前言

我们小组的在课程设计完成过程中的注重点没有放在应用 Java 语言实现功能，或者是 Swing 框架细节等一些技术性质的问题上。我们花费了大量的时间去思考代码架构的搭建，每个地方该如何编写更契合面向对象 SOLID 原则。 所以在关于项目的介绍中我会着重介绍整个游戏的架构设计，并且以碰撞检测为例子解释这种架构设计的好处。

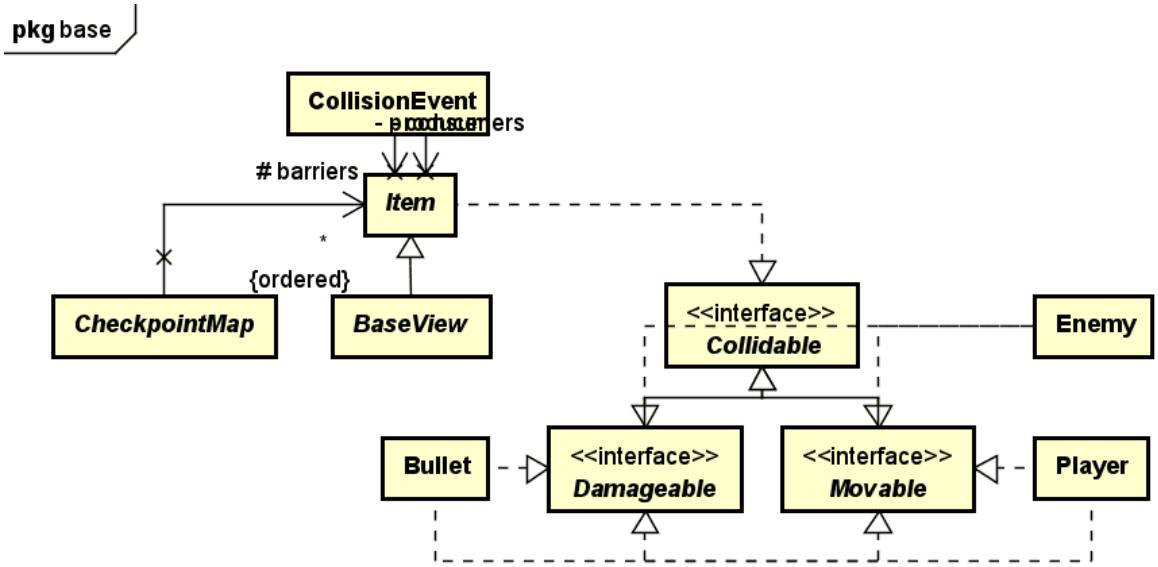
整体架构

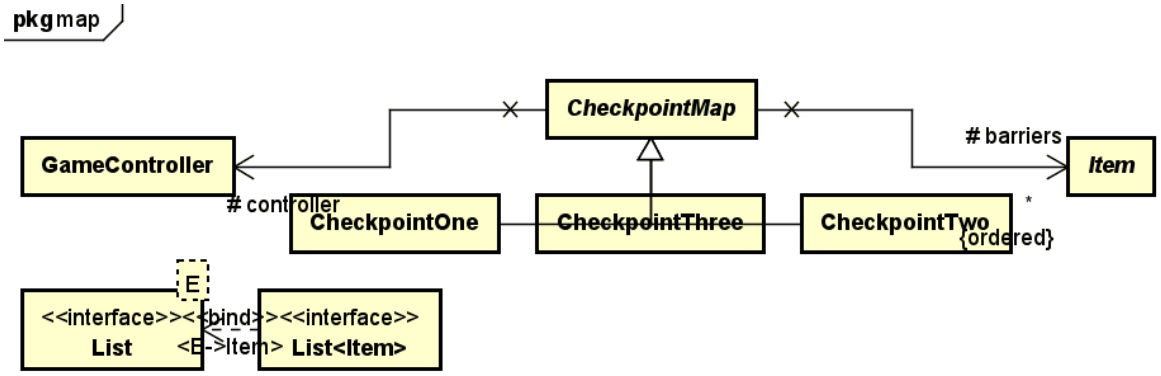
首先整体的说一下我们最终设计出的程序的基本架构思想：以 MVC 以及 MVVM 架构的思想为基础，每个页面有各自的 M，V，C 以及其他可能需要的模块，以各种`事件`作为驱动程序运行的核心。
其中 View 只包含一个ItemComponent类，负责图片绘制，不接触任何业务逻辑。
Model 或者 ViewModel 有自己的位置信息，同时持有 View 控件，View 的变化都随着 Model 和 ViewModel 的变化进行，他们起到了适配器的作用，讲数据映射成 View控件的坐标。
同时 Model，ViewModel 也处理一些业务逻辑，由于我们的架构中把几乎全部事情都抽象为了`事件`，所谓 ViewModel 进行的业务逻辑处理也多数为产生或消费事件。
Controller 负责事件的分发，程序除初始化外的所有动作，比如从键盘输入信息，items在行动时产生碰转，在我们的架构中都被抽象为了`事件`这个概念。事件产生后会被发送到Controller持有的一个事件队列中，再由 Controller 根据队列中的事件的信息，将事件发配给消费者去执行（如果是碰撞事件，生产者也会受到调度）。
同时 Controller 也持有一些游戏整体流程调度的基本信息。

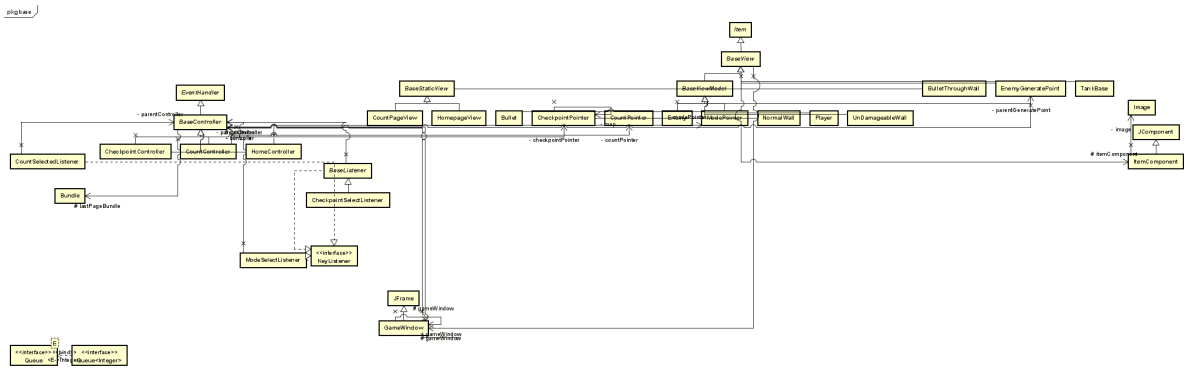
例子：子弹运行时的碰撞检测

子弹的运行逻辑是：运行在一个死循环中，如果不产生事件让子弹停止，子弹就会一直移动下去。
而碰撞事件就是让子弹停下来的时间，子弹的移动过程中包含一个检测移动时候会不会发生碰撞的函数，在子弹每次移动之前都会调用这个函数，检查下一步移动会不会发生碰撞，如果不发生则继续移动。
当发生碰撞时，就是体现我们架构的设计的思想的时候了。这时会产生一个碰撞事件，这个碰撞事件记录了生产者（子弹）以及消费者（墙壁，坦克，或者其他子弹）的信息，之后他会被传递到 Controller 的成员 CollisionHandler 的 collisionEventQueue 里面等待处理。
而这个 Handler 在程序开始时就开启了一个线程，这个线程中有一个死循环，循环中有这样的逻辑：当 collisionEventQueue 不为空时，就从队列头拿出一个事件，获得其生产者和消费者的引用，由于生产者和消费者都是实现了 Collidable 接口的，所以就再调用生产者和消费者的 onCollision 方法，告诉生产者和消费者，他们身上发生了碰转，需要他们进行处理，这样，从子弹的碰撞检测中产生的事件成功被分配给了碰撞双方进行处理。

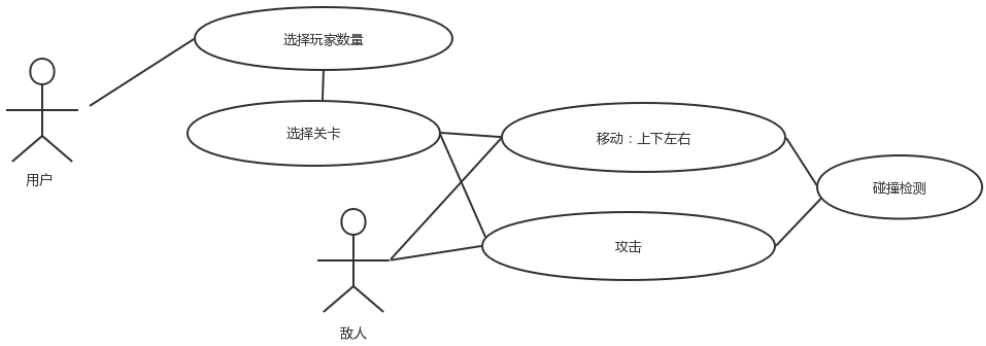
类图


pkg base
CollisionEvent
- producers
- consumers
barriers
Item
*
{ordered}
CheckpointMap
BaseView
<<interface>>
Collidable
Enemy
Bullet
<<interface>>
Damageable
<<interface>>
Movable
Player

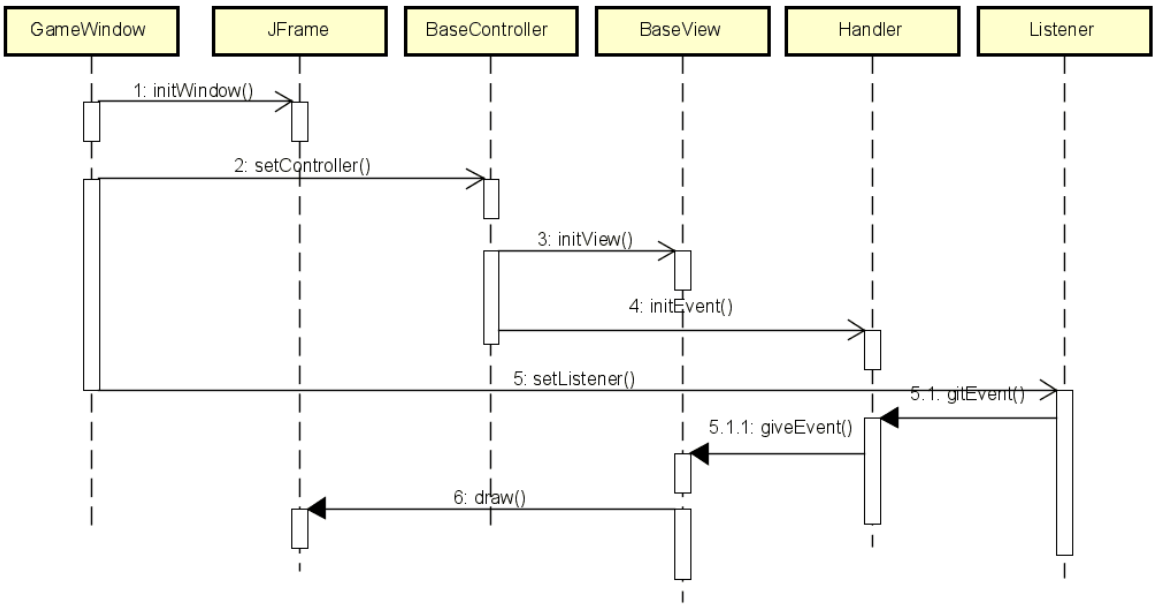

pkg map
CheckpointMap
GameController
barriers
Item
controller
CheckpointOne
CheckpointThree
CheckpointTwo
*
{ordered}
E
<<interface>>
List
<<bind>>
<E->Item>
<<interface>>
List<Item>



用例图



时序图



项目成员分工

- 孙凯一：框架搭建，后期项目合并，指导
- 何雅：墙、基地类的搭建，敌人及子弹的贴图、方向调整
- 王格格：敌人的初始化、生成点，敌人的事件处理，后期文档及ppt制作
- 王怡贤：敌人及玩家运动处理、图片处理、后期文档及ppt制作
- 韦长沙：用户选择界面、关卡选择界面、游戏结束界面、分数统计

完成情况说明

完成全部基本功能，有**实心墙**，**普通墙**，**海洋**三种障碍物类型，其中，

- 坦克会被所有障碍物阻挡，
- 子弹可以穿越海洋，
- 普通墙会被子弹打坏，实心墙不会受到伤害。

规则均依照原版游戏。除基本功能完成外，游戏有**单人**和**双人**两个模式，设立三个关卡，每个关卡的地图不同

游戏开始时让用户选择模式和关卡。游戏中按照玩家打死的敌人数量和被子弹共计的次数计分。

虽然我们小组扩展部分的功能实现的较少，但我们小组的在课程设计完成过程中的注重点没有放在应用 Java 语言实现功能，或者是 Swing 框架细节等一些技术性质的问题上。我们花费了大量的时间去思考**代码架构的搭建**，每个地方该如何编写更契合面向对象 SOLID 原则。整体以 MVC 以及 MVVM 架构的思想为基础，每个页面有各自的 M，V，C 以及其他可能需要的模块，以各种事件作为驱动程序运行的核心。