开发文档

一、 开发经历、思路、遇到的难题以及解决方法

我的 pj 是基于 lab7 实现的。

我遇到的第一个问题是判断碰撞的问题：若判断的是当前的方块经过操作后是否会碰撞，则会导致当前方块碰上了之后要再输入一次“s”，才能出现新的方块的状况。我一开始在碰撞函数外进行了一系列复杂的操作，试图通过判断多种情况下的碰撞来解决这一问题，后来发现，只要将当前的方块经过操作后再向下平移一个单位，判断这个经过向下平移后的方块有没有撞上，就可以实现一撞上就出现新的方块。这样，就在 collide 碰撞函数内部的行数判断中直接+1 就可以简单地解决了。

我遇到的第二个问题是：因为我输出 print 的算法是分别判断并输出已有的堆积起来的方块和当前的方块，这样就会导致当游戏结束时，若最上方一行原本就有方块（堆积方块）了，我还会输出新的当前的方块，从而导致方块重叠而产生多余。解决方法：写一个函数，当游戏结束时，判断第一行新方块出现的位置是否为均空，只要有一个位置有堆积的，就不再输出新的方块了。

我遇到的第三个问题是：因为我写了 7 个不同的类去存储 7 个基本方块，而在声明当前方块 currentblock 时又只能声明一种类，无法在声明的时候做到随机声明当前方块的类，也就无法做到随机生成基本方块。解决方法：将 7 个基本方块的类都继承自同一个父类，在声明时先声明当前方块为父类类型，在创建实例时再将对象当前方块随机实例化为一个子类，也就是 7 个基本方块中的一种，从而实现随机生成下一个基本方块。

我遇到的第四个问题，也是我**至今仍还没解决的问题**：如前所述，我写了 7 个不同的类去存储 7 个基本方块，这 7 个类又继承自同一父类。在 7 个子类中，有两个与旋转有关的方法，调用的变量名均是相同的（如：rows，cells，columns 等），这样就导致了他们写出来的形式完全相同，从而导致 ide 报重，也确实是有很多重复。我原想在父类中写这个方法，从而避免每个子类都写一遍的重复，但是问题来了，虽然方法调用的变量名相同，写出来的形式相同，但是子类的变量也就是属性实际上是不同的（如：不同的基本方块的 cells[][]就不相同），而父类又无法得到子类的属性，如果通过在父类中构造子类的对象来获取子类的属性也只能获得某一确定子类的属性，无法随子类变化而改变。这就导致了我的代码在这方面有很多重复又无法消除。

我遇到的第五个问题：判断随机出来的俄罗斯方块是否合法的问题。我创造随机俄罗斯方块的方法是先随机出行和列，在用两个 for 循环随机每一行列是否有格子，接着再判断是否合法，若不合法再重头来过。对于判断是否合法，我原想从第一行第一列扫到最后一行最后一列，判断是否存在某一方块存在而周围均无方块，这样的问题是对于随机出的俄罗斯方块若有两个完全独立的小方块组，也会判定其合法。接着我又优化这个方法，又想了很多方法，均有缺陷。最后的解决方法：运用递归，当搜索到一个方块存在时，消除它以及它四周的方块（若存在），再消除它存在的四周的方块的存在的四周方块，以此类推。若这样的消除在从第一行第一列扫到最后一行最后一列进行且仅进行了一次时，说明该随机出的俄罗斯方块只有一个方块组，该俄罗斯方块合法，否则不合法。

我遇到的第六个问题：想不出怎么实现跨左右模式的俄罗斯方块。我原本的想法是新构造一个与 UI 数组等大的数组，若当前方块超出边界，就跨到该等大的数组的对应坐标上，当当前方块全部跨到等大的数组上时，就将等大的数组上方块的位置赋给原 UI 数组，接着清空该等大的数组，以备下一次超出边界时调用。他的问题是，往左往右需要构造两个等大数组，就算只要构造一个等大的数组，每次赋值来赋值去也会很麻烦，很容易赋值

错。解决方法：在助教的提醒下，重写原来的各种方法并将其中一些用到 column 的地方改成 column 模 UI 数组的宽度，当处于跨左右模式的俄罗斯方块下时，就调用这些方法，轻松愉快就能实现跨左右。

我遇到的第七个问题是由第六个问题延伸出来的：JAV 中的%是取余而不是取模，取余就会导致产生负数从而导致数组溢出。解决方法：百度一下，查到 Math.floorMod 是取模，用 Math.floorMod 代替%即可实现取模。

我遇到的第八个问题：对文件存档和读取一无所知。解决方法：在超星上学习了文件 IO 流有关内容，套用了超星 ppt 上一种读取和输出的方法，实现了文件存档和读取。

我遇到的第九个问题：难以实现排行榜，主要原因是在于读取，存入通过问题八的学习我已经基本没有问题了，但是这里的读取不同于一般的读取一个文件，而是要读取一个文件夹下所有的文件并且根据他们里面存储的数据（即分数）的大小对他们进行排序并输出，通过百度查到了一些方法，但都使用了 list，我对 list 不甚了解，最终的解决方法是，使用数组代替了一部分 list 的功能，通过对数组进行排序实现输出。

其他一些问题：各种报错，尤其是数组溢出。解决方法：耐心地 debug 调试，看看是不是自己哪里又蠢了漏写了一个减 1 之类的。

一些小 bug：每次 game 结束或退出时要将 score 调为 0，否则不退出程序，调用同一实例 game，得到的 score 会是上次残余的 score

二、 主要函数及功能、主要参数及作用

主要解释 Game 中的参数和函数：

参数（解释如注释所示）：

```
// the cells are used to hold the blocks
// the cells are the big UI
public boolean[][] cells;
// rows and columns are the height and width od the cells
public int rows;
public int columns;

// "next" is used to store the int representation of the next block.
public short next;

// "score" is used to record the score you get.
public int score = 0;

// "randomGame" is used to judge whether the game mode the player choose is random mode.
// it will be changed to "true" when initializing random block if the game mode the player choose is random mode.
public boolean randomGame = false;

// "currentBlock" and "nextBlock" are respectively used to store the current block cells and the nextBlock cells(their
property).
public Block currentBlock;
private Block nextBlock;

```

```
// seven random block
private RandomBlock randomBlock1;
private RandomBlock randomBlock2;
private RandomBlock randomBlock3;
private RandomBlock randomBlock4;
private RandomBlock randomBlock5;
private RandomBlock randomBlock6;
private RandomBlock randomBlock7;

// the global position of the cells[0][0] of currentBlock
public int posRow;
public int posColumn;
```

函数（解释如注释所示）：

1.initial

```
    /**
 * initialize the cells with height and width, assign all cells except the bottom cells with "false" and assign the
bottom cells with "true".

 * @param height: the rows of UI cells
 * @param width: the columns of UI cells
 */
public void initial(int height, int width)
```

2.nextBlock

```
    /**
 * judge the game mode with @param randomGame.
 * assign currentBlock with nextBlock or based on @param next.
 * if the game mode is randomGame, employ @method renewBlock to reset the property of the random block.（因为random
block在第一次实例化后就要定下来，不能每次都new一个新的对象，不然会导致每次随机出来不同的方块，所以只能写一个方法手动
重置random block的属性，主要是重置它的旋转状态等。）
 * assign posRow and posColumn with 0 and columns / 2 - blockColumns / 2.
*/
public void nextBlock()
```

3.nextNextBlock

```
    /**
 * randomize.
 * get next random block after current block.
 * record which random block you get with @param next.
 */
public void nextNextBlock()
```

4.initialRandomBlock

```
    /**
 * initialize the seven random block before the random game start.
 * employ @method isSame to judge whether the random block you get is the same as the block you get before.
 * if isSame ,get a new random block until it is different from all the blocks you get before.
 * print the random block you get.
 * employ @method printBlock to print.
 * change @param randomGame to "true" ,meaning the game mode is random game.
 */
public void initialRandomBlock()
```

5.isOver

```
    /**
 * since posRow, posColumn, currentBlock will never collide with game cells except when nextBlock appeared.
 * we can invoke "collide()" below to check whether the game is over.
 * @return: whether the game is over
 */
public boolean isOver()
```

6.print

```
    /**
 * print the game UI based on the game mode(normal game mode or random game mode).
 */
public void print()
```

7.executeCommand

```
    /**
 * execute next command, invoke collide() to check collision before rotating or moving.
 * when the command is "s" and the block has reached to the bottom,
 * loadBlock(), eliminate(), nextBlock() will be invoked sequentially.
 * @param command: the next command {"w","s","a","d"}
 */
public void executeCommand(String command) {
```

8.collide

```
    /**
 * judge whether blockCells at position(nextRow, nextColumn) collide with game cells.
 * @param nextPosRow: the next posRow after rotating or moving.
 * @param nextPosColumn: the next posColumn after rotating or moving.
 * @param nextBlockCells: the next blockCells after rotating or moving.
 * @return: true if collided, false otherwise.
 */
private boolean collide(int nextPosRow, int nextPosColumn, boolean[][] nextBlockCells) {
```

9.loadBlock

```
    /**
 * assign "true" to the game cells occupied by currentBlock.
 */
private void loadBlock() {
```

10.eliminate

```
    /**
 * eliminate all the "full" rows. "full" means all true.
 */
private void eliminate() {
```

11.eliminateOneRow

```
    /**
 *  Force to eliminate one row, and all the cells above it will drop by one row.
 *  @param row: the index of the eliminated row.
 */
private void eliminateOneRow(int row) {
```

12.empty

```
    /**
 * 判断一行是否为空，用于决定 gameover 时是否要在第一行输出 new 的 block，若 gameover 时恰好是第一行已经有了，则不必再输出
 * @param row:the row to judge.
 * @return whether the position of the new block is empty.
 */
private boolean empty(int row)
```

13.downOneBlock

```
    /**
 * when current blocks will collide the UI cells, load it, try to eliminate full rows, and get next block on the top
of the UI.
 */
private void downOneBlock()                    //下降一格并碰底
```

14.downToBottomRow

```
    /**
 * @param currentBlockCells:current block.
 * @return the first row of current block when it collide the UI cells directly under it.
 */
private int downToBottomRow(boolean[][] currentBlockCells)
```

15.downToBottom

```
    /**
 * make current block fall directly to the UI cells under it.
 */
private void downToBottom()
```

16.printInitial

```
    /**
 * print the initial game UI.
 */
public void printInitial()
```

17.printBlock

```
    /**
 * print the block you want to print with its parameters.
 * @param rows:the rows of the the block you want to print.
 * @param columns:the columns of the block you want to print.
 * @param block:the cells of the block you want to print.
 */
private void printBlock(int rows,int columns,boolean[][] block)
```

18.isSame

```
    /**
 * jugde whether the two blocks is the same.
 * @param row1:the rows of the first block.
 * @param column1:the columns of the first block.
 * @param cells1:the cells of the first block.
 * @param row2:the rows of the second block.
 * @param column2:the columns of the second block.
 * @param cells2:the cells of the second block.
 * @return whether the two blocks is the same.
 */
private boolean isSame(int row1,int column1,boolean[][] cells1,int row2,int column2,boolean[][] cells2){
```

19.writeRecord

```
    /**
 * delete previous file with the same name as current file and create current file.
 * record the property of current game when the player quit.
 * @param name:player's name
 * @param mode:game mode
 */
public void writeRecord(String name,String mode)
```

20.readRecord

```
    /**
* read the property of the previous game under the player's name of the game mode.
* @param name:player's name
* @param mode:game mode
*/
public void readRecord(String name,String mode)
```

21.hasRecord

```
    /**
* judge whether there has been any record under the name and the game mode.
* @param name:player's name
* @param mode:game mode
* @return
*/
public boolean hasRecord(String name,String mode)
```

22.writeScore

```
    /**
* record the player's score under a specific game mode when game over.
* used to compose the ranking list.
* @param name:player's name
* @param mode:game mode
*/
public void writeScore(String name,String mode)
```

23.readScore

```
    /**
* print the ranking list.
* @param mode:the game mode of the ranking list printed
*/
public void readScore(String mode)
```