

三峡大学

计算机与信息学院

《 软件测试课程考核报告 》 报告

开课学期： 2023 年 春季 学期

网选班号： 3 班

学 号： 2020112802

姓 名： 陆万奇

完成日期： 2023 年 5 月 25 日

目录

1	概述.....	1
1.1	什么是 BDD.....	1
1.2	Cucumber 简介	1
1.3	Jpacman 简介.....	2
2	任务说明	2
2.1	任务目标	2
2.2	测试环境.....	2
2.3	条件与限制	2
3	测试计划	3
3.1	测试范围	3
3.2	测试目标	4
4	测试内容及方法.....	5
4.1	测试工具	5
4.2	Cucumber 测试框架准备	5
4.2.1	Cucumber 安装	5
4.2.2	Cucumber 配置	5
4.3	测试用例编写	6
4.4	实现自动化测试.....	8
4.4.1	编写 feature 文件	8
4.4.2	编写测试代码	8
4.4.3	执行 cucumber 测试.....	10
5	测试结果分析	10
5.1	执行测试用例结果.....	10
5.2	测试覆盖率分析.....	11
5.2.1	Jacoco 配置	11
5.2.2	Jacoco 覆盖率报告分析.....	12
6	总结与回顾	13
	参考文献	13

基于 cucumber 框架的 Java 实现 BDD 测试

陆万奇

摘要：本文基于 cucumber 框架，对经典的软件工程教学项目 Jpacman 进行 BDD 测试。根据官方文档给出的“故事”，进行自动化的功能集测试。主要针对故事 2：玩家移动以及故事 3：幽灵移动进行功能测试。验证了游戏单位在不同故事步骤描述下的正确行为。同时使用 jacoco 覆盖率测试工具，通过配置 gradle 任务完成实现测试 cucumber 集成测试覆盖率。

关键词：集成测试；Jpacman；cucumber；BDD 自动化测试

1 概述

我选择的考核题目是：基于 cucumber 框架的 Java 实现 BDD 测试。

1.1 什么是BDD

行为驱动开发（英语：Behavior-driven development，缩写 BDD）是一种敏捷软件开发的技术，它鼓励软件项目中的开发者、QA 和非技术人员或商业参与者之间的协作。

BDD 最初是由 Dan North 在 2003 年命名[1]，它包括验收测试和客户测试驱动等的极限编程的实践，作为对测试驱动开发的回应。在过去数年里，它得到了很大的发展。

2009 年，在伦敦发表的“敏捷规格，BDD 和极限测试交流”中，Dan North 对 BDD 给出了如下定义：

BDD 是第二代的、由外及内的、基于拉(pull)的、多方利益相关者的(stakeholder)、多种可扩展的、高自动化的敏捷方法。它描述了一个交互循环，可以具有带有良好定义的输出（即工作中交付的结果）：已测试过的软件。

1.2Cucumber简介

Cucumber 是 BDD（Behavior-Driven Development，行为驱动开发）的一个自动化测试工具，使用自然语言来描述测试用例，使得 非研发（QA、PM）也可以理解甚至编写 测试用例。

Gherkin 是 Cucumber 用来描述 测试用例 的语言，以下为关键字的用意与关联关系。



1.3 Jpacman简介

类似于吃豆子的游戏，用于软件测试教学。它让学生接触到 git、Gradle、JUnit 和 mockito 的使用。

部分代码经过了很好的测试，而其他的代码则故意不进行测试。作为软件测试的学生，你可以扩展测试套件，或使用代码库以测试驱动的方式建立扩展。作为一名教师，你可以使用 JPacman 来创建你自己的测试练习。

我们已经在荷兰代尔夫特理工大学的软件测试课程中开发并正在使用这段代码。有兴趣看到我在那里使用的练习的老师，请与我联系。

其他使用该材料的大学包括安特卫普、蒙斯、埃因霍温和 UBC（温哥华）。在代尔夫特理工大学，我们将其与 GitLab 结合使用，作为持续集成和反馈服务器。

2 任务说明

2.1 任务目标

对 JPacman 实现 BDD 自动化集成测试，可以针对 doc/sceraios.md 中 Story 2: Move the Player, Story 3: Move The Ghost 用户故事测试；

2.2 测试环境

硬件环境：Windows 11，HUAWEI MateBook 14 2020

软件环境：IntelliJ IDEA 2023.1

2.3 条件与限制

硬件环境：无严格要求，可以运行 JVM 环境，且具有基础图形化能力的硬件架构即可

软件环境：Java8 及以上且具有 Gradle 的开发环境即可

3 测试计划

3.1 测试范围

根据 Jpacman 项目提供的官方“场景”文档，可以得到四个以用户故事形式描述的功能说明（以故事 1 和故事 2 为例）。详细内容见项目源码，路径为：/doc/scenarios.md

Story 1: Startup

As a player

I want to start the game
so that I can actually play

Scenario S1.1: Start.

Given the user has launched the JPacman GUI;

When the user presses the "Start" button;

Then the game should start.

Story 2: Move the Player

As a player,

I want to move my Pacman around on the board;
So that I can earn all points and win the game.

Scenario S2.1: The player consumes

Given the game has started,

and my Pacman is next to a square containing a pellet;

When I press an arrow key towards that square;

Then my Pacman can move to that square,

and I earn the points for the pellet,

and the pellet disappears from that square.

Scenario S2.2: The player moves on empty square

Given the game has started,

and my Pacman is next to an empty square;

When I press an arrow key towards that square;
Then my Pacman can move to that square
and my points remain the same.

Scenario S2.3: The move fails

Given the game has started,
and my Pacman is next to a cell containing a wall;

When I press an arrow key towards that cell;

Then the move is not conducted.

Scenario S2.4: The player dies

Given the game has started,
and my Pacman is next to a cell containing a ghost;

When I press an arrow key towards that square;

Then my Pacman dies,
and the game is over.

Scenario S2.5: Player wins, extends S2.1

When I have eaten the last pellet;

Then I win the game.

本文主要针对其中的故事 2 和故事 3 进行测试。

3.2 测试目标

表格 1 测试目标表

类别	目标
进度目标	Story2 测试完成日期: 2023/05/20
	Story3 测试完成日期: 2023/05/25
执行目标	测试用例通过率 100%
	缺陷清除率 100%

4 测试内容及方法

4.1 测试工具

本文主要使用了 cucumber, Junit5, hamcrest 三大框架实现 BDD 集成测试。项目构建工具为 Gradle。使用了 Idea 插件 Cucumber+协助编写测试文件。

4.2 Cucumber测试框架准备

4.2.1Cucumber 安装

通过 gradle 构建工具进行 cucumber 安装配置，依赖语句如下：

```
testImplementation "io.cucumber:cucumber-java:$cucumberVersion"
testImplementation "io.cucumber:cucumber-junit:$cucumberVersion"
```

由于项目同时使用了 Junit5，需要同时配置 cucumber 的适配环境，依赖语句如下：

```
testImplementation "io.cucumber:cucumber-junit-platform-engine:$cucumberVersion"
```

这一配置使得 cucumber 可以以单元测试的形式启动。

4.2.2Cucumber 配置

在 build.gradle 中配置出 cucumber 测试任务，具体代码如下：

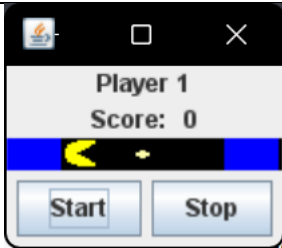
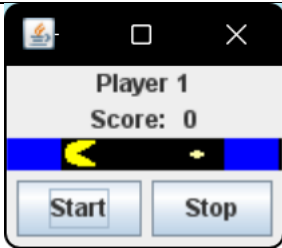
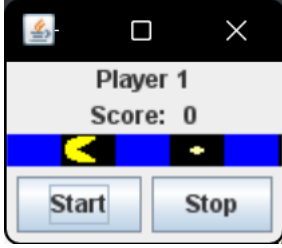
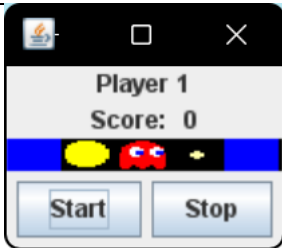
```
task cucumber() {
    dependsOn assemble, compileTestJava, processTestResources
    doLast {
        javaexec {
            main = "io.cucumber.core.cli.Main"
            classpath = configurations.cucumberRuntime + sourceSets.main.output +
sourceSets.test.output
            args = ['--plugin', 'pretty',
                '--plugin', 'html:target/cucumber-report.html',
                '--glue', 'nl.tudelft.jpacman.cucumberTest',
                'src/test/resources/cucumberTest']
        }
    }
}
```

其中 args 参数配置了 cucumber 对应的启动参数，本文主要配置了以下四项：--plugins 配

置了生成报告的形式为美化(pretty)的 html 形式(html:target/cucumber-report.html)。--glue 配置了 cucumber 框架使用的 java 解析代码包位置，尾行是 cucumber 的 feature 文件位置，按照惯例应放在 test 的 resource 文件夹下。

4.3 测试用例编写

根据故事文档，可以由每个场景编写一个对应的测试用例：

场景编号	步骤过程	测试地图	断言点
S2.1 The player consumes	玩家和豆子相邻，玩家移动到豆子上吃掉豆子并得分		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 玩家右侧方格上有且只有一个单位 3. 这个单位是豆子的一个实例 4. 玩家能否正确移动 5. 玩家移动后，豆子消失，且玩家得分加一
S2.2 The player moves on empty square	玩家和空方格相邻，玩家移动到空方格上		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 玩家右侧方格上是否没有单位 3. 玩家能否正确移动 4. 玩家移动后分数没有产生变化
S2.3 The move fails	玩家和墙相邻，玩家无法完成向墙的移动		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 玩家右侧方格为一个墙的实例 3. 玩家无法完成移动
S2.4 The player dies	玩家和幽灵相邻，玩家向幽灵移动后，玩家死亡		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 玩家右侧方格有且只有一个单位 3. 玩家右侧单位为一个幽灵实例 4. 玩家向右移动后游戏结束

S2.5 Player wins, extends S2.1	在 S2.1 场景基础上， 玩家吃掉的是游戏中 最后一个豆子，于是玩 家胜利了		<ol style="list-style-type: none"> 1. 重新执行 S2.1 场景，并遍历其中的断言点 2. 地图上已经没有豆子了 3. 游戏结束了
--	--	---	--

表格 2 Story2 场景测试用例

场景编号	步骤过程	测试地图	断言点
S3.1 A ghost moves	幽灵和空方格相邻，下一帧时幽灵移动到了		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 幽灵右侧方格上没有单位 3. 幽灵能否正确移动
S3.2 The ghost moves over a square with a pellet	幽灵和豆子相邻，移动到豆子上时，会覆盖豆子，使豆子不可见		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 幽灵右侧方格上有且仅有一个单位 3. 这个单位为一个豆子的实例 4. 幽灵能否正确移动 5. 移动后豆子不可见 6. 移动完成后，方格上应该有两个单位
S3.3 The ghost leaves a cell with a pellet	在 3.2 的基础上，幽灵又离开了豆子所在的方格，豆子又重新可见了		<ol style="list-style-type: none"> 1. 重新执行 S3.2 场景，并遍历其中的断言点 2. 幽灵可以继续向右移动 3. 豆子所在方格有且仅有一个单位，为一个豆子的实例
S3.4 The player dies	幽灵和一个玩家相邻，幽灵移动到玩家上，游戏结束		<ol style="list-style-type: none"> 1. 游戏是否启动 2. 幽灵右侧方格上有且仅有一个单位 3. 该单位为玩家实例 4. 幽灵能否完成移动 4. 移动后游戏结束

4.4 实现自动化测试

4.4.1 编写 feature 文件

由于文档中用户故事描述方式和 feature 文件语法高度相似，可以直接在文档基础上稍作修改。

以 S2.1 为例，简单介绍 feature 文件的编写思路：

```
@S2.1
Scenario: The player consumes
  Given the game has started with "/cucumberTest/strings/player_pellet.txt"
  And my Pacman is next to a square containing a pellet
  When I press an arrow key towards that square
  Then my Pacman can move to that square
  And I earn the points for the pellet
  And the pellet disappears from that square
```

Scenario 为场景名，用 TagS2.1 唯一标注这个场景，可以通过命令行的 tag 参数指定运行某个场景测试。

Given 为条件，用双引号标注参数，将地图文件路径以字符参数的形式传输，这一步初始化游戏并运行，完成断言点 1：游戏是否运行

And 为附加条件，在测试中主要在这一步完成地图相关断言，例如断言点 2、3。

When 描述了用户的操作动作，在测试中在这一步进行用户操作的模拟，本例中手动完成玩家的移动。

Then 表示场景的执行结果，在测试中主要测试 When 动作的效果是否被正确执行了，例如断言点 4。

在 Then 之后的两个 And 表示对测试结果的补充，在这两步中完成对动作结果进一步的测试，例如断言点 5。

其余 feature 文件可以在源码中查看，路径为：/src/test/resources/cucumberTest。

4.4.2 编写测试代码

以 S2.3 的 And my Pacman is next to a cell containing a wall 这一步骤为例，讲解对应的 java 测试代码的编写思路。代码如下：

```
@And("my Pacman is next to a cell containing a wall")
public void pacmanNextToAWall() throws ClassNotFoundException {
```

```

player = game.getPlayers().get(0);
assert player != null;
originScore = player.getScore();

playerSquare = player.getSquare();
Square nextSquare = playerSquare.getSquareAt(Direction.EAST);

MatcherAssert.assertThat(nextSquare,
CoreMatchers.instanceOf(Class.forName("nl.tudelft.jpacman.board.BoardFactory$Wall")));
}

```

场景 S2.3 的这一步骤对应的断言点为：玩家右侧方格为一个墙的实例。

首先从 `game` 实例中获取玩家列表，由于地图中只存在一个玩家，从列表的第 0 位取出玩家实例。

然后获得玩家实例当前所处的方格，使用当前所处方格的 `getSquareAt` 方法获取东侧（右侧）的方格。理想情况下，右侧方格即为墙对应的方格。

通过阅读项目源码可以发现，墙实际上是方格的继承子类，而且为存在于 `BoardFactory` 中的一个私有子类。

于是选用 `hamcrest` 框架的匹配器进行断言，将墙对应的方格匹配至墙的实例。这里需要使用 `java` 反射机制，通过类名直接获取墙的对象。

这样就完成了场景 S2.3 对应的断言点的 `java` 测试代码的编写。

其他测试代码可以参考源码，路径为：`/src/test/java/nl/tudelft/jpacman/cucumberTest/S2` 以及 `/src/test/java/nl/tudelft/jpacman/cucumberTest/S3`。

在代码的编写过程中，我还解决了以下问题：

- ◆ 在判断单位种类为幽灵时，由于地图在加载过程中，会自动将幽灵加载为 `Blinky`、`Clyde`、`Inky`、`Pinky` 中的一种（默认为 `Blinky`），所以无法通过 `Junit5` 中的 `assertEquals` 将单位类对象与 `Ghost` 类对象进行相等断言，这样无论如何都无法通过断言，于是需要是同 `hamcrest` 框架的匹配器进行断言，将对象与 `Ghost` 类进行实例匹配，这样就可以匹配到 `Ghost` 类及其子类了
- ◆ 在针对场景 S3.2 的断言点 5：移动后豆子不可见进行测试时，发现无法进行测试。因为在游戏渲染过程中，是通过单位列表顺序的覆盖从而实现不可见的效果的，而非更改方格表现的图片，这一效果并没有在游戏中实现出来。0
- ◆ 在实现故事 3 中的 `When a tick event occurs` 这一步骤时，游戏中没有提供相关接口，无法手动进行游戏移动到下一帧的操作，于是需要通过“开始游戏，手动模拟幽灵

ai 移动，暂停游戏”这一流程，从而实现一帧（也就是全体单位的一次移动）的效果模拟。

此外，本文利用了 Idea 的插件 Cucumber+ 进行 feature 文件与 java 代码间的关联，以可视化的方式快速定位场景中的每个步骤所关联到的 java 测试方法代码。

4.4.3 执行 cucumber 测试

本文对 cucumber 测试的执行使用了两种方法，junit 以及 gradle 任务。

Gradle 任务即直接执行 4.1.2 中配置的 cucumber 任务，将配置写入 build.gradle 后，在项目根目录下控制台执行 gradle cucumber 即可运行配置的 feature 文件及 glue 代码。

要使用 Junit5 执行 Cucumber 集成测试，需要编写测试启动类。使用来自 Junit Runner 的 RunWith 注解来设置测试以 Cucumber 启动，以及来自 cucumber.junit 的 CucumberOption 注解来完成 Cucumber 配置，具体代码如下：

```
@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty",
    "html:build/cucumber-reports/html-report.html",
    "json:build/cucumber-reports/json-report.json"},
    features = {"./src/test/resources/cucumberTest"},
    glue = {"nl.tudelft.jpacman.cucumberTest"})
public class CucumberTest {

}
```

这样，就可以通过 Junit 启动 CucumberTest 类进行对应的 cucumber 集成测试。

5 测试结果分析

5.1 执行测试用例结果

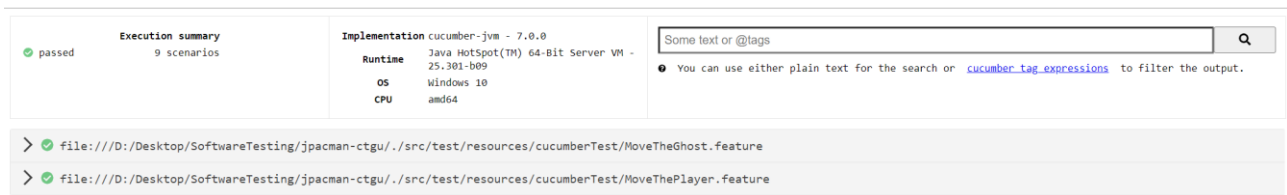
✓ Test Results	1 sec 708 ms
✓ CucumberTest	1 sec 708 ms
✓ Move The Ghost	1 sec 123 ms
✓ Move the Player	585 ms

使用 Idea 插件 Cucumber+ 可以实时可视化检查场景每个步骤的执行情况，运行 cucumber 测试启动类后查看 feature 文件（以 S2.1 为例）：

```
6
7 @S2.1
8 Scenario: The player consumes
9   Given the game has started with "/cucumberTest/strings/player_pellet.txt"
10  And my Pacman is next to a square containing a pellet
11  When I press an arrow key towards that square
12  Then my Pacman can move to that square
13  And I earn the points for the pellet
14  And the pellet disappears from that square
15
```

可以发现，所有场景所有步骤均通过测试。

通过配置启动项的 `plugins` 属性，可以生成测试报告，结果如下（例图）：



@S3.2

Scenario: The ghost moves over a square with a pellet

- ✓ **Given** the game has started with `"/cucumberTest/strings/ghost_pellet.txt"`.
- ✓ **And** a ghost is next to a cell containing a pellet
- ✓ **When** a tick event occurs
- ✓ **Then** the ghost can move to the cell with the pellet
- ✓ **And** the pellet on that cell is not visible anymore

所有场景的所有步骤均通过测试。测试报告生成位置为： `/build/cucumber-reports/`

也可以在 <https://reports.cucumber.io/reports/764883cf-fd79-4f67-90b0-be9ecef0750> 查看本次测试报告。

5.2 测试覆盖率分析

本文使用 Jacoco 进行测试覆盖率分析。

5.2.1 Jacoco 配置

在 `build.gradle` 中配置 Jacoco，首先需要下载并配置 Jacoco 插件，可参考：

<https://star.jmhui.com.cn/p1/585.html>

安装 Jacoco 插件完毕后，在 `build.gradle` 中配置添加 jacoco 报告任务。

要想使得 Jacoco 测试出 cucumber 集成测试覆盖率，需要完成以下两步：

1. 配置 cucumber 任务

在 `gradle` 的 `cucumber` 任务中添加以下内容：

```
def jacocoAgent = zipTree(configurations.jacocoAgent.singleFile).filter
{ it.name == "jacocoagent.jar" }.singleFile
jvmArgs =
["-javaagent:$jacocoAgent=destfile=$buildDir/jacoco/cucumber.exec,
append=true"]
```

完整代码可以查看 build.gradle

这一步使得在执行 cucumber 测试时，能够通过 jacoco 打桩得到 cucumber.exec 文件。

保存在 \$buildDir/jacoco/cucumber.exec

2. 配置 jacoco 报告任务

在 build.gradle 中添加以下代码：

```
jacocoTestReport {
    executionData files("${buildDir}/jacoco/cucumber.exec")
    reports {
        csv.enabled true
    }
    dependsOn("cucumber")
}
```

这一步执行第一步生成的 cucumber.exec 文件，得到 cucumber 测试的 jacoco 测试报告，

需要依赖于（dependsOn 字段）cucumber 任务。

配置完毕后，执行该任务即可获得 cucumber 集成测试覆盖率报告。

5.2.2 Jacoco 覆盖率报告分析

生成的报告如下：

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
nl.tudelft.jpacman.level	<div><div></div></div>	67%	<div><div></div></div>	64%	62 151	101 335	21 68	4 12
nl.tudelft.jpacman.npc.ghost	<div><div></div></div>	53%	<div><div></div></div>	43%	53 89	67 145	14 30	3 8
nl.tudelft.jpacman.ui	<div><div></div></div>	79%	<div><div></div></div>	65%	34 86	21 144	7 31	0 6
default	<div><div></div></div>	0%	<div><div></div></div>	0%	12 12	21 21	5 5	1 1
nl.tudelft.jpacman.sprite	<div><div></div></div>	83%	<div><div></div></div>	68%	22 70	13 113	6 38	0 5
nl.tudelft.jpacman.board	<div><div></div></div>	84%	<div><div></div></div>	74%	27 93	7 110	1 40	0 7
nl.tudelft.jpacman	<div><div></div></div>	66%	<div><div></div></div>	16%	11 30	19 52	6 24	1 2
nl.tudelft.jpacman.npc	<div><div></div></div>	34%	<div><div></div></div>	0%	5 8	10 17	2 5	0 1
nl.tudelft.jpacman.points	<div><div></div></div>	59%	<div><div></div></div>	75%	1 11	5 21	0 9	0 2
nl.tudelft.jpacman.game	<div><div></div></div>	92%	<div><div></div></div>	80%	5 24	1 45	1 14	0 3
Total	1,355 of 4,555	70%	243 of 613	60%	232 574	265 1,003	63 264	9 47

可以看到，绝大多数基本包的覆盖率都在 75% 以上，只有 ghost 和 level 两个包的覆盖率稍差，分别查看这两个包的详细报告：

nl.tudelft.jpacman.level

















Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
CollisionInteractionMap	<div><div></div></div>	0%	<div><div></div></div>	0%	19 19	46 46	7 7	1 1
MapParser	<div><div></div></div>	83%	<div><div></div></div>	78%	7 26	7 69	1 10	0 1
DefaultPlayerInteractionMap	<div><div></div></div>	0%	<div><div></div></div>	n/a	5 5	17 17	5 5	1 1
Level	<div><div></div></div>	88%	<div><div></div></div>	83%	14 55	5 105	1 15	0 1
Level.NpcMoveTask	<div><div></div></div>	33%	<div><div></div></div>	0%	2 3	6 10	1 2	0 1
LevelFactory	<div><div></div></div>	73%	<div><div></div></div>	20%	4 8	4 17	0 4	0 1
PlayerCollisions	<div><div></div></div>	80%	<div><div></div></div>	64%	4 14	5 28	1 7	0 1
CollisionInteractionMap.InverseCollisionHandler	<div><div></div></div>	0%	<div><div></div></div>	n/a	2 2	5 5	2 2	1 1
Player	<div><div></div></div>	85%	<div><div></div></div>	66%	3 11	3 24	1 8	0 1
LevelFactory.RandomGhost	<div><div></div></div>	0%	<div><div></div></div>	n/a	2 2	3 3	2 2	1 1
PlayerFactory	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 3	0 5	0 3	0 1
Pellet	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 3	0 6	0 3	0 1
Total	431 of 1,337	67%	56 of 159	64%	62 151	101 335	21 68	4 12

对于 level 包来说，主要是 CollisionInteractionMap 类没有收到测试，阅读源码发现，这个类并没有得到应用，可能是用于测试的接口类。

而对于 ghost 包来说，覆盖率低的主要原因在于四种幽灵中有三种没有得到测试，实际上在游戏的实现中，无法指定生成幽灵的种类，若是只有一个幽灵，则默认类型为 Blinky，所以难以在简单的集成测试中将所有四种幽灵都得到充分测试。

在排除了这两个覆盖率较低包的原因后可以发现，这个集成测试对项目代码的覆盖率其实还是足够高的，可以证明本次集成测试是充分完全的。

n1.tudelft.jpacman.npc.ghost

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
Inky		0%		0%	15 15	23 23	4 4	1 1
Clyde		0%		0%	12 12	22 22	3 3	1 1
Pinky		0%		0%	11 11	13 13	3 3	1 1
GhostFactory		36%		n/a	3 5	3 7	3 5	0 1
Navigation		93%		86%	6 28	3 49	0 6	0 1
Blinky		80%		68%	5 11	2 13	0 3	0 1
Navigation.Node		92%		100%	1 6	1 13	1 5	0 1
GhostColor		100%		n/a	0 1	0 5	0 1	0 1
Total	321 of 687	53%	67 of 118	43%	53 89	67 145	14 30	3 8

6 总结与回顾

在本学期的软件测试课程中，我学习到了单元测试和集成测试的实现方法，掌握了 Junit, Cucumber, pitest, jacoco 等多种优秀的软件测试工具，具备了独立设计边界值分析，等价类分析，决策表分析，基路径分析等多种测试用例构建方法及软件测试手段的能力。在对 Jpacman 项目的不断深入学习的过程中，我也加强了自身对完整陌生项目代码的理解阅读能力和测试能力。可以说初步具备了软件测试人员的基本素养。

参考文献

- [1] <https://cucumber.io/>
- [2] <https://star.jmhui.com.cn/p1/zy.html>
- [3] <https://github.com/Luanderr21/jpacman-testing-with-cucumber>
- [4] <https://stackoverflow.com/questions/58536904/code-coverage-for-cucumber-tests-using-jacoco>
- [5] <https://blog.csdn.net/lvshuangtao/article/details/52495977>