():
186                 print("player "+str(currentPlayer)+" no possible moves")
187                 moves = False
188                 for i in range(noOfPlayers):
189                     currentPlayer += 1
190                     if currentPlayer > noOfPlayers:
191                         currentPlayer = 1
192                         win.title("Othello: Player " + str(currentPlayer) + " Turn")
193                         findValidMoves()
194                         boardDisplay()
195                         if anyMoves():
196                             moves = True
197                             break
198                 if not moves:
199                     scores = {1: 0, 2: 0, 3: 0, 4: 0}
200                     for y in range(boardSize):
201                         for x in range(boardSize):
202                             c = board[y][x]
203                             if type(c) is Counter:
204                                 scores[c.player] += 1
205                     scoreStr = ""
206                     print(scores)
207                     if noOfPlayers == 1:
208                         noOfPlayers = 2
209                     for i in range(noOfPlayers):
210                         scoreStr += "Player "+str(i+1)+": " + str(scores[i+1]) + "
211
212                 win.title(scoreStr)
213     else:
214         print("player " + str(currentPlayer) + " no possible moves")
215         currentPlayer += 1
216         if currentPlayer > noOfPlayers:
217             currentPlayer = 1
218             win.title("Othello: Player " + str(currentPlayer) + " Turn")
219             findValidMoves()
220             boardDisplay()
221         ...
222     for y in range(boardSize):
223         for x in range(boardSize):
224             if type(board[y][x]) is Counter:
225                 print(board[y][x].player)
226             else:
227                 print(0)
228         ...
229
230 def hint(event):
231     boardDisplay()
232     pos = findIndex(event.widget)
233     x = pos[0]
234     y = pos[1]
235     if validMoves[y][x] == 1:
236         for direction in directions:

```

```

237         for c in Counter(x, y, currentPlayer, True).search(direction,
currentPlayer, []):
238             buttons[c.y][c.x].config(bg="yellow")
239
240 def game():
241     global win, buttons, boardSize
242     win.destroy()
243     win = Tk()
244     win.title("Othello: Player 1 Turn")
245     win.geometry("600x600")
246
247     if noOfPlayers == 4:
248         boardSize = 2*boardSize
249         setupBoard()
250         mid = boardSize // 2
251         Counter(mid, mid, 1)
252         Counter(mid + 1, mid, 1)
253         Counter(mid + 1, mid + 1, 1)
254         Counter(mid, mid + 1, 1)
255         Counter(mid - 1, mid - 1, 2)
256         Counter(mid - 2, mid - 1, 2)
257         Counter(mid - 2, mid - 2, 2)
258         Counter(mid - 1, mid - 2, 2)
259         Counter(mid - 1, mid, 3)
260         Counter(mid - 2, mid, 3)
261         Counter(mid - 2, mid + 1, 3)
262         Counter(mid - 1, mid + 1, 3)
263         Counter(mid, mid - 1, 4)
264         Counter(mid + 1, mid - 1, 4)
265         Counter(mid + 1, mid - 2, 4)
266         Counter(mid, mid - 2, 4)
267     else:
268         mid = boardSize // 2
269         Counter(mid, mid, 1)
270         Counter(mid - 1, mid - 1, 1)
271         Counter(mid - 1, mid, 2)
272         Counter(mid, mid - 1, 2)
273
274     buttons = []
275     for y in range(boardSize):
276         temp = []
277         for x in range(boardSize):
278             b = Button(master=win)
279             b.bind("<Button-1>", turn)
280             b.bind("<Button-3>", hint)
281             temp.append(b)
282             b.grid(row=y, column=x, sticky=N+S+E+W)
283         buttons.append(temp)
284
285     for i in range(boardSize):
286         Grid.rowconfigure(win, i, weight=1)
287         Grid.columnconfigure(win, i, weight=1)
288
289
290
291     findValidMoves()
292     boardDisplay()
293     win.mainloop()
294
295 win = Tk()

```

```

296 win.geometry("300x100")
297
298 def resetArrays():
299     global vars
300     #labels = [None for i in range(noOfPlayers)]
301     vars = [None for i in range(noOfPlayers)]
302     #dropdowns = [None for i in range(noOfPlayers)]
303
304 resetArrays()
305
306 def var_callback(var,player):
307     for i in range(noOfPlayers):
308         if vars[i] != None:
309             if vars[i] != var:
310                 if vars[i].get() == var.get():
311                     vars[i].set(playerColours[player])
312
313     for i in range(noOfPlayers):
314         if vars[i] != None:
315             playerColours[i+1] = vars[i].get()
316
317 def setup():
318     global noOfPlayers, noOfPlayers_var, boardSize, boardSize_var, frame
319
320     try:
321         noOfPlayers = noOfPlayers_var.get()
322         boardSize = boardSize_var.get()
323         frame.destroy()
324     except:
325         pass
326
327     resetArrays()
328     setupBoard()
329     win.geometry(str(75*noOfPlayers+310)+"x90")
330
331     frame = Frame(master=win)
332     frame.grid()
333
334     #print(noOfPlayers)
335     for i in range(noOfPlayers):
336         label = Label(master=frame,text="Player "+str(i+1))
337         label.grid(row=0,column=i,sticky=N+S+E+W)
338         vars[i] = StringVar(win)
339         vars[i].trace_variable("w",lambda *_ , var=vars[i], player=i+1:
var_callback(var,player))
340         vars[i].set(playerColours[i+1])
341         dropdown = OptionMenu(frame,vars[i],"black","white","green","blue")
342         dropdown.grid(row=1,column=i,sticky=N+S+E+W)
343
344     noOfPlayers_label = Label(master=frame,text="Number Of Players")
345     noOfPlayers_label.grid(row=0, column=noOfPlayers, sticky=N + S + E + W)
346
347     noOfPlayers_var = IntVar(value=noOfPlayers)
348
349     noOfPlayers_select = Spinbox(master=frame,values=(1, 2,
4),textvariable=noOfPlayers_var,command=setup)
350     noOfPlayers_select.delete(0, 'end')
351     noOfPlayers_select.insert(0,noOfPlayers)
352     noOfPlayers_select.grid(row=1, column=noOfPlayers, sticky=N + S + E + W)
353

```

```
354     boardSize_label = Label(master=frame, text="Board Size")
355     boardSize_label.grid(row=0, column=noOfPlayers+1, sticky=N + S + E + W)
356
357     boardSize_var = IntVar(value=boardSize)
358
359     boardSize_select = Spinbox(master=frame, values=(4, 8, 16),
textvariable=boardSize_var, command=setup)
360     boardSize_select.delete(0, 'end')
361     boardSize_select.insert(0, boardSize)
362     boardSize_select.grid(row=1, column=noOfPlayers+1, sticky=N + S + E + W)
363
364     playButton = Button(master=frame, text="Play", command=game)
365     playButton.grid(row=2, column=noOfPlayers+2, sticky=N+S+E+W)
366
367     for i in range(boardSize):
368         Grid.rowconfigure(win, i, weight=1)
369         Grid.columnconfigure(win, i, weight=1)
370
371 setup()
372 win.mainloop()
373
374
375
376
377
378
```