# 2048小游戏

# 概要设计说明书

姓名：孙阳

学号：SA18225329

目录

[一、用例方案精化 3](#_Toc530419628)

[1.1 提取边界类、实体类和控制类 3](#_Toc530419629)

[1.1.1 边界类 3](#_Toc530419630)

[1.1.2 实体类 3](#_Toc530419631)

[1.1.3 控制类 3](#_Toc530419632)

[1.2 构造交互图 3](#_Toc530419633)

[1.2.1 格子移动用例交互图 3](#_Toc530419634)

[1.2.2 历史分数显示用例交互图 4](#_Toc530419635)

[1.3 精化类图 4](#_Toc530419636)

[二、体系结构设计 5](#_Toc530419637)

[2.1 体系结构图 5](#_Toc530419638)

[2.2 架构风格 5](#_Toc530419639)

[三、构件设计 6](#_Toc530419640)

[3.1 接口设计 6](#_Toc530419641)

[3.2 算法和数据结构 6](#_Toc530419642)

[四、用户界面设计 8](#_Toc530419643)

[4.1 主界面设计 8](#_Toc530419644)

[4.2 游戏界面设计 9](#_Toc530419645)

[4.3 游戏结束界面设计 10](#_Toc530419646)

[五、数据持久设计 11](#_Toc530419647)

[5.1 定义数据格式 11](#_Toc530419648)

[5.2 定义数据存取操作 11](#_Toc530419649)

# 一、用例方案精化

## 1.1 提取边界类、实体类和控制类

### 1.1.1 边界类

边界类描述系统与外部环境的交互，主要任务包括界面控制、外部接口和环境隔离。在用例图中，执行者与用例之间的一种通信连接对应一个边界类。但是，如果两个以上的用例与同一执行者交互，并且这些交互具有共同的行为、 完成相同或类似的任务，就可以考虑用同一边界类实现用例与执行者之间的交互。分析以上用例图得出边界类如下所示：

|  |  |
| --- | --- |
| 类名 | 用途 |
| Move | 检测用户按下键盘的方向键 |
| DisplayBox | 显示数字格子 |
| DisplayScore | 显示分数 |

### 1.1.2 实体类

实体类表示目标软件系统中具有持久意义的信息项及其操作。本模块实体类如下所示：

|  |  |
| --- | --- |
| 类名 | 用途 |
| UserScore | 用户的历史分数 |

### 1.1.3 控制类

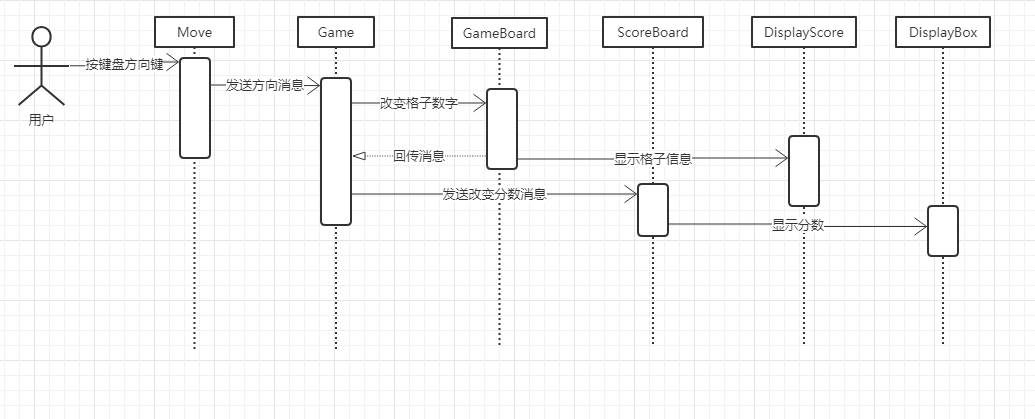
控制类作为完成用例任务的责任承担者，协调、控制其他类共同完成用例规定的功能或行为。

|  |  |
| --- | --- |
| 类名 | 用途 |
| Game | 处理游戏过程 |
| GameBoard | 处理游戏数字格子变动的功能 |
| ScoreBoard | 处理游戏分数的功能 |
| GameOver | 处理判定游戏是否结束功能 |

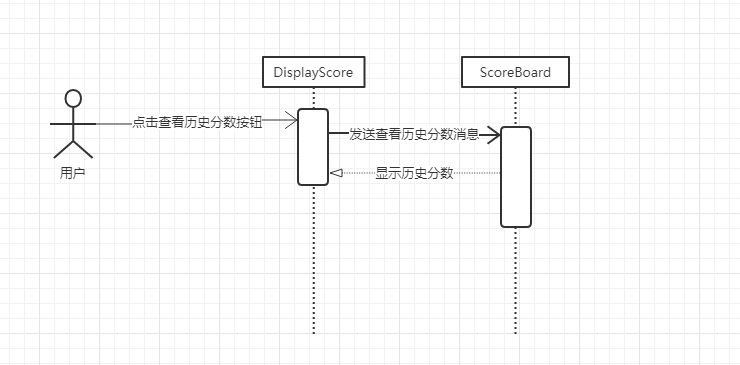
## 1.2 构造交互图

将分析模型中的用例描述转化成UML交互图， 以交互图作为用例的精确实现方案。部分用例描述及其顺序图如下所示：

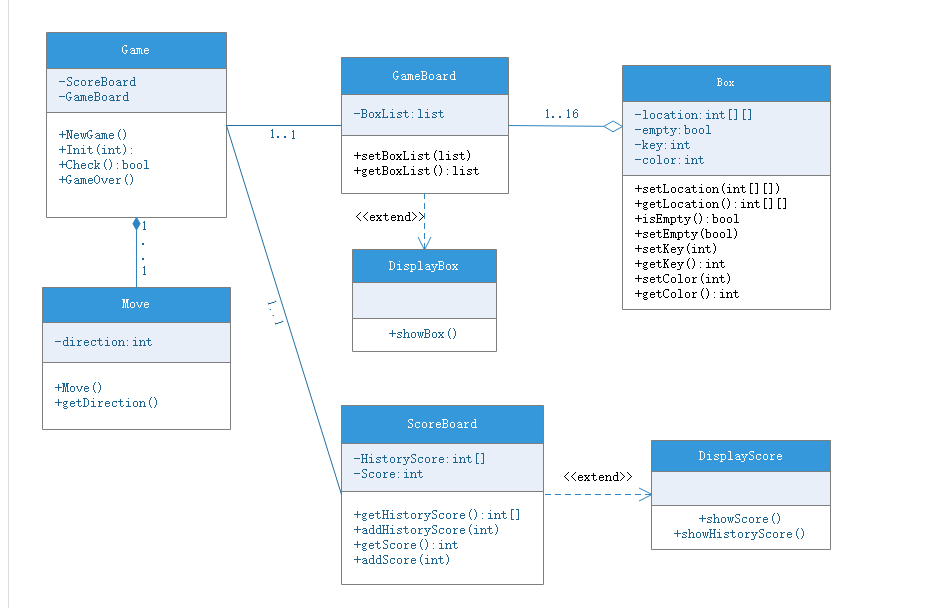
### 1.2.1 格子移动用例交互图



### 1.2.2 历史分数显示用例交互图



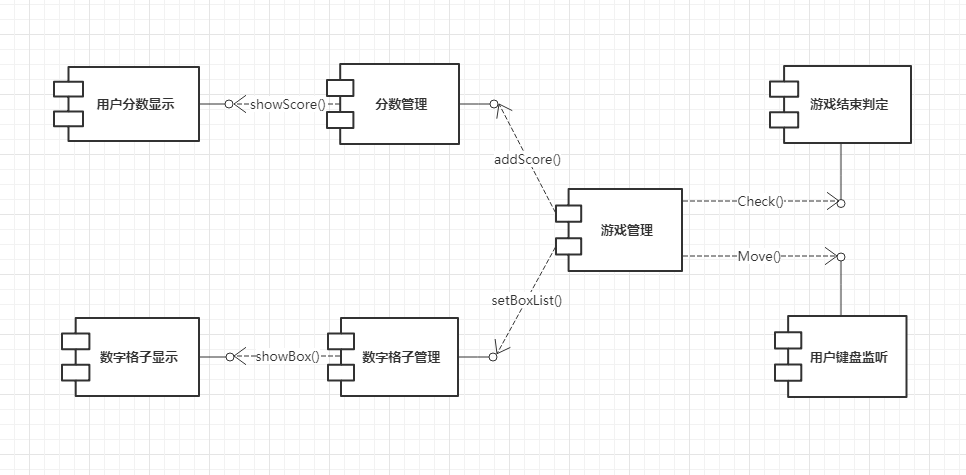
## 1.3 精化类图



# 二、体系结构设计

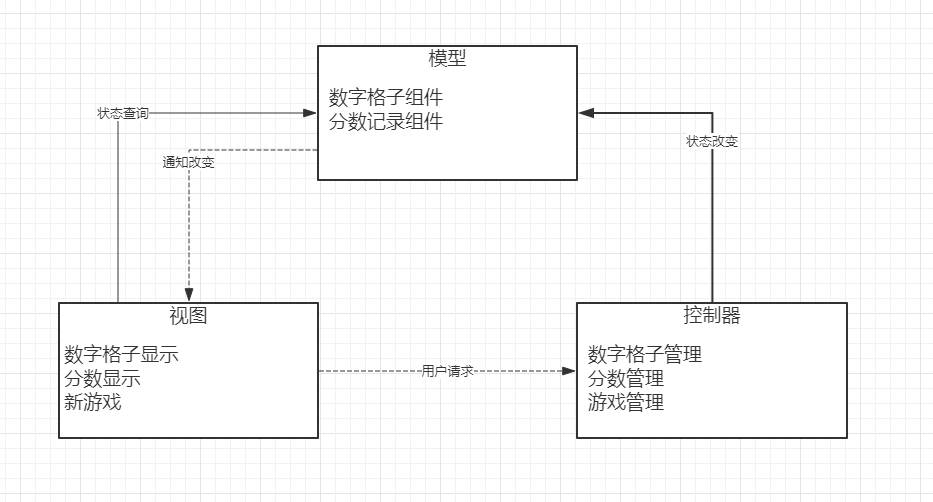
## 2.1 体系结构图

定义原始模型，系统的体系结构可有这些原始模型组成。原始模型可从前面的分析设计中获得；对体系结构设计是一个逐步精化得过程；将体系结构精化为构件，当体系结构精华为构件时，系统的结构开始显现。



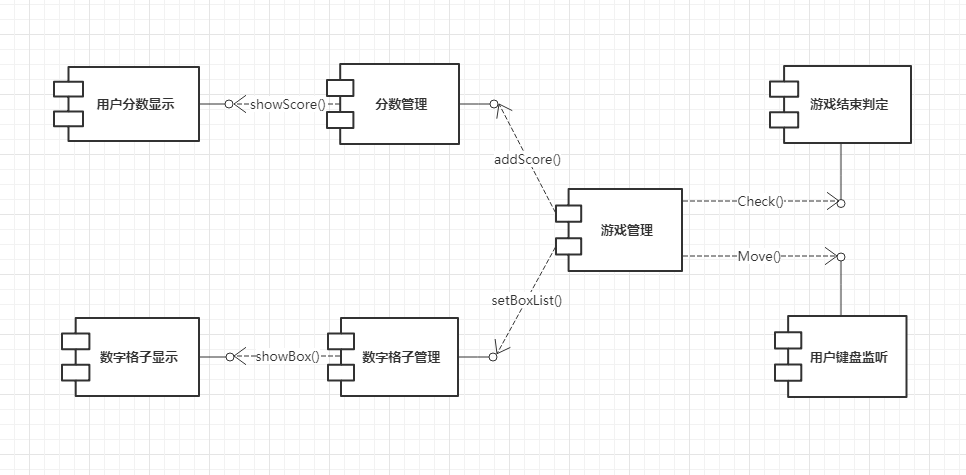
## 2.2 架构风格

该游戏采用MVC模式作为软件体系结构。MVC模式主要包含三个组件：模型、视图、控制器。模型，包含核心功能和数据（核心业务逻辑）。视图，向用户显示信息。控制器，处理用户输入。



# 三、构件设计

体系结构图就为构件图，以下是我们的构件图，包括构件和接口。



## 3.1 接口设计

对于用户分数显示组件，我们提供了一个showScore的接口，用来显示用户的分数。

对于数字格子显示组件，我们提供了一个showBox接口，用来显示每个格子的数字。

对于分数管理组件，提供一个addScore接口，用来接收改变当前分数的消息。

对于数字格子管理组件，提供一个setBoxList接口，用来接收改变每个格子中数字改变的消息。

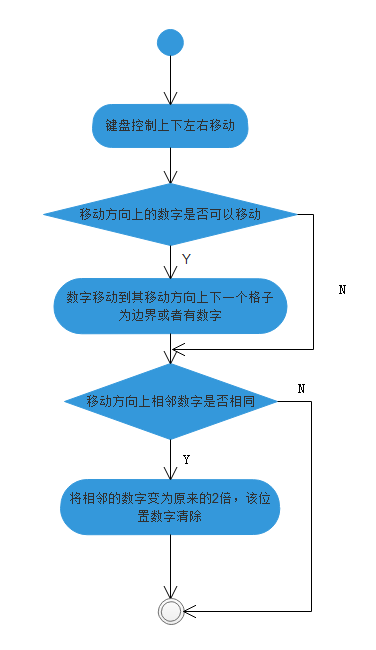
对于游戏结束判定组件，提供一个Check接口，每当用户操作一次时，游戏管理组件就调用一次该接口，判定游戏是否满足结束的条件。

对于用户键盘监听组件，提供一个Move接口，调用该接口，监听用户的操作，同时游戏管理类根据用户操作做出相应的反应。

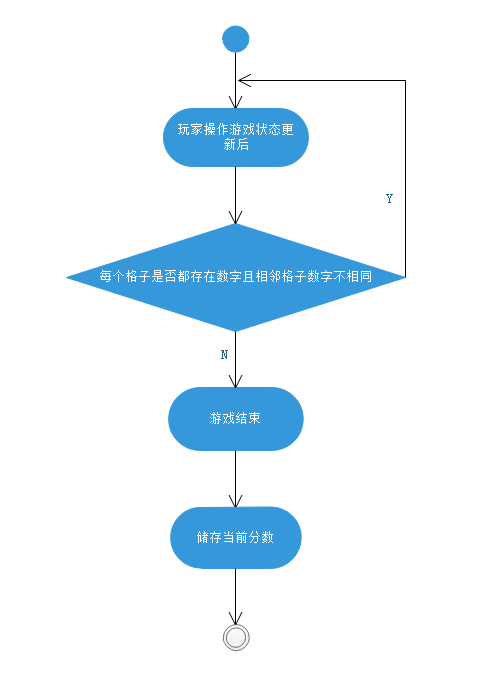
## 3.2 算法和数据结构

对本项目的主要算法采用流程图叙述。

1.对游戏移动合并数字的判断：

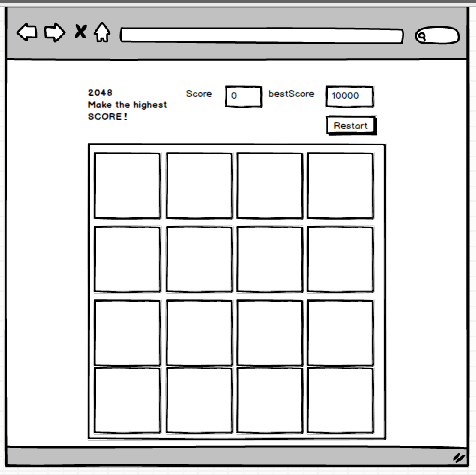


2.对游戏结束的判断：



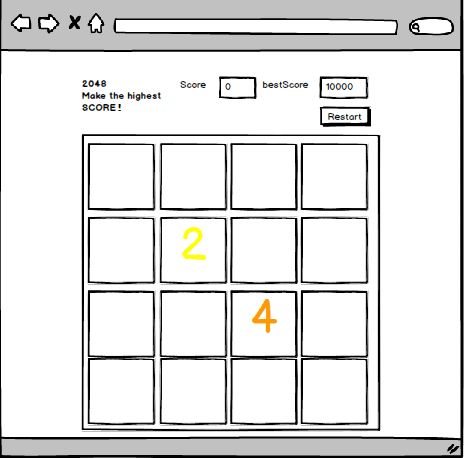
# 四、用户界面设计

## 4.1 主界面设计



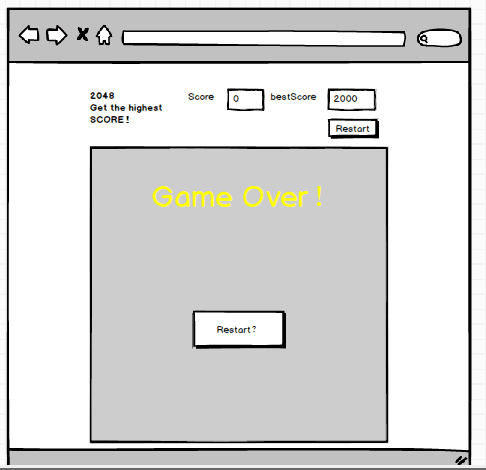
|  |  |  |
| --- | --- | --- |
| 控件名称 | 位置 | 说明 |
| 分数信息 | 上方 | 显示用户的当前分数以及历史最高分数 |
| 数字格子信息 | 下方 | 显示4×4的格子 |
| Restart | 右上方 | 点击该按钮将会重置分数以及数字格子开始新一局游戏 |

## 4.2 游戏界面设计



|  |  |  |
| --- | --- | --- |
| 控件名称 | 位置 | 说明 |
| 数字格子信息 | 下方 | 显示4×4的格子，并在每个格子中显示数字 |

## 4.3 游戏结束界面设计



显示“GameOver”的文字，并在下方有一个按钮“Restart”，点击该按钮可以进行一局新的游戏。

# 五、数据持久设计

## 5.1 定义数据格式

当每局游戏结束时，把用户该局的分数存储到用户本地的文件中，并按分数的高低对用户的历史分数进行排序，只存储前十高的用户分数。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 类型 | 关键字(yes/no) | 约束 |
| HistoryScore | Int | Yes | 默认值为0 |

## 5.2 定义数据存取操作

在判定游戏结束后，Game类调用ScoreBoard中的addHistoryScore函数，打开本地文件HistoryScore.txt文件，将此次的分数按规定存储进该文件。

每次开始新游戏时，Game类调用ScoreBoard中的getHistoryScore函数，打开本地文件HistoryScore.txt文件，读取第一行数字，即历史最高分，调用DisplayScore类将该分数显示。