**ATHER ENERGY**

**PROBLEM:**

To build a Working replica of the 2048 game.

**LANGUAGE USED:**

Python

**CODE: (2048.py)**

from \_\_future\_\_ import print\_function

try:

    import tkinter as tk # For Python 3

    import tkinter.messagebox as messagebox

except:

    import Tkinter as tk # For Python 2

    import tkMessageBox as messagebox

import random

class Grid:

    '''The data structure representation of the 2048 game.

    '''

    def \_\_init\_\_(self, n):

        self.size = n

        self.cells = self.generate\_empty\_grid()

        self.compressed = False

        self.merged = False

        self.moved = False

        self.current\_score = 0

    def random\_cell(self):

        cell = random.choice(self.retrieve\_empty\_cells())

        i = cell[0]

        j = cell[1]

        self.cells[i][j] = 2 if random.random() < 0.9 else 4

    def retrieve\_empty\_cells(self):

        empty\_cells = []

        for i in range(self.size):

            for j in range(self.size):

                if self.cells[i][j] == 0:

                    empty\_cells.append((i, j))

        return empty\_cells

    def generate\_empty\_grid(self):

        return [[0] \* self.size for i in range(self.size)]

    def transpose(self):

        self.cells = [list(t) for t in zip(\*self.cells)]

    def reverse(self):

        for i in range(self.size):

            start = 0

            end = self.size - 1

            while start < end:

                self.cells[i][start], self.cells[i][end] = \

                    self.cells[i][end], self.cells[i][start]

                start += 1

                end -= 1

    def clear\_flags(self):

        self.compressed = False

        self.merged = False

        self.moved = False

    def left\_compress(self):

        self.compressed = False

        new\_grid = self.generate\_empty\_grid()

        for i in range(self.size):

            count = 0

            for j in range(self.size):

                if self.cells[i][j] != 0:

                    new\_grid[i][count] = self.cells[i][j]

                    if count != j:

                        self.compressed = True

                    count += 1

        self.cells = new\_grid

    def left\_merge(self):

        self.merged = False

        for i in range(self.size):

            for j in range(self.size - 1):

                if self.cells[i][j] == self.cells[i][j + 1] and \

                   self.cells[i][j] != 0:

                    self.cells[i][j] \*= 2

                    self.cells[i][j + 1] = 0

                    self.current\_score += self.cells[i][j]

                    self.merged = True

    def found\_2048(self):

        for i in range(self.size):

            for j in range(self.size):

                if self.cells[i][j] >= 2048:

                    return True

        return False

    def has\_empty\_cells(self):

        for i in range(self.size):

            for j in range(self.size):

                if self.cells[i][j] == 0:

                    return True

        return False

    def can\_merge(self):

        for i in range(self.size):

            for j in range(self.size - 1):

                if self.cells[i][j] == self.cells[i][j + 1]:

                    return True

        for j in range(self.size):

            for i in range(self.size - 1):

                if self.cells[i][j] == self.cells[i + 1][j]:

                    return True

        return False

    def set\_cells(self, cells):

        self.cells = cells

    def print\_grid(self):

        print('-' \* 40)

        for i in range(self.size):

            for j in range(self.size):

                print('%d\t' % self.cells[i][j], end='')

            print()

        print('-' \* 40)

class GamePanel:

    '''The GUI view class of the 2048 game showing via tkinter.'''

    CELL\_PADDING = 10

    BACKGROUND\_COLOR = '#92877d'

    EMPTY\_CELL\_COLOR = '#9e948a'

    CELL\_BACKGROUND\_COLOR\_DICT = {

        '2': '#eee4da',

        '4': '#ede0c8',

        '8': '#f2b179',

        '16': '#f59563',

        '32': '#f67c5f',

        '64': '#f65e3b',

        '128': '#edcf72',

        '256': '#edcc61',

        '512': '#edc850',

        '1024': '#edc53f',

        '2048': '#edc22e',

        'beyond': '3c3a32'

    }

    CELL\_COLOR\_DICT = {

        '2': '#776e65',

        '4': '#776e65',

        '8': '#f9f6f2',

        '16': '#f9f6f2',

        '32': '#f9f6f2',

        '64': '#f9f6f2',

        '128': '#f9f6f2',

        '256': '#f9f6f2',

        '512': '#f9f6f2',

        '1024': '#f9f6f2',

        '2048': '#f9f6f2',

        'beyond': '#f9f6f2'

    }

    FONT = ('Verdana', 24, 'bold')

    UP\_KEYS = ('w', 'W', 'Up')

    LEFT\_KEYS = ('a', 'A', 'Left')

    DOWN\_KEYS = ('s', 'S', 'Down')

    RIGHT\_KEYS = ('d', 'D', 'Right')

    def \_\_init\_\_(self, grid):

        self.grid = grid

        self.root = tk.Tk()

        self.root.title('2048')

        self.background = tk.Frame(self.root, bg=GamePanel.BACKGROUND\_COLOR)

        self.cell\_labels = []

        for i in range(self.grid.size):

            row\_labels = []

            for j in range(self.grid.size):

                label = tk.Label(self.background, text='',

                                 bg=GamePanel.EMPTY\_CELL\_COLOR,

                                 justify=tk.CENTER, font=GamePanel.FONT,

                                 width=4, height=2)

                label.grid(row=i, column=j, padx=10, pady=10)

                row\_labels.append(label)

            self.cell\_labels.append(row\_labels)

        self.background.grid()

    def paint(self):

        for i in range(self.grid.size):

            for j in range(self.grid.size):

                if self.grid.cells[i][j] == 0:

                    self.cell\_labels[i][j].configure(

                         text='',

                         bg=GamePanel.EMPTY\_CELL\_COLOR)

                else:

                    cell\_text = str(self.grid.cells[i][j])

                    if self.grid.cells[i][j] > 2048:

                        bg\_color = GamePanel.CELL\_BACKGROUND\_COLOR\_DICT.get('beyond')

                        fg\_color = GamePanel.CELL\_COLOR\_DICT.get('beyond')

                    else:

                        bg\_color = GamePanel.CELL\_BACKGROUND\_COLOR\_DICT.get(cell\_text)

                        fg\_color = GamePanel.CELL\_COLOR\_DICT.get(cell\_text)

                    self.cell\_labels[i][j].configure(

                        text=cell\_text,

                        bg=bg\_color, fg=fg\_color)

class Game:

    '''The main game class which is the controller of the whole game.'''

    def \_\_init\_\_(self, grid, panel):

        self.grid = grid

        self.panel = panel

        self.start\_cells\_num = 2

        self.over = False

        self.won = False

        self.keep\_playing = False

    def is\_game\_terminated(self):

        return self.over or (self.won and (not self.keep\_playing))

    def start(self):

        self.add\_start\_cells()

        self.panel.paint()

        self.panel.root.bind('<Key>', self.key\_handler)

        self.panel.root.mainloop()

    def add\_start\_cells(self):

        for i in range(self.start\_cells\_num):

            self.grid.random\_cell()

    def can\_move(self):

        return self.grid.has\_empty\_cells() or self.grid.can\_merge()

    def key\_handler(self, event):

        if self.is\_game\_terminated():

            return

        self.grid.clear\_flags()

        key\_value = event.keysym

        print('{} key pressed'.format(key\_value))

        if key\_value in GamePanel.UP\_KEYS:

            self.up()

        elif key\_value in GamePanel.LEFT\_KEYS:

            self.left()

        elif key\_value in GamePanel.DOWN\_KEYS:

            self.down()

        elif key\_value in GamePanel.RIGHT\_KEYS:

            self.right()

        else:

            pass

        self.panel.paint()

        print('Score: {}'.format(self.grid.current\_score))

        if self.grid.found\_2048():

            self.you\_win()

            if not self.keep\_playing:

                return

        if self.grid.moved:

            self.grid.random\_cell()

        self.panel.paint()

        if not self.can\_move():

            self.over = True

            self.game\_over()

    def you\_win(self):

        if not self.won:

            self.won = True

            print('You Win!')

            if messagebox.askyesno('2048', 'You Win!\n'

                                       'Are you going to continue the 2048 game?'):

                self.keep\_playing = True

    def game\_over(self):

        print('Game over!')

        messagebox.showinfo('2048', 'Oops!\n'

                                    'Game over!')

    def up(self):

        self.grid.transpose()

        self.grid.left\_compress()

        self.grid.left\_merge()

        self.grid.moved = self.grid.compressed or self.grid.merged

        self.grid.left\_compress()

        self.grid.transpose()

    def left(self):

        self.grid.left\_compress()

        self.grid.left\_merge()

        self.grid.moved = self.grid.compressed or self.grid.merged

        self.grid.left\_compress()

    def down(self):

        self.grid.transpose()

        self.grid.reverse()

        self.grid.left\_compress()

        self.grid.left\_merge()

        self.grid.moved = self.grid.compressed or self.grid.merged

        self.grid.left\_compress()

        self.grid.reverse()

        self.grid.transpose()

    def right(self):

        self.grid.reverse()

        self.grid.left\_compress()

        self.grid.left\_merge()

        self.grid.moved = self.grid.compressed or self.grid.merged

        self.grid.left\_compress()

        self.grid.reverse()

if \_\_name\_\_ == '\_\_main\_\_':

    size = 4

    grid = Grid(size)

    panel = GamePanel(grid)

    game2048 = Game(grid, panel)

    game2048.start()

**OUTPUT:**









