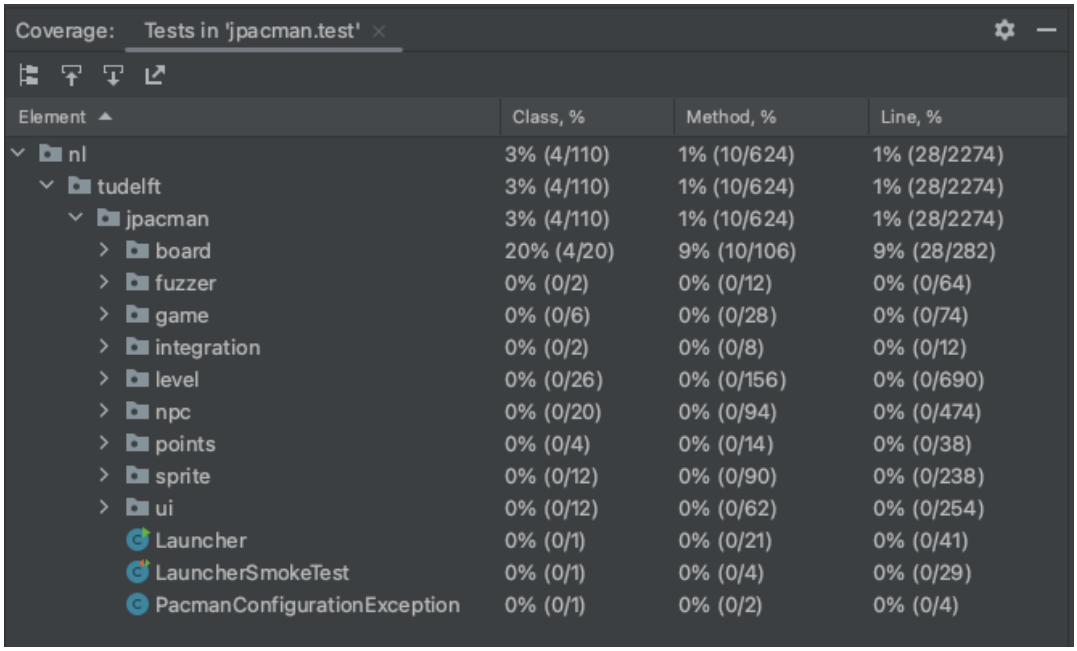
Link to repository: https://github.com/ks-moss/472-2023-G3.git

Task 1

Coverage before testing

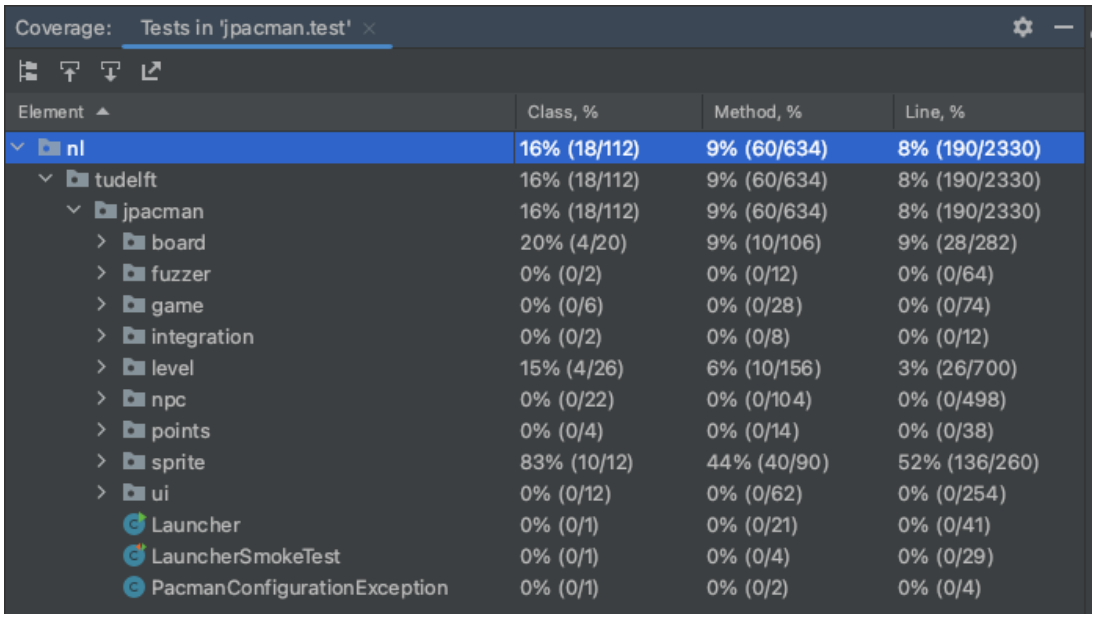


Element	Class, %	Method, %	Line, %
nl	3% (4/110)	1% (10/624)	1% (28/2274)
tudelft	3% (4/110)	1% (10/624)	1% (28/2274)
jpacman	3% (4/110)	1% (10/624)	1% (28/2274)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	0% (0/26)	0% (0/156)	0% (0/690)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	0% (0/12)	0% (0/90)	0% (0/238)
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)

Is the coverage good enough? No, it's not good enough. Let's add some more.

Task 2

```
1  package nl.tudelft.jpacman.level;
2
3  import nl.tudelft.jpacman.sprite.PacManSprites;
4  import org.junit.jupiter.api.Test;
5  import static org.assertj.core.api.Assertions.assertThat;
6
7
   no usages  new *
8  public class PlayerTest {
9
       1 usage
10     private static final PacManSprites SPRITE_STORE = new PacManSprites();
       1 usage
11     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
       2 usages
12     private Player ThePlayer = Factory.createPacMan();
13
14
       no usages  new *
15     @Test
16     void testAlive() {
17         assertThat(ThePlayer.isAlive()).isEqualTo( expected: true);
18         System.out.println("Value of isAlive() : " + ThePlayer.isAlive());
19     }
20
21 }
```



Task 2.1

src/main/java/nl/tudelft/jpacman/level/pellet.java

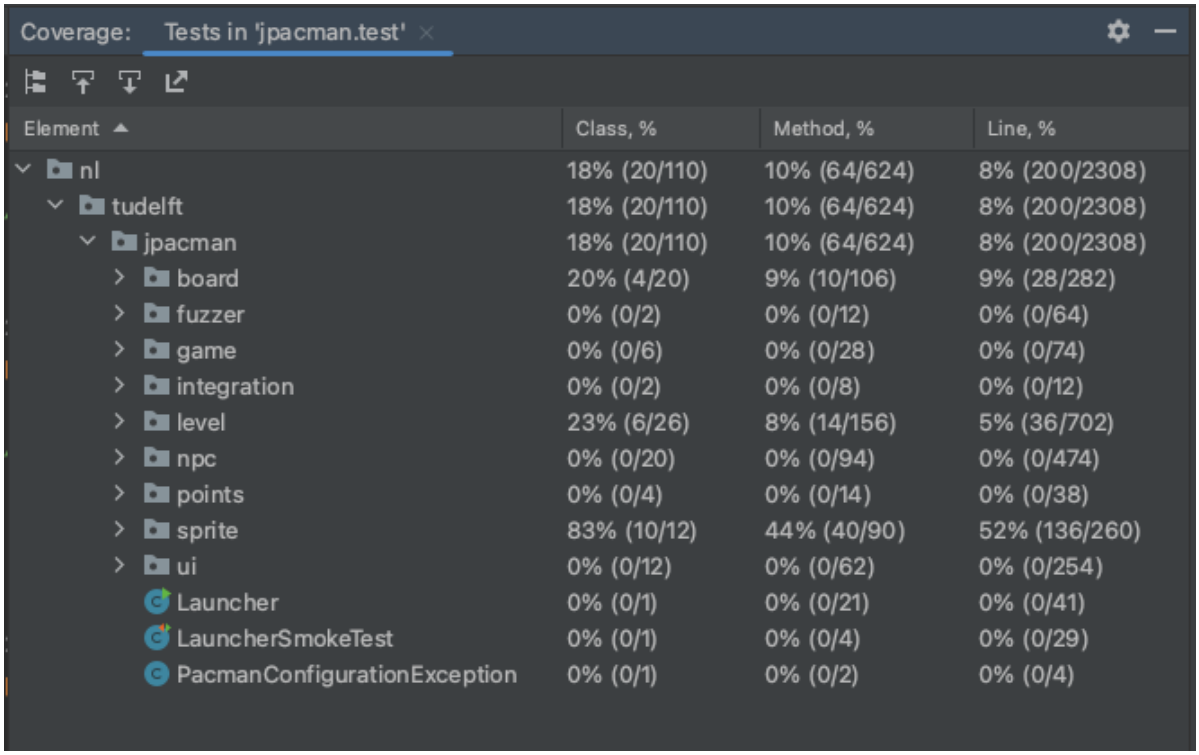
```java
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.sprite.PacManSprites;
import nl.tudelft.jpacman.sprite.Sprite;
import org.junit.jupiter.api.Test;

public class PelletTest {

    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    private Player ThePlayer = Factory.createPacMan();
    int points;
    Sprite SPRITE;
    public Pellet PELLET = new Pellet(points, SPRITE);

    @Test
    void testGetValue(){
        System.out.println("Value of PELLET.getValue() : " + PELLET.getValue());
    }
}
```



Element	Class, %	Method, %	Line, %
nl	18% (20/110)	10% (64/624)	8% (200/2308)
tudelft	18% (20/110)	10% (64/624)	8% (200/2308)
jpacman	18% (20/110)	10% (64/624)	8% (200/2308)
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	23% (6/26)	8% (14/156)	5% (36/702)
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)

src/main/java/nl/tudelft/jpacman/npc/ghost/GhostFactory.java

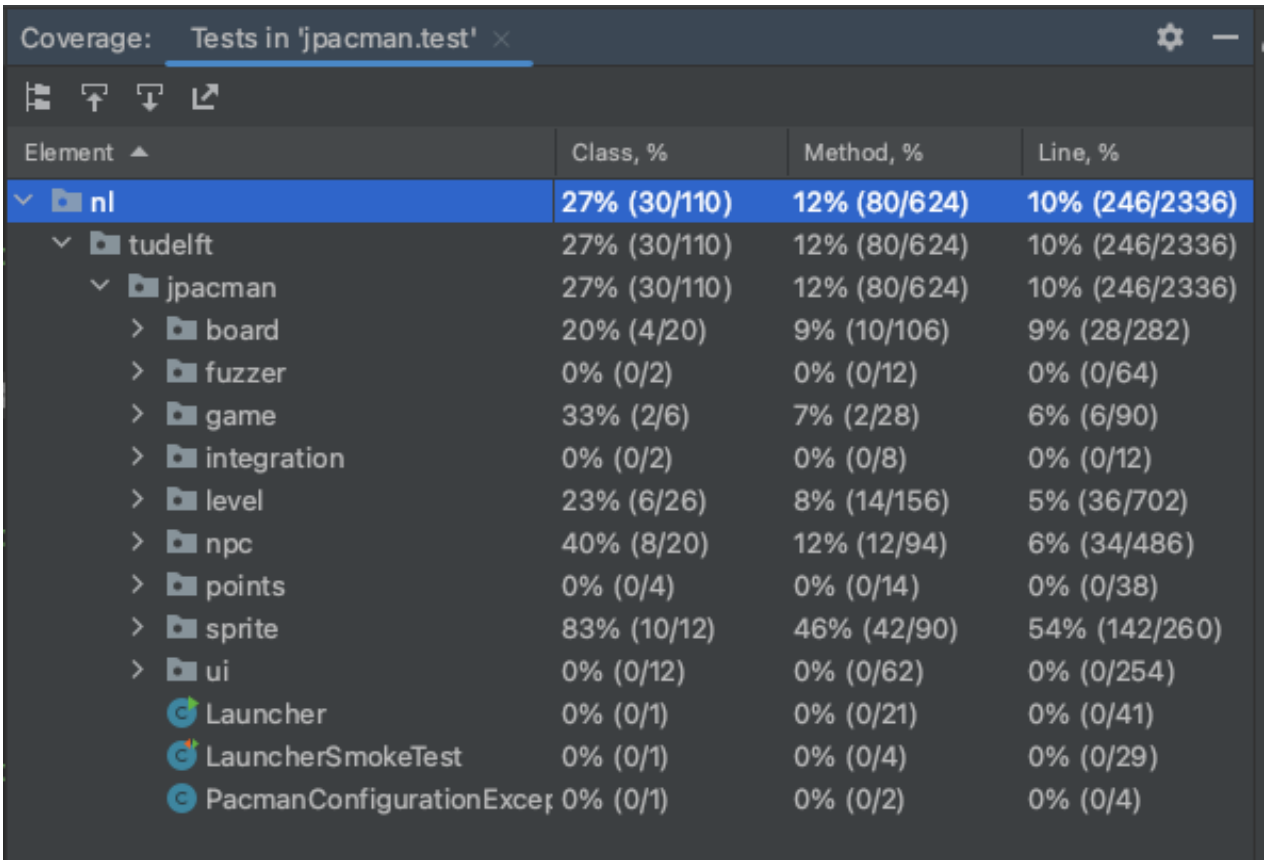
```
1   package nl.tudelft.jpacman.npc.ghost;                                  ✓2 ^ v
2
3   import nl.tudelft.jpacman.sprite.PacManSprites;
4   import org.junit.jupiter.api.Test;
5
    no usages   new *
6   public class GhostFactotryTest {
7
        1 usage
8       private static final PacManSprites SPRITE_STORE = new PacManSprites();
        1 usage
9       GhostFactory tempGhost = new GhostFactory(SPRITE_STORE);
10
        no usages   new *
11      @Test
12      void testGhostFactory(){
13
14          System.out.println("Value of createBlinky() : " + tempGhost.createBlinky());
15      }
16
17  }
18
```

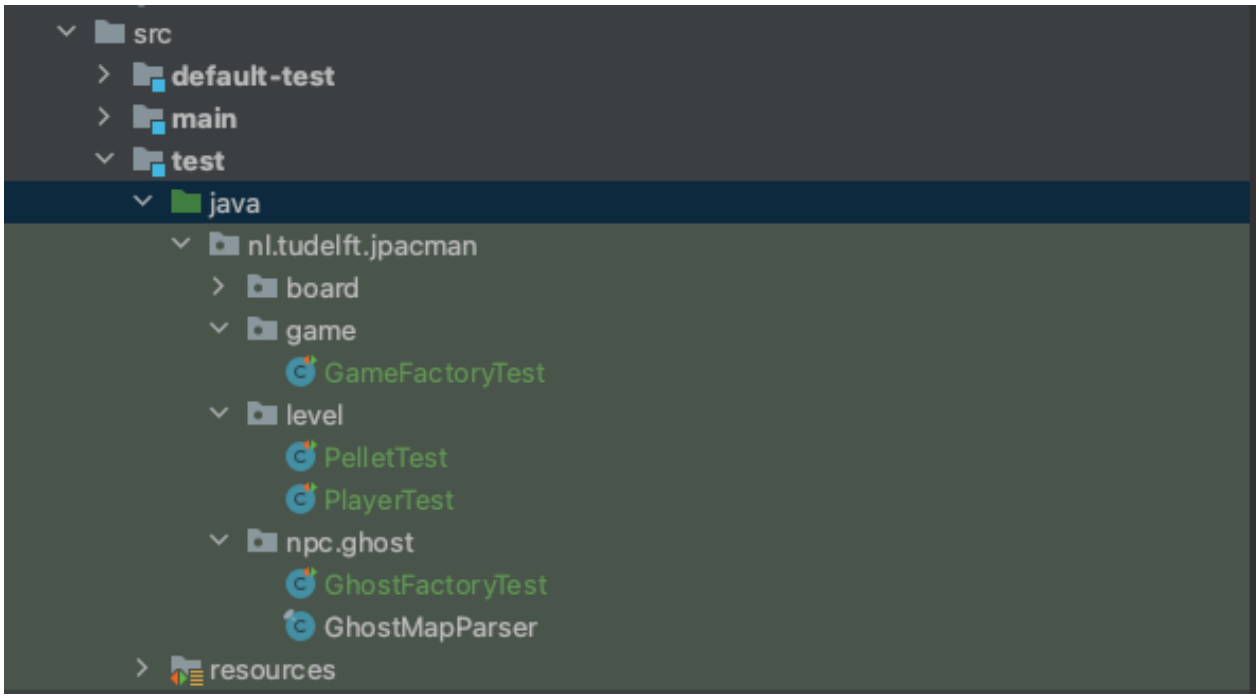
Coverage: Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	25% (28/110)	12% (78/624)	10% (240/2320)
tudelft	25% (28/110)	12% (78/624)	10% (240/2320)
jpacman	25% (28/110)	12% (78/624)	10% (240/2320)
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	23% (6/26)	8% (14/156)	5% (36/702)
npc	40% (8/20)	12% (12/94)	6% (34/486)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	54% (142/260)
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)

src/main/java/nl/tudelft/jpacman/game/GameFactory.java

```
1  package nl.tudelft.jpacman.game;                                  ⚠ 2
2
3
4  import nl.tudelft.jpacman.level.PlayerFactory;
5  import nl.tudelft.jpacman.sprite.PacManSprites;
6  import org.junit.jupiter.api.Test;
7
   no usages  new *
8  public class GameFactoryTest {
       1 usage
9      private static final PacManSprites SPRITE_STORE = new PacManSprites();
       1 usage
10     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
       no usages  new *
11     @Test
12     void testGameFactory(){
13         GameFactory GAME_FACTORY = new GameFactory(Factory);
14     }
15
16 }
17
```


Coverage:
Tests in 'jpacman.test'
Element
Class, %
Method, %
Line, %
nl 27% (30/110) 12% (80/624) 10% (246/2336)
tudelft 27% (30/110) 12% (80/624) 10% (246/2336)
jpacman 27% (30/110) 12% (80/624) 10% (246/2336)
board 20% (4/20) 9% (10/106) 9% (28/282)
fuzzer 0% (0/2) 0% (0/12) 0% (0/64)
game 33% (2/6) 7% (2/28) 6% (6/90)
integration 0% (0/2) 0% (0/8) 0% (0/12)
level 23% (6/26) 8% (14/156) 5% (36/702)
npc 40% (8/20) 12% (12/94) 6% (34/486)
points 0% (0/4) 0% (0/14) 0% (0/38)
sprite 83% (10/12) 46% (42/90) 54% (142/260)
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)
PacmanConfigurationExcep 0% (0/1) 0% (0/2) 0% (0/4)


src
default-test
main
test
java
nl.tudelft.jpacman
board
game
GameFactoryTest
level
PelletTest
PlayerTest
npc.ghost
GhostFactoryTest
GhostMapParser
resources

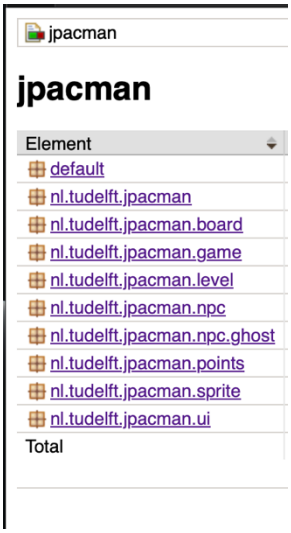
Task 3

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 obtained from the JaCoCo coverage tool differ from those previously obtained through IntelliJ in a prior task. The coverage results within IntelliJ are contingent upon various factors such as the type of coverage tool used, its configuration, the build process utilized, and the extent of test coverage within the codebase.

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

Affirmative, I discovered that the JaCoCo coverage tool provides a visualization of the source code, specifically highlighting any uncovered branches. This information can be readily accessed as demonstrated in the following illustration:



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

I found the JaCoCo coverage report to be preferable due to its convenient accessibility of information regarding uncovered branches, as well as its well-structured format. The report also employs a clear and effective color-coding scheme, utilizing red and white to display the coverage percentage as demonstrated below: