## 实验报告

### 引言

项目原为java高级程序设计的课程作业，作业要求是利用java语言设计一个带有GUI的闯关游戏，能够进行地图选择和存档，并带有联网模式，可以进行联网合作或者对战。

在实验中，用到了java的swing框架来搭建GUI界面，net的socket实现网络交流，KeyListener实现交互。

### 设计思路（详细内容请参考report.pdf）

2.1地图设计

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。

2.2角色设计

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。

2.3界面设计

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。

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。

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。

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。

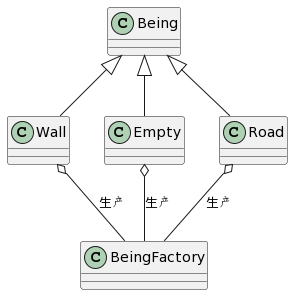
为了让远程攻击者可以攻击，需要设置一个子弹类 BulletLabel，子弹的名称就用的是发射者的名称， 攻击力就是发射者的攻击力，初始坐标为发射者自身中心位置。子弹图片文件路径为”img/bullet/” + this.name + ”/” + this.direction + ”.png”。子弹的运动依旧是靠 Timer 和 TimerTask 实现的，move 函数中要判断子弹是否碰到墙或者边界，判断方式和 RoleLabel 一样，如果未碰到边界或者墙，则继 续判断是否命中。由怪物发射的子弹就判断是否命中率玩家，由玩家命中的就判断是否命中怪物。命 中或者碰到边界/墙之后，就要停止 timer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

2.4 设计模式的插入

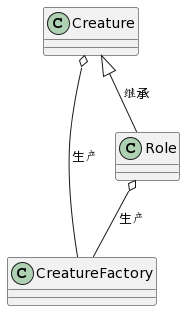
* 简单工厂模式

由于在World中要插入相当多的Being和Creature，以及要在图形化界面中加入相当多的CreatureLabel和BulletLabel，由于其构造方式和初始化要进行的操作比较复杂，所以要构建一些Factory类来生产这些产品。

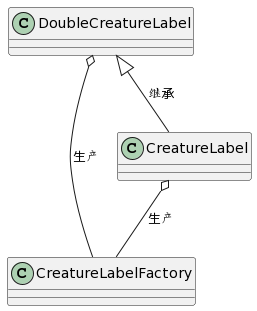
BeingFactory会根据传入的String生产对应名称的类，即“Wall”字符串生成Wall类，“Road”字符串生成Road类，“Empty”字符串生成Empty类。



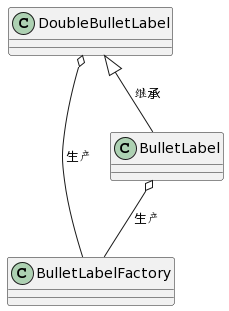
CreatureFactory同上，不过还要传入Creature的名称，通过名称去读取对应的配置文件。



CreatureLabelFactory调用的有两种情况，一种是在开始新游戏时，这时单人模式和多人模式都会调用，要根据不同的模式生成两种不同的CreatureLabel，一种是本就有的基础类型CreatureLabel（用于单人模式），一种是继承自CreatureLabel的DoubleCreatureLabel（用于双人模式）。另外一种要调用工厂的情况是从存档文件中读取存档时，因为这时候要将读取进来的字符串拆解并且换成对应的数据类型，操作相对较为复杂，如果写在读取存档工作中的函数中，会相当复杂，所以要写在工厂类中。

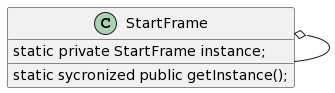


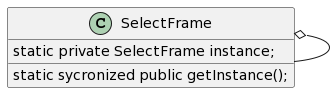
BulletLabel和CreatureLabel类型情况相似，都需要在这两种情况下进行生产。

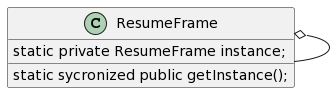


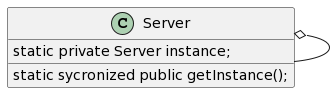
* 单例模式

为了节约空间，有些界面在一个进程中只能出现一个，例如开始界面，模式选择界面，存档读取界面（每次出现时更新一下内容即可），服务器界面。而游戏界面因为会重复产生，并且每次产生的过程都不一样，所以不适合用单例模式来进行操作。



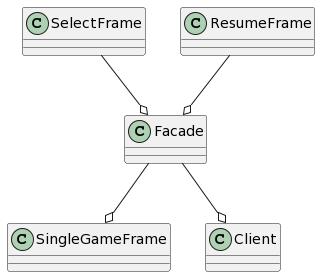






* 外观模式

因为在进行模式选择后，如果选择了不同的模式，它们调用的类（单人模式）是不一样，这种情况下，是需要使用façade模式来调用的。这里创建了一个GameFacade，根据用户的参数来决定是resume还是开始新游戏，是多人模式还单人模式。如果以后进行拓展，只需要再façade中添加即可。



* 观察者模式

其实在之前的设计中已经用上了，例如KeyListener,WindowsListener这些的，但是鉴于是用的现成的代码，所以并不算。

而之前的游戏逻辑基本是在GameFrame中写的，这就导致GameFrame和BulletLabel，CreatureLabel，RoleLabel耦合度过高，且拓展性不高，不利于创建新的游戏模式。可以利用Observer模式，来进行逻辑操作。

在BulletLabel位移之后，会向Observer发送消息，observer会根据子弹的发射发去寻找在子弹攻击范围内的敌人，并调用其受击函数。

在近战类型的RoleLabel和CreatureLabel发动攻击时，也会发送相应的消息，让observer去寻找攻击范围的敌人，并调用对应的受击函数。

在Role或者Creature死亡后，会向Observer发送对应的死亡信息，当死亡数量达到失败或者获胜条件时，就会进入获胜和失败界面。

1. **实现方法**

对于简单工厂模式，用的是静态方法实现，根据传入参数和调用工厂的方法的不同进行生产。

在进行单例模式的设计中，我使用了懒汉式双重检查的方式，因为在当前设计情况下，不会太多的线程来同时访问同一个单例，所以这种方式还是相对高效的，如果线程相对较多的情况下，则会考虑采用双重检查机制来进行设计，提高同步效率。

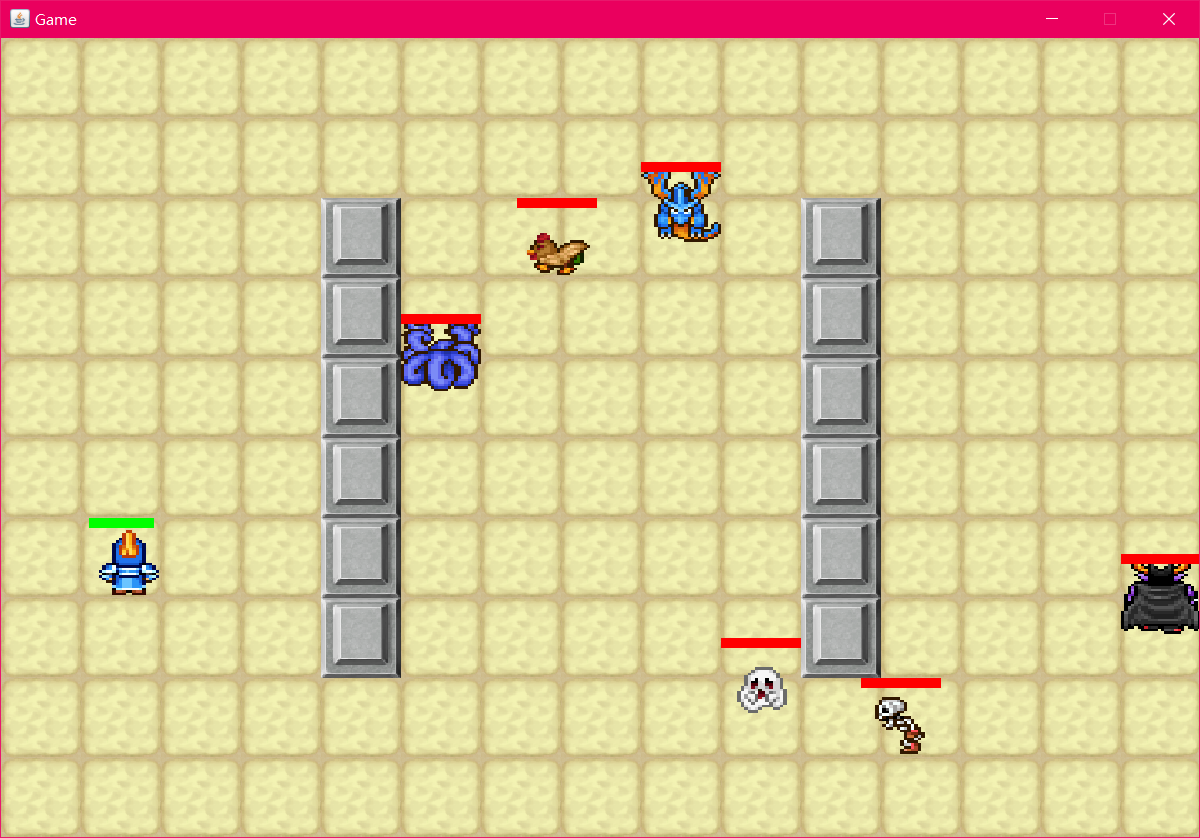
外观模式也是用静态方法和传入参数和调用方法的不同实现不同的调用。

对于观察者模式，在其中用ArrayList来存储多个需要被观察的同类对象，当相应对象发出信号后，会遍历可能被影响的对象，查找可能会被影响的对象。

1. **实现功能**

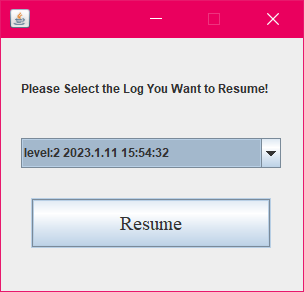
* 单人游戏

点击start按钮，选择single模式即可进入单人游戏模式，用wasd控制方向，j为攻击，q为暂停/开始，在暂停时点击e可以进行存档并退出。



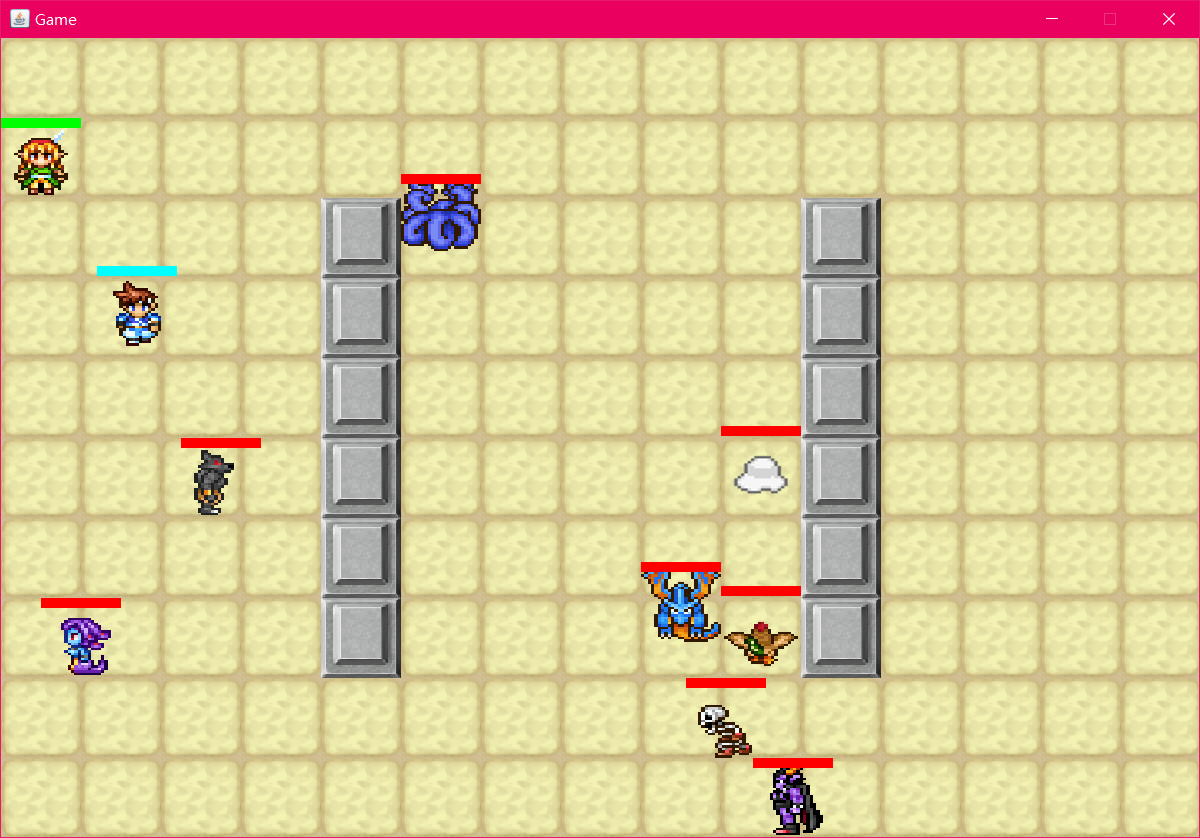
* 游戏读档

点击resume按钮，可以读取存档文件保存的游戏，继续进行。



* 双人合作

点击double模式，进入匹配，如果匹配到玩家就可以进入游戏，还是用wasd控制方向，j进行攻击。



1. **总结**

这次的修改程序还是比较劳累的，因为之前的代码完全是为了满足一个功能而写的代码，没有用到太多的抽象去实现，所以在进行设计模式改造的时候会非常复杂。这其实也说明了在设计代码的时候就必须要提前设计一些框架，来进行程序设计，否则在后期进行拓展的时候，就不得不面对很多困难，特别是要去修改原码，这是不符合开闭原则的。

像工厂模式和单例模式这样的类型创建模式，确实帮我解决了很多问题，比如之前点击按钮若出现卡顿就会出现多个界面这类的问题，通过单例模式就可以很好的解决。本来client界面也想做成单例模式，但是由于测试代码的问题就没有进行操作。而之前没有用工厂模式的时候，将那些类的创建代码夹杂在其他类的代码中，就会显得非常冗长，极其破坏其可读性，而工厂模式简化了代码，使得其更加整洁。

而façade模式，也帮助了我解决frame代码过于复杂的问题，减少了很多if语句并降低了Frame类之间的耦合度。Observer类也在一定程度降低了编码的复杂性，使得在创建新的游戏模式的时候不必在Frame中添加过多的代码。