Nathaniel Almaguer

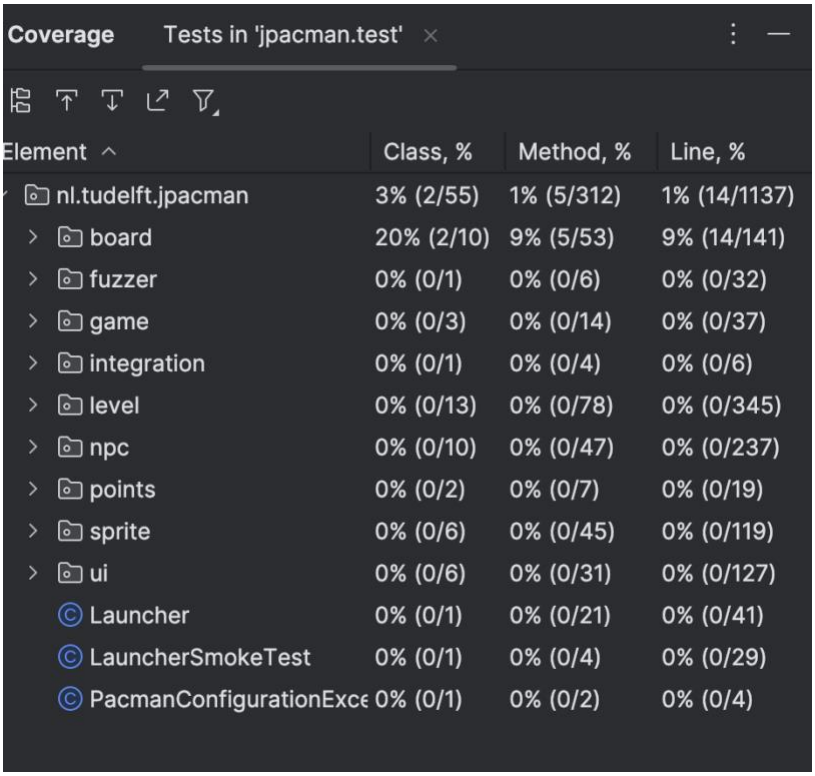
Dynamic Analysis Lab

Link to fork repository:
https://github.com/nathalmg/Group8

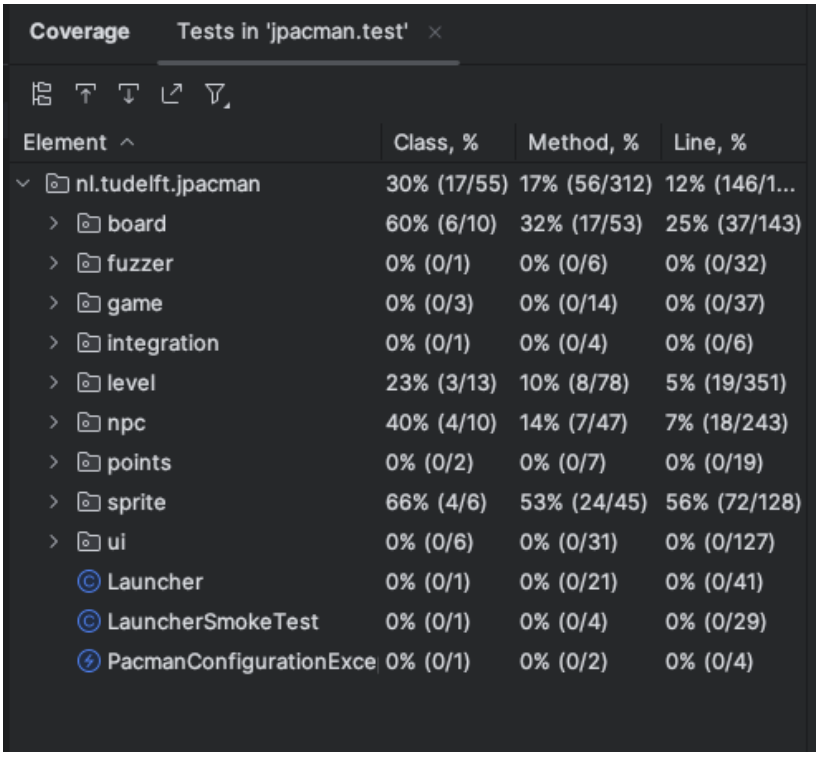
Task 1

- Is the coverage good enough?

Initially the coverage is not good enough since it is almost down to 0% in almost all the packages and very few lines are tested.

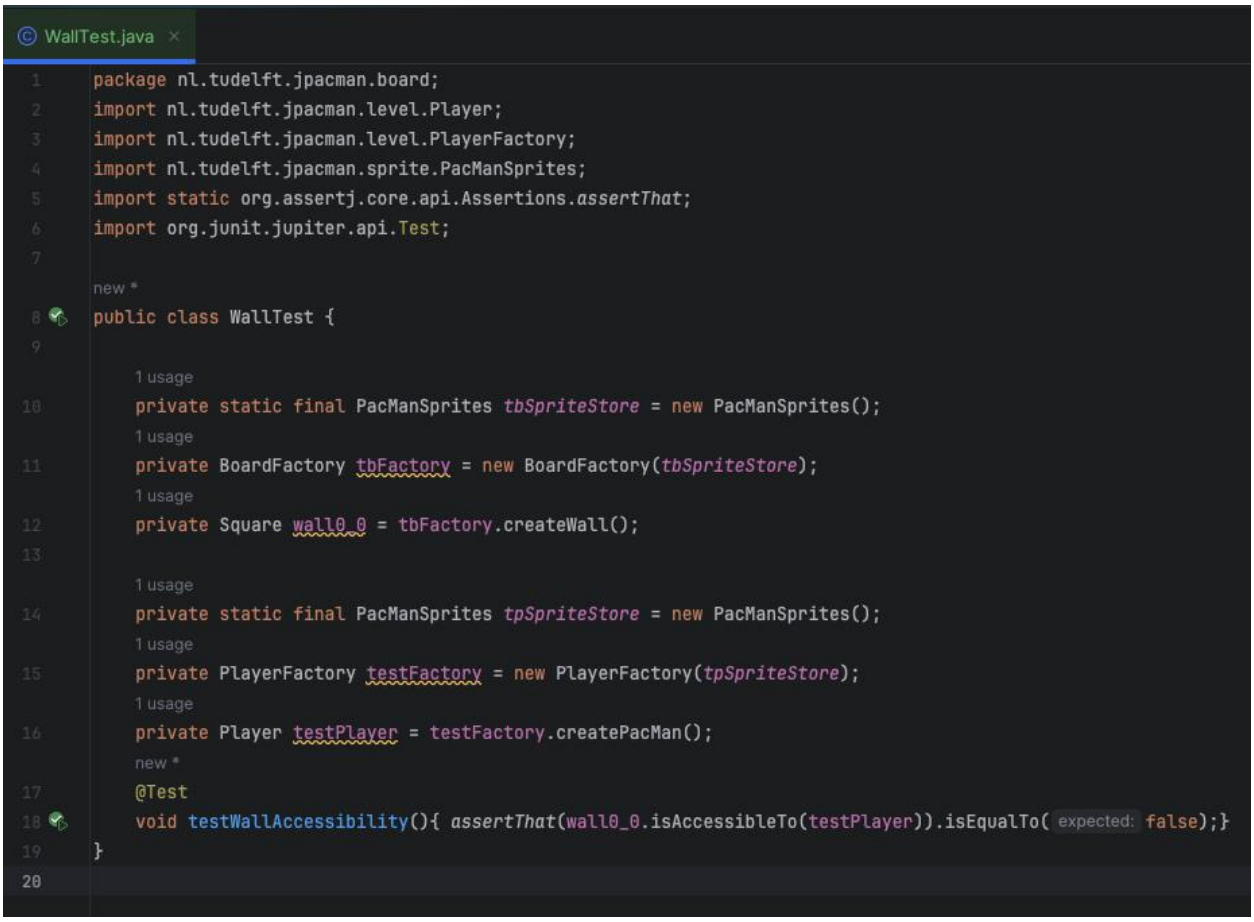


Task 2



The test coverage after implementing the unit tests.

Unit Test 1



```java
package nl.tudelft.jpacman.board;
import nl.tudelft.jpacman.level.Player;
import nl.tudelft.jpacman.level.PlayerFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;

new *
public class WallTest {

    1 usage
    private static final PacManSprites tbSpriteStore = new PacManSprites();
    1 usage
    private BoardFactory tbFactory = new BoardFactory(tbSpriteStore);
    1 usage
    private Square wall0_0 = tbFactory.createWall();

    1 usage
    private static final PacManSprites tpSpriteStore = new PacManSprites();
    1 usage
    private PlayerFactory testFactory = new PlayerFactory(tpSpriteStore);
    1 usage
    private Player testPlayer = testFactory.createPacMan();
    new *
    @Test
    void testWallAccessibility(){ assertThat(wall0_0.isAccessibleTo(testPlayer)).isEqualTo( expected: false);}
}
```

The first unit test I worked on was to ensure that the walls created are inaccessible to the pacman player. This was done by creating a sprite from the PacManSprites file, which turned into a wall through the BoardFactory function createWall(). Then a player was created, similarly to the PlayerTest example that was given, to use in the isAccessibleTo() function that determines whether a player or not is able to move into that space. The output was tested to assert that this was false since a player cannot move into a wall.

Unit Test 2

GroundTest.java

```java
package nl.tudelft.jpacman.board;
import nl.tudelft.jpacman.level.Player;
import nl.tudelft.jpacman.level.PlayerFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;
new *
public class GroundTest {
    1 usage
    private static final PacManSprites tbSpriteStore = new PacManSprites();
    1 usage
    private BoardFactory tbFactory = new BoardFactory(tbSpriteStore);
    1 usage
    private Square testGround = tbFactory.createGround();

    1 usage
    private static final PacManSprites tpSpriteStore = new PacManSprites();
    1 usage
    private PlayerFactory testFactory = new PlayerFactory(tpSpriteStore);
    1 usage
    private Player testPlayer = testFactory.createPacMan();
    new *
    @Test
    void testGroundAccessibility(){ assertThat(testGround.isAccessibleTo(testPlayer)).isEqualTo( expected: true);}
}
```

This test was to ensure that player can access ground squares. It is similar to the last test where I utilized BoardFactory and its function createGround() to turn a sprite into a ground square. Then a player was made to test that the ground is accessible to them, which of course would equal to true.

Unit Test 3

PelletTest.java

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

import nl.tudelft.jpacman.sprite.PacManSprites;
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;

new *
public class PelletTest {
    2 usages
    private final int tnum = 1;
    2 usages
    private static final PacManSprites SS = new PacManSprites();

    2 usages
    private Pellet tPellet = new Pellet(tnum, SS.getPelletSprite());

    new *
    @Test
    void testPelletNum(){ assertThat(tPellet.getValue()).isEqualTo(tnum); }
    new *
    @Test
    void testPelletSprite() { assertThat(tPellet.getSprite()).isEqualTo(SS.getPelletSprite()); }
```

The last test I covered was to ensure that the pellets created matched the values used to instantiate them. This was done by creating an int and a PacManSprite, which are both need to create a new Pellet. In the creation of the Pellet, getPelletSprite() was used to get the image of the needed for the Pellet. Afterwards, getValue() and getSprite() were used to determine if the Pellet values were equal to the initial values entered into the function.

Task 3

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

  The coverage results for the JaCoCo had more coverage than the one I got from IntelliJ. This is probably since JaCoCo can test between branches while IntelliJ only tested the ones I had on my local computer.

- Did you find helpful the source code visualization from JaCoCo on uncovered branches?

  The source code visualization from JaCoCo helps to see how up to date the branches are at a quick glance and if they require more testing, which is helpful to reduce potential errors when merging to the main branch in the future.

- Which visualization did you prefer and why? `IntelliJ`'s coverage window or `JaCoCo`'s report?

The JaCoCo report looks like the better visualization with the progress bars and has more information than IntelliJ. It is also helpful to see the coverage of the other branches.