实训

核心算法、可视化

任务：组合、扩展

2048 游戏项目

- 设计思路

  1.游戏中界面

  - 上下左右移动
  - 随机位置添加2或4
  - 判断游戏是否结束
  - 分数计算与显示、重新开始

  2.游戏结束界面

  - 显示分数、重新开始按钮
  - 弹窗选择是否继续游戏

1.项目核心功能

- 实现对二维数组进行上、下、左、右移动，实现对应方向上的元素合并，合成2048或横纵无法移动则游戏结束

1.1 零元素后移

目的：用于移动时实现相同元素合并

```
# 零元素后移

# 有一个列表，从后往前判断，如果是0，将元素删掉，在最后追加一个0
def zero_to_end():
    '''
        将0元素移动到末尾
    :return: None
    '''
    for i in range(len(list_merge) - 1,-1,-1):
        if list_merge[i] == 0:
            del list_merge[i]
            list_merge.append(0)


list_merge = [2,4,0,2]
zero_to_end()
# 输入[2,4,2,0]
```

```
print(list_merge)
```

1.2 相同元素合并

目的：实现对应移动方向上的相同元素合并

思路：将0元素往后移动，如果相邻的元素相同，则相加

2 0 0 2

2+2 0 0 0

```
def merge():
    '''
        相邻元素合并相加
    :return: None
    '''
    # 0元素移动到末尾
    zero_to_end()

    for i in range(len(list_merge) - 1):
        # 判断相邻元素是否相等，且不为0
        if list_merge[i] == list_merge[i + 1] and list_merge[i] != 0 :
            # 前一个元素加上后一个元素
            list_merge[i] += list_merge[i + 1]
            # 删除后一个元素
            del list_merge[i + 1]
            # 保证长度一致，追加一个0
            list_merge.append(0)


list_merge = [2,2,0,2]
merge()
# 输入[4,2,0,0]
print(list_merge)
```

1.3 左移

```
map = [
    [2,4,2,2],
    [0,2,2,0],
    [2,2,2,2],
    [0,4,2,2]
]

def move_left():
    '''
        元素左移
    :return: None
    '''
    for line in map: # line : 表示map的每一行
        global list_merge
        list_merge = line
```

```
        merge()

move_left()
# 输出
# [2,4,4,0]
# [4,0,0,0]
# [4,4,0,0]
# [4,4,0,0]
for line in map:
    print(line)
```

1.4 右移

```
def move_right():
    '''
        元素右移
    :return: None
    '''
    for line in map:
        global  list_merge
        # 将每一行反向
        list_merge = line[::-1]
        # 左移合并
        merge()
        # 反向填充
        line[::-1] = list_merge

move_right()
# 输出
# [0,2,4,4]
# [0,0,0,4]
# [0,0,4,4]
# [0,0,4,4]
for line in map:
    print(line)
```

1.5 矩阵转置

- 思路
  - 循环交换行和列
    - 中心对称轴上的数无需交换，行号循环次数为3次
    - 列号 = 行号+1 ~ 行数
  - 包对应位置的(行号，列号) 与（列号，行号）对应的数交换

行变列，列变行

m * n 矩阵 n * m

原理：按照中心斜线，上下同位置数据翻转

```
map = [
    [2,2,0,2],
    [4,4,4,0],
    [4,4,2,2],
    [2,2,0,0]
]

def  metrix_deivce():
    '''
        矩阵转置
    :return: None
    '''
    for r in range(len(map) - 1):
        for c in range(r + 1 , len(map)):
            # print(r,c)
            map[c][r],map[r][c] = map[r][c],map[c][r]
metrix_deivce()
for line in map:
    print(line)
```

1.6 上移

- 思路

  1.将map转置

  2.所有元素左移

  3.再转置

```
map = [
    [2,2,0,2],
    [4,4,4,0],
    [4,4,2,2],
    [2,2,0,0]
]
def move_up():
    '''
        元素上移
    :return: None
    '''
    # map转置
    metrix_deivce()
    # 左移合并
    move_left()
    # 转置
    metrix_deivce()


move_up()
for line in map:
    print(line)
```

1.7 下移

- 思路：

  1.转置map

  2.右移合并

  3.转置

```
map = [
    [2,2,0,2],
    [4,4,4,0],
    [4,4,2,2],
    [2,2,0,0]
]
def move_down():
    '''
        元素下移
    :return: None
    '''
    # map转置
    metrix_deivce()
    # 左移合并
    move_right()
    # 转置
    metrix_deivce()
move_down()
for line in map:
    print(line)
```

1.8 填充2或4

思路：

1.计算空白位置：列表中的哪些位置（行和列）上的元素为0

抽象位置为一个类 LocationModel： x ， y

2. 选择随机位置：从计算出来的空白位置随机选择一个位置

random.choice() 返回随机项

3. 随机生成4： 2 出现概率为90%，4 ： 10%

4. 将二维列表中的元素改为2或4

```
import random

class LocationModel:
    def __init__(self,x,y):
        self.x = x # x轴（行数）
        self.y = y # y轴（列数）


class GameController:
    def __init__(self):
```

```
        self.__map = [
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0]
        ]
        self.__list_empty_location = []

    @property
    def map(self):  # map 是只读模式
        return self.__map

    def generate_number(self):
        # 1.随机位置（计算map中0的位置）存储
        #  清空之前存储的空白位置
        self.__list_empty_location.clear()
        self.__calculate_empty_loction()

        # 2.从空白位置随机选择1个  random.choice
        loc = random.choice(self.__list_empty_location)

        # 3.随机生成一个新的数字 2/4 - 2概率:0.9  4:0.1
        random_number = self.__random_generate_two_four()

        # 4.修改对应随机位置的数，修改为随机的2/4
        self.__map[loc.x][loc.y] = random_number

    def __random_generate_two_four(self):
        ''' 随机生成 2 或 4'''
        return  2 if random.randint(1,10) != 1 else 4

    def __calculate_empty_loction(self):
        ''' 计算空白位置  '''
        for x in range(len(self.__map)): # 行
            for y in range(len(self.__map[x])): # 列
                if self.__map[x][y] == 0:
                    # 将当前这个元素追加到空白位置列表中
                    self.__list_empty_location.append(LocationModel(x,y))


a = GameController()
a.generate_number()
a.generate_number()
for line in a.map:
    print(line)
```

1.9 判断游戏是否结束

思路：

- 1.若存在空白位置，游戏不结束
- 2.若可横向或纵向移动，游戏不结束

```
def isGameOver(self):
    '''
```

```
        判断游戏是否结束
    :return: 结束与否
    '''
    # 1. 存在空白位置
    if len(self.__list_empty_location) != 0 :
        return False

    # 2. 可以横纵移动
    for row in range(len(self.__map)):
        for col in range(len(self.__map[row]) - 1):
            if self.__map[row][col] == self.__map[row][col + 1] or \
                    self.__map[col][row] == self.__map[col+1][row]:
                return False

    return True
```

1.10 游戏重置

扩展

思路：重置矩阵、分数等，重新生成2个2或4

```
def reset(self):
    '''
        重置：self.__map,成绩，生成2个数字
    :return: None
    '''
    self.__map = [
        [0,0,0,0],
        [0,0,0,0],
        [0,0,0,0],
        [0,0,0,0]
    ]
    self.generate_number()
    self.generate_number()
```

2.游戏可视化

2.1 绘制游戏界面

导包：

```
import random
from tkinter import *
```

```
import random
from tkinter import *
```

```python
# 零元素后移
# 有一个列表，从后往前判断，如果是0，将元素删掉，在最后追加一个0
def zero_to_end():
    '''
        将0元素移动到末尾
    :return: None
    '''
    for i in range(len(list_merge) - 1, -1, -1):
        if list_merge[i] == 0:
            del list_merge[i]
            list_merge.append(0)

def merge():
    '''
        相邻元素合并相加
    :return: None
    '''
    # 0元素移动到末尾
    zero_to_end()

    for i in range(len(list_merge) - 1):
        # 判断相邻元素是否相等，且不为0
        if list_merge[i] == list_merge[i + 1] and list_merge[i] != 0:
            # 前一个元素加上后一个元素
            list_merge[i] += list_merge[i + 1]
            # 删除后一个元素
            del list_merge[i + 1]
            # 保证长度一致，追加一个0
            list_merge.append(0)

# map = [
#     [2,4,2,2],
#     [0,2,2,0],
#     [2,2,2,2],
#     [0,4,2,2]
# ]

def move_left():
    '''
        元素左移
    :return: None
    '''
    for line in map:  # line : 表示map的每一行
        global list_merge
        list_merge = line
        merge()

def move_right():
    '''
        元素右移
    :return: None
    '''
    for line in map:
        global list_merge
        # 将每一行反向
        list_merge = line[::-1]
        # 左移合并
        merge()
```

```
        # 反向填充
        line[::-1] = list_merge

def metrix_deivce():
    '''
        矩阵转置
    :return: None
    '''
    for r in range(len(map) - 1):
        for c in range(r + 1, len(map)):
            # print(r,c)
            map[c][r], map[r][c] = map[r][c], map[c][r]

def move_up():
    '''
        元素上移
    :return: None
    '''
    # map转置
    metrix_deivce()
    # 左移合并
    move_left()
    # 转置
    metrix_deivce()

def move_down():
    '''
        元素下移
    :return: None
    '''
    # map转置
    metrix_deivce()
    # 左移合并
    move_right()
    # 转置
    metrix_deivce()

class LocationModel:
    def __init__(self,x,y):
        self.x = x # x轴（行数）
        self.y = y # y轴（列数）


class GameController:
    def __init__(self):
        self.__map = [
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0]
        ]
        # 存放空白位置
        self.__list_empty_location = []

    @property
    def map(self):  # map 是只读模式
        return self.__map
```

```
    def generate_number(self):
        # 1.随机位置（计算map中0的位置）存储
        #  清空之前存储的空白位置
        self.__list_empty_location.clear()
        self.__calculate_empty_loction()

        # 2.从空白位置随机选择1个  random.choice
        loc = random.choice(self.__list_empty_location)

        # 3.随机生成一个新的数字 2/4 - 2概率:0.9  4:0.1
        random_number = self.__random_generate_two_four()

        # 4.修改对应随机位置的数，修改为随机的2/4
        self.__map[loc.x][loc.y] = random_number

    def __random_generate_two_four(self):
        ''' 随机生成 2 或 4'''
        return  2 if random.randint(1,10) != 1 else 4

    def __calculate_empty_loction(self):
        ''' 计算空白位置  '''
        for x in range(len(self.__map)): # 行
            for y in range(len(self.__map[x])): # 列
                if self.__map[x][y] == 0:
                    # 将当前这个元素追加到空白位置列表中
                    self.__list_empty_location.append(LocationModel(x,y))

    def isGameOver(self):
        '''
            判断游戏是否结束
        :return: 结束与否
        '''
        # 1. 存在空白位置
        if len(self.__list_empty_location) != 0 :
            return False

        # 2. 可以横纵移动
        for row in range(len(self.__map)):
            for col in range(len(self.__map[row]) - 1):
                if self.__map[row][col] == self.__map[row][col + 1] or \
                        self.__map[col][row] == self.__map[col+1][row]:
                    return False

        return True

    def reset(self):
        '''
            重置：self.__map,成绩，生成2个数字
        :return: None
        '''
        # TODO

class GameView:

    def __init__(self):
        # 创建窗口对象
        self.__root = Tk()
        # 设置背景颜色
```

```
        self.__game_bg_color = '#bbada0'
        # 进行界面初始配置
        self.__gameInit()

    def __gameInit(self):
        '''
            游戏界面初始配置
        :return:
        '''
        # 设置窗口标题
        self.__root.title('2048')
        # 设置窗口大小不可变
        self.__root.resizable(width=False,height=False)

    def main(self):
        # 创建组件容器对象
        # self.__frame = Frame(self.__root,bg=self.__game_bg_color)
        self.__frame =
Frame(self.__root,bg=self.__game_bg_color,width=800,height=600)
        # 设置显示位置（默认0行0列位置）
        self.__frame.grid()
        # 窗口事件循环
        self.__root.mainloop()

if __name__ == '__main__':
    view = GameView()
    view.main()
```

2.2 绘制数值文本

```
class GameView:
    def __init__(self):
        # 创建窗口对象
        self.__root = Tk()
        # 设置背景颜色
        self.__game_bg_color = '#bbada0'
        # 创建核心业务逻辑对象 -- Controller 对象
        self.__controller = GameController()
        # 存储数值的label文本对象
        self.__map_lable = []
        # 进行界面初始配置
        self.__gameInit()
```

```
def __gameScreenDraw(self):
    '''
        绘制显示的数值及区域
    :return:
    '''
    for row in range(len(self.__controller.map)):
        for col in range(len(self.__controller.map[row])):
            # 获取每个位置的值
```

```
            value = self.__controller.map[row][col]
            # 当位置上的值为0 显示空，否则显示对应的值
            text = str(value) if value else ""
            # 构建显示的文本对象
            # Label 置于窗口上的部件，通常用于显示文本或图像
            '''
                text:显示的字符串
                width、height:部件的宽高
                font:字体（字体名称,字体大小，字体属性）
            '''
            lable =
Label(self.__frame,bg="pink",text=text,width=4,height=2,font=("黑体",20,"bold"))
            # 绘制文本显示的位置
            lable.grid(row = row,column=col,padx=5,pady=5)

def main(self):
    # 创建组件容器对象
    self.__frame = Frame(self.__root,bg=self.__game_bg_color)
    # self.__frame =
Frame(self.__root,bg=self.__game_bg_color,width=800,height=600)
    # 设置显示位置（默认0行0列位置）
    self.__frame.grid()
    # 初识窗口绘制
    self.__gameScreenDraw()
    # 窗口事件循环
    self.__root.mainloop()
```

2.3 游戏重置

思路:

- 重置map，生成2个数值 -- controller.reset()
- 游戏更新
    - 1.获取矩阵map中的数值
    - 2.获取数值的Label对象
    - 3.设置Label的属性：text、背景、前景
    - 4.设置分数（扩展）

```
class GameView:
    def __init__(self):
        # 创建窗口对象
        self.__root = Tk()
        # 设置背景颜色
        self.__game_bg_color = '#bbada0'
        # 创建核心业务逻辑对象 -- Controller 对象
        self.__controller = GameController()
        # 存储数值的label文本对象
        self.__map_lable = []
        # 进行界面初始配置
        self.__gameInit()
        # 设置游戏中每个数据对应的位置色块的颜色
        self.__mapcolor = {
            0: ("#cdc1b4", "#776e65"),
```

```
            2: ("#eee4da", "#776e65"),
            4: ("#ede0c8", "#f9f6f2"),
            8: ("#f2b179", "#f9f6f2"),
            16: ("#f59563", "#f9f6f2"),
            32: ("#f67c5f", "#f9f6f2"),
            64: ("#f65e3b", "#f9f6f2"),
            128: ("#edcf72", "#f9f6f2"),
            256: ("#edcc61", "#f9f6f2"),
            512: ("#e4c02a", "#f9f6f2"),
            1024: ("#e2ba13", "#f9f6f2"),
            2048: ("#ecc400", "#f9f6f2"),
            4096: ("#ae84a8", "#f9f6f2"),
            8192: ("#b06ca8", "#f9f6f2"),
            # ----其它颜色都与8192相同---------
            2 ** 14: ("#b06ca8", "#f9f6f2"),
            2 ** 15: ("#b06ca8", "#f9f6f2"),
            2 ** 16: ("#b06ca8", "#f9f6f2"),
            2 ** 17: ("#b06ca8", "#f9f6f2"),
            2 ** 18: ("#b06ca8", "#f9f6f2"),
            2 ** 19: ("#b06ca8", "#f9f6f2"),
            2 ** 20: ("#b06ca8", "#f9f6f2")
        }
```

添加内容：

```
def __gameScreenDraw(self):
    '''
        绘制显示的数值及区域
    :return:
    '''
    for row in range(len(self.__controller.map)):
        # 存储每一行数值的label对象
        row_list = []
        for col in range(len(self.__controller.map[row])):
            # 获取每个位置的值
            value = self.__controller.map[row][col]
            # 当位置上的值为0 显示空，否则显示对应的值
            text = str(value) if value else ""
            # 构建显示的文本对象
            # Label 置于窗口上的部件，通常用于显示文本或图像
            '''
                text:显示的字符串
                width、height:部件的宽高
                font:字体（字体名称,字体大小，字体属性）
            '''
            lable = Label(self.__frame,text=text,width=4,height=2,font=("黑
体",35,"bold"))
            # 绘制文本显示的位置
            lable.grid(row = row,column=col,padx=5,pady=5)
            # 将文本对象添加到存储每行label的list中
            row_list.append(lable)
        # 存储4行4列文本对象
        self.__map_lable.append(row_list)
```

```
def __updateGame(self):
    '''
        更新游戏
    :return:
    '''
    for row in range(len(self.__controller.map)):
        for col in range(len(self.__controller.map[row])):
            # 获取map的数值
            number = self.__controller.map[row][col]
            # 获取数值的lable对象
            label = self.__map_lable[row][col]
            # 设置显示的数值
            label['text'] =  str(number) if number else ""
            # 设置数值的背景色
            label['bg'] = self.__mapcolor[number][0]
            # 设置数值的前景色
            label['fg'] = self.__mapcolor[number][1]
        # 设置分数 TODO


def __restart(self):
    '''
        重新开始游戏，游戏更新
    :return:
    '''
    self.__controller.reset()
    self.__updateGame()


def main(self):
        # 创建组件容器对象
        self.__frame = Frame(self.__root,bg=self.__game_bg_color)
        # self.__frame =
Frame(self.__root,bg=self.__game_bg_color,width=800,height=600)
        # 设置显示位置（默认0行0列位置）
        self.__frame.grid()
        # 初识窗口绘制
        self.__gameScreenDraw()
        # 初始化分数按钮绘制
        self.__gameScoreDraw()
        # 初始化游戏按钮绘制
        self.__gameButtonDraw()
        # 初始化游戏
        self.__restart()
        # 窗口事件循环
        self.__root.mainloop()
```

2.4 绘制按钮、分数（扩展）

```
class GameView:

    def __gameScoreDraw(self):
        '''
```

```
            绘制显示分数
        :return:
        '''

    def __gameButtonDraw(self):
        '''
            绘制游戏重新开始按钮
        :return:
        '''
        restart_button = Button(self.__frame,text="重新开始",\
                                font=("黑体",13,"bold"),height=2,\
                                bg="#E4492D",fg="#fff",command=self.__restart)
        restart_button.grid(row=4,column=3,padx=5,pady=5)

    def main(self):
        # 初始化分数按钮绘制
        self.__gameScoreDraw()
        # 初始化游戏按钮绘制
        self.__gameButtonDraw()
```

2.5 按键操作

思路:

- 将移动函数转移至GameController类中
- 在GameView中添加和移动函数映射的按键
- 添加键盘按下的处理
    - 判断按键是否在[按键与移动函数的映射]中
    - 判断游戏是否结束
    - 弹窗询问：是否继续游戏
        - 是： restart()
        - 否：窗口退出

Class GameController:

```
# 零元素后移
# 有一个列表，从后往前判断，如果是0，将元素删掉，在最后追加一个0
def __zero_to_end(self):
    '''
        将0元素移动到末尾
    :return: None
    '''
    for i in range(len(self.__lists) - 1, -1, -1):
        if self.__lists[i] == 0:
            del self.__lists[i]
            self.__lists.append(0)

def __merge(self):
    '''
        相邻元素合并相加
    :return: None
```

```
    '''
    # 0元素移动到末尾
    self.__zero_to_end()
    for i in range(len(self.__lists) - 1):
        # 判断相邻元素是否相等，且不为0
        if self.__lists[i] == self.__lists[i + 1] and self.__lists[i] != 0:
            # 前一个元素加上后一个元素
            self.__lists[i] += self.__lists[i + 1]
            # 删除后一个元素
            del self.__lists[i + 1]
            # 保证长度一致，追加一个0
            self.__lists.append(0)

def move_left(self):
    '''
        元素左移
    :return: None
    '''
    for line in self.__map:  # line : 表示map的每一行
        self.__lists = line
        self.__merge()

def move_right(self):
    '''
        元素右移
    :return: None
    '''
    for line in self.__map:
        global list_merge
        # 将每一行反向
        self.__lists = line[::-1]
        # 左移合并
        self.__merge()
        # 反向填充
        line[::-1] = self.__lists

def __metrix_deivce(self):
    '''
        矩阵转置
    :return: None
    '''
    for r in range(len(self.__map) - 1):
        for c in range(r + 1, len(self.__map)):
            # print(r,c)
            self.__map[c][r], self.__map[r][c] = self.__map[r][c], self.__map[c]
[r]

def move_up(self):
    '''
        元素上移
    :return: None
    '''
    # map转置
    self.__metrix_deivce()
    # 左移合并
    self.move_left()
    # 转置
    self.__metrix_deivce()
```

```
def move_down(self):
    '''
        元素下移
    :return: None
    '''
    # map转置
    self.__metrix_deivce()
    # 左移合并
    self.move_right()
    # 转置
    self.__metrix_deivce()
```

GameView:

```
# 添加按键和移动函数的映射
self.__keymap = {
    'a' : self.__controller.move_left,
    'd' : self.__controller.move_right,
    'w' : self.__controller.move_up,
    's' : self.__controller.move_down,
    'Left': self.__controller.move_left,
    'Right': self.__controller.move_right,
    'Up': self.__controller.move_up,
    'Down': self.__controller.move_down,
    'q': self.__root.quit
}

    def __on_key_down(self,event):
        '''
            键盘按下的处理
        :param event:
        :return:
        '''
        # 获取键盘按键操作
        keysym = event.keysym
        print("当前的按键操作为：",keysym)
        # 判断按键是否在按键与移动函数的映射当中
        if keysym in self.__keymap:
            # 判断游戏是否结束
            if self.__controller.isGameOver():
                # 弹窗询问游戏是否继续
                mb = messagebox.askyesno(title="游戏结束",message="GameOver!\n是否
继续")
                if mb :
                    # 继续游戏
                    self.__restart()
                else:
                    # 退出游戏
                    self.__root.quit()
            else:
                # 调用对应的移动方法
                self.__keymap[keysym]()
                # 更新游戏
                self.__updateGame()
```

```
                # 只要移动填充一个2或4
                self.__controller.generate_number()
```

完整代码

```
import random
from tkinter import *
# from tkinter import Tk
from tkinter import messagebox


class LocationModel:
    def __init__(self,x,y):
        self.x = x # x轴（行数）
        self.y = y # y轴（列数）


class GameController:
    def __init__(self):
        self.__map = [
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0]
        ]
        # 存放空白位置
        self.__list_empty_location = []

    @property
    def map(self):  # map 是只读模式
        return self.__map

    # 零元素后移
    # 有一个列表，从后往前判断，如果是0，将元素删掉，在最后追加一个0
    def __zero_to_end(self):
        '''
            将0元素移动到末尾
        :return: None
        '''
        for i in range(len(self.__lists) - 1, -1, -1):
            if self.__lists[i] == 0:
                del self.__lists[i]
                self.__lists.append(0)

    def __merge(self):
        '''
            相邻元素合并相加
        :return: None
        '''
        # 0元素移动到末尾
        self.__zero_to_end()
        for i in range(len(self.__lists) - 1):
```

```
            # 判断相邻元素是否相等，且不为0
            if self.__lists[i] == self.__lists[i + 1] and self.__lists[i] != 0:
                # 前一个元素加上后一个元素
                self.__lists[i] += self.__lists[i + 1]
                # 删除后一个元素
                del self.__lists[i + 1]
                # 保证长度一致，追加一个0
                self.__lists.append(0)

    def move_left(self):
        '''
            元素左移
        :return: None
        '''
        for line in self.__map:  # line : 表示map的每一行
            self.__lists = line
            self.__merge()

    def move_right(self):
        '''
            元素右移
        :return: None
        '''
        for line in self.__map:
            global list_merge
            # 将每一行反向
            self.__lists = line[::-1]
            # 左移合并
            self.__merge()
            # 反向填充
            line[::-1] = self.__lists

    def __metrix_deivce(self):
        '''
            矩阵转置
        :return: None
        '''
        for r in range(len(self.__map) - 1):
            for c in range(r + 1, len(self.__map)):
                # print(r,c)
                self.__map[c][r], self.__map[r][c] = self.__map[r][c],
self.__map[c][r]

    def move_up(self):
        '''
            元素上移
        :return: None
        '''
        # map转置
        self.__metrix_deivce()
        # 左移合并
        self.move_left()
        # 转置
        self.__metrix_deivce()

    def move_down(self):
        '''
            元素下移
```

```
        :return: None
        '''
        # map转置
        self.__metrix_deivce()
        # 左移合并
        self.move_right()
        # 转置
        self.__metrix_deivce()

    def generate_number(self):
        # 1.随机位置（计算map中0的位置）存储
        #   清空之前存储的空白位置
        self.__list_empty_location.clear()
        self.__calculate_empty_loction()

        # 2.从空白位置随机选择1个   random.choice
        loc = random.choice(self.__list_empty_location)

        # 3.随机生成一个新的数字 2/4 - 2概率:0.9   4:0.1
        random_number = self.__random_generate_two_four()

        # 4.修改对应随机位置的数，修改为随机的2/4
        self.__map[loc.x][loc.y] = random_number

    def __random_generate_two_four(self):
        ''' 随机生成 2 或 4'''
        return  2 if random.randint(1,10) != 1 else 4

    def __calculate_empty_loction(self):
        ''' 计算空白位置  '''
        for x in range(len(self.__map)): # 行
            for y in range(len(self.__map[x])): # 列
                if self.__map[x][y] == 0:
                    # 将当前这个元素追加到空白位置列表中
                    self.__list_empty_location.append(LocationModel(x,y))

    def isGameOver(self):
        '''
            判断游戏是否结束
        :return: 结束与否
        '''
        # 1. 存在空白位置
        if len(self.__list_empty_location) != 0 :
            return False

        # 2. 可以横纵移动
        for row in range(len(self.__map)):
            for col in range(len(self.__map[row]) - 1):
                if self.__map[row][col] == self.__map[row][col + 1] or \
                        self.__map[col][row] == self.__map[col+1][row]:
                    return False

        return True

    def reset(self):
        '''
            重置：self.__map,成绩，生成2个数字
        :return: None
```

```
        '''
        self.__map = [
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0],
            [0,0,0,0]
        ]
        self.generate_number()
        self.generate_number()

class GameView:
    def __init__(self):
        # 创建窗口对象
        self.__root = Tk()
        # 设置背景颜色
        self.__game_bg_color = '#bbada0'
        # 创建核心业务逻辑对象 -- Controller 对象
        self.__controller = GameController()
        # 存储数值的label文本对象
        self.__map_lable = []
        # 进行界面初始配置
        self.__gameInit()
        # 设置游戏中每个数据对应的位置色块的颜色
        self.__mapcolor = {
            0: ("#cdc1b4", "#776e65"),
            2: ("#eee4da", "#776e65"),
            4: ("#ede0c8", "#f9f6f2"),
            8: ("#f2b179", "#f9f6f2"),
            16: ("#f59563", "#f9f6f2"),
            32: ("#f67c5f", "#f9f6f2"),
            64: ("#f65e3b", "#f9f6f2"),
            128: ("#edcf72", "#f9f6f2"),
            256: ("#edcc61", "#f9f6f2"),
            512: ("#e4c02a", "#f9f6f2"),
            1024: ("#e2ba13", "#f9f6f2"),
            2048: ("#ecc400", "#f9f6f2"),
            4096: ("#ae84a8", "#f9f6f2"),
            8192: ("#b06ca8", "#f9f6f2"),
            # ----其它颜色都与8192相同---------
            2 ** 14: ("#b06ca8", "#f9f6f2"),
            2 ** 15: ("#b06ca8", "#f9f6f2"),
            2 ** 16: ("#b06ca8", "#f9f6f2"),
            2 ** 17: ("#b06ca8", "#f9f6f2"),
            2 ** 18: ("#b06ca8", "#f9f6f2"),
            2 ** 19: ("#b06ca8", "#f9f6f2"),
            2 ** 20: ("#b06ca8", "#f9f6f2")
        }
        # 添加按键和移动函数的映射
        self.__keymap = {
            'a' : self.__controller.move_left,
            'd' : self.__controller.move_right,
            'w' : self.__controller.move_up,
            's' : self.__controller.move_down,
            'Left': self.__controller.move_left,
            'Right': self.__controller.move_right,
            'Up': self.__controller.move_up,
            'Down': self.__controller.move_down,
            'q': self.__root.quit
```

```
        }

    def __gameInit(self):
        '''
            游戏界面初始配置
        :return:
        '''
        # 设置窗口标题
        self.__root.title('2048')
        # 设置窗口大小不可变
        self.__root.resizable(width=False,height=False)

    def __gameScreenDraw(self):
        '''
            绘制显示的数值及区域
        :return:
        '''
        for row in range(len(self.__controller.map)):
            # 存储每一行数值的label对象
            row_list = []
            for col in range(len(self.__controller.map[row])):
                # 获取每个位置的值
                value = self.__controller.map[row][col]
                # 当位置上的值为0 显示空，否则显示对应的值
                text = str(value) if value else ""
                # 构建显示的文本对象
                # Label 置于窗口上的部件，通常用于显示文本或图像
                '''
                    text:显示的字符串
                    width、height:部件的宽高
                    font:字体（字体名称,字体大小，字体属性）
                '''
                lable = Label(self.__frame,text=text,width=4,height=2,font=("黑
体",35,"bold"))
                # 绘制文本显示的位置
                lable.grid(row = row,column=col,padx=5,pady=5)
                # 将文本对象添加到存储每行label的list中
                row_list.append(lable)
            # 存储4行4列文本对象
            self.__map_lable.append(row_list)

    def __gameScoreDraw(self):
        '''
            绘制显示分数
        :return:
        '''


    def __updateGame(self):
        '''
            更新游戏
        :return:
        '''
        for row in range(len(self.__controller.map)):
            for col in range(len(self.__controller.map[row])):
                # 获取map的数值
                number = self.__controller.map[row][col]
                # 获取数值的lable对象
                label = self.__map_lable[row][col]
```

```
                # 设置显示的数值
                label['text'] =  str(number) if number else ""
                # 设置数值的背景色
                label['bg'] = self.__mapcolor[number][0]
                # 设置数值的前景色
                label['fg'] = self.__mapcolor[number][1]
            # 设置分数 TODO


    def __restart(self):
        '''
            重新开始游戏，游戏更新
        :return:
        '''
        self.__controller.reset()
        self.__updateGame()

    def __gameButtonDraw(self):
        '''
            绘制游戏重新开始按钮
        :return:
        '''
        restart_button = Button(self.__frame,text="重新开始",\
                                font=("黑体",13,"bold"),height=2,\
                                bg="#E4492D",fg="#fff",command=self.__restart)
        restart_button.grid(row=4,column=3,padx=5,pady=5)

    def __on_key_down(self,event):
        '''
            键盘按下的处理
        :param event:
        :return:
        '''
        # 获取键盘按键操作
        keysym = event.keysym
        print("当前的按键操作为：",keysym)
        # 判断按键是否在按键与移动函数的映射当中
        if keysym in self.__keymap:
            # 判断游戏是否结束
            if self.__controller.isGameOver():
                # 弹窗询问游戏是否继续
                mb = messagebox.askyesno(title="游戏结束",message="GameOver!\n是否
继续")
                if mb :
                    # 继续游戏
                    self.__restart()
                else:
                    # 退出游戏
                    self.__root.quit()
            else:
                # 调用对应的移动方法
                self.__keymap[keysym]()
                # 更新游戏
                self.__updateGame()
                # 只要移动填充一个2或4
                self.__controller.generate_number()
```

```
    def main(self):
        # 创建组件容器对象
        self.__frame = Frame(self.__root,bg=self.__game_bg_color)
        # self.__frame =
Frame(self.__root,bg=self.__game_bg_color,width=800,height=600)
        # 设置显示位置（默认0行0列位置）
        self.__frame.grid()
        # 初识窗口绘制
        self.__gameScreenDraw()
        # 初始化分数按钮绘制
        self.__gameScoreDraw()
        # 初始化游戏按钮绘制
        self.__gameButtonDraw()
        # 初始化游戏
        self.__restart()
        # 设置焦点
        self.__frame.focus_set()
        # 控件获取检点，处理按键事件
        self.__frame.bind("<Key>",self.__on_key_down)
        # 窗口事件循环
        self.__root.mainloop()

if __name__ == '__main__':
    view = GameView()
    view.main()
```

要求

对2048进行拆分、扩展

拆分：拆成3个模块，

M:Module 模型

V: View视图

C:Controller控制器（核心逻辑功能）

扩展：添加新的功能（2个起步）

例如：分数、音乐、最高分、暂停等

答辩：

PPT、代码