中国科学 : 信息科学 在审文章: 1–5

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

罗一逖 231820309[1*]

1. 南京大学信息管理学院，江苏南京 210023

* 通信作者. E-mail: 231820309@smail.nju.edu.cn

摘要 本文阐述了一款结合 Roguelike 机制的 Java 动作对战游戏的设计与实现。游戏设有单人剧情、本地多人对战与网络对战等三种模式。系统基于面向对象思想，采用组件-实体系统（ECS）与工厂模式，实现了逻辑解耦与模块化开发。在核心技术方面，网络模块应用 Client-Server 架构与 Reactor 模式以提升通信吞吐量；并发控制通过线程池优化物理碰撞检测算法，解决了性能瓶颈；数据交互采用 Gson 库实现统一的 JSON 序列化处理。此外，本项目通过单元测试，覆盖度超过百分之五十，有效保障了代码的可维护性与开发效率。

关键词 Java 游戏开发；ECS 模式；Roguelike；Reactor；

1 开发目标

我开发的是一个葫芦娃闯关对战的小游戏。一共有三种主要的游戏模式：单人剧情模式、多人本地对战模式以及网络对战模式。单人剧情模式是游戏开发的核心，而多人本地对战和网络对战模式则是为了增加游戏的趣味性和可玩性，同时也是对 Java 网络开发能力的测试与实现。

其中，单人剧情模式的主要灵感来源于经典的 Roge Like 游戏佳作《哈迪斯》（Hades）。《哈迪斯》游戏以其具有挑战性的关卡设计、丰富的角色成长系统以及引人入胜的剧情叙述而闻名。我则是希望通过借鉴《哈迪斯》的设计理念，结合葫芦娃这一中国传统文化元素的叙述背景，再通过定义丰富多样的怪物模式和不同的玩家技能模式，来打造一个既有深度又富有趣味性的单人剧情模式。

具体而言，《哈迪斯》游戏有这样几个特点：玩家角色的随机强化，敌人怪物的随机生成，逐层递进的闯关模式和剧情等。我在我的游戏中也实现了类似的机制，比如，玩家在每一层关卡结束后会被赋予来自“金木水火土”五大元素不同类型的随机技能升级，而敌人怪物则是从预定义的 3 个类型中随机生成，这大大增加了游戏的不可预测性和挑战性。此外，游戏还包含了一个简单的剧情线，在玩家和怪物的交流过程中呈现，需要通过不断的闯关来揭开葫芦娃故事的全景。最后，游戏的关卡设计以中国传统的章回体格式呈现，共设计了“大娃：山脚遭遇战”、“二娃：山道伏击”、“三娃：铁壁防线”、“四娃：火焰试炼”、“五娃：水淹妖洞”、“六娃：隐身突袭”、“七娃最终关：收妖大战”七个关卡，玩家可以逐步挑战，完整的体验葫芦娃的游戏故事，并最终战胜妖怪，拯救爷爷。

并且，由于 Roge Like 类型的游戏通常具有极高的挑战性，我在设计关卡中也保持了这一特色。正常而言，游戏的难度会随着关卡的递进而逐渐增加，并且玩家自身的属性也会有一定的进步，但总体上会保证在一个高难度的状态下进行。经多个玩家的测试发现，一般第一次进行的玩家仅能勉强通过第一关，而经过多次重复尝试游玩之后的玩家则会不断提升自己的技巧，最终通关。证明了游戏的挑战性和重复可玩性。

引用格式: 罗一逖 231820309. Developing a Java Game from Scratch. 中国科学: 信息科学, 在审文章
罗一逖 231820309. Developing a Java Game from Scratch. Sci Sin Inform, for review

对于多人本地对战模式和网络对战模式，则是借鉴了经典的双人/多人对战游戏设计理念，比如《街头霸王》、《任天堂全明星大乱斗》等。玩家可以选择不同的葫芦娃角色进行对战，每个角色会随机到金木水火土中不同的技能属性。通过本地或网络连接，玩家可以与朋友进行实时对战，增加了游戏的互动性和趣味性。并且，多人对战的人数可以设置为 2-4 人，能够满足广泛的多人游戏需求。

总的来说，本游戏的总体开发目标就是通过借鉴 Roge Like 游戏的设计理念，结合葫芦娃这一中国传统文化元素，构造一个极具挑战性的 Java 对战游戏，同时通过多人本地对战和网络对战模式以增加游戏的互动性和可玩性。

2 设计理念

在本游戏的设计中，我总体遵循了面向对象编程（OOP）的设计理念。参照组件-实体系统 (ECS) 设计模式，游戏中的主要实体即使玩家角色、怪物角色、装饰性的树角色等，它们都被设计为了独立的类，并都继承自 GameObject 抽象类，并集合在 game 包中。在此基础上，每个类拥有来自 components 包中的多个组件，这些组件负责处理实体的不同功能模块，比如物理组件 (PhysicsComponent)、渲染 (RenderComponent)、生命组件 (LifeFeatureComponent) 以及对话组件等 (DialogueComponent)。通过这种方式，我实现了实体与组件的解耦，使得每个实体可以根据需要灵活地组合不同的组件，实现丰富的功能；并且，由于组件的存在，使得我在 GameLogic 类中可以通过统一处理组件而不是类的方式集中管理游戏逻辑，这样大大简化了游戏逻辑的设置过程，使得我能够从更高层次的抽象中处理游戏逻辑定义，提高了开发的效率。游戏类的整体结构如图 1 所示：

而游戏的主要页面则定义在 app 包中，这个包中的每一个类都是一个页面，比如主菜单页面 MenuScene, 单人游戏页面 GameScene, 录像回放页面 RecordingScene 等等，这些页面都继承自 GameScene 类。这使得游戏场景的定义和管理变得更加高效和容易。在统一继承的基础上定义每一个页面的特殊功能，使得我能够快速地创建和切换不同的游戏场景，同时保持代码的整洁和可维护性。值得注意的是，这些页面内并没有过多的定义游戏的具体功能。例如，OnlineGameScene 类中并没有定义任何有关网络通信逻辑的内容，其整体界面定义与单人游戏界面类 GameScene 基本一致，而其核心网络联机功能的实现则是通过调用 network 包中的方法进行的。这种设计使得网络通信逻辑与游戏场景解耦，提高了代码的模块化程度，使得网络功能可以独立开发和测试，同时也方便了后续的网络功能扩展和维护。

此外，在设计游戏关卡的时候发现 GameScene 类过于臃肿，严重影响了更复杂的关卡逻辑开发，所以我将关卡的定义和管理单独抽象成了 LevelManager 类，并且在 level 包中定义了多个具体的关卡实现类，比如 LevelConfig, EnemyFactory 等等。其中，EnemyFactory 类的定义参照了工厂模式，由它统一管理敌人的生成，使得怪物的生成地点和生成数量、属性等可以集中定义和管理。通过这种方式，我实现了关卡逻辑的模块化，使得每个关卡可以独立地进行开发和测试，同时也方便了后续的关卡扩展和修改，使得即使有 7 个主要关卡和 1 个无尽模式的情况下，整体类的大小也不会过大，十分有助于代码的开发流程。

游戏的核心规则逻辑则是定义在 core 包中的 GameLogic 类中。这个类负责处理游戏的主要规则和逻辑，比如玩家的移动、攻击、技能使用、怪物的行为模式、关卡的进程等。通过集中管理游戏逻辑，我能够更好地控制游戏的整体流程和规则，使得游戏的开发和调试变得更加高效和便捷。并且，我在 GameLogic 类中定义的逻辑尽可能使用组件的方式进行处理，而不是直接操纵实体类，这大大扩展了代码的复用性，使得同样的游戏逻辑，例如玩家移动，玩家碰撞检测等，可以被 GameScene、OnlineGameScene 以及 OfflineGameScene 等多个页面共享使用，避免了代码的重复编写，提高了代码的可维护性。

之后，如网络游戏的扩展、录像回放功能的实现、对话功能的添加等，都是通过新增包的方式进行。则样做能够在不影响原有代码结构的情况下，快速地实现新的功能模块，提高了代码的可扩展性和维护性。目前来看，整体的代码结构清晰，模块划分明确，各个功能模块之间的耦合度较低，并且主要游戏类的大小也没有过于臃肿，保证在了一个合理的范围内。

总的来说，通过采用面向对象编程的设计理念，并结合组件-实体系统 (ECS) 设计模式，我实现了游戏代码的高内聚低耦合，使得各个模块之间的关系清晰，功能划分明确。这不仅提高了代码的可维护性和可扩展性，也大大

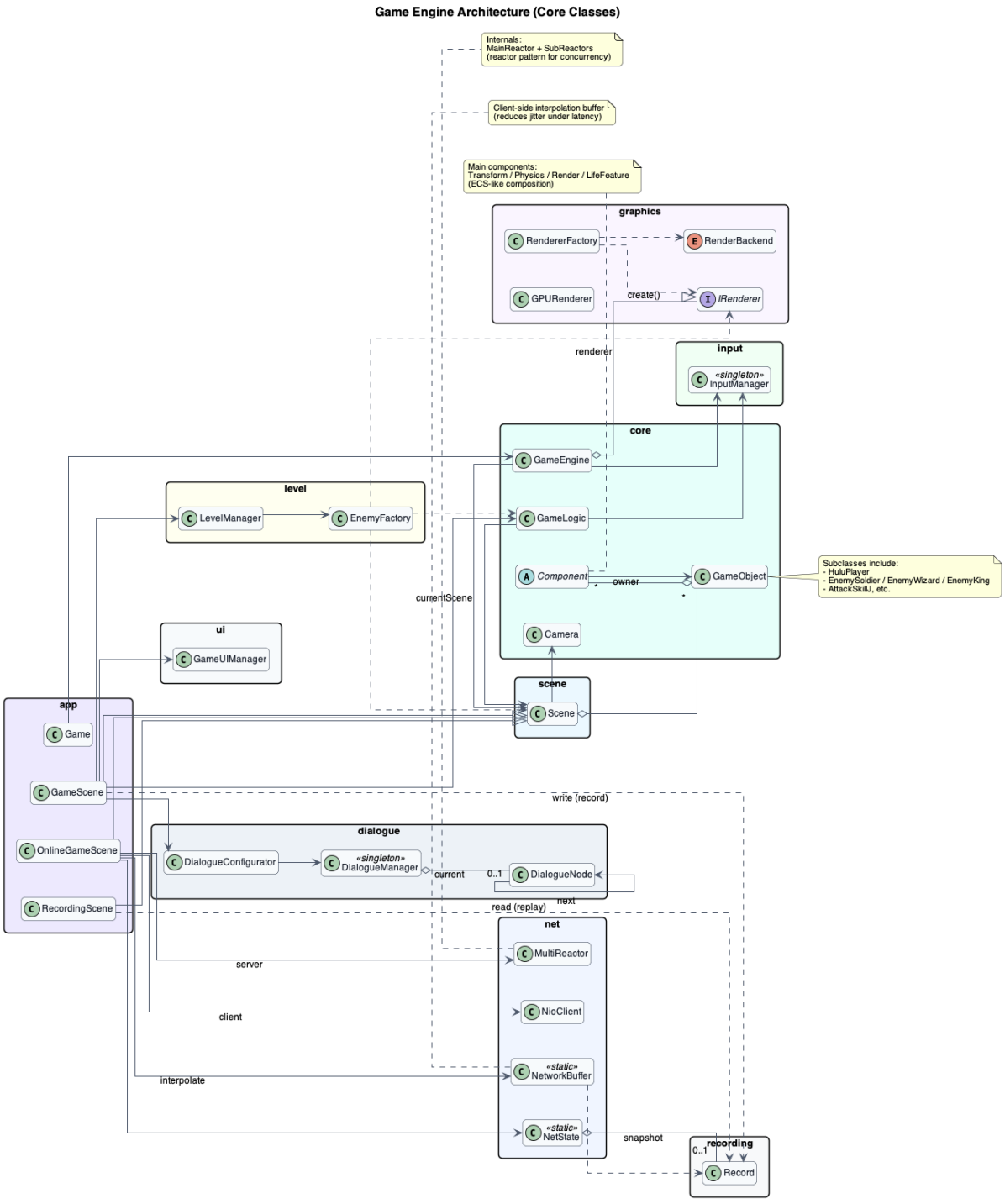

图 1 主要类图

Figure 1 主要类图示意图

提升了开发效率，使得我能够更专注于游戏功能的实现和优化，而不是陷入复杂的代码结构中。

3 技术问题

就通信效率而言，我首先在网络对战模式中采用了客户端-服务器 (Client-Server) 架构设计。服务器端负责处理游戏的核心逻辑和状态管理，而客户端则负责用户输入和界面渲染。通过这种方式，我能够集中管理游戏状态，减少了客户端之间的直接通信，从而提高了通信效率。此外，我还定义了一套新的一套 Reactor 模式来处理服务端的网络事务，定义在了 Reactor 类中，以提升在大型网络通信中的效率。服务器端中定义了两种类型的 Reacotor，分别是 MainReactor 和 SubReactor。MainReactor 负责监听客户端的连接请求，并将连接分发给不同的 SubReactor 进行处理，而每个 SubReactor 则负责处理一定数量的网络读写操作。通过这种方式，实现了在大型网络通信过程中的性能优化处理，为日后进行更大规模的网络对战模式打下了基础。

就并发控制而言，我在 GameLogic 类中定义了主要的并发机制，通过分析发现，主要的游戏性能开销主要来自算法复杂度为 n^2 的碰撞检测机制和物理系统更新机制等部分。就碰撞检测部分而言，我将其设计为了一个独立的线程，使用 ExecutorService 创建线程池，并按批次分配给不同线程，由每个线程独立根据 updateSinglePhysis 方法独立更新自己的组件。这样做不仅提高了碰撞检测的效率，还避免了线程频繁创建和销毁带来的性能开销。并且，我对方法加上了 synchronized 关键字，以确保在多线程环境下对共享资源的访问是安全的，避免了多个葫芦娃或敌人在同一时刻移动到同一位置的冲突情况。

就输入输出而言，我主要使用了 Google 的 Gson 库来处理游戏数据的序列化和反序列化。Gson 库提供了高效且易用的 JSON 处理功能，使得我能够方便地将葫芦娃和敌人对象移动的步骤和渲染方式定义为记录类，再由记录类转换为 JSON 格式进行存储和传输，在读取记录的过程中，将原本定义好的 JSON 数据反序列化为记录类对象即可。并且，我借鉴了 java 核心 Object 类中定义 toString() 方法的例子，将游戏过程的核心记录逻辑都定义在 GameObject 中的，由 GameObject 直接记录并返回记录的 record 对象。这样做使得记录逻辑能够被很好的泛化和使用，在之后我添加新的敌人类和技能类的过程中，都不需要为它们写额外的记录逻辑，从而大大提高了代码的复用性和可维护性。并且，在网络对战模式中，客户端和服务器之间的数据交换也复用了录像功能的 JSON 格式，从而避免了重复定义数据交换格式的麻烦，提高了代码的简洁性和一致性。

4 工程问题

在本游戏的开发过程中，我遇到的主要工程问题其实并不是代码是否正确上，而是集中于在代码规模增长的背景下如何保持代码的可扩展、可复用和可维护上。为了解决这些问题，我采用了多种设计方法和工程方法来提高开发效率和代码质量。

就例如，在游戏的核心，单人游戏模式设计类 GameScene 的开发上，我进行了多次的代码重构，以保证代码的简洁，可用。最初，游戏的所有对象都以内部类的形式定义在 GameScene 类中，随着游戏功能的增加，GameScene 类变得越来越臃肿，为了解决这个问题，我将游戏中的主要实体类都转变成了 game 包下的外部类。这更符合面向对象方法的设计理念，同时也使得我能够对每一个类进行单独的代码测试。此外，原先的血条、技能条渲染、对话逻辑等功能也定义在 GameScene 类中，为了提高代码的模块化程度，我也将这些功能进行了解耦，遵循实体-组件 (Entity-Component) 设计模式，将部分功能模块抽象成了独立的组件类，又将部分渲染逻辑放在了 ui 包中，大大优化了 GameScene 类的代码结构，使得代码更加清晰易懂。

此外，在游戏对象的定义上，我也采用了工厂模式来管理敌人对象的生成。通过定义 EnemyFactory 类，我能够集中管理敌人的生成逻辑，使得我可以轻松地添加新的敌人类型，而不需要修改现有的代码。这不仅提高了代码的可扩展性，也使得我能够更好地控制敌人的生成过程，帮助我实现游戏的关卡设计。

加之，为了确保代码的质量，我也为所有核心模块添加了 JUnit 单元测试，并以 Mock 渲染器隔离图形依赖，使测试可在普通环境下运行。例如：LevelManagerTest 验证关卡生成逻辑，GameObjectTest/各组件测试验证组件系统的基本行为，NioServerTest 与 NetworkPerformanceTest 覆盖连接、消息处理与性能边界。通过定义的测试体系，关键模块的修改能更早暴露问题，从而提高代码的准确度。但是，由于很多类的定义仅仅是数据类，不太涉及

具体的方法逻辑，并且很多类涉及渲染逻辑，所以并没有为所有类都添加测试代码，致使测试用例的覆盖率并不是很高，但总体上还是保证了代码的质量。具体的测试覆盖率如图 2 所示：

最后，我也用到了很多插件和工具来辅助开发过程。例如，我使用了 Maven 来管理项目的依赖和构建过程，使用 VSCode 作为主要的代码编辑器，并安装了多个有用的插件来提高开发效率，比如 Java Extension Pack、Checkstyle for Java 等等。并且，我也用到了 GitHub Copilot 来辅助代码的编写，AI 不仅帮助我定义了很多重复性的代码，还帮我优化了一些复杂的算法逻辑，大大提高了我的开发效率。此外，值得一提的是，目前项目中的几乎所有美术图片也都是由 gemini nano banana pro 生成的，这大大克服了单人游戏开发中美术资源匮乏的问题，使得我能够专注于游戏的核心功能开发，而不需要花费大量时间在美术的制作上。

5 课程感言

作为信息管理学院的学生，我平时接触到的编程课程并不多，之前也只有过一些 python 和 C 语言的基础编程经验，并且，我之前的学习和实践方向主要聚焦于数据科学上，以至于我几乎没有任何的面向对象编程基础。所以，这门课对我的挑战还是相当之大的，并发编程、网络通信等内容对我来说几乎都是从零开始。不过，也正因如此，我在这门课中也有了相当大的收获。在课堂中，我学会了 Java 语言的基本语法和面向对象编程的设计理念，并且通过完成这个游戏项目，我也是第一次完整地开发了一个较为复杂的软件项目，对软件的架构设计、模块划分、代码管理等方面都有了更深刻的理解。并且，我还学到了关于 IO、多线程编程、网络通信等方面的知识，加深了我对计算机科学的理解。

对于课程形式和内容方面，我觉得整体设计还是相当不错的。特别是游戏项目的设计，相对于传统的课程作业而言，更具有趣味性和实际意义，激发了我编程的兴趣和热情。但是，我也有一点小建议。例如，我个人仍为在讲本地代码交互、注解等章节时速度有点过快，内容过于密集，导致我无法完全理解。此外，我也希望课程在 java 的大数据处理、网络后端开发等方面有更深入的讲解和实践机会，因为这些内容对我未来的学习和职业发展可能会更有帮助。

总的来说，这门课对我来说是一次宝贵的学习经历。在此我感谢老师和助教的辛勤付出和耐心指导，使我能够顺利完成这个具有挑战性的项目，并从中学到了很多宝贵的知识和技能。

b 站视频请看：

Developing a Java Game from Scratch

罗一逖 231820309[1*]

1. *Nanjing University School of Information Management, Jiangsu Nanjing 210023*
* Corresponding author. E-mail: 231820309@smail.nju.edu.cn

Abstract 我开发的是一个葫芦娃闯关对战的小游戏。一共有三种主要的游戏模式：单人剧情模式、多人本地对战模式以及网络对战模式。单人剧情模式是游戏开发的核心，而多人本地对战和网络对战模式则是为了增加游戏的趣味性和可玩性，同时也是对 Java 网络开发能力的测试与实现。

Keywords Java Game Development; ECS Pattern; Roguelike; Reactor;