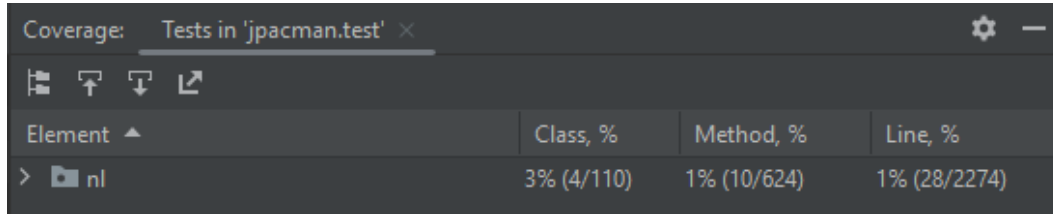


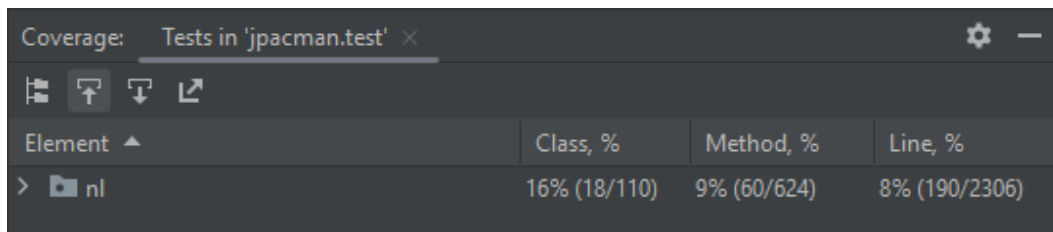
Testing Lab



Coverage: Tests in 'jpacman.test' ×

Element ▲	Class, %	Method, %	Line, %
> nl	3% (4/110)	1% (10/624)	1% (28/2274)

Initial Coverage Before adding any tests



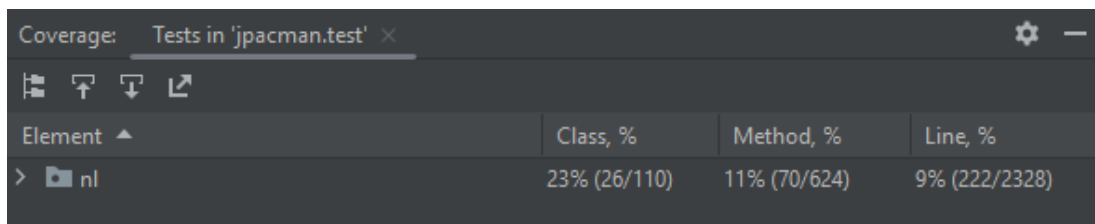
Coverage: Tests in 'jpacman.test' ×

Element ▲	Class, %	Method, %	Line, %
> nl	16% (18/110)	9% (60/624)	8% (190/2306)

After adding the isAlive() test for Task 2

The tests I added for Task 2.1

\\jpacman\\src\\main\\java\\nl\\tudelft\\jpacman\\level\\LevelFactory	createPellet()
\\jpacman\\src\\main\\java\\nl\\tudelft\\jpacman\\level\\PlayerCollisions	playerVersusPellet()
\\jpacman\\src\\main\\java\\nl\\tudelft\\jpacman\\board\\Board	withinBorders()



Coverage: Tests in 'jpacman.test' ×

Element ▲	Class, %	Method, %	Line, %
> nl	23% (26/110)	11% (70/624)	9% (222/2328)

After testing createPellet()

Coverage: Tests in 'jpacman.test' ×			
Element ▲	Class, %	Method, %	Line, %
> nl	25% (28/110)	13% (86/624)	10% (254/2342)

After testing playerVersusPellet()

Coverage: Tests in 'jpacman.test' ×			
Element ▲	Class, %	Method, %	Line, %
> nl	38% (42/110)	25% (156/624)	21% (514/2348)

After testing withinBorders()

Task 3

As we can see at the end of it, we had a 22% increase of coverage from adding `isAlive()` to the three tests I added in Task 2.1. Comparing the results of the JaCoCo report and the one in IntelliJ, the results are similar but due to the more in-depth factors of the JaCoCo report it did have a lesser percent. The visualization from the JaCoCo report on the uncovered branches does help a lot to see what you are missing from each package as well as each Java class. My preference visualization though is IntelliJ's coverage window because it is built in and much cleaner to read when going for a quick and less in depth look of how your tests are covering classes.

Local Fork Link: <https://github.com/Ray-Jarrold/jpacman>

Code Snippets

```
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.DefaultPointCalculator;
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;

no usages
public class PelletTest {
    /**
     * Creates all objects needed to test if createPellet() is functioning
     */
    no usages
    @Test
    void createPelletTest() {
        PacManSprites testPacman = new PacManSprites();
        GhostFactory testGhostFact = new GhostFactory(testPacman);
        PointCalculator testCalc = new DefaultPointCalculator();

        LevelFactory testLevel = new LevelFactory(testPacman, testGhostFact, testCalc);

        assertThat(testLevel.createPellet() != null);
    }
}
```

createPellet() test

```

package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.DefaultPointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

no usages
public class CollisionsTest {
    /**
     * Tests to make sure that the collision with Player and Pellet is functioning
     */
    no usages
    @Test
    void testPelletCollision() {
        PacManSprites testPacman = new PacManSprites();
        PlayerFactory testFactory = new PlayerFactory(testPacman);
        GhostFactory testGhostFact = new GhostFactory(testPacman);
        DefaultPointCalculator testCalc = new DefaultPointCalculator();
        PlayerCollisions testCollision = new PlayerCollisions(testCalc);
        Player pacTest = testFactory.createPacMan();

        LevelFactory testLevel = new LevelFactory(testPacman, testGhostFact, testCalc);

        Pellet testPellet = testLevel.createPellet();

        testCollision.playerVersusPellet(pacTest, testPellet);
        //Makes sure the players score increases to determine the collision worked
        assertThat(pacTest.getScore()).isEqualTo( expected: 10);
    }
}

```

playerVersusPellet() test

```

package nl.tudelft.jpacman.board;

import nl.tudelft.jpacman.level.Level;
import nl.tudelft.jpacman.level.LevelFactory;
import nl.tudelft.jpacman.level.MapParser;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.DefaultPointCalculator;
import nl.tudelft.jpacman.points.PointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.assertj.core.util.Lists;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

no usages
public class BordersTest {
    /**
     * A test that determines if withinBorders() is functioning properly
     */
    no usages
    @Test
    void inBorderTest() {
        PacManSprites pacSprites = new PacManSprites();
        GhostFactory testGhosts = new GhostFactory(pacSprites);
        BoardFactory testFactory = new BoardFactory(pacSprites);
        PointCalculator testCalc = new DefaultPointCalculator();
        LevelFactory testLevel = new LevelFactory(pacSprites, testGhosts, testCalc);
        MapParser testMapParser = new MapParser(testLevel, testFactory);

        /**
         * Creates test board that looks like
         * ###
         * #.#
         * ###
         */
        Level parsedLevel = testMapParser.parseMap(Lists.newArrayList("###", "#.#", "###"));
        Board testBoard = parsedLevel.getBoard();

        assertThat(testBoard.withinBorders(x: 1, y: 1));
    }
}

```

withinBorders() test