# 西安交通大学

|  |  |
| --- | --- |
| **课程：** | JAVA语言程序设计 |
| **班级：** | 物试2202 |
| **学号：** | 2225015585 |
| **姓名：** | 黄得清 |
| **时间：** | 2024年6月 |

# JAVA语言程序设计

# 大作业

# 《连连看》实验报告

## 项目名称

连连看小游戏（支持计时、回放、存储及网络传输等功能）

（GitHub项目地址：https://github.com/Dytchem/LianLianKan）

## 项目设计

1. 支持基本的练练看游戏玩法，包括：生成游戏局面（合法自定义大小）、实现连线消除（同色且拐弯次数小于等于2）和判断最终胜利（方块全部被消除）。
2. 游戏有计时系统，可自定义开启和关闭，而且适应在特殊情况下（如胜利）自动暂停。
3. 游戏有作弊模式，可自定义开启和关闭，而且支持如同正常消除的回放和存储等功能。
4. 游戏支持通过进度条拖拽使游戏回到当前局面之前的任意局面，这等价于回放功能。
5. 游戏支持保存当前游戏局面至文件，也支持导入文件完全复刻保存时的状态。
6. 在网络层面加强5所述功能，使客户端能进行从服务器端读取存档、保存存档的等操作。

## 项目分工

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **班级** | **学号** | **分工** | **备注** |
| 黄得清 | 物试2202 | 2225015585 | 项目设计1~4 | 游玩功能 |
| 郑佳明 | 网安2201 | 2224213619 | 项目设计5~6 | 存储功能 |

**我的分工主要是项目设计中的1~4，即实现练练看在游玩方面的功能。**

**（以下内容全是关于我的部分）**

## 项目结构

1. **主菜单**由类MainMenu维护，它继承自Frame，实际上就是一个可展现窗口。在该类的构造方法中进行窗口的调整和各种组件的添加，包括展现游戏标题的Label、执行相关操作的三个Button、顶栏关于菜单MenuBar。由于这里的组件触发的事件较为简单，所以添加的是匿名监视器。

xxx.addActionListener(**new** ActionListener(){…})



1. **创建游戏**由类CreateGame维护，它也是个Frame，可展现。界面元素有Label标题、三个SingleInput（继承自Panel，内含一个标签和一个输入框，用于数据的输入与提取）、开启计时功能和作弊功能的复选框Checkbox、开始游戏的按钮Button。对按钮添加匿名监视器，先判断已填入的数据是否合法，然后根据数据创建对应初始参数的游戏界面。

图形用户界面

描述已自动生成

1. **游戏界面**由类Game维护，也是Frame，主要展现的就是游戏前端。该类较为复杂，将分解为后续的几点。

图片包含 表面图

描述已自动生成

1. 类Game的成员GameBoard g主要掌管**方块盘面**，即方块所在的区域。它继承自JPanel，额外（主要）增加成员GameData g和Cell[][] bs。前者把游戏数据从具体界面中剥离出来，专注于利用数据结构与算法来实现游戏中的生成、寻路、回溯等功能（将在算法分析模块中讨论）。后者为具体的方块数组，可点击，它继承自Button，并增加了监视点击实现消除（设置不可见）的功能。在类Game中，将进行由Cell触发然后由g指导的bs的可见性变化，这就是基本的连连看功能。
2. 类Game的成员Draw c主要掌管**方块连线**，即在进行消除操作或回放时连接对应方块。c中装的是预先准备的Label[] ls，用于展示指定粗细的线段（实际上就是空白标签加上纯黑背景色），和记录线段的ArrayList<Segment> ss。单次连线操作其实就是g指导的线段的添加和Label的可见性改变，开个Thread让它固定时间消失同时不影响消除操作。
3. 类Game的成员Timer t主要掌管**游戏计时**。它继承自Label，同时实现了Runnable接口。在Label方面它实现了将当前记录的秒数转换为自身格式化显示的时间，在Thread方面它实现了计时累加秒数，同时能自由开始计时和暂停计时。当然，在类Game中要提供控制t的相关代码。
4. 类Game的成员JSlider s主要掌管**游戏回放**。我这里没有单独将它的功能包装成一个类，而是直接功能代码放在Game里。forward()和backward()函数分别实现了进度条前移和后移时所有组件要进行的操作，它们交由s的监视器来控制。
5. 类Game的成员Checkbox fake1主要掌管**游戏作弊**。实际上就是为了提供一个布尔变量，为真时点击任意方块即可消除，需配合以上已实现的功能使之不矛盾。

## 算法分析

算法主要集中在GameData中。

1. **盘面生成算法**

用int数组存盘面，注意要用比盘面指定大小边长大4的规格，因为要存外墙和外路，方便处理。用1表示不能连线的墙，0表示可以连线的通路，2及以上表示方块颜色。

在GameData构造的时候生成同等大小的一维数组，先以取模的手段填充该数组，达到分配颜色的目的。然后用Random随机打乱数组，并按照该数组按次序依次复制填充到游戏盘面int数组中。最后复制一份盘面方便各种回放操作。

可以调试看到生成的盘面例如下面这样：

图片包含 日历

描述已自动生成

1. **方块寻路算法**

这是精髓之所在，我们实现了**改进版的广度优先搜索（BFS）**。

连连看消除的基本要求是同色、连线不能越过障碍、拐弯次数必须小于或等于2次。

同不同色只需判断一下int数组对应位置存的数据是否一样就行了。

连线不能越过障碍，只需要在BFS时判断一下下一步的位置是否为0即可。

拐弯次数必须小于或等于2次，这是问题最复杂的地方。我看网上很多人对这个限制的实现都较为复杂，他们大多都是直接分类讨论加判断，考察各种可能出现的连线情况，代码量较大，不够优雅。较为优雅的方法是对基础版的BFS做改进，能够解决问题且基本没有增加代码量。以下为我的思路：

我们需要增加BFS遍历到某一点时记录的状态。

由于这个问题增加了对连线方向的考虑，我们需要在状态中增加对当前方向和拐弯次数的记录。

我们最后要能还原出连线的样子，所以需要在状态中再存一个它的前驱，即所有结点整体维护了一个生成树树结构，最后能通过从叶子结点反向遍历得到从叶到根的链表。定义节点如下：

文本

描述已自动生成

我们还需要准备一个boolean类型vis数组（四维）记录已遍历过的位置、方向及拐弯次数和一个队列存储下面要依次遍历的节点。

然后就可以开始BFS了：把初始位置加入队列。每次取队首元素弹出，将后继的所有vis不为true的结点全部加入队尾并更新vis，直到到达目标位置。我们只需要额外在遍历的过程中处理好结点状态中的d,t,pre，其他的代码和基础版BFS完全相同。要输出这个连线的样子，只需返回在目标位置的结点，由外部函数自行遍历获得连线路径。

1. **方块回溯算法**

由于forward()和backward()也是要对GameData进行改变的，我们需要在GameData中额外存储修改情况的信息，包括方块编号和是否作弊。

在前面的方块寻路以及最终消除中，我们不仅要改变盘面int数组的值，还有把方块编号和是否作弊的信息存储到数组（可以视为一个栈）中。

调用backward()时，我们需要出栈并根据出栈的元素来恢复方块，同时要注意到作弊消除需要按目前情况恢复1块，而正常消除需要强制再出栈一个元素同时恢复同色的2块方块。

调用forward()时，由于我们是用数组模拟的栈，可以直接通过移动指针恢复已出栈的元素，根据这个元素便可重新消除对应方块。

另外，如果在回放时进行游戏操作，则会使原本能forward()恢复的步骤被覆盖，无法恢复，这只需要强制改一下栈的大小即可。

## 运行结果

见展示视频

## 反思总结

1. 纵观整个项目，代码过于繁杂。我们利用面向对象的编程范式来解决问题，确实是要比面向过程更为清晰明了。但是有一点，就是我发现自己懒得封装一个类，因为完全的封装代码要写得太多太多，对每一个成员都要诸如getter和setter的操作。这导致我写出来的代码中甚至出现了诸如**int** k = g.g.kills[g.g.havekilled];的代码，直接访问对象属性。这确实有违封装思想，需要后面进一步改进。
2. 在设计界面的过程中，我尝试过利用布局管理器来管理组件的相对位置，但发现很难符合自己的审美，所以后面索性都基本改成无布局，然后通过手动计算绝对位置的方式来布局，微调以达到目标。这是一种很麻烦但又特别精准的布局方式。对于布局管理器的使用我还需要进一步了解。
3. 数据结构与算法是解决这个连连看问题的核心手段，在本项目中我们充分利用了队列、栈、链表等数据结构，并且拓展改进了BFS，优雅地解决了连线问题。辩证意义上，编程语言只是形式，而算法才是内容。
4. 还有一些需要改进的地方，比如说：

Timer计时主要用到了sleep(1000)来打断线程，其计时的准确性是不佳的，可能会受到线程调度影响，秒数不精准——更好的实现应该是加一个获取系统时间的过程，动态调整sleep的秒数。

GameBoard和Draw都是用基本组件拼接来绘图，而不是用Canvas，这似乎能称为“奇淫技巧”了。因为我们尝试了用Canvas绘图，但发现repaint()方法总是不能按我们预期按时展示出图像。系统调度总是神出鬼没，我们上网搜了很多资料都不能实现出达到我们预期的效果。最后我们发现基本组件的可见性改变可以立即奏效，于是就这样操作了。我们需要进一步了解Canvas的机理。

## 源代码

见src目录