1. **《软件测试》**

**软件测试实验八：综合测试3**

**姓 名： 胡国昌 学 号： 202210120518**

**院 系： 计算机与信息学院 专 业：计算机与科学**

**实 验 室： B1-119 实验日期： 2025.4.15**

**总评成绩： 审阅教师： 杨青**

# 一、课程目标

****目标2****： 能够运用功能测试、基于控制流和数据流的测试等软件测试的核心技术和原理，结合相关文献，对测试问题及其影响因素进行分析或计算，得出有效结论。

# 二、实验目的：

1. 学习在一定规模实际项目中综合测试方法
2. 学习研读开源代码的技术
3. 熟悉项目构建工具gradle
4. 理解项目类结构
5. 掌握模拟框架mockito

# 三、实验步骤

1. 在实验六的代码基础上， 完成下面要求，提交到自己的代码仓库
2. 使用模拟框架，设计测试用例并实现
3. 编写测试报告，提交到雨课堂“软件测试实验八”

# 实验题目

1. 针对 level.MapParser 类， 首先从阳光行为开始考虑，棋盘文件包含期望的字符。使用 Mockito 模拟工厂方法，使用 Mockito 验证读取的地图与工厂具有正确的交互。
2. 扩展测试用例考虑异常错误场景，强制抛出正确的异常。很容易考虑到异常情景，但是可能有更重要的情况需要考虑，比如从文件中读取数据的时候可能发生的错误。
3. 实现测试用例，并执行覆盖率检查和代码规范检查。

## 代码实现

解析字符串列表－行列转换正确

解析输入流－正确转换解析单行地图－验证Board尺寸

解析不存在的资源文件－抛出异常

解析最小地图－创建成功

解析行长度不一致－抛出异常

解析幽灵－验证占据方法调用

解析豆子－验证占据方法调用

解析无效字符－抛出PacmanConfigurationException

解析空格字符－创建地面解析行长度不一致－抛出异常

解析空行地图－抛出异常解析null列表－抛出异常

解析多玩家位置－收集所有起始点

解析有效地图－正确统计所有元素

解析null输入－抛出NPE解析空地图－抛出异常

package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.PacmanConfigurationException;

import nl.tudelft.jpacman.board.Board;

import nl.tudelft.jpacman.board.BoardFactory;

import nl.tudelft.jpacman.board.Square;

import nl.tudelft.jpacman.npc.Ghost;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

import org.mockito.ArgumentCaptor;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.ArgumentMatchers.any;

import static org.mockito.ArgumentMatchers.argThat;

import static org.mockito.Mockito.\*;

// 声明测试类并使用JUnit5的显示名称注解

@DisplayName("地图解析器单元测试")

public class MapParserTest {

// 被测试的地图解析器实例

private MapParser mapParser;

// 使用Mockito模拟依赖项

@Mock

private LevelFactory levelFactory; // 关卡工厂模拟

@Mock

private BoardFactory boardFactory; // 面板工厂模拟

@Mock

private Board mockBoard; // 模拟游戏面板

@Mock

private Level mockLevel; // 模拟游戏关卡

// 每个测试方法执行前的初始化操作

@BeforeEach

@DisplayName("初始化测试环境")

void setUp() {

// 初始化Mockito注解

MockitoAnnotations.initMocks(this);

// 创建地图解析器实例，注入模拟依赖

mapParser = new MapParser(levelFactory, boardFactory);

// 创建模拟的棋盘元素

Square wall = mock(Square.class);

Square ground = mock(Square.class);

Ghost ghost = mock(Ghost.class);

// 配置模拟对象的行为

when(boardFactory.createWall()).thenReturn(wall); // 当创建墙壁时返回模拟对象

when(boardFactory.createGround()).thenReturn(ground); // 创建地面返回模拟对象

when(levelFactory.createGhost()).thenReturn(ghost); // 创建幽灵返回模拟对象

when(levelFactory.createPellet()).thenReturn(mock(Pellet.class)); // 创建豆子返回模拟

when(boardFactory.createBoard(any())).thenReturn(mockBoard); // 创建面板返回模拟

when(levelFactory.createLevel(any(), any(), any())).thenReturn(mockLevel); // 创建关卡返回模拟

// 配置幽灵占位方法不执行任何操作

doNothing().when(ghost).occupy(any());

}

// 测试用例：成功解析有效地图

@SuppressWarnings("unchecked")

@Test

@DisplayName("解析有效地图 - 正确统计所有元素")

void testParseMapSuccess() {

// 准备测试数据（3行不同结构的地图）

List < String > map = Arrays.asList(

"# #.#",

"#PGG ",

"##. #"

);

// 执行解析操作

mapParser.parseMap(map);

// 验证工厂方法调用次数符合预期

verify(boardFactory, times(7)).createWall(); // 总共有7个墙壁符号#

verify(boardFactory, times(8)).createGround(); // 总共有8个地面符号（空格和P、G所在位置）

verify(levelFactory, times(2)).createGhost(); // 两个G字符

verify(levelFactory, times(2)).createPellet(); // 两个.字符

// 验证关卡创建时的参数是否正确

ArgumentCaptor < List < Ghost >> ghostCaptor = ArgumentCaptor.forClass(List.class); // 捕获幽灵列表

ArgumentCaptor < List < Square >> startCaptor = ArgumentCaptor.forClass(List.class); // 捕获起始点

verify(levelFactory).createLevel(eq(mockBoard), ghostCaptor.capture(), startCaptor.capture());

// 断言幽灵数量和起始点数量

assertEquals(2, ghostCaptor.getValue().size()); // 应有两个幽灵

assertEquals(1, startCaptor.getValue().size()); // 应有一个玩家起始点（第二个P被统计？可能需要确认逻辑）

}

// 参数化测试：验证各种非法字符

@ParameterizedTest

@ValueSource(chars = {

'X',

'\t',

'0',

'@'

}) // 测试四个非法字符

@DisplayName("解析无效字符 - 抛出PacmanConfigurationException")

void testInvalidCharacters(char invalidChar) {

// 创建包含非法字符的地图

char[][] map = {

{

invalidChar

}

};

// 验证解析时抛出预期异常

assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap(map));

}

// 空地图测试（可能需要调整预期异常类型）

@Test

@DisplayName("解析空地图 - 抛出异常")

void testEmptyMap() {

// 传入空数组，预期数组越界异常（可能需要改为更合适的异常类型）

assertThrows(ArrayIndexOutOfBoundsException.class, () -> mapParser.parseMap(new char[0][]));

}

// 单行地图验证

@Test

@DisplayName("解析单行地图 - 验证Board尺寸")

void testSingleRowMap() {

// 创建3列的单行地图

char[][] map = {

{

'#',

'P',

' '

}

};

mapParser.parseMap(map);

// 验证创建的面板尺寸为1行3列

verify(boardFactory).createBoard(argThat(grid -> grid.length == 1 && grid[0].length == 3));

}

// 多个玩家起始点测试

@SuppressWarnings("unchecked")

@Test

@DisplayName("解析多玩家位置 - 收集所有起始点")

void testMultiplePlayers() {

// 两个P字符

char[][] map = {

{

'P'

},

{

'P'

}

};

mapParser.parseMap(map);

// 捕获起始点列表

ArgumentCaptor < List < Square >> startCaptor = ArgumentCaptor.forClass(List.class);

verify(levelFactory).createLevel(any(), any(), startCaptor.capture());

// 验证有两个起始点

assertEquals(2, startCaptor.getValue().size());

}

// 字符串列表格式测试

@Test

@DisplayName("解析字符串列表 - 行列转换正确")

void testParseFromStringList() {

// 创建3x5的地图

List < String > lines = Arrays.asList("#####", "#P.G#", "#####");

mapParser.parseMap(lines);

// 验证中间行的特定位置元素

verify(boardFactory).createBoard(argThat(grid -> grid[1][1] != null && // 玩家位置

grid[3][1] != null // 豆子位置

));

}

// 行长度不一致测试

@Test

@DisplayName("解析行长度不一致 - 抛出异常")

void testInconsistentRowLength() {

// 第一行4字符，第二行2字符

List < String > lines = Arrays.asList("####", "##");

assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap(lines));

}

// 行长度不一致测试

@Test

@DisplayName("解析行长度不一致 - 抛出异常")

void testListEmptyMap() {

// 第一行4字符，第二行2字符

// List<String> lines = list.of();

List < String > lines = Collections.emptyList();

assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap(lines));

}

// 空指针测试

@Test

@DisplayName("解析null输入 - 抛出NPE")

void testNullMap() {

// 传入null地图数组

assertThrows(NullPointerException.class, () -> mapParser.parseMap((char[][]) null));

}

// 豆子占位测试

@Test

@DisplayName("解析豆子 - 验证占据方法调用")

void testPelletOccupation() {

// 单个豆子地图

char[][] map = {

{

'.'

}

};

Pellet pellet = mock(Pellet.class);

when(levelFactory.createPellet()).thenReturn(pellet);

mapParser.parseMap(map);

// 验证豆子的occupy方法被调用

verify(pellet).occupy(any(Square.class));

}

// 幽灵占位测试

@Test

@DisplayName("解析幽灵 - 验证占据方法调用")

void testGhostOccupation() {

Ghost ghost = mock(Ghost.class);

when(levelFactory.createGhost()).thenReturn(ghost);

char[][] map = {

{

'G'

}

};

mapParser.parseMap(map);

// 验证幽灵的occupy方法被调用

verify(ghost).occupy(any(Square.class));

}

// 最小有效地图测试

@Test

@DisplayName("解析最小地图 - 创建成功")

void testMinimalValidMap() {

char[][] map = {

{

'#'

}

};

assertNotNull(mapParser.parseMap(map)); // 验证解析成功

verify(boardFactory).createWall(); // 应创建墙壁

}

// 空格字符处理测试

@Test

@DisplayName("解析空格字符 - 创建地面")

void testSpaceCharacter() {

char[][] map = {

{

' '

}

};

mapParser.parseMap(map);

verify(boardFactory).createGround(); // 验证创建地面

verify(levelFactory, never()).createPellet(); // 确保没有创建豆子

}

// null列表测试

@Test

@DisplayName("解析null列表 - 抛出异常")

void testNullListMap() {

assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap((List < String > ) null));

}

// 输入流解析测试

@Test

@DisplayName("解析输入流 - 正确转换")

void testParseInputStream() throws IOException {

// 模拟输入流数据

String input = "###\n#P#\n###";

InputStream stream = new ByteArrayInputStream(input.getBytes());

mapParser.parseMap(stream);

// 验证中间行的玩家位置

verify(boardFactory).createBoard(argThat(grid -> grid.length == 3 && // 3行

grid[0][1] != null // 可能需要确认索引是否正确（第二行中间位置）

));

}

// 不存在的资源文件测试

@Test

@DisplayName("解析不存在的资源文件 - 抛出异常")

void testNonExistingResource() {

assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap("/nonexistent.txt"));

}

// 空行地图测试

@Test

@DisplayName("解析空行地图 - 抛出异常")

void testEmptyLineMap() {

List < String > lines = Collections.singletonList("");

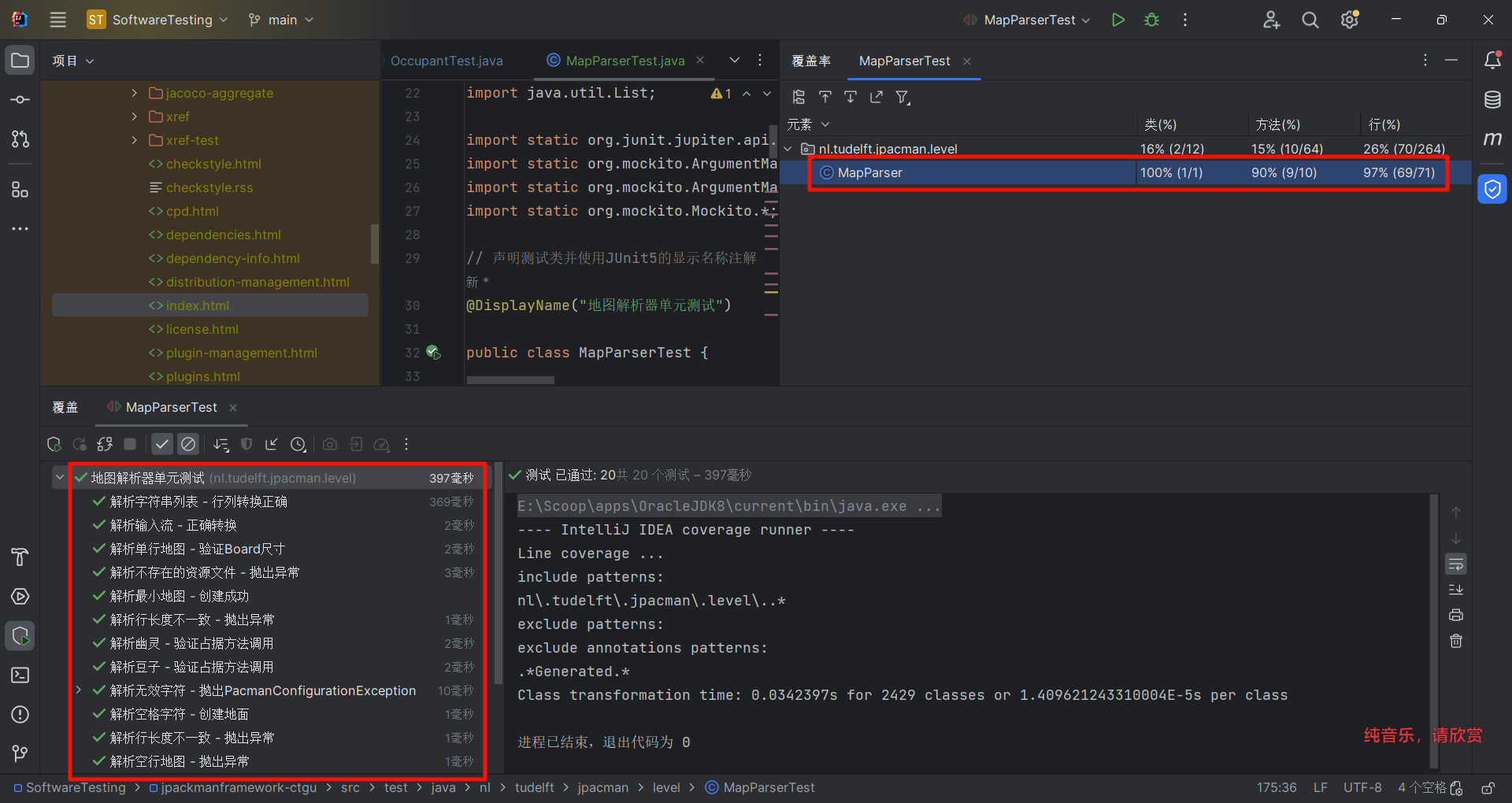
assertThrows(PacmanConfigurationException.class, () -> mapParser.parseMap(lines));

}

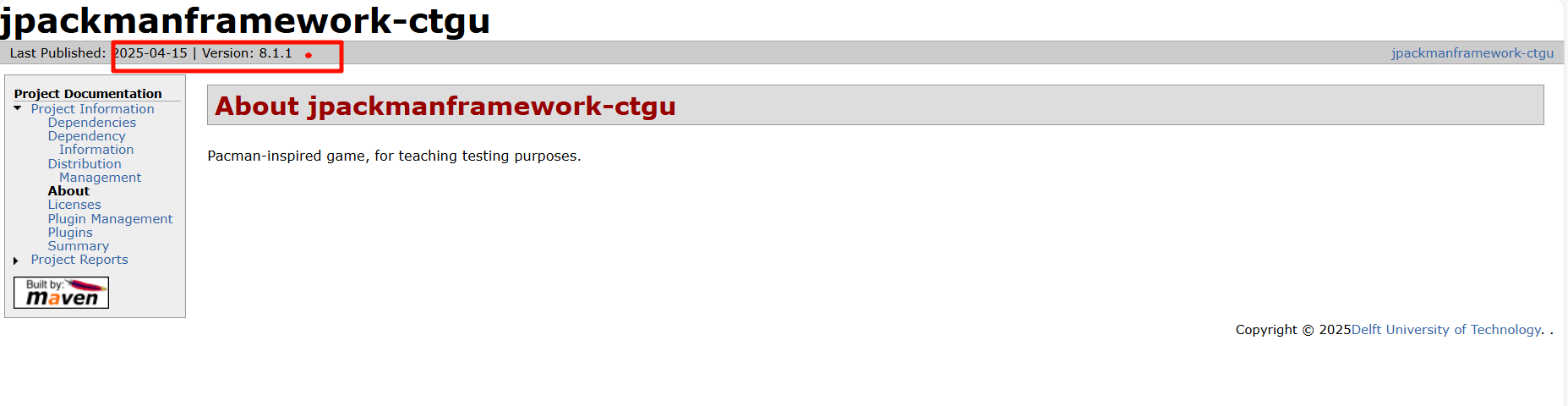
}

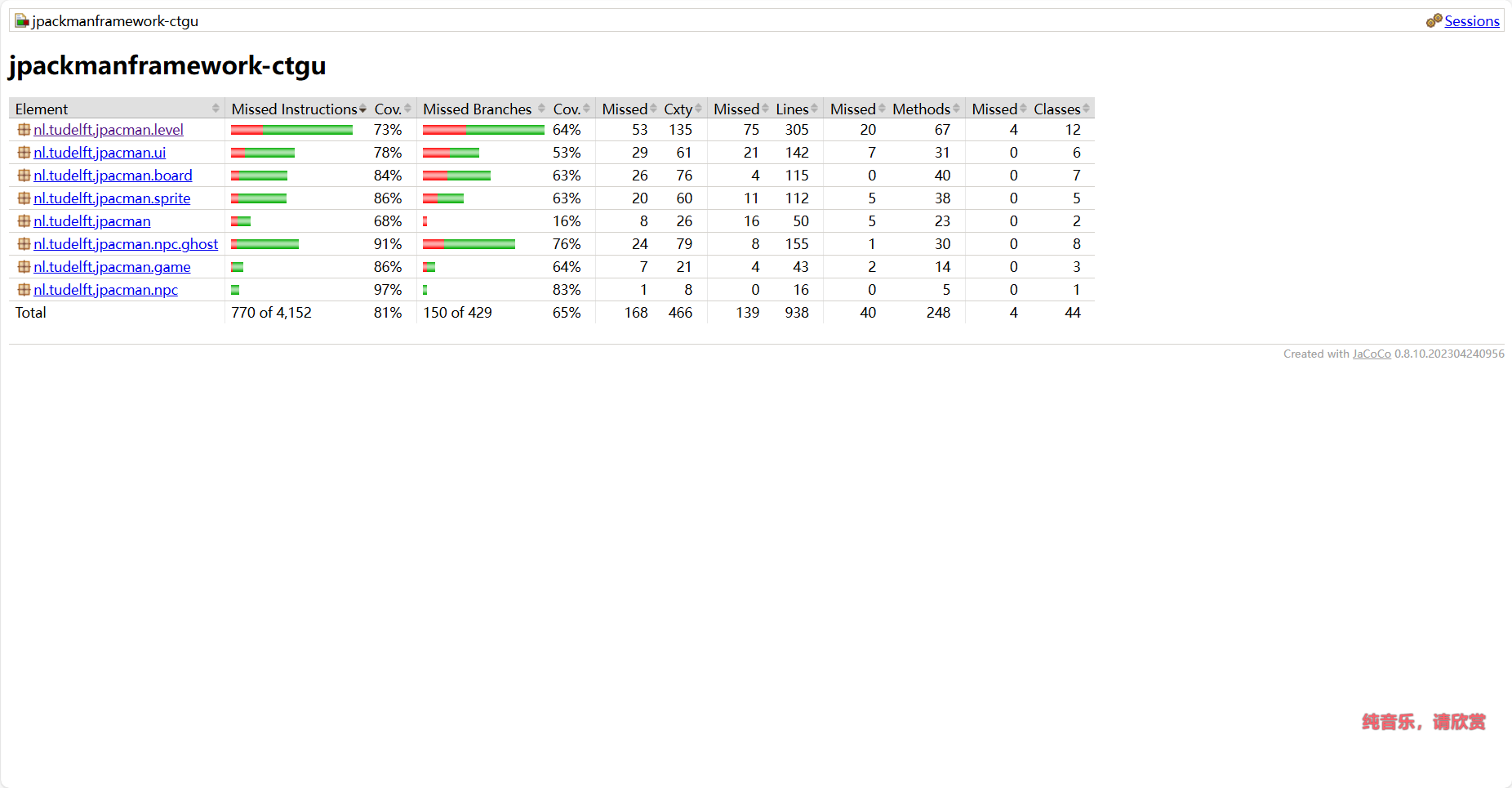
## 测试结果

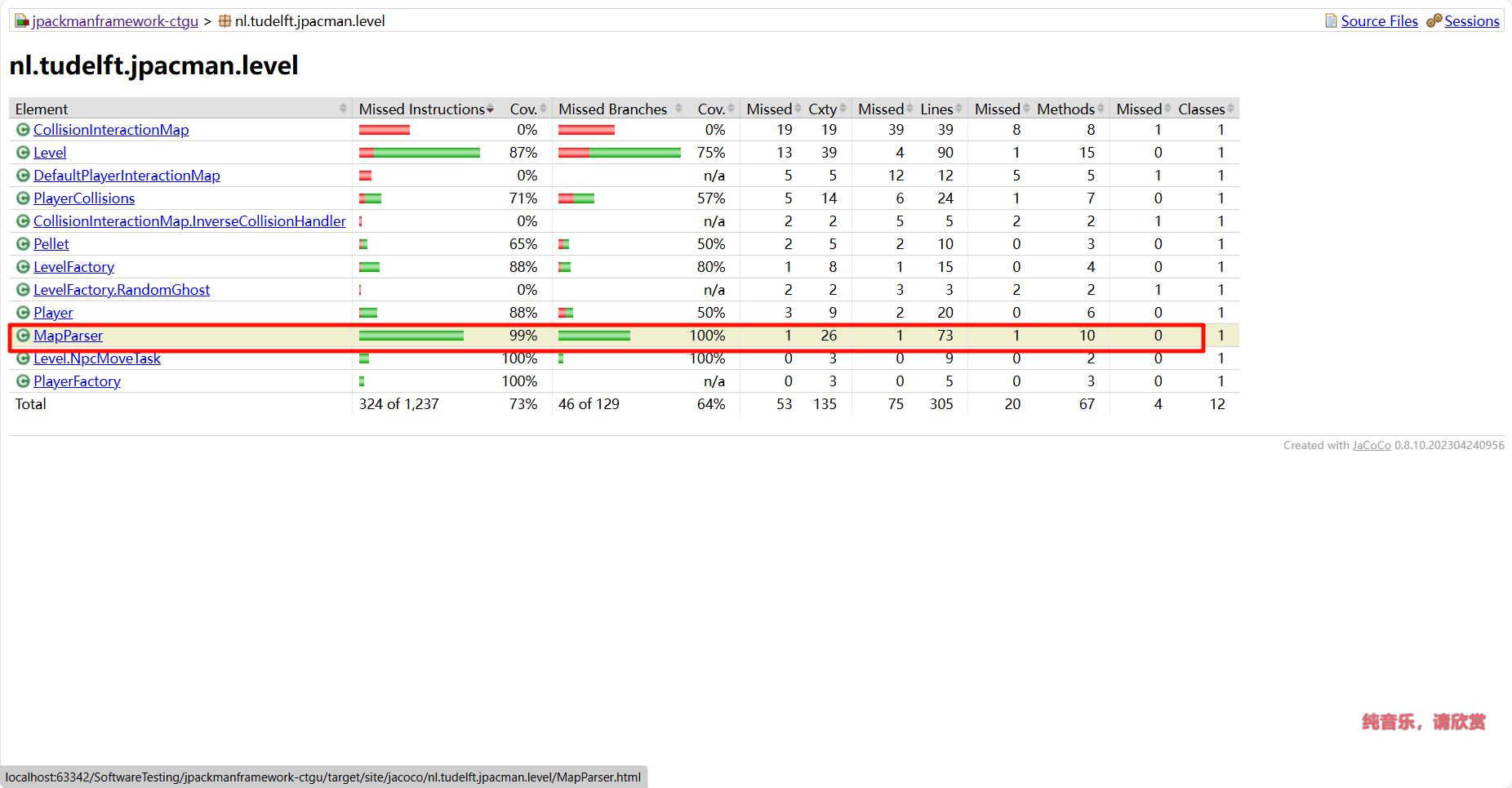
### 覆盖率



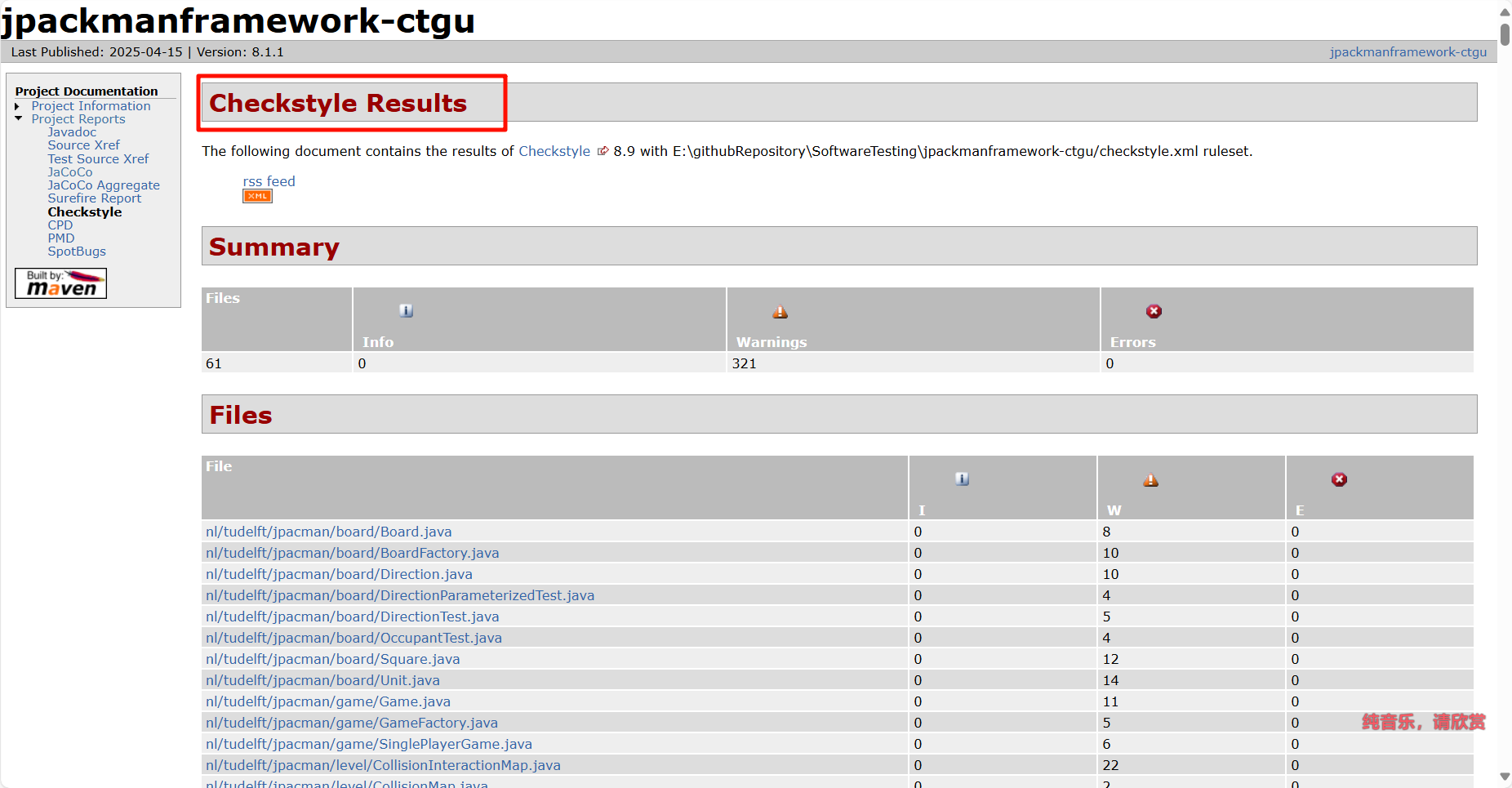
### Jacoco覆盖率







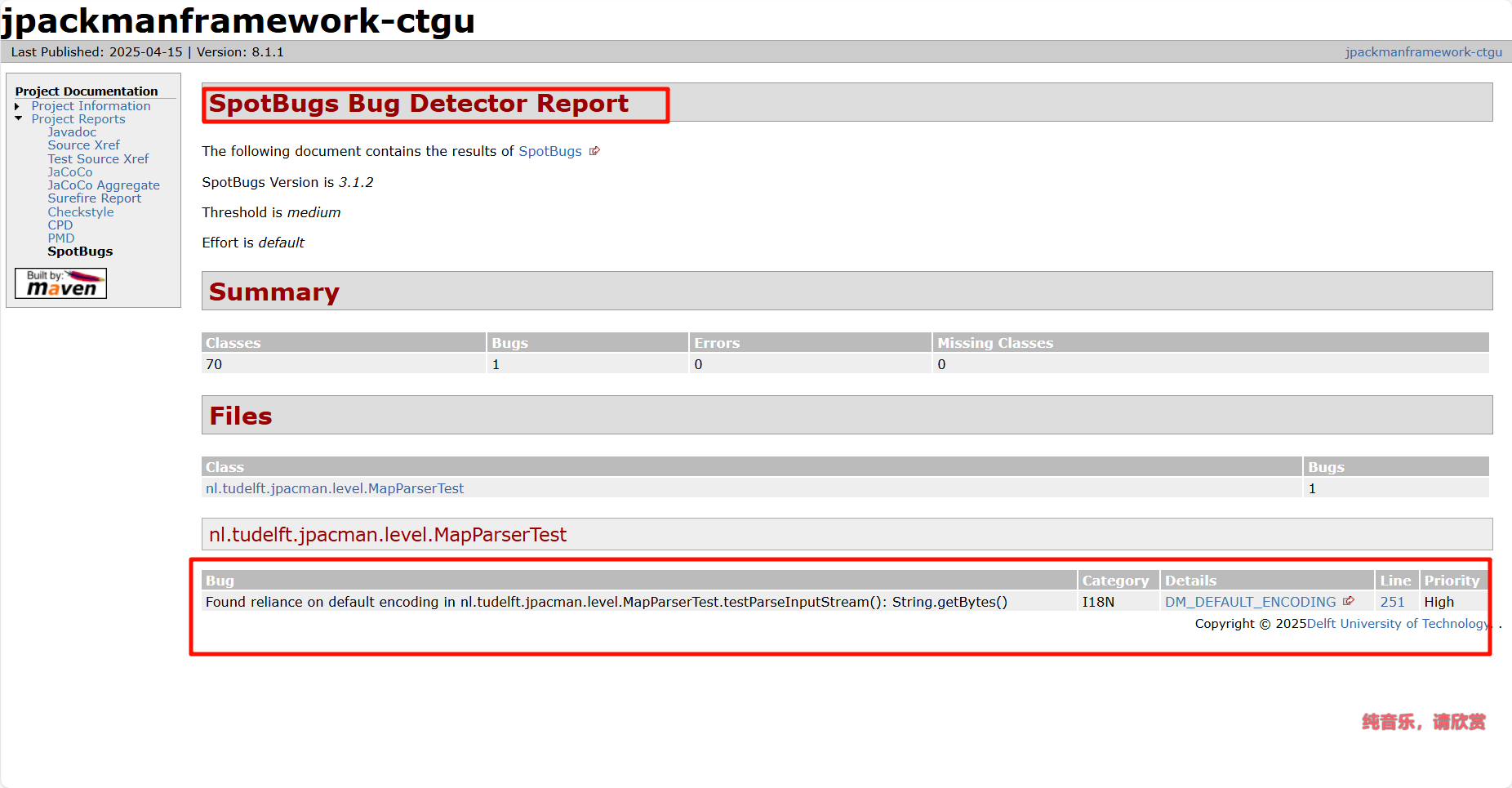
### Checkstyle



### Pmd



### Spotbugs



# 结论分析与体会

关于这次对于地图类的测试 我学会了使用mock来虚拟调用的对象 简化了检查被调用的类测试 屏蔽外部依赖 避免因外部问题导致测试失败。​​  
​简化复杂依赖:当依赖对象构造复杂（如需要多步初始化）时，直接 Mock 其行为，减少测试代码冗余。  
​提升覆盖率:通过 Mock 不同返回值，更容易覆盖所有代码分支（如 if-else 逻辑）。

关于pacman这个项目 我深入的分析了一些类 了解了一些工厂类 怎么创建相应的对象

关于要做的测试类 我也深入的了解 在老师的提醒下 发现了一些源代码的问题 并在文章中解决了这些问题

在面对整个项目的测试 我认为首先要读懂核心类的代码逻辑 让后对代码做测试

# 仓库地址

https://github.com/xieyangyuyue/SoftwareTesting

[xieyangyuyue/SoftwareTesting: 软件测试--java (github.com)](https://github.com/xieyangyuyue/SoftwareTesting)