中国科学 : 信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

卢润邦[1]

1. 南京大学, 南京210000

E-mail: lurunbang955@gmail.com

摘要 基于JAVA对上个世纪60年代的吃豆人游戏进行二次开发，并加入了存档和联机对战功能

关键词 JAVA，OOP，JACKSON,Non-Blocking Server,Junit

1 开发目标

1.1 游戏介绍

单机模式下玩家操纵Pacman在躲避Ghost追捕的同时吃掉尽可能多的金币到达关底，也可以吃掉大力丸后进行反扑，吃掉Ghost以通关。

联机模式下，入侵者可以假扮四种Ghost其中之一，对pacman展开围追堵截。

1.2 灵感来源

整体灵感来自上个世纪60年代风靡全球的吃豆人游戏，四个Ghost的AI均是启发自岩谷彻设计的四种ai，Blinky会一直追逐敌人，Pinky会潜伏在Pacman前进的方向，Clyde是游戏中的气氛组，它起初会追逐主角，当靠近后又会立刻反方向逃跑，而Inky则与世无争，盘旋在地图一角。游戏同时吸收了吃豆人中一个很重要的设计理念，**给玩家喘息的机会，而不是一味增加压力**,所以每过一段时间，Ghost会放弃追逐。

联机的灵感来自2021的双满分游戏《死亡循环》，入侵者可以扮演四个Ghost其中之一，通过模仿原生Ghost的行为迷惑Pacman玩家，比如可以扮演Clyde在接近主角时改变策略，不再转身逃跑进而攻击主角。

2 设计理念

2.1 Creature

这一块主要使用了代理，使用CreatureAI操控creature。委托的一个好处就是解耦，即事件源并不需要知道到底是哪个类的方法处理事件，简单地说，A产生事件，传给了委托者B，，B再传给C来实现事件的处理。委托解耦简单说就是把产生事件的代码和处理事件的代码通过委托者给隔离开来。放在这里，就是当我的Creature被创建出来，我并不知道他是Blinky还是Coin，所以他的route(寻路),就交

给ai处理。再进一步，不同的Ghost也有不同的寻路方式，惊吓时会变成随机游走，也可以转交出控制权，听从一个入侵者指挥，所以，我再把router抽象出来，这样随时可以很方便的更换Routing Strategy，这里的设计我生成uml图如下。

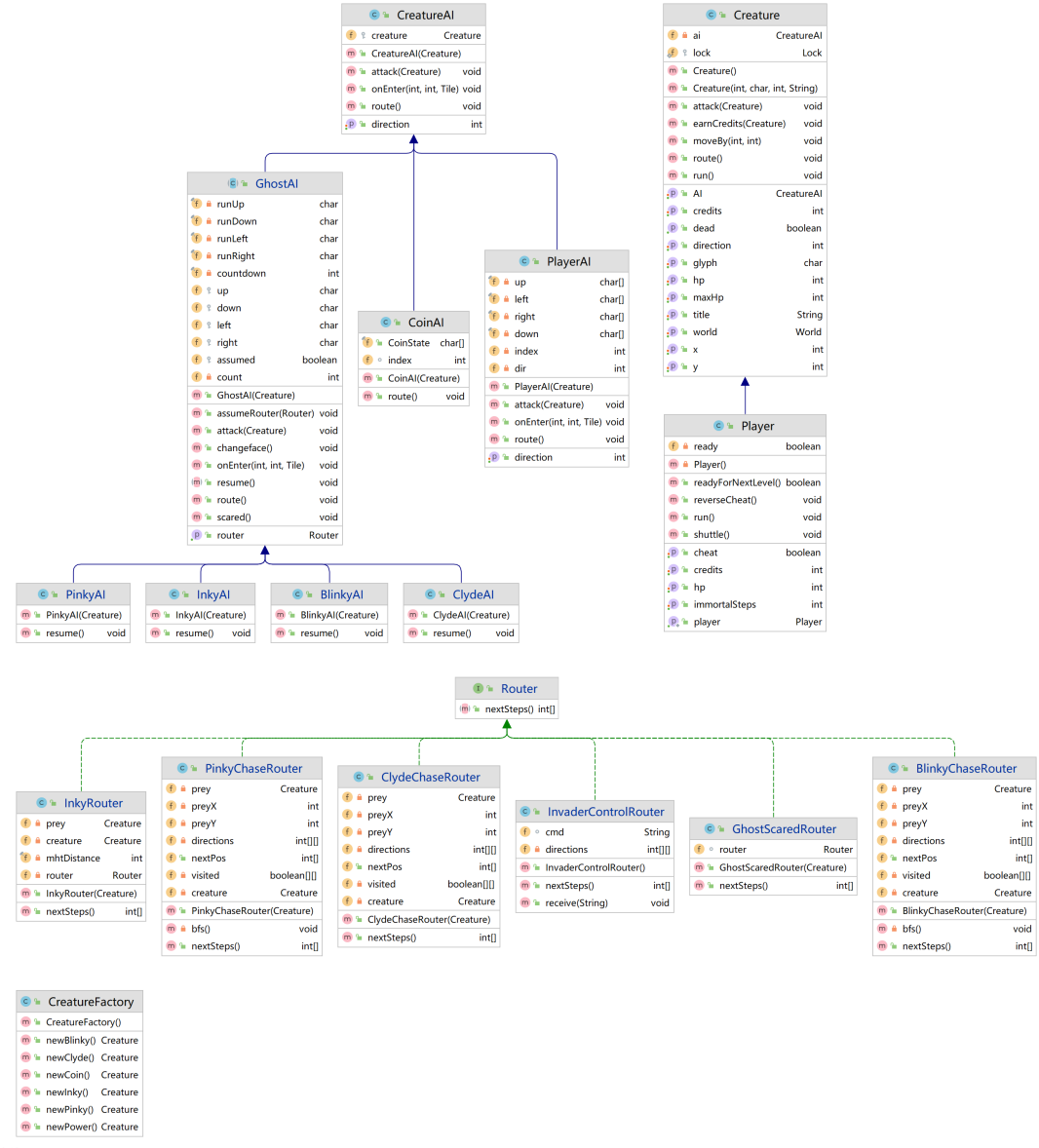


图 1　Creature uml

Figure 1

2.2 其他模块

2.2.1 世界生成

首先通过Prim算法生成01的简单迷宫并打出大面积的洞穴，将01数组交给WorldBuilder，建造出世界。这样的好处是可以用简单01迷宫的生成来初始化整个world的初步格局，并在world初步初始化后可以很方便的生成caves，做到解耦。

2.2.2 存档生成

借助jackson第三方库，自定义serializer和deserializer，将world，creature，player存储为json，这里没有使用ObjectStream的原因是，他所生成的二进制文件，不能很好的兼容其他平台，也不能直接进行查看。加上现代工业界已经基本抛弃这个，多使用json进行存储和传输。

2.2.3 网络联机

得益于我存档生成选择使用了json，它可以很好的适应网络传输，所以我在这里设计了几个“协议”，以及两种server，Blocking和Non-Blocking的。server的具体细节以及协议设计会留到下一节技术问题详细分析。这里阐释为什么选择了两套server。Blocking的server用来刷新client的屏幕，第一次传输World，creature和player的数据，之后的每次刷新都只需要传输creatures和player的数据即可，但因为第一次传输的World数据偏大，我没有把握能很好的构建一个Non-blocking server，加上开发目的是Developing a Java Game from Scratch，我决定一点点开始写，每一步都要有反馈，所以设计了第一个server。第二套Non-Blocking server主要是考虑到，invader可能不会一直发送消息，持续的轮询是对性能的极大损耗，所以采用了Non-blocking的设计，因为每次传输的消息很少，所以不会出现并发问题。

在这套设计下，我以pacman的主机为服务器，invader的屏幕只不过是对游戏的播放，同时会抓取invader的移动操作，发送到服务器，服务器处理完之后再将新的数据发送到invader的主机，即client，client只需要将其解析为画面显示出来即可，client其本身除了解析画面和发送数据外并无别的能力。某种程度上pacman即为**胖客户**，invader即为**瘦客户**，这种CS设计并没有什么特别的优点，只是对存档生成实现的一个自然承接。

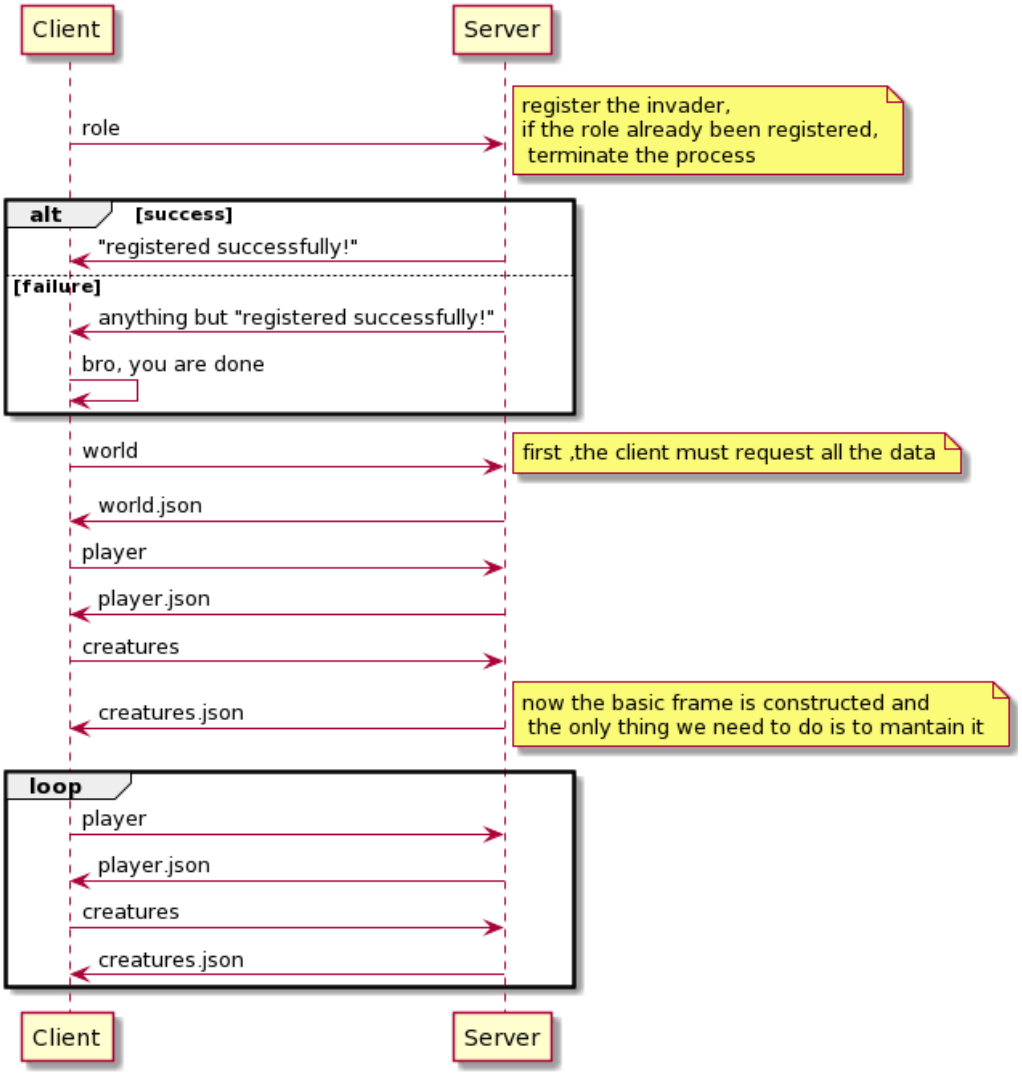
3 技术问题

3.1 通信问题

构思是想将invader当作瘦客户处理，那么就要想好服务器要传什么数据，什么时候传，客户端的数据怎么传回，服务器怎么接受，接受完如何处理。

3.1.1 Blocking Server

我单独开了一个线程专门用来刷新画面，因为他是只读的，我不担心并发问题，改变的内容这一帧没刷新上，那就下一帧再说，那么什么时候向服务器请求数据就明了了，客户端也要刷新时。第一次需要请求整个World，同时将这个用户register到服务器，以parse新输入的指令，如果该角色已经被注册则会返回失败信息，那么client就会自己结束生命，后面只需要请求Creatures 和Player 的数据即可，怎么控制请求的内容，可以发现我需要的是一个**网络协议**，这个协议图示如下



server.png

图 2　protocol uml

Figure 2

3.1.2　Non Blocking Server

当invader注册进来，便可以对这个Ghost发号施令，因为注册时我已经将被注册的ghost的router替换为parse客户输入的router，就是简单将上下左右的键值转换移动的指令，这里略去协议。这里使用了NIO的selector，因为客户几乎不会高强度输入，所以持续的轮询和中断是性能的损耗。使用seletor的好处是，只有当我有兴趣的事情，比如可读可写，准备好了的时候，我才会去处理这个事件，避免了空闲的损耗。selector 的示意图如下。

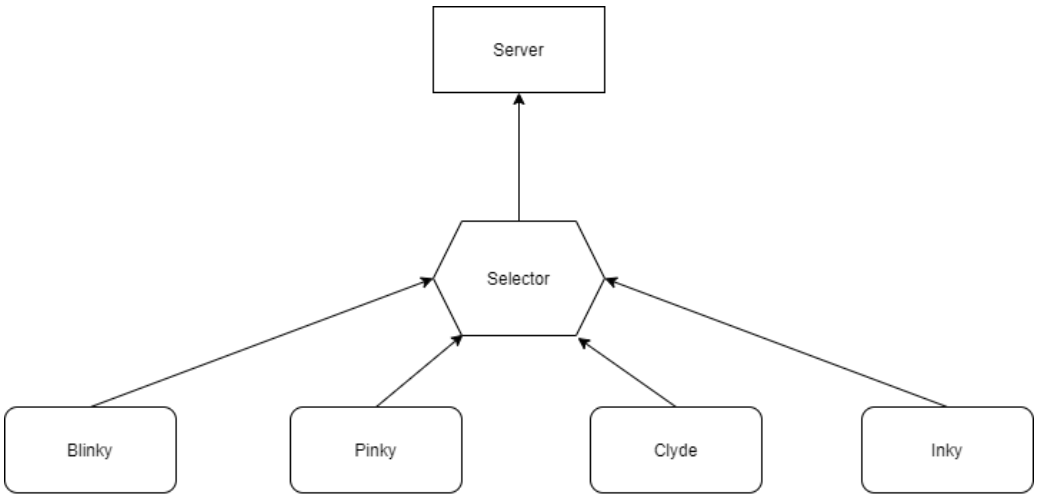


图 3 selector

Figure 3

3.2 并发控制

3.2.1 一个静态锁

两个生物不能同时踏入同一个格子，实现这个需求的方法有三，对Tile加锁，对World加锁，对Creatures加锁，第一种加锁方式粒度最小，但是实现起来比较麻烦，有上千个锁，对World粒度过大，我选择在Creature这个基类添加一个**静态锁**，静态物体并不会对锁进行申请，只有5个移动的角色会对锁进行申请和释放，在这种工作负载下，我认为这样的一个静态锁可以较好的完成他的任务。

3.2.2 Synchronized关键字

因为Cpu的Cache的存在，某些写并不能得到立刻的更新，同时因为Cpu的reordering，某些指令也不会像我们想当然的那样运行，我遇到的这个问题出现在Client向Server发送移动指令的时候，我将指令初始化为Stay，client通过指令修改这个命令，但是我发现即使client成功发送，server也接受到了，但是它依然选择执行Stay，这里是server选择读取cache里缓存的脏数据导致的，将问题函数前加上关键字Synchronized就可以让每次的读取都从disk上读取。

4 工程问题

4.1 设计模式

4.1.1 Singleton Pattern

所有Ghost的本质都是Creature+一个GhostAI，但是Player我将其单独抽出来，写成了单例模式。处于两种考虑，一是Player，即Pacman本身只有一位，将其抽象为单例模式是符合逻辑的。同时Player的Hp以及位置对于Ghost来说，是需要经常被get的，将其设计成Singleton可以很方便Ghost读取他的信息，避免了对Player的传递，构造和销毁。

4.1.2 Factory Pattern

将所有Creature的构造隐藏到CreatureFactory中这样的好处有很多，比如一个调用者，这里是World，想创建一个对象，只要知道其名称就可以了。其次是扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。最后它可以屏蔽产品的具体实现，调用者只关心产品的接口。

4.1.3 Strategy Pattern

不同的Ghost在不同时期有不同的寻路策略，所以将寻路策略抽象出来，以便随时替换。这样的好处是方便替换，方便扩展，每当我想要写一个新的寻路策略，只需要写一个新的Router然后替换掉原来的就可以了。

4.1.4 Builder Pattern

因为World的构造是基于Prim算法生成的01迷宫，没有洞穴的存在是肯定不适合作为地图的，加上还有关底的存在，同时早期版本为了丰富地图元素，还加入了岩浆元素，这样一个World的构造是一个循序渐进的过程，这样的Builder独立易扩展，并且便于控制细节，比如我最新的版本不支持岩浆元素的生成，我只需要将那个过程在build中剔除。

5 课程感言

5.1 对于面向对象的更深理解

学期初对一个奇妙bug问老师的时候，被指不懂side effect，其实当时也不是很懂OOP，直到看了老师分享的那个文章《Execution in the Kingdom of Nouns》，verb是java里地位最低的存在，它虽然做的事最多，但是却不能外泄。暴露出去的接口应该是一个动宾结构比如provide an object，或者consume an object之类的结构。所以那些个纯副作用应该被牢牢限制在类的内部，不应该被放出来。虽然我现在还不是很懂oop，但是当我想尽可能地oop的时候，我会先从命名入手，规范好命名可以更容易地帮助自己在面向对象的角度编程。这是这门课我最大的收获。

参考文献

1 http://tutorials.jenkov.com/
2 https://www.baeldung.com/