# 第五小组作品报告提纲

## 作品名称: 简单2048小游戏

1. 成员介绍

**组长 谭小明：框架构思+部分代码编写+汇报**

**组员 徐薇：主要代码编写及测试**

**组员 朱敏：ppt制作**

**组员 刘思雨：ppt制作+汇报**

**组员 刘雪岩：部分代码编写及测试  
组员 张航：主要代码编写  
组员 李波：相关学习资料的收集与提供**

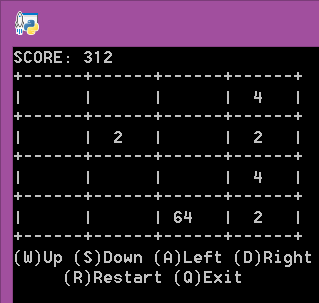
1. 作品简介

应用场景：日常的玩耍

1. 总体设计
   1. 基本思路：
      1. 构建棋盘；
      2. 通过键盘上的指定键控制数字的移动（数字2，4随机生成）；
      3. 数字移动时若移动方向相邻两个数字有相等的，则把该两个数字相加合并；
      4. 重复上述步骤，若得分达到2048，游戏结束，否则，游戏继续。
   2. 主要技术难点和解决方案
      1. 首要难点：数字的移动，将移动进行分解，分为上下左右移动；
      2. 用什么记录位置，怎样记录每个数字的值，边界，相同值相加？

选择用一个二维数组记录位置相关信息。

1. 特色和创新点
   1. 特色：基于cmd实现，操作简易；
   2. 创新点：设置重置按钮。
2. 运行截图



1. 项目组成员的工作心得

第五组的成员整体上来说，计算机专业的同学占了大多数，只有两个同学是经济学院的，项目应该是比较轻松能完成的，但是却一直拖到汇报前的一两周才开始，而且效率也不是很高，我们组里一个很严重的问题就是成员的积极性极低，群里几乎没有人回应我的安排，但是让我非常高兴的是我们在最后两周一起齐心的把作业完成了，不管怎么样，这样的经历是十分难忘的。

1. 存在的问题、建议及其他需要说明的情况

小组成员积极性不高，沟通上存在问题。建议以后每个小组里可以把老师或者助教拉进群里一起讨论，这样遇到问题也更好解决。

1. 附件：代码
2. *# import，导入所需要的包，库***import** curses  
   **from** random **import** randrange, choice *# generate and place new tile***from** collections **import** defaultdict  
     
   *# 获得有效键值列表*letter\_codes **=** [ord(ch) **for** ch **in 'WASDRQwasdrq'**]  
     
   *# 定义用户动作*actions **=** [**'Up'**, **'Left'**, **'Down'**, **'Right'**, **'Restart'**, **'Exit'**]  
   *# 将输入与行为进行关联*actions\_dict **=** dict(zip(letter\_codes, actions **\*** 2))  
     
     
   *# 用户输入处理，这里直到获得用户有效输入才返回对应行为。***def get\_user\_action**(*keyboard*)**:** char **= "N"  
    while** char **not in** actions\_dict**:** char **=** *keyboard*.**getch**()  
    **return** actions\_dict[char]  
     
   *# 矩阵转置***def transpose**(*field*)**:  
    return** [list(row) **for** row **in** zip(**\****field*)]  
   *# 矩阵逆转***def invert**(*field*)**:  
    return** [row[**::-**1] **for** row **in** *field*]  
     
     
   *# 创建棋盘，初始化棋盘参数，  
   # 这里指定了棋盘的高和宽以及游戏胜利条件  
   # 在这里我们制定的是最经典的4x4的模式。***class GameField**(object)**:  
    def \_\_init\_\_**(self, *height***=**4, *width***=**4, *win***=**2048)**:** self.height **=** *height* self.width **=** *width* self.win\_value **=** *win # 过关分数* self.score **=** 0 *# 当前分数* self.highscore **=** 0 *# 最高分* self.**reset**() *# 重置棋盘  
     
    # 重置棋盘* **def reset**(self)**:***# highscore为程序初始化过程中定义的一个变量。记录你赢得游戏的最高分数记录。* **if** self.score **>** self.highscore**:** self.highscore **=** self.score  
    self.score **=** 0  
     
    *# 生成一个二维数组。  
    # 其中，第一维长度是self.width。第二维长度为self.height。  
    # 相当于把棋盘上每一个下棋的地方(横纵线交叉产生的方格)赋值为0。* self.field **=** [[0 **for** i **in** range(self.width)] **for** j **in** range(self.height)]  
    self.**spawn**()  
    self.**spawn**()  
     
     
    *#按键后数字合并操作，棋盘走一步。  
    #通过对矩阵进行转置与逆转，可以直接从左移得到其余三个方向的移动操作。* **def move**(self, *direction*)**:  
    def move\_row\_left**(*row*)**:  
    def tighten**(*row*)**:** *# squeese non-zero elements together* new\_row **=** [i **for** i **in** *row* **if** i **!=** 0]  
    new\_row **+=** [0 **for** i **in** range(len(*row*) **-** len(new\_row))]  
    **return** new\_row  
     
    *# 定义合并函数* **def merge**(*row*)**:** pair **= False** new\_row **=** []  
    **for** i **in** range(len(*row*))**:  
    if** pair**:** new\_row.**append**(2 **\*** *row*[i])  
    self.score **+=** 2 **\*** *row*[i]  
    pair **= False  
    else:  
    if** i **+** 1 **<** len(*row*) **and** *row*[i] **==** *row*[i **+** 1]**:** pair **= True** new\_row.**append**(0)  
    **else:** new\_row.**append**(*row*[i])  
    **assert** len(new\_row) **==** len(*row*)  
    **return** new\_row  
    **return tighten**(**merge**(**tighten**(*row*)))  
     
    moves **=** {}  
    moves[**'Left'**] **= lambda** *field***:** \  
    [**move\_row\_left**(row) **for** row **in** field]  
    moves[**'Right'**] **= lambda** *field***:** \  
    **invert**(moves[**'Left'**](**invert**(field)))  
    moves[**'Up'**] **= lambda** *field***:** \  
    **transpose**(moves[**'Left'**](**transpose**(field)))  
    moves[**'Down'**] **= lambda** *field***:** \  
    **transpose**(moves[**'Right'**](**transpose**(field)))  
     
    **if** *direction* **in** moves**:  
    if** self.**move\_is\_possible**(*direction*)**:** self.field **=** moves[*direction*](self.field)  
    self.**spawn**()  
    **return True  
    else:  
    return False** *# 判断输赢* **def is\_win**(self)**:  
    return** any(any(i **>=** self.win\_value **for** i **in** row) **for** row **in** self.field)  
     
    *# 游戏界面的绘制* **def is\_gameover**(self)**:  
    return not** any(self.**move\_is\_possible**(move) **for** move **in** actions)  
     
    **def draw**(self, *screen*)**:** help\_string1 **= '(W)Up (S)Down (A)Left (D)Right'** help\_string2 **= ' (R)Restart (Q)Exit'** gameover\_string **= ' GAME OVER'** win\_string **= ' YOU WIN!'  
    def cast**(*string*)**:** screen.**addstr**(*string* **+ '**\n**'**)  
     
    *# 绘制水平分割线* **def draw\_hor\_separator**()**:** line **= '+' +** (**'+------' \*** self.width **+ '+'**)[1**:**]  
    separator **= defaultdict**(**lambda:** line)  
    **if not** hasattr(draw\_hor\_separator, **"counter"**)**:** draw\_hor\_separator.counter **=** 0  
    **cast**(separator[draw\_hor\_separator.counter])  
    draw\_hor\_separator.counter **+=** 1  
     
    **def draw\_row**(*row*)**:  
    cast**(**''**.**join**(**'|{: ^5} '**.**format**(num) **if** num **>** 0 **else '| ' for** num **in** *row*) **+ '|'**)  
     
    *screen*.**clear**()  
    **cast**(**'SCORE: ' +** str(self.score))  
    **if** 0 **!=** self.highscore**:  
    cast**(**'HIGHSCORE: ' +** str(self.highscore))  
    **for** row **in** self.field**:  
    draw\_hor\_separator**()  
    **draw\_row**(row)  
    **draw\_hor\_separator**()  
    **if** self.**is\_win**()**:  
    cast**(win\_string)  
    **else:  
    if** self.**is\_gameover**()**:  
    cast**(gameover\_string)  
    **else:  
    cast**(help\_string1)  
    **cast**(help\_string2)  
     
    *# 棋盘数字生成，随机生成一个2或者4* **def spawn**(self)**:** new\_element **=** 4 **if randrange**(100) **>** 89 **else** 2  
    (i,j) **= choice**([(i,j) **for** i **in** range(self.width) **for** j **in** range(self.height) **if** self.field[i][j] **==** 0])  
    self.field[i][j] **=** new\_element  
     
    *# 判断能否移动* **def move\_is\_possible**(self, *direction*)**:  
    def row\_is\_left\_movable**(*row*)**:   
    def change**(*i*)**:** *# true if there'll be change in i-th tile* **if** row[*i*] **==** 0 **and** row[*i* **+** 1] **!=** 0**:** *# Move* **return True  
    if** row[*i*] **!=** 0 **and** row[*i* **+** 1] **==** row[*i*]**:** *# Merge* **return True  
    return False  
    return** any(**change**(i) **for** i **in** range(len(*row*) **-** 1))  
     
    check **=** {}  
    check[**'Left'**] **= lambda** *field***:** \  
    any(**row\_is\_left\_movable**(row) **for** row **in** field)  
     
    check[**'Right'**] **= lambda** *field***:** \  
    check[**'Left'**](**invert**(field))  
     
    check[**'Up'**] **= lambda** *field***:** \  
    check[**'Left'**](**transpose**(field))  
     
    check[**'Down'**] **= lambda** *field***:** \  
    check[**'Right'**](**transpose**(field))  
     
    **if** *direction* **in** check**:  
    return** check[*direction*](self.field)  
    **else:  
    return False***# 下面是主逻辑代码，游戏会不断循环，直到达到Exit终结状态结束程序***def main**(*stdscr*)**:  
    def init**()**:** *#重置游戏棋盘* game\_field.**reset**()  
    **return 'Game'  
     
    def not\_game**(*state*)**:** *#画出 GameOver 或者 Win 的界面* game\_field.**draw**(stdscr)  
    *#读取用户输入得到action，判断是重启游戏还是结束游戏* action **= get\_user\_action**(stdscr)  
    responses **= defaultdict**(**lambda:** *state*) *#默认是当前状态，没有行为就会一直在当前界面循环* responses[**'Restart'**], responses[**'Exit'**] **= 'Init'**, **'Exit'** *#对应不同的行为转换到不同的状态* **return** responses[action]  
     
    **def game**()**:** *#画出当前棋盘状态* game\_field.**draw**(stdscr)  
    *#读取用户输入得到action* action **= get\_user\_action**(stdscr)  
     
    **if** action **== 'Restart':  
    return 'Init'  
    if** action **== 'Exit':  
    return 'Exit'  
    if** game\_field.**move**(action)**:** *# move successful* **if** game\_field.**is\_win**()**:  
    return 'Win'  
    if** game\_field.**is\_gameover**()**:  
    return 'Gameover'  
    return 'Game'** *# state存储当前状态，state\_actions这个词典变量作为状态转换的规则，它的key是状态，value是返回下一个状态的函数。* state\_actions **=** {  
    **'Init':** init,  
    **'Win': lambda: not\_game**(**'Win'**),  
    **'Gameover': lambda: not\_game**(**'Gameover'**),  
    **'Game':** game  
    }  
     
    *# 使用默认颜色作为背景* curses.**use\_default\_colors**()  
     
    *# 设置终结状态最大数值为 2048q* game\_field **= GameField**(win**=**2048)  
     
    state **= 'Init'** *#状态机开始循环* **while** state **!= 'Exit':** state **=** state\_actions[state]()  
      
    *# 此处的意思是: state=init()或者  
    # state=not\_game(‘Win')或者  
    # 是另外的not\_game(‘Gameover')/game()*curses.**wrapper**(main)