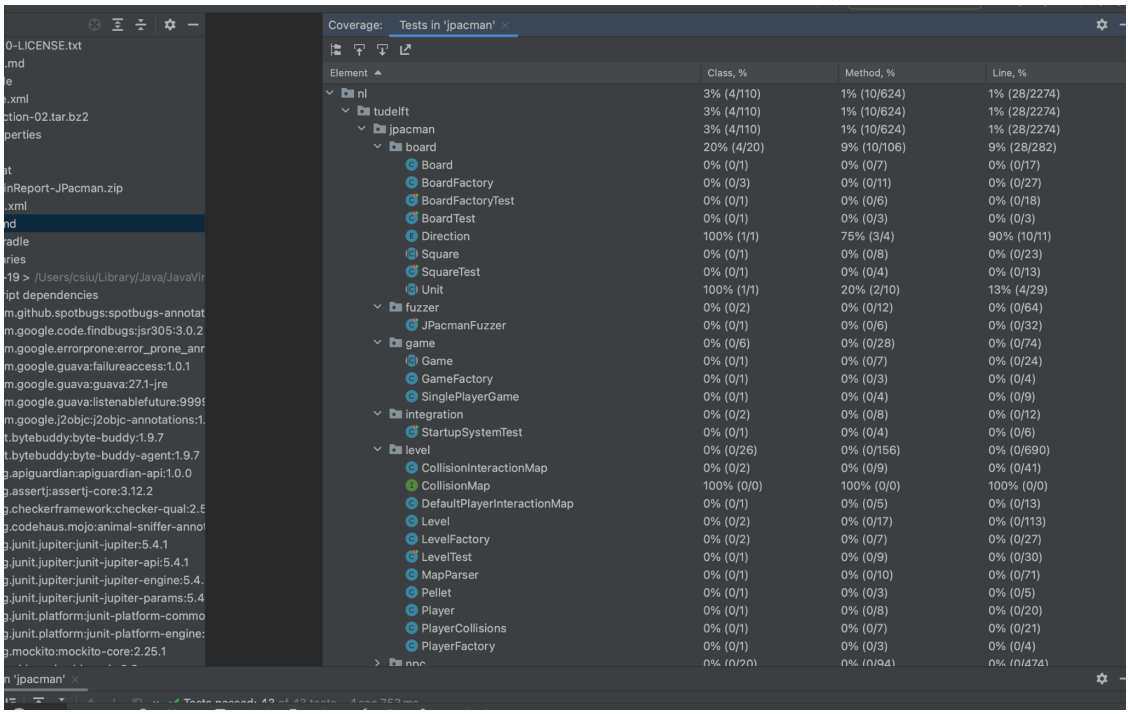
Repository:

https://github.com/yaaacii/jpacman

Task 1:

Before isAlive()

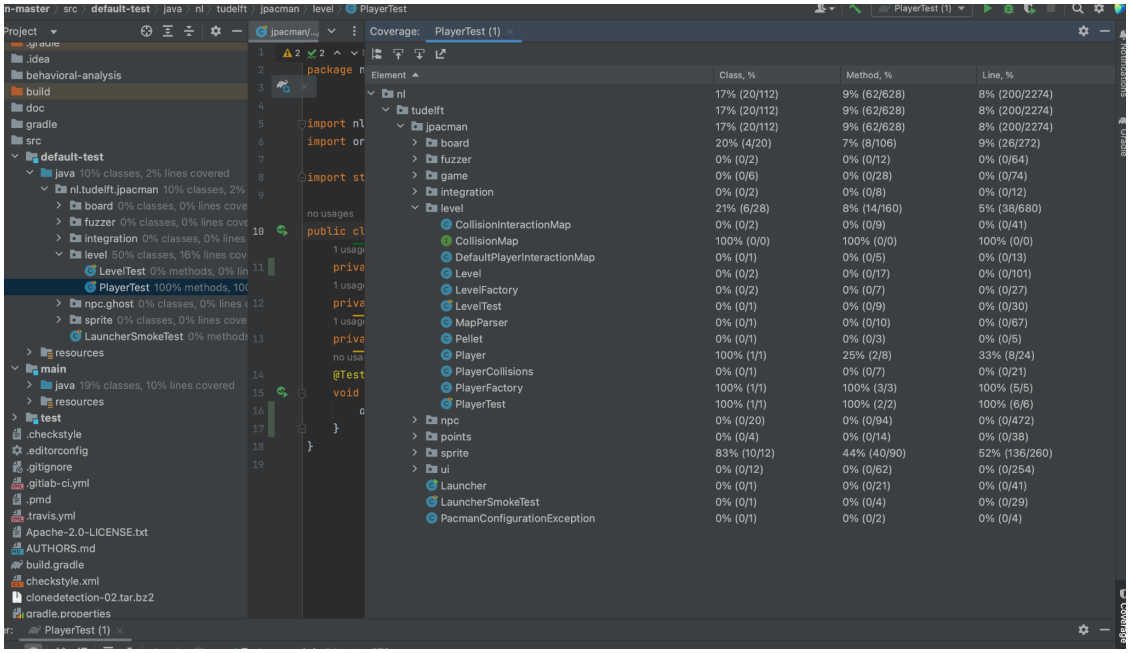


Question: Is this coverage good enough?

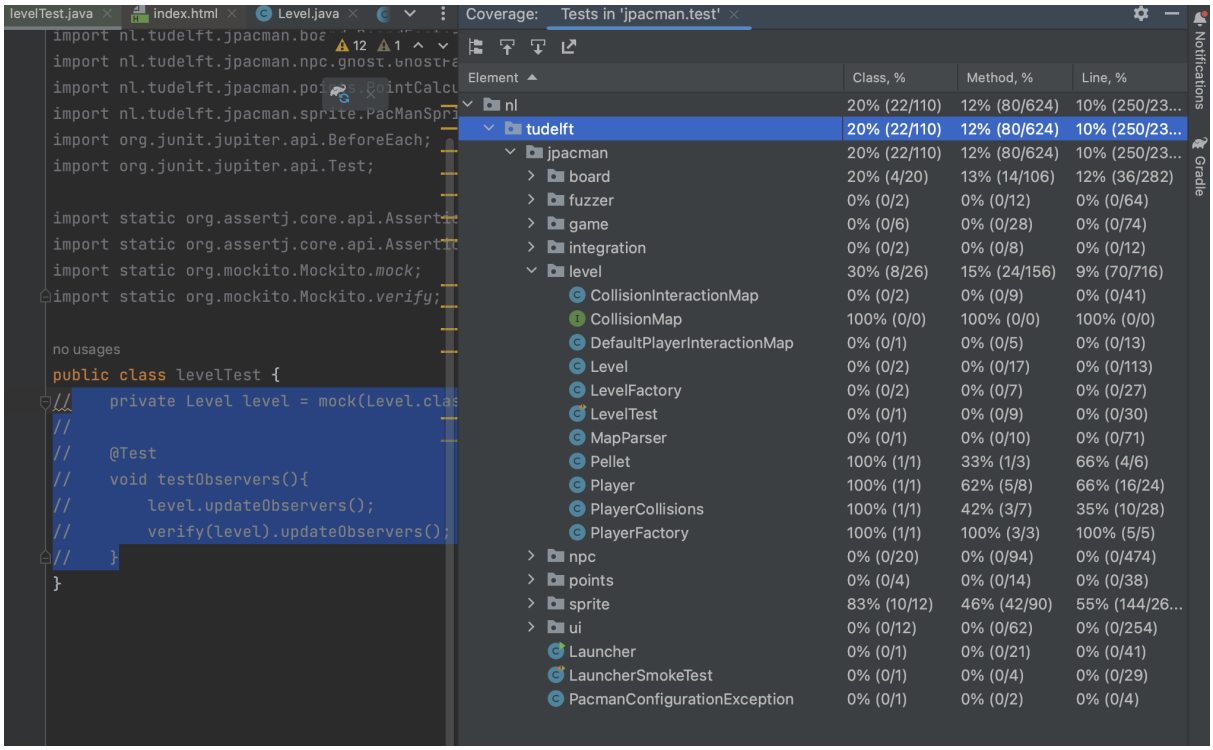
Answer: No. The higher the coverage the better. The best code coverage should be above 85%

Task 2:

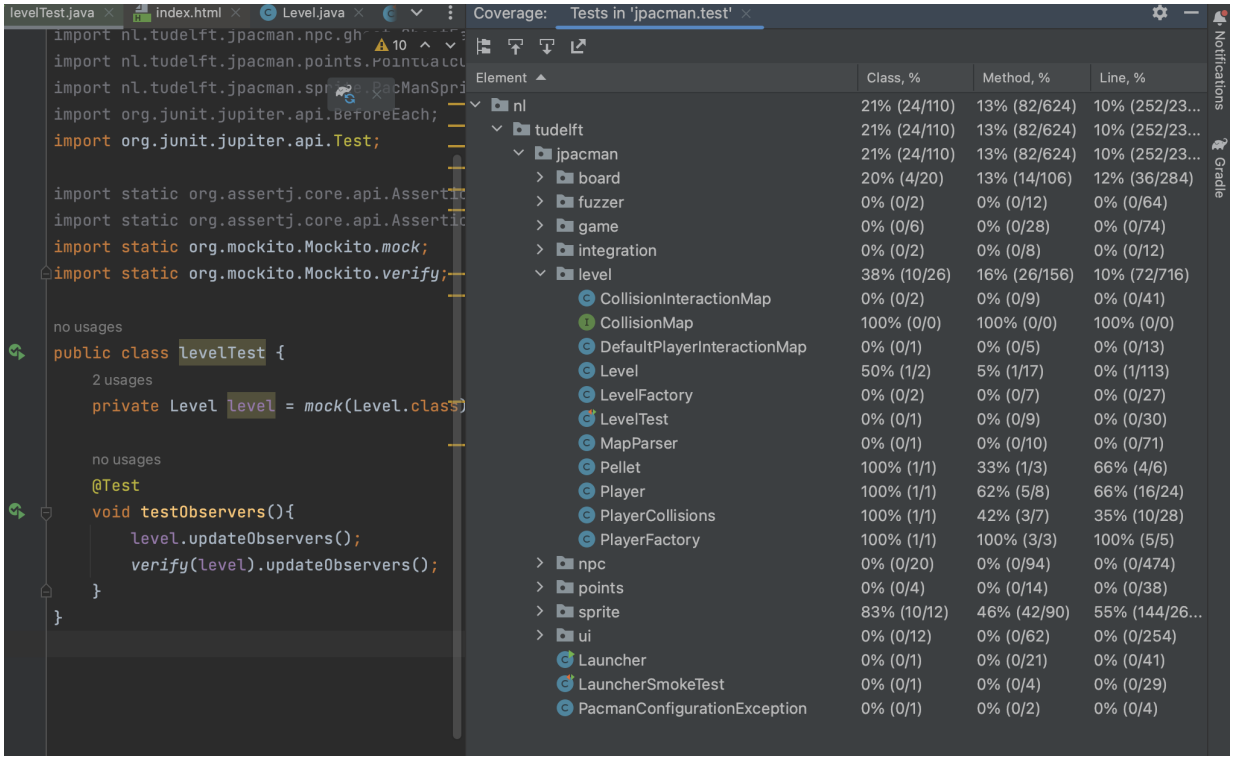
After isAlive()



Task 2.1:
Test1:
Before updateObservers():

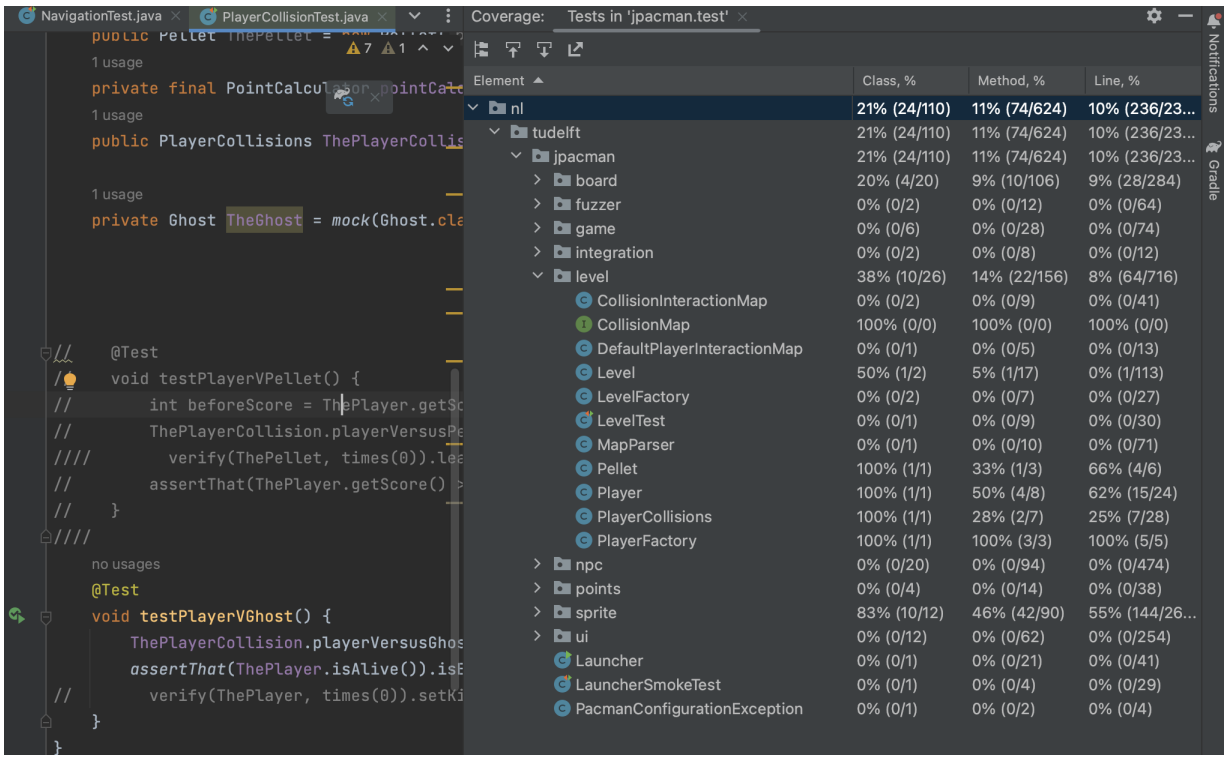


After updateObservers():

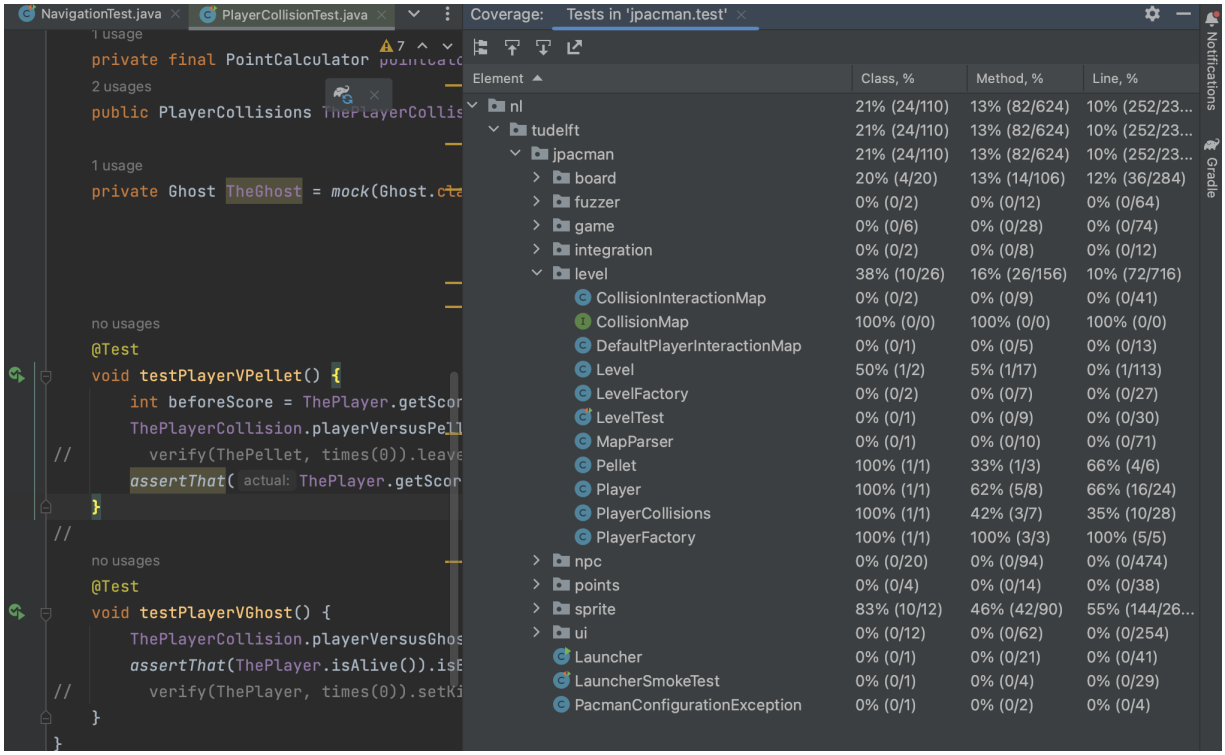


Test2:

Before PlayerVersusPellet():

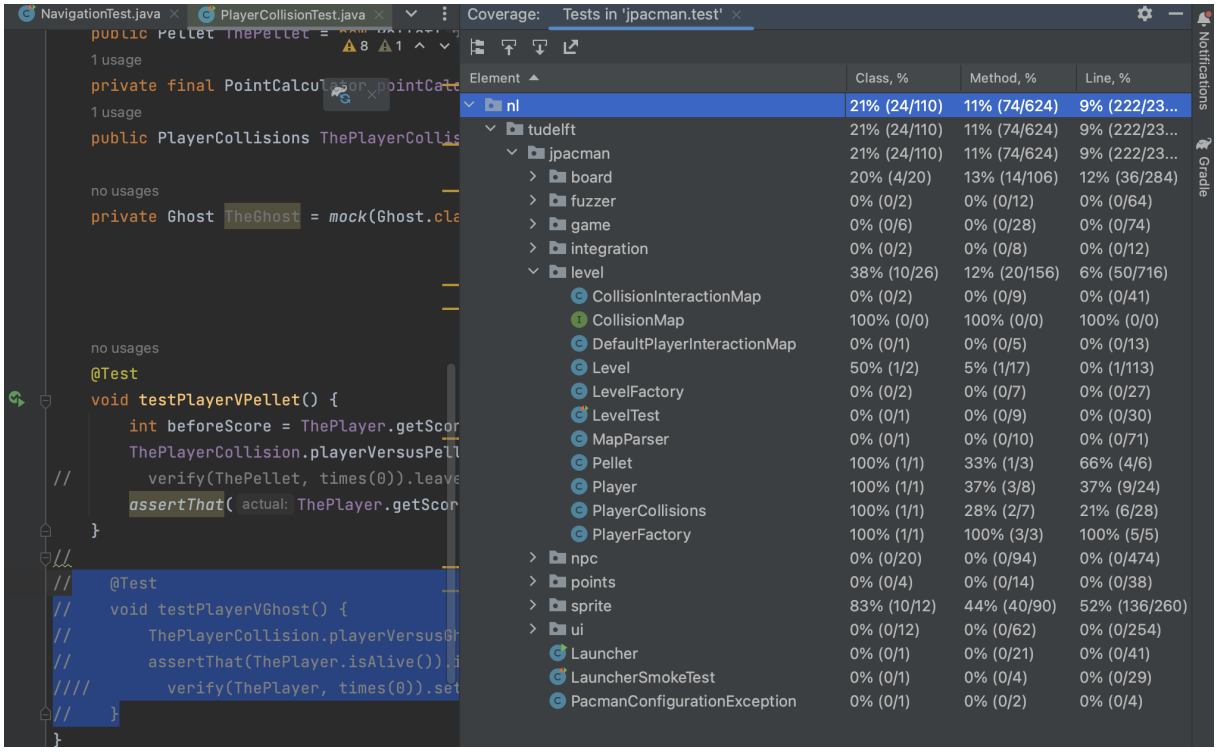


After PlayerVersusPellet():

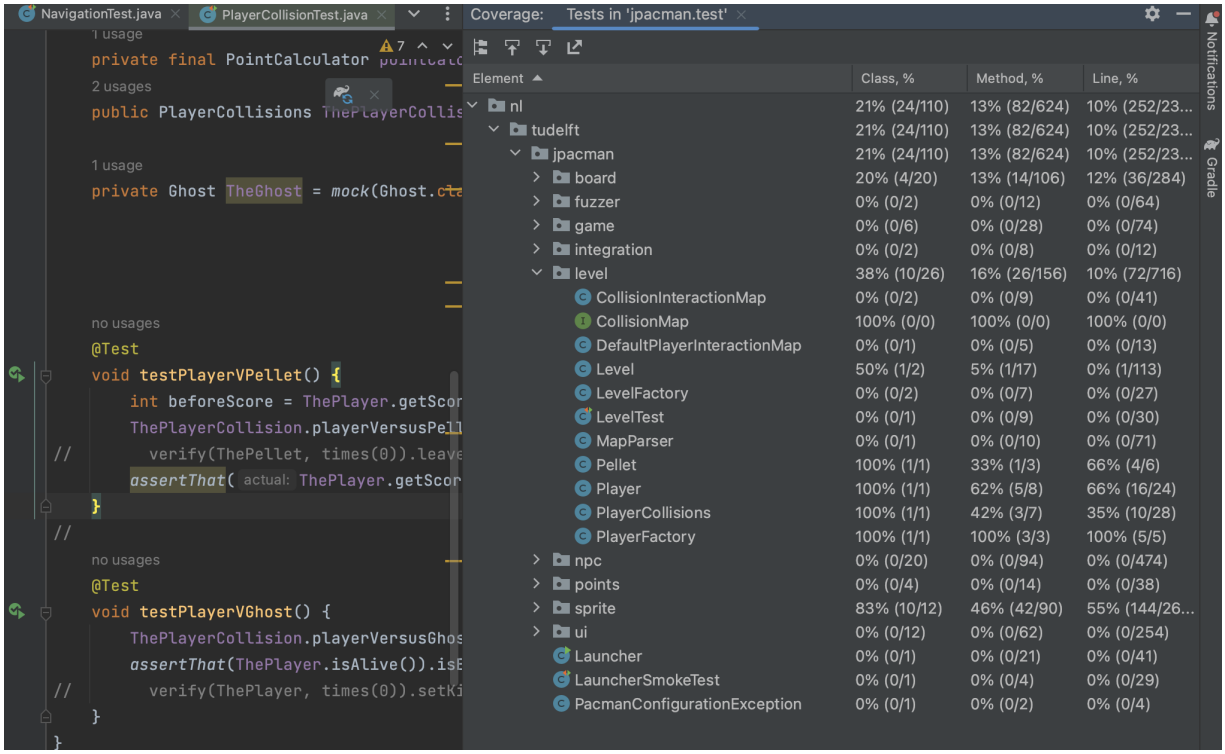


Test 3:

Before playerVersusGhost():



After playerVersusGhost():



Code Snippets:

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

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;

no usages
public class levelTest {
    2 usages
    private Level level = mock(Level.class);

    no usages
    @Test
    void testObservers(){
        level.updateObservers();
        verify(level).updateObservers();
    }
}
```

nl.tudelft.jpacman.level.Level

void updateObservers()

Updates the observers about the state of this level.

jpacman.main

```java
public class PlayerCollisionTest {
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    1 usage
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    5 usages
    private Player ThePlayer = Factory.createPacMan();
    1 usage
    private static final EmptySprite emptySprite = new EmptySprite();
    1 usage
    public Pellet ThePellet = new Pellet( points: 1, emptySprite);
    1 usage
    private final PointCalculator pointCalc = mock(PointCalculator.class);
    2 usages
    public PlayerCollisions ThePlayerCollision = new PlayerCollisions(pointCalc);
    1 usage
    private Ghost TheGhost = mock(Ghost.class);
    no usages
    @Test
    void testPlayerVPellet() {
        int beforeScore = ThePlayer.getScore();
        ThePlayerCollision.playerVersusPellet(ThePlayer, ThePellet);
//        verify(ThePellet, times(0)).leaveSquare();
        assertThat( actual: ThePlayer.getScore() > beforeScore);
    }
    no usages
    @Test
    void testPlayerVGhost() {
        ThePlayerCollision.playerVersusGhost(ThePlayer, TheGhost);
        assertThat(ThePlayer.isAlive()).isEqualTo( expected: false);
//        verify(ThePlayer, times(0)).setKiller(any());
    }
```

Task 3:

```
        return inProgress;
    }

    /**
     * Updates the observers about the state of this level.
     */
    void updateObservers() {
        if (!isAnyPlayerAlive()) {
            for (LevelObserver observer : observers) {
                observer.levelLost();
            }
        }
        if (remainingPellets() == 0) {
            for (LevelObserver observer : observers) {
                observer.levelWon();
            }
        }
    }
```

Questions:

- 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 from JaCoCo are similar enough, but there are some differences.
- Did you find helpful the source code visualization from JaCoCo on uncovered branches?
  - The source code visualization from JaCoCo helps to show which part of the code I need to add to my unit test.
- Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?
  - I liked JaCoCo's report because it has more information where I can see where their numbers are coming from and I like the source code visualization.

```
67.
68.     /**
69.      * Actual case of player bumping into ghost or vice versa.
70.      *
71.      * @param player
72.      *         The player involved in the collision.
73.      * @param ghost
74.      *         The ghost involved in the collision.
75.      */
76.     public void playerVersusGhost(Player player, Ghost ghost) {
77.         pointCalculator.collidedWithAGhost(player, ghost);
78.         player.setAlive(false);
79.         player.setKiller(ghost);
80.     }
81.
82.     /**
83.      * Actual case of player consuming a pellet.
84.      *
85.      * @param player
86.      *          The player involved in the collision.
87.      * @param pellet
88.      *          The pellet involved in the collision.
89.      */
90.     public void playerVersusPellet(Player player, Pellet pellet) {
91.         pointCalculator.consumedAPellet(player, pellet);
92.         pellet.leaveSquare();
93.     }
94.
95. }
```