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.

Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	16% (9/55)	9% (30/312)	8% (95/1153)
tudelft	16% (9/55)	9% (30/312)	8% (95/1153)
jpacman	16% (9/55)	9% (30/312)	8% (95/1153)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	15% (2/13)	6% (5/78)	3% (13/350)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	83% (5/6)	44% (20/45)	52% (68/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationExc	0% (0/1)	0% (0/2)	0% (0/4)

Coverage | jpacman-master [test]

Element	Class, %	Method, %	Line, %
nl	30% (17/55)	16% (51/308)	12% (142/1154)
tudelft	30% (17/55)	16% (51/308)	12% (142/1154)
jpacman	30% (17/55)	16% (51/308)	12% (142/1154)
board	60% (6/10)	30% (15/49)	28% (37/131)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	30% (4/13)	15% (12/78)	8% (29/353)
npc	10% (1/10)	2% (1/47)	1% (3/243)
points	50% (1/2)	14% (1/7)	14% (3/21)
sprite	83% (5/6)	48% (22/45)	53% (70/130)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

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 | PointCalculatorTest.java

```java
package nl.tudelft.jpacman.board;

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 1/3
 *
 * @author Christian Johnson
 */
public class BoardFactoryTest {
    2 usages
    private static final PacManSprites spritesForTest = new PacManSprites();
    1 usage
    private BoardFactory testBoardFac = new BoardFactory(spritesForTest);

    // Tests if the ground sprite is set correctly
    @Test
    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.

PlayerTest.java | BoardFactoryTest.java | LevelFactoryTest.java | PointCalculatorTest.java

```java
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.PointCalculator;
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 2/3
 *
 * @author Christian Johnson
 */
public class LevelFactoryTest {
    2 usages
    private static final PacManSprites spritesForTest = new PacManSprites();
    1 usage
    private GhostFactory testGhostFac = new GhostFactory(spritesForTest);
    1 usage
    PointCalculator testPtCalc = null;
    1 usage
    private LevelFactory testLevelFac = new LevelFactory(spritesForTest,
        testGhostFac, testPtCalc);

    // Tests that a valid Pellet was created, it has a sprite, and the pellet value type is int
    @Test
    void testCreatePellet(){
        Pellet testPellet = testLevelFac.createPellet();
        assertThat( actual: testPellet != null).isEqualTo( expected: true);
        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

```java
import nl.tudelft.jpacman.level.Player;
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 Pellet testPellet = new Pellet( points: 10, spritesForTest.getPelletSprite());
    1 usage
    private PointCalculator testPtCalc = new DefaultPointCalculator();

    // Tests if consuming a pellet increases player's score correctly
    @Test
    void testConsumedAPellet() {
        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