中国科学 : 信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

李帮平

1. 南京大学, 南京 210033

E-mail: 191820091@smail.nju.edu.cn

摘要 在本学期中，使用 Java 语言从零开始完成了一个小游戏。充分利用了 Java 的语言特性来实现了这样一个游戏，在代码中使用了泛型使得代码能够适用于更加复杂的场景，使用到了 Java 提供的线程技术来完成游戏中人物的诸多行为，同时用到了网络编程使得游戏从一个单机游戏变成一个可以联网游戏。文章将介绍在工程中是怎样使用这些特性，并列出编码过程中遇到的问题和解决方案。

关键词 Java, 游戏, 多线程, 网络编程

1 开发目标

1.1 灵感来源

此次游戏设计除开在代码层面上的硬性要求，对于游戏成品的要求只有“对战”，由于要求过于开放性，所以一开始各种各样的想法和游戏模式就从脑子里面冒了出来。最开始想到的当然就是去复刻某一个 4399 上面的经典游戏，类似于坦克大战、飞机大战等等，又或者是“回顾童年”将以前玩过的某些经典游戏进行重现，但是最后发现做出这样一个游戏缺乏了最原始的东西——创意。只不过是在前人已经给出了游戏思路之后进行的代码复现，我想要的并不是这样的一个东西，于是我开始摒弃“借鉴”的想法，从室友那里去寻找灵感，遗憾的是他们的想法过于天马行空，以我的现在的能力很难将某些重要的细节进行实现（比如需要使用鼠标来将生物进行选中），一旦失去了这些细节那么这样一个游戏就没有了“灵魂”。而最终我是在室友玩英雄联盟的时候，看到“传送”这一脱离连续性的移动方法，找到了灵感，于是在人物的移动方法上进行了改进，除了有传统的利用“WASD”来进行上下左右移动，还新增了利用玩家留下的某样东西来实现玩家进行瞬间移动的功能，并以此为核心来构思出了其他的东西。

1.2 代码总体目标

游戏从总体上来说是玩家通过发射子弹来打败另一方或多方玩家，并在对战途中小心不被玩家以外的生物袭击，直至游戏中只剩一名玩家时结束游戏。从比较细节的方面来说，玩家的攻击方式非常传统简单，有朝着某一方向发射线性范围的“长距离子弹”，还有拥有向四周进行发射子弹的范围攻击，并且再进行了攻击之后，必须要等到该次攻击效果完全消失之后才能够发动下一次攻击。但是对于移动方式进行了创新，玩家可以通过按下“瞬移”按键，来实现瞬间移动到“长距离子弹”所在位置，来实现对于某一个玩家进行“突击”的效果。除此之外，在地图的设计上面，增加陷阱功能也就是

“水区”，当玩家不慎掉入之后就会直接死亡，“水区”几乎将左右地图分为两个部分，因此除了特定的位置通过之外，还可以使用“瞬移”的功能直接从一边穿越到另一边，为的就是增加游戏的操作性。

2 设计理念

代码一共分为四个部分：核心框架、UI 显示、功能拓展、启动，下面将分别介绍这四个部分。

2.1 核心框架

核心框架是指游戏内部的逻辑实现，包括了生物的移动逻辑实现、生物的攻击方法和攻击效果实现，注重的是游戏的中的生物的自主行为和生物与生物之间的交互行为。而在其中的更加基础的部分，则是使用了 jw04 中的“World-Tile-Thing”基本结构，简单说就是 World 容纳了多个 Tile，而每个 Tile 将空间分为上中下三个区域，在三个区域中可以容纳不同的 Thing，所以对于 Tile 上空间的争夺其实也就是对于 Tile 上同一位置的争夺。使用这样一个基本结构的好处就在于比较好的模拟了现实，以至于从面向对象的角度去编写代码的时候，会更加容易和高可扩展性。在此之上，就是以继承为主体，以 Thing 为父类，实现了 Ceature 和 NonLiving 两个子类，并且在 Creature 上继续实现子类 Gourd 和 Monster，在 NonLiving 上实现 Stone 等等子类，形成一套继承体系。使用这样的一套继承体系，可以在很大程度上减少代码的书写，避免了大量重复的代码，使得代码更加简洁。除此之外，在这样一套继承体系之下，我们可以很好的利用多态的特性来提高代码的可读性，具体来说，在后面实现存档功能的时候，并不需要对每一个类具体是什么来进行判断，而是统一的当成是 Thing 这个父类来进行操作，极大的避免了由于生物和非生物过多而带来的判断上的巨大的麻烦。

2.2 UI 框架

UI 显示是指将游戏的过程进行可视化，如果不进行可视化，那么游戏的过程无非等着电脑处理完一堆二进制数据，无趣至极。和前面的核心框架进行比较的话，框架下代码的运作，就是微生物的运动，是看不见的，UI 显示就是利用显微镜将其暴露出来。所以一个漂亮的 UI 会在很大程度上影响一个游戏的游玩体验。为了能够尽可能地将 UI 和核心代码分来，使得两者的耦合度降低，我选择了创建一个接口为 Displayable，UI 框架会将实现了 Display 接口的类进行展示。在这里并没有直接将 Thing 的继承体系中的子类放到 UI 框架中，因为这样的话，这样一个 UI 框架几乎和 Thing 这样一个继承体系进行绑定了，会出现一个非常高的耦合，后期想要进行修改难度非常大。引入了 Displayable 接口之后，UI 框架就是面向接口中提供的方法编程，任何想要被进行展示的类，都可以通过实现接口来达到目的。而 Thing 就实现了这样一个接口，响应的其整个继承体系都能够被进行展示。从原本的 UI 框架依赖于 Thing 的具体实现，变成了 UI 和 Thing 通过接口进行耦合度非常低的连接，可拓展性有了非常大的提高。前面只是说明了引入 Displayable 接口的必要性和优点，下面介绍 UI 框架的基本结构。为了能够尽可能的降低 Thing 对于 UI 框架的依赖性，我引入了一个类 ItemImage，这个类将图片进行包装，并提供相应的操作方法，同时在在这个类中有多个静态的成员变量，每一个成员变量都是 ItemImage 类的一个具体对象。此时 Thing 在进行构造的时候，就不需要考虑自己去封装一个图片，并以此达到 Displayable 接口的某些要求，而是直接访问 ItemImage 类的静态成员变量来完成初始化，然后再依赖于其提供的某些方法来实现 Displayable 接口中的方法，从而直接在 Thing 类就将实现方法定义好，而不需要在每一个类中去进行具体实现，再一次提高了可拓展性。UI 框架的运行流程是，通过构造函数来初始化一个缓冲区，然后由外界调用方法将需要进行展示的 Displayable 的实

现类对象放入缓冲区的某一个特定位置，最后通过 paint 函数将缓冲区数据展示到频幕上。

2.3 功能拓展

功能拓展我主要分为三个部分：接受外界控制（键盘）、提供存档功能、提供网络对战功能。这些功能都是在前面的核心框架下进行的拓展，与 UI 基本没有关系。

2.3.1 接受键盘控制

接受外界控制是最容易实现的功能，绑定键盘监控事件，接收到事件重新将事件包装成另一个自定义的类，传递给 World 类中，交由核心框架中的某个函数进行处理，并产生反馈。

2.3.2 提供存档功能

提供存档功能也就是当游戏退出的时候，将游戏当前的画面和各个生物的状态进行记录，并写入到文件中以便于下一次直接进行读取然后恢复游戏进度。在前面简单提到了存档功能是依赖于 Thing 的一连串继承关系来实现的，具体来说我使用的是让 Thing 这个类去实现 Java 提供的 Serializable 接口，然后利用 Java 提供的序列化机制将其中我们感兴趣的东西保留下来并写入文件，而对于读档则是存档的逆过程，不同的是读取文件并不能直接创建线程，因此我们需要手动的为上次强制停下来的线程主动创建线程。由于存档和读档是一对匹配的过程，于是我们将这两个过程包装成一个类，并提供相应的存档和读档方法，使其封装性更好。

2.3.3 实现网络对战

提供网络对战功能的本质就是数据的传输，而更加实际一点的就是对战画面的传输。为了保证运行时的一致性，我们将游戏的功能部分全部放在服务器端进行运行，此时我们除了传输对战画面，还应该将客户端的键盘控制信号传输给服务器端，并交由服务器端进行处理。

2.4 启动

所谓启动也就是为整个游戏提供一个 main 函数入口，从这里来讲前面三个模块进行整合并开始运行。在上面核心框架使用了一个 WorldScreen 类来间接访问，这样一个类实现了 Screen 接口，使得其不仅可以响应键盘操作，还可以将核心框架和 UI 框架进行连接起来。所以在主函数需要创建 WorldScreen 对象来启动游戏内部逻辑；创建 ImagePanel 对象，用来进行 UI 展示；在联机模式下进行游戏还应该创建 Server/Client 对象来实现游戏数据的传输。此时就将前三个模块整合到了一起。

3 技术问题

在游戏实现过程中遇到了诸多的问题，主要分为四个方面：通信效率、客户端画面显示、并发控制、UI 刷新。

3.1 通信效率

在前面的网络对战中介绍过，传输的数据主要是游戏的画面，而游戏的画面则是很多张图片，在最一开始我是直接将图片进行传输，最后的效果非常糟糕，具体说就是客户端一秒钟刷新一两次，延迟异常严重。分析其原因，每次网络传输的数据接近 3M，但是这只是画面刷新一帧所需要传输的数据量，所以当刷新率稍微大一点点，就会出现严重的掉帧现象。为了解决这一问题，我借鉴缓冲的思

想，在客户端提前存储好需要进行展示的数据，服务器端就只需要发送一些指令数据，用来指示客户端什么图片需要在什么位置进行展示即可，使用一个 byte 的数据就能够完成一个位置的图片的指定，此时粗略计算发现传数据从一次 3M 下降到了 3K，在很大程度上提高了通信效率。

3.2 客户端画面显示

虽然通信效率进行了提高，但是在客户端却会产生一定的网络延迟，并且随着时间的累积，延迟会越来越高，运行 1 分钟延迟可以达到 3s，这样的延迟是无法接受的；或者是客户端刷新过快直接导致屏幕出现严重的闪烁现象。具体来说，原本的版本是将服务器端的一帧数据全部放入到客户端的队列中，客户端从队列中取出数据进行展示，但是由于网络的不稳定，导致客户端可能会在短时间去除多个数据进行刷新出现“画面加速”，甚至是画面刷新过快导致屏幕出现闪烁现象。虽然想过通过控制客户端取数据的事件来解决，但是带来的问题就是前面提到的高网络延迟。所谓“鱼和熊掌不可兼得”，前面的问题来源于客户端接受了服务器端的全部数据，然后放到队列中进行处理，于是我选择了丢弃数据，来确保画面的低延迟和高刷新率。具体来说就是每次服务器端都会将数据放入缓冲区，但是缓冲区只能够容纳一帧数据的大小，前面的数据会覆盖掉前面的数据，于是乎客户端每隔固定时间去取出数据的时候，得到的都是最新数据，避免了前面的高延迟现象，同时控制客户端取数据的频率（刷新率），就能够减少甚至避免画面闪烁的问题。但是也会存在一些小问题，一是客户端和服务器端数据并非是完全“同步”，虽然在感官上面，数据几乎一致；二是客户端可能会在两次临近刷新使用的是同样的数据，因为网络延迟服务器端数据还没有完全传输完毕，但是无伤大雅。

3.3 并发控制

并发控制在很多地方都需要进行考虑，而最主要的并发控制体现在 Creature 移动、Thing 碰撞、客户端获取缓冲区数据。

3.3.1 Creature 移动

主要考虑的问题就是，两个生物是两个线程，他们在某一个时间点对 Tile 的某一个位置进行观察之后，同时移动到这个地方了，造成了一个空间存在两个 Thing 这种奇特的事情。所以为了避免出现这种情况，需要对“移动”这一事件进行控制，最简单的控制当然是直接利用整个 world 作为锁，但是出现的情况就是一个物体进行移动的时候，其他物体均无法移动，这种锁的粒度太大。于是从更加细节的方向进行考虑，发生冲突是因为进入了同一个 Tile，于是我将下一个需要进入的 Tile 作为锁，也就是在一个物体开始朝着某一个 Tile 进行移动的时候，其他也想要移动进去的同一个 Tile 的物体必须维持等待状态，直到获得锁为止，此时已经避免了位置冲突问题。

3.3.2 Thing 碰撞

在我的代码逻辑中，当物体 A 碰撞到了物体 B 的时候，A 会调用“碰撞别人”方法，而 B 会调用“被别人碰撞”方法，因此多个线程就会存在 A 碰撞 B，B 又去碰撞 C，B 此时会在不同的线程中被调回不同方法，导致了碰撞冲突问题的出现。解决的方法就是对这两个方法进行加锁处理，当在一个线程中处理完 B 的某种行为之后，再去处理另一种结果，达到了碰撞的有序性。

3.3.3 客户端获取缓冲区数据

在客户端和服务器端进行通信的过程，存在的最大问题就是传输的画面数据相对于其他数据过于庞大了，导致如果客户端需要空等待服务器端，为了避免这种情况，我使用了缓冲区来对传输数据进

行暂存。但是在客户端的 UI 刷新也是重新创建了一个线程，这个线程会每隔固定时间去缓冲区拿数据并解析显示在屏幕上，如果此时缓冲区正在写入数据，那么 UI 拿到的将会是一个错误的数据，为了避免这种情况，我对缓冲区的操作进行了加锁操作，每次都只允许对缓冲区进行一种操作，避免了数据错误。

3.4 UI 刷新

UI 刷新采用的最主要的机制就是将大量的小图片（1000+ 张）显示在屏幕上，所以会频繁地在频幕上打印图片导致屏幕出现闪烁的现象，所以采用“画布”这样的缓冲技术，具体来说就是将所有的小图片全部都预先加载到一个固定大小的画布上，最后直接将画布中的内容一次性的写到屏幕上面，不仅可以在一定程度上消除闪烁，还能够减少 IO 次数，提高效率。

4 工程问题

在本次工程中使用了 Maven 进行项目管理，并使用单元测试对于代码中的部分方法进行了测试，使得其更具可靠性和安全。同时使用 Maven 来完成了依赖管理，不需要自己进行复杂的其他操作，方便管理。

5 游戏展望

在网络通信方面，虽然使用到了 Channel、Buffer、Selector 技术，但是却并不是最好的，如果能够将其改变成使用 Reactor 设计模式来进行设计的话，在线程数比较多的时候，其效果将会是非常好的。服务器端将数据传输给客户端的时候，虽然已经将原先的 Mb 量级降低到了 Kb 量级，实现了比较好的传输效率，但是分析具体的游戏场景就知道，在一段时间内只有很少的物体进行了很短的移动，所以会传输大量的静止状态下的物体，这是没有必要的，可以考虑将每次传输数据改为“增量式”的传输方法，只传输那些状态发生改变的物体，这样可以进一步降低传输的数据量。对于 AI 的攻击方式，只是做了最最基本的考虑，也就是在玩家出现在了某一个区域之后就会发起攻击，但是对于地形的判定等等还没有考虑很充分，导致 AI 看起来并不是那么的智能甚至有点傻，所以考虑以后将 AI 的行为调整到更加智能，增加游戏的趣味性。

6 课程感言

先说结论，我认为这门课非常好，我不仅学到了东西甚至还收获了快乐。这门课我是从跨院系选修课选到的，当初想选这门课单纯是因为以前学过一点 Java，以为这门课学起来会比较轻松，所以一开始就是冲着学分来的，但是上了三节课之后，我的这种想法完全被改变了。首先，老师的讲课风格非常不同于以前的老师。尽量少讲书上有的东西，让我们自己去看书去主动的进行学习，自己去了解那些最基本的语言层面上的东西，自己去学习 API 自己看文档，可以培养自学能力。所以当课堂上讲的东西都尽量和书上简单的知识不进行重叠了之后，可以讲的东西就变成了编程的思维，思考的角度。印象最深的就是面向对象这一块内容，“代码模拟现实”已经成了我编码时首要考虑的东西，虽然写出来的代码还是很难看，但是思考的角度已经发生了非常大的变化，以前真就如老师所说那般，写一个类的时候考虑的并不是如何模拟现实，而是将过程抽象成一个类。其次，老师的作业风格我非常喜欢。在大学里面也上过许多课程了，也做过很多次作业进行过很多次实验了，但是以往的作业往往

只局限于当前知识，没有将知识进行串起来导致学的东西东一块西一块很难成一个体系，理解自然也不会太深。Java 的作业我认为是我做过最好的作业，并不是说这样的作业有多难对知识考察的范围达到了细致全面的地步，而是这样几次作业前后是可以相互关联起来的，知识的连贯性得到了保证。每一次的作业都是对于新知识的学习和旧知识的复习，并建立了关联性，非常好。最后，跳出课程本身而是回到老师身上，老师是我见过最有趣的。和其他古板的老师不同，Java 老师不仅不显得古板，反而透露出一股很潮的感觉，以至于在线下上课之前我一直以为年龄在 30 左右，平时的老师的各种发言都能够给我一种亲切的感觉，不至于会有其他教师一发言就具有很强的距离感，让我难以接近。

Developing a Java Game from Scratch

Li Bangping

1. *Nanjing University, Nanjing*210033
E-mail: 191820091@smail.nju.edu.cn

Abstract In this semester, I finished a small game from scratch using java language. It makes full use of the language characteristics of Java to realize such a game. Generics are used in the code to make the code applicable to more complex scenes. The thread technology provided by Java is used to complete many behaviors of characters in the game. At the same time, network programming is used to make the game from a stand-alone game to an online game. This article will introduce how to use these features in engineering, and list the problems and solutions encountered in the coding process.

Keywords Java, Game, Multithreading, Network Programming