中国科学：信息科学

SCIENTIA SINICA Informationis

报告

Developing a Java Game from Scratch

191220107 王皓冉

E-mail: 239955612@qq.com

1 开发目标

实现四人联机对战像素风游戏。

1.1 游戏规则

玩家身为爷爷不断收集宝箱，解锁新的具有特殊能力的葫芦娃。期间与怪物敌人进行战斗，并获取道具恢复体力和生命值，以及获取可以挖掘墙壁的能力。

当爷爷收集到了所有宝箱，即集齐了所有葫芦娃之后，锦旗将随机生成在地图中，最先获取锦旗的玩家将夺得胜利。

1.2 灵感来源

来自于《PUBG》等游戏，即玩家在游戏的开始一无所有，在游戏的过程中不断获取道具强化自己，并且零和游戏使得只有最快完成目标的玩家可以胜出。

2 设计理念

2.1 总体设计

程序的总体设计如图1所示。

在原有的 AsciiPanel 给出的 RogueLike 分支的基础上：

- 对于处理用户键盘输入和屏幕输出的 Screen 类没有进行太大的改动，除了简单的功能选择界面之外，OnlinePlayScreen 是独创性最高的一个 Screen 子类。
- 对于游戏核心实现的 Creature 类和 CreatureAI 类进行了自己的自定义：增加了 Player 类作为 Creature 的子类，用于描述玩家角色所特有的一些特点；增加了 PlayerAI 类及其众多派生类（老爷爷和葫芦娃的 AI），以及增加了 EnemyAI 类及其派生类（远程攻击怪物和自爆怪物）。
- 添加了 MyServer 类，用于实现网络通信对战。

以下将以 Screen 类及其派生类、Creature 类及其派生类、CreatureAI 类及其派生类、MyServer 类为角度，对具体的实现细节进行说明。

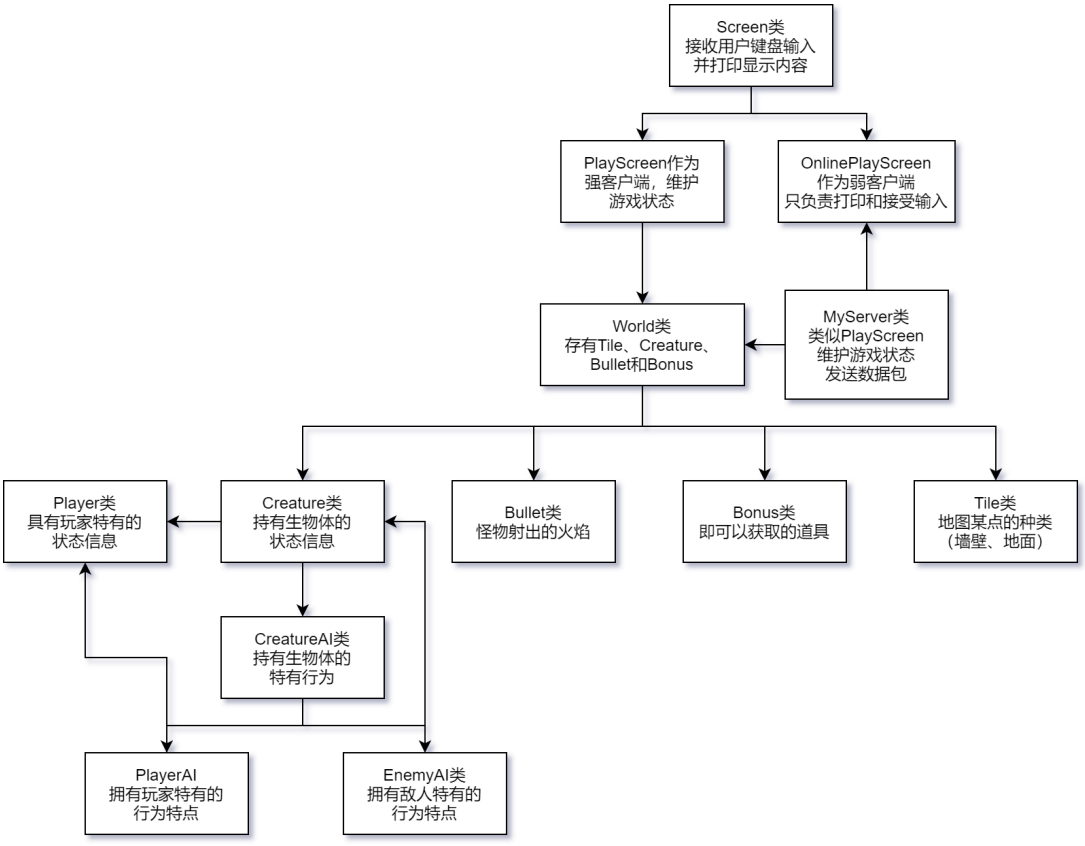

图 1 总体设计

2.2 Screen 类

Screen 及其派生类的继承关系和主要成员变量、成员方法如图2所示。

其中大多数子类都只有：

- 屏幕输出
- 处理简单的用户键盘输入

两个简单功能，因此不再赘述。这里只对有更加复杂实现的类进行详细说明。

2.2.1 RestartScreen

所有 RestartScreen 的派生类都有相同的特点：即按下回车键之后，都会回到 StartScreen，因此其中的 respondToUserInput 实现都是相同的。

2.2.2 SaveLoadScreen

用于本地的存档和读档，因此涉及到了文件的 IO 操作。

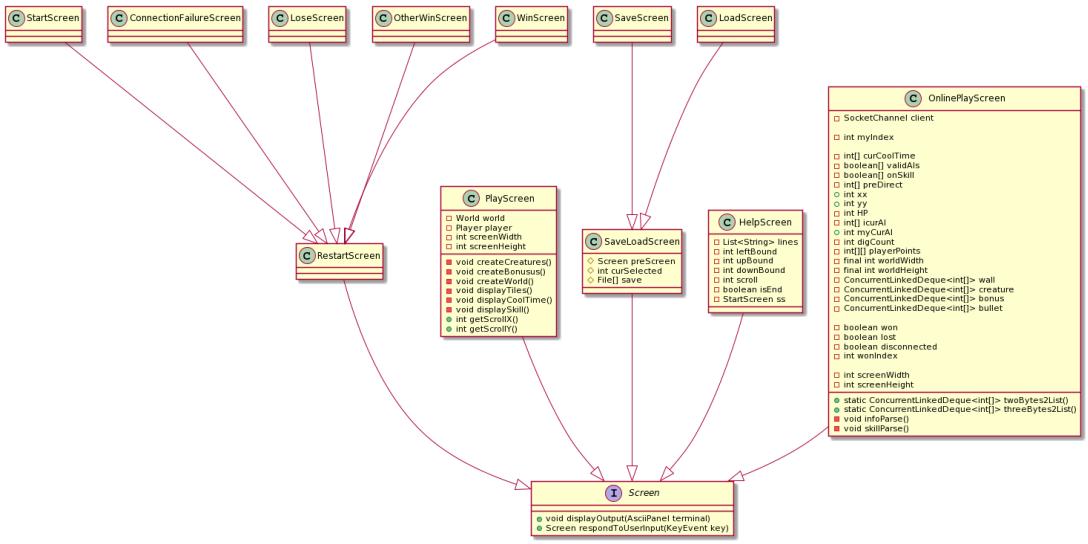

图 2 Screen 及其派生类

2.2.3 PlayScreen

与其他 Screen 类不同，PlayScreen 类是相对“胖”的。其中维护了 World 类和 Player 类的对象，并且定义了许多更加细分的屏幕打印函数。同时用户的输入也有了更多的细化，如移动位置和释放技能等等。

同时由于其维护了一场本地游戏所有的全部状态,因此本地存档就可以通过序列化一个 PlayScreen 实现。最终的存档大小大约在 800kB-900kB 左右。对于本地的磁盘 IO 来说开销很小，但是对于网络通信每秒要更新几十次的高速 IO 来说（而且又是使用 NIO Selector 的非阻塞方式），序列化这样一个庞大的类显然并不合适。

2.2.4 OnlinePlayScreen

专门由于联机游戏的 Screen 派生类。其可以处理的用户输入和 PlayScreen 大致相同（即控制自己玩家的各种操作），并且打印在屏幕的内容也与 PlayScreen 并无二致，但是内部的实现却有着非常大的区别。

OnlinePlayScreen 是一个更加“轻”的客户端，本身并不维护任何状态，所有的游戏信息都是通过网络 IO 得到的字节缓冲，内部私有的字节缓冲处理函数可以从中提取到游戏信息，最终将这些字节信息而不是序列化的对象通过网络进行传输。所有的用户输入都会变成放入相应的字节缓冲区，并发送给服务器；所有的屏幕输出都来自服务器，从字节缓冲池中取出。

2.3 Creature 及其派生类

Creature 及其派生类的继承关系和主要成员变量、成员方法如图3所示。

Creature 类包含了生物体（包括敌人和玩家）的所有状态信息, 如:

- 坐标值 x 和 y

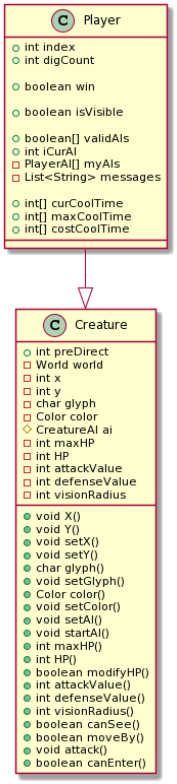

图 3 Creature 及其派生类

- 生命值 HP
- 攻击力 attackValue
- 防御力 defenseValue
- 可视范围 visionRadius
- 所持有的 CreatureAI 对象

而 Player 作为 Creature 的派生类，在 Creature 的基础上，添加了玩家所特有的状态信息，如：

- 多人游戏中的下标索引
- 冷却时间
- 是否可见（对应葫芦娃的隐身技能）
- 多个 PlayerAI 对象（葫芦娃形态的切换）

2.4 CreatureAI 及其派生类

CreatureAI 及其派生类的继承关系和主要成员变量、成员方法如图4所示。

CreatureAI 提供了生物体特有的行为特征方法，以及内部的非状态信息。同时 CreatureAI 继承了 Thread 类，即每一个 CreatureAI 最后都会在独立的线程生运行，实现了多个生物体行为的并发。

PlayerAI 作为 CreatureAI 的派生类，其中有着玩家特有的，但又是多个葫芦娃所共有的行为

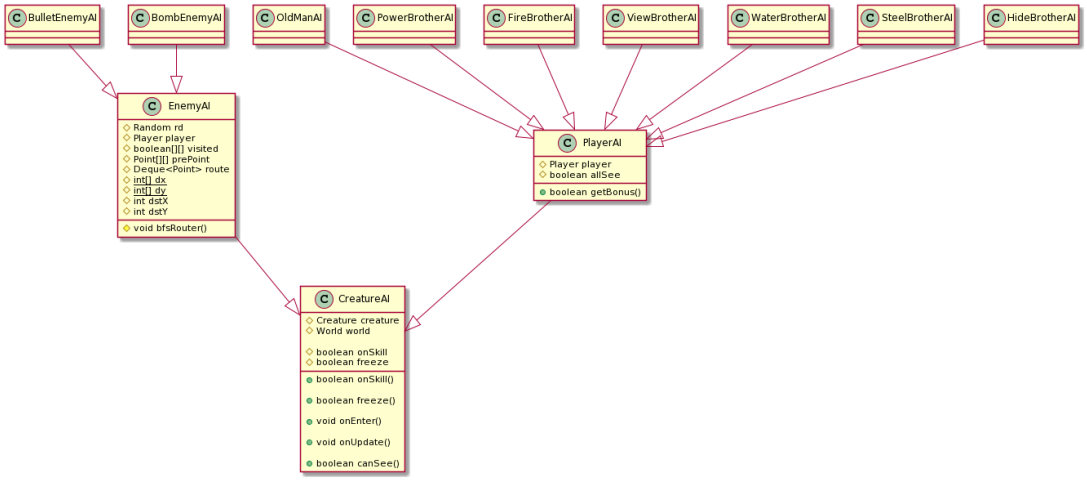

图 4 CreatureAI 及其派生类

特征，如：

- 获得道具的处理

其他 PlayerAI 的派生类，主要是实现了 Thread 的 run 接口，以体现不同葫芦娃的独特技能，如：

- 大娃近距离攻击
- 二娃解锁视野
- 三娃火焰攻击
- ……

EnemyAI 作为 CreatureAI 的派生类，其中有着敌人怪物所特有的行为特征，如：

- 通过给定的玩家对象，使用 BFS 算法得到最短路径

不同的 EnemyAI 的派生类，主要是实现了 Thread 的 run 接口，以体现不同敌人怪物的独特功能，如：

- 远距离攻击怪物可以射出火焰子弹
- 自爆怪物会快速靠近玩家并自爆产生伤害

2.5 MyServer 类

MyServer 类要完成以下的功能：

- 维护和更新游戏状态
- 处理用户输入
- 将游戏状态打包，发送给客户端

以下就将以这三个功能，分别对 MyServer 类进行说明。

2.5.1 维护游戏状态

MyServer 类维护游戏状态的方法，和 PlayScreen 类的大体逻辑相同，不同点在于：

- 要处理多个玩家的信息
- 生成怪物时，随机选择玩家作为怪物的攻击对象

2.5.2 处理用户输入

MyServer 类处理用户输入的方法，和 PlayScreen 类的 respondToUserInput 函数大致相同。不同之处在于:

- respondToUserInput 处理单用户的输入，MyServer 类需要加入额外的用户标签信息
- respondToUserInput 的输入来源是 KeyEvent 事件，而 MyServer 类的输入来源是来自网络 IO 的字节缓冲

2.5.3 通信包定义

通信包分为以下几个种类:

- 自己控制的玩家信息
- 地图（墙壁）位置信息
- 生物体（敌人和玩家）种类和位置信息
- 道具种类和位置信息
- 子弹方向和位置信息
- 玩家的技能信息

自己控制的玩家信息 通信包结构如图5所示。

该通信包的长度为定长 19 字节（包头不计算在内）

地图位置信息 通信包结构如图6所示。

该通信包的长度不定，因此加入了 4 字节的整型数，用于指示包的长度（二元组的个数，不是字节数，不包括包头）。

生物体种类和位置信息 通信包结构如图7所示。

该通信包的长度不定，因此加入了 4 字节的整型数，用于指示包的长度（三元组的个数，不是字节数，不包括包头）。

道具种类和位置信息 通信包结构如图8所示。

该通信包的长度不定，因此加入了 4 字节的整型数，用于指示包的长度（三元组的个数，不是字节数，不包括包头）。

子弹方向和位置信息 通信包结构如图9所示。

该通信包的长度不定，因此加入了 4 字节的整型数，用于指示包的长度（三元组的个数，不是字节数，不包括包头）。

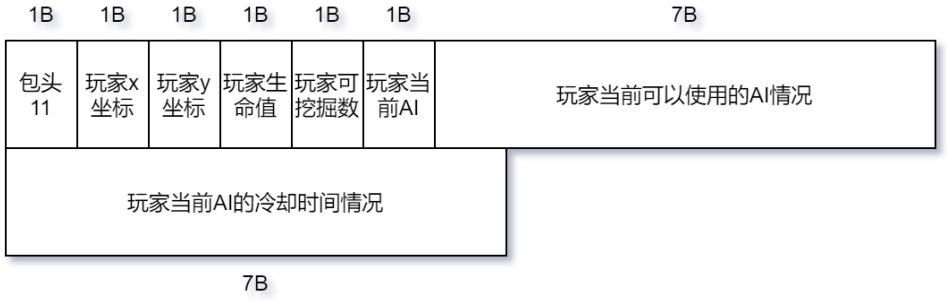

图 5 玩家信息通信包格式

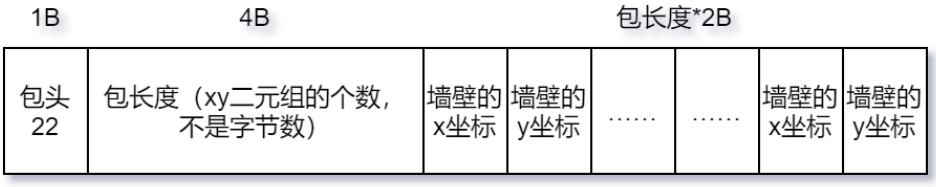

图 6 地图信息通信包格式

图 7 地图信息通信包格式

图 8 道具信息通信包格式

图 9 子弹信息通信包格式

玩家技能信息 通信包结构如图10所示。

该通信包的长度为定长 20 字节（包头不计算在内）.

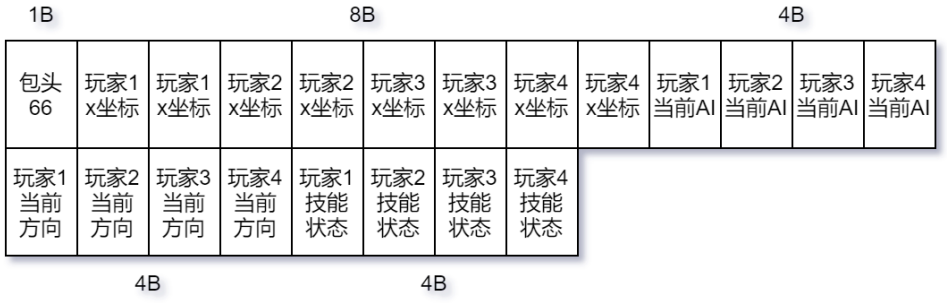

图 10 玩家技能信息通信包格式

3 技术问题

3.1 网络通信的方式

在完成了本地的存档（磁盘文件 IO）之后，认识到了网络通信也不过是另一个封装成 IO 的方式。在本地存档中使用了 Java 自带的对象序列化功能，其对于循环引用的处理也无需再进行额外处理，十分方便。但是在网络通信中，尝试使用对象序列化进行发包之后，遇到了以下的问题：

1. NIO Selector 使用非阻塞的通信方式，数据的操作对象是 Buffer 而不是 Stream。这样就使得 ObjectStream 的对象序列化方式需要进行一定的额外处理。
2. 使用对象序列化得到的数据包体积过大，使得在远程主机上部署 Server 根本无法正常运行，只有在本地的回环地址上才可以勉强低刷新率运行。
3. 对象序列化带来了低容错度，传输过程中的一点差错都会使得客户端无法对数据包进行反序列化。大体积的数据包又使得错误出现的次数提升了许多。

作为解决方案，最终采用了设计理念中提到的，MyServer 自定义数据包的方式。在服务端和客户端都额外添加了封装通信包、拆解通信包的方法，虽然代码量上升，但是带来了以下的好处：

1. 封装通信包、拆解通信包都是对于 byte[]，即字节数组的操作。而 ByteBuffer 和 byte[] 之间的转换是很自然的。
2. 自定义字节序列化方式之后，通信包的体积明显减少。实际测量，在 20Hz 的更新频率下，四个玩家同时进行游戏，服务端的网络上行速率大约在 400kB/s。完全是可以接受的范围。
3. 采用自定义字节序列化方式，网络传输的容错率得到了很大的提升，甚至可以直接忽略单个字节的错误（反应到客户端的屏幕输出上的时间很短，几乎立刻就会被下一个正确的数据包所覆盖）

3.2 PlayScreen 的臃肿问题

由于 PlayScreen 直接接管了用户输入和屏幕输出，在其上直接维护游戏的状态似乎是最自然的(不需要考虑访问限制，效率似乎也是最高的)。在本地游戏中一个 “胖” 的 PlayScreen 和相对 “瘦” 的 World、Creature、CreatureAI 类最后证实也是可以正常工作的。

但是在网络通信中，由于 Screen 的实现使用了瘦客户端 OnlinePlayerScreen 类，因此应该将更多地游戏状态转移到游戏 “Model” 类如 World、Creature、CreatureAI 类中，而不是在一个 “View” 类 PlayScreen 中。如将冷却时间 int[] coolTime 转移到了 Player 类中。

3.3 多线程并发问题

每一个生物体都占用着一个线程，大量的线程并发看似会带来不小的开发难度，但是实际上这些线程的并发模式十分简单：对于临界区的写入非常少（只有玩家修改 World 中的墙壁 Tile)，绝大多数对于临界区的操作都是读取。

因此处理多线程并发的策略相对简单，主要有以下几种：

1. 使用并发容器。尤其是 ConcurrentLinkedDeque 类。ConcurrentLinkedDeque 类最终的呈现效果是*弱一致性*，对于本程序的并发模式来说已经足够了。
2. 对于较高频次写入同时的较长时间的读取操作（即 OnlinePlayScreen 的 displayScreen)，使用 synchronized 操作进行原子化。

4 工程问题

开发工具使用的是 Visua Studio Code，较短的学习周期和相对轻量的开发环境，在大作业开发前期体现出了简单高效的特点。但是在开发后期的自动构建工具和单元测试部署上，就体现出了许多的不便之处。

4.1 Maven 的使用

Maven 的初始化在 Visual Studio Code 中是一键生成的，但是在实际的部署过程中还是出现了许多问题

4.1.1 jar 包的导出

在使用 mvn package 导出了 jar 包之后，直接使用鼠标双击运行没有反应，使用命令行运行之后发现报告 “无法加载主类” 的错误。

解决方法是，在 maven-jar-plugin 的插件配置中，加入 <mainClass> 属性块。

4.1.2 资源文件的使用

Maven 默认将资源文件路径设为./src/main/resources 目录下。但是在加载时直接使用 File 对象加载对应路径，发现无法正确获得资源文件的信息，并抛出 IOException 异常。

解决方法是，Maven 实际上将资源文件也放到了 class path 之中，加载资源应该使用类加载器的机制。即使用 getClassLoader().getResource() 函数来加载 Maven 管理下的外部文件资源。

4.2 单元测试部署

单元测试在部署的过程中，遇到了一系列的问题，如：

1. Visual Studio Code 中的 Coverage Gutters 插件和 Maven 的配合使用问题
2. 使用专门的测试类测试 main 中的实际工程文件，所遇到的包访问权限问题
3. 对于耦合度较高的，且产生了副作用的方法，如何构造测试用例的问题

对于问题 1 仅仅是工具的使用问题，可以通过查看插件的文档得以解决；对于问题 2，可以通过较为“暴力”的方法解决，即将需要测试的所有方法的包访问限制设为 public，但是这样会破坏整体的一致性；对于问题 3，只能将整个工程的代码进行一个较大程度的重构，由于时间关系没有并没有如此进行。因此最后单元测试并没有达到 50% 的高覆盖率。

5 课程感言

Java 作为一种编程语言本身，好用的地方可能就是有着大量的第三方库可以使用。在实际应用的开发过程中（在这个大作业中尤其是网络通信和 UI 的框架），找到一个成熟和灵活的第三方库，要远远比起自己再造轮子要高效和有成果的多。

这门课作为一门将编程语言作为主题的课程（即使有着“高级”二字），大量的调包似乎并不能够对语言的特性和更加通用的工程管理方式有进一步更深层的认识和理解。因此希望比起最后比拼谁最后的成果更加“酷炫”，甚至不惜使用成熟的第三方游戏引擎，在给定框架下的（也不能完全不用第三方库）自己程序的自洽和优雅可能是更重要的。