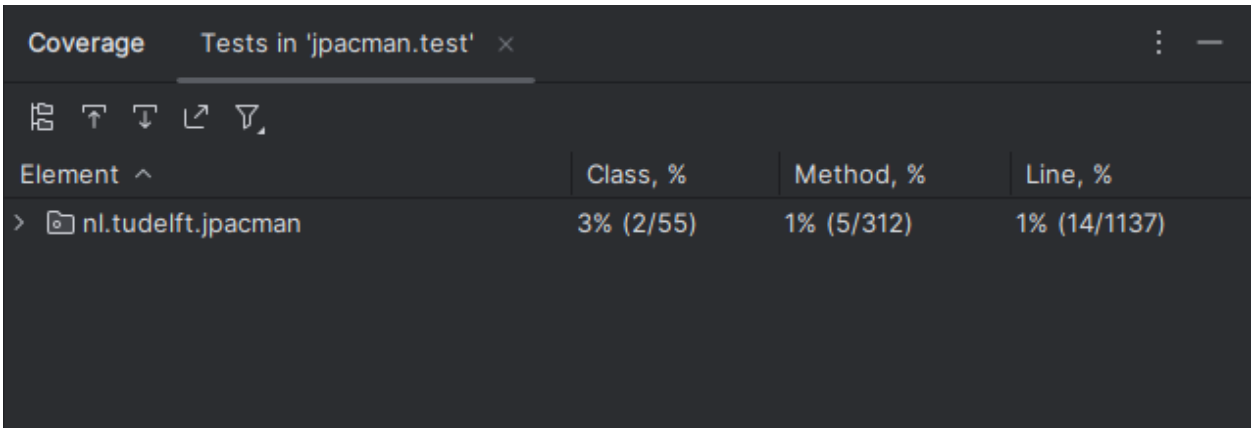
CS472 - Dynamic Analysis

Task 1 – JPacman Test Coverage

Initial Coverage



Is the coverage good enough?

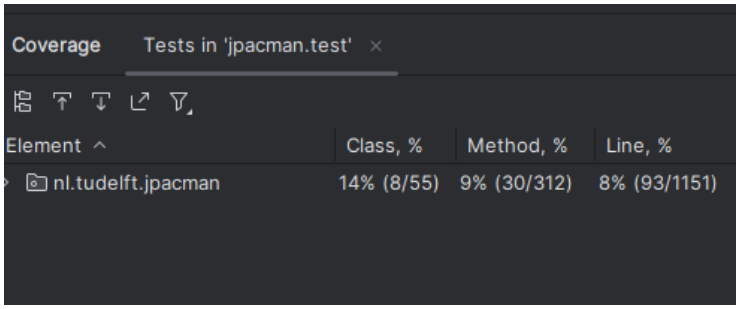
Simply put, no. The coverage is not nearly extensive to confirm a viable working product.

Task 2 – Increasing Coverage on JPacman

After PlayerTest:

```java
/**
 * New Test Case example
 * @author John Businge
 */
new *
public class PlayerTest {
    /**
     * I prefer to save the instances for this test in particular
     * because it is really a pain to instantiate Player, and I
     * will want to test other methods of Player in here.
     */
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    1 usage
    private Player ThePlayer = Factory.createPacMan();

    new *
    @Test
    void testAlive(){
        assertThat(ThePlayer.isAlive()).isEqualTo( expected: true);
    }
}
```



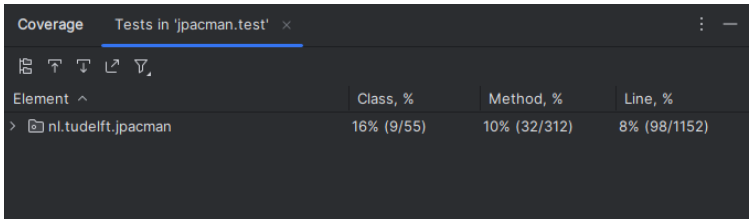
After PelletTest:

```java
/**
 * Tests for the {@link Pellet} class focusing on the getValue() method.
 */
public class PelletTest {

    @Test
    void testGetValue() {
        // Arrange
        int expectedValue = 10;
        Sprite dummySprite = mock(Sprite.class); // Mocking Sprite just to fulfill constructor requirement.
        Pellet pellet = new Pellet(expectedValue, dummySprite);

        // Act
        int actualValue = pellet.getValue();

        // Assert
        assertThat(actualValue).isEqualTo(expectedValue);
    }
}
```



After DefaultPointCalculatorTest:

```java
/**
 * Test class for DefaultPointCalculator focusing on consumedAPellet method.
 */
new *
public class DefaultPointCalculatorTest {

    3 usages
    private Player thePlayer;
    3 usages
    private Pellet pellet;
    2 usages
    private DefaultPointCalculator calculator;
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    new *
    @BeforeEach
    void setUp() {
        // Use PlayerFactory to create a Player, similar to PlayerTest
        PlayerFactory factory = new PlayerFactory(SPRITE_STORE);
        thePlayer = factory.createPacMan();


        // Mocking the Pellet to return a specific value
        pellet = mock(Pellet.class);
        when(pellet.getValue()).thenReturn( t: 10); // Assuming each pellet is worth 10 points

        // Initialize the calculator
        calculator = new DefaultPointCalculator();
    }
```

Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	18% (10/55)	11% (35/312)	8% (103/1153)

After EmptySpriteTest:

```java
/**
 * Test class for EmptySprite.
 */
public class EmptySpriteTest {

    5 usages
    private EmptySprite emptySprite;
    3 usages
    private Graphics graphics;

    @BeforeEach
    void setUp() {
        emptySprite = new EmptySprite();
        graphics = mock(Graphics.class); // Mocking Graphics to verify draw doesn't interact with it.
    }

    @Test
    void drawDoesNothing() {
        emptySprite.draw(graphics, x: 0, y: 0, width: 10, height: 10);
        verifyZeroInteractions(graphics); // Adjusted method call
    }

    @Test
    void splitReturnsNewEmptySprite() {
        Sprite result = emptySprite.split( x: 0, y: 0, width: 10, height: 10);
        assertThat(result).isInstanceOf(EmptySprite.class);
    }

    @Test
    void getWidthReturnsZero() { assertThat(emptySprite.getWidth()).isEqualTo( expected: 0); }

    @Test
    void getHeightReturnsZero() { assertThat(emptySprite.getHeight()).isEqualTo( expected: 0); }
}
```

Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	12% (39/312)	9% (107/1153)

Task 3 – JaCoCo Report on JPacman (10 points)

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

They are similar in the fact that they provide for the ability to drill down to see class method coverage to the line. However, They appear to calculate test coverage in different ways which can lead to slightly different results.

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

Absolutely, the visualization provided by JaCoCo was very helpful when drilling down to induvial branches. It allowed me to clearly see the parts of the code that are not yet covered by tests.

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

Personally, I prefer JaCoCo's report as it is more detailed, providing in-depth insights, which might be preferred for a thorough analysis of test coverage.

Juan Moreno Berber
https://github.com/morenj/CS472_Group1.git