中国科学: 信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

王毓琦

1. 南京大学, 南京 210023

E-mail: ricky.wang@smail.nju.edu.cn

收稿日期: 2022-01-01; 接受日期: 2021-01-01; 网络出版日期: 2022-01-01

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

摘要 一个基于 FXGL 11.7 的简单的吃豆人小游戏, 支持游戏保存和加载, 支持最多 4 人通过网络连接对战. 针对 FXGL 框架网络模块中的一些功能和性能不足问题做了一些简单的优化.

关键词 Java, 游戏开发, JavaFX, FXGL

1 游戏介绍

一个简单的吃豆人小游戏, 支持最多 4 人网络对战.

1.1 灵感来源

功能机时代手机上总会附带一些以.jar 结尾的游戏, 小时候并不知道这些后缀名的含义, 现在回想起来才发现原来这些手机上就跑着一个 Java 虚拟机, 这些游戏也是用 Java 开发并打包的. 因此想复刻一个吃豆人小游戏.

1.2 操作方法

使用 WASD 改变玩家移动方向.

1.3 游戏机制

玩家需要收集尽可能多的豆子, 每吃到一个豆子加 50 分. 两个玩家相遇时, 得分更高的玩家可以吃掉得分低的玩家.

出现如下情况时游戏结束:

1. 时间耗尽. 游戏中有一个计时器, 初步设定为 100 秒. 时间耗尽则游戏结束, 得分最高的玩家获胜.

引用格式: 王毓琦. Developing a Java Game from Scratch. 中国科学: 信息科学, 在审文章
Yuqi Wang. Developing a Java Game from Scratch (in Chinese). Sci Sin Inform, for review

2. 存活最久. 如果地图上只剩一个玩家, 则该玩家获胜.

为了防止某些玩家”苟”在角落里通过规则 2 获胜 (类比各种吃鸡游戏里的打法), 如果某个玩家一段时间内没有得分进账, 将会扣去 50 分作为”体能消耗”, 借此鼓励左右玩家活跃起来.

1.4 效果展示

游戏运行时截图如下:

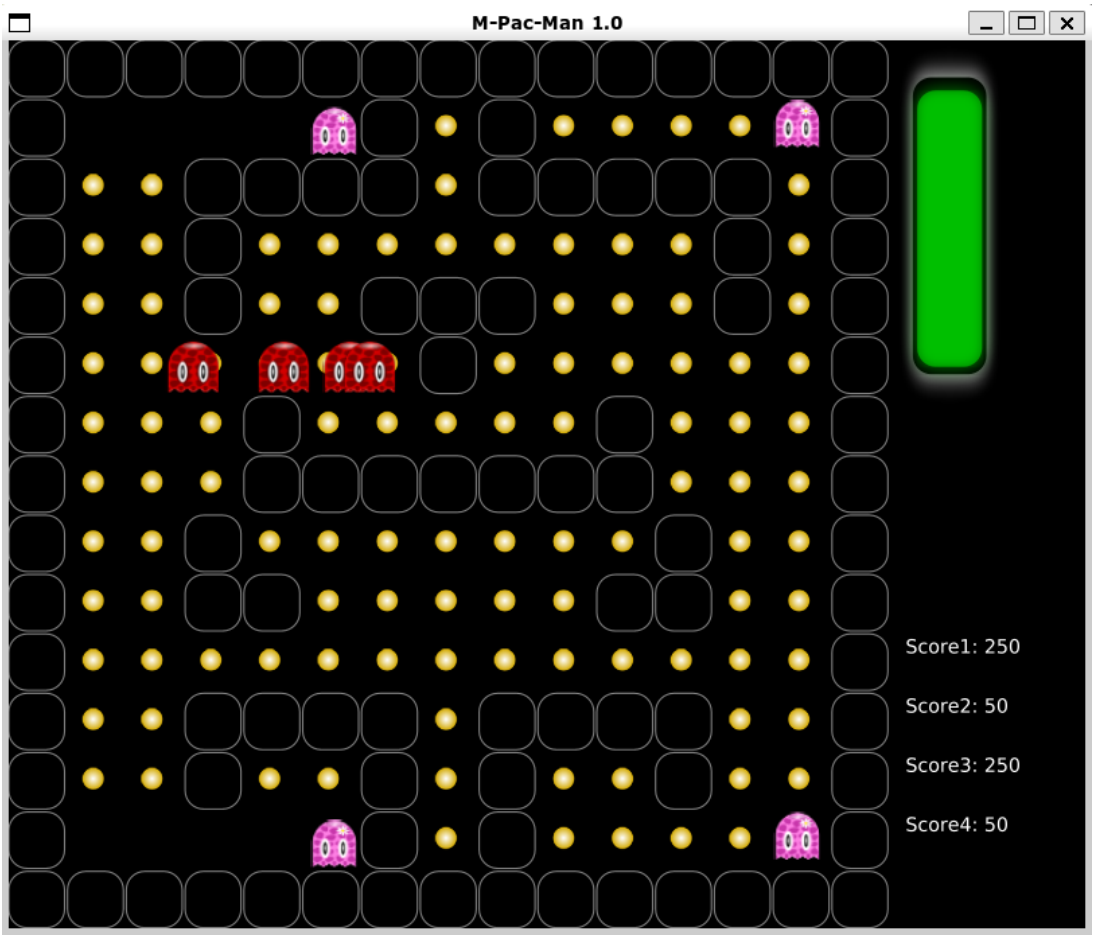

图 1 游戏截图

JUnit 单元测试覆盖率如下:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
pac	64.2 %	2,325	1,294	3,619
src/main/java	60.6 %	1,945	1,262	3,207
src/test/java	92.2 %	380	32	412

图 2 测试覆盖率

2 游戏开发

2.1 设计理念

游戏开发过程中我粗浅地学习了一些游戏引擎开发的相关资料, 对于游戏引擎的设计理念也有了一些收获和自己的思考.

2.1.1 搭积木

游戏中的实体可以采用类似” 搭积木” 的方式实现. 例如实体的显示部分可以放在 texture 模块, 实体的移动可以放在 MoveComponent 模块, 实体的生命值可以放在 HealthComponent 模块等等. 要给实体添加某种属性或者能力, 只需要给它加上某个模块就行了.

可以实现一系列通用模块供所有实体使用. 模块自身是功能完备的, 独立于其他模块和所有实体的. 这样对所有模块都可以分别添加单元测试保证代码可靠性, 不同游戏都可以复用同一个模块, 提高代码复用率.

2.1.2 功能服务化

游戏开发中往往需要用到一些相同的功能模块, 例如文件 I/O, 网络通信, 资源管理等. 可以将这些功能” 服务化” 并实现在单独的模块中. 在游戏应用中, 如果要使用这些功能, 只需要注册相应服务就可以直接将其调出. 服务与具体游戏逻辑的实现相分离, 并且游戏开发中需要调用服务时做到开箱即用, 可以让游戏主题逻辑更加清晰.

2.1.3 字段存取

游戏中会有很多全局变量, 例如时间, 关卡等信息. 使用单独的变量存储是最直观的方案, 但是使用起来往往有诸多不便. 为此, 可以在游戏中放置一个字典, 将所有全局信息存储到字典中之后就可以直接根据变量名调用相应变量.

这种方式最大的缺点在于, 编译器无法对开发者提供的字段进行检查, 因此出现字段错误, 类型错误等问题的可能性很高. 因此使用这种方式存储和调用变量时需要进行更加严格的检查, 测试和异常处理, 以便在错误发生时能够快速定位问题.

此外, 某些引擎还支持将函数名以字符串方式传入从而调用某些方法. 开发过程中同样无法得到编译器的提示, 因此同样需要比较严格的检查和测试.

这两种方法都能一定程度上提高引擎的易用性, 并且某些情况下可以让游戏主要逻辑更加清晰, 在能够保证引擎鲁棒性的前提下可以加入相关支持.

2.1.4 分离游戏逻辑

游戏引擎需要保证比较好的执行效率, 因此一般用 C 语言等性能较好的语言开发. 但是这类语言往往开发效率偏低, 因此对效率不敏感而对开发效率敏感的游戏控制逻辑部分就可以采用其他语言实现. 学习过程中我发现, 不少著名的游戏引擎都支持这一分离策略.

要实现这样的分离就需要游戏引擎提供比较好的统一的 API, 脚本语言可以通过这些 API 方便地操控游戏中的所有组件实现相应目的.

除了核心的游戏逻辑之外, 很多数据也可以通过" 外部" 的实现来支持. 例如本项目中的 UI 控制部分, 使用 xml 文件来结构化地存储 UI 布局的相关信息, 只要读取 xml 配置文件就可以知道 UI 中各个组件的摆放位置等信息.

2.1.5 结构化信息

游戏开发往往大量涉及数据存储和传输, 例如存档, 回放, 网络通信等功能都需要以某种方式将数据进行格式化再进行相应操作. 预先定义好数据结构, 再将这些数据放到指定位置, 用到的时候按照某种规范统一地提取, 这样可以大大减轻工作负担.

2.2 同步方式

网络游戏中需要保持每个客户端中的表现效果一致, 帧同步和状态同步是保证这种一致性的最常用的两种方法. 学习两种不同同步模式的原理及其特点, 选择最适合本项目使用的同步方式, 这是本次开发过程中网络模块的重点之一.

帧同步和状态同步最大的区别在于游戏核心逻辑写在哪.

帧同步的核心逻辑在客户端中, 服务器只负责将各个客户端中玩家的操作进行转发, 客户端收到服务器发来的信息后进行相应的处理. 这种模式下服务器端需要进行的处理较少, 处理任务分散在各个客户端中; 服务器端的处理逻辑也相对简单. 但是帧同步也存在一些问题:

1. 随机事件处理相对复杂. 游戏中经常用到一些随机事件, 例如 NPC 的随机移动, 物品的随机掉落等. 在不同客户端之间同步这些随机事件是重点. 一种可行的方案是实现一种可靠的伪随机数发生器, 各个客户端采用相同的种子, 服务器负责产生和分发随机种子, 以在各个客户端中生成相同的" 随机" 事件.

2. 复杂网络环境下的鲁棒性维护. 游戏核心逻辑分散在不同客户端中进行处理, 当网络不畅, 客户端之间无法维持正常通信时, 如何恢复和保持这些客户端的状态一致相对比较难处理. 例如有客户端掉线, 则服务器可能需要将掉线后到恢复这一段时间的所有操作发送给客户端, 客户端加速执行这些逻辑以将自身同步到最新的统一状态.

状态同步则将绝大部分游戏逻辑放在服务器端进行处理. 客户端监视用户输入, 并将这些输入定时发送给服务器. 服务器对所有客户端的输入进行处理并将处理后的游戏状态广播给所有客户端. 这种模式下, 服务器能够以" 上帝视角" 处理问题, 因此处理逻辑相对更好实现.

但是状态同步也存在一些问题. 由于核心逻辑都在服务器端进行处理, 因此对服务器的压力较大; 此外如果游戏比较复杂, 每次更新涉及状态比较多的话对网络要求也更高.

考虑到本次项目要实现的游戏逻辑并不复杂, 且主要运行环境是局域网或本机, 因此本次项目采用状态同步的模式实现.

实际开发中也发现一些问题. FXGL 本身提供的网络模块功能还不完善, 且性能不足, 因此本项目重新实现了状态更新的逻辑.

2.3 输入输出

由于游戏使用状态同步模式, 因此输入输出的处理相对比较简答. 同时运行着服务器的游戏程序直接在本地处理用户输入即可, 而客户端程序则需要将所有数据打包发送给服务器. 输入数据已经封装成 Input 类, 将 Input 类序列化并通过网络发送给服务器即可.

2.4 效率优化

首先考虑需要进行同步的项目. 简单列举如下:

1. 游戏中的全局状态, 例如时间, 得分等

2. 游戏中的各个" 实体" 及其属性. 例如对" 玩家" 实体, 可能需要同步其位置, 血量等信息.

全局状态信息一般比较简单, 也都是基本的数据类型, 序列化后通过网络发送即可. 实体的同步就相对复杂, 实现方式也有很多, 也是游戏中需要着重优化的地方.

最简单粗暴的思路可能是, 在游戏中维护一个全局的" 实体" 集合, 每次同步时将集合中的实体全部序列化并通过网络发送, 客户端收到消息后再反序列化, 按照发来的数据重新生成所有实体. 但是这会带来大量重复工作:

1. 实体中某些属性是不变的, 无需每次都更新

2. 某些实体是不变的, 同样无需每次都更新

因此在进行同步时需要对同步的实体和实体中需要同步的信息进行选择.

本项目中, 地图当中的" 墙" 是保持不变的, 无需进行同步; 玩家, 敌人, 豆子等物体则会发生变化, 因此需要同步. 这些实体需要有 3 中同步操作:

1. 创建. 例如游戏初始化时服务器端通知客户端各个实体的生成位置

2. 移动. 例如玩家和怪物都需要移动

3. 删除. 例如玩家吃掉某个豆子后需要将后者从世界中移除

在实现网络通信时, 先定义了通信字段, 字段中包含操作种类, 客户端收到信息后根据字段中的种类信息选择合适的处理方式.

如果想要做到客户端无需重新生成全部实体, 则需要能够根据某种方式访问实体, 并对实体做出某些修改. FXGL 提供了 IDComponent 来标记实体, 并提供了统一的接口来根据 ID 访问指定实体. 在 ID 部分, 要求每个实体的 ID 全局唯一. Java 库提供了 UUID 来生成永不重复的全局 ID. 结合这两个库就可以实现实体的选择性同步.

FXGL 中原有的网络模块也提供了服务器-客户端的同步, 但是这一同步是基于连接的, 因此对每个连接都需要维护一组同步, 无形中增加了工作量; 这种同步也只是简单地将所有实体全部复制一遍发送给客户端, 因此做了很多不必要的工作. 实际测试中发现使用 FXGL 的实现, 地图稍大 (21x21) 就会出现明显卡顿, CPU 占用率也上升到 50% 左右. 而使用改进后的同步方案, 需要同步的数据量小了很多 (一般每帧只有 10 条左右信息), 保证 4 个客户端同时流畅运行的同时 CPU 占用率也只有 20% 以下.

后续工作: 以后如果有时间的话可能会再研究 FXGL 网络通信部分的源码, 增加一些实现, 甚至提出 PR.

3 工程方法

3.1 协议化和规范化

开发和文档工作一起进行可以一定程度上让工程开发更加规范, 从而提高效率. 例如本次项目中, 在实现网络通信时, 由于要发送的消息中包含很多字段, 因此需要仔细考虑如何让这些字段能够统一地封装在一个类当中, 并且能够用统一的方法对消息进行处理. 类比网络通信协议的确立, 设计字段时也预先考虑好所有情况, 再将字段的定义规范化写进注释和文档, 开发时速查. 这样可以大大提高开发效率, 降低写到后面忘了前面的可能性.

3.2 Tricks

代码越写越长, 折叠功能很有用...

4 课程感言

很喜欢老曹的讲课风格, 简单的东西给出链接自学, 课上讲一些需要重点理解的东西, 并且多做拓展. 这样就在比较有限的时间里能把主要精力放在对一些核心概念的理解上, 例如 JVM 虚拟机的工作方式, 类如何被加载等等.

此外 Java 课上不仅能学到 Java 语言, 还能学到一些语言以外的东西, 包括一些设计理念和方法论. 例如网络通信部分的 Reactor 设计模式等, 已经不仅仅局限于 Java, 其他开发领域中也可以用到这些模式. 软件测试等内容也是非常实用的知识, 而这些知识常常被其他课程忽略.

其实一直希望系里能开一个入门课, 讲一些开发环境配置, 命令行使用这些基本知识, 或者即使不讲也收集一些质量比较高的资源, 给出一个学习路线建议. 上大学以来, 自己花了不少时间配置开发环境, 学习 Linux 基础知识等, 付出了很多努力, 踩了很多坑之后才算搭好了一个比较好用的开发环境. 身边很多同学也经常反映各种开发环境配置上的问题. 如果能在这方面传授一些经验, 或许对学习和开发效率的提升有很大帮助. 麻省理工好像有一门课叫 The Missing Semester of Your CS Education, 主要讲了一些工具性知识, 希望我们也能有类似的活动 (我为什么要在这里写上这一段? 也算是一学期学习的感言吧, 好的环境对学习和开发效率的提升的确很大).

这学期后半段因为拖延症和一些其他原因, jw05 写得并不能让自己满意, 所以在 jw08 里弃用了之前的项目, 基于 FXGL 开发了新的游戏. 但是也由于时间所限, 对引擎的工作原理研究得也没有那么透彻. 以后如果有时间希望能继续跟着引擎项目的源码学习一些开发上的知识.

其实 jw05 以后自己也在尝试实现一个游戏引擎, 但是自己对它并不满意, 由于之前没有接触过游戏开发, 引擎设计上存在很多不合理的地方. jw08 之所以选择 FXGL, 很大一部分原因也是想学习成熟项目的设计理念. 日后争取能自己实现一个简易的游戏引擎吧.

最后, 祝老师身体健康, 工作顺利!

参考资料

1 FXGL Gitter
2 Java 序列化
3 FXGL Samples
4 Java NIO Implement of Reactor Model

Developing a Java Game from Scratch

Yuqi Wang

1. *Nanjing University, Nanjing* 210023*, China*
E-mail: ricky.wang@smail.nju.edu.cn

Abstract A Pac-man game based on FXGL 11.7 implemented in Java. Saving and reloading game is supported. It also supports multi-player mode for up to 4 players. Addressing the deficiencies in the FXGL framework, this project has made some simple optimizations.

Keywords Java, Game Developing, JavaFX, FXGL