Christian Johnson

Professor Businge

CS 472 Section 1001 Fall 2023

26 September 2023

Lab 2 Report

• Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?

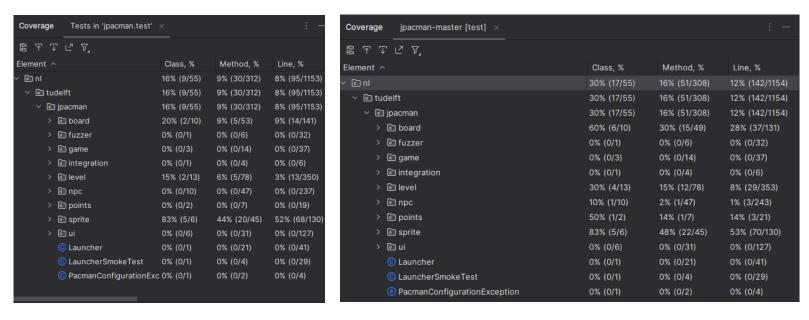
JaCoCo's coverage results are different than IntelliJ's. After brief research, IntelliJ's seems to use the codebase to determine class/method/line/etc totals and perecents, while JaCoCo analyzes bytecode.

- Did you find helpful the source code visualization from JaCoCo on uncovered branches?
 Yes, it specifically highlights missed lines which speeds up finding missed branches.
 - Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?

I like IntelliJ's coverage window because of how it's incorporated into the IDE, but I imagine JaCoCo's source code visualization features would work better for me since it provides more granularity than IntelliJ. If I could, I'd use IntelliJ to start testing then JaCoCo to refine my tests.

• Task 2.1 test code and testing coverage snippets:

My 3 implemented tests dealt with the BoardFactory, LevelFactory, and PointCalculator classes. Initial coverage with the "testAlive()" example from the class website is on the left, while the new coverage after my tests is on the right. There were slight increases in coverage in all categories, but my new tests reached 3 different classes that had 0% testing.



The 1st test I implemented was a BoardFactory test for the createGround() method. This checks a newly created ground square for the correct sprite from PacManSprites.

```
PlayerTest.java
                   © BoardFactoryTest.java ×
                                             © LevelFactoryTest.java
       package nl.tudelft.jpacman.board;
                                                                                                               import nl.tudelft.jpacman.sprite.PacManSprites;
                                                                                                               E P
       import org.junit.jupiter.api.Test;
       import static org.assertj.core.api.Assertions.assertThat;
                                                                                                               0
       public class BoardFactorvTest {
           private static final PacManSprites spritesForTest = new PacManSprites();
           private BoardFactory testBoardFac = new BoardFactory(spritesForTest);
            void testCreateGround(){
               Square testGround = testBoardFac.createGround();
               assertThat(testGround.getSprite()).isEqualTo(spritesForTest.getGroundSprite());
```

My next test was a LevelFactory test for the createPellet() method. In summary, it checks that the pellet and its fields were initialized properly.

```
    BoardFactoryTest.java

                                      © LevelFactoryTest.java ×
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.PointCalculator;
                                                                                                          \mathbb{Z}^2
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
                                                                                                          \odot
public class LevelFactoryTest {
    private static final PacManSprites spritesForTest = new PacManSprites();
    private GhostFactory testGhostFac = new GhostFactory(spritesForTest);
    private LevelFactory testLevelFac = new LevelFactory(spritesForTest,
    void testCreatePellet(){
       Pellet testPellet = testLevelFac.createPellet();
        assertThat( actual: testPellet.getSprite() != null).isEqualTo( expected: true);
        assertThat(Integer.class.isInstance(testPellet.getValue())).isEqualTo( expected: true);
```

My final test dealt with the PointCalculator and consumedAPellet() method. This records the player's initial score, simulates the player eating a pellet, and compares what score was stored against the initial score plus the pellet's value.

```
© PlayerTest,java © BoardFactoryTest,java © LevelFactoryTest,java © PointCalculatorTest,java × : import nl.tudelft.jpacman.level.Player;

import nl.tudelft.jpacman.level.PlayerFactory;

import nl.tudelft.jpacman.level.PlayerFactory;

import nl.tudelft.jpacman.sprite.PacManSprites;

import org.junit.jupiter.api.Test;

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

/**

* CS472 Testing Lab Task 2.1: added test case 3/3

* * @author Christian Johnson

*/

public class PointCalculatorTest {

2 usages

private static final PacManSprites spritesForTest = new PacManSprites();

1 usage

private PlayerFactory testPlayerFactory = new PlayerFactory(spritesForTest);

3 usages

private Player testPlayer = testPlayerFactory.createPacMan();

2 usages

private PointCalculator testPtCalc = new DefaultPointCalculator();

// Tests if consuming a pellet increases player's score correctly
@Test

void testConsumedAPellet() {

int initialScore = testPlayer.getScore();

testPtCalc.consumedAPellet() {

int initialScore = testPlayer.getScore();

testPtCalc.consumedAPellet((testPlayer, testPellet);

assertThat( actual testPlayer.getScore() == (initialScore + testPellet.getValue())).isEqualTo()

}

}
```

Fork repository link: https://github.com/chrisj117/CS-472-2023-GROUP-2