

Developing a Java Game from Scratch

郭百川

南京大学计算机科学与技术系, 南京 210023

E-mail: 201220087@smail.nju.edu.cn

摘要 本项目使用 Java 17 编写了一个简单的多人游戏, 使用 Maven 构建, GUI 界面基于 Java Swing, 网络交互基于 Java NIO。本文将介绍该项目的游戏设计及相关技术细节。

关键词 Java 游戏开发, Java Swing, Java NIO, Maven

1 项目介绍

1.1 概述

该项目使用 Java 语言, 实现了一个多人游戏, 至多四个玩家可以同时连接到一个游戏服务端上游玩。项目基于 Java Swing 制作了一个简单的 GUI 游戏界面, 可对玩家的操作做出实时的反应。网络交互部分基于 Java NIO, 非阻塞式的网络交互架构可以使服务端处理多个客户端交互时, 避免创建过多线程, 节约资源。

测试视频: <https://www.bilibili.com/video/BV1NY4y1o7H1>



图 1 游戏界面

1.2 我做了什么？

1. 定期更新的游戏状态
2. 玩家角色和敌人的交互
3. 多样化的敌人
4. 简单的 GUI 游戏界面
5. 存档和读档功能
6. 使用 Maven 构建项目，打包生成了可运行的 jar 文件
7. 实现了网络通信，支持多人同时游玩

2 游戏设计

此部分将介绍游戏的宏观设计

2.1 灵感来源

游戏灵感来自节奏地牢（Crypt of the NecroDancer），一款由 Brace Yourself Games 开发的 Roguelite 地牢探险游戏。基本的操作十分简单，类似传统的玩家和敌人先后行动的棋类游戏，但是区别在于节奏地牢限制了玩家回合的思考时间，加快了游戏节奏，更加契合 Roguelite 的游戏类型，同时也对玩家的反应提出了更高的要求。



图 2 节奏地牢游戏截图

本游戏的设计与节奏地牢有很高的相似性。

2.2 启动参数

服务端 GameServer

<saveName> <newSaveName> <port>

依次表示原存档名、新存档名、运行端口号。

<newSaveName> <port>

依次表示新存档名、运行端口号。将使用默认初始存档启动游戏。

启动后会在命令行中显示服务端的局域网 ip 和运行端口。

客户端 GameClient

<ip> <port>

依次表示服务端 ip 、服务端端口号。

2.3 基本操作

游戏设计为 500ms 一个回合，玩家回合和敌人回合交替进行，回合时间结束后行动。每个玩家角色都有一个运动方向，玩家可通过 WASD 或方向键操控。按键 I 可让服务端存档，按键 P 可让游戏暂停/恢复。

2.4 地图设计

目前设计了两种地形，以下为地形介绍：

地面 Floor



id: 0

生物可以通过**地面地形**。

墙 Wall



id: 1

生物不可以通过**墙地形**，生物尝试进入**墙地形**位置的行为会失败。

2.5 生物设计

目前共设计了五种生物，包括一种玩家操控角色和四种敌人，以下为生物介绍：

小人 Player



生命：30

攻击力：5

方向由用户控制（WASD 或方向键），每回合移动一格，如果目标位置有敌对生物，会发起攻击。

史莱姆 Slime



生命：5

攻击力：5

，每回合移动一格，如果目标位置有敌对生物，会发起攻击。每回合结束前**史莱姆**都会右转，呈现顺时针运动。

史莱姆的移动频率与**小人**相同，由于位置的奇偶性，**小人**和**史莱姆**交互的时机有限，再加上生物较多时，**史莱姆**的运动轨迹会比较复杂，因此**史莱姆**更多是干扰玩家判断而非直接造成威胁，但是如果玩家想要击杀**史莱姆**，需仔细调整**小人**位置。

骷髅 Skeleton



生命：15

攻击力：5

每两回合行动一次，每次行动移动一格，追踪玩家，如果目标位置有敌对生物，会发起攻击。与**骷髅**的对抗是游戏的核心对抗，击杀**骷髅**需要三次攻击，要求玩家要做至少一次规避动作

蘑菇 Fungus



生命：10

攻击力：5

无法移动，每四回合行动一次，每次行动发起攻击，对上下左右 3 格共 12 格中的敌对生物造成伤害。

蘑菇能够进行远距离的攻击，玩家需要注意周边**蘑菇**的位置以避免受伤，击杀**蘑菇**也需要注意**蘑菇**的攻击周期。但由于**蘑菇**无法移动，所以**蘑菇**主要起限制玩家行动范围的作用。

炸弹怪 Boomy



生命: 20

攻击力: 10

每回合行动一次, 每次行动移动两格, 追踪玩家, 与玩家距离(哈密顿距离, 下同)为 1 后不再移动, 两个回合后爆炸, 对离爆炸点距离小于等于 2 的所有生物造成伤害。

炸弹怪移动迅速, 玩家无法永远远离**炸弹怪**, 且它的生命让玩家无法在它爆炸前击杀, 因此**炸弹怪**紧贴玩家后, 玩家需要迅速做规避动作。由于**炸弹怪**的爆炸对所有生物都会造成伤害, 因此诱导**炸弹怪**在一群敌人中爆炸会是个不错的策略。

3 技术细节

3.1 概述

本项目共分为三大模块: 游戏逻辑、GUI 界面、网络交互, 另外需要强调的部分还有并发控制、文件 IO、工程构建与测试, 因此该部分将分 6 点做介绍。

3.2 游戏逻辑

游戏逻辑相关类关系如下:

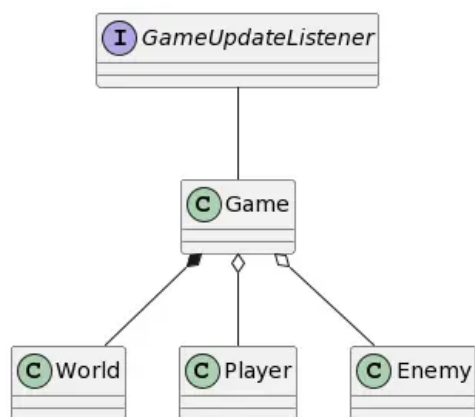


图 3 游戏主循环相关类

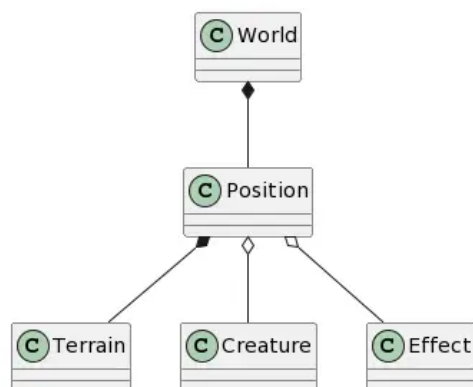


图 4 世界相关类

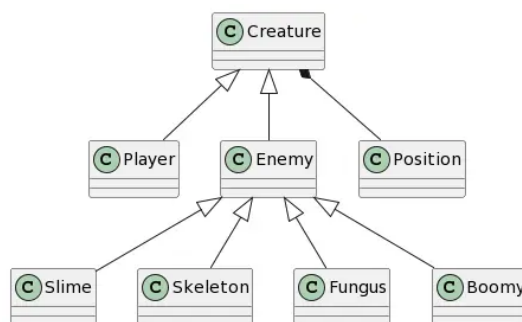


图 5 生物相关类

Game 代表整个游戏进程。

World 代表游戏世界，包含一个 **Position** 矩阵。每个 **Position** 都有一个 **Terrain** 属性表示该位置的地形，另外还可以容纳一个生物 **Creature** 和多个绘制效果 **Effect**。

Game 包含一个 **Timer** 类型计时器，游戏主循环由这个计时器驱动。计时器结束后会触发 **Game** 类实现的 `actionPerform()`，继而调用 `update()` 做游戏状态的更新，更新结束后，会向实现了 **GameUpdateListener** 接口的上层模块发出信号，通知游戏状态有更新。

每次调用 `update()` 时，会先根据回合类型，调用所有玩家或敌人的 `update()` 方法，之后清除所有死亡的玩家或敌人。如果本回合是敌人回合，还会将敌人刷新倒计时减 1，敌人刷新倒计时归零后，会在随机位置生成一个随机敌人，并重置倒计时，倒计时重置规则为 $\text{summonCountDown} = 10 - 2 * \text{playerNum} + \text{random.nextInt}(5)$ ，使更多玩家加入时敌人生成更频繁，并引入随机数以防止敌人的周期行动出现无趣的一致性。

3.3 GUI 界面

GUI 游戏界面基于 Java Swing，相关类关系如下：

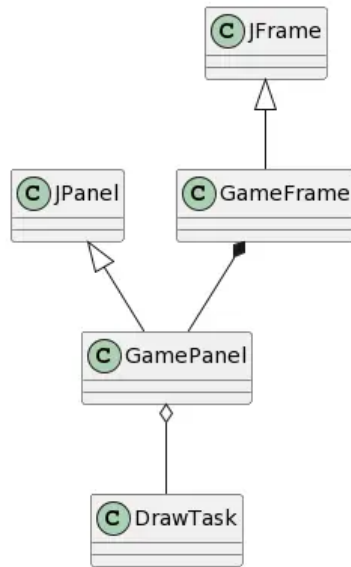


图 6 界面相关类

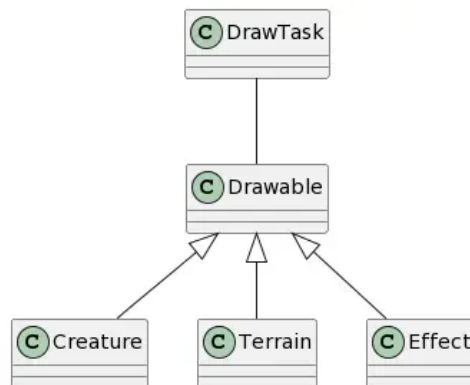


图 7 绘制任务相关类

GameFrame 表示游戏窗口，可以监测玩家的键盘操作。包含一个 GameFrame.GamePanel，重写了 paint() 方法，每次重绘时被调用，所有绘图操作都在绘制在 GamePanel 上。GamePanel 持有一个 DrawTask 列表，每次重绘时根据此列表绘制。

DrawTask 表示一个绘制任务，使用链式的类型设计，持有一个 DrawTask 类型成员表示前景，绘制任务执行时会递归绘制前景。游戏中需要绘制的元素都实现了接口 Drawable，用于生成一个无前景的 DrawTask。

3.4 网络交互

网络交互基于 Java NIO，在服务端和客户端之间建立非阻塞的通讯信道，并设计了一套通讯协议。考虑到网络交互情况复杂，此部分相关设计将着重说明。

相关类关系如图：

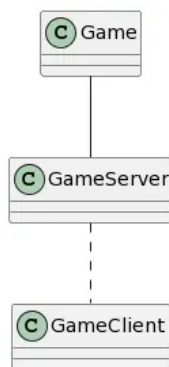


图 8 服务端相关类

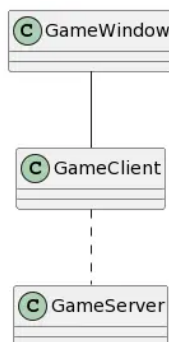


图 9 客户端相关类

3.4.1 主要线程

客户端和服务端共有四个主要线程：

GameServer：服务端，有一个主循环，负责与多个客户端的通信。

Game：由一个 Timer 驱动，定时更新游戏状态，生成绘制任务列表并通知服务端分发。

GameClient：客户端，有一个主循环，负责与服务端的通信。

GameWindow：客户界面，从客户端接收绘制任务列表并显示游戏画面，接收用户操作并将用户操作传递到客户端。

可以发现，无论是客户端还是服务端，都需要处理异步事件（玩家操作、绘制任务分发通知）。客户端由于主循环仅做读取，对于异步的玩家操作直接发送给服务端即可；服务端主循环读写都有涉及，因此准备好绘制任务列表并通知服务端后，服务端会在下一次主循环同步处理，将绘制任务列表分发给各个客户端。

3.4.2 通信协议

服务端到客户端协议

LINKED <playerID> <width> <height>

新通道产生后, 向客户端回报客户对应的玩家 id、世界尺寸。

DRAWTASKS <drawTaskListSize>

分发绘制任务列表, 之后会跟随一个序列化的包含若干 DrawTask 对象的 'List。drawTaskListSize 表示之后绘制任务列表对象的大小。

GAMEOVER

游戏结束后, 通知客户端结束程序。

MESSAGE <message>

传输普通的字符串信息, 主要用于调试。

客户端到服务端协议

DIRECTION <id> <direction>

改变对应玩家的运动方向。

SAVE

通知服务端存档。

PAUSE

通知服务端暂停或恢复游戏。

MESSAGE <message>

传输普通的字符串信息, 主要用于调试。

3.4.3 客户端游戏过程统一

为了保持各客户端所见游戏过程一致 Game 对象唯一, 运行在服务端, 服务端会将绘制任务列表分发到各个客户端, 客户端根据绘制任务列表自行绘制界面; 客户端检测到用户操作后, 将操作内容发送到服务端作用于 Game 对象。

这种设计避免了两个问题: 1. 若采用分布式的多 Game 对象同时运行的设计, 用户操作发生后, 由于网络通信的时延, 可能会出现不同 Game 对象处理操作时机不一致的现象, 继而导致各用户所见游戏过程不一致。2. 分发绘制任务列表, 而不是完整的 Game 实例或完整的绘制结果, 减少了通信的信息量。

但这种设计也有隐患, 这种设计方式成立的关键前提是此游戏绘制内容简单。对于绘制内容更复杂的游戏, 即使是分发绘制任务列表也会显得十分笨重。此时在各个服务端都运行一个 Game, 自行生成绘制内容就十分必要了。

实测网络情况糟糕时, 客户端可能会崩溃, 暂时未确认崩溃原因, 猜测为服务端写缓冲区满导致客户端接收内容混乱。

3.4.4 程序结束

考虑了两类导致程序结束的事件：客户端窗口关闭和游戏结束

GameWindow 重写了 processWindowEvent(), 监听到窗口关闭事件后, 通知客户端设置结束标志位以结束主循环, 释放相关资源后结束客户端程序。服务端不会做太多响应, 原属于此客户端的玩家还会继续行动 (当然, 不再会改变方向)。

当 Game 启动后, 所有玩家都死亡, 游戏结束。服务端主循环的条件是游戏未结束, 游戏结束后, 服务端会向各个客户端通知游戏已结束, 通知完毕后结束服务端进程。客户端接收到游戏结束通知后, 处理过程与客户端窗口关闭类似。

3.5 并发控制

除去上节所述的四个主要线程外, 由于此游戏设计位置之间分割明确, 适合采用了并发技术进行更新、绘图, 但是使用并发技术时要注意线程间的同步和竞争, 以及要采用线程安全的容器。

3.5.1 并发框架设计

程序共有三处大规模并发, 包括生物行动、位置提交绘制任务、执行绘制任务, 都采用了类似的设计。以下以生物行动为例做介绍。

Creature 为了提供多线程支持, 拥有一个内部类 CreatureUpdate, 继承自 Runnable, 其中的 run() 会调用 update(), 上层通过调用 Creature 对象的 getUpdate() 即可获得 Creature.CreatureUpdate 对象。

Game 调用各 Creature 的 Update() 时, 会创建若干个线程, 交由一个线程池并发执行, 利用 CountdownLatch, 待所有线程执行完毕后再返回。

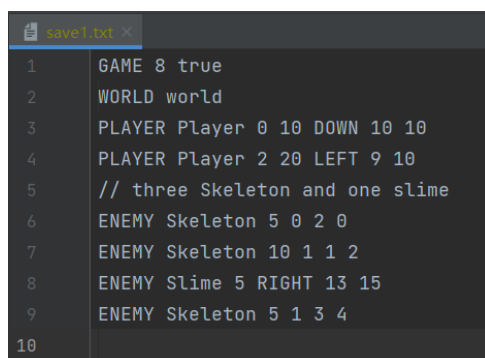
3.5.2 同步和竞争

线程同步如上一小节描述, 利用 CountdownLatch, 主线程等待所有子线程结束后再继续执行。由于游戏设计中位置间分割明确, 使得可能出现的竞争较少。最主要的竞争出现在生物行动时, 多个生物尝试进入同一个位置: 每个 Position 对象至多容纳一个生物 creatureGetIn() 和 creatureLeave() 都带有 synchronized 属性。creatureGetIn() 让一个 Creature 对象尝试进入本位置, 会做地形阻挡和其他生物阻挡判断, 返回一个 boolean 表示进入是否成功; creatureLeave() 让处于本位置的 Creature 对象离开, 简单地将 this.creature 设置为 null 即可。

3.6 文件 IO

3.6.1 存档文件

存档文件为文本文件, 由若干条记录构成。如前文所述, 服务端启动时会指定一个起始存档文件和一个新存档文件, 游戏启动时会解析存档中的各条记录生成相应的游戏内容, 玩家控制存档时会将游戏状态转化为若干条记录存入新存档文件。



```

1  GAME 8 true
2  WORLD world
3  PLAYER Player 0 10 DOWN 10 10
4  PLAYER Player 2 20 LEFT 9 10
5  // three Skeleton and one slime
6  ENEMY Skeleton 5 0 2 0
7  ENEMY Skeleton 10 1 1 2
8  ENEMY Slime 5 RIGHT 13 15
9  ENEMY Skeleton 5 1 3 4
10

```

图 10 一个典型的存档文件

记录格式如下:

GAME <summonCountDown> <isPlayerTurn>

必选, 仅一项。

表示游戏的状态。

WORLD <worldName>

必选, 仅一项。

表示世界名称, 读取时通过此条目得知该读取哪个世界描述文件。

PLAYER <playerType> <id> <hp> <direction> <x> <y>

可选, 至多四项。

表示一个玩家的状态。

ENEMY <enemyType> ... <x> <y>

可选, 数量无限制。(当然, 实际的游戏存档中, ENEMY 条目数量不可能超过世界的位置数)

表示一个敌人的状态。其中 ‘...’ 包含若干项, 表示具体的敌人的状态, 含义由具体的敌人类定义。

3.6.2 世界描述文件

描述文件第一行为两个正整数, 分别表示地图宽度 width 和高度 height; 之后是一个 width * height 大小的非负整数矩阵, 表示地图各个位置的地形 id。



图 11 一个典型的世界描述文件

3.6.3 资源文件

除去源代码外，程序运行还需要许多资源文件的支持，程序运行时需要读取这些资源文件。

本项目中涉及的资源文件包括图片素材、存档文件、默认存档文件、世界描述文件。这些资源文件分为两类：第一类文件只被读取，自项目打包发布后不再发生变化，图片素材、默认存档文件、世界描述文件属于此类；第二类文件会因为程序运行被创建或被修改，存档文件属于此类。开发完成后，第一类文件可被打包入 jar 发布，所以需要放入 resources 目录；第二类则文件不需要被打包加入 jar，因此统一在 saves 目录后置于项目根目录，发布后则置于 jar 同目录。

第二类文件的读取方式较简单，直接根据路径读取即可。第一类文件考虑到发布后将被打包进 jar 文件，不能直接通过路径读取，需要调用 `getResourceAsStream()` 方法根据相对于 resources 目录的路径读取。

3.7 工程构建与测试

3.7.1 工程构建

本项目使用 IntelliJ IDEA 开发，使用 Maven 构建。实际上依赖较少，因此 pom.xml 没有过多的修改。

打包后的 jar 文件已发布在 github 的 release 栏，经测试可以运行。

3.7.2 测试

单元测试在作业 j06 后没有做更多补充。

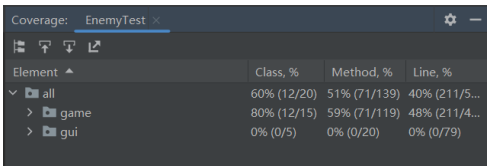


图 12 j06 阶段的测试结果

4 更多，更好

这次项目是我的第二次面向对象项目开发，但是这次收益不比上次小。这次项目开发是我第一次涉及并发编程、网络编程和代码测试，或许能为未来的学习和工作积攒宝贵的经验。不过对于这次的项目，我觉得还有许多需要改进的地方。

让用户通过 main 方法参数传递必要参数的方法非常不友好，未来可以改用 GUI 界面实现。

我自认为网络交互相关的代码写得很糟糕，逻辑混乱，功能分割也有待改进。目前我发现网络交互部分有一个 bug：在网络条件糟糕的情景下运行时，客户端可能会崩溃，猜测可能是由服务端写缓冲区满造成的断包现象导致。

虽然已做简化设计，但是通过网络传输绘制任务列表依然是一个繁重的任务，未来可以采用多线程的方法，将传输绘制任务列表这一操作另建一个线程执行，服务端主线程即可马上返回主循环处理其他交互。

重绘游戏窗口时，每次都需要重绘整个游戏世界，但事实上很多位置相比上一次更新并没有变化，未来可以考虑采用仅重绘更新位置的方式，这能够大大减小通过网络传输的绘制任务数。

开发时发现，按照目前的代码框架，编写新敌人时，需要修改的类过多，除了新敌人自己的代码外，还需要更新随机生成敌人和读取 ENEMY 存档条目的 switch case 代码块。这种开发体验很糟糕，或许能通过反射机制解决这个问题。

最开始设计项目框架时，有计划实现道具 Item 类，再实现一个恢复玩家生命值的道具，开发时也留了部分给该功能的接口。但是这个功能现在暂未实现。

5 致谢

感谢优秀的像素画制作平台 *Pixlart*，本项目的所有素材都使用此平台制作。

感谢 <https://www.jianshu.com/p/03055b90c596> 提供的 ByteBuffer 转 InputStream 的方案。

感谢 <https://www.cnblogs.com/tfxz/p/12621599.html> 提供的局域网内 IP 获取方法。

Developing a Java Game from Scratch

Baichuan Guo

Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
E-mail: 201220087@smail.nju.edu.cn

Abstract This project has developed a simple multi-player game. The project is constructed by Maven. GUI is based on Java Swing. Network communicating is based on Java NIO. The article will introduce the game design and relevant technical details.

Keywords Java game development, Java Swing, Java NIO, Maven