益智游戏——“三”

设计说明书

徐栩海

工四二 2014010902

13588550938

orson.xu@foxmail.com

目录

1. 引言

1.1 编写目的

这是一份对“益智游戏——三”小程序的一份较为详细的设计说明书。

1.2 文档概述

该设计书中，首先对程序中各个类和类的成员进行详细说明，之后对编程过程当中的个人所遇到的一些难点和实现的亮点进行说明。

1.3 游戏概述

进入程序，玩家有以下选择：

（1）直接开始游戏

（2）查看排行榜

（3）查看游戏规则

（4）查看作者个人信息

（5）退出程序

开始游戏后，玩家可以选择自己的身份——即是否为 VIP 玩家（贵宾玩家拥有特殊功能）。玩家需要输入密码（password: three/THREE）来成为 VIP 玩家。

身份确定之后，玩家需要确定自己的玩家名。玩家可以选择自己直接输入玩家名，也可以在原有的表格中挑选自己之前玩过的玩家名。

玩家名确认之后，进入游戏。玩家可以自定义游戏的界面边长。

具体的游戏规则可参见玩家手册。

碰到游戏结束，若为 VIP 玩家，则有机会使用一次特殊功能——消去所有的 1,2,3 数字，从而得以继续进行游戏，但是此功能在一次游戏中只能使用一次。若不使用功能，或者不是 VIP 玩家，则直接计算分数，并记录在 Excel 中。

记录完毕之后，玩家可以选择退回到主界面，也可以选择重新进行游戏。

2. 类的设计

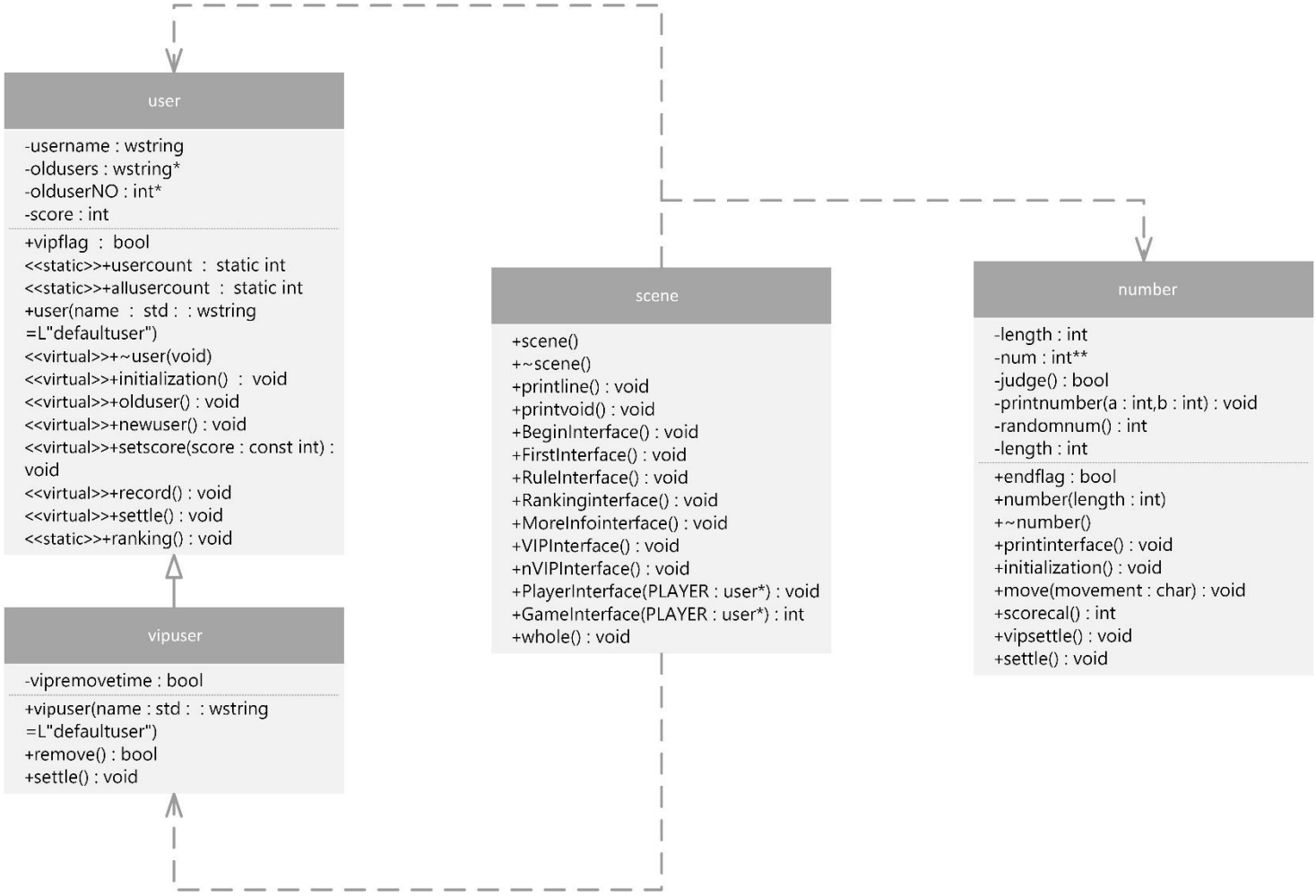
2.1 类图

以下是该程序整体类图情况，其中，

scene 类为界面调用模块，将界面的所有打印操作封装为一个类。

user 为用户模块，将用户的信息、对用户信息的操作和对 Excel 中的信息的处理封装为一个类。vipuser 类从 user 派生。

number 为游戏主体模块，将各方块中数字和对数字的操作封装为一个类。



2.2 类的各成员说明

下面对各个类中的各个成员进行说明

2.2.1 Scene 类：

本类将界面的所有打印操作封装为一个类 ,其中所有的成员均为公有非静态函数成员

scene()

scene 类的默认构造函数

~scene()

scene 类的默认析构函数

void printline()

打印横线条，为方便排版界面所设

void printvoid()

打印空行，和两边边界，为方便排版界面所设

void BeginInterface()

打印欢迎界面

void FirstInterface()

打印最开始的选择界面

void RuleInterface()

打印游戏规则

void Rankinginterface()

打印 excel 表格中现有的玩家前 5 名（若不足 5 名就打印所有）

void MoreInfointerface()

打印作者的个人信息

void VIPInterface()

对 VIP 用户，创建 vipuser 类

void nVIPInterface()

对非 VIP 用户，创建 user 类

void PlayerInterface(user* PLAYER)

不论是否是 VIP 用户，统一处理，实现多态。（其中用 vipflag 辨别是否是 VIP 用户 ,详见下文)其处理内容包括对用户的姓名的确定 ,对 GameInterface()函数的调用，对用户得分进行记录，并且可以实现重复游戏或者回到主界面等等一系列功能。

int GameInterface(user* PLAYER)

游戏运行，返回值为最终得分，退出前重置所有数字（保险起见）

void whole()

作为主调函数，对所有的函数进行调用。当选择了最主要的“开始游戏之后”，先通过用户输入确定是否是 VIP 玩家。经(n)VIPInterface()之后进入 PlayerInterface()。

2.2.2 User 类以及其派生的 Vipuser 类：

两个类将用户的信息、对用户信息的操作和对 Excel 中的信息的处理封装，前者为非 vip 用户对应的类，后者从前者派生，对应于 vip 用户。

(2.1)user 类

a. 私有非静态数据成员

std::wstring username

记录当前玩家的名字

std::wstring* oldusers

记录不同老用户的名字数组首地址

int* olduserNO

对应不同老用户对应编号数组首地址

int score

记录玩家得分

b. 公有非静态数据成员

bool vipflag

手动设置 ID，用于记录是否是 vip 用户，方便判断

c. 公有静态数据成员

static int usercount

记录 excel 中不同用户的总数目

static int allusercount

记录整个已有的 excel 表格的总条目数量（包括相同）

d. 公有非静态函数成员

user(std::wstring name=L"defaultuser")

构造函数，默认用户名，同时确定标记 bool vipflag，user 对象为非 VIP 用户，故其值为 false

virtual ~user(void)

析构函数，释放动态内存

virtual void initialization()

初始化信息 ,把表格中的信息——所有条目数量 ,不同的用户名记录到*olduser 为首地址的动态数组中，同时为其变编号，编号记录到 *olduserNO 为首地址的动态数组中。

virtual void olduser()

对于老玩家，确定用户名时调用这个函数，打印出所有的已有用户的用户名，并且让用户从中选择用户名，从而确定本次游戏中用户的名字

virtual void newuser()

对于新玩家，确定用户名时调用这个函数，由用户自己输入名字。

virtual void setscore(const int score)

记录本次玩家的得分

virtual void record()

将本次游戏的成绩记录到 Excel 中

virtual void settle()

对除了用户名之外的内容进行重置

e. 公有静态函数成员

//之所以设置为静态函数，因为在游戏一开始，还没有创建 user/vipuser 类时就需要实现查看排行榜的功能。所以这个函数有些独立于别的函数之外，用户总数量是自己另外设置的，覆盖了本类中原有的 allusercount

static void ranking()

打印出 Excel 中的前五名排行榜，若不足五人则全打印

(2.2)vipuser 类

Vipuser 类由 user 类继承，为实现游戏过程中贵宾用户的存在，贵宾拥有特殊功能，可以在游戏本当结束时有机会使用一次将所有的 1,2,3 数字消去。

下面只列出新增数据成员和函数成员。

a. 私有非静态数据成员

bool vipremovetime

用于记录 VIP 消除功能是否被使用

b. 公有非静态函数成员

vipuser(std::wstring name=L"defaultuser")

另外写构造函数，同时设置区别于 user 类的标记,将 vipflag 设置为 true，vipuser 对象为 VIP 用户

bool remove()

布尔值，记录 VIP 功能是否被使用

void settle()

覆盖原有的重置函数，且重置用于记录 VIP 功能是否被使用的布尔值

2.2.3 Number 类：

Number 类将各方块中数字和对数字的操作封装为一个类。

a. 私有非静态数据成员

int length

用于记录游戏界面的边长

int num**

用于动态申请二维数组，在其中存放各个数字

b. 私有非静态函数成员

bool judge()

判断游戏是否已经结束，若游戏结束，则返回 0，若可以继续，则返回 1，被 move()

函数调用

void printnumber(int a,int b)

打印各数字，并且实现不同数字不同颜色，被 printinterface()函数调用

int randomnum()

按照一定比例随机生成数字，被 initialization()和 move()函数调用

c. 公有非静态数据成员

bool endflag

用于对外界（本游戏中即为 scene 界面模块）显示游戏是否结束，提高函数效率，避免多次重复判断。详见 3.2.3。

d. 公有非静态函数成员

number(int length)

构造函数，动态分配二维数组内存

~number()

析构函数，释放二维数组

void printinterface()

打印整体游戏界面

void initialization()

初始化游戏

void move(char movement)

实现数字的移动，合并。这是本游戏算法的核心，实现了所有的判断条件，详细内容请见 3.2

int scorecal()

计算总分并返回分数值

void vipsettle()

vip 玩家可以随机消除 1,2,3 数字。仅仅只在玩家为 VIP 时可以调用。

void settle()

重置，对所有的数字进行清零，将边长参数清零

3. 技术难点及亮点

下面对自己编程过程当中所遇到的难点和一些运用小技巧实现的亮点进行阐述和讲解

3.1 实现对 Excel 文件的所有处理

3.1.1 Excel 的基本读写

Excel 不同于 Txt 文件，其内部结构非常复杂，普通用户对其代码实现几乎一无所知。这也对作者本身代码能力的提出了巨大的挑战。搜寻网络，最常见的一些方法为 OLE 方式、ODBC 方式、ADO 方式等。对于 C++初学者而言，对这几种方式的原理理解感到困难。

最后，作者选用借助 LibXL 库对 Excel 进行处理。LibXL 是一个专门为处理 excel 而编写的一个库，拥有非常强大的功能和相对简洁明了的接口。对 Excel 的读取、写入、设置格式等操作非常简单，从而在本程序中提供了用表格记录用户名，用户得分和用户游戏日期的记录功能。

3.1.2 Excel 数据实时更新

一般程序具有一次性，即游戏结束之后，将结果导入文件中之后程序即结束。但是作为一个相对成熟的游戏，需要实现在一次程序的运行过程中，可以多次从头开始游戏，并且保持游戏的排行榜显示最新数据情况的功能。

为此，对于 Excel 表格需要实现的功能在于——每一次游戏之后，不仅需要将结果保存入表格，还需要将数据实时更新到程序运行的内存当中。

为实现这一功能，同时保证程序代码的复用率，专门为 user 类和 vipuser 类编写数据的初始化函数(user::initialization()以及 vipuser::initialization())，其功能在于将函数调用时的 Excel 中的信息输入到内存当中，若原有记录，则覆盖其记录。如此一来，只需要在适当时候重复调用初始化函数，便可将表格中当时的数据读入，完全不用顾及前一次游戏的数据情况，非常方便。

3.2 实现游戏过程中的所有判断条件

这算是游戏模块中算法的核心。本程序的算法本身的原理十分简单，只需要判断数字之间的能否合并即可。但是考虑到游戏规则在各方面的约束，算法中“判断”成为了核心。

在游戏过程中，实现略有困难的判断内容包括（为方便起见，不再重述原来的需求分析报告中的大段内容，而将其简单转述如下）：

（1）本列/行数字是否可以移动

（2）本列/行数字是否可以在末尾添加数字以及本操作是否有效

（3）游戏是否完全结束

下面，分别对这三块内容的难点所在和解决方法进行阐述。

3.2.1 数字移动的判断

只有在以下两种情况下，数字才可以移动：

（1）该列前端有空位可以移动，同时该列不为全空。

（2）该列前端没有空位，但是出现数字合并情况。

为此，对于每一列/行，需要设置循环判断数字对是否可合并，若碰到了数字是零，即没有数字 ,则需要进一步判断 ,如果这一列之后的所有数字都是零 ,那么这列数字无法移动，如果这一列之后的数字存在不是零的，那么这列数字可以被移动。为实现这一个判断，需要嵌套一个循环，检查当前所在位置之后的所有数字是否全部为零。

3.2.2 数字添加和操作有效的判断

首先需要明确，每一次有效的操作都伴随着一个数字的加入。对一次操作，只有在所有的列/行中，存在至少一列数字发生了移动，才能够被称为是有效的，同时也才能够向末尾添加数字。

而对哪几列/行能够添加数字的列的判断 ,则相对简单——只要该列/行的最末端是数字零，便可作为待加入列/行。但是需要为此开辟一个数组，记录能够处理的列/行的序号。如果最后判断结果为能够添加数字，则将运用随机函数在这些列/行中挑选一个进行添加。

为此，设置一个标志进行辅助。在对数字对的每一次判断中，一旦发现满足 2.2.1（即数字可以移动）的判断，该标志加一。在对所有的列/行判断结束之后，若该标志不为零，则说明该操作是有效的。若该标志为零，就需要进入 2.2.3 的判断——游戏是否真的结束了？

3.2.3 游戏结束的判断

当发现 2.2.2 的判断结果为本操作无效，无法添加数字的时候，容易想当然的认为，这正是游戏结束的标志。但事实上 ,可能存在这样的情况——对于玩家输入的方向并没有数字可以合并，但是其他方向上存在数字可以合并。这意味着游戏并没有结束，依然有机会可以继续游戏。

因此，在判断 2.2.2 中标志为零时，需要遍历矩阵，对矩阵中所有的数字对进行检查，如果发现存在可合并的数字对，则最终结果仅为本操作无效，而游戏可以继续进行。如果最终无法找到可合并的数字对，则为游戏结束，计算得分。

同时 ,为了实现遍历判断 ,专门设置 number::judge()函数。并且为提高程序运行效率，一旦发现可合并数字对便 return。

以上所述皆为在 number 游戏模块中的判断实现，而要将这个信息传递到 scene 模块中的 GameInterface 函数中，却并不好实现。如果通过借用上述的 judge 函数，需要重复判断，调用次数多，代码效率低。故因此设置一个 endflag，对外界，即 scene 模块传递游戏结束的信号，提高代码的效率。

3.3 实现游戏足够的自由度

游戏的边界长度可以由用户的喜好决定，选择自己希望进行的界面大小。具体的实现方式为用将 number 类的数据成员设置为动态数组，所有的函数都建立在动态数组的基础之上。

至于对数字的直接输入，需要涉及到 3.6 节当中的内容，将在 3.6 节进行详细讲解。

3.4 实现几乎所有的容错

考虑用户的输入错误，对所有期望输入值之外的值纳入考虑范围。同时，对于错误的输入，一般希望用户重新输入一个值（事实上本程序中有的地方有一定的偷懒，若用户输入错误的值则默认为某一种值。但是这一默认值符合我们日常的理解。）。故需要用死循环来实现这一点，用 continue 重新输入，用 break 跳出循环。

在本程序中，用两种方法实现了容错。

（1）采用 switch 函数中的 default 进行容错。同时，由于循环的跳出和 switch 的每种情况结尾都采用了 break，故无法用一个 break 实现。为此，设置标记，对错误的答案进行标记，在 switch 结束之后通过判断标记的情况，决定是 break 还是 continue。

（2）采用 if else 进行容错，这一结构比 switch 好一些，可以直接使用 break 跳出循环。但是需要输入的内容略多于 switch。（个人认为这是 switch 还存在的一个原因，除了可读性略好之外，也只有这一个优点了。相对而言自己更加喜欢用 if）

另外，值得一提的是对方向键和 F1、F2 键等存在两个字符码的按键的容错实现。这一问题在编程过程当中一开始并没有考虑到，在测试过程当中发现问题——这些键由两个字符码组成，造成的后果为——按一个键，触发两次容错判断，程序会进行两次错误提示。容易造成用户的困惑。

其解决方法需要涉及到 3.6 节内容，故放在 3.6 讲解。

3.5 实现代码的高复用率

在程序最终版之前，将 VIP 与非 VIP 玩家完全分开处理，分别为其创建对应的 vipuser 类或者是 user 类，进而分别调用 GameInterface()函数。在游戏结束时，VIP 玩家在游戏结束的时候进行判断是否要运用特殊功能，而非 VIP 玩家则直接结束。程序可读性较高，比较直白。

但是经过分析，发现不论是 VIP 玩家还是非 VIP 玩家，一旦为其创建了相应类的实例，之后进行游戏的过程都是几乎一样的。唯一区别在于——VIP 玩家在游戏结束的时候，可以选择使用 VIP 功能。而这一区别并不是非常大。故可以采用多态来提高代码的复用率。

vipuser 继承于 user 类，故对于 scene::PlayerInterface 函数，采用 user* PLAYER 作为参数，可以实现多态。没有必要分开为 vip 和非 vip 用户重新书写 PlayerInterface 函数了

而要实现在游戏结束判断是否要触发 VIP 功能，尝试了很多直接判断的方法，但是发现

仅仅采用多态无法非常好的实现。故最后采用设置标记的方式，用 PLAYER->vipflag 判断玩家是否为 VIP。这也是为何 user 类和 vipuser 类中都需要出现 vipflag 的原因。

3.6 **实现良好的用户体验**

尽管本程序中并没有编写 UI，但是依然希望尽可能地提高用户体验并且改善用户交互。

最主要的一个问题在于——对于编程者而言，每次输入都以回车结尾是并不奇怪的，这也是一开始本程序所采用的方式。但是对于普通玩家而言，这一点就会显得非常不习惯。为此，本程序希望实现了对几乎所有的输入，都可以无需回车，直接响应的功能。（在某些特殊地方，作者认为采用回车最为结尾标志更好，更加符合用户的心理。）

一开始，采用 iostream 头文件中自带的 getchar()函数，但是该函数只有吸收回车的功能，并不能够非常好的实现对任何输入的直接读取，故放弃此方案。

之后，也是最后采用的方法，本程序中运用 conio.h 头文件中的_getch()函数，可以比较好的实现这一点。不需要回车即可将键盘的输入写入到对应内存当中。

但是存在两个问题，第一，_getch()函数只能够读取一个 char 字符，这与自定义游戏边长，进行动态数组的分配等过程中对 int 类型的需求是矛盾的，与字符串的输入也是不相符合的。第二，输入的内容不会将输入显示在屏幕上，用户并不能确切的看到自己输入了什么内容，没有很好的体验效果。

为解决第一个问题，对于数字输入，需要进行 char 到 int 的转换，之后再对 int 进行处理。而对于字符串的输入，只能够采用回车键入的方式。

为解决第二个问题，则在每一个_getch()之后都重新调用 cout，输出刚才的输入，使得用户可以看到自己的键入值。

下面讲解对方向键，F1 键等特殊键的处理。

先测试上下左右键，INS 键，DEL 键，HOME PAUSE 键等特殊的按键的代码，发现这些键之间的一个共同点——第一个字符码相同！

设置一个 char 的变量，用_getch()函数进行赋值，再将其输出，显示为空。而不设置 char 变量，直接 cout 由_getch()收到的值，显示值为 224。究其原因，应该是 char 类型的变量无法输出 127 以上的值对应的字符。

再用相同的方法测试 F1~F12 键，发现其第一个字符码依然相同。

设置一个 char 的变量，用_getch()函数进行赋值，再将其输出，显示为 0。而不设置 char 变量，直接 cout 由_getch()收到的值，显示值同样也为 0。这也当然可以理解，当其值小于 127 时，自然可以通过 char 输出。

发现了这些特殊的键之间的共同点。那么解决问题的思路在于——一旦碰到了接受的字符对应的值为 224 或 0，直接再次调用_getch()函数吞入其后一个字符。这样对程序而言，只需要进行一次容错判断即可。

但是经过尝试，无法间接对 char 字符的变量判断其是否等于 224，原因不详。

再尝试着直接对_getch()接受的字符进行判断，但是也存在问题——这需要用形如

if(_getch()==224||_getch()==0) _getch();

的代码来实现，但是考虑到“||”的特性，如果 _getch()!=224，则会再次调用_getch()

函数，即两次判断调用两次，并不符合实际情况。

最后采用这样的方式——将 char 型接受，强制转化为 int 型，方向键、INS 键等对应的值为-32，而 F1~F12 对应的值为 0。这样就可以通过 if 函数来实现了。

此外，在 DOS 界面当中，程序运行的速度非常快，这容易造成用户的疲劳。为此，利用系统的 Sleep()函数，对许多地方进行短时间（如 0.3 秒，0.5 秒）的延迟，从而造成逼真的交互效果。比如对话过程的短暂延迟，游戏开始的 loading 过程等等。