中国科学 : 信息科学

SCIENTIA SINICA Informationis

Java设计报告

# Developing a Java Game from Scratch

尹凯[1,2]

1. 南京大学, 南京210000

E-mail: 1249011730@qq.com

收稿日期: 2021-12-28; 接受日期: 2021-12-29

国家自然科学基金(批准号: 0000000, 0000000, 00000000)

**摘要** 本文是南京大学计算机科学与技术系2021 秋季课程《Java 高级程序设计》的大作业总结报告，内容主要包括游戏的开发目标、设计理念、技术问题、工程问题、课程感言等部分。本次设计使用Java语言和面向对象编程范式，实现了一个包含游戏存档、多人联机对战等功能的图形化Ruguelike游戏

**关键词** Java, 面向对象, 存档, 联网对战, 图形化

## 1 开发目标

本次设计的开发目标是实现一个二维平面内的Ruguelike游戏。单人模式的地图是随机生成的，具有保存游戏进度的功能(单人模式下按P键)，目标是消灭所有的怪物，可通过接触(即移动，分别对应键盘上的上，下，左，右键)或发射子弹(D按键)的方式对怪物进行攻击，游戏开始时会在地图中随机刷新一把宝剑，捡到它可以使子弹类型变为“飞刀”，拥有更大的杀伤力；地图中除边界以外所有的砖块都可以被Player的攻击破坏掉，打掉砖块可能会出现一些类型的Bonus，获得它们会有加血量和加攻击力等效果。

多人模式则需要先启动Server(**启动后还需要按一下任意键**)再启动Client，**且必须等所有参加游戏的Client接入后才能开始操作**.目标是活到最后，最多支持3名玩家共同游戏，宝剑的刷新位置将总是在地图中央，每操作到一定次数地图就会缩小，在边界之外的玩家将会立刻被淘汰。

以下是游戏截图

**引用格式:** 作者1, 作者2, 作者3, 等. 中国科学LaTeX模版. 中国科学: 信息科学, 在审文章

Xing M, Xing M M, Xing M, et al. Title for citation (in Chinese). Sci Sin Inform, for review

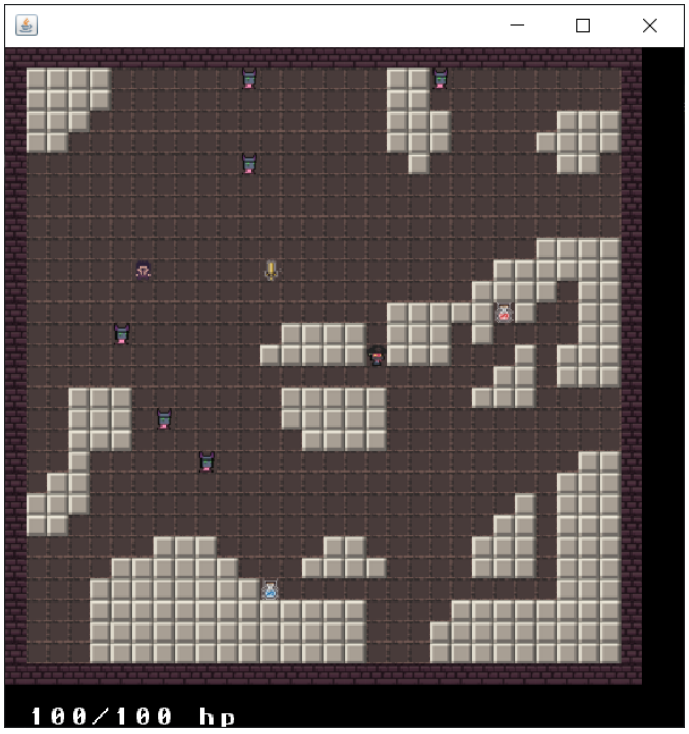


**图 1** 游戏截图

## 2 设计理念

游戏的设计灵感来自几个不同的游戏，Ruguelike的部分来自“元气骑士”，打砖块的部分来自“坦克大战”，地图变化的部分来自“BattleGround”

整个游戏使用面向对象编程范式，尽可能地通过对象之间发送消息的方式来模拟真实的交互场景。游戏的基本框架复用了jw04的Ruguelike例子的框架，文件结构如图2：

UI基于ASCIIPanel，后面通过在源图片上贴图的方式实现图形化

screen文件夹下的文件用来显示画面，并承担部分有关运算任务，以及各个类别的初始化。

network文件夹下的类负责网络通信，包括Server，Client和一个用来监听Client的读事件的ClientListener类

GameData文件夹下的类负责游戏进度的保存。

world文件夹下的类组成游戏的各种实体，其中的部分类关系如图3：

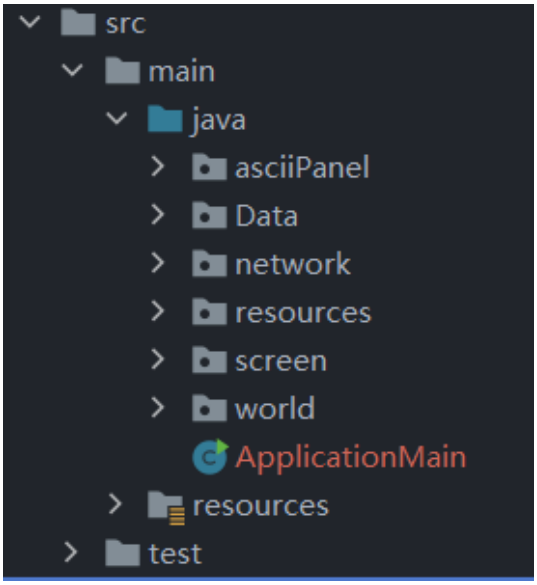


**图 2** 结构

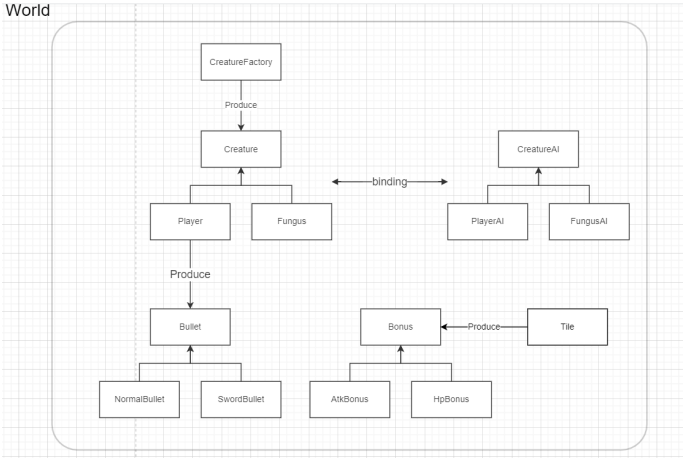


**图 3** World

## 3 工程问题

### 3.1 图形化

这次游戏我使用了基于AsciiPanel的UI，ASCIIPanel的UI的原理是对素材图片进行分割和存储，所以使用的时候只需要通过下标访问；由此可以看出，我们想更换UI只需要用自己的图片替换原图中对应的部分。首先确定地砖的颜色，因为这会是后面所有图片的背景色

图 4　Ground

接下来陆续收集到了一些实体的UI，开始时我直接将其贴在了素材图片上，结果发现现实的时候会带出它本身的背景色，所以我编写了一个脚本将素材的背景颜色换成Ground的颜色，效果如下： 最后，还要修改ASCIIPanel的存储格式，确保图片显示出的时候是本来的颜色，经过查找，

图 5　原图片

图 6　修改后

只需修改如下语句（被注释掉的是原语句）：

```
BufferedImage img = glyphs[chars[x][y]];
//BufferedImage img = op.filter(glyphs[chars[x][y]], null);
```

图 7　Code

### 3.2　网络通信

网络通信是整个工程的难点，因为以前从来没有在项目中使用过这方面的内容，而且出现bug不容易调试。本次作业我使用的是基于Java NIO的网络通信，这样可以避免由于线程阻塞所导致的通信不同步问题，同时我使用了一个selector来管理server和client之间一对多的通信。示意图如图4：

#### 3.2.1　计算中心的选择

在通信过程中由什么来完成计算，是需要重点考虑的问题。我主要想到了两种方案：

1.Client发消息给Server，由Server进行计算再将计算后的状态转发给各Client。这样做的好处是各客户端之间可以严格保持同步，这也是现在较为流行的实现方式但有一个明显的缺点，就是每

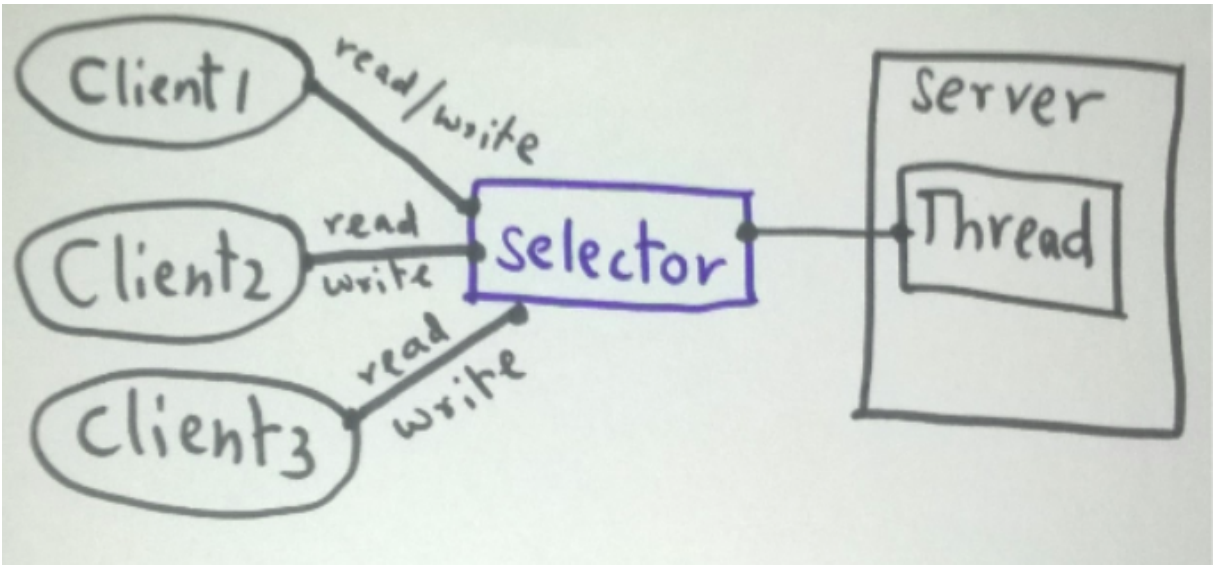


**图 8** NIO通信结构

次传输的数据量巨大，因为要包含地图、实体等诸多信息。

2.每个Client单独进行计算，只将操作发送给Server，Server进行简单处理即进行转发。由于有TCP协议的保障，只要**在所有参与游戏的客户端接入之后再开始操作**(这也是操作的要求)，也可以保证游戏是同步的，而且每次只需要传一个操作的代号，数据量很小。

最后我经过考虑还是牺牲了严格的同步而是选择了第二种方案，这样就使得按照规范操作变得非常重要。

### 3.3 屏幕刷新

屏幕刷新的方案也有两个选择

1.保留jw04的做法，类似于事件驱动，在有键盘输入的时候进行屏幕的刷新，这种方案显然不适合本次作业中多线程、高计算量的环境，于是我选择了另一种方式

2.单独开一个线程控制屏幕的刷新率，通过线程的Sleep函数决定游戏的帧率，在图像不是很复杂的时候表现很理想，也是本次作业我使用的方案（图9）

```
public static void main(String[] args) {
    ApplicationMain app = new ApplicationMain();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
    new Thread(() -> {
        while (true) {
            app.repaint();
            try {
                Thread.sleep( millis: 50);
            } catch (Exception e) {
            }
        }
    }).start();
}
```

**图 9** 屏幕刷新线程

# 4 工程问题

## 4.1 设计模式

### 4.1.1 工厂模式

对于应用比较多，创建逻辑较为复杂的Creature类和World类及其子类，实现了对应的CreatureFactory和World行创建，这样使得外层的函数调用时不必纠结于如何实现对应实体的初始化，只需要委托给对应的工厂进行“生产”

### 4.1.2 委托模式

对于实体中行为逻辑比较复杂的Creature类及其子类，实现了对应的CreatureAI类作为委托。这使得我们可以通过简单的聚合来完成一些可能需要通过继承来实现的操作

## 4.2 单元测试

本次实验为World中的实体和PlayScreen(主屏幕)的部分重点方法准备了单元测试来验证它们的正确性，而对于成熟的Asciipanel和方法较难测的network则没有单独去设置。即使将这些模块都算上，整个单元测试的覆盖度也能超过50

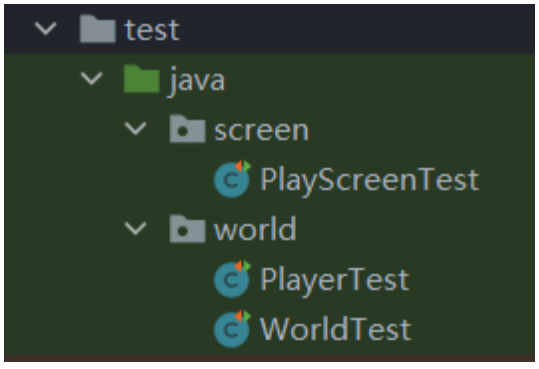


图 10 测试结构

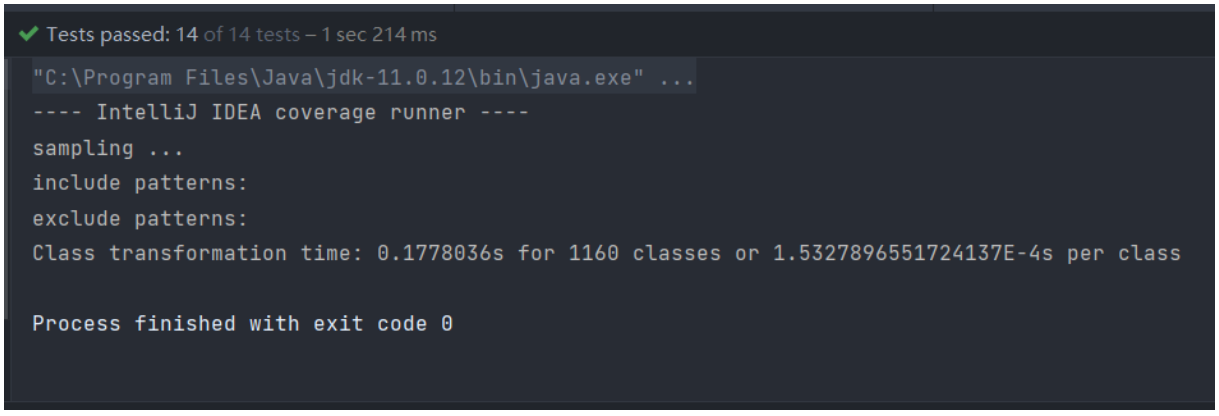


**图 11** 测试结果

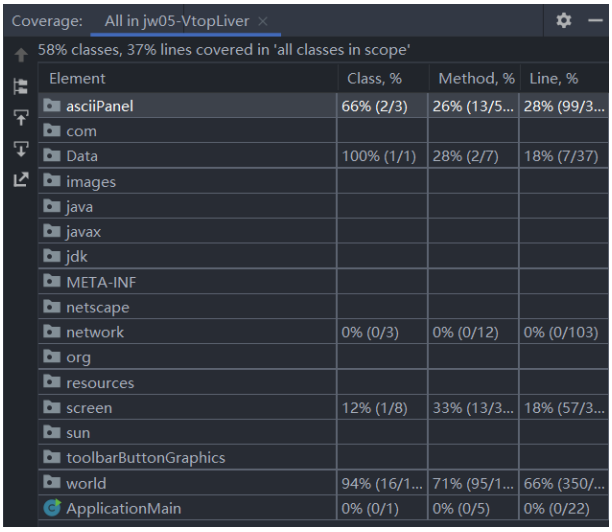


**图 12** 覆盖度

## 5 课程感言

在学习这门课之前，我以为它只是像高程一样简单的讲一下面向对象的相关知识，到学完之后才感觉收获颇丰。老曹讲课很有趣，课程的内容也很有意思。在上这门课之前，我根本想不到凭我也能完成集网络通信、多线程、图形化为一体的游戏，也想不到Java这门成熟的面向对象语言背后那些先进的机制，让我对Java这门语言，以及面向对象更加充满了好奇和求知欲。感谢老曹！感谢这门课!