# Task Report

# Introduction

This task involves object-oriented programming for the classic game Space Invaders. In this game, the player controls a spaceship, eliminates enemies on the screen, avoids enemy attacks, and tries to score as many points as possible while staying alive. I will be extending and developing this game project, applying object-oriented programming principles within the existing code framework, and implementing new features through reasonable extensions without disrupting the existing code.

In this report, I will provide a detailed overview of my workflow, including code review and feature expansion. I will also ensure that the code runs smoothly in a specific environment configuration. Through this report, I will demonstrate how to analyze existing code and perform extensions while adhering to design and development principles of object-oriented programming.

# 代码审查

## A.面向对象编程和设计原则

在审查代码库的过程中，我注意到有多处位置都使用到了OOP原则。

1.单一职责原则 SRP

一个类只有一个引起修改变化的原因，也就是只负责一个职责。核心思想：高内聚，低耦合

假设一个类有多个功能，当修改其中一个功能的时候，可能会对其他功能造成影响。

在代码中，gameWindow类负责游戏界面的渲染，而gameEngine类则负责游戏逻辑。

例如，游戏中的渲染方法就放在gameWindow中（图1），draw()方法会随着游戏进度的更新而被调用。而游戏中敌人实体的状态更新方法update()就不必放在gamwindow中，而应放在gameEngine中（图2）。

这样一来SRP就使得类的复杂度降低，一个类负责一个职责，比负责多项职责要简单，并且

代码的可读性提高了，也方便以后的维护。

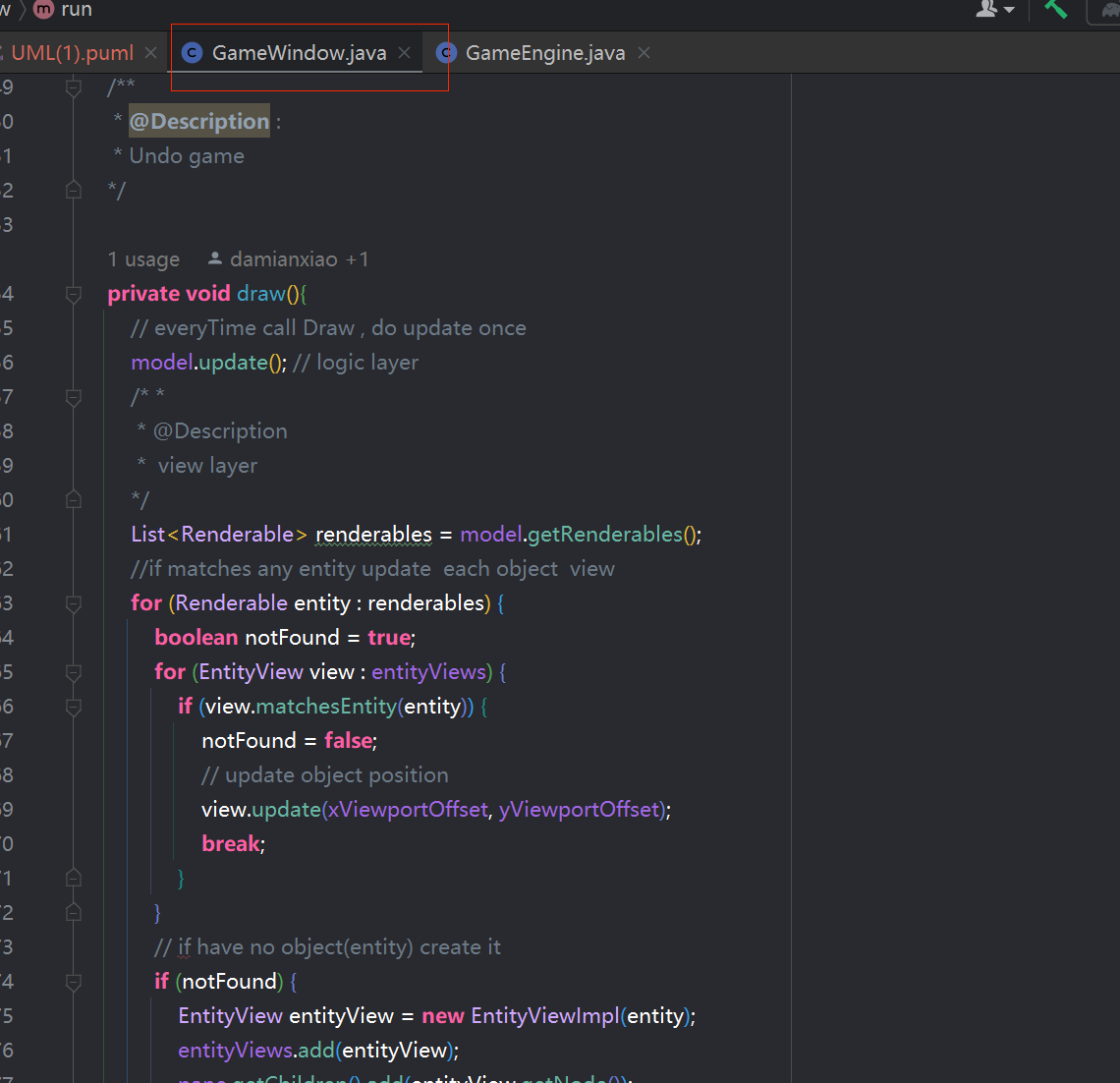


Fig 1

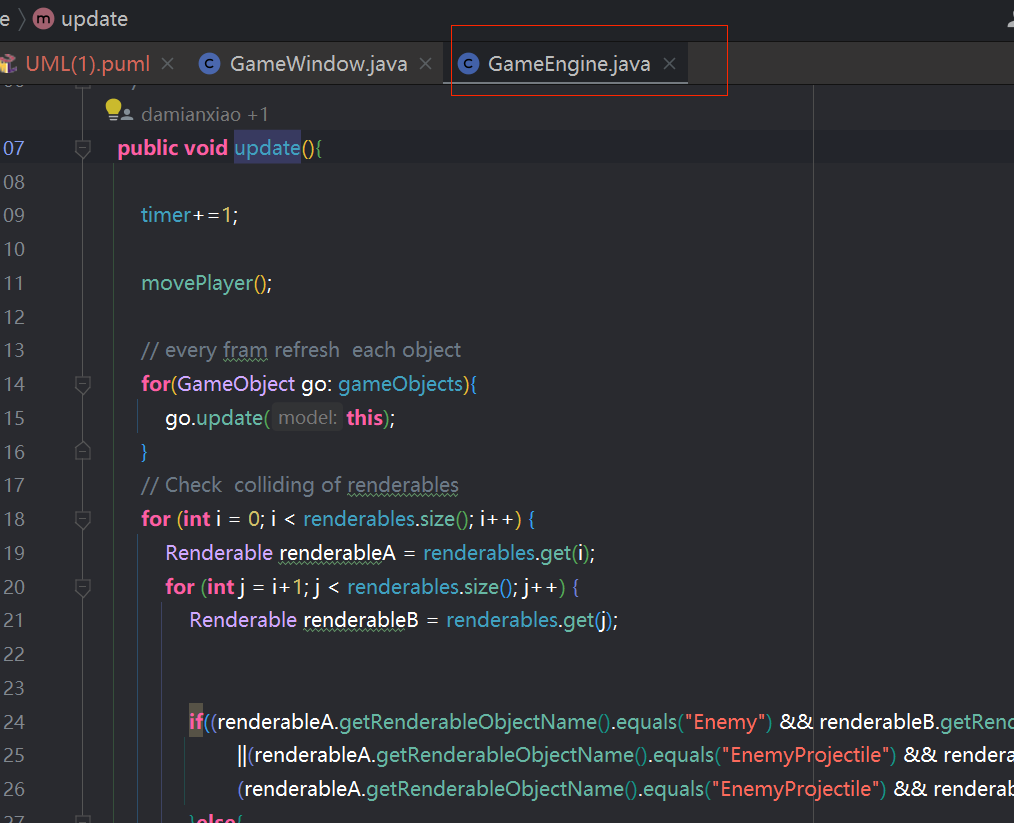


图 2

1. 开闭原则 OCP

在设计一个模块的时候，应当使这个模块在不被修改的情况下被扩展，换而言之，可以在不修改源代码的情况下改变这个模块的行为。

在代码中，所有的游戏实体都继承自Renderable或是gameobject。在有关于要修改游戏实体的操作时，我们大可重新创建一个类来改变这些逻辑（图3）。在需要实现新功能时，我们只需再实现这个接口。（图4）。

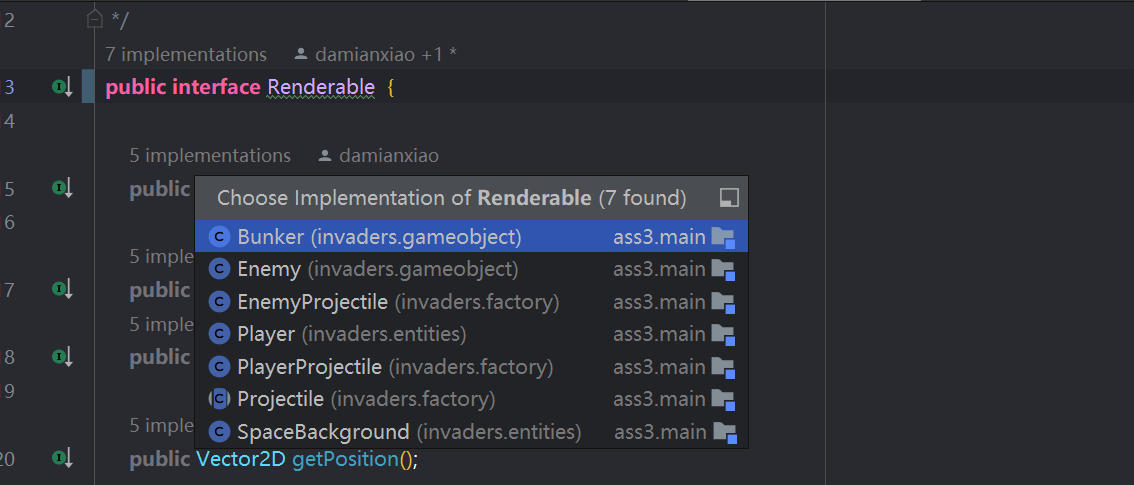


图 3

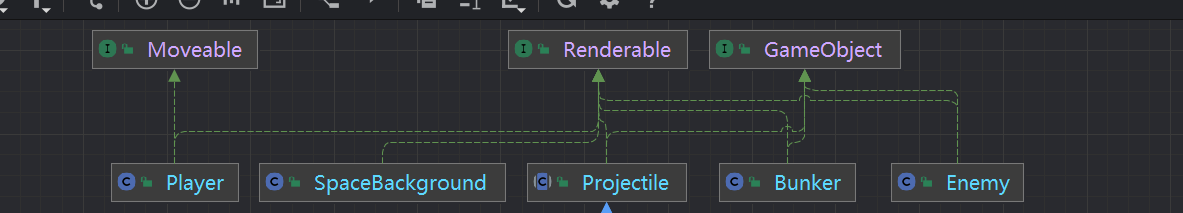


图 4

例如，在实现敌人的不同发射子弹的逻辑中，要实现slow、normal、fast不同的子弹逻辑，应该实现不同的类，而不是全部写在一个类中，如果还需要其他的子弹逻辑，也要优先创建新的类去实现接口（图5，图6）。

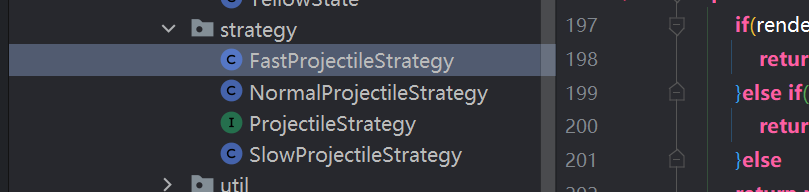


图 5

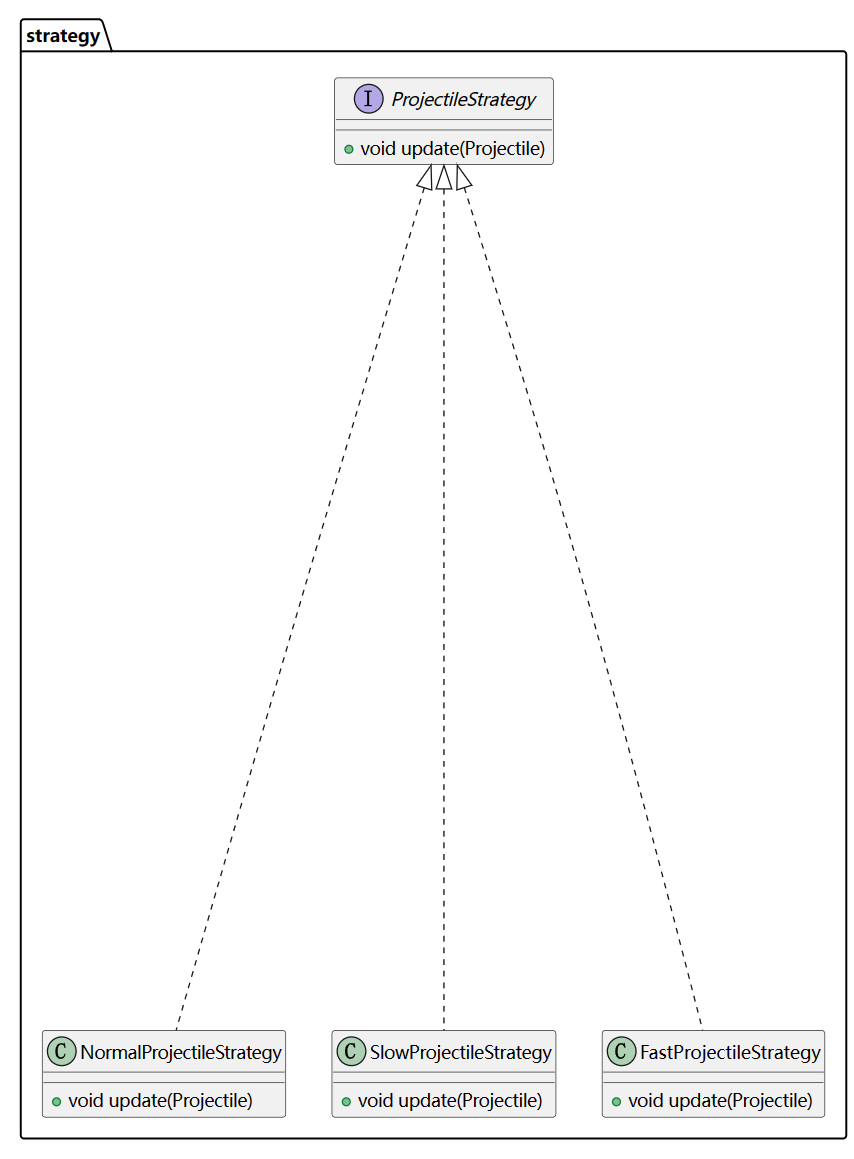


图 6

## 文档情况

1.路径问题：

在我进行项目依赖导入过程中，我发现gradle似乎不能正确的读取到源码所在的路径，经过排查我发现项目目录结构存在问题，在原始项目目录结构中，源码java和静态资源resource并未放置在src目录下，导致gradle不能正确读取（图7），我新建目录src并将他们放入，才正确读到这些资源。

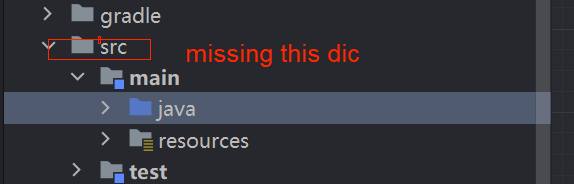


图 7

2.缺少 readme

另外， 本项目缺少readme文档，这可能会导致其他开发者或是使用用户难以理解项目的配置、安装。这会降低项目的可用性，同时缺少文档会使得维护人员难以理解工作原理，导致排查bug遇到阻碍。

1. 注释

在阅读源码的过程中，我注意到代码中缺少一些关键功能的注释。在核心功能的位置，应该提供详细的注释，以解释代码的运作原理。这将帮助其他开发者更好地理解代码，特别是在需要理解核心逻辑时。

## 开发难度

在本次task中，总共要求的功能拓展有以下几项：包括难度级别、时间和得分、以及撤销和作弊功能。

1. 难度级别

切换难度的本质是在gameEngine初始化时使其读取不同的难度配置文件，本项目源码给出的配置文件有三个：esay、medium、hard。我的思路是在进入正式游戏先进入一个难度选择的scene，在这个scene中放置三个按钮分别代表选取三个难度。点击按钮即可读取对应的难度文件，从而进入不同难度的游戏中。简单。

1. 游戏计时

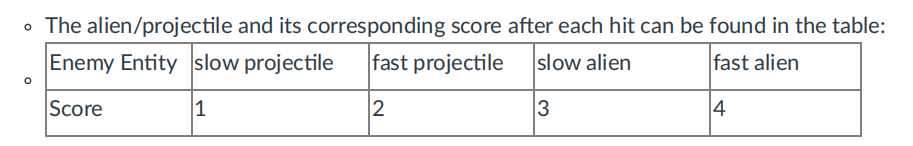
游戏计时将在用户进入游戏时，在游戏界面显示一个数据，代表当前玩家已进行的游戏时间。

我在游戏界面设置一个label对象，通过修改label text属性来代表游戏已进行的时间，并已秒为单位。要使得游戏时间随着游戏进度而记录更新，我们需要在gameWindow的run()方法中new 一条新的timeline。在这条timeline中我们设置每1s就执行一次更新时间的方法。

并且设置一个全局变量，每执行一次更新方案就使得变量加1。最终实现游戏计时。简单。

1. 得分

在游戏中，各个外星人实体都具有一定的分数如下图。如上面的计时功能一样，我们需要在游戏界面new一个label对象，label对象内的text代表游戏已得分数。为此我们需要设计更新分数的逻辑。而分数更新的逻辑应该遵循：当任意敌人实体被我们的玩家子弹破坏并死亡，则计分板加上对应的实体分数。



因此，我们需要在玩家的projectile与任意敌人实体碰撞时进行判断，如果敌对实体通过这次碰撞使得其health值归为0那么我们将更新计分板。此功能的难点在于我们需要对接触到的每一个实体进行判断，以按照表中的计分标准进行更新。稍简单。

1. 撤销

撤销功能允许玩家在任意时刻进行游戏保存，并可以之后的任意时刻回到这个时刻的游戏状态，但一次仅仅保存一个状态。这意味着在游戏中实现的撤销功能并不是通过将每个游戏状态都写入磁盘或保存在内存中的不同状态快照来实现的。相反，游戏只保存一个当前状态，当执行撤销操作时，当前状态会被覆盖，而不是添加到撤销历史中。这意味着在任何给定时间点，游戏只记录一个状态，撤销功能将恢复到上一个状态，而不能无限撤销到不同的历史状态。

我将这个功能开发分为几个步骤：1 任意时刻通过键入S来保存当前游戏状态，这个状态包括游戏时间、得分、实体位置等。2 将这个状态存入内存。 3 通过键入BackSpace回退到这个状态。首先，我们需要创建一个类gameState用于保存游戏状态，这个类应该包含上述的游戏属性，我定义为4个：1 时间 2 分数 3 所有渲染对象 4 所有gameObject对象。渲染对象和gameObject分别代表了此时刻的画面实体位置和它们的实体属性。

我们只需在任意时刻保存游戏状态后，通过键入BackSpace来将当前的gameState替换为保存时的gameState。退回状态时，需要先将当前的游戏渲染对象和gameObject清理掉，再进行置换。然而问题是，在保存状态时我发现并不能用直接拷贝的方式获取当前游戏的渲染对象和gameObject。在new gameState时，获取的renderables和gameObjects实际上是对象的引用。如果直接进行赋值拷贝这会导致在执行撤销操作时引用的仍然是相同对象，使得游戏状态并不发生改变。我推测这是代码中多个实体都同时实现了renderable接口和gameobject接口导致的结果。为了解决这个问题，我们需要对gameState的构造方法获取renderables和gameobjects的过程进行深拷贝处理。深拷贝方法需要遍历对象的属性并为每个属性创建新的对象，确保gameState引用的是新的对象而不是原始的引用。难。

1. 作弊

作弊功能要求玩家键入能够通过键入或按钮来删除特定类型的所有敌人。我将这个功能实现分为几个步骤：1创建键盘启用作弊：键入H触发操作，使得游戏的作弊功能开启，这样能够防止作弊功能被误触触发。并且，在开启作弊功能时，要在游戏界面中给出反馈。2 实现作弊逻辑，游戏中有四种类型的敌人，我们创建的作弊功能要能够对某类型的敌人删除，并结算这部分分数。3 更新游戏画面。

本功能的难点在于判定界面中敌人的类型并删除，积分更新的同时进行游戏界面的刷新。简单。

# 功能扩展

## 代码实现

1.难度级别功能：

本功能简单，在初始化Label样式时做了样式初始化类：styleModify的扩展，代码如下（图8,9）。

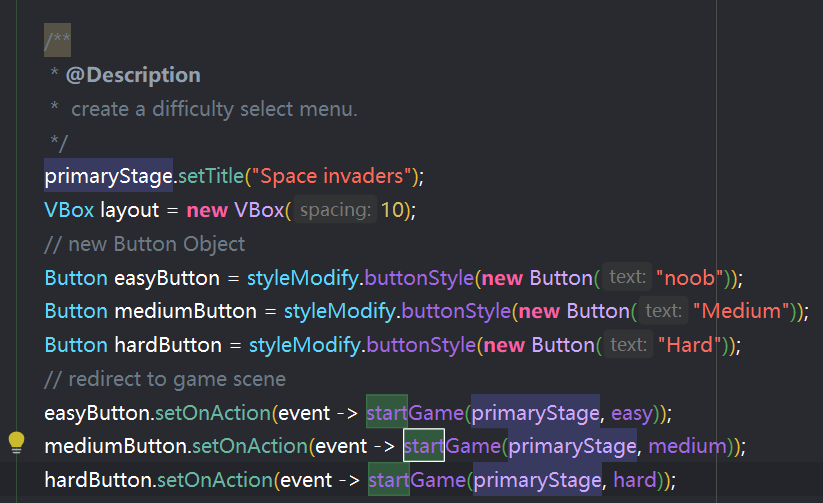


图 8

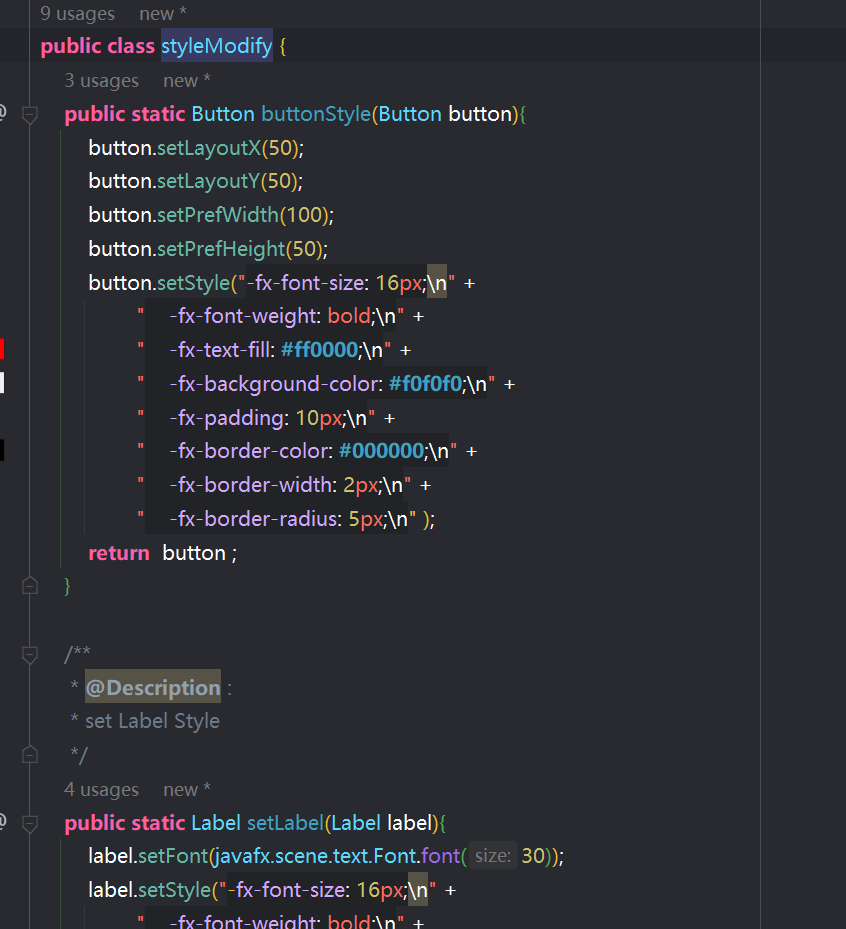


图 9 样式初始化类

1. 计时器、计分板功能

计时器功能代码在gameWindow的run方法中添加了一条新时间线。另外，Label的样式初始化方法同样放在styleModify类中，如下图（图10,11）。

计分板功能新增了scoreBroad类，其中的updateScoreCount用于计算获得分数。这个方法敌人实体被玩家子弹击败后触发，最后修改位于gameWindow中的scoreLabel，如下图（图12,13，14）。

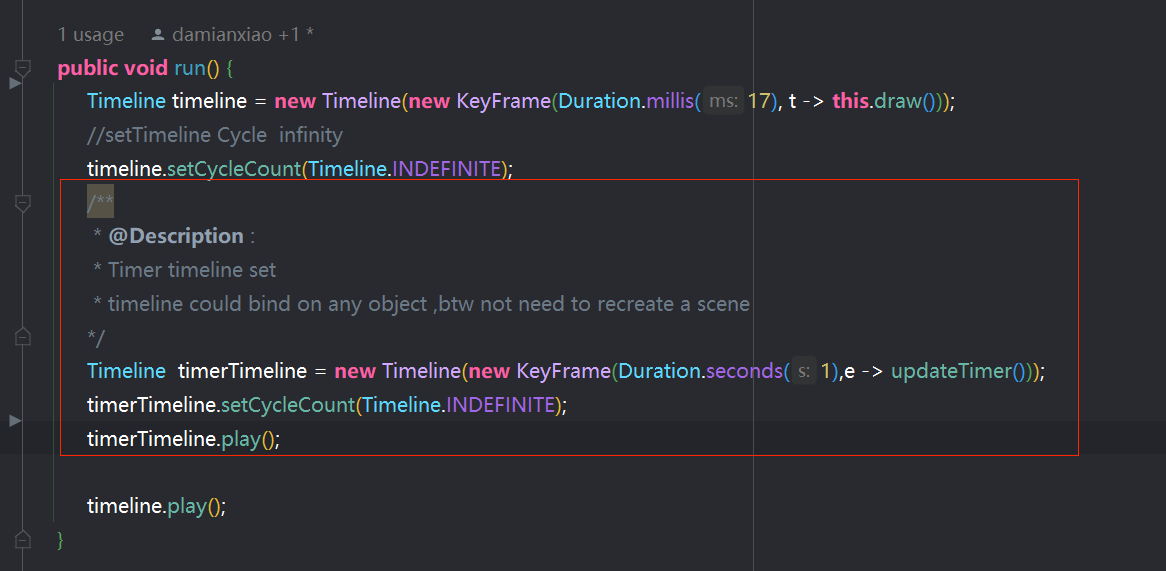


图 10 timeline set

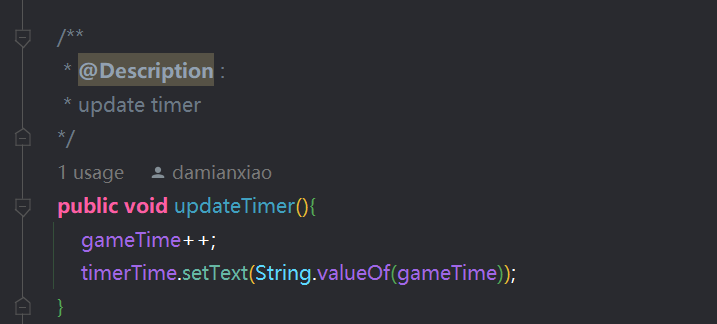


图 11 update time value

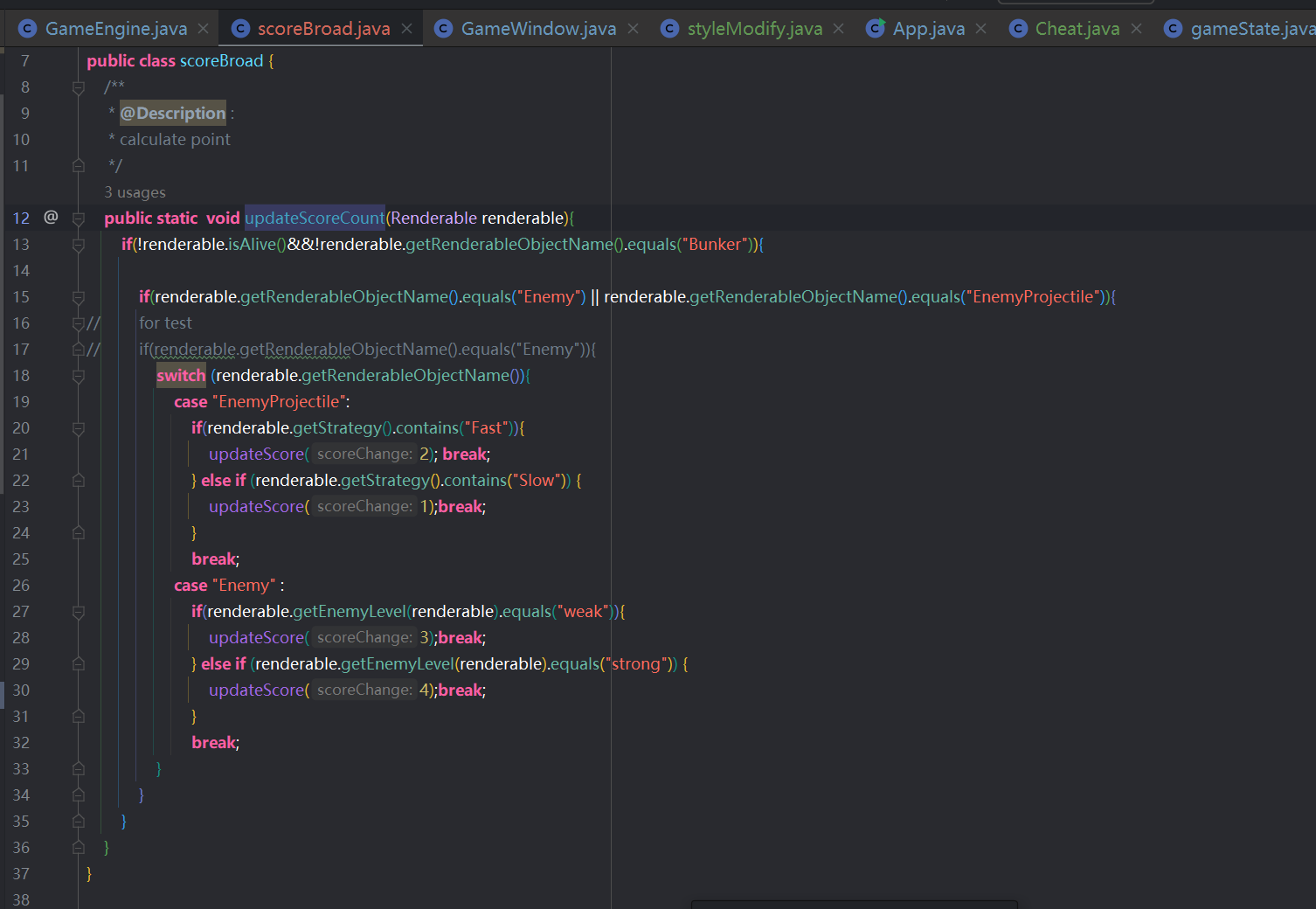


图 12 ScoreBroad class

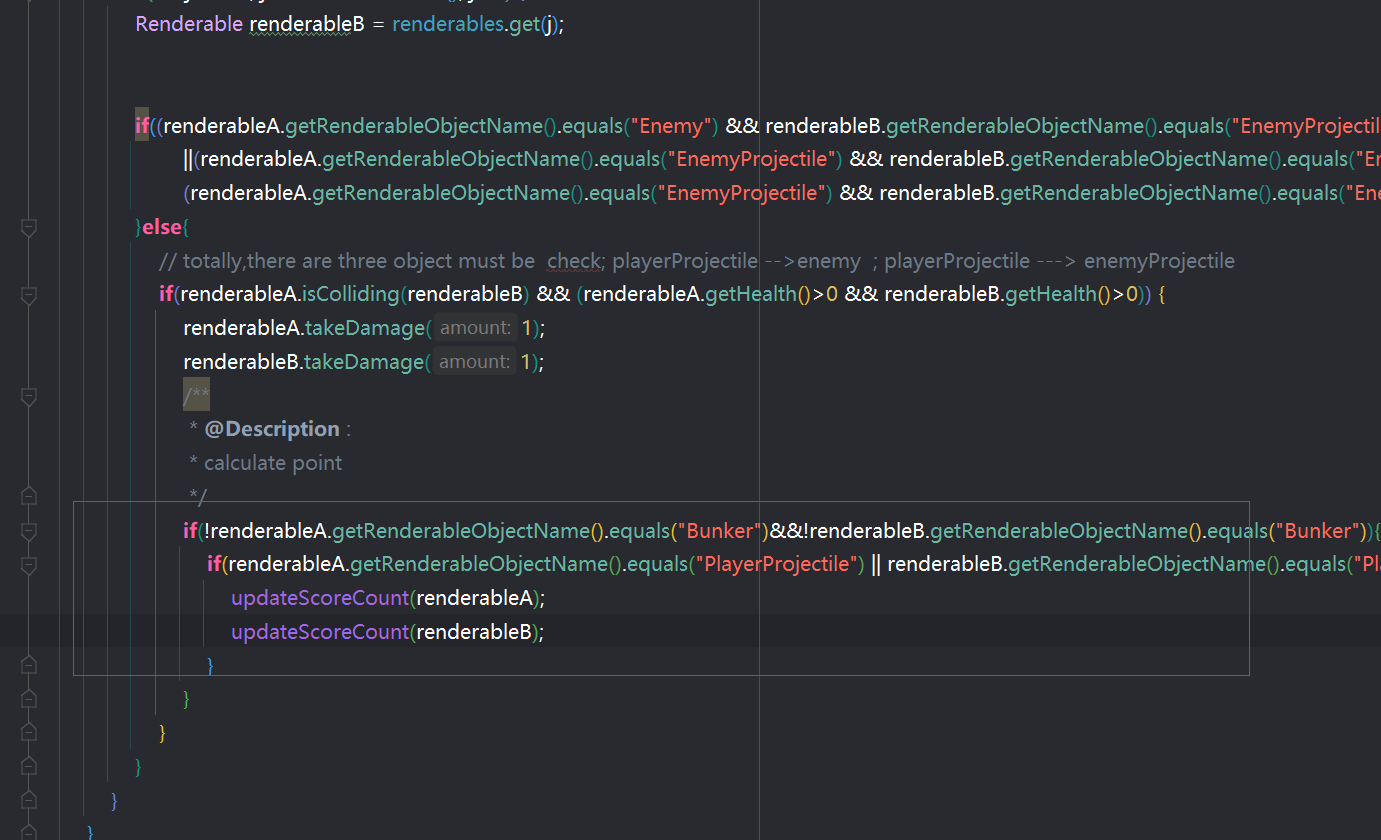


图 13 trigger this function

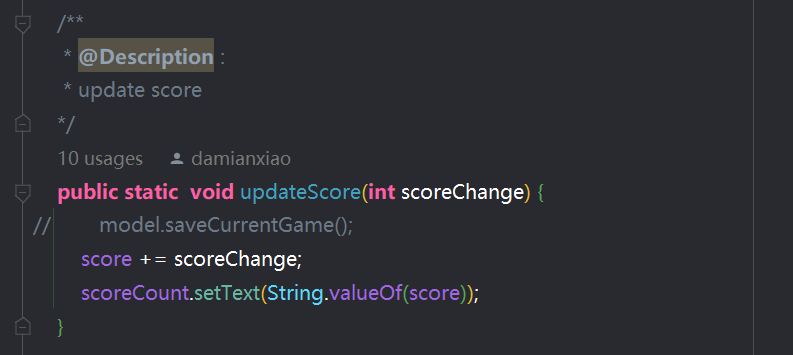


图 14 update scorelabel

1. 撤销功能

实现撤销功能，绑定存档键盘按键输入，绑定撤销键盘按键输入，创建gameState类保存游戏状态，创建gameUndo类实现撤销方法。对所有的实体完成深拷贝，实现保存当前状态功能，实现撤销到上一状态功能，最后更新实体渲染状态以及逻辑状态，代码如图下（图15-24）。

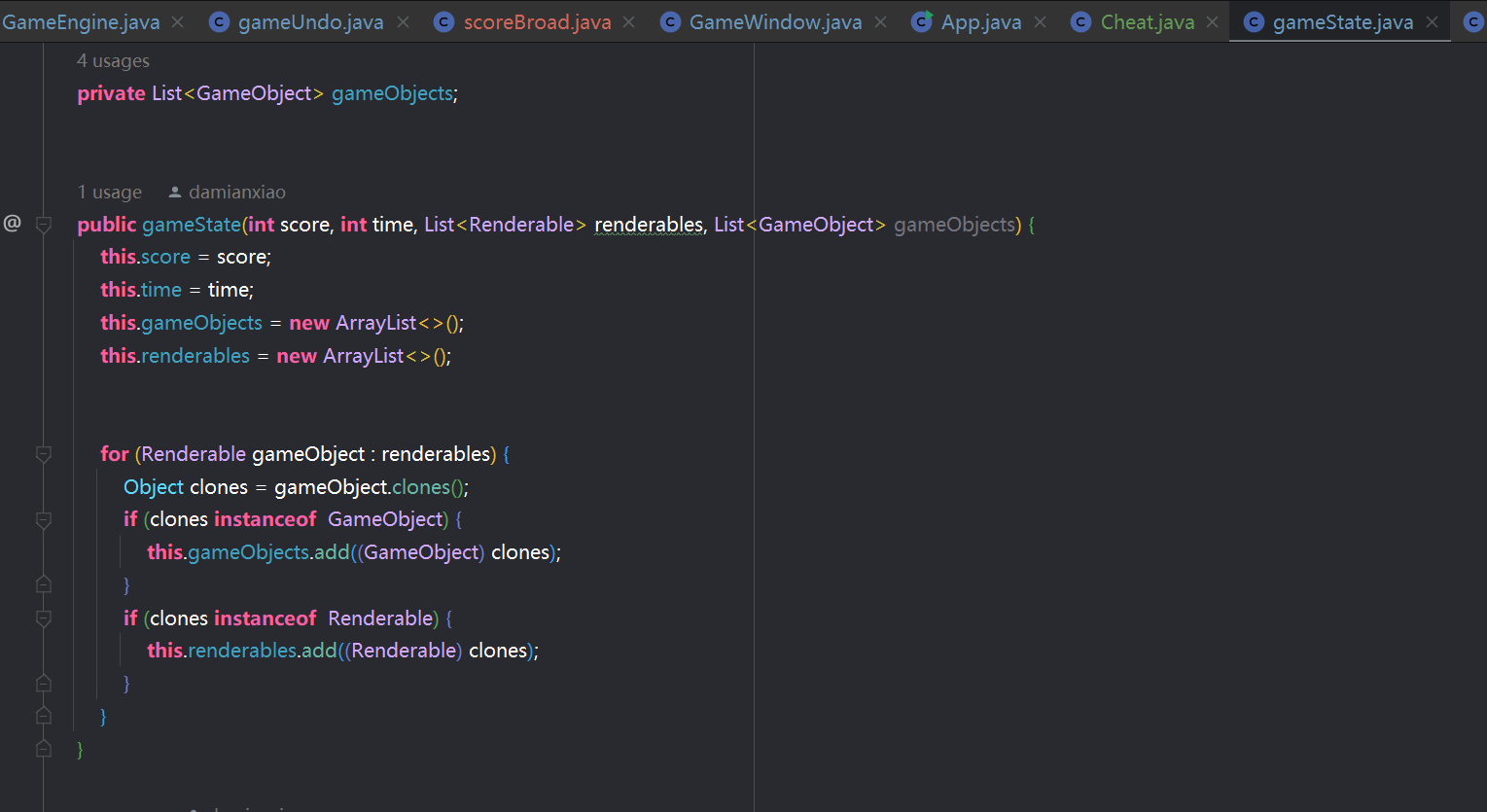


图 15 gameState class deepCopy

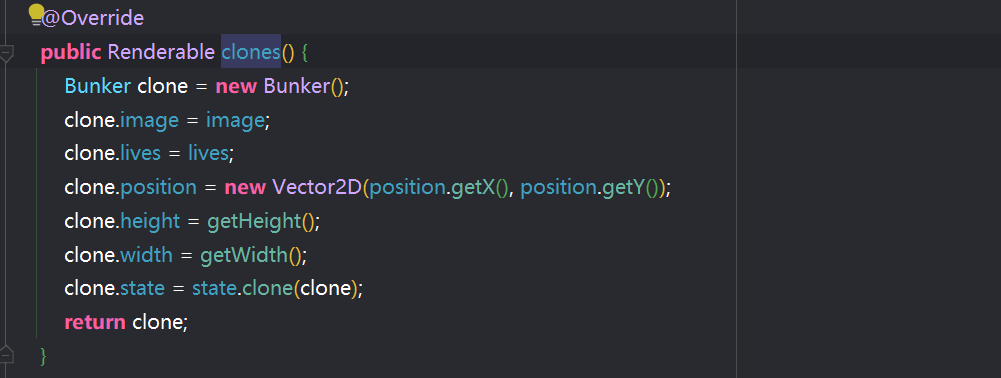


图 16 renderable’s implements DeepCopy

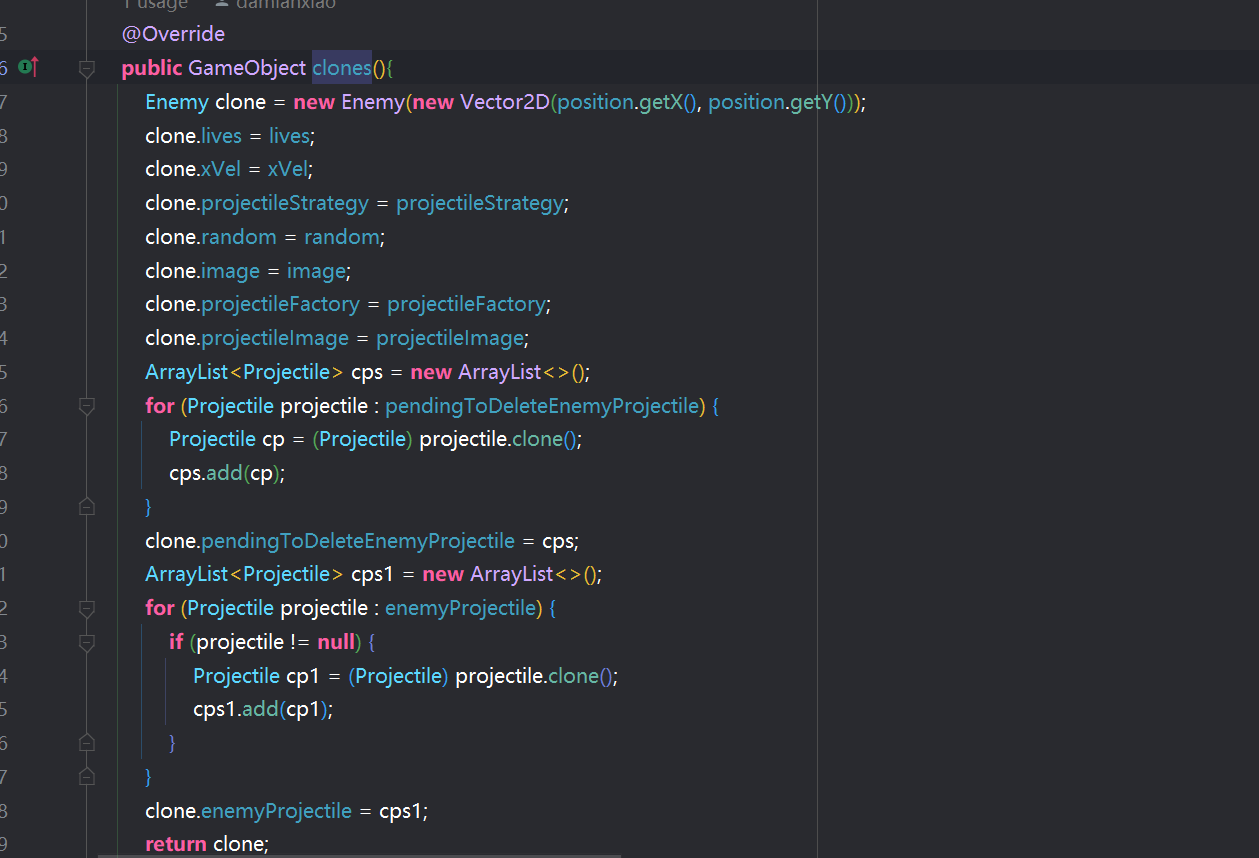


图 17 Enemy deepCopy

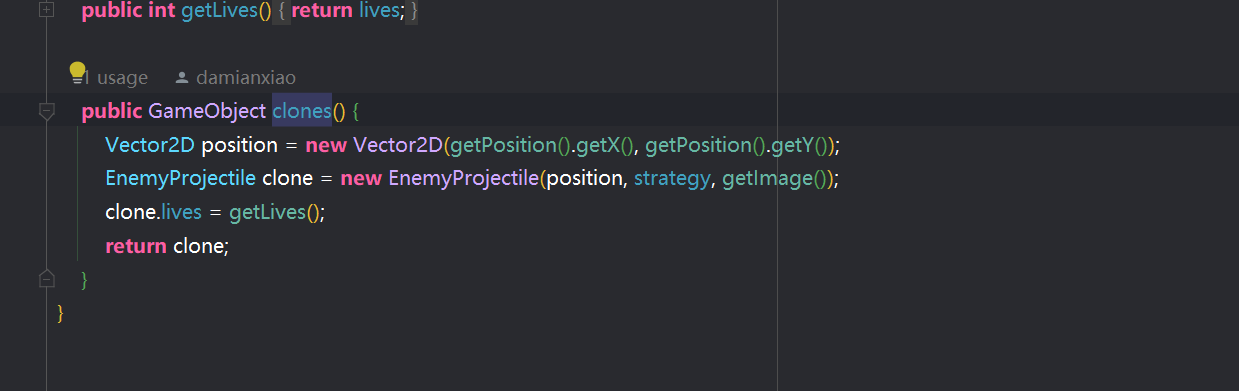


图 18 Projectile deepCopy

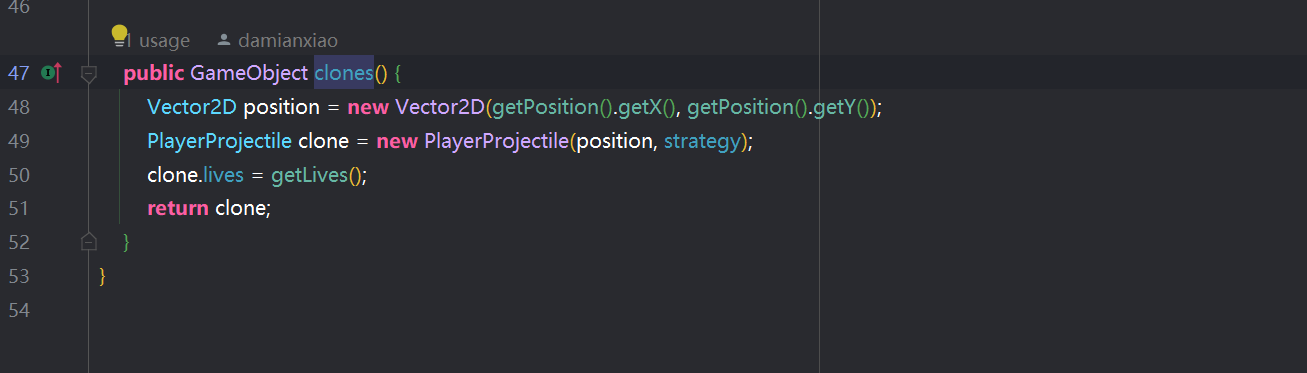


图 19 PlayerProjectile deepCopy

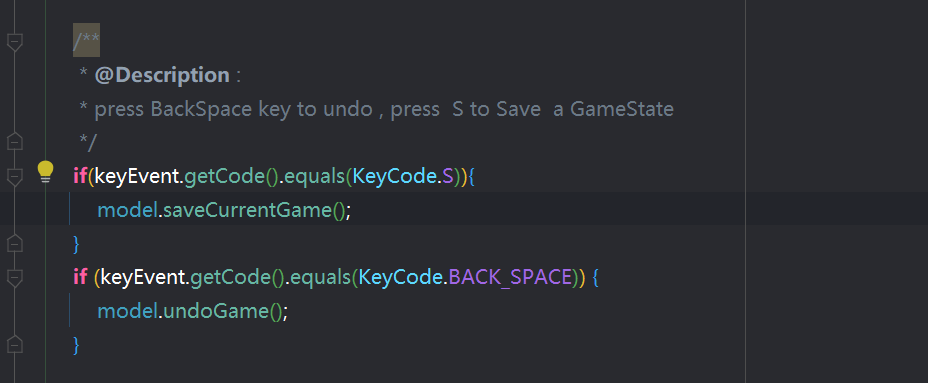


图 20 keybinds

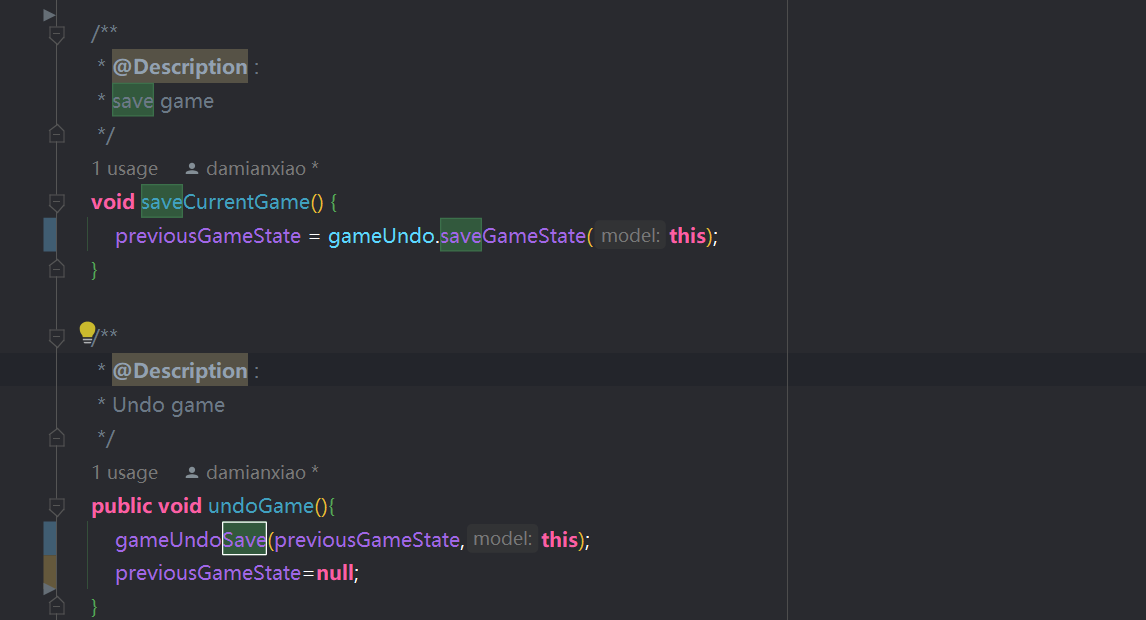


图 21 trigger function

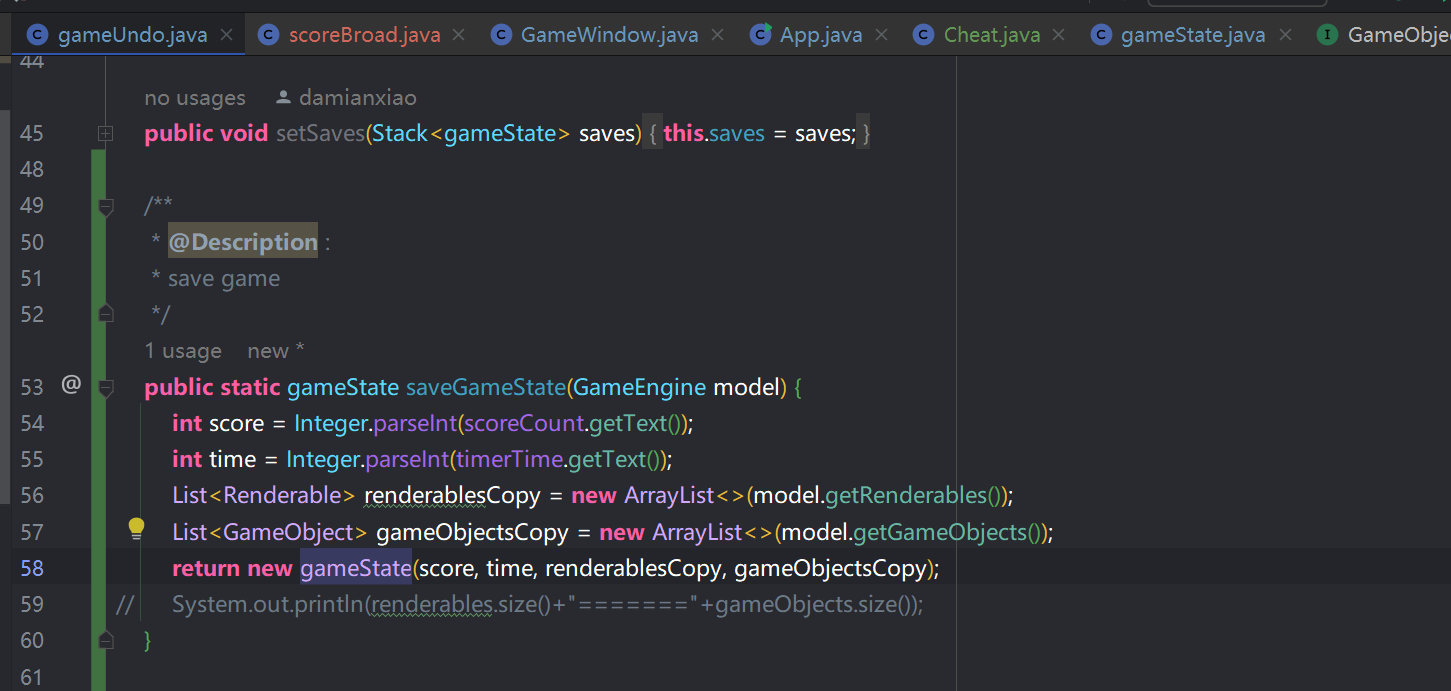


图 22 saveGameState

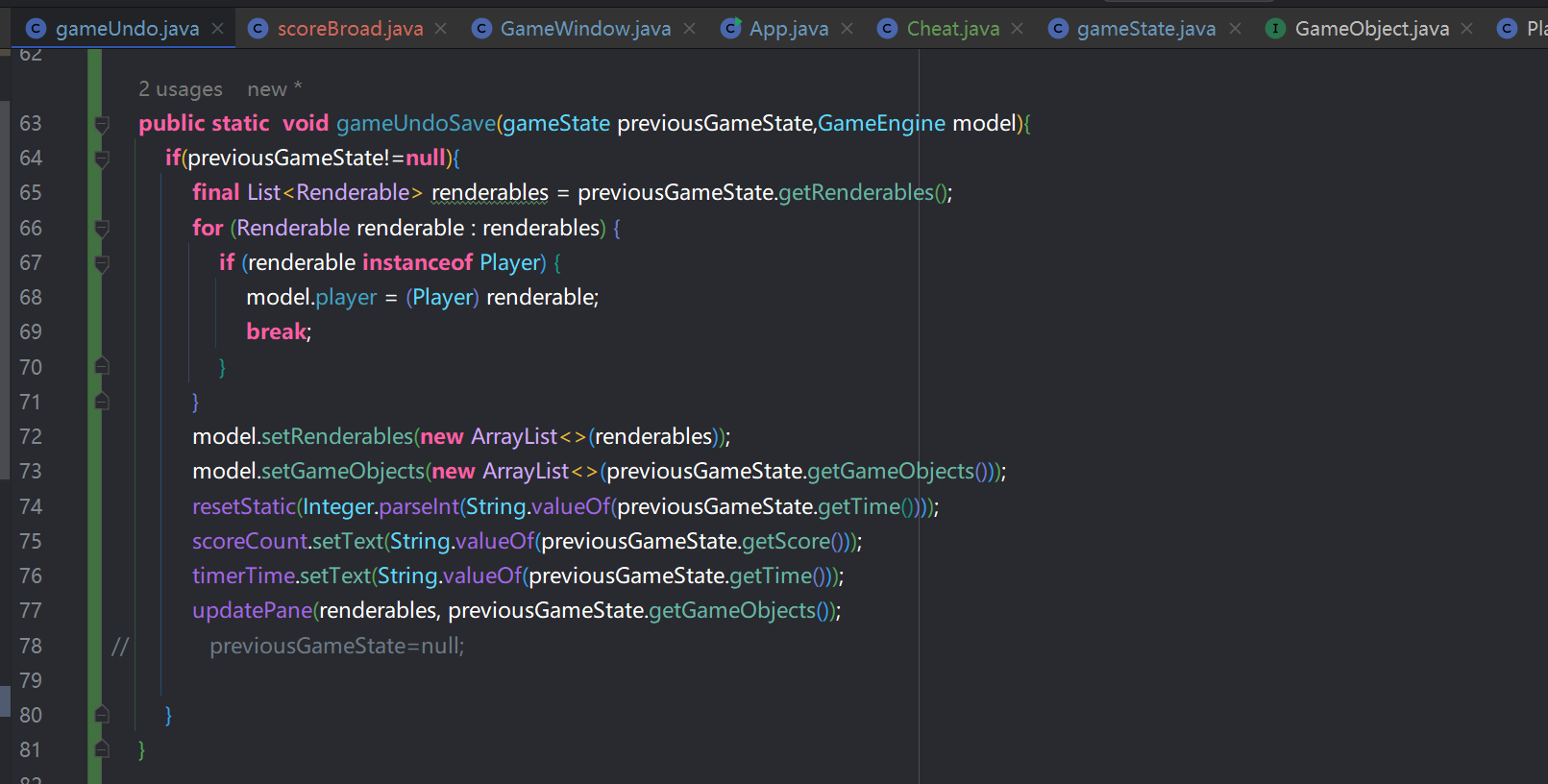


图 23 gameUndo

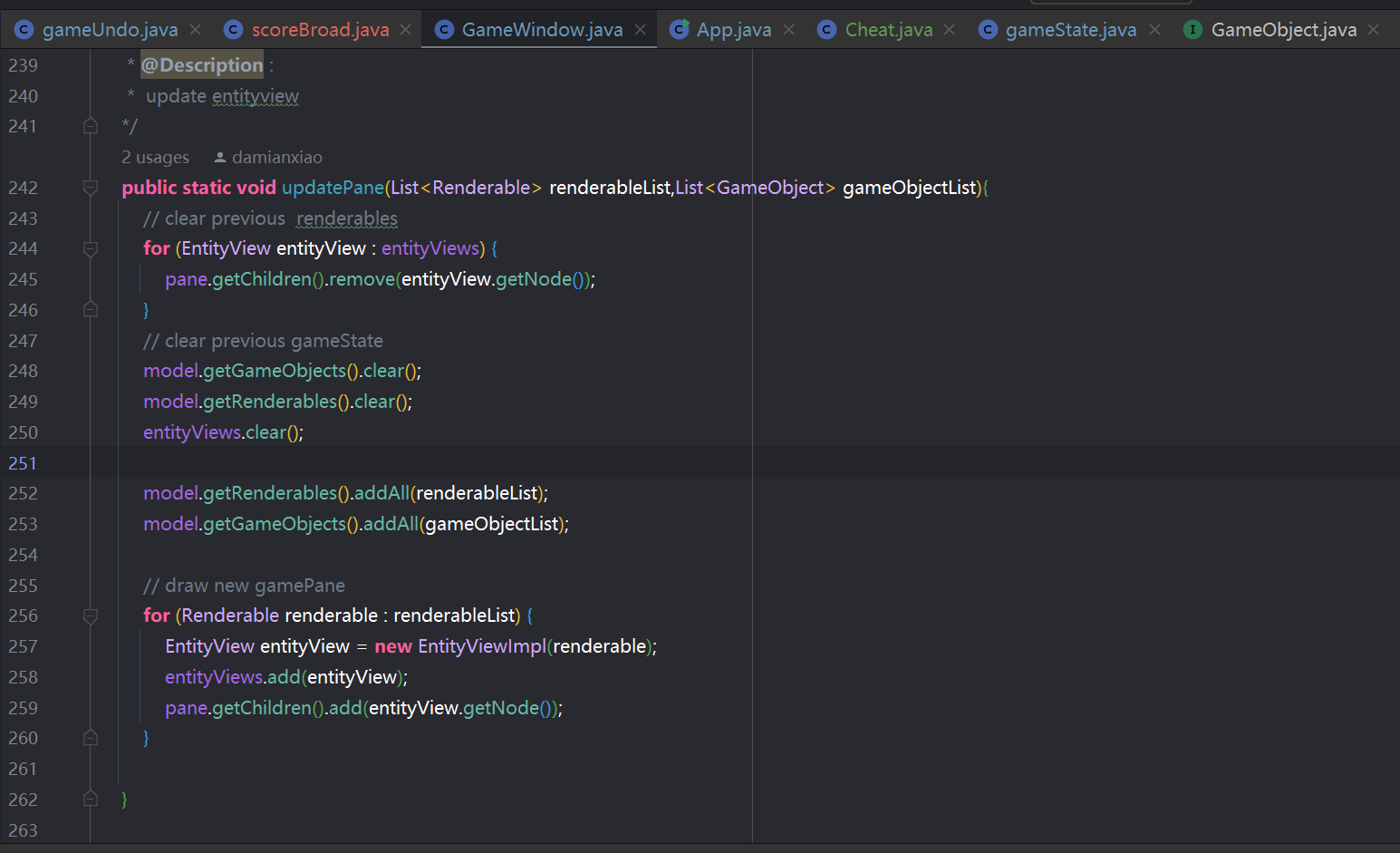


图 24 updatePane

1. 作弊功能

首先实现按键触发开启作弊，触发后在gameWindow中显示作弊已开启字样提示。获得提示后，可通过按下不同的按键触发作弊功能，最后更新得分和渲染对象。代码如图下（图25）。

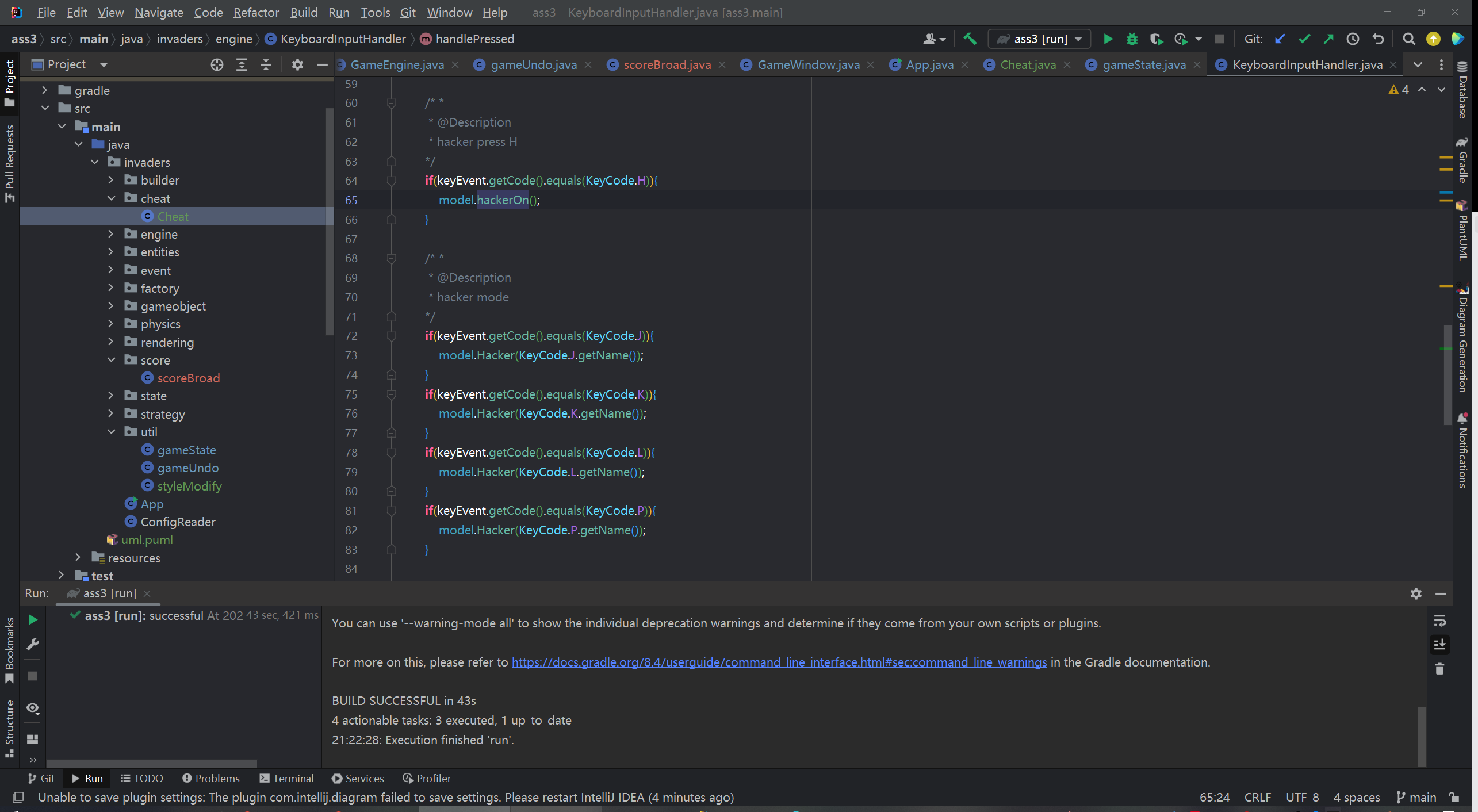


图 25 keybinds

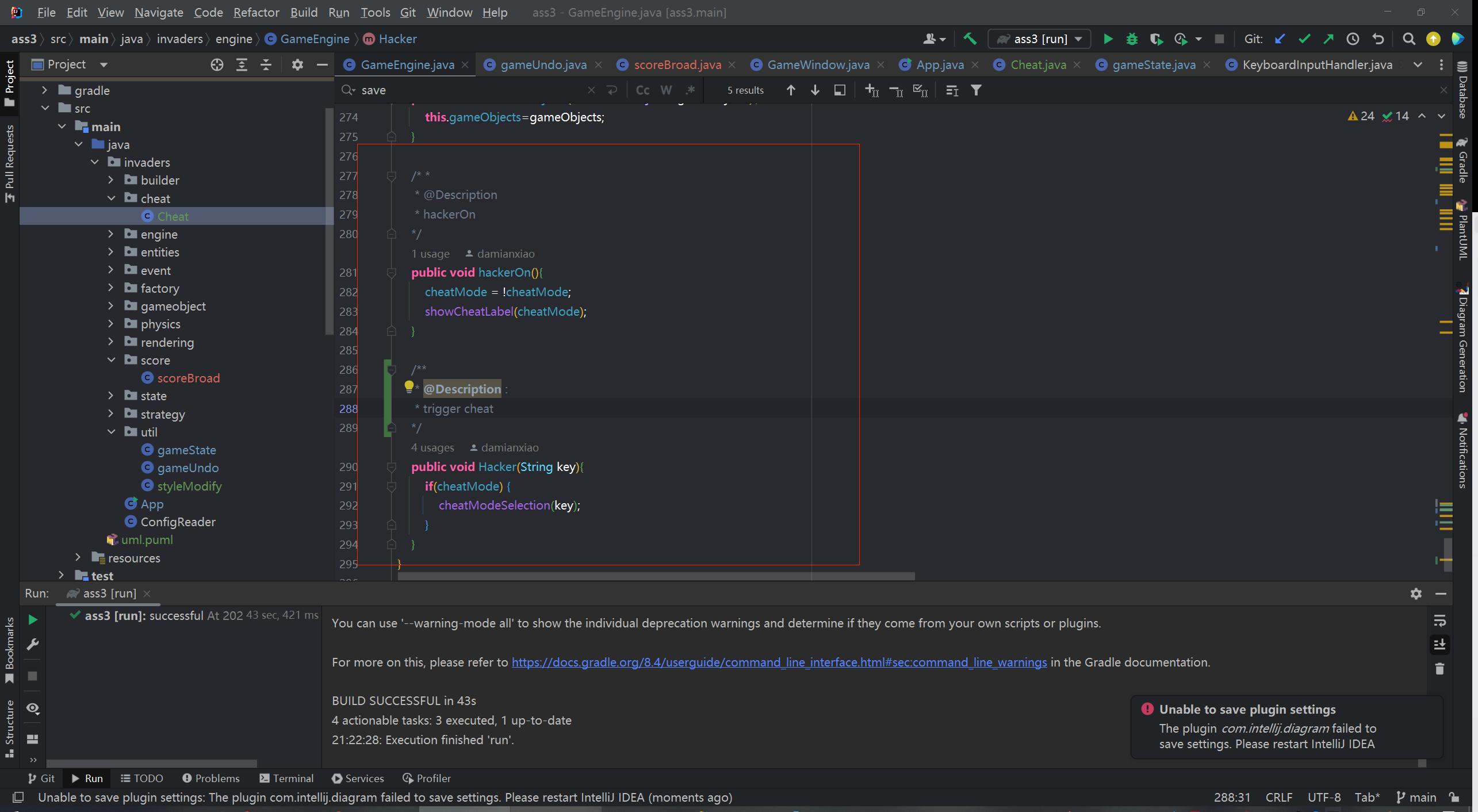


图 26 trigger function in gameEngine

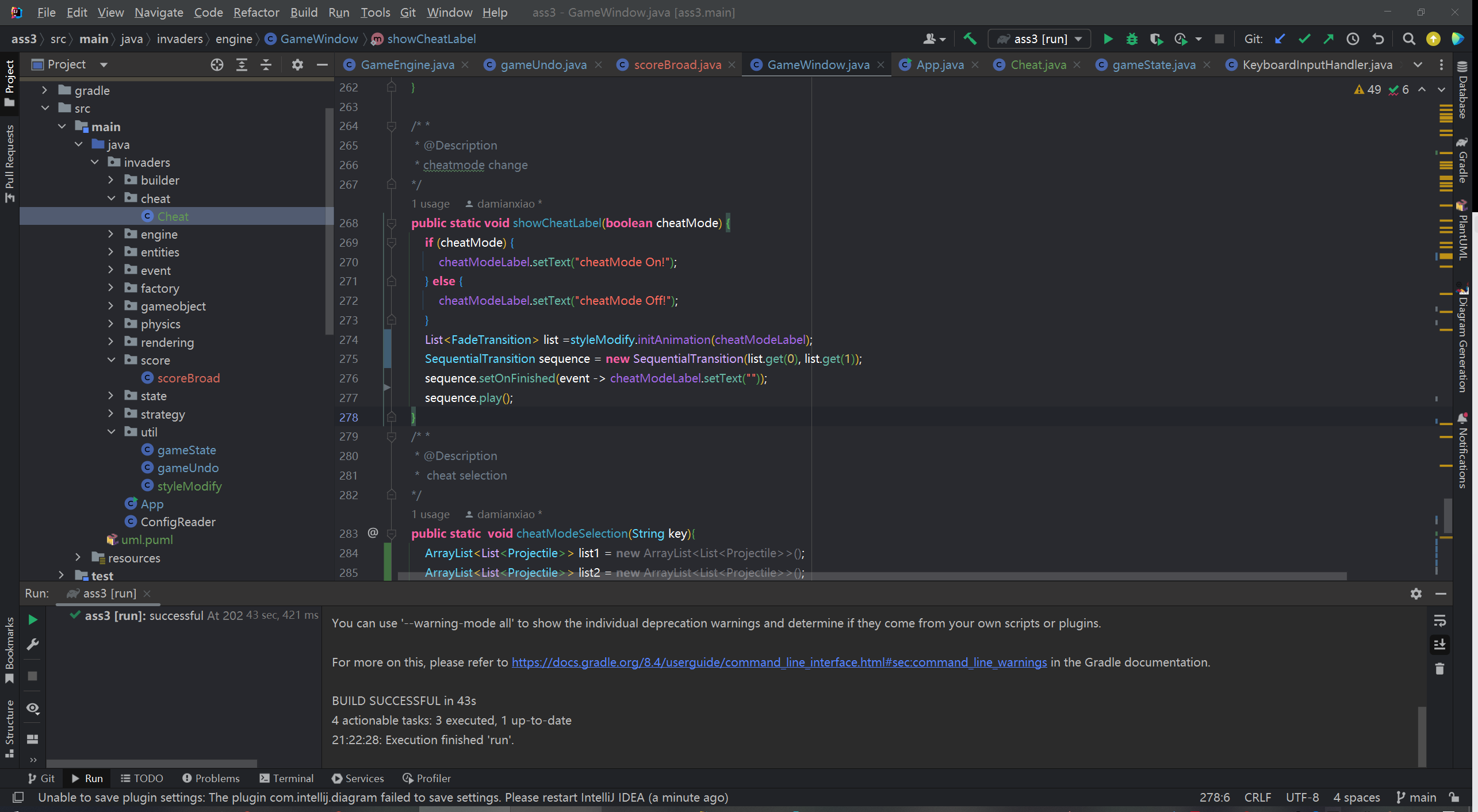


图 27 cheat on label signs

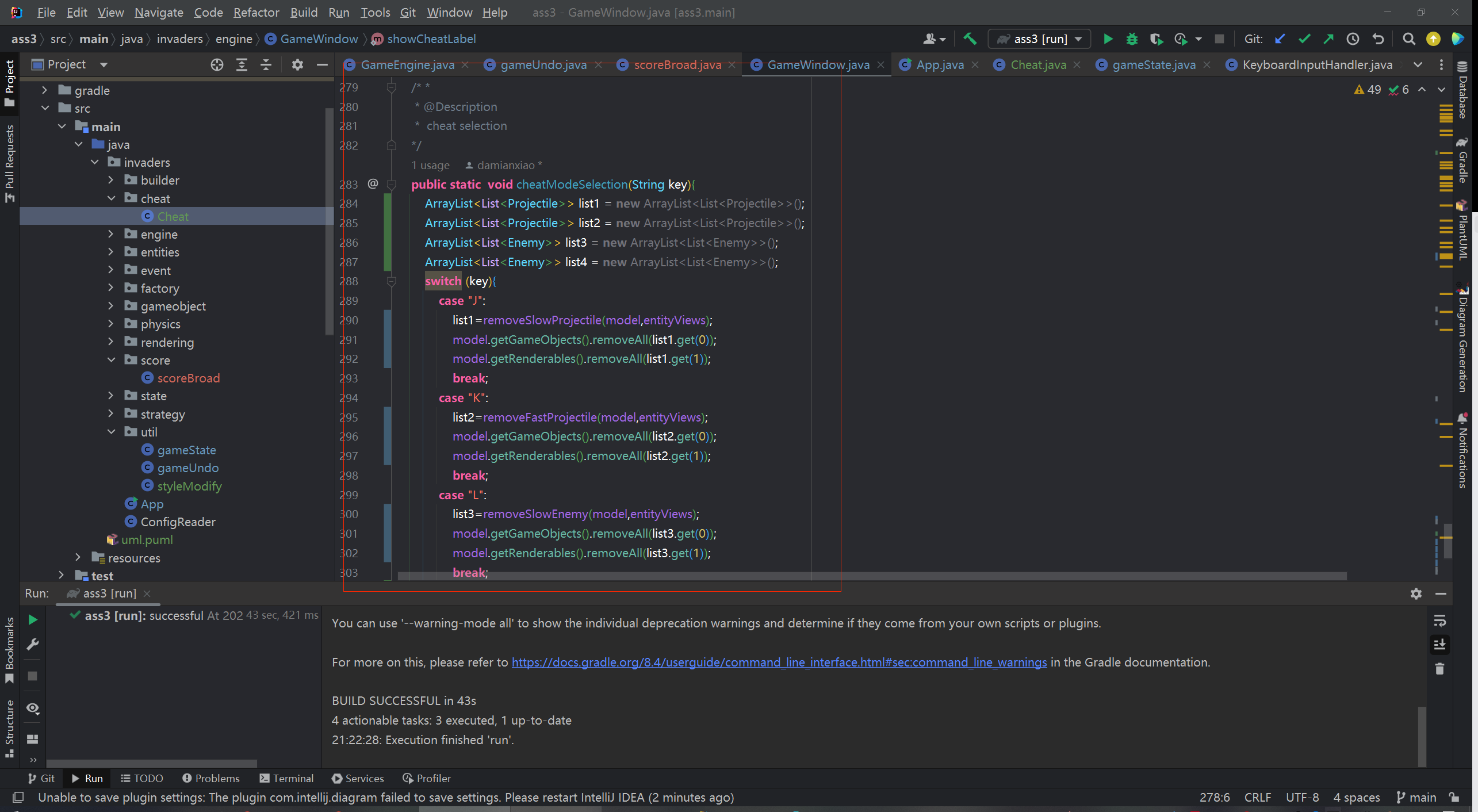


图 28 press different key to trigger cheat effects

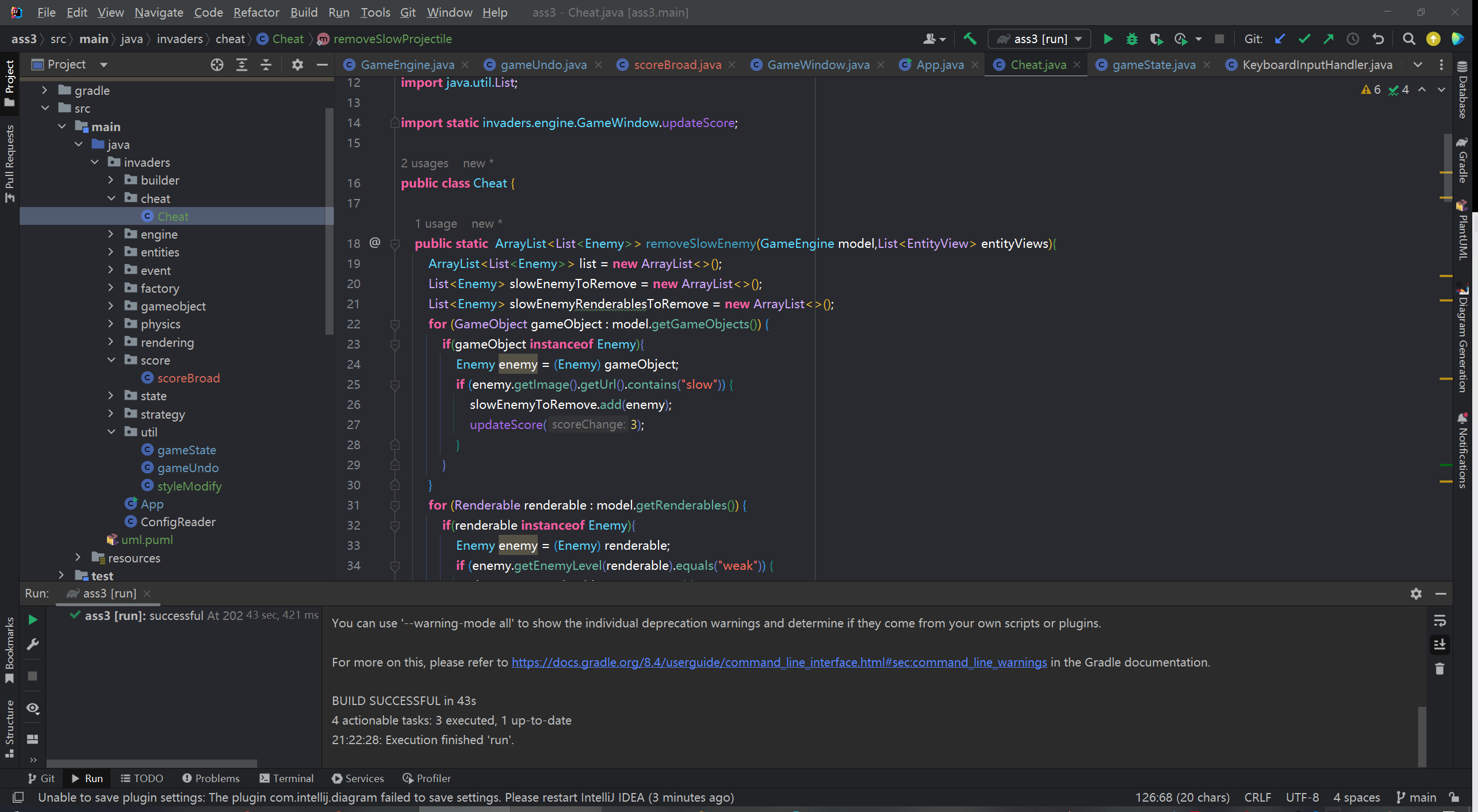


图 29 implements effect function in Cheat class