中国科学: 信息科学

SCIENTIA SINICA Informationis

论文

从零开始的 Java 小游戏开发

钟煜尧[1]

1. 南京大学, 南京 210000

E-mail: 201220056@smail.nju.edu.cn

摘要 主要阐述了如何利用 Swing 框架写一个小游戏的设计思路，代码框架，测试用例，过程中遇到的问题和感想总结。

关键词

1 设计思路

1.1 地图内容：

地图主要由三种地形－－ Road,Wall,Empty 构成，这三种地形统一继承一个父类为 Being，只有 Road 可以在上面活动，其他几种类型是无法进入的。用一个 Boolean 型函数 BeOccupied（）来表示该类是否被占用，对于 Wall 和 Empty，是一定会返回 true 的，毕竟这两种地形是不可以让生物走上去的；而对于 Road 类来说，如果上面有生物则会返回 true，否则会返回 false。如果一个生物想走上某一个地形，就需要调用 CreatureMoveTo（）函数。如果是 Wall 和 Empty 类，则不会任何效果，如果是 Road 类型，则需要判断是否被占据，只有在不被占据的情况下，生物才可以移动到上面。而生物在移动到另一个地形类的时候，也需要放弃其目前所在的地形类，所以要调用 CreatureMoveOut（）函数。

对于每一种地形类都有一个 Tile 来包含这个类，Tile 中可以找到对应的地形，并且用两个 int 类型数据来表示 Tile 所在坐标。

对于整个世界会有一个 World 类，其中用一个 int 类型来表示其关卡等级，一个 Tile 的二维数组来表示各个坐标上的地形。World 的初始化需要读取对应的设定文件，为了便于修改和理解，这里用的是 txt 文件格式，命名格式为 level+.txt，在 setting 目录下，每一行以（x,y）的形式存储每一个 Road 地形对应的坐标，而与 Road 相邻且不是 Road 的坐标对应的地形就是 Wall，其他就是 Empty。

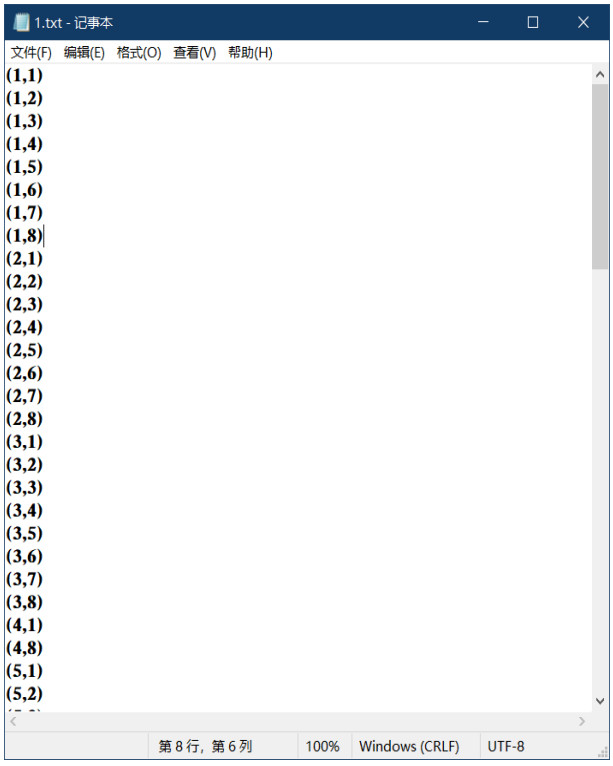

图 1　地图 txt 文件

1.2　人物和怪物：

创建一个 Creature 类，怪物就直接调用 Creature 类，而角色则用一个继承该类的 Role 表示，用整数类型来表示其 hp,maxhp（用来控制血条的长度），atk（攻击力），一个 Boolean 类型来表示其攻击模式（true 表示远程，false 表示近战），String 类型表示名称。这里依然是用 txt 文件来作为其设定文件，格式为 name+.txt，怪物的设定文件在 monster 文件夹下，每一行代表一个数值，而角色的设定文件在 role 文件夹下，所以其读取函数需要重写。

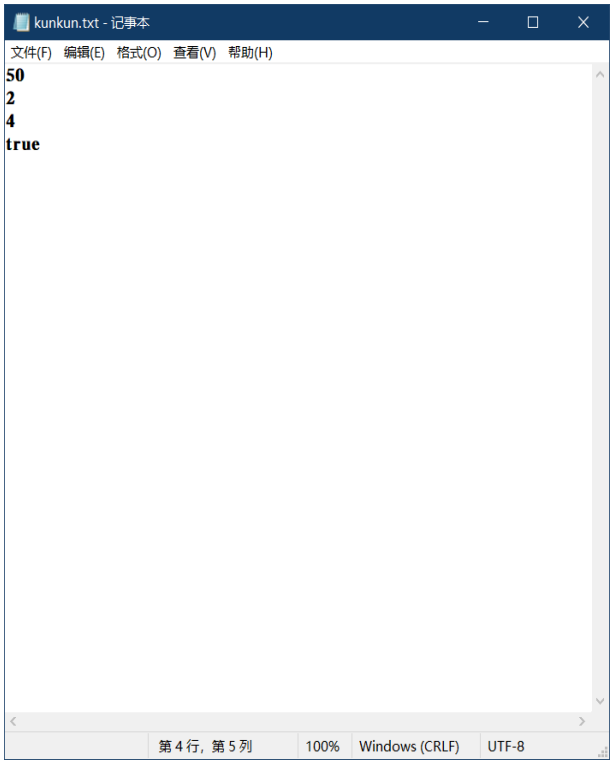

图 2　怪物 txt 文件

1.3　图形化界面：

图形化界面主要调用的是 javax.swing 框架，利用其中的 Jframe，JLabel 等进行编程。

1.3.1　初始界面：

初始界面为一个带有两个 JButton 的界面，背景为火娃喷火界面，Start 按键为开始新游戏，而 Resume 按键为从文件中读取存档并重新开始，用两个 actionListener 分别来监听两者的活动。

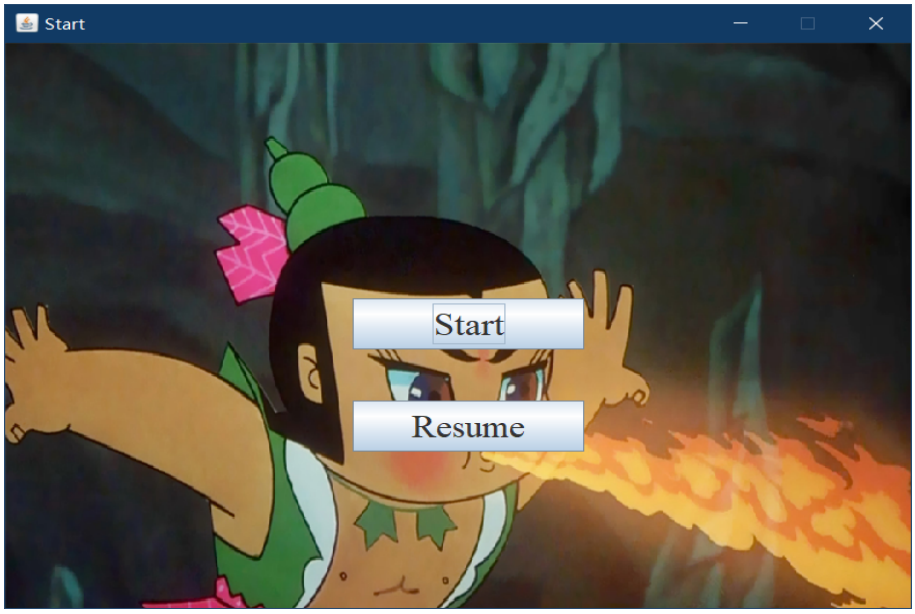

图 3　初始界面

1.3.2　选择界面：

选择界面中有三个 JComboBox，第一个选择游戏角色（目前只有三个，elf，soldier 和 fighter），第二个为游戏模式选择（single 表示单人模式，double 表示双人模式，需要联网），第三个为关卡选择（总共只有三关），点击下方 Play 按钮即可进入游戏，用一个 actionListener 监听 Play 按钮的动作。如果选择了单人模式，就直接进入游戏，如果选择了双人模式，则需要建立一个客户端去连接服务器，等待服务器响应后再进入游戏。

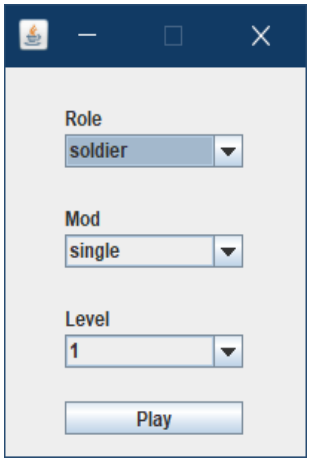

图 4　选择界面

1.3.3　怪物图标：

对于每一个 Monster，都有一个 CreatureLabel 类与之对应，利用 Creature 的名称、初始方向和初

始所在 Tile，宽度 width，高度 height 和所处世界 world 进行创建。首先用 Monster 的名称创建一个 Creature，用初始坐标来绑定人物的位置图像的位置 (x* width, y * height)，并设置其坐标 cur-x = Tile.x * width, cur-y = Tile.y* height，定义一个整数类型的 index，文件的图像路径为”img/”+ name + “/”+ direction + “-”+ index + “.png”，用一个 Boolean 变量 detected 来判断 Monster 是否探测到了敌人，初始设置为 false。

为了显示血条，创建一个继承自 JLabel 血条类 HpBar，并且将其宽度初始设置成和 CreatureLabel 一样。每当对应的 Creature 受到伤害的时候，就可以调用 Update 函数对血条进行更新，使其宽度变为 hp/max-hp*width，在进行移动时，也可以调用 Move 函数，将血条的位置移动到对应的坐标上。

为了生物可以移动，需要有 move 和 changeDirection 函数，先调用 changeDirection 函数来对生物移动的方向进行修改，如果生物前方的 tile 被占据了，则优先向两侧拐，若两侧的 tile 全部被占据，则会掉头，如果被围住了，则在原地等待，直到出现一个方向可以行走。而 move 函数，是改变生物所在坐标的，即向生物所朝方向移动 speed 数目的像素。在没有探测到敌人的情况下，如果生物走到了当前所在 tile 的边缘，则需要判断它所朝方向的下一个 tile 是否可以走上去，若是可以走上去，则放弃占据当前 tile，并占据前方 tile，否则留在原地等待；在探测到敌人之后，首先会用 BFS 算法来检测找到敌人的最短路径，并根据算法找到的路径来改变自己的接下来的运动方向，这时 Monster 就不用考虑前方的 tile 是否被占据了，否则在多个 Monster 探测到同一个敌人，并且在同一个方向时，前面的 Monster 就会挡住后面的 Monster，出现排队枪毙的情况。而为了让他们的运动自动化，需要用到 java.util.timer 和 java.util.timertask，创建一个 timer，初始设定间隔时间为 150ms，task 中的 run 函数就是 creatureLabel 对应的 move 函数。在 Monster 探测到敌人的时候，为了让其可以快速找到敌人，就将 timer 的间隔改成 80ms，让其可以快速移动。每调用一次，Monster 图片的 index 就要加 1 并且模 4，以实现动画的效果。

生物的攻击行为也要用到 timer 和 timertask，CreatureLabel 创建之初并不调用，而当生物遭受攻击的时候且之前 detected 值为 false 的时候调用 startAttack 函数，创建一个 timer 和，如果 creature 的攻击模式为 false，则 timerTask 中的 run 函数为 attack()，即攻击在范围内的敌人（由于没有在网上找到对应资源所以没做出对应的动画）；而攻击模式为 true 的生物则调用 launchBullet 函数，向上下左右四个方向分别发射子弹。

生物的受击行为也要分为两种，第一种为被近程攻击进攻，要判断进攻者和受害者之间的距离，如果在受击范围内就要扣血。第二种是被子弹袭击，如果子弹触碰到了 Monster 就要扣血，判断机制也很简单，只要子弹的四个角有一个处于 CreatureLabel 的方形范围内，即判定为命中。如果在受击之前 detected 为 false，就要将 detected 改为 true，move 的 timer 间隔改为 80ms，并调用 startAttack。这两个函数都得设置成 synchronized，避免出现数据冲突。在扣血之后要调用更新血条，如果 hp 为 0，则要将图标的 visible 参数设置为 false，停止攻击行为和移动行为的 Timer，并且向所在的游戏界面传递消息，减少其中的怪物的数量。

1.3.4 人物图标：

对于玩家的角色，也会有一个特殊的 Label —— RoleLabel，利用 label 的名称和初始位置即可创

建。RoleLabel 的 move 函数相比 CreatureLabel 的要简单很多，因为如果要写得和 Monster 一样要去占用 Tile 的话，可能会出现相当多的问题，比如它出现在四个 tile 上时（Creature 的运动是根据程序来动的，它们的边界一定会与 tile 的边界重叠，而 Role 的运动是被玩家操控的，这样就可能出现在四个 tile 上），那么到底算占据哪个 tile，要是运动中突然变向，又该如何判断是占据哪几个 tile，或者再双人模式里，它是否会挡住队友的路等等，所以就没有让 Role 去占据 Tile。当然由于 role 的运动相对复杂，所以对于是否会撞到边界或者墙的判断也要比 Monster 更加复杂，即先将原有坐标改成运动后的坐标，再判断 RoleLabel 的四个角是否出现在 wall 或者边界之内，如果进入其中了，就将坐标回退到原有坐标。运动的判断也是不太一样的，需要一个 Boolean 型的 movable，在 false 的情况下直接返回，在 true 的情况下再进行运动。

控制 Role 的方向用的是常规的 wasd 方向键盘，为了和其他游戏一样的运动控制效果，这里用了两个栈来实现按下和松效果。摁下其中一个键的时候就把对应的 char 值插入到 moveStack 栈顶，并且把 Role 的方向改成按键对应的方向。松开的时候就把对应的 char 插入 releaseStack 的栈顶，并且判断两个栈的栈顶是否为同一个 char 值，如果是则两个栈一同 pop 直达栈顶不相等或者空栈。在空栈的时候 role 的 movable 置为 false，不空栈的时候把人物的方向改成 moveStack 栈顶指令对应的方向。

Role 的攻击行为也要分为两种，和 monster 相同，但是攻击条件和 move 类似，即按下按钮时再开始攻击。这里同样设置了一个 Boolean 型变量 attackable，如果为 true 就可以进行攻击，否则直接返回。

Role 的受击行为就更加简单，只要判断是否在攻击范围内或者子弹是否命中即可，如果在范围内或者被子弹命中，就要扣血。如果受击后死亡，也要像游戏界面发送消息。

1.3.5 子弹图标：

为了让远程攻击者可以攻击，需要设置一个子弹类 BulletLabel，子弹的名称就用的是发射者的名称，攻击力就是发射者的攻击力，初始坐标为发射者自身中心位置。子弹图片文件路径为"img/bullet/" + this.name + "/" + this.direction + ".png"。子弹的运动依旧是靠 Timer 和 TimerTask 实现的，move 函数中要判断子弹是否碰到墙或者边界，判断方式和 RoleLabel 一样，如果未碰到边界或者墙，则继续判断是否命中。由怪物发射的子弹就判断是否命中率玩家，由玩家命中的就判断是否命中怪物。命中或者碰到边界/墙之后，就要停止 timer 并且 visible 参数设置为 false。

1.3.6 单机游戏界面：

单机游戏界面是 GameFrame 类，创建时需要 level 和所用角色的名称。利用一个 JLayerPane(即分层 Pane) 进行布局规划，将背景设置在最底层，即 DEFAULT-LAYER。将地形图片设置在 MODEL-LAYER，创建一个和 level 对应的 World，并读取其 map，将每一个坐标中对应的地形图片用 JLabel 加载出来，并且把图片放在对应的位置 (x* width, y * height)。而对于每个关卡中的怪物，也用 txt 文件写在 setting 文件夹中，命名格式为 monster-level-(level 的值).txt，每一行代表着一个怪物，第一串字符代表着怪物的名称，第二串字符代表着怪物初始时朝向的方向，第三串字符代表着怪物的初始坐标，用这些信息创建 CreatureLabel，并且每个 Label 创建一个 CreatureThread，开始运行这些

Thread。接着创建 RoleLabel，在 world 的 map 中找到的第一个未被占据的 Tile，把 RoleLabel 设置在该 Tile 上并开始游戏。

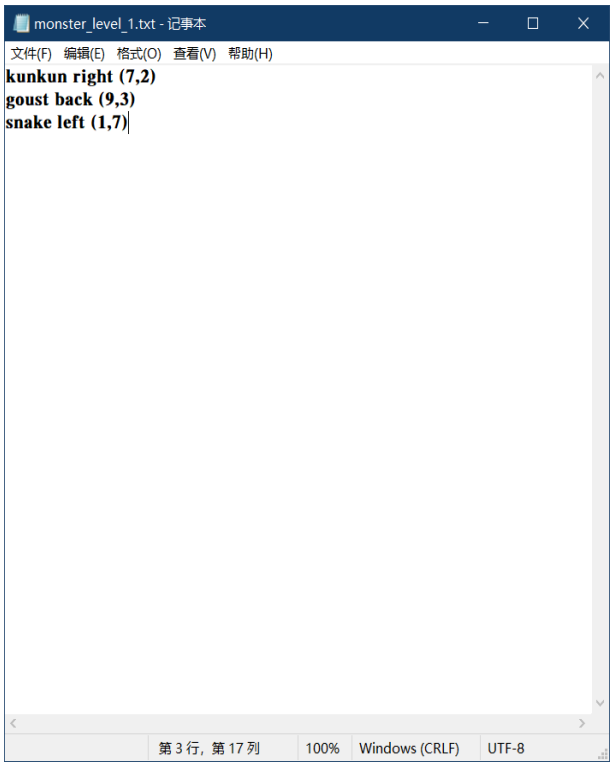

图 5　关卡设定文件

控制方式用的是 KeyListener，如果按下/松开的是 wasd 或者 j（即攻击键），则会把操作信息传递给 RoleLabel，让 RoleLabel 进行操作；如果按下的是 q 键，则会进行暂停/开始操作，把每个可移动 Label 的 moveable 变量改为 false/true，并且把 StopLabel（一个表示暂停的 JLabel）的 visible 设置为 true/false；如果按下的是 e 键，则可以在暂停的情况下把当前游戏存档，文件命名格式为 level-(level)-year-month-day-hour-minute-second.txt，会把界面中还可以活动的 Creature，Role，Bullet 记录下来，Creature 记录的是名称、方向、hp、所在 Tile 坐标，所在像素坐标和探测情况，Role 记录的是名称、方向、hp 和所在像素坐标，Bullet 记录的是子弹名称，方向，攻击力和所在像素坐标；如果按下的是 r 键，则游戏直接结束。

存档过后的游戏在初始界面点击 Resume 按钮就可以再次开始，在 ResumeFrame 会进入 save 文件夹中读取所有的节点信息，并把文件名转换成如下图的中的格式，在这个 ComboBox 中选择想要重启的游戏即可。这里会把文件路径作为 GameFrame 的构造参数，GameFrame 会根据文件中的内容进行构建，将游戏恢复到存档时的情况。

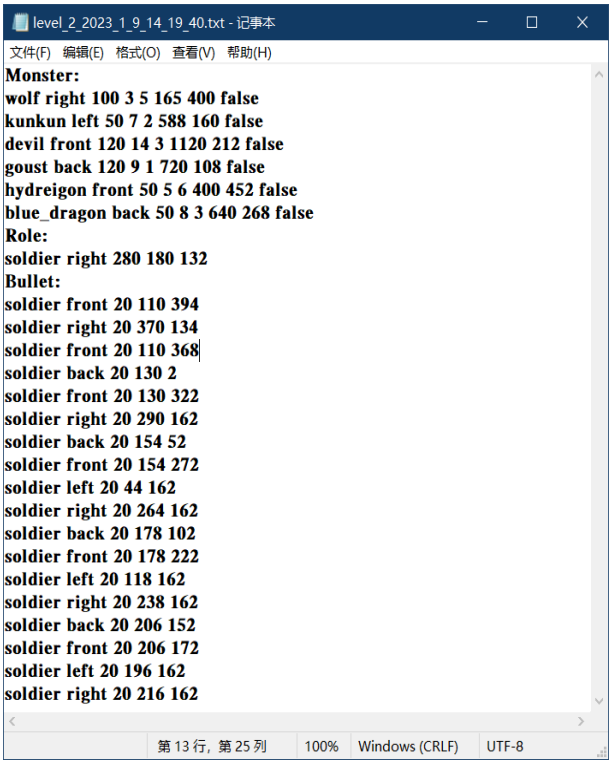

图 6　存档文件

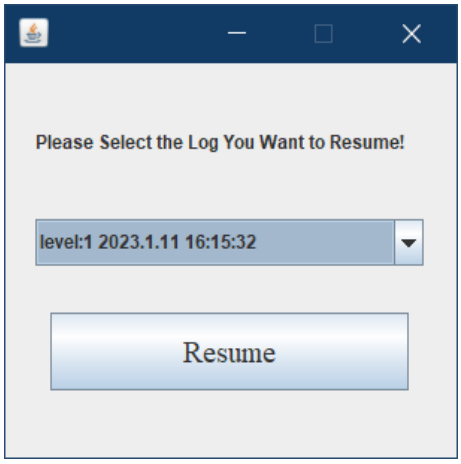

图 7　load 界面

在单人游戏中，游戏的获胜条件为击败所有的怪物，而失败条件为 Role 被击败。在 Role 或者 Monster 被击败的时候都会向 GameFrame 发送消息，GameFrame 会根据该消息判断是否失败或者胜利。游戏中有一个 winLabel 和一个 failLabel，初始状态下 visible 参数为 false，在胜利或者失败的时候，对应的 Label 的 visible 参数要设置为 true，并且把所有的 monster，role 和 bullet 暂停下来。

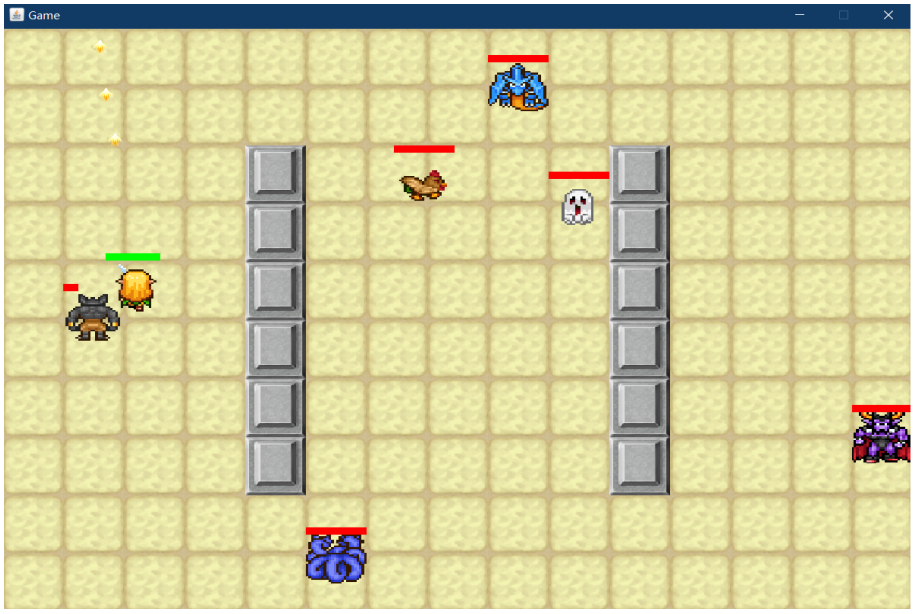

图 8　游戏界面

图 9　游戏结束界面

1.4　网络设计：

1.4.1　服务端：

首先要实现服务端，服务端占据电脑的 8080 端口（毕竟我的电脑也不是网页服务器），建立后等待客户端进行连接，由于只要供两台机器进行联机，所以只建立了两个 socket，通过 InputStream 和 BufferedReader 来读取客户端发送的消息。首先根据用户发来的角色和 level 创建继承自 DoubleGame-

Frame 的 ServerFrame（即双人游戏界面），若两用户选择的 level 信息不一致则已第一个用户的 level 信息为准。再用 OutputStream 和 BufferedWriter 把两个角色以及等级信息传送回客户端。这时建立四个线程，两个线程用来监听客户的键盘操作，并且把客户的操作信息传递给本地的 ServerFrame 使其执行，另外两个线程则是通过 Timer 和 TimerTask 定时向客户端发送 SeverFrame 上的信息，进行双边同步操作。SeverFrame 的信分为两种，一种是 Role 的信息，需要 Role 的索引，血量，朝向，位置，图片索引和攻击状态，还有就是 Monster 的信息，需要 Monster 的索引，血量，朝向，位置，图片索引和检测到敌人的状态，如果死亡，则在发送一条 hp = 0 的消息之后停止发送该实体的消息。Server 在游戏结束后会重新初始化自身状态，并重新开始连接新的客户端。

1.4.2 客户端：

客户端在用户选择进入 double 模式的时候会自动创建，向服务端发送用户选择的 level 和角色信息，并等待服务器的返回的信息，等到服务器返回信息之后再根据角色和等级信息创建 ClientFrame。然后会创建两个线程，一个用于将 ClientFrame 接受到的键盘命令传送给服务器，还有一个接收服务器传来的信息将其界面更新成和服务器一样。

1.4.3 联网游戏界面：

由于双人模式和单人模式的框架不太一样，所以为双人模式专门写了一个 Frame 为 DoubleGameFrame，服务器用的是 ServerFrame，客户端用的 ClientFrame。ServerFrame 要为所有的 Creature 创建线程，使之可以自动运行。而 ClientFrame 不需要这些线程，而且这些线程影响后面的同步过程。而在这些当中的 ServerFrame 和 ClientFrame 中的 CreatureLabel，RoleLabel 和 BulletLabel 也要重写一些方法，比如 move 方法，在双人游戏中，Monster 会优先追踪和攻击先对其造成伤害的 Role，在这个 Role 被击败后再去攻击剩余的 Role，而在单人模式中只有一个 Role，所以 Creature 并不需要在意自己的目标是谁，而双人模式有两个 Role，就要求我们要让 Monster 知道发起攻击的对象，因此要重写 CreatureLabel 的 move()，hitByBullet()，getAttacked() 方法，Role 的 LaunchBullet() 和 attack() 方法，Bullet 的 move() 方法，使之传递 Role 的索引参数。

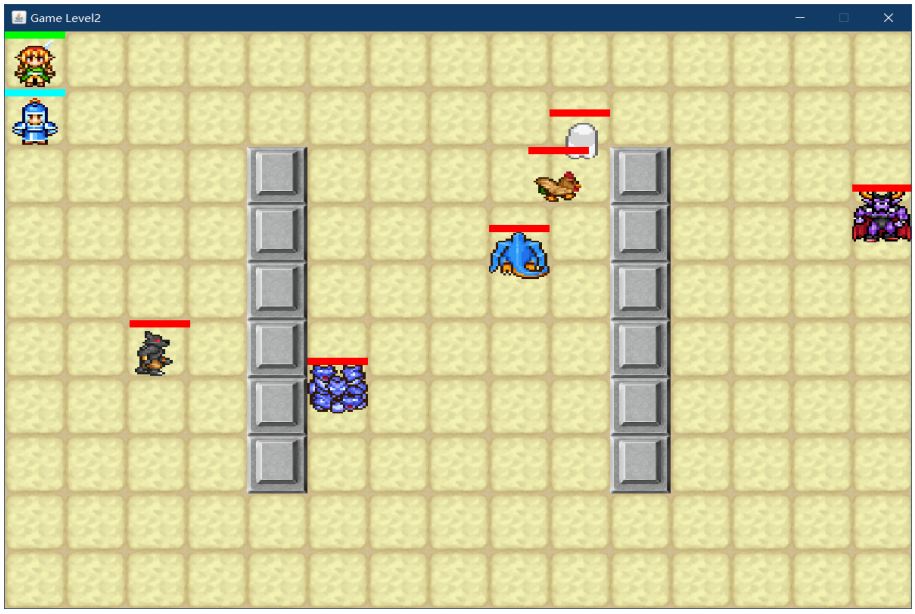

图 10　客户端界面

图 11　服务器界面

2　代码框架：

主要列举了几项比较重要的类的框架，一些比如 timertask，keylistener，thread 的就省略了。

图 12　Being 框架

图 13　Label 框架

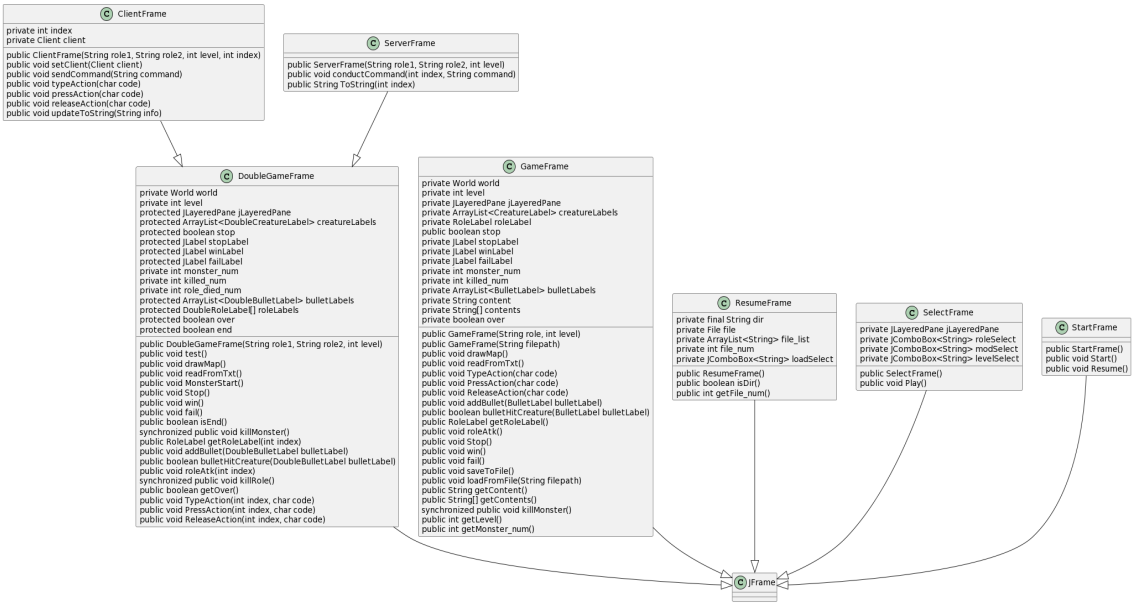

图 14　Frame 框架

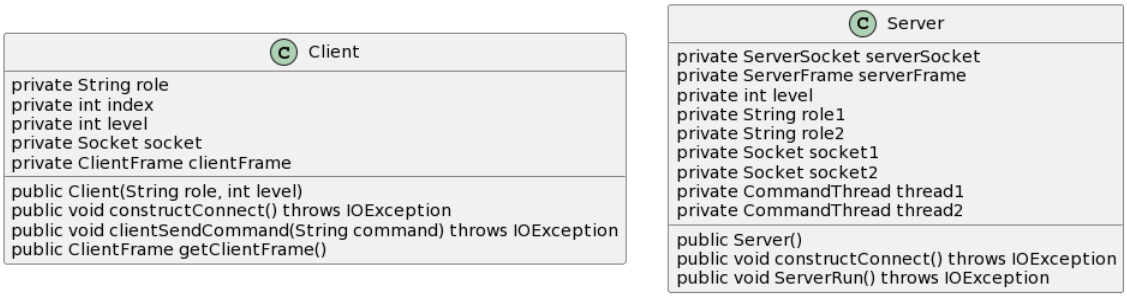

图 15　Network 框架

3　测试用例：

总共写了九个测试，分别测试不同的功能。

第一个测试是 Creature 的构造函数，判断 wolf 和 dragon 的数值是否符合 txt 的设定。

第二个测试是 Role 的构造函数，判断 elf 的数值是否符合 txt 的设定。

第三个是 GameFrame 的构造函数，判断 GameFrame 能否正常构造。

第四个是 ResumeFrame 的测试，判断其是否读取了 save 文件目录下的所有内容。

第五个是 StartFrame 的测试，测试其两个按钮能否正常运转。

第六个是测试 GameFrame 的监听系统，这里用的是模拟监听，只是调用了监听之后会调用的一些函数，并没有真去按键盘。

第七个是 GameFrame 的 Load 功能，判断 GameFrame 是否可以从存档的文件中恢复过来，在这之后也要进行模拟监听测试，观察是否能够正常运行指令。

第八个是 DoubleGameFrame 的测试功能，在其中加上一个测试的 KeyListener，操纵其中的一个 Role，判断是否可以正常运行（这里要自己去按键盘进行操作）。

第九个是测试网络链接功能，创建了为服务器和客户端分别创建了三个线程，并且让客户端发送消息（利用函数模拟）。

测试结果：

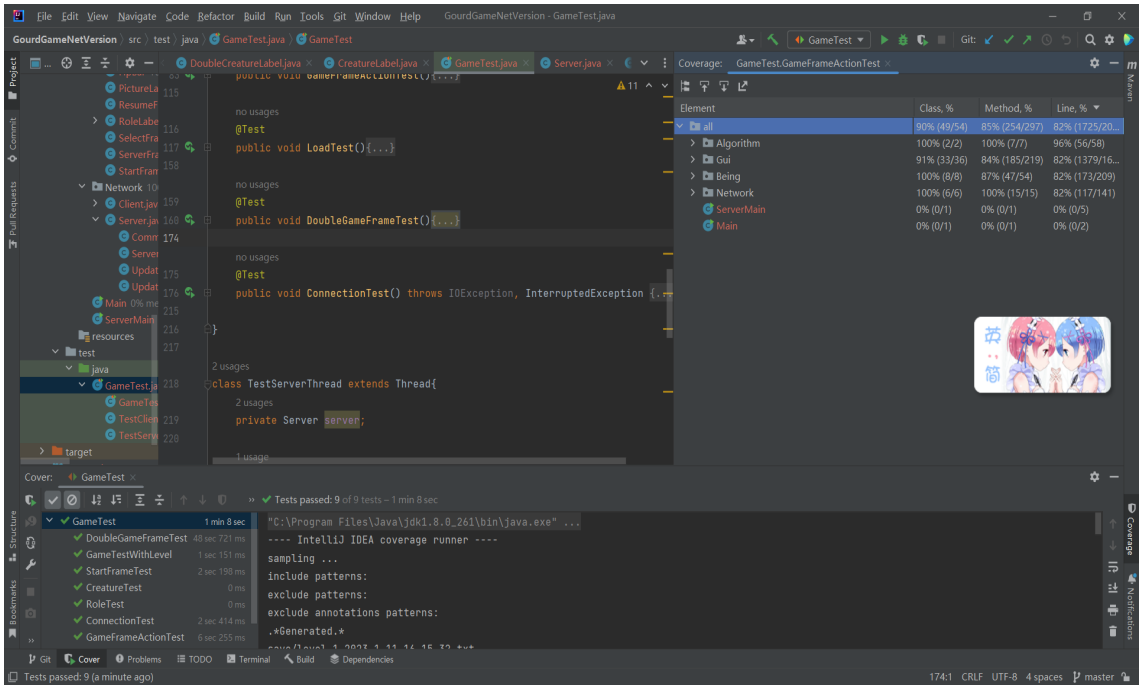

图 16　测试结果

4　过程中遇到的问题：

4.1　问题：

4.1.1　描述：

在进行字符串比对时，Java 中可以像 C++ 一样用 == 符号进行比较，但是在某些情况下，这个符号会失效，比如当我们一个从 txt 文件中读取的字符串和一个内置在代码中的字符进行比对时，== 号的比对结果就会出错，明明是两个相同的字符，却得到 true 的结果。

4.1.2　原因：

猜测可能是 txt 编码和代码文件不一样的原因。

4.1.3　解决方案：

用 equals() 函数代替就可以获得正确的比对结果。个人来说，我还是比较喜欢有运算符重载的，因为运算符重载可以让很多表示变得清晰明了，而且 C++ 的例子也说明了运算符重载其实并不会过分的影响性能（当然也有可能是 C 极其衍生语言语言特有的高性能致使的），当然 Java 作为一门能让

程序员可以快速上手的语言，为了保持其简单性和清晰性，自然要避免运算符重载这种复杂的操作，而且运算符的重载会导致其拥有过多的含义，污染代码，造成许多不必要的错误，也会使得语言的静态分析难度增加，不利于编译器进行处理。

4.2 问题：

4.2.1 描述：

在向界面中添加各种插件的时候，比如在发射子弹后添加子弹插件，经常会被之前添加的插件挡住而无法显示。

4.2.2 原因：

swing 框架中，向 JFrame 添加插件时，默认的是先添加的插件在上层，后添加的插件在下层。

4.2.3 解决方案：

在 csdn 论坛上查询的时候了解到，swing 框架中提供了一种 Pane，叫做 JLayerPane，它将界面分成五个层级，利用它就可以在添加插件的过程中，将插件插入到指定的层次里，就可以避免出现后面添加的插件被覆盖的情况。

4.3 问题：

4.3.1 描述：

在进行用户命令栈即 moveStack 的操作时，会出现连续插入了好几个相同命令的情况，这样即便送开所有的键盘，角色还会继续移动。

4.3.2 原因：

由于 KeyListener 在监听键盘操作的时候，只要摁下键盘不放开，就会不停的释放 Press 和 Type 信号，所以会一直在栈顶插入命令，而 release 信号只会放出一次，所以在退栈过程中无法使得命令栈为空。

4.3.3 解决方案：

在插入命令的时候，如果不是空栈，就要判断栈顶和插入的命令是否是同一条命令，如果不是再插入，就可以避免一次性插入多条相同指令。

4.4 问题：

4.4.1 描述：

在进行网络同步的时候，如果把所有子弹的同步信息加上就会出现卡顿现象。

4.4.2 原因：

由于在进行更新操作时，使用的是顺序操作，当只要更新人物和怪物的状态的时候，由于需要进行的操作比较少，所以可以在更新间隔的时间内完成。但是当加入子弹更新的时候，就需要进行几十甚至上百次操作，在短短的 50ms 内操作不过来，这样就会出现卡顿现象。

4.4.3 解决方案：

后面尝试了几种方式，第一种是建立一个精简版的子弹类，把其内部空间弄到最小，构造函数也简单化，但是依旧不奏效，毕竟没有真正从时间复杂度的数量级角度来减少子弹创建的过程，只是改变了其系数。第二种方式是传递人物和怪物的子弹发射状态，让人物或者怪物自动发射子弹。这种方式虽然可以大大减少服务端像客户端传送的信息量，避免了卡顿现象的出现，但是由于 Server 和 Client 之间存在信息延迟，而且发射子弹的时间和信息更新的时间并不一致，所以这种方式还是会导致信息不统一，即 ServerFrame 和 ClientFrame 中的子弹不统一。最后一种方式是只加入未被添加的子弹，在子弹被添加到客户端的 Frame 后，让子弹启动内部的 Timer，自己进行移动，这样就会减少很多不必要的信息更新。当然这种方式也不能做到和 Server 端完全一致，因为子弹的移动时机不一定和服务端相同，但是这是这些方法里面最为合理的一种，既能满足性能要求，又可以实现信息同步。

5 实验总结和感想：

这次实验过程非常的漫长而且艰苦，因为有很多新的事物要去学习，比如说 Java 的 GUI 编程，Socket 的使用方式等等，在 csdn 等论坛上的文章极其抽象，没有案例的情况下相当的晦涩难懂，后面才想到哔哩哔哩上可以看大佬讲解，但是由于大佬的视频很多很长，也不清楚自己需要用到的是哪一部分，所以仍旧需要花费很多时间来进行筛选和学习，就如同大海捞针。当然，初学的过程自然是很痛苦的，但是在学到知识后能把它运用出来，这就是一种甜蜜的收获。

在实验的过程中也会碰到各种各样的问题，上面列举的也只是几个比较重要而且映象深刻的问题，其他各种小问题也是层出不穷，不过最后还是解决了并且可以正常运行出来了。这个 debug 的过程也相当的有意思，特别是这种多线程中的 debug 过程，由于 debug 会影响其他线程的正常运行，而且对于某些操作的 debug 过程，也会影响其后续操作，比如 KeyListener，如果你的断点是在 PressAction 当中的，当你在运行到断点时放开摁下的键，那么后续就无法实现 Release 操作，如此种种导致很多时候不能采用打断点的方式来进行操作，只能运用控制台输出的方式来看是程序是否按照的正常的路径进行运转了。控制台输出的弊端当然很明显，就是你要写很多冗余代码，使得文件看起来相当混乱，但这也是没办法的办法。

这次的代码自然还是有些小瑕疵的，比如子弹的瞄准系统，因为图片的旋转似乎并不支持，所以只做了可以上下左右发射的子弹，而且为了避免极其复杂的计算系统，也没有做像元气骑士一样的近距离追踪。而选择界面也是内部就设定好的，没有写成可以根据设定文件来自我改变的那种。还有一个比较影响运行的问题就是电脑自带的输入法的问题，在操作界面必需手动把输入法改成英文模式才可以进行操作，否则会被输入法捷足先登。这个问题如果要自动化解决，可能需要通过系统来进行更改，而且得对输入法的程序足够了解才能做到自动化更改，目前来说我还是欠缺处理这个问题的能力。现在先把代码传输到 github，或许在进行更加深入地学习之后，我在翻阅自己曾经写过的代码时又看到了它，会利用更加优质的代码对其进行优化，使其可以真正意义上成为一款优秀的游戏。

Title

James Chong[1]

1. *Nanjing University, Nanjing* 210000*, China*
E-mail: 201220056@smail.nju.edu.cn

Abstract The article is mainly about design ideas, code framework, test cases, problems encountered and feelings of developing a game via Swing framework.

Keywords