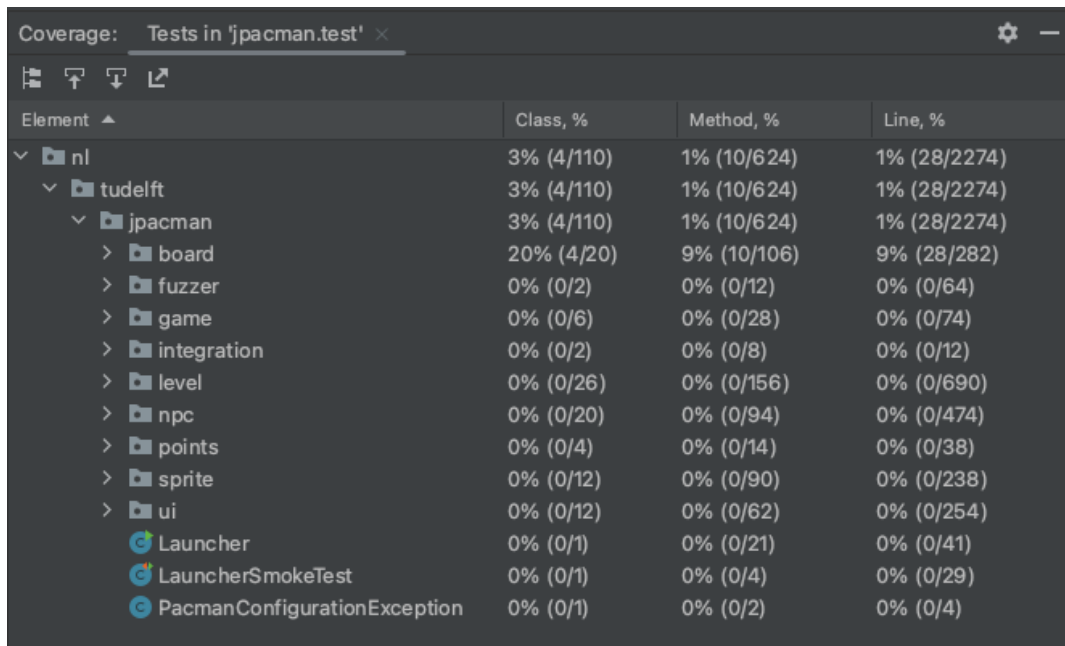


Link to repository: <https://github.com/ks-moss/472-2023-G3.git>

Task 1

Coverage before testing

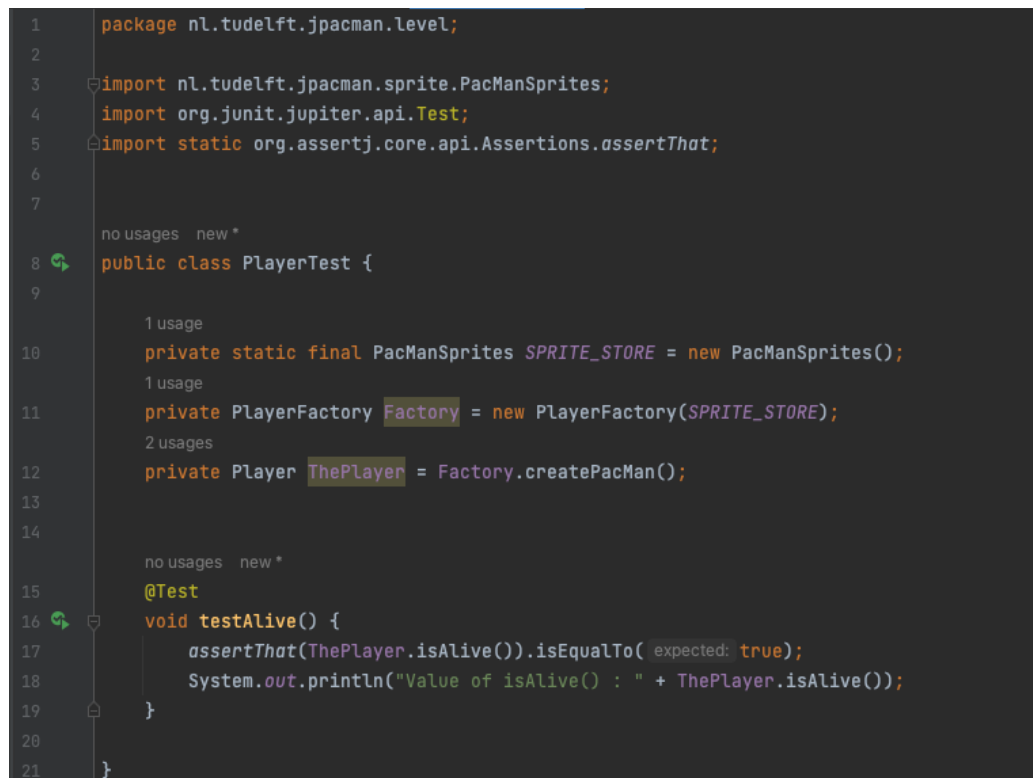


The screenshot shows the 'Coverage' window in IntelliJ IDEA for the test suite 'Tests in 'jpacman.test''. The table displays coverage metrics for various elements, including packages, classes, and methods. The 'Element' column lists the hierarchy from 'nl' down to specific classes like 'Launcher' and 'PacmanConfigurationException'. The 'Class, %' column shows the percentage of class coverage, 'Method, %' shows method coverage, and 'Line, %' shows line coverage. The 'board' package shows 20% class coverage, while most other packages and classes show 0% coverage.

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.

Task 2



The screenshot shows a Java code editor with the following code:

```
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
8 public class PlayerTest {
9
10     private static final PacManSprites SPRITE_STORE = new PacManSprites();
11     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
12     private Player ThePlayer = Factory.createPacMan();
13
14
15     @Test
16     void testAlive() {
17         assertThat(ThePlayer.isAlive()).isEqualTo(expected: true);
18         System.out.println("Value of isAlive() : " + ThePlayer.isAlive());
19     }
20
21 }
```

Coverage: Tests in 'jpacman.test' x			
Element	Class, %	Method, %	Line, %
nl	16% (18/112)	9% (60/634)	8% (190/2330)
tudelft	16% (18/112)	9% (60/634)	8% (190/2330)
jpacman	16% (18/112)	9% (60/634)	8% (190/2330)
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% (0/22)	0% (0/104)	0% (0/498)
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

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

```

1  package nl.tudelft.jpacman.level;
2
3  import nl.tudelft.jpacman.sprite.PacManSprites;
4  import nl.tudelft.jpacman.sprite.Sprite;
5  import org.junit.jupiter.api.Test;
6
7  no usages new *
8  public class PelletTest {
9
10     1 usage
11     private static final PacManSprites SPRITE_STORE = new PacManSprites();
12     1 usage
13     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
14     no usages
15     private Player ThePlayer = Factory.createPacMan();
16     1 usage
17     int points;
18     1 usage
19     Sprite SPRITE;
20     1 usage
21     public Pellet PELLET = new Pellet(points, SPRITE);
22
23     no usages new *
24     @Test
25     void testGetValue(){
26         System.out.println("Value of PELLET.getValue() : " + PELLET.getValue());
27     }
28 }

```

Coverage: Tests in 'jpacman.test' ×			
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
3  import nl.tudelft.jpacman.sprite.PacManSprites;
4  import org.junit.jupiter.api.Test;
5
6  no usages new *
7  public class GhostFactoryTest {
8
9      1 usage
10     private static final PacManSprites SPRITE_STORE = new PacManSprites();
11
12     1 usage
13     GhostFactory tempGhost = new GhostFactory(SPRITE_STORE);
14
15     no usages new *
16     @Test
17     void testGhostFactory(){
18
19         System.out.println("Value of createBlinky() : " + tempGhost.createBlinky());
20     }
21 }

```

Coverage: Tests in 'jpacman.test' x			
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
3
4  import nl.tudelft.jpacman.level.PlayerFactory;
5  import nl.tudelft.jpacman.sprite.PacManSprites;
6  import org.junit.jupiter.api.Test;
7
8  no usages new *
9  public class GameFactoryTest {
10     1 usage
11     private static final PacManSprites SPRITE_STORE = new PacManSprites();
12     1 usage
13     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
14     no usages new *
15     @Test
16     void testGameFactory(){
17         GameFactory GAME_FACTORY = new GameFactory(Factory);
18     }
19 }

```

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)
PacmanConfigurationException	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

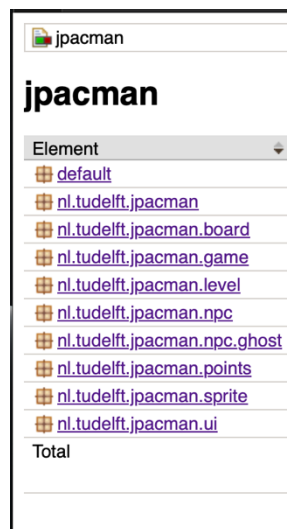
Task 3

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:

jpacman												
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default	<div><div></div></div>	0%	<div><div></div></div>	0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman	<div><div></div></div>	69%	<div><div></div></div>	25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.board	<div><div></div></div>	86%	<div><div></div></div>	58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman.game	<div><div></div></div>	87%	<div><div></div></div>	60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.level	<div><div></div></div>	67%	<div><div></div></div>	57%	74	155	104	344	21	69	4	12
nl.tudelft.jpacman.npc	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	4	0	8	0	4	0	1
nl.tudelft.jpacman.npc.ghost	<div><div></div></div>	71%	<div><div></div></div>	55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.points	<div><div></div></div>	60%	<div><div></div></div>	75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.sprite	<div><div></div></div>	87%	<div><div></div></div>	59%	29	70	10	113	4	38	0	5
nl.tudelft.jpacman.ui	<div><div></div></div>	77%	<div><div></div></div>	47%	54	86	21	144	7	31	0	6
Total	1,211 of 4,694	74%	293 of 637	54%	292	590	228	1,039	50	268	6	47

Created with JaCoCo 0.8.3.201901230119