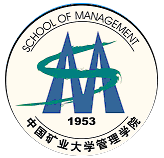
**中国矿业大学管理学院**

**技能训练报告**



**专业班级： 电子商务2014-3班**

**小组成员： 何东琴、黄琦珺、任芷霖**

**学 号：09143703、09143794、09143797**

**指导教师： 杨明智**

**选择题目： 2048小游戏**

**2017年9月**

目 录

[一、2048介绍 - 3 -](#_Toc493110067)

[二、系统分析和系统中类的设计 - 3 -](#_Toc493110068)

[（1）数据交换对象Dto类 - 3 -](#_Toc493110069)

[（2）游戏资源类Resources - 3 -](#_Toc493110070)

[（3）游戏服务类gameService - 3 -](#_Toc493110071)

[（4）游戏控制类GameControl - 4 -](#_Toc493110072)

[（5）游戏分数类Score - 4 -](#_Toc493110073)

[（6）游戏背景类Background - 4 -](#_Toc493110074)

[（7）游戏计分板类Scoreboard - 4 -](#_Toc493110075)

[（8）游戏主界面类MainFrame - 4 -](#_Toc493110076)

[（9）游戏显示类MainGame - 4 -](#_Toc493110077)

[（10）游戏统筹类MainPanel - 4 -](#_Toc493110078)

[（11）游戏开始类Start - 4 -](#_Toc493110079)

[三、系统的调试和实现 - 5 -](#_Toc493110080)

[四、系统的使用说明 - 5 -](#_Toc493110081)

[附1：完成该系统的所有成员名单及其工作 - 7 -](#_Toc493110082)

[附2：程序部分代码 - 7 -](#_Toc493110083)

## 一、2048介绍

2048游戏是一款在Android/ios平台上很流行的休闲游戏，游戏的规则很简单：游戏开始时，在一个4×4的方格中随机生成链两个数字值为2或4的方块，通过上下左右四个方向的移动，使得数字值相同的两个方块合并成一个值为两者之和的方块，每移动一次，会在未被占用的方格中生成一个2或4的方块，在所有方格未被占满且无法移动之前合成出2048，则游戏胜利，否则失败。

## 二、系统分析和系统中类的设计

根据游戏规则和面向对象Java程序设计要求，程序的主要类有以下几种：

### （1）数据交换对象Dto类

Dto类用于储存游戏需要用到的主要数据，并定义用于返回和修改其中私有数据成员的公共接口方法；

其中的数据成员有:

当前得分int score；

最高分int highScore；

最高分更新标记boolean needToRefresh；

次背景横坐标int x；

次坐标纵坐标int y;

方块所处位置和数值对应图片的索引值 int[4][4] gameMap;

### （2）游戏资源类Resources

Resources类用于提供游戏需要使用到所有文件的引用，包括方块2~方块2048、背景和次背景、得分和最高分计分板、分数数字的png图片文件，以及最高分记录txt文件，还定义了播放游戏音效的方法playSound()。

### （3）游戏服务类gameService

gameService类用于实现游戏功能需要调用的一些接口方法：

从文件中载入最高分loadHighScore();

更新文件中的最高分refreshHighscore();

重新开始restart();

生成一个新方块newBlock();

判断游戏是否结束isGameOver();

向上移动moveUp();

向下移动moveDown();

向左移动moveLeft();

向右移动moveRight();

向上消除removeUp();

向下消除removeDown();

向左消除removeLeft();

向右消除removeRight()。

### （4）游戏控制类GameControl

GameControl类用于接收从键盘输入按键键值，并根据不同的键值调用不同的方法；

按←键，键值为37，调用GameService类中的moveleft();和removeLeft();方法实现向左移动和向左消除；根据返回值isMove和isRemove判断移动是否有效，决定是否生成新的方块和播放移动或禁止移动音效；

按↑↓→键同理；

按f1，键值为112，可以再游戏中途重新开始；

按esc，键值为27，可在游戏中途退出。

### （5）游戏分数类Score

将当前得分和最高分转化成数字图片并显示在指定位置。

### （6）游戏背景类Background

在指定的坐标载入游戏的背景和次背景图。

### （7）游戏计分板类Scoreboard

在指定的坐标载入游戏的得分和最高分计分板图片。

### （8）游戏主界面类MainFrame

设置游戏主界面的标题，大小，可缩放性，可见性，并定义窗口监听器，在点击关闭按钮时弹出确认窗口。

### （9）游戏显示类MainGame

通过接收Dto类中存储方块位置和数值的gameMap数组，决定在相应位置显示索引所对应图片。

### （10）游戏统筹类MainPanel

通过实例化Background、Scoreboard、Score、MainGame、Dto、gameService类，实现游戏界面各部分的显示，并统一传递参数来实现类与类之间的数据交换。

### （11）游戏开始类Start

游戏开始的入口，实例化Dto、GameService、MainFrame类，统一传递参数实现数据交换。

## 三、系统的调试和实现

程序代码完成的初期有很多不完善的地方：

数据上的，一些参数没有统一传输，导致该参数在A类中更新了在B类中好是旧数据的现象，最后通过定义Start和MainPanel两个类是保证了数据交换的准确性和即时性；

界面上的，一开始的时候分数显示不居中，查看API接口drawImage，通过更改参数，使分数字符串长度与之关联，实现居中显示；

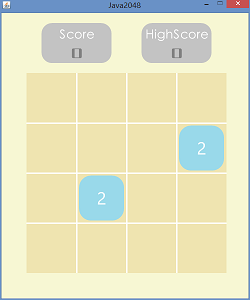
开始时的界面很丑，配色也很不和谐，通过上网查找配色方案和修图，终于使界面更美观；

逻辑上的，一开始的时候新方块出现2和4的几率是相等的，自己测试的时候发现很难玩，4出现的几率太高了，导致方格很容易就满了，于是修改方块生成函数返回值，确定八分之一是一个比较合适的概率。

功能上的，组员提醒游戏中途不能重新开始，于是加了一个重新开始的功能，为了不破坏界面风格的统一性，没有使用按钮而是快捷键f1来调用重新开始。

## 四、系统的使用说明

在安装配置好Java运行环境（jre）的电脑上，双击打包好的2048.jar文件，出现游戏开始界面，通过键入上下左右来控制方块的合并，移动方块的同时会发出特定音效，当移动违规时，没有方块生成并播放警告音，



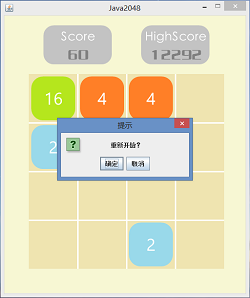
在方块占满方格切无法继续移动之后，弹出游戏结束提示框，显示最终得分，播放游戏结束音效，并提示选择重新开始或结束游戏。



若达成2048，弹出提示框显示游戏成功，播放游戏成功音效，并提示选择重新开始还是退出游戏。



游戏中途按f1，弹出提示框是否重新开始。



## 附1：****完成该系统的所有成员名单及其工作****

何东琴：类Dto、GameControl、GameService、MainPanel的编写，部分素材的搜集和修改。

任芷霖：类Start、MainFrame、Resources、Score的编写，部分素材的搜集和修改。

黄琦珺：类MainGame、Background、Scoreboard的编写，部分素材的搜集和修改。

## 附2：程序部分代码

#### Dto类

**package** data;

**public** **class** Dto {

**private** **int** score = 0;

**private** **int** highScore = 0;

**private** **boolean** needToRefresh = **false**;

**private** **int** x; //次背景的横坐标

**private** **int** y; //次背景的纵坐标

**private** **int**[][] gameMap = **new** **int**[4][4];

**public** **int** getScore() {

**return** score; //返回游戏分数

}

**public** **void** setScore(**int** s) {

score = s; //设置游戏分数

}

**public** **int** getHighScore(){

**return** highScore; //返回最高分

}

**public** **void** setHighScore(**int** hs){

highScore = hs; //设置最高分

}

**public** **boolean** getNeedToReresh(){

**return** needToRefresh;

}

**public** **void** setNeedToRefresh(**boolean** ntr){

needToRefresh = ntr;

}

**public** **int**[][] getGameMap() {

**return** gameMap; //返回方块数字对应号码

}

**public** **void** setGameMap(**int**[][] gameMap) {

**this**.gameMap = gameMap; //设置方块数字对应索引

}

//返回和设置此背景坐标

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

}

#### Resources类

**package** data;

**import** java.awt.Image;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** sun.audio.AudioPlayer;

**import** sun.audio.AudioStream;

**import** javax.swing.ImageIcon;

//导入资源文件

**public** **class** Resources {

**public** **static** Image *img\_num* = **new** ImageIcon("res/num.png").getImage();

**public** **static** Image *img\_fg* = **new** ImageIcon("res/fg.png").getImage();

**public** **static** Image *img\_bg* = **new** ImageIcon("res/bg.png").getImage();

**public** **static** Image *img\_score* = **new** ImageIcon("res/score.png").getImage();

**public** **static** Image *img\_highscore* = **new** ImageIcon("res/highscore.png").getImage();

**public** **static** Image *imgOf2* = **new** ImageIcon("res/2.png").getImage();

**public** **static** Image *imgOf4* = **new** ImageIcon("res/4.png").getImage();

**public** **static** Image *imgOf8* = **new** ImageIcon("res/8.png").getImage();

**public** **static** Image *imgOf16* = **new** ImageIcon("res/16.png").getImage();

**public** **static** Image *imgOf32* = **new** ImageIcon("res/32.png").getImage();

**public** **static** Image *imgOf64* = **new** ImageIcon("res/64.png").getImage();

**public** **static** Image *imgOf128* = **new** ImageIcon("res/128.png").getImage();

**public** **static** Image *imgOf256* = **new** ImageIcon("res/256.png").getImage();

**public** **static** Image *imgOf512* = **new** ImageIcon("res/512.png").getImage();

**public** **static** Image *imgOf1024* = **new** ImageIcon("res/1024.png").getImage();

**public** **static** Image *imgOf2048* = **new** ImageIcon("res/2048.png").getImage();

**public** **static** File *record* = **new** File("res/record.txt");

//播放游戏音效的静态方法，参数是声音文件名

**public** **static** **void** playSound(String soundName)

{

**try** {

FileInputStream in = **new** FileInputStream(soundName);

AudioStream as = **new** AudioStream(in);

AudioPlayer.*player*.start(as);

} **catch** (Exception e) {}

}

}

#### GameService类

**package** service;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.util.Random;

**import** javax.swing.JOptionPane;

**import** data.Dto;

**import** data.Resources;

**public** **class** GameService{

**private** **int** Score;

**private** Random random;

**private** Dto dto;

**private** **int**[][] gameMap;

//构造函数，通过传递dto的参数来初始化GamedService类

**public** GameService(Dto dto) {

**this**.random = **new** Random();

**this**.dto = dto;

**this**.Score = dto.getScore();

**this**.gameMap = dto.getGameMap();

}

**public** **void** loadHighScore() //从文件载入最高分

{

BufferedReader br;

**try**{

br=**new** BufferedReader(**new** FileReader(Resources.*record*));

dto.setHighScore(Integer.*valueOf*(br.readLine()).intValue());

br.close();

}**catch**(Exception e){

JOptionPane.*showMessageDialog*(**null**,"找不到同目录下的Record文件或文件已损坏!", "提示", JOptionPane.*INFORMATION\_MESSAGE*);

System.*exit*(0);

}

}

**public** **void** refreshHighscore()

{

**if**(dto.getNeedToReresh()){

BufferedWriter bw;

**try**{

bw=**new** BufferedWriter(**new** FileWriter(Resources.*record*));

bw.write(String.*valueOf*(dto.getHighScore()),0,String.*valueOf*(dto.getHighScore()).length());

bw.close();

}**catch**(Exception e1){

JOptionPane.*showMessageDialog*(**null**,"找不到同目录下的Record文件或文件已损坏!", "提示", JOptionPane.*INFORMATION\_MESSAGE*);

};

}

}

**public** **void** restart() //重新开始，地图、分数清零，生成2个方块并入栈

{

dto.setScore(0);

dto.setNeedToRefresh(**false**);

**this**.dto.setGameMap(**new** **int**[4][4]);

gameMap = **new** **int**[4][4];

Score = 0;

newBlock();

newBlock();

}

// 获取新坐标

**public** **int** getBlockPos() {

**int** pos = random.nextInt(4);//得到一个0到3的随机数，用于确定新生成方块的位置

**return** pos;

}

// 获取新方块号

**public** **int** get2Or4() {

**int** x = random.nextInt(8); //得到一个1到2的随机数，用于确定新方块中是2或4,得到4的概率约为1/8

**if**(x != 0)

**return** 1;

**else**

**return** 2;

}

// 新建一个方块

**public** **void** newBlock() {

**int** x = getBlockPos();

**int** y = getBlockPos();

//若方块已被占用，随机另外一个坐标

**while** (gameMap[x][y] != 0) {

x = getBlockPos();

y = getBlockPos();

}

//获取一个方块编号

gameMap[x][y] = get2Or4();

**this**.dto.setGameMap(gameMap);

}

// 判断地图是不是满

**public** **boolean** isFull() {

**int** blockCount = 0;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 0; j < 4; j++) {

**if** (gameMap[i][j] != 0) {

blockCount++;

}

}

}

**if** (blockCount == 16 && !canRemove()) {

**return** **true**;

}

**else** {

**return** **false**;

}

}

//判断是否可以移动

**public** **boolean** canRemove() {

**boolean** canRemove = **false**;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 0; j < 4; j++) {

**if**(i + 1 < 4)

**if**(gameMap[i][j] == gameMap[i + 1][j]){

canRemove = **true**;

**break**;

}

**if**(i + 1 < 4)

**if**(gameMap[i][j] == gameMap[i + 1][j]){

canRemove = **true**;

**break**;

}

**if**(i - 1 > -1)

**if**(gameMap[i][j] == gameMap[i - 1][j]){

canRemove = **true**;

**break**;

}

**if**(j - 1 > -1)

**if**(gameMap[i][j] == gameMap[i][j - 1]){

canRemove = **true**;

**break**;

}

}

}

**return** canRemove;

}

//判断游戏是否结束

**public** **void** isGameOver() {

**if** (isFull()) {

Resources.*playSound*("res/fail.wav");

Object[] options = {"重新开始","退出游戏"};

**int** choice = JOptionPane.*showOptionDialog*(

**null**,

"\no(╯□╰)o 游戏结束!\n 分数为：

"提示",

JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*,

**null**, options,

options[0]

);

refreshHighscore();

//选择重新开始还是结束游戏

**if**(choice ==0){

restart();

}

**else** {

System.*exit*(0);

}

}

**else** **if**(is2048()) {

Resources.*playSound*("res/victory.wav");

Object[] options = {"重新开始","退出游戏"};

**int** choice = JOptionPane.*showOptionDialog*(

**null**,

" Y^o^Y恭喜你！达成2048 ",

"提示",

JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*,

**null**, options,

options[0]

);

refreshHighscore();

//选择重新开始还是结束游戏

**if**(choice ==0){

restart();

}

**else** {

System.*exit*(0);

}

}

}

// 判断是否合成2048

**public** **boolean** is2048() {

**boolean** isok = **false**;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 0; j < 4; j++) {

**if** (gameMap[i][j] == 11) {

isok = **true**;

}

}

}

**return** isok;

}

// 左移

**public** **int** moveLeft() {

**int** isMove = 0;

**for** (**int** j = 0; j < 4; j++) {

**for** (**int** i = 1; i < 4; i++) {

**int** mov\_i = i;

**int** mov\_j = j;

**while** (gameMap[mov\_i][mov\_j] != 0

&& gameMap[mov\_i - 1][mov\_j] == 0) {

gameMap[mov\_i - 1][mov\_j] = gameMap[mov\_i][mov\_j];

gameMap[mov\_i][mov\_j] = 0;

**if** (mov\_i > 1) {

mov\_i--;

}

isMove = 1;

}

}

}

**return** isMove;

}

// 右移

**public** **int** moveRight() {

**int** isMove = 0;

**for** (**int** j = 0; j < 4; j++) {

**for** (**int** i = 2; i >= 0; i--) {

**int** mov\_i = i;

**int** mov\_j = j;

**while** (gameMap[mov\_i][mov\_j] != 0

&& gameMap[mov\_i + 1][mov\_j] == 0) {

gameMap[mov\_i + 1][mov\_j] = gameMap[mov\_i][mov\_j];

gameMap[mov\_i][mov\_j] = 0;

**if** (mov\_i < 2) {

mov\_i++;

}

isMove = 1;

}

}

}

**return** isMove;

}

// 下移

**public** **int** moveDown() {

**int** isMove = 0;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 2; j >= 0; j--) {

**int** mov\_i = i;

**int** mov\_j = j;

**while** (gameMap[mov\_i][mov\_j] != 0

&& gameMap[mov\_i][mov\_j + 1] == 0) {

gameMap[mov\_i][mov\_j + 1] = gameMap[mov\_i][mov\_j];

gameMap[mov\_i][mov\_j] = 0;

**if** (mov\_j < 2) {

mov\_j++;

}

isMove = 1;

}

}

}

**return** isMove;

}

// 上移

**public** **int** moveUp() {

**int** isMove = 0;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 1; j < 4; j++) {

**int** mov\_i = i;

**int** mov\_j = j;

**while** (gameMap[mov\_i][mov\_j] != 0

&& gameMap[mov\_i][mov\_j - 1] == 0) {

gameMap[mov\_i][mov\_j - 1] = gameMap[mov\_i][mov\_j];

gameMap[mov\_i][mov\_j] = 0;

**if** (mov\_j > 1) {

mov\_j--;

}

isMove = 1;

}

}

}

**return** isMove;

}

// 左消

**public** **int** removeLeft() {

**int** isRemove = 0;

**for** (**int** j = 0; j < 4; j++) {

**for** (**int** i = 0; i < 3; i++) {

**while** (gameMap[i][j] == gameMap[i + 1][j]

&& gameMap[i][j] != 11 && gameMap[i][j] != 0) {

gameMap[i][j]++;

gameMap[i + 1][j] = 0;

// 加分

**this**.Score += Math.*pow*(2, gameMap[i][j]); //分数的增量为合并后产生的数字大小

**this**.dto.setScore(Score);

isRemove = 1;

}

}

}

moveLeft();

**return** isRemove;

}

// 右消

**public** **int** removeRight() {

**int** isRemove = 0;

**for** (**int** j = 0; j < 4; j++) {

**for** (**int** i = 3; i > 0; i--) {

**while** (gameMap[i][j] == gameMap[i - 1][j]

&& gameMap[i][j] != 11 && gameMap[i][j] != 0) {

gameMap[i][j] = gameMap[i - 1][j] + 1;

gameMap[i - 1][j] = 0;

// 加分

**this**.Score += Math.*pow*(2, gameMap[i][j]);

**this**.dto.setScore(Score);

isRemove = 1;

}

}

}

moveRight();

**return** isRemove;

}

// 上消

**public** **int** removeUp() {

**int** isRemove = 0;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 0; j < 3; j++) {

**while** (gameMap[i][j] == gameMap[i][j + 1]

&& gameMap[i][j] != 11 && gameMap[i][j] != 0) {

gameMap[i][j] = gameMap[i][j + 1] + 1;

gameMap[i][j + 1] = 0;

// 加分

**this**.Score += Math.*pow*(2, gameMap[i][j]);

**this**.dto.setScore(Score);

isRemove = 1;

}

}

}

moveUp();

**return** isRemove;

}

// 下消

**public** **int** removeDown() {

**int** isRemove = 0;

**for** (**int** i = 0; i < 4; i++) {

**for** (**int** j = 3; j > 0; j--) {

**while** (gameMap[i][j] == gameMap[i][j - 1]

&& gameMap[i][j] != 11 && gameMap[i][j] != 0) {

gameMap[i][j] = gameMap[i][j - 1] + 1;

gameMap[i][j - 1] = 0;

// 加分

**this**.Score += Math.*pow*(2, gameMap[i][j]);

**this**.dto.setScore(Score);

isRemove = 1;

}

}

}

moveDown();

**return** isRemove;

}

}

#### GameControl类

**package** control;

**import** java.awt.event.KeyAdapter;

**import** java.awt.event.KeyEvent;

**import** javax.swing.JOptionPane;

**import** data.Resources;

**import** service.GameService;

**import** ui.MainPanel;

**public** **class** GameControl **extends** KeyAdapter {

**private** MainPanel mainPanel;

**private** GameService gameService;

**public** GameControl(MainPanel mainPanel, GameService gameService) {

**this**.mainPanel = mainPanel;

**this**.gameService = gameService;

}

@Override

**public** **void** keyReleased(KeyEvent e) {

**super**.keyReleased(e);

**switch** (e.getKeyCode()) {

// 左移

**case** 37: {

**int** isMove = gameService.moveLeft();

**int** isRemove = gameService.removeLeft();

**if** (isMove==1||isRemove==1) {

gameService.newBlock();

Resources.*playSound*("res/slide.wav"); //播放滑动声音

}

**else**{

Resources.*playSound*("res/immovable.wav"); //播放不可移动声音

}

}

**break**;

// 上移

**case** 38: {

**int** isMove = gameService.moveUp();

**int** isRemove = gameService.removeUp();

**if** (isMove==1||isRemove==1) {

gameService.newBlock();

Resources.*playSound*("res/slide.wav");//播放滑动声音

}

**else**{

Resources.*playSound*("res/immovable.wav"); //播放不可移动声音

}

}

**break**;

// 右移

**case** 39: {

**int** isMove = gameService.moveRight();

**int** isRemove = gameService.removeRight();

**if** (isMove==1||isRemove==1) {

gameService.newBlock();

Resources.*playSound*("res/slide.wav");//播放滑动声音

}

**else**{

Resources.*playSound*("res/immovable.wav"); //播放不可移动声音

}

}

**break**;

// 下移

**case** 40: {

**int** isMove = gameService.moveDown();

**int** isRemove = gameService.removeDown();

**if** (isMove==1||isRemove==1) {

gameService.newBlock();

Resources.*playSound*("res/slide.wav");//播放滑动声音

}

**else**{

Resources.*playSound*("res/immovable.wav"); //播放不可移动声音

}

}

**break**;

**case** 27: {

**if**(JOptionPane.*showConfirmDialog*(

**null**,

" 退出游戏？",

"提示",

JOptionPane.*OK\_CANCEL\_OPTION*)== 0) {

System.*exit*(0);

};

}

**break**;

**case** 112: {

**if**(JOptionPane.*showConfirmDialog*(

**null**,

" 重新开始？",

"提示",

JOptionPane.*OK\_CANCEL\_OPTION*)== 0) {

gameService.restart();

}; //重新开始前提示

}

**break**;

**default**:

**break**;

}

**this**.mainPanel.repaint(); //刷新游戏窗口

gameService.isGameOver(); //判断是不是gameover

**this**.mainPanel.repaint();

}

}

#### Score类

**package** ui;

**import** java.awt.Graphics;

**import** data.Dto;

**import** data.Resources;

**public** **class** Score **extends** Scoreboard{

**private** **static** **final** **int** *SIZE\_NUM* = 21;

**private** **static** **final** **int** *SIZE\_SCORE* = 80;

**private** **int** numPos;

**private** Dto dto;

**public** Score(Dto dto) {

**this**.dto = dto;

}

//显示当前得分

**public** **void** drawScore(Graphics g){

//若当前得分大于最高分，最高分随当前得分更新，且高分更新标记设为真

**if**(dto.getScore() > dto.getHighScore())

{

dto.setHighScore(dto.getScore());

dto.setNeedToRefresh(**true**);

}

String scoreString = Integer.*toString*(dto.getScore());

**for** (**int** i = 0; i < scoreString.length(); i++) {

**int** scoreBit = scoreString.charAt(i)-'0';

**this**.numPos = scoreBit\**SIZE\_NUM*;

g.drawImage(Resources.*img\_num*,

**this**.s\_x-10+*SIZE\_SCORE*+i\**SIZE\_NUM*-scoreString.length()\*10,

**this**.s\_y+50,

**this**.s\_x-10+*SIZE\_SCORE*+*SIZE\_NUM*+i\**SIZE\_NUM*-scoreString.length()\*10,

**this**.s\_y+50+*SIZE\_NUM*,

numPos,

0,

*SIZE\_NUM*+numPos,

*SIZE\_NUM*,

**null**);

}

}

//显示最高分

**public** **void** drawHighScore(Graphics g) {

String highScoreString = Integer.*toString*(dto.getHighScore());

**for** (**int** i = 0; i < highScoreString.length(); i++) {

**int** HighScoreBit = highScoreString.charAt(i)-'0';

**this**.numPos = HighScoreBit\**SIZE\_NUM*;

g.drawImage(Resources.*img\_num*,

**this**.s\_x+190+*SIZE\_SCORE*+i\**SIZE\_NUM*-highScoreString.length()\*10,

**this**.s\_y+50,

**this**.s\_x+190+*SIZE\_SCORE*+*SIZE\_NUM*+i\**SIZE\_NUM*-highScoreString.length()\*10,

**this**.s\_y+50+*SIZE\_NUM*,

numPos,

0,

*SIZE\_NUM*+numPos,

*SIZE\_NUM*,

**null**);

}

}

}