1. **《软件测试》**

**软件测试实验七：综合测试2**

**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： 2025.4.11**

**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. 熟悉分析项目结构方法

# 三、实验步骤

1. 在实验六的代码基础上， 完成下面题目1，提交到自己的代码仓库
2. 在完成题目1的基础上完成题目2，题目2选做（学有余力者做）
3. 编写测试报告，提交到雨课堂“软件测试实验七”

# 实验题目

## 题目一

1. Jpacman有一些重要的行为没有经过测试，单元测试测试：Ghost(幽灵)的移动行为。
2. 查看nl.tudelft.jpacman.npc.ghost包下的Clyde类，认真阅读该类的Javadoc描述，理解这类Ghost的行为方式。
3. 查看Clyde类的 Optional nextAiMove()”方法，考虑测试用例以确保Clyde按预期行为工作。
4. 实例化Clyde对象  
   在test模块java/nl/tudelft/jpacman包中的GhostMapParser类，它是提供快速生成地图的辅助工具类。比如，你提供下面这样三行字符串，将自动生成一个3X12大小，且含有一个玩家和一个Clyde幽灵的地图。玩家和Clyde之间间隔8个Square。 {“############”, “ #P C#”, “############”}, GhostMapParser的parseMap()方法接收上述字符串数组，并返回一个Level对象。而这个Level对象管理了Board、Player、Ghost等几乎游戏涉及到的所有模型对象。  
   比如，如果你的测试涉及到Player，需要利用Player的PlayerFactory生成Player对象。然后再调用Level的RegisterPlayer来注册这个新创建的Player对象，并调用Player的setDirection方法来设置一个初始的行走方向。 除了创建Player，通常还需要Ghost。四种Ghost对象在地图解析时，就已经创建好，可以通过调用Navigation的findUnitInBoard()方法来获取.
5. 实现这个ClydeTest测试类时，需要构建游戏运行的所有所需对象。这些对象包括PacManSprites（用于角色显示）、PlayerFactory（用于构造Player）、GhostFactory（提供给LevelFactory），BoardFactory（游戏场景）、GhostMapParser（地图解析）。  
   思考以上对象构造过程，建议利用JUnit5的[@BeforeEach](https://github.com/BeforeEach" \o "@BeforeEach)或[@BeforeAll](https://github.com/BeforeAll" \o "@BeforeAll)来为不同的测试方法创建统一的所需对象集。 在ClydeTest类中至少测试4个情况下的行为； 为你的每个函数和类增加javadoc注解。
6. 为了测试生成Clyde魔鬼，可以增加一个地图解析器，[Clyde魔鬼地图解析器](https://star.jmhui.com.cn/u/cms/www/202205/0915094561cy.rar)

## 题目2

1. 创建InkyTest测试类
2. 查看Inky幽灵类，可与Clyde幽灵类作对比。特别注意一下它的nextAiMove()方法有什么不同。
3. 注意GhostMapParser的addSquare函数对Inky幽灵的处理，请补充完整。
4. 实现对Inky测试至少5种情况下的行为。
5. 每个函数和类增加javadoc注解

## 代码实现

### Clyde存在bug及调整

根据下述测试 发现源代码有bug 在clyde距离玩家小于8个时候 Clyde改变方向 这时会产生一个bug 如果反方向有墙 clyde仍然会向东 而不是没有路径

@Test

@DisplayName("Clyde离Player距离小于8个方块 但是有墙")

public void testClydeSmallFromPlayerSOUTH() {

// 创建一个地图，玩家和Clyde位于两端

List < String > map = Arrays.asList(

"############",

"#P#######C##",

"#..........#"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会向西移动，靠近玩家

// assertThat(clyde.nextAiMove()).isEmpty();

assertThat(clyde.nextAiMove()).contains(Direction.SOUTH);

}

}

### 调整Clyde代码

@Override

public Optional < Direction > nextAiMove() {

assert hasSquare();

Unit nearest = Navigation.findNearest(Player.class, getSquare());

if (nearest == null) {

return Optional.empty();

}

assert nearest.hasSquare();

Square target = nearest.getSquare();

List < Direction > path = Navigation.shortestPath(getSquare(), target, this);

if (path != null && !path.isEmpty()) {

Direction direction = path.get(0);

if (path.size() <= SHYNESS) {

// 检查反方向是否可行

Direction escapeDir = OPPOSITES.get(direction);

Square nextSquare = getSquare().getSquareAt(escapeDir);

if (nextSquare != null && nextSquare.isAccessibleTo(this)) {

return Optional.of(escapeDir);

} else {

return Optional.empty();

}

} else {

return Optional.of(direction);

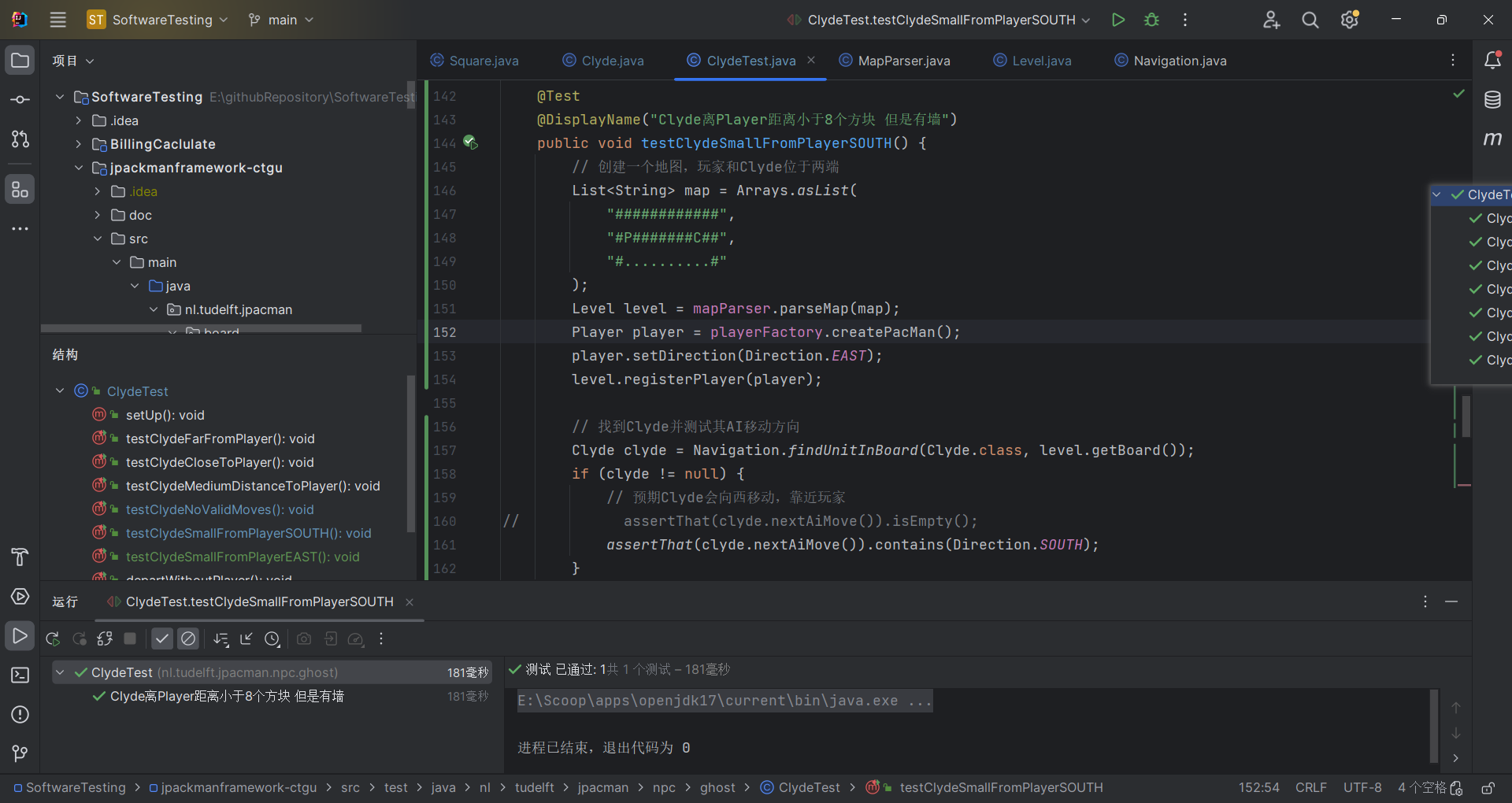
}

}

return Optional.empty();

}

测试可得



### 测试情形：

#### clyde

1. Clyde离Player距离大于8个方块
2. Clyde离Player距离小于8个方块；
3. Clyde与Player处于适当距离的行为
4. Clyde与Player直接没有路径
5. Clyde没有Player
6. Clyde离Player距离小于8个方块但是有墙
7. Clyde离Player距离小于8个方块但是有墙方向东

#### Inky

1. 当Player面朝北方时，Inky的目标点计算包含经典偏移
2. 当Inky、Blink和Player在同一直线时，Inky应朝Player方向移动
3. 当Blinky远离Player时，Inky朝延长路径方向移动
4. 当Blinky不存在时，Inky无法计算路径
5. 当Player不存在时，Inky无目标点
6. 当Player前方路径被阻挡时，Inky向Blinky移动

### 代码详情

#### Clyde.java

package nl.tudelft.jpacman.npc.ghost;

import nl.tudelft.jpacman.board.Direction;

import nl.tudelft.jpacman.board.Square;

import nl.tudelft.jpacman.board.Unit;

import nl.tudelft.jpacman.level.Player;

import nl.tudelft.jpacman.npc.Ghost;

import nl.tudelft.jpacman.sprite.Sprite;

import java.util.EnumMap;

import java.util.List;

import java.util.Map;

import java.util.Optional;

/\*\*

\* 经典Pac-Man幽灵Clyde（别名Pokey）的实现。

\* <p>

\* Clyde的特性：

\* - 最后一个离开重生区域的幽灵，通常独自巡逻迷宫左下角

\* - 当距离Pac-Man超过8格时，像Blink一样精准追踪

\* - 当距离小于等于8格时，切换为逃跑模式

\* - 行为具有一定随机性，危险系数较高

\* <p>

\* AI逻辑说明：

\* 使用双模式AI，通过路径长度判断距离状态。基于策略Wiki的行为描述实现。

\*

\* @author Jeroen Roosen

\* @see <a href="http://strategywiki.org/wiki/Pac-Man/Getting\_Started">行为策略参考</a>

\*/

public class Clyde extends Ghost {

/\*\*

\* 警戒距离阈值（单位：网格），当与Pac-Man距离≤8时切换行为

\*/

private static final int SHYNESS = 8;

/\*\*

\* 移动间隔随机性参数（毫秒），增加幽灵行为动态性

\*/

private static final int INTERVAL\_VARIATION = 50;

/\*\*

\* 基础移动间隔（毫秒）

\*/

private static final int MOVE\_INTERVAL = 250;

/\*\*

\* 方向反向映射表，用于逃跑时取反方向

\*/

private static final Map < Direction, Direction > OPPOSITES = new EnumMap < > (Direction.class);

// 静态初始化方向反向映射

static {

OPPOSITES.put(Direction.NORTH, Direction.SOUTH);

OPPOSITES.put(Direction.SOUTH, Direction.NORTH);

OPPOSITES.put(Direction.WEST, Direction.EAST);

OPPOSITES.put(Direction.EAST, Direction.WEST);

}

/\*\*

\* 构造函数，初始化Clyde的精灵和移动参数

\*

\* @param spriteMap 包含各方向精灵动画的映射

\*/

public Clyde(Map < Direction, Sprite > spriteMap) {

super(spriteMap, MOVE\_INTERVAL, INTERVAL\_VARIATION);

}

/\*\*

\* 计算下一步移动方向（AI核心逻辑）

\*

\* @return Optional包装的移动方向，空表示无有效移动

\*/

// @Override

// public Optional<Direction> nextAiMove() {

// // 断言确保幽灵当前位于某个方格上

// assert hasSquare();

//

// // 寻找最近的玩家单位

// Unit nearest = Navigation.findNearest(Player.class, getSquare());

// if (nearest == null) {

// return Optional.empty(); // 无玩家可追踪

// }

//

// // 获取玩家所在方格并计算最短路径

// assert nearest.hasSquare();

// Square target = nearest.getSquare();

// List<Direction> path = Navigation.shortestPath(getSquare(), target, this);

//

// if (path != null && !path.isEmpty()) {

// Direction direction = path.get(0); // 取路径第一步方向

//

// // 根据距离阈值决定行为模式

// if (path.size() <= SHYNESS) {

// // 近距离模式：朝反方向移动（逃跑）

// return Optional.ofNullable(OPPOSITES.get(direction));

//// Optional<Direction> direction1= Optional.ofNullable(OPPOSITES.get(direction));

//

// } else {

// // 远距离模式：向玩家方向移动（追击）

// return Optional.of(direction);

// }

// }

//

// return Optional.empty(); // 无有效路径

// }

@Override

public Optional < Direction > nextAiMove() {

assert hasSquare();

Unit nearest = Navigation.findNearest(Player.class, getSquare());

if (nearest == null) {

return Optional.empty();

}

assert nearest.hasSquare();

Square target = nearest.getSquare();

List < Direction > path = Navigation.shortestPath(getSquare(), target, this);

if (path != null && !path.isEmpty()) {

Direction direction = path.get(0);

if (path.size() <= SHYNESS) {

// 检查反方向是否可行

Direction escapeDir = OPPOSITES.get(direction);

Square nextSquare = getSquare().getSquareAt(escapeDir);

if (nextSquare != null && nextSquare.isAccessibleTo(this)) {

return Optional.of(escapeDir);

} else {

return Optional.empty();

}

} else {

return Optional.of(direction);

}

}

return Optional.empty();

}

}

#### clydeTest.java

package nl.tudelft.jpacman.npc.ghost;

import nl.tudelft.jpacman.board.BoardFactory;

import nl.tudelft.jpacman.board.Direction;

import nl.tudelft.jpacman.level.\*;

import nl.tudelft.jpacman.sprite.PacManSprites;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import static org.assertj.core.api.Assertions.assertThat;

/\*\*

\* 测试Clyde在不同情况下的行为

\*/

public class ClydeTest {

private MapParser mapParser; // 用于解析地图的工具

private PlayerFactory playerFactory; // 用于创建玩家的工厂

@BeforeEach

public void setUp() {

// 初始化 sprites、boardFactory、ghostFactory 和 levelFactory

PacManSprites sprites = new PacManSprites();

BoardFactory boardFactory = new BoardFactory(sprites);

GhostFactory ghostFactory = new GhostFactory(sprites);

LevelFactory levelFactory = new LevelFactory(sprites, ghostFactory);

// 初始化 mapParser 和 playerFactory

mapParser = new GhostMapParser(levelFactory, boardFactory, ghostFactory);

playerFactory = new PlayerFactory(sprites);

}

/\*\*

\* 测试Clyde离玩家远距离的行为

\*/

@Test

@DisplayName("Clyde离Player距离大于8个方块")

public void testClydeFarFromPlayer() {

// 创建一个地图，玩家和Clyde位于两端

List < String > map = Arrays.asList(

"############",

"#P........C#",

"############"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会向西移动，靠近玩家

assertThat(clyde.nextAiMove()).contains(Direction.WEST);

}

}

/\*\*

\* 测试Clyde离玩家近距离的行为

\*/

@Test

@DisplayName("Clyde离Player距离小于8个方块")

public void testClydeCloseToPlayer() {

// 创建一个地图，玩家和Clyde靠近

List < String > map = Arrays.asList(

"############",

"#P...C #",

"#.....######"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会移动，但方向可能随机

assertThat(clyde.nextAiMove()).isNotEmpty();

assertThat(clyde.nextAiMove()).contains(Direction.EAST);

}

}

/\*\*

\* 测试Clyde和玩家处于适当距离的行为

\*/

@Test

@DisplayName("Clyde与Player处于适当距离的行为")

public void testClydeMediumDistanceToPlayer() {

// 创建一个地图，玩家和Clyde相距一段距离

List < String > map = Arrays.asList(

"############",

"#P......C #",

"############"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会向东移动

assertThat(clyde.nextAiMove()).contains(Direction.EAST);

}

}

/\*\*

\* 测试Clyde触碰边界的行为

\*/

@Test

@DisplayName("Clyde与Player直接没有路径")

public void testClydeNoValidMoves() {

// 创建一个地图，Clyde被墙包围

List < String > map = Arrays.asList(

"############",

"##P######C##",

"############"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde无法移动

assertThat(clyde.nextAiMove()).isEmpty();

// assertThat(clyde.nextAiMove()).contains(Direction.EAST);

}

}

/\*\*

\* 测试Clyde离玩家远距离的行为

\*/

@Test

@DisplayName("Clyde离Player距离小于8个方块 但是有墙")

public void testClydeSmallFromPlayerSOUTH() {

// 创建一个地图，玩家和Clyde位于两端

List < String > map = Arrays.asList(

"############",

"#P#######C##",

"#..........#"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会向南移动，靠近玩家

// assertThat(clyde.nextAiMove()).isEmpty();

assertThat(clyde.nextAiMove()).contains(Direction.SOUTH);

}

}

@Test

@DisplayName("Clyde离Player距离小于8个方块 但是有墙 方向东")

public void testClydeSmallFromPlayerEAST() {

// 创建一个地图，玩家和Clyde位于两端

List < String > map = Arrays.asList(

"############",

"#P.......C##",

"#..........#"

);

Level level = mapParser.parseMap(map);

Player player = playerFactory.createPacMan();

player.setDirection(Direction.EAST);

level.registerPlayer(player);

// 找到Clyde并测试其AI移动方向

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

if (clyde != null) {

// 预期Clyde会向东移动，远离玩家

assertThat(clyde.nextAiMove()).isEmpty();

// assertThat(clyde.nextAiMove()).contains(Direction.EAST);

}

}

/\*\*

\* 测试当关卡中没有存活玩家时Clyde（幽灵）的行为

\*/

@Test

@DisplayName("Clyde没有Player")

// 测试用例显示名称，说明测试场景

void departWithoutPlayer() {

// 构建测试地图（ASCII结构）

List < String > map = Arrays.asList(

"##############", // 顶部围墙

"#.C...........", // 中间行：C表示Clyde，.为路径

"##############" // 底部围墙

);

// 解析地图生成关卡对象

Level level = mapParser.parseMap(map);

// 在关卡中查找Clyde实例

Clyde clyde = Navigation.findUnitInBoard(Clyde.class, level.getBoard());

// 验证Clyde实例存在

assertThat(clyde).isNotNull();

// 验证Clyde初始方向为EAST（东）

assertThat(clyde.getDirection()).isEqualTo(Direction.valueOf("EAST"));

// 确认关卡中没有存活玩家（地图无P标识符）

assertThat(level.isAnyPlayerAlive()).isFalse();

// 获取Clyde的下一步AI移动决策

Optional < Direction > opt = clyde.nextAiMove();

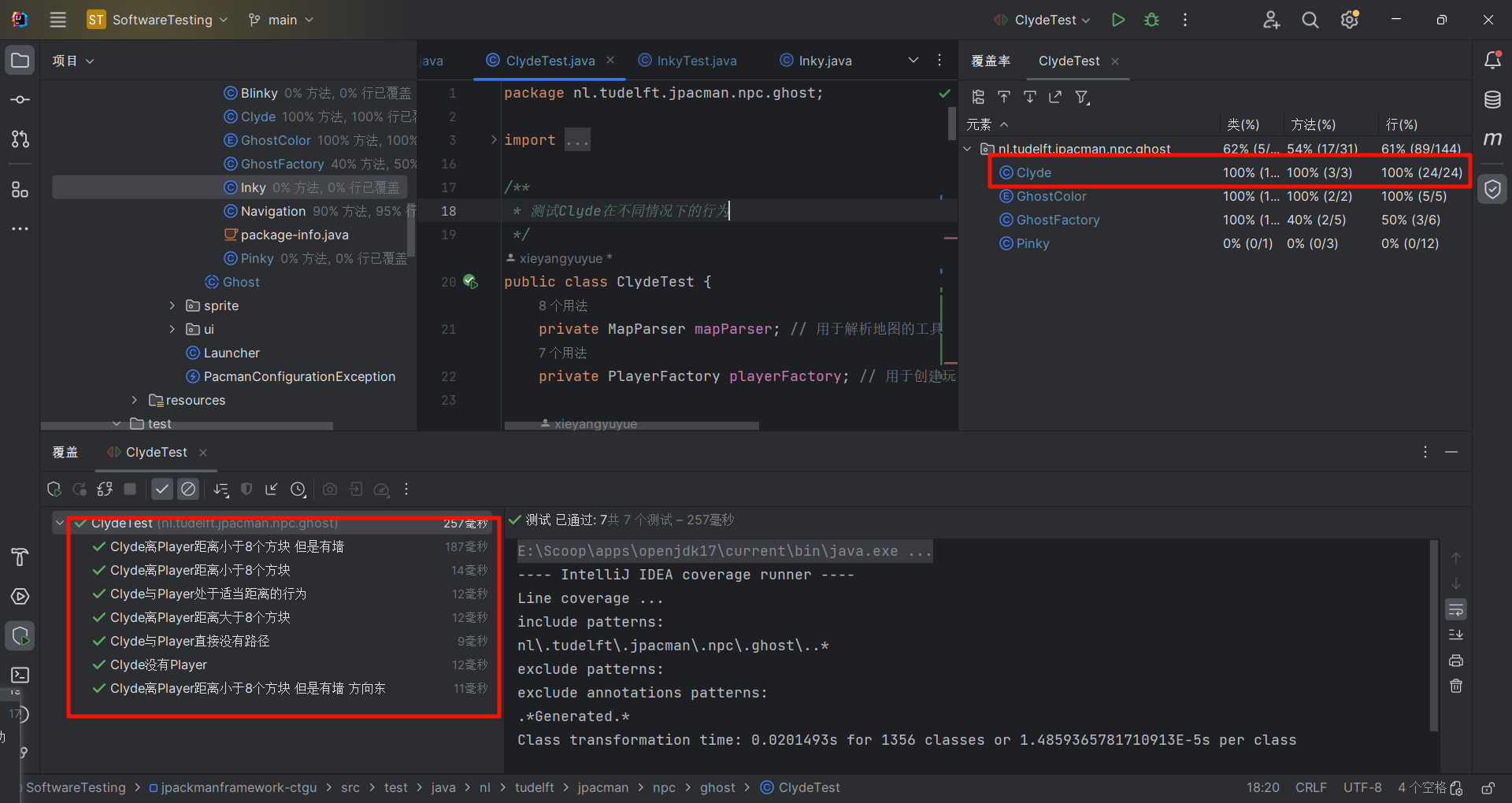
// 验证无移动决策（因玩家不存在，幽灵应停止移动）

assertThat(opt.isPresent()).isFalse();

}

}

#### clyde测试结果



#### InkyTest.java

package nl.tudelft.jpacman.npc.ghost;

import nl.tudelft.jpacman.board.BoardFactory;

import nl.tudelft.jpacman.board.Direction;

import nl.tudelft.jpacman.level.Level;

import nl.tudelft.jpacman.level.LevelFactory;

import nl.tudelft.jpacman.level.Player;

import nl.tudelft.jpacman.level.PlayerFactory;

import nl.tudelft.jpacman.sprite.PacManSprites;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import java.util.Arrays;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

public class InkyTest {

private static final PacManSprites SPRITES = new PacManSprites();

private PlayerFactory playerFactory;

private GhostMapParser ghostMapParser;

@BeforeEach

void setup() {

playerFactory = new PlayerFactory(SPRITES);

BoardFactory boardFactory = new BoardFactory(SPRITES);

GhostFactory ghostFactory = new GhostFactory(SPRITES);

LevelFactory levelFactory = new LevelFactory(SPRITES, ghostFactory);

ghostMapParser = new GhostMapParser(levelFactory, boardFactory, ghostFactory);

}

/\*\*

\* 注册Player到关卡并设置方向。

\*/

private void registerPlayer(Level level, Direction direction) {

Player player = playerFactory.createPacMan();

player.setDirection(direction);

level.registerPlayer(player);

}

/\*\*

\* 解析自定义地图布局。

\*/

private Level parseMap(List < String > grid) {

return ghostMapParser.parseMap(grid);

}

@Test

@DisplayName("当Inky、Blink和Player在同一直线时，Inky应朝Player方向移动")

void shouldMoveTowardsPlayerWhenAlignedWithBlinky() {

Level level = parseMap(Arrays.asList(

"#####################",

"# P B I #",

"#####################"

));

registerPlayer(level, Direction.EAST);

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

assertThat(inky.nextAiMove()).contains(Direction.WEST);

}

}

@Test

@DisplayName("当Player前方路径被阻挡时，Inky向Blinky移动")

void shouldMoveWhenPathIsBlocked() {

Level level = parseMap(Arrays.asList(

"#####################",

"#P# B I #",

"#####################"

));

registerPlayer(level, Direction.EAST);

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

assertThat(inky.nextAiMove()).contains(Direction.WEST);

}

}

@Test

@DisplayName("当Blinky不存在时，Inky无法计算路径")

void shouldNotMoveWithoutBlinky() {

Level level = parseMap(Arrays.asList(

"#############",

"# P I #",

"#############"

));

registerPlayer(level, Direction.WEST);

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

assertThat(inky.nextAiMove()).isEmpty();

}

}

@Test

@DisplayName("当Player不存在时，Inky无目标点")

void shouldNotMoveWithoutPlayer() {

Level level = parseMap(Arrays.asList(

"#############",

"# B I #",

"#############"

));

// 不注册Player

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

assertThat(inky.nextAiMove()).isEmpty();

}

}

@Test

@DisplayName("当Player面朝北方时，Inky的目标点计算包含经典偏移")

void shouldCalculateTargetWithOffsetWhenPlayerFacesNorth() {

Level level = parseMap(Arrays.asList(

"######################",

"# P #",

"# B #",

"# I #",

"######################"

));

registerPlayer(level, Direction.NORTH);

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

assertThat(inky.nextAiMove()).contains(Direction.WEST);

}

}

@Test

@DisplayName("当Blinky远离Player时，Inky朝延长路径方向移动")

void shouldMoveToExtendedPathWhenBlinkyIsDistant() {

Level level = parseMap(Arrays.asList(

"######################",

"# P I #",

"# #",

"# B #",

"######################"

));

registerPlayer(level, Direction.EAST);

Inky inky = Navigation.findUnitInBoard(Inky.class, level.getBoard());

if (inky != null) {

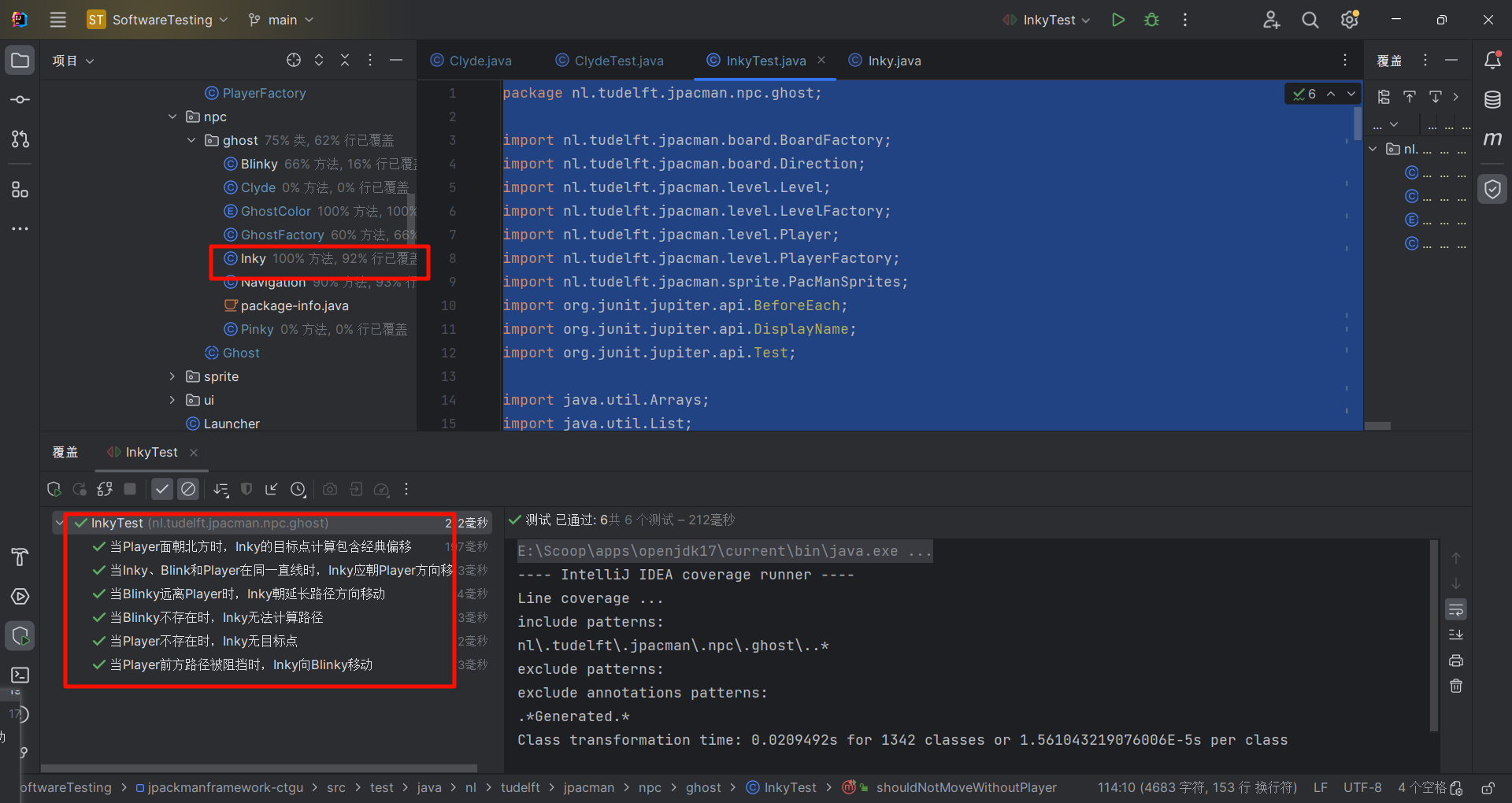
assertThat(inky.nextAiMove()).contains(Direction.SOUTH);

}

}

}

#### InkyTest测试结果



# 五、结论分析与体会

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)