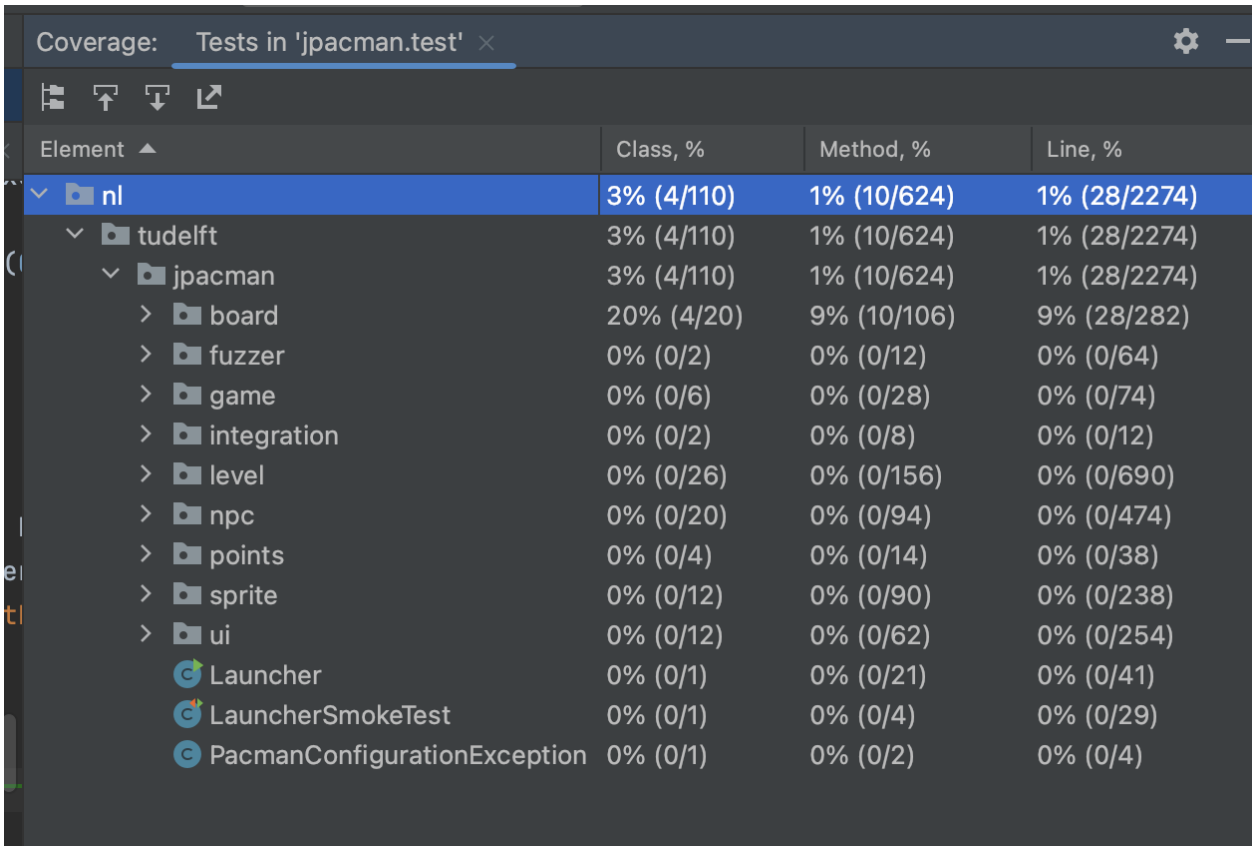
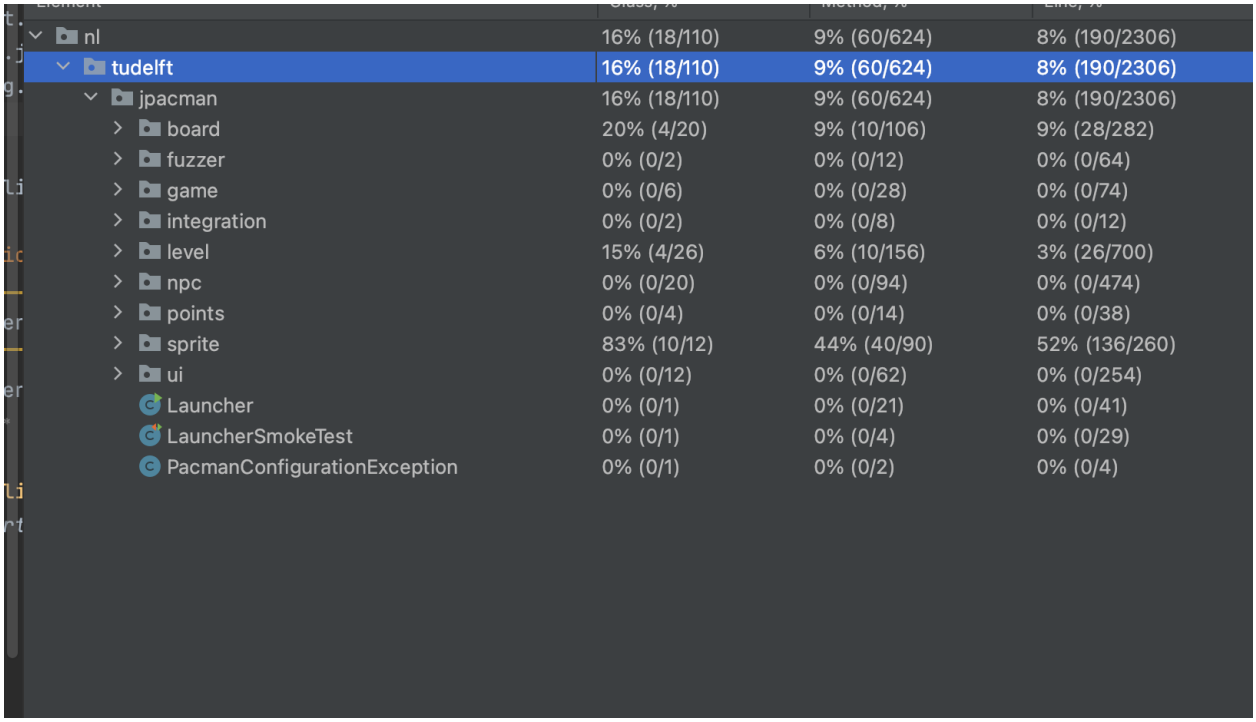
Link to fork repository:
https://github.com/SnellJ2/cs472project

Task-1



Task2



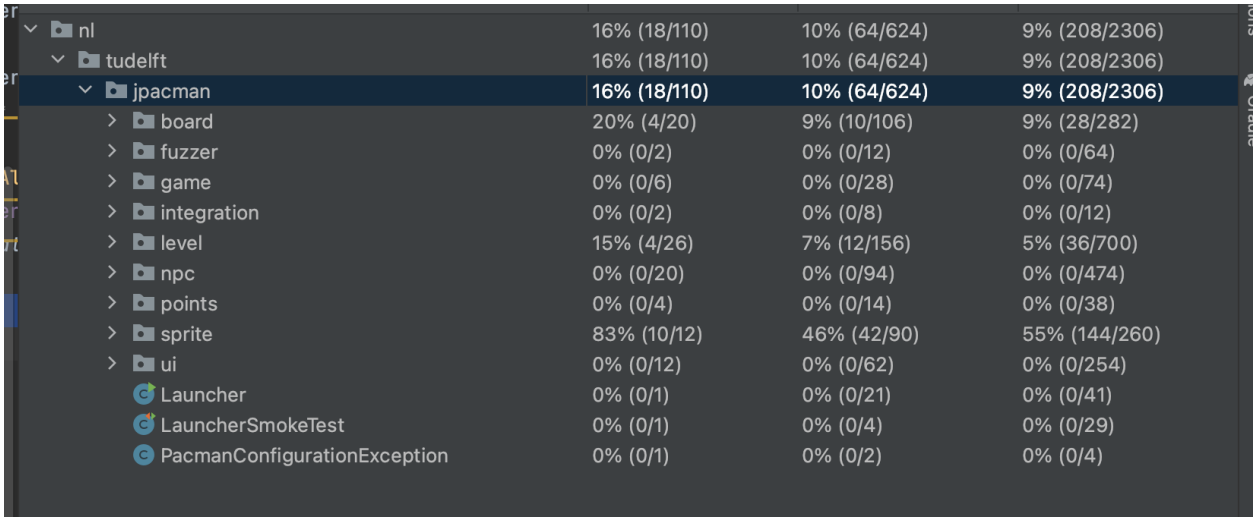
nl	16% (18/110)	9% (60/624)	8% (190/2306)
tudelft	16% (18/110)	9% (60/624)	8% (190/2306)
jpacman	16% (18/110)	9% (60/624)	8% (190/2306)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	6% (10/156)	3% (26/700)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	44% (40/90)	52% (136/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
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)

Task 2.1

The first method I tested was the setAlive() method in Player.java.
Here is the code.

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

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

no usages  new *
public class setAliveTest {
    1 usage
    private static final PacManSprites SPRITES = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITES);
    2 usages
    private Player the_player = Factory.createPacMan();
    no usages  new *
    @Test
    void testSetAlive(){
        the_player.setAlive(false);
        assertThat(the_player.isAlive()).isEqualTo( expected: false);
    }
}
```

These are the results after I ran it.



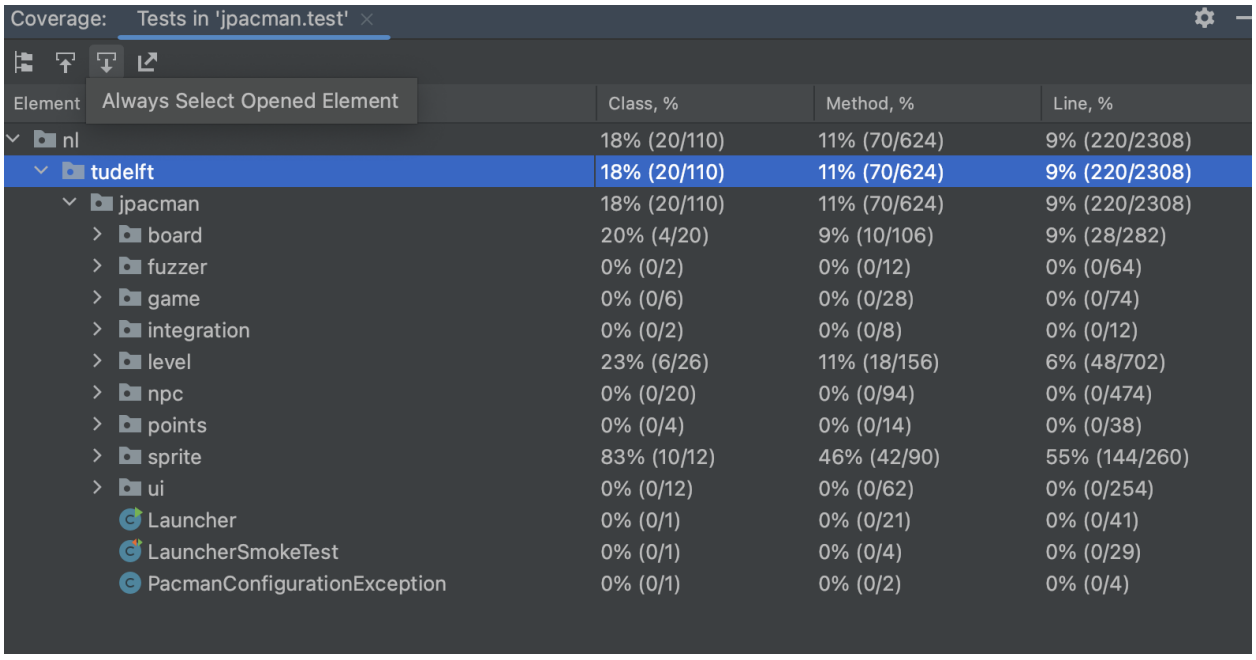
nl	16% (18/110)	10% (64/624)	9% (208/2306)
tudelft	16% (18/110)	10% (64/624)	9% (208/2306)
jpacman	16% (18/110)	10% (64/624)	9% (208/2306)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	7% (12/156)	5% (36/700)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/260)
ui	0% (0/12)	0% (0/62)	0% (0/254)
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 second method I tested was the constructor for the Pellet class. This essentially tests the whole class.

Here is the code.

```java
package nl.tudelft.jpacman.level;
import nl.tudelft.jpacman.sprite.EmptySprite;
import org.junit.jupiter.api.Test;
import nl.tudelft.jpacman.sprite.PacManSprites;

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

no usages  new *
public class pelletConstructTest {
    4 usages
    private static final EmptySprite sprite = new EmptySprite(); //the sprite to test
    2 usages
    private int value = 300; //and positive test
    2 usages
    private int valueNegative = -300; //a negative test

    no usages  new *
    @Test
    void pelletConstructTest(){
        Pellet test1 = new Pellet(value, sprite);
        assertThat(test1.getValue()).isEqualTo(value);
        assertThat(test1.getSprite()).isEqualTo(sprite);
        Pellet test2 = new Pellet(valueNegative, sprite);
        assertThat(test2.getValue()).isEqualTo(valueNegative);
        assertThat(test2.getSprite()).isEqualTo(sprite);

    }
}
```

Here is the coverage after testing.

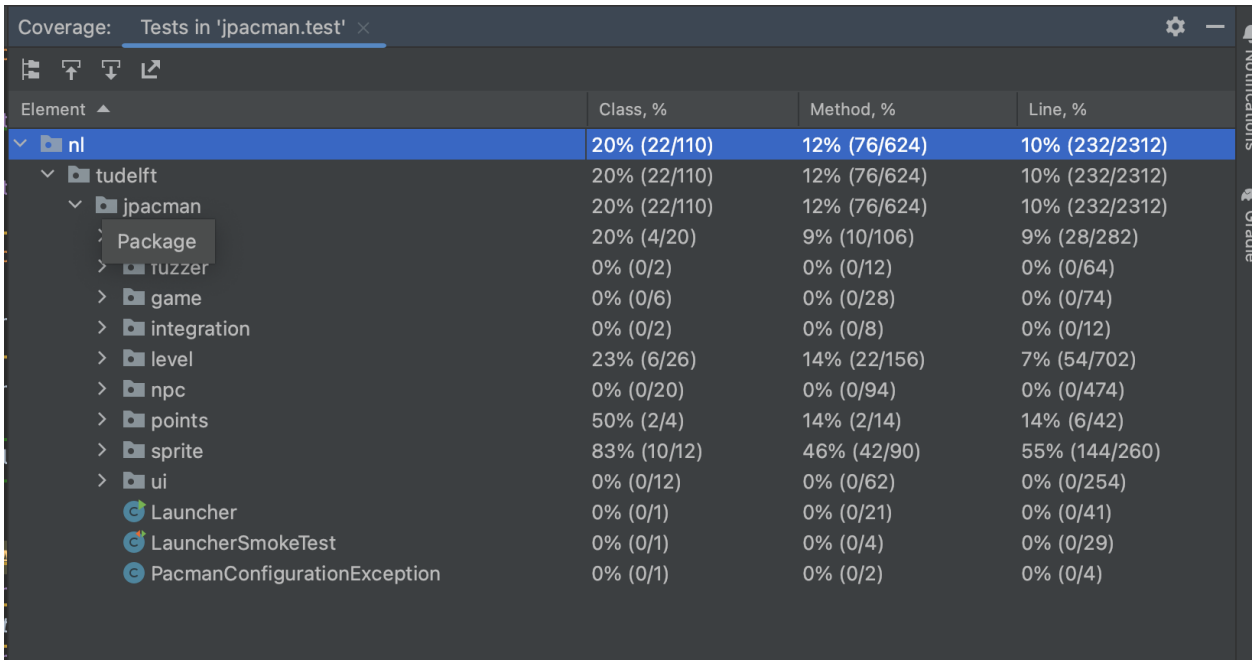


The final test I did was the consumedAPellet method in DefaultPointCalculator.

Here is the code for the test.

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

no usages  new *
public class consumedAPelletTest {
    2 usages
    private static final EmptySprite sprite = new EmptySprite(); //the sprite to test doesnt matter what sprite it real
    1 usage
    Pellet posTest = new Pellet( points: 30, sprite);
    1 usage
    Pellet negTest = new Pellet( points: -30, sprite); //we're going to check if both positive and negative tests work
    1 usage
    private static final PacManSprites SPRITES = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITES);
    4 usages
    private Player the_player = Factory.createPacMan();
    2 usages
    private DefaultPointCalculator calculator = new DefaultPointCalculator();
    no usages  new *
    @Test
    void consumedAPelletTest(){
        calculator.consumedAPellet(the_player, posTest); //consume positive
        assertThat(the_player.getScore()).isEqualTo( expected: 30);
        calculator.consumedAPellet(the_player, negTest);
        assertThat(the_player.getScore()).isEqualTo( expected: 0); //since it was 30 before it should be 0 now

    }
```

Here is the coverage results.



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 results are much different as my coverage was at most mine was 20% while there's was 54%. In addition the visualization of things like missed methods and classes are helpful in JaCoCo.

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

Yes I did find it helpful.

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

I found JaCoCo's report more helpful than IntelliJ's due to the visualization. However I am unsure if I would use it since I don't know how difficult it is to set up the tests. Even though it looks nice IntelliJ's coverage window seems to be detailed as well.