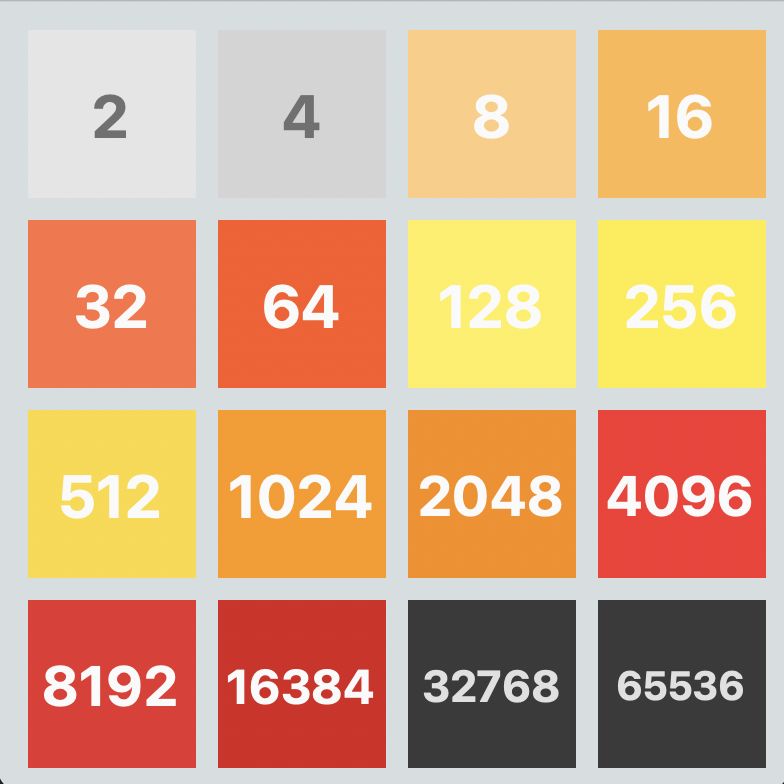
首发于[编程笔记](https://www.zhihu.com/column/i-code)



**Python练习：200行代码实现一个2048游戏**

[JonCode](https://www.zhihu.com/people/code2020)

[JonCode](https://www.zhihu.com/people/code2020)

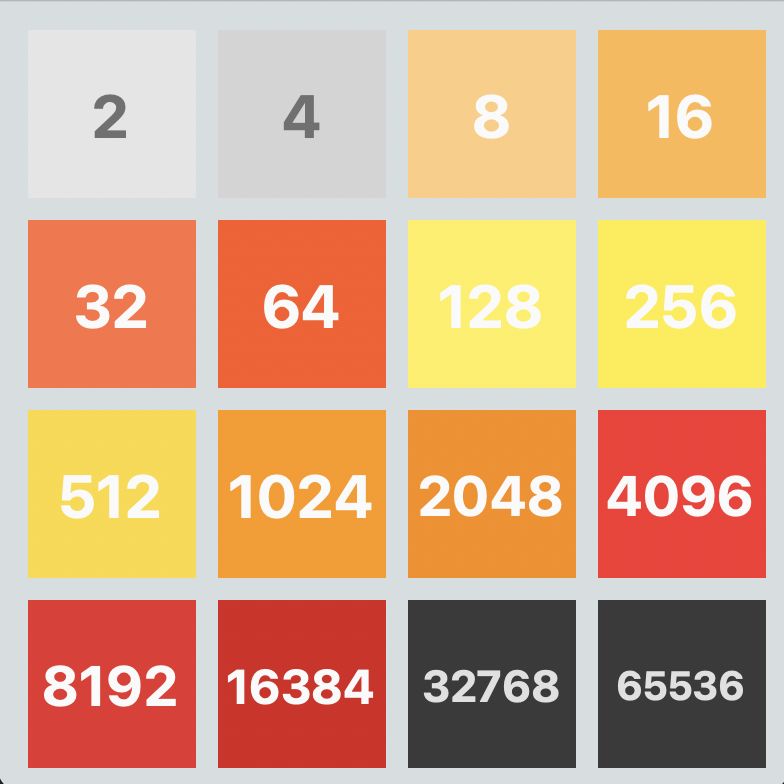
由于部分童鞋对本文产生了一些疑问，所以在原教程的基础上进行了重置，添加了图形界面，并增加了部分解释，希望能帮到大家。增加图形界面后代码行数大概不到200多行（后续又对代码进行了重构，并增加了得分、返回上一步等功能，代码量略有增加）。  
本教程基于Python3.9

一、**任务目标：实现一个有图形界面的2048游戏**

**主要实现功能：**

* 判断游戏是否结束；
* 在随机的空白位置生成新的数字；
* 接收键盘输入并在上下左右四个方向上移动数字并合并相同的数字；
* 计算得分；
* 撤销操作（返回上一步）；
* 游戏结束时增加重置游戏选项；

**完成效果预览：**

Python 2048游戏

二、**知识基础**

需要以下知识才能完全理解本文

1. python基础知识：数据类型、变量、list、循环和判断等；
2. python进阶知识：函数、切片、迭代以及类的基础用法等；
3. random模块中的几个简单函数；
4. tkinter模块基础知识，如果知识想要做一个命令行下的2048游戏则不需要本部分；

#以廖雪峰python教程为例的话学到【面向对象编程】就可以自己做起来没什么压力了。

三、**游戏设计思路**

首先要有一个用于我们存储阵列数组（也就是游戏里要显示的那些数字）的东西，我选择的是嵌套list，例如[[2,0,2],[0,2,2],[2,0,4]]。

其次难点在于如何实现四个方向滑动的计算逻辑，以向左为例要实现，原始数据如下图式①，首先可以将0去掉变为式②，再逐行检测相邻的数字是否有相同的，如果有相同的数字则合并数字变为式③，再左移得到式④，最后再补0得到最终结果式⑤。

代码示例如下：

matrix = [[2,0,2],[0,0,2],[2,0,4]]

# 数字左移

def left\_():

for row in range(3):

# 先去掉0

while 0 in matrix[row]:

matrix[row].remove(0)

# 再补齐0

while len(matrix[row]) != 3:

matrix[row].append(0)

left\_()

# 数字向左合并，注意合并完成后会产生0，所以这里先左移再合并再左移

for row in range(3):

for col in range(2):

if matrix[row][col] == matrix[row][col+1] and matrix[row][col] != 0:

matrix[row][col] = 2 \* matrix[row][col]

matrix[row][col+1] = 0

left\_()

上面我们完成了数字的左移合并，那其他方向上该怎么做呢？最直接的方法就是以上面的这种思路修改上面的代码为其他方向上的操作，其实也有简单的方法，右移合并其实可以看作左移合并的镜像操作，如下图假设我们原数组为式①，我们只需要先将原数组反转变为式②，再将数组左移变为式③，最后再将数组反转得到式④，这样就通过左移的方法再加上少量操作即可完成右移，可以看出通过这样操作得出的结果和直接重新设计逻辑的出的结果是完全相同的，但是这样可以一定程度上简化代码量。

同样上移我们可以看作将矩阵转置(第一行变为第一列，第二行变为第二列，以此类推)后进行左移再转置得到的结果 ，下移我们可以看作上移的反转操作，过程可以看下图就不再过多解释。

四、**代码框架**

核心功能的思路我们整理完成了，接下来我们根据功能要求设计代码框架。

将业务逻辑及GUI设计进行分离，便于后续修改。

代码框架如下：

class Matrix2048():

'''用于实现核心逻辑'''

def \_\_init\_\_(self, column: int = 4):

self.column = column

self.matrix = [[0 for i in range(column)] for li in range(column)]

self.history = []

self.score = 0

# 生成新的数字

def generate\_number(self):

pass

# 判断游戏是否结束 结束则返回True， 否则返回False

def gameover(self) -> bool:

pass

# 游戏初始化 用于新建游戏或重置游戏

def init(self):

pass

# 移动合并 并记录历史数据

def matrix\_move(self, direction):

pass

# 向左移动合并

def move\_left(self):

pass

# 向右移动合并

def move\_right(self):

pass

# 向上移动合并

def move\_up(self):

pass

# 向下移动合并

def move\_down(self):

pass

# 返回上一步

def prev\_step(self):

pass

class Window2048():

def \_\_init\_\_(self, column: int = 4):

pass

# 配置初始化

def init\_setting(self, column):

pass

# 窗口初始化

def init\_root(self):

pass

# 更新UI数据

def update\_ui(self):

pass

# 事件循环 接受键盘输入

def key\_event(self, event):

pass

# 重置游戏

def reset\_game(self):

pass

# 主程序

def main(self):

pass

**四、程序设计**

**下面进行每个方法的具体实现**

* **判断游戏是否结束**

# 判断游戏是否结束 结束则返回True， 否则返回False

def gameover(self) -> bool:

# 1. 矩阵(self.matrix)中有0表示还有空位，则必定未结束

if 0 in [i for li in self.matrix for i in li]:

return False

# 2. 水平方向上有相同的数字则未结束，注意下标不要越界

for row in range(self.column):

for col in range(self.column-1):

if self.matrix[row][col] == self.matrix[row][col+1]:

return False

# 3. 垂直方向上有相同的数字则未结束

for row in range(self.column-1):

for col in range(self.column):

if self.matrix[row][col] == self.matrix[row+1][col]:

return False

# 以上条件都不满足则证明即没有空位也没有相邻的相同的数字，则游戏结束返回True

return True

* **生成数字**

# 生成新的数字

def generate\_number(self):

matrix = self.matrix

column = self.column

# 找出所有为0的位置

zero = [(x, y) for x in range(column)

for y in range(column) if matrix[x][y] == 0]

# 随机选择一个为0的位置并填充为随机数字

if zero != []:

x, y = random.choice(zero)

matrix[x][y] = random.choice([2, 2])

* **游戏初始化**

# 游戏初始化 用于新建游戏或重置游戏

def init(self):

self.matrix = [[0 for x in range(self.column)]

for y in range(self.column)]

self.generate\_number()

self.generate\_number()

self.history = []

self.score = 0

* **四个方向的移动合并操作**

# 本部分不理解的看一下设计思路部分

# 整合移动合并接口 并记录历史数据

def matrix\_move(self, direction):

if direction in ['L', 'R', 'D', 'U']:

# 记录历史

prev\_step = {

'score': copy.deepcopy(self.score),

'matrix': copy.deepcopy(self.matrix)

}

self.history.append(prev\_step)

# 只记录最近10次

if len(self.history) > 10:

self.history = self.history[-10:]

if direction == 'U':

self.move\_up()

if direction == 'D':

self.move\_down()

if direction == 'L':

self.move\_left()

if direction == 'R':

self.move\_right()

# 向左移动合并

def move\_left(self):

column = self.column

matrix = self.matrix

# 数字左移

def move\_left\_(matrix):

for row in range(column):

while 0 in matrix[row]:

matrix[row].remove(0)

while len(matrix[row]) != column:

matrix[row].append(0)

return matrix

# 数字向左合并

def merge\_left(matrix):

for row in range(column):

for col in range(column-1):

if matrix[row][col] == matrix[row][col+1] and matrix[row][col] != 0:

matrix[row][col] = 2 \* matrix[row][col]

matrix[row][col+1] = 0

self.score = self.score + matrix[row][col]

return matrix

matrix = move\_left\_(matrix)

matrix = merge\_left(matrix)

self.matrix = move\_left\_(matrix)

# 向右移动合并

def move\_right(self):

self.matrix = [li[::-1] for li in self.matrix]

self.move\_left()

self.matrix = [li[::-1] for li in self.matrix]

# 向上移动合并

def move\_up(self):

column = self.column

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

self.move\_left()

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

# 向下移动合并

def move\_down(self):

self.matrix = self.matrix[::-1]

self.move\_up()

self.matrix = self.matrix[::-1]

* **返回上一步**

# 返回上一步

def prev\_step(self):

if self.history:

prev\_data = self.history[-1]

self.score = prev\_data['score']

self.matrix = prev\_data['matrix']

self.history = self.history[0:-1]

* **数据处理部分整体代码**

import copy

import random

class Matrix2048():

def \_\_init\_\_(self, column: int = 4):

self.column = column

self.matrix = [[0 for i in range(column)] for li in range(column)]

self.history = []

self.score = 0

self.init()

# 生成新的数字

def generate\_number(self):

matrix = self.matrix

column = self.column

# 找出所有为0的位置

zero = [(x, y) for x in range(column)

for y in range(column) if matrix[x][y] == 0]

# 随机选择一个为0的位置并填充为随机数字

if zero != []:

x, y = random.choice(zero)

matrix[x][y] = random.choice([2, 2])

# 判断游戏是否结束 结束则返回True， 否则返回False

def gameover(self) -> bool:

matrix = self.matrix

column = self.column

# 1. 矩阵(self.matrix)中有0则未结束

if 0 in [i for li in matrix for i in li]:

return False

# 2. 水平方向上有相同的数字则未结束

for row in range(column):

for col in range(column-1):

if matrix[row][col] == matrix[row][col+1]:

return False

# 3. 垂直方向上有相同的数字则未结束

for row in range(column-1):

for col in range(column):

if matrix[row][col] == matrix[row+1][col]:

return False

return True

# 游戏初始化 用于新建游戏或重置游戏

def init(self):

self.matrix = [[0 for x in range(self.column)]

for y in range(self.column)]

self.generate\_number()

self.generate\_number()

self.history = []

self.score = 0

# 移动合并 并记录历史数据

def matrix\_move(self, direction):

if direction in ['L', 'R', 'D', 'U']:

# 记录历史

prev\_step = {

'score': copy.deepcopy(self.score),

'matrix': copy.deepcopy(self.matrix)

}

self.history.append(prev\_step)

# 只记录最近10次

if len(self.history) > 10:

self.history = self.history[-10:]

if direction == 'U':

self.move\_up()

if direction == 'D':

self.move\_down()

if direction == 'L':

self.move\_left()

if direction == 'R':

self.move\_right()

# 向左移动合并

def move\_left(self):

column = self.column

matrix = self.matrix

# 数字左移

def move\_left\_(matrix):

for row in range(column):

while 0 in matrix[row]:

matrix[row].remove(0)

while len(matrix[row]) != column:

matrix[row].append(0)

return matrix

# 数字向左合并

def merge\_left(matrix):

for row in range(column):

for col in range(column-1):

if matrix[row][col] == matrix[row][col+1] and matrix[row][col] != 0:

matrix[row][col] = 2 \* matrix[row][col]

matrix[row][col+1] = 0

self.score = self.score + matrix[row][col]

return matrix

matrix = move\_left\_(matrix)

matrix = merge\_left(matrix)

self.matrix = move\_left\_(matrix)

# 向右移动合并

def move\_right(self):

self.matrix = [li[::-1] for li in self.matrix]

self.move\_left()

self.matrix = [li[::-1] for li in self.matrix]

# 向上移动合并

def move\_up(self):

column = self.column

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

self.move\_left()

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

# 向下移动合并

def move\_down(self):

self.matrix = self.matrix[::-1]

self.move\_up()

self.matrix = self.matrix[::-1]

# 返回上一步

def prev\_step(self):

if self.history:

prev\_data = self.history[-1]

self.score = prev\_data['score']

self.matrix = prev\_data['matrix']

self.history = self.history[0:-1]

# 命令行显示

def show(self):

r = '+-----' \* self.column + '+\n'

for li in self.matrix:

for i in li:

s = '|' + ' '\*5 if i == 0 else '|' + str(i).center(5, ' ')

r = r + s

r = r + '|\n' + '+-----' \* self.column + '+\n'

print(r)

上面完成了整体的数据处理部分的设计，下面进行界面设计，使用的是Python自带的图形库Tkinter。

* **UI界面初始化**

# 初始化样式信息，如背景色，字体颜色，字体大小等，这里参考原版的配色的同时进行了一定的修改

# 配置初始化

def init\_setting(self, column):

# 游戏的棋盘的大小，默认为4\*4

self.column = column

# 棋盘中格子的间隔，单位为px

self.space\_size = 12

# 棋盘中格子的大小，单位为px

self.cell\_size = 80

# 用于存储tkinter.Lable对象

self.emts = [] # 存储lable对象

# 用于存储游戏的样式信息，如背景色，字体颜色，字体大小等

self.style = {

'page': {'bg': '#d6dee0', },

# 0 ~ 4 灰色系 bg 背景颜色， fg字体颜色， fz字体大小

0: {'bg': '#EEEEEE', 'fg': '#EEEEEE', 'fz': 30},

2\*\*1: {'bg': '#E5E5E5', 'fg': '#707070', 'fz': 30},

2\*\*2: {'bg': '#D4D4D4', 'fg': '#707070', 'fz': 30},

# 8 ～ 16 橙色系

2\*\*3: {'bg': '#FFCC80', 'fg': '#FAFAFA', 'fz': 30},

2\*\*4: {'bg': '#FFB74D', 'fg': '#FAFAFA', 'fz': 30},

# 32 ～ 64 红色系

2\*\*5: {'bg': '#FF7043', 'fg': '#FAFAFA', 'fz': 30},

2\*\*6: {'bg': '#FF5722', 'fg': '#FAFAFA', 'fz': 30},

# 128～2048 黄色系

2\*\*7: {'bg': '#FFEE58', 'fg': '#FAFAFA', 'fz': 30},

2\*\*8: {'bg': '#FFEB3B', 'fg': '#FAFAFA', 'fz': 30},

2\*\*9: {'bg': '#FDD835', 'fg': '#FAFAFA', 'fz': 30},

# 1024~2048 橙色系

2\*\*10: {'bg': '#FF9800', 'fg': '#FAFAFA', 'fz': 30},

2\*\*11: {'bg': '#FB8C00', 'fg': '#FAFAFA', 'fz': 28},

# 4096 + 红色系

2\*\*12: {'bg': '#fb3030', 'fg': '#FAFAFA', 'fz': 28},

2\*\*13: {'bg': '#e92e2e', 'fg': '#FAFAFA', 'fz': 28},

2\*\*14: {'bg': '#da1e1e', 'fg': '#FAFAFA', 'fz': 24},

# 2\*\*15 + 黑色 超过2\*\*15颜色不再改变

2\*\*15: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 22},

2\*\*16: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*17: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*18: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*19: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 18},

2\*\*20: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 17},

2\*\*21: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 16},

2\*\*22: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 15},

2\*\*23: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 14},

2\*\*24: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 13},

2\*\*25: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 12},

}

* **创建窗口**

# 窗口初始化

def init\_root(self):

column = self.column

space\_size = self.space\_size

cell\_size = self.cell\_size

# 创建根窗口

root = tkinter.Tk()

root.title('2048')

# 根窗口尺寸设置

window\_w = column \* (space\_size + cell\_size) + space\_size

window\_h = window\_w + cell\_size + 2 \* space\_size

root.geometry(f'{window\_w}x{window\_h}')

# 顶栏

header\_h = cell\_size + space\_size \* 2

header = tkinter.Frame(root, height=header\_h, width=window\_w)

self.init\_header(header)

# 棋盘

table = tkinter.Frame(root, height=window\_w, width=window\_w)

self.init\_table(table)

return root

# 顶栏初始化

def init\_header(self, master):

master['bg'] = self.style['page']['bg']

# 分数

emt\_score = tkinter.Label(master, bd=0)

emt\_score['fg'] = '#707070'

emt\_score['bg'] = self.style['page']['bg']

emt\_score['font'] = ("黑体", 30, "bold")

img = Image.new('RGB', (self.cell\_size, self.cell\_size),

self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

emt\_score.configure(image=img)

emt\_score['image'] = img

emt\_score['text'] = 'score:' + str(self.data.score)

emt\_score['compound'] = 'center'

self.emt\_score = emt\_score

emt\_score.place(x=15, y=15)

master.pack()

# 棋盘初始化

def init\_table(self, master):

column = self.column

cell\_size = self.cell\_size

space\_size = self.space\_size

master['bg'] = self.style['page']['bg']

# 创建棋盘子元素

emts = [[0 for x in range(column)] for y in range(column)]

for row in range(column):

for col in range(column):

emt = tkinter.Label(master, bd=0)

emt['width'] = self.cell\_size

emt['height'] = self.cell\_size

emt['text'] = ''

emt['compound'] = 'center'

y = space\_size + (cell\_size + space\_size) \* row

x = space\_size + (cell\_size + space\_size) \* col

emt.place(x=x, y=y)

emts[row][col] = emt

self.emts = emts

master.pack()

* **UI界面更新**

# 更新UI数据

def update\_ui(self):

# 更新分数

def update\_score():

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

self.emt\_score.configure(image=img)

self.emt\_score['image'] = img

self.emt\_score['text'] = 'score:' + str(self.data.score)

update\_score()

# 更新棋盘数据

matrix = self.data.matrix

for row in range(self.column):

for col in range(self.column):

num = matrix[row][col]

emt = self.emts[row][col]

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style[num]['bg'])

img = ImageTk.PhotoImage(img)

emt.configure(image=img)

emt['fg'] = self.style[num]['fg']

emt['bg'] = self.style[num]['bg']

emt['image'] = img

emt['font'] = ("黑体", self.style[num]['fz'], "bold")

emt['text'] = str(num) if num != 0 else ''

* **事件循环**

# 事件循环 根据键盘输入移动数字并更新界面显示，然后判断游戏是否结束，

def key\_event(self, event):

# print(f"键盘输入:{ event.keysym }, ASCII码:{ event.keycode }")

# 注意：在不同系统下的同一按键的keycode可能不同

if event.keysym in ['Up', 'w', 'Down', 's', 'Left', 'a', 'Right', 'd']:

if event.keysym in ['Up', 'w']: # 向上

self.data.matrix\_move('U')

elif event.keysym in ['Down', 's']: # 向下

self.data.matrix\_move('D')

elif event.keysym in ['Left', 'a']: # 向左

self.data.matrix\_move('L')

elif event.keysym in ['Right', 'd']: # 向右

self.data.matrix\_move('R')

self.data.generate\_number()

# 按z 返回上一步

if event.keysym == 'z':

if self.data.history != []:

self.data.prev\_step()

self.update\_ui()

# 重置游戏

def reset\_game():

self.t = 0

self.data.init()

self.update\_ui()

if self.data.gameover() is True:

# 直接结束显式存在问题，显示的是上次的数据

if self.t == 0:

self.t = 1

else:

res = tkinter.messagebox.askyesno(

title="2048", message="Game Over!\n是否重新开始!")

if res is True:

reset\_game()

else:

self.root.quit()

# 事件循环，根据键盘输入移动数字并更新界面显示，然后判断游戏是否结束，

def key\_event(self, event):

# 这里的event对象是绑定键盘事件的时候传入的，event.keycode表示的是按下的不同按键的ascii代码，可以通过下行代码测试按键的具体代码

# print(f"键盘输入:{ event.char }, ASCII码:{ event.keycode }")

if event.keycode in [8320768, 8255233, 8124162, 8189699]:

if event.keycode == 8320768: # 向上

self.move\_up()

elif event.keycode == 8255233: # 向下

self.move\_down()

elif event.keycode == 8124162: # 向左

self.move\_left()

elif event.keycode == 8189699: # 向右

self.move\_right()

# 每次有效输入后自动添加一个数字

# 这里在评论区有人提出有bug，提出说不能移动的时候能移动

# 其实是程序设计的问题，假如棋盘中只有左下角有一个数字，那么按下键的时候这个数字不会移动，但是会新生成一个数字，我们这里没有对不能移动的情况下是否添加数字做鉴别，所以会出现这种情况

self.generator()

# 更新UI界面

self.update\_ui()

# 判断游戏是否结束

if self.gameover() is True:

# 如果结束则弹窗提示是否重新开始

res = tkinter.messagebox.askyesno(

title="2048", message="Game Over!\n是否重新开始!")

# 如果用户点击了弹窗提示中的yes则重新开始游戏，否则退出游戏

if res is True:

self.reset()

self.generator()

self.generator()

self.update\_ui()

else:

self.root.quit()

* **主程序**

# 主程序

def main(self):

root = self.init\_ui()

self.root = root

# 绑定键盘事件

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

# 主循环，这个必须要有，用于保持窗口，缺失的话窗口只会闪一下消失

root.mainloop()

* **命令行显示**

# 如果你想使用命令行显示游戏也可以通过这个函数来进行显示

# 不过就不能使用按键操作了，可以使用input函数来进行输入接收

# 也可以用keyboard等其他模块来进行接收键盘操作

def show(self):

r = '+-----' \* self.column + '+\n'

for li in self.matrix:

for i in li:

s = '|' + ' '\*5 if i == 0 else '|' + str(i).center(5, ' ')

r = r + s

r = r + '|\n' + '+-----' \* self.column + '+\n'

print(r)

自此设计基本完成，下面将UI部分重新整理如下，将部分过长的函数进行拆分重构。

class Window2048():

def \_\_init\_\_(self, column: int = 4):

self.init\_setting(column)

self.data = Matrix2048(column)

self.root = self.init\_root()

self.t = 0 # 判断游戏结束时用

self.main()

# 配置初始化

def init\_setting(self, column):

# 游戏的棋盘的大小，默认为4\*4

self.column = column

# 棋盘中格子的间隔，单位为px

self.space\_size = 12

# 棋盘中格子的大小，单位为px

self.cell\_size = 80

# 用于存储tkinter.Lable对象

self.emts = [] # 存储lable对象

# 用于存储游戏的样式信息，如背景色，字体颜色，字体大小等

self.style = {

'page': {'bg': '#d6dee0', },

# 0 ~ 4 灰色系 bg 背景颜色， fg字体颜色， fz字体大小

0: {'bg': '#EEEEEE', 'fg': '#EEEEEE', 'fz': 30},

2\*\*1: {'bg': '#E5E5E5', 'fg': '#707070', 'fz': 30},

2\*\*2: {'bg': '#D4D4D4', 'fg': '#707070', 'fz': 30},

# 8 ～ 16 橙色系

2\*\*3: {'bg': '#FFCC80', 'fg': '#FAFAFA', 'fz': 30},

2\*\*4: {'bg': '#FFB74D', 'fg': '#FAFAFA', 'fz': 30},

# 32 ～ 64 红色系

2\*\*5: {'bg': '#FF7043', 'fg': '#FAFAFA', 'fz': 30},

2\*\*6: {'bg': '#FF5722', 'fg': '#FAFAFA', 'fz': 30},

# 128～2048 黄色系

2\*\*7: {'bg': '#FFEE58', 'fg': '#FAFAFA', 'fz': 30},

2\*\*8: {'bg': '#FFEB3B', 'fg': '#FAFAFA', 'fz': 30},

2\*\*9: {'bg': '#FDD835', 'fg': '#FAFAFA', 'fz': 30},

# 1024~2048 橙色系

2\*\*10: {'bg': '#FF9800', 'fg': '#FAFAFA', 'fz': 30},

2\*\*11: {'bg': '#FB8C00', 'fg': '#FAFAFA', 'fz': 28},

# 4096 + 红色系

2\*\*12: {'bg': '#fb3030', 'fg': '#FAFAFA', 'fz': 28},

2\*\*13: {'bg': '#e92e2e', 'fg': '#FAFAFA', 'fz': 28},

2\*\*14: {'bg': '#da1e1e', 'fg': '#FAFAFA', 'fz': 24},

# 2\*\*15 + 黑色 超过2\*\*15颜色不再改变

2\*\*15: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 22},

2\*\*16: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*17: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*18: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*19: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 18},

2\*\*20: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 17},

2\*\*21: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 16},

2\*\*22: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 15},

2\*\*23: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 14},

2\*\*24: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 13},

2\*\*25: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 12},

}

# 窗口初始化

def init\_root(self):

column = self.column

space\_size = self.space\_size

cell\_size = self.cell\_size

# 创建根窗口

root = tkinter.Tk()

root.title('2048')

# 根窗口尺寸设置

window\_w = column \* (space\_size + cell\_size) + space\_size

window\_h = window\_w + cell\_size + 2 \* space\_size

root.geometry(f'{window\_w}x{window\_h}')

# 顶栏

header\_h = cell\_size + space\_size \* 2

header = tkinter.Frame(root, height=header\_h, width=window\_w)

self.init\_header(header)

# 棋盘

table = tkinter.Frame(root, height=window\_w, width=window\_w)

self.init\_table(table)

return root

# 顶栏初始化

def init\_header(self, master):

master['bg'] = self.style['page']['bg']

# 分数

emt\_score = tkinter.Label(master, bd=0)

emt\_score['fg'] = '#707070'

emt\_score['bg'] = self.style['page']['bg']

emt\_score['font'] = ("黑体", 30, "bold")

img = Image.new('RGB', (self.cell\_size, self.cell\_size),

self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

emt\_score.configure(image=img)

emt\_score['image'] = img

emt\_score['text'] = 'score:' + str(self.data.score)

emt\_score['compound'] = 'center'

self.emt\_score = emt\_score

emt\_score.place(x=15, y=15)

master.pack()

# 棋盘初始化

def init\_table(self, master):

column = self.column

cell\_size = self.cell\_size

space\_size = self.space\_size

master['bg'] = self.style['page']['bg']

# 创建棋盘子元素

emts = [[0 for x in range(column)] for y in range(column)]

for row in range(column):

for col in range(column):

emt = tkinter.Label(master, bd=0)

emt['width'] = self.cell\_size

emt['height'] = self.cell\_size

emt['text'] = ''

emt['compound'] = 'center'

y = space\_size + (cell\_size + space\_size) \* row

x = space\_size + (cell\_size + space\_size) \* col

emt.place(x=x, y=y)

emts[row][col] = emt

self.emts = emts

master.pack()

# 更新UI数据

def update\_ui(self):

def update\_score():

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

self.emt\_score.configure(image=img)

self.emt\_score['image'] = img

self.emt\_score['text'] = 'score:' + str(self.data.score)

update\_score()

matrix = self.data.matrix

for row in range(self.column):

for col in range(self.column):

num = matrix[row][col]

emt = self.emts[row][col]

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style[num]['bg'])

img = ImageTk.PhotoImage(img)

emt.configure(image=img)

emt['fg'] = self.style[num]['fg']

emt['bg'] = self.style[num]['bg']

emt['image'] = img

emt['font'] = ("黑体", self.style[num]['fz'], "bold")

emt['text'] = str(num) if num != 0 else ''

# 事件循环

def key\_event(self, event):

# print(f"键盘输入:{ event.keysym }, ASCII码:{ event.keycode }")

# 注意：在不同系统下的同一按键的keycode可能不同

if event.keysym in ['Up', 'w', 'Down', 's', 'Left', 'a', 'Right', 'd']:

if event.keysym in ['Up', 'w']: # 向上

self.data.matrix\_move('U')

elif event.keysym in ['Down', 's']: # 向下

self.data.matrix\_move('D')

elif event.keysym in ['Left', 'a']: # 向左

self.data.matrix\_move('L')

elif event.keysym in ['Right', 'd']: # 向右

self.data.matrix\_move('R')

self.data.generate\_number()

# 按z 返回上一步

if event.keysym == 'z':

if self.data.history != []:

self.data.prev\_step()

self.update\_ui()

# 重置游戏

def reset\_game():

self.t = 0

self.data.init()

self.update\_ui()

if self.data.gameover() is True:

# 直接结束显式存在问题，显示的是上次的数据

if self.t == 0:

self.t = 1

else:

res = tkinter.messagebox.askyesno(

title="2048", message="Game Over!\n是否重新开始!")

if res is True:

reset\_game()

else:

self.root.quit()

# 重置游戏

def reset\_game(self):

self.t = 0

self.data.init()

self.update\_ui()

# 主程序

def main(self):

self.update\_ui()

# 绑定键盘事件

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

# 主循环

self.root.mainloop()

**五、程序扩展**

对方的这个小游戏其实有很多可以扩展的地方，比如模式选择（4\*4，5\*5），难度选择，主题配色等等，接下来我给大家一个思路，有兴趣的可以自己扩充。

模式选择：这个比较简单，我们已经设置了self.column属性，只要添加相关UI并处理相关事件即可，这也是为什么我是用self.column属性来设置棋盘规模的原因，不是写死的，很容易即可扩充功能。

主题配色：这个也跟上面的模式选择一样，只要变更self.style属性即可。

难度选择：可以考虑从每次出现的数字个数和数字大小上去修改，也很简单。

以下是完整的代码，供大家参考：

import copy

import random

import tkinter # UI模块

import tkinter.messagebox # UI弹窗

from PIL import ImageTk, Image # 图片处理

class Matrix2048():

def \_\_init\_\_(self, column: int = 4):

self.column = column

self.matrix = [[0 for i in range(column)] for li in range(column)]

self.history = []

self.score = 0

self.init()

# 生成新的数字

def generate\_number(self):

matrix = self.matrix

column = self.column

# 找出所有为0的位置

zero = [(x, y) for x in range(column)

for y in range(column) if matrix[x][y] == 0]

# 随机选择一个为0的位置并填充为随机数字

if zero != []:

x, y = random.choice(zero)

matrix[x][y] = random.choice([2, 2])

# 判断游戏是否结束 结束则返回True， 否则返回False

def gameover(self) -> bool:

matrix = self.matrix

column = self.column

# 1. 矩阵(self.matrix)中有0则未结束

if 0 in [i for li in matrix for i in li]:

return False

# 2. 水平方向上有相同的数字则未结束

for row in range(column):

for col in range(column-1):

if matrix[row][col] == matrix[row][col+1]:

return False

# 3. 垂直方向上有相同的数字则未结束

for row in range(column-1):

for col in range(column):

if matrix[row][col] == matrix[row+1][col]:

return False

return True

# 游戏初始化 用于新建游戏或重置游戏

def init(self):

self.matrix = [[0 for x in range(self.column)]

for y in range(self.column)]

self.generate\_number()

self.generate\_number()

self.history = []

self.score = 0

# 移动合并 并记录历史数据

def matrix\_move(self, direction):

if direction in ['L', 'R', 'D', 'U']:

# 记录历史

prev\_step = {

'score': copy.deepcopy(self.score),

'matrix': copy.deepcopy(self.matrix)

}

self.history.append(prev\_step)

# 只记录最近10次

if len(self.history) > 10:

self.history = self.history[-10:]

if direction == 'U':

self.move\_up()

if direction == 'D':

self.move\_down()

if direction == 'L':

self.move\_left()

if direction == 'R':

self.move\_right()

# 向左移动合并

def move\_left(self):

column = self.column

matrix = self.matrix

# 数字左移

def move\_left\_(matrix):

for row in range(column):

while 0 in matrix[row]:

matrix[row].remove(0)

while len(matrix[row]) != column:

matrix[row].append(0)

return matrix

# 数字向左合并

def merge\_left(matrix):

for row in range(column):

for col in range(column-1):

if matrix[row][col] == matrix[row][col+1] and matrix[row][col] != 0:

matrix[row][col] = 2 \* matrix[row][col]

matrix[row][col+1] = 0

self.score = self.score + matrix[row][col]

return matrix

matrix = move\_left\_(matrix)

matrix = merge\_left(matrix)

self.matrix = move\_left\_(matrix)

# 向右移动合并

def move\_right(self):

self.matrix = [li[::-1] for li in self.matrix]

self.move\_left()

self.matrix = [li[::-1] for li in self.matrix]

# 向上移动合并

def move\_up(self):

column = self.column

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

self.move\_left()

self.matrix = [[self.matrix[x][y]

for x in range(column)] for y in range(column)]

# 向下移动合并

def move\_down(self):

self.matrix = self.matrix[::-1]

self.move\_up()

self.matrix = self.matrix[::-1]

# 返回上一步

def prev\_step(self):

if self.history:

prev\_data = self.history[-1]

self.score = prev\_data['score']

self.matrix = prev\_data['matrix']

self.history = self.history[0:-1]

# 命令行显示

def show(self):

r = '+-----' \* self.column + '+\n'

for li in self.matrix:

for i in li:

s = '|' + ' '\*5 if i == 0 else '|' + str(i).center(5, ' ')

r = r + s

r = r + '|\n' + '+-----' \* self.column + '+\n'

print(r)

class Window2048():

def \_\_init\_\_(self, column: int = 4):

self.init\_setting(column)

self.data = Matrix2048(column)

self.root = self.init\_root()

self.t = 0 # 判断游戏结束时用

self.main()

# 配置初始化

def init\_setting(self, column):

# 游戏的棋盘的大小，默认为4\*4

self.column = column

# 棋盘中格子的间隔，单位为px

self.space\_size = 12

# 棋盘中格子的大小，单位为px

self.cell\_size = 80

# 用于存储tkinter.Lable对象

self.emts = [] # 存储lable对象

# 用于存储游戏的样式信息，如背景色，字体颜色，字体大小等

self.style = {

'page': {'bg': '#d6dee0', },

# 0 ~ 4 灰色系 bg 背景颜色， fg字体颜色， fz字体大小

0: {'bg': '#EEEEEE', 'fg': '#EEEEEE', 'fz': 30},

2\*\*1: {'bg': '#E5E5E5', 'fg': '#707070', 'fz': 30},

2\*\*2: {'bg': '#D4D4D4', 'fg': '#707070', 'fz': 30},

# 8 ～ 16 橙色系

2\*\*3: {'bg': '#FFCC80', 'fg': '#FAFAFA', 'fz': 30},

2\*\*4: {'bg': '#FFB74D', 'fg': '#FAFAFA', 'fz': 30},

# 32 ～ 64 红色系

2\*\*5: {'bg': '#FF7043', 'fg': '#FAFAFA', 'fz': 30},

2\*\*6: {'bg': '#FF5722', 'fg': '#FAFAFA', 'fz': 30},

# 128～2048 黄色系

2\*\*7: {'bg': '#FFEE58', 'fg': '#FAFAFA', 'fz': 30},

2\*\*8: {'bg': '#FFEB3B', 'fg': '#FAFAFA', 'fz': 30},

2\*\*9: {'bg': '#FDD835', 'fg': '#FAFAFA', 'fz': 30},

# 1024~2048 橙色系

2\*\*10: {'bg': '#FF9800', 'fg': '#FAFAFA', 'fz': 30},

2\*\*11: {'bg': '#FB8C00', 'fg': '#FAFAFA', 'fz': 28},

# 4096 + 红色系

2\*\*12: {'bg': '#fb3030', 'fg': '#FAFAFA', 'fz': 28},

2\*\*13: {'bg': '#e92e2e', 'fg': '#FAFAFA', 'fz': 28},

2\*\*14: {'bg': '#da1e1e', 'fg': '#FAFAFA', 'fz': 24},

# 2\*\*15 + 黑色 超过2\*\*15颜色不再改变

2\*\*15: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 22},

2\*\*16: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*17: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*18: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 20},

2\*\*19: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 18},

2\*\*20: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 17},

2\*\*21: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 16},

2\*\*22: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 15},

2\*\*23: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 14},

2\*\*24: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 13},

2\*\*25: {'bg': '#3a3a3a', 'fg': '#E0E0E0', 'fz': 12},

}

# 窗口初始化

def init\_root(self):

column = self.column

space\_size = self.space\_size

cell\_size = self.cell\_size

# 创建根窗口

root = tkinter.Tk()

root.title('2048')

# 根窗口尺寸设置

window\_w = column \* (space\_size + cell\_size) + space\_size

window\_h = window\_w + cell\_size + 2 \* space\_size

root.geometry(f'{window\_w}x{window\_h}')

# 顶栏

header\_h = cell\_size + space\_size \* 2

header = tkinter.Frame(root, height=header\_h, width=window\_w)

self.init\_header(header)

# 棋盘

table = tkinter.Frame(root, height=window\_w, width=window\_w)

self.init\_table(table)

return root

# 顶栏初始化

def init\_header(self, master):

master['bg'] = self.style['page']['bg']

# 分数

emt\_score = tkinter.Label(master, bd=0)

emt\_score['fg'] = '#707070'

emt\_score['bg'] = self.style['page']['bg']

emt\_score['font'] = ("黑体", 30, "bold")

img = Image.new('RGB', (self.cell\_size, self.cell\_size),

self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

emt\_score.configure(image=img)

emt\_score['image'] = img

emt\_score['text'] = 'score:' + str(self.data.score)

emt\_score['compound'] = 'center'

self.emt\_score = emt\_score

emt\_score.place(x=15, y=15)

master.pack()

# 棋盘初始化

def init\_table(self, master):

column = self.column

cell\_size = self.cell\_size

space\_size = self.space\_size

master['bg'] = self.style['page']['bg']

# 创建棋盘子元素

emts = [[0 for x in range(column)] for y in range(column)]

for row in range(column):

for col in range(column):

emt = tkinter.Label(master, bd=0)

emt['width'] = self.cell\_size

emt['height'] = self.cell\_size

emt['text'] = ''

emt['compound'] = 'center'

y = space\_size + (cell\_size + space\_size) \* row

x = space\_size + (cell\_size + space\_size) \* col

emt.place(x=x, y=y)

emts[row][col] = emt

self.emts = emts

master.pack()

# 更新UI数据

def update\_ui(self):

def update\_score():

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style['page']['bg'])

img = ImageTk.PhotoImage(img)

self.emt\_score.configure(image=img)

self.emt\_score['image'] = img

self.emt\_score['text'] = 'score:' + str(self.data.score)

update\_score()

matrix = self.data.matrix

for row in range(self.column):

for col in range(self.column):

num = matrix[row][col]

emt = self.emts[row][col]

img = Image.new(

'RGB', (self.cell\_size, self.cell\_size), self.style[num]['bg'])

img = ImageTk.PhotoImage(img)

emt.configure(image=img)

emt['fg'] = self.style[num]['fg']

emt['bg'] = self.style[num]['bg']

emt['image'] = img

emt['font'] = ("黑体", self.style[num]['fz'], "bold")

emt['text'] = str(num) if num != 0 else ''

# 事件循环

def key\_event(self, event):

# print(f"键盘输入:{ event.keysym }, ASCII码:{ event.keycode }")

# 注意：在不同系统下的同一按键的keycode可能不同

if event.keysym in ['Up', 'w', 'Down', 's', 'Left', 'a', 'Right', 'd']:

if event.keysym in ['Up', 'w']: # 向上

self.data.matrix\_move('U')

elif event.keysym in ['Down', 's']: # 向下

self.data.matrix\_move('D')

elif event.keysym in ['Left', 'a']: # 向左

self.data.matrix\_move('L')

elif event.keysym in ['Right', 'd']: # 向右

self.data.matrix\_move('R')

self.data.generate\_number()

# 按z 返回上一步

if event.keysym == 'z':

if self.data.history != []:

self.data.prev\_step()

self.update\_ui()

# 重置游戏

def reset\_game():

self.t = 0

self.data.init()

self.update\_ui()

if self.data.gameover() is True:

# 直接结束显式存在问题，显示的是上次的数据

if self.t == 0:

self.t = 1

else:

res = tkinter.messagebox.askyesno(

title="2048", message="Game Over!\n是否重新开始!")

if res is True:

reset\_game()

else:

self.root.quit()

# 重置游戏

def reset\_game(self):

self.t = 0

self.data.init()

self.update\_ui()

# 主程序

def main(self):

self.update\_ui()

# 绑定键盘事件

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

# 主循环

self.root.mainloop()

g = Window2048(4)