目录

摘 要

RPG 游戏的主要宗旨就是让玩家自己制造属于自己的剧情，拥有自己的主观能动性。由于，本人对游戏极为痴迷，很小就想自己编写游戏，如今我可以实现这个梦想，所以选择了 RPG 游戏的制作，其中也因为 RPG 制作的灵活度比较高。

本次实验做的 RPG 口袋妖怪，有很大程度上模仿了原版游戏的风格，具备打怪升级，经验值获得，当经验值到一定临界值玩家的宠物就会升级。玩家在游戏设定的地图里可以自由移动（当然，障碍物时不能走的），在游戏地图中，还设置了三个 NPC（Non-Player-Controlled Character），分别是医生、商人和敌人，医生可以将玩家的宠物生命值完全恢复，商人可以让玩家买道具，而敌人是可以让玩家挑战的。

在战斗系统中，玩家可以选择使用道具，也可以选择逃跑，攻击的方式也较为简单，有攻击对方，治疗自己，防御对方的攻击以及和对方同归于尽。

在实现过程中，功能逻辑这一点的规划很重要，还有界面切换的效果也是一个小难点，动画是在开发中比较头疼的问题，因为在动画中总是出现闪烁的问题，后来解决的方式是通过双缓冲的技术来避免了闪烁的问题。

关键词：RPG 游戏元素，界面切换，动画效果

1. 课题背景及需求分析

1.1为什么选做 RPG 口袋妖怪这个游戏

RPG 游戏-全称 Role-playing Game，角色扮演游戏。本次 RPG 游戏选题主要来源于童年时玩的掌机--Gameboy，当时最喜爱的游戏就是口袋妖怪，曾经痴迷到了手绘了一个游戏地图，所以这次做这个也是圆我童年时的一个梦。

1.2 RPG 游戏的简单介绍

角色扮演游戏（Role-playing game）是一个游戏类型。角色扮演游戏的核心是扮演。在游戏玩法上，玩家扮演一位角色在一个写实或虚构的世界中活动。玩家负责扮演这个角色在一个结构化规则下通过一些行动令所扮演角色发展 。玩家在这个过程中的成功与失败取决于一个规则或行动方针的形式系统（Formal system）。

1.3 RPG 游戏的优势

这个游戏是 2D 低像素的画风，它的主要优势就是实现较为简单，贴图网上可以找到对应的，下载还需再稍微修整一下，一人可以在较短时间内开发出产品。这类游戏从一开始出现一直持续到现在，始终为大众所喜爱。因为这类游戏并不关心画质需要多么精细，玩家更多关注的是游戏性，比如游戏的剧情，游戏的操作方式等等。所以这类游戏的前景一直十分明朗。

2. 系统设计

这个游戏的主要内容是玩家拥有一个神奇宝贝，在一块地图上打怪升级。其中的具体内容有随机出现怪物与玩家对战，可以在医生那补血，可以找敌人单挑赚钱，可以从商人那买到精神草（玩家宠物生命值回满）、奇异果（玩家宠物升一级）和精灵球（抓捕怪物），背包调出使用道具。

技术难题有地图铺设方式走路动画的实现，攻击动画的实现，按键功能的实现，动态界面的实现，界面刷新时闪烁的问题，界面的切换，攻击的逻辑，怪物遇到的算

法，道具计算和使用的逻辑等等。

各类的属性和方法

玩家｛

属性：

宠物

道具：精灵球、生命草、奇异果

金钱（通过挑战敌人获得金钱奖励）

玩家坐标

玩家贴图数组

玩家方向

方法：

设置玩家位置；

设置玩家方向；

设置玩家贴图的索引号；

｝

宠物｛

属性：

生命值（与等级相关）、

等级、

当前经验值（获得经验值和敌人的等级相关）、

满经验值（与等级相关）、

基本信息（包括名字、性别等）

方法：

重置草丛遇见的怪物的参数（生命值、等级、贴图、名字等等）；

玩家被攻击的实现；

敌人被攻击的实现；

玩家治疗的实现；

玩家防御的实现；

判断是否为防御；

重置防御的状态；

自爆攻击方式的实现；

判断宠物是否死亡；

刷新当前对象的属性参数；（通过脉冲信号，每隔特定时间刷新一次）

获得经验值；

}

商人｛

给玩家加道具，

需要消耗金钱

}

医生｛

使玩家宠物满血复活

}

对手｛

挑战玩家，玩家赢了的到金钱奖励，输了扣除一定金钱

属性：

位置坐标

敌人贴图

}

战斗模式｛

属性：

战斗界面的坐标

指针的坐标

标志是否为第一次载入战斗界面

是否为战斗选项

是否可以关闭战斗界面

血条、经验值、等级、宠物名称、菜单等参数的宽度、坐标以及值

方法：

设置第一栏的指针

设置第二栏的指针

重置指针位置

}

探索模式{

属性：

玩家是否在移动；

玩家的方向；

}

背包模式{

属性：

道具名称和对应道具的数量的坐标和偏移量；

指针选择的道具的编号；

}

商人模式{

属性：

商人对话框的坐标和偏移量；

}

工具类{

属性：

各类贴图的定义

贴图宽高的定义

贴图数组的定义

是否是战斗界面

是否是探索界面

是否是背包界面

是否是商人模式

当前模式

是否开启攻击动画

是否开启走路动画

动画循环次数

最小贴图宽度

屏幕的宽高

方法：

设置贴图数组

}

贴图设定｛

地图大小固定，用矩阵表示，矩阵中值根据贴图编号设定

}

2.1 探索系统

玩家可以在地图上自由移动，地图上有三个人物分别是医生（帮助玩家宠物恢复生命值）、商人（可以购买精灵球、精神草和奇异果），在玩家移动过程中会随机遇见怪物，然后进入战斗界面。玩家在地图上移动过程中会遇到障碍物，游戏中设定遇到障碍物不能行走。

2.2 战斗系统

玩家在遇到怪物后，进入到战斗界面。战斗界面主要有四个选项，分别是逃跑（玩家可以逃离战斗，退回到探索界面）、怪兽（可以切换不同宠物来对战）、背包（玩家可以使用道具）以及战斗，战斗选项里又有四个选择，分别是攻击、防御、治疗、自爆（双方同归于尽，但游戏设定本局算玩家胜，这只是测试，后面会修改）。如果玩家赢了，则可以获得经验值，当经验值达到一定数量会升级。

2.3 背包系统

背包里包含道具，分别是精灵球（用于捕捉怪物）、精神草（将宠物生命值加满）和奇异果（宠物等级升一级）。

3. 系统开发平台

3.1 主机配置

虚拟机主机配置：

操作系统：win8.1，

CPU：i5-6400H

内存：8G

3.2 开发平台（开发环境或 IDE）:eclipse

Eclipse 是著名的跨平台开源集成开发环境（IDE）。最初主要用来 Java 语言开发，目前亦有人通过插件使其作为 C++、Python、PHP 等其他语言的开发工具。

Eclipse 的本身只是一个框架平台，但是众多插件的支持，使得 Eclipse 拥有较佳的灵活性，所以许多软件开发商以 Eclipse 为框架开发自己的 IDE。

历史

Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下一代 IDE 开发环境，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理。2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构。2007 年 6 月，稳定版 3.3 发布；2008 年 6 月发布代号为 Ganymede 的 3.4 版；2009 年 6 月发布代号为 Galileo 的 3.5 版；2010 年 6 月发布代号为 Helios 的 3.6 版；2011 年 6 月发布代号为 Indigo 的 3.7 版；2012 年 6 月发布代号为 Juno 的 4.2 版；2013 年 6 月发布代号为 Kepler 的 4.3 版；2014 年 6 月发布代号为 Luna 的 4.4 版；2015 年 6 月发布代号为 Mars 的 4.5 版。

架构

Eclipse 的基础是富客户机平台（即 RCP）。RCP 包括下列组件：

核心平台（启动 Eclipse，运行插件）

OSGi（标准集束框架）

SWT（可移植构件工具包）

JFace（文件缓冲，文本处理，文本编辑器）

Eclipse 工作台（即 Workbench，包含视图（views）、编辑器（editors）、视角（perspectives）、和向导（wizards））

Eclipse 采用的技术是 IBM 公司开发的(SWT),这是一种基于 Java 的窗口组件,类似 Java 本身提供的 AWT 和 Swing 窗口组件;不过 IBM 声称 SWT 比其他 Java 窗口组件更有效率。Eclipse 的用户界面还使用了 GUI 中间层 JFace，从而简化了基于 SWT 的应用程序的构建。

3.3 编程语言:Java

Java 是一种计算机编程语言，拥有跨平台、面向对象、泛型编程的特性，广泛应用于企业级 Web 应用开发和移动应用开发。

任职于太阳微系统的詹姆斯·高斯林等人于 1990 年代初开发 Java 语言的雏形，最初被命名为 Oak，目标设置在家用电器等小型系统的程序语言，应用在电视机、电话、闹钟、烤面包机等家用电器的控制和通信。由于这些智能化家电的市场需求没有预期的高，Sun 公司放弃了该项计划。随着 1990 年代互联网的发展，Sun 公司看见 Oak 在互联网上应用的前景，于是改造了 Oak，于 1995 年 5 月以 Java 的名称正式发布。Java 伴随着互联网的迅猛发展而发展，逐渐成为重要的网络编程语言。

Java 编程语言的风格十分接近 C++语言。继承了 C++语言面向对象技术的核心，Java 舍弃了 C++语言中容易引起错误的指针，改以引用替换，同时移除原 C++与原来运算符重载，也移除多重继承特性，改用接口替换，增加垃圾回收器功能。在 Java SE 1.5 版本中引入了泛型编程、类型安全的枚举、不定长参数和自动装/拆箱特性。太阳微系统对 Java 语言的解释是:“Java 编程语言是个简单、面向对象、分布式、解释性、健壮、安全与系统无关、可移植、高性能、多线程和动态的语言”

Java 不同于一般的编译语言或直译语言。它首先将源代码编译成字节码，然后依赖各种不同平台上的虚拟机来解释执行字节码,从而实现了“一次编写,到处运行”的跨平台特性。在早期 JVM 中，这在一定程度上降低了 Java 程序的运行效率。但在 J2SE1.4.2 发布后，Java 的运行速度有了大幅提升。

与传统类型不同，Sun 公司在推出 Java 时就将其作为开放的技术。全球数以万计的 Java 开发公司被要求所设计的 Java 软件必须相互兼容。“Java 语言靠群体的力量而非公司的力量”是 Sun 公司的口号之一，并获得了广大软件开发商的认同。这与微软公司所倡导的注重精英和封闭式的模式完全不同，此外，微软公司后来推出了与之竞争的.NET 平台以及模仿 Java 的 C#语言。后来 Sun 公司被甲骨文公司并购,Java 也随之成为甲骨文公司的产品。

3.4 引用的库

Java swing

Swing 是一个为 Java 设计的 GUI 工具包。Swing 是 Java 基础类的一部分。Swing 包括了图形用户界面（GUI）组件如：文本框，文本域，按钮，分隔窗格和表。

Swing 提供许多比 AWT 更好的屏幕显示元素。它们用纯 Java 写成，所以同 Java 本身一样可以跨平台运行，这一点不像 AWT。它们是 JFC 的一部分。它们支持可更换的面板和主题（各种操作系统默认的特有主题），然而不是真的使用原生平台提供的设备，而是仅仅在表面上模仿它们。这意味着你可以在任意平台上使用 Java 支持的任意面板。轻量级组件的缺点则是执行速度较慢，优点就是可以在所有平台上采用统一的行为。

Java awt

Java 释出的时候，AWT 作为 Java 最弱的组件受到不小的批评。最根本的缺点是 AWT 在原生的用户界面之上仅提供了一个非常薄的抽象层。例如，生成一个 AWT 的复选框会导致 AWT 直接调用下层原生例程来生成一个复选框。不幸的是，一个 Windows 平台上的复选框同 MacOS 平台或者各种 UNIX 风格平台上的复选框并不是那么相同。

这种糟糕的设计选择使得那些拥护 Java“一次编写，到处运行（write once, run everywhere）”信条的程序员们过得并不舒畅，因为 AWT 并不能保证他们的应用在各种平台上表现得有多相似。一个 AWT 应用可能在 Windows 上表现很好可是到了 Macintosh 上几乎不能使用，或者正好相反。在 90 年代，程序员中流传着一个笑话：Java 的真正信条是“一次编写，到处测试（write once, test everywhere）”。导致这种糟糕局面的一个可能原因据说是 AWT 从概念产生到完成实现只用了一个月。

在第二版的 Java 开发包中，AWT 的器件很大程度上被 Swing 工具包替代。Swing 通过自己绘制器件而避免了 AWT 的种种弊端：Swing 调用本地图形子系统中的底层例程，而不是依赖操作系统的高层用户界面模块。

Java util

Java 工具类

4. 系统实现

4.1 探索界面

通过一个二维矩阵定义地图，实现原理就是将每块编码，然后每块的编码对应各自的贴图，然后通过循环将所有贴图画在面板上。

图 4.1 水池贴图

图 4.2 商人、敌人、医生的贴图

图 4.3 界面的设计效果

障碍物不能行走实现方式是通过障碍物特定编码作为特征来判断是否为障碍物，如果是，行走功能触发无效。

图 4.4 小树、石头、小草的贴图

整个游戏的操作，包括玩家行走的功能，通过键盘监听器 KeyListener 来实现。当检测到按下指定按键时，游戏将作出对应的响应，比如按下方向键，将玩家的坐标做相应的改变，然后再重绘整个面板。

```
public void keyPressed(KeyEvent e) {
    switch (e.getKeyCode()) {
        case KeyEvent.VK_UP:
            if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)   ;
            else        up();
            break;
        case KeyEvent.VK_DOWN:
            if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)   ;
            else        down();
            break;
        case KeyEvent.VK_LEFT:
            if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)   ;
            else        left();
            break;
        case KeyEvent.VK_RIGHT:
            if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)   ;
            else        right();
            break;
       case KeyEvent.VK_Z:
            if (MyUtil.isBagMode)       bagModeHandle();
            else if (MyUtil.ACTTACK_MODE) battleModeHandle();
            else if (MyUtil.isBussinessMode)    bussinessModeHandle();
            else if (MyUtil.isDoctorMode)   MyUtil.isDoctorMode = false;
            else if (MyUtil.isEnemyMode)    MyUtil.isEnemyMode = false;
            else        discoveryModeHandle();
            break;
       case KeyEvent.VK_X:
            if (MyUtil.ACTTACK_MODE && MyUtil.isBagMode)
                MyUtil.isBagMode = false;
            if (!MyUtil.ACTTACK_MODE && MyUtil.isBussinessMode) {
                MyUtil.isBussinessMode = false;
                repaint();
            }
            if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)   ;
            else {
                battleMode.FistLoad = true;
                battleMode.isFighting = false;
            }
            break;
        case KeyEvent.VK_A:
            if (MyUtil.isBagMode) { // 切换出背包界面
                MyUtil.isBagMode = false;
                repaint();
            } else if (!MyUtil.ACTTACK_MODE) {
                MyUtil.isBagMode = true;
```

```
                repaint();
            }
                break;
        default:
                break;
    }
}
```

人物对话功能，当玩家走到指定人物的四周便可以触发对话事件。

为了让这个界面看起来是活的，所以我在探索界面设置了一个脉冲信号，每隔 1 秒刷新界面，并每一秒随机改变医生、敌人和商人的转向，以此来到的动态的效果。

脉冲信号通过 Timer 类中的 timetask 产生的线程来实现，下面是代码片段。

```
timer = new Timer();
timer.schedule(new PulseSignalTask(), 1000, 1000);
public class PulseSignalTask extends TimerTask {
            public void run() {
                doctor.index = (int) (Math.random() * 1000 % 2);
                bussinessman.index = (int) (Math.random() * 1000 % 3);
                enemy.index = (int) (Math.random() * 1000 % 3);
                …… …… ……
                repaint();
            }
}
```

图 4.5 奔跑的预览效果

从上面代码中可以看出我是通过 math 类中的 random 方法生成随机数来作为贴图编号。通过取余来限制生成随机数的范围，使生成的随机数不会超过贴图数组的边界。

玩家的移动的动画的实现较为复杂。首先，需要玩家各个方向行走的贴图各两张，分别是迈左脚和迈右脚。在移动过程中通过不断改变他的坐标偏移直至偏移至下一个贴图区域。开始实现时出现了非常有趣的现象------当我移动到下一个的过程中，角色竟然走到了背景后面去，然后就消失了。后来发现原来草坪是晚于角色绘制的，所以当角色移动时后来画的草就将角色给覆盖了。于是我将草坪的绘制放在了最前面，这样角色移动时都是在草坪上。动画的设置是通过一个循环来实现，再加一个循环来实现动画重播。

图 4.6 上图是行走动画的每一帧贴图

通过循环扫描整个画板判断玩家状态的坐标，然后通过玩家状态来决定是否开启走路动画效果。

实现该功能的关键代码:

```
for (int i = 0; i < 200; i++)
    if (i == (player.postionX + player.postionY * 20)) {
        if (!discoveryMode.isMoving)
          g.drawImage(参数);          //角色静态贴图的绘制
      else
```

walkEffect(图像对象 g，玩家行走的方向);

以下是玩家分为 4 个方向的动画效果处理。

walkEffect(图像对象 g，玩家行走的方向) 方法的实现代码片段：

```
switch (玩家行走的方向) {
case 1:
    g.drawImage(玩家向左走的贴图,
            当前 x 坐标值 - x 偏移值，当前 y 坐标值，
            贴图的宽，贴图的高)
    break;
case 2:
    g.drawImage(玩家向右走的贴图,
            当前 x 坐标值 + x 偏移值,当前 y 坐标值，
            贴图的宽，贴图的高)
    break;
case 3:
    g.drawImage(玩家向下走的贴图,
            当前 x 坐标值，当前 y 坐标值 + y 偏移值，
            贴图的宽，贴图的高)
    break;
case 4:
    g.drawImage(玩家向上走的贴图,
            当前 x 坐标值，当前 y 坐标值 - y 偏移值，
            贴图的宽，贴图的高)
    break;
default:
    break;
}
```

在每个对应方向的触发方法中都会调用以下类似的代码，通过 sleep（）方法来延迟

动画，实现一秒多少帧画面的切换。人眼判读画面是否卡顿的临界帧数是 25 帧，帧数越大，画面越流畅。

调用的代码片段：

```
for (int i = 0; i < 帧数; i++) {
        Thread.sleep(一次动画的时间 / 2);
        MyUtil.偏移量 = i;
        player.setIndex(左脚贴图编号);
        repaint();
        Thread.sleep(一次动画的时间 / 2);
        player.setIndex(右脚贴图编号);
        repaint();
}
```

4.2 战斗界面

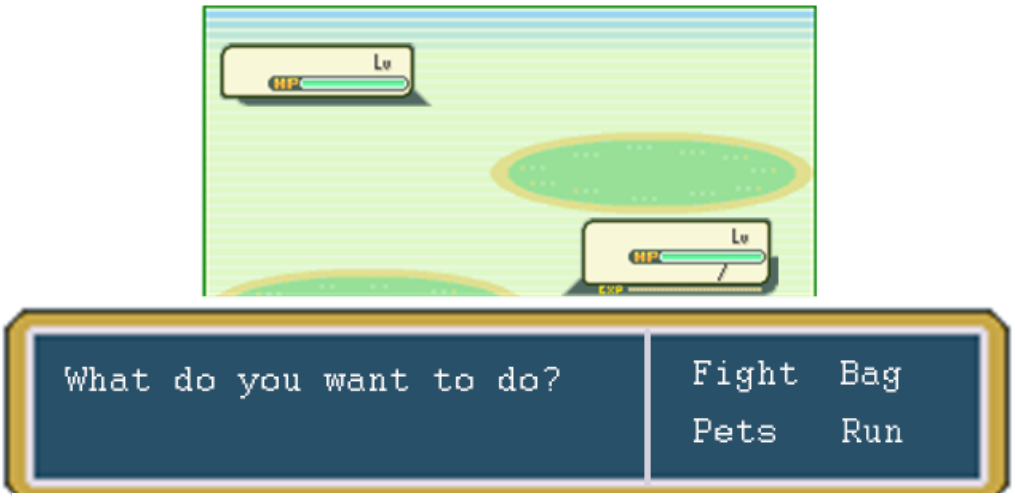

图 4.7 战斗模式主界面

逃跑功能的实现就是将是否是战斗界面的值改为 false，然后重绘。

背包界面功能的实现就是将是否为背包界面的值改为 true，然后重绘。

战斗界面功能的实现就是将是否战斗的值改为 true，然后重绘。

进入战斗界面后，将有四个选项。

双方进行回合制的攻击方式的实现，通过定义了 turn 变量，该变量是属于 Pet 宠物类，当玩家的回合结束后将自己的 turn 变量设为 false，将对方的 turn 变量设为 true，当对方回合结束后做相反的操作。

图 4.8 敌人的贴图

战斗界面的攻击动画是我比较头疼的问题，当时动画的实现我是做出来了，但是动画老是出现闪烁的问题，看起来很难受。我先解释一下攻击动画的实现。

图 4..9 战斗模式的攻击选项内容

图 4.10 玩家宠物的贴图

攻击动画由于时间原因做得比较简单，只是当宠物被攻击时会出现抖动，以此来表示被攻击了。抖动的做法和走路的动画有几分相似，以静态贴图的坐标为中心，左右来回移动，实现的方式就是通过偏移量来改变 x 轴的坐标值，就像下面的这一行代

码

```
g.drawImage(宠物贴图,
                当前 x 坐标值 + x 偏移值 - 移动范围的宽度/2，当前 y 坐标值，
                贴图的宽，贴图的高)
```

双缓冲解决界面刷新时闪烁问题：

在前面我提到过刷新界面闪烁的问题，之前不清楚 repaint（）的工作原理，后来明白，repaint 是将前面的界面直接覆盖，而且在刷新新界面的时候是逐行刷新，这样会有上一张界面的贴图残留，所以出现了界面刷新时闪烁的问题。从网上搜索解决方案，他们提到了双缓冲，原理就是将即将要显示的图片完整的先缓存起来，然后清空面板，最后将整个图片画上去。

代码实现如下：

```
public void update(Graphics g) { // 双缓冲，解决动画闪烁问题
    if (offScreenImage == null)
        offScreenImage = this.createImage(640, 320); // 新建一个图像缓存空间,这里图像大小为 640*320
        Graphics gImage = offScreenImage.getGraphics(); // 把它的画笔拿过来,给 gImage 保存着
        paint(gImage); // 将要画的东西画到图像缓存空间去
        g.drawImage(offScreenImage, 0, 0, null); // 然后一次性显示出来
}
```

4.3 背包系统

界面为一整张贴图，背包的实现主要是坐标的计算。每个道具名的位置之间的间隔是固定的，道具名和道具数量的间隔也是固定的。

图 4.11 背包界面效果

背包系统的道具数量是 player 玩家类中的属性。

背包中的指针指向某一道具时按下 z 键（即确认键）后将消耗掉一个道具，并获得对应的值加成。奇异果可以使自己的宠物升一级，精神草可以使自己的宠物生命值完全恢复。

4.4 商人系统

商人的对话框中的内容的位置是根据坐标来确定的。

图 4.12 对话框贴图

指针根据内容作偏移调整即可确定。

图 4.13 商人对话效果

当指针指向特定的道具时按下 z 键，表示确认购买，对应的道具将会加一，数量会在背包中显示。

5. 测试

测试游戏功能：

分别测试游戏的探索界面、战斗界面和背包界面

图 5. 1 开始游戏的界面

图 5. 2 与敌人对话

图 5. 3 去医生那里治疗宠物

图 5. 4 在商人那买道具

图 5. 5 战斗模式的界面

图 5. 6 攻击模式的选项

图 5. 7 治疗选项

图 5. 8 防御选项

图 5. 9 自爆选项

图 5. 10 自爆后的效果

图 5. 11 背包界面

图 5. 12 使用背包中的道具

6. 总结

由于时间只有一个星期，而且我还是第一次接触图形化界面的编程（android 开发例外），特别是 java swing 我也是刚刚接触，刚开始决定做这个游戏时，心里是没底的。能不能完成都是一个未知数，更别说实现了。

好在我在对的时间发现了对的方法。刚开始没有接触过 java swing，不知道怎样搭建出一个图形界面，怎么去响应事件，于是我从网上搜索 java swing 开发游戏的流程，最终我看到了有人用 java swing 编写俄罗斯方块游戏的视频，于是我按照他编写游戏的思路开始了编写 RPG 口袋妖怪游戏的漫漫长路。首先一个游戏得有框架，要不然一上去就写代码，到最后的代码维护将会变得非常麻烦，因为根本找不到，自己实现某一功能的代码具体在哪里。于是，我先做了游戏的简单策划，首先确定这个游戏中会有哪些角色，这些角色具有哪些功能和属性，然后分别为每个角色创建对应的类来实现这个角色的功能和属性。角色确定好后，就要确定功能。功能主要包括行走功能，对话功能，战斗功能，交易功能，背包功能，治疗功能以及怪物遇见方式的功能等等，刚开始，这些功能的具体实现都在一个主类中，也就是工程中的 FrameView.java 文件中。由于一开始没有太多考虑到代码结构的问题，只是一心想着先把功能实现，因为当初计划的游戏功能还是挺多的，时间这么短未必能完成，所以就将代码结构问题先抛在脑后了。幸运的是我们已经将最核心的功能都实现了，而且一些辅助的小功能也实现了不少。刚开始将功能实现时，没有动画的过渡，操作感觉很生硬，游戏性特别差。于是我从网上找了很多动画的素材，制作动画贴图。由于自己没有系统的学过美工，也没有那么多时间去做，所以贴图质量和动画效果没有想象中的那么好。

从这次的游戏开发的过程可以明显的感受到结构的重要性。当代码量大到一定程度，回头再去找对应功能的实现方法就很难了，而且良好的编程习惯，也很重要。必要的注释不可缺少，要不然自己写的代码，到后来自己都看不懂了。到后期由于时间比较紧迫，所以就没有太多注意代码结构和注释的问题，不过后续我还会接着将代码重构一下，让整个结构更加的清晰明了，代码逻辑更容易理解，这是软件工程师必备的职业素养。

参考文献

[1]. Gary Cornell、Cay S. Horstmann 著，周立新，陈波，叶乃文译.Java 核心技术 卷 I[M].北京：机械工业出版社，2013

[2]. 何青著.Java 游戏程序设计教程（第二版）[M].北京：人民邮电出版社，2014

[3]. [美]Timothy Wright 著.Java 2D 游戏编程入门（游戏设计与开发）[M]. 哈尔滨：哈尔滨出版社，2015

[4]. 赵满来著.可视化 Java GUI 程序设计教程：基于 swing 组件库及 NetBeans IDE[M].北京：清华出版社，2015

[5]. [美]马克.艾伦.维斯著.数据结构与算法分析：Java 语言描述（原书第 3 版）[M].北京：机械工业出版社，2016

附录

```
/*
 * 双缓冲解决图片刷新时闪烁的问题
 */
@Override
	public void update(Graphics g) { // 双缓冲，解决动画闪烁问题
		// TODO Auto-generated method stub
		if (offScreenImage == null)
			offScreenImage = this.createImage(640, 320); // 新建一个图像缓存空间,这里图像大小为640*320
		Graphics gImage = offScreenImage.getGraphics(); // 把它的画笔拿过来,给gImage保存着
		paint(gImage); // 将要画的东西画到图像缓存空间去
		g.drawImage(offScreenImage, 0, 0, null); // 然后一次性显示出来
	}



/*
 * 图片的绘制
 */
   @Override
	protected void paintComponent(Graphics g) {
		// TODO Auto-generated method stub

		if (!MyUtil.ACTTACK_MODE) {
			discoveryMode(g);
		} else {
			BattleMode(g);
		}

		if (MyUtil.isBagMode) {
			drawBagItem(g);
		}

	}



/*
 * 键盘监听事件处理
 */
```

```
@Override
        public void keyPressed(KeyEvent e) {
            // TODO Auto-generated method stub

            switch (e.getKeyCode()) {
            case KeyEvent.VK_UP:
                if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)
                    ;
                else
                    up();
                break;
            case KeyEvent.VK_DOWN:
                if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)
                    ;
                else
                    down();
                break;
            case KeyEvent.VK_LEFT:
                if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)
                    ;
                else
                    left();
                break;
            case KeyEvent.VK_RIGHT:
                if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)
                    ;
                else
                    right();
                break;
            case KeyEvent.VK_Z:
                if (MyUtil.isBagMode) {
                    bagModeHandle();
                } else if (MyUtil.ACTTACK_MODE) {
                    battleModeHandle();
                } else if (MyUtil.isBussinessMode) {
                    bussinessModeHandle();
                } else if (MyUtil.isDoctorMode) {
                    MyUtil.isDoctorMode = false;
                } else if (MyUtil.isEnemyMode) {
                    MyUtil.isEnemyMode = false;
                } else {
                    discoveryModeHandle();
                }
                break;
            case KeyEvent.VK_X:
```

```
                if (MyUtil.ACTTACK_MODE && MyUtil.isBagMode)
                    MyUtil.isBagMode = false;
                if (!MyUtil.ACTTACK_MODE && MyUtil.isBussinessMode) {
                    MyUtil.isBussinessMode = false;
                    validate();
                    update(getGraphics());
                }
                if (MyUtil.ACTTACK_MODE && !petPlayer.myTurn)
                    ;
                else {
                    battleMode.FistLoad = true;
                    battleMode.isFighting = false;
                }
                break;
            case KeyEvent.VK_A:
                if (MyUtil.isBagMode) { // 切换出背包界面
                    MyUtil.isBagMode = false;
                    validate();
                    update(getGraphics());
                } else if (!MyUtil.ACTTACK_MODE) {
                    MyUtil.isBagMode = true;
                    validate();
                    update(getGraphics());
                }
                break;
            default:
                break;
            }
        }




/*
 * 随机遇怪的时钟处理
 */

        class MyTask extends TimerTask {
            @Override
            public void run() {
                // TODO Auto-generated method stub
                if (!MyUtil.ACTTACK_MODE && discoveryMode.isMoving) {
                    MyUtil.ACTTACK_MODE = true;
```

```
                    // System.out.println("ready");
                    repaint();
                }
            }
        }



/*
 * 脉冲信号实现，定期刷新界面
 */
        public class PulseSignalTask extends TimerTask {
            @Override
            public void run() {
                // TODO Auto-generated method stub
                doctor.index = (int) (Math.random() * 1000 % 2);
                bussinessman.index = (int) (Math.random() * 1000 % 3);
                enemy.index = (int) (Math.random() * 1000 % 3);
                if (!MyUtil.ACTTACK_MODE) {
                    petEnemy.level = petPlayer.level + (int) ((Math.random()
* 1000) % 10) - 5;
                    petEnemy.enemyPetImageIndex = (int) ((Math.random() *
1000) % 8);
                    petEnemy.refreshParameter();
                    petPlayer.refreshParameter();
                    battleMode.CloseBattleMode_Enable = MyUtil.ACTTACK_MODE;
                }
                validate();
                update(getGraphics());
            }
        }
```