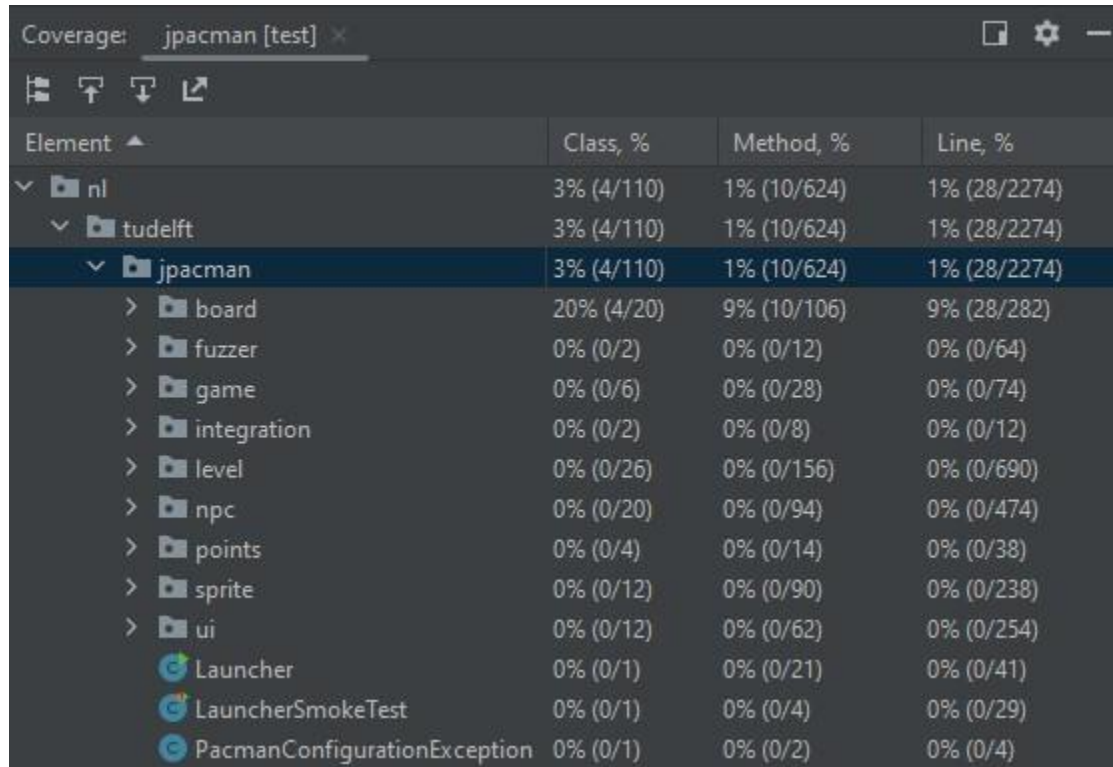


Testing Report

Kristy Nguyen
CS 472-1001

Task 1: JPacman Test Coverage

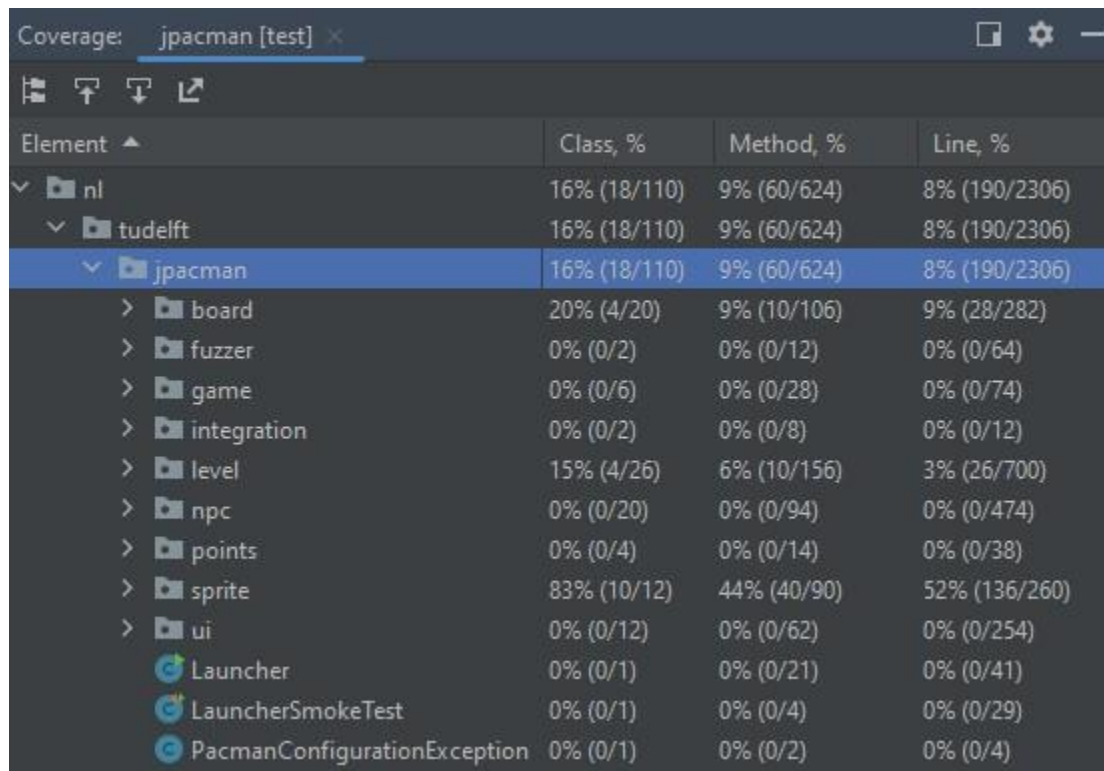


The screenshot shows the IntelliJ IDEA Coverage tool window for the test run 'jpacman [test]'. The window displays a tree view of the project structure with columns for Element, Class, %, Method, %, and Line, %. The 'jpacman' package is highlighted, showing its sub-packages and classes with their respective coverage percentages and counts.

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)

The code coverage report from IntelliJ initially shows 3% (4/110) class coverage, 1% (10/624) method coverage, and 1% (28/2274) line coverage.

Task 2: Increasing Coverage on JPacman

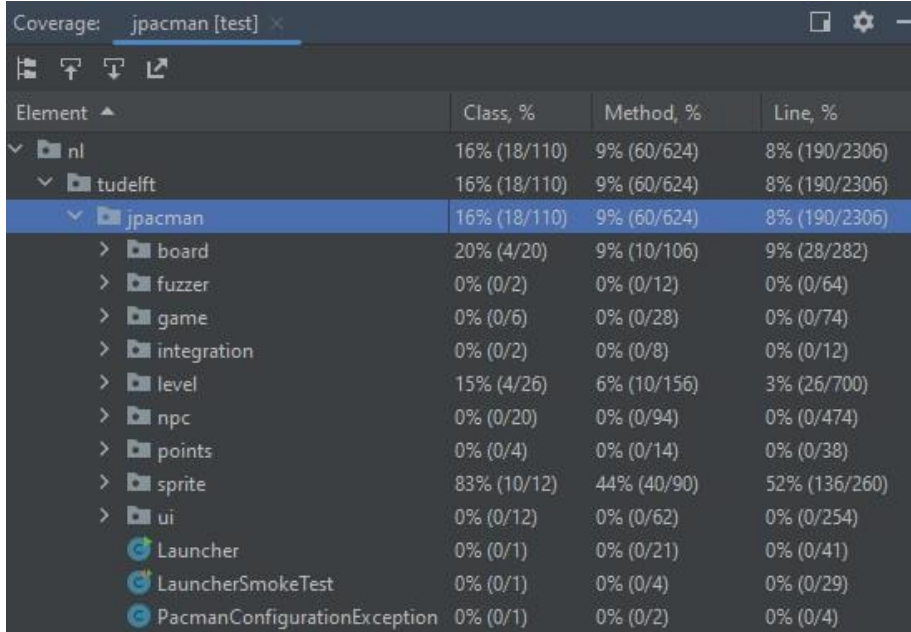
The screenshot shows the 'Coverage: jpacman [test]' window in IntelliJ IDEA. It displays a tree view of the project structure with coverage percentages for classes, methods, and lines. The 'jpacman' package is highlighted, showing 16% class coverage (18/110), 9% method coverage (60/624), and 8% line coverage (190/2306).

Element	Class, %	Method, %	Line, %
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)

After implementing `isAlive()`, the code coverage showed an increase in coverage, showing 16% (18/110) class coverage, 9% (60/624) method coverage, and 8% (190/2306) line coverage.

Task 2.1

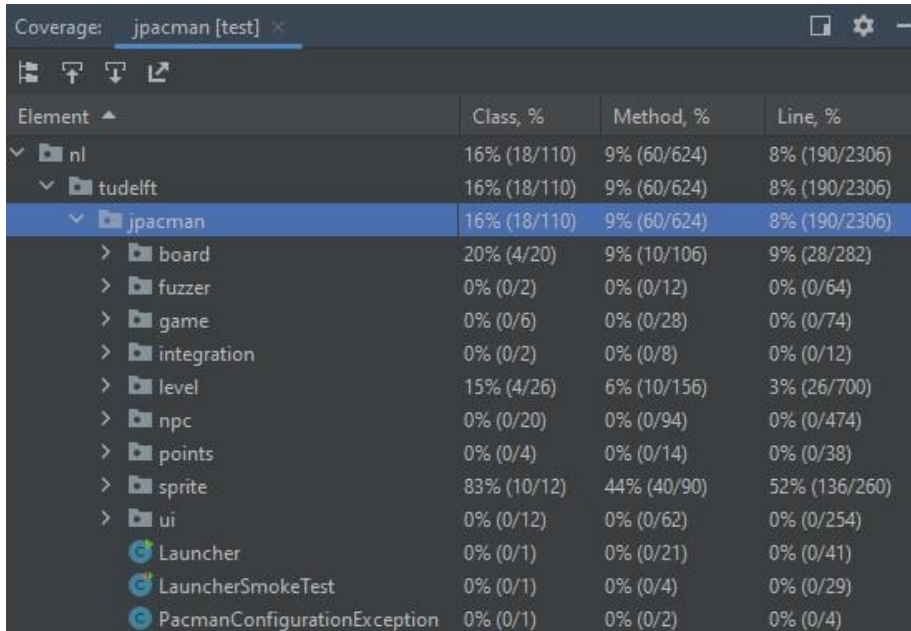
Before implementing unit test for `getDeltaY()`



Coverage: jpacman [test] ×

Element	Class, %	Method, %	Line, %
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)

After implementing unit test for `getDeltaY()`



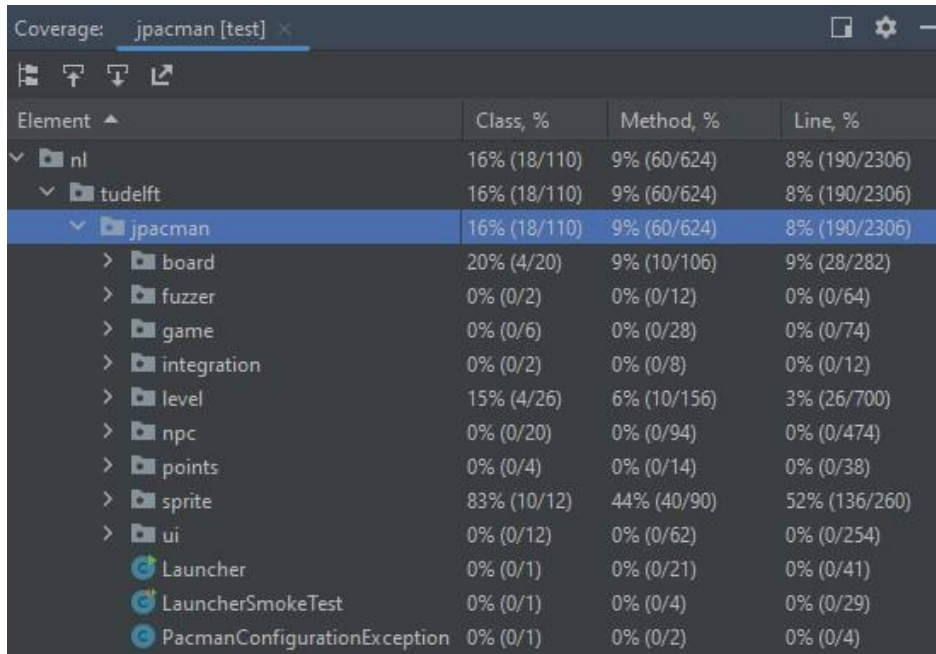
Coverage: jpacman [test] ×

Element	Class, %	Method, %	Line, %
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)

`testGetDeltaY()` code snippet

```
@Test
void testGetDeltaY() {
    Direction TheDirection = Direction.SOUTH;
    assertThat(TheDirection.getDeltaY()).isEqualTo(expected: 1);
}
```

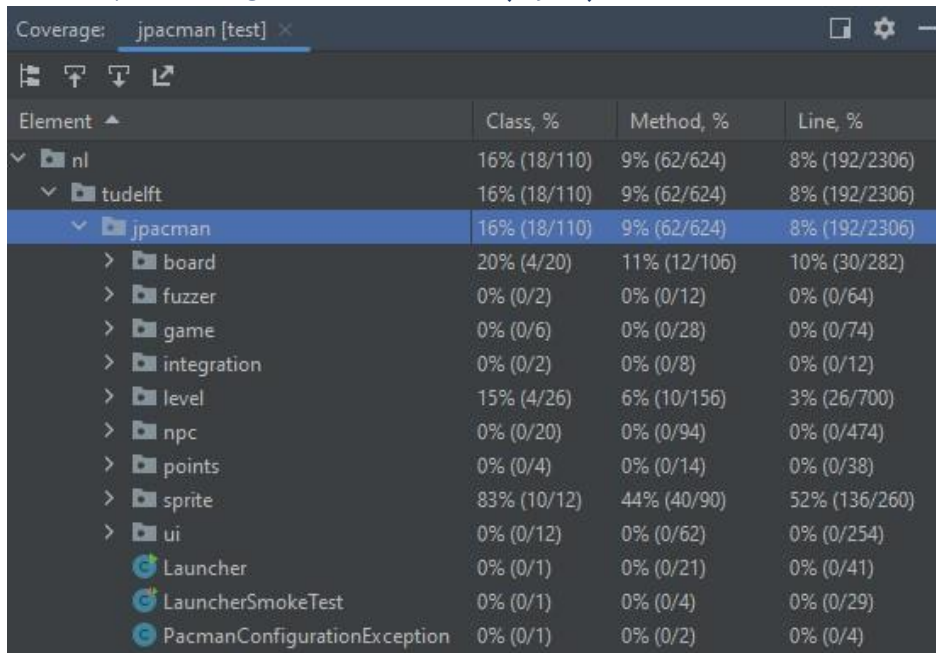
Before implementing unit test for `WEST(-1, 0)`



Coverage: jpacman [test]

Element	Class, %	Method, %	Line, %
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)

After implementing unit test for `WEST(1, 0)`



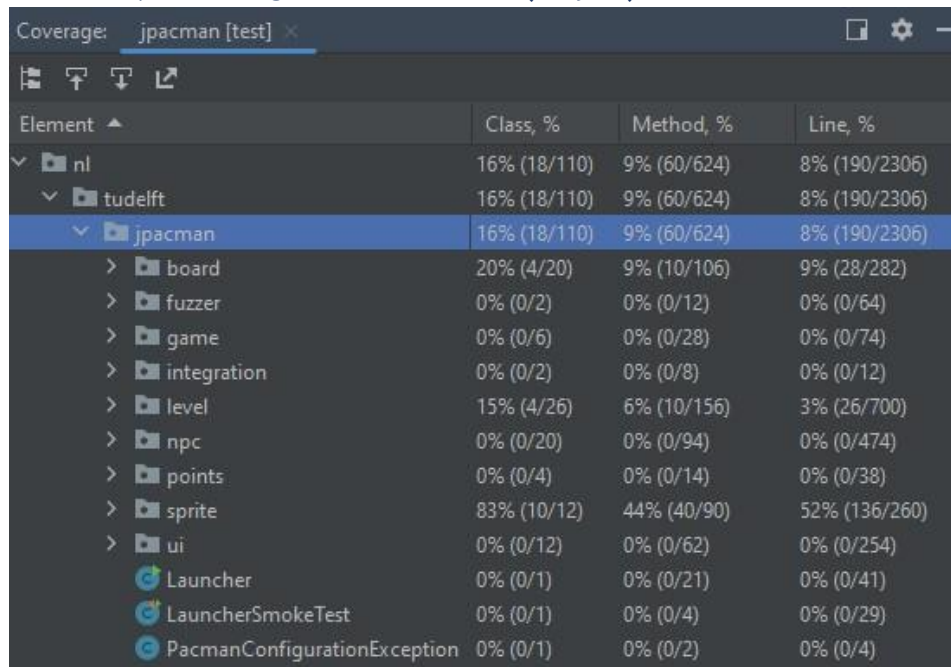
Coverage: jpacman [test]

Element	Class, %	Method, %	Line, %
nl	16% (18/110)	9% (62/624)	8% (192/2306)
tudelft	16% (18/110)	9% (62/624)	8% (192/2306)
jpacman	16% (18/110)	9% (62/624)	8% (192/2306)
board	20% (4/20)	11% (12/106)	10% (30/282)
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)

`testWest()` code snippet

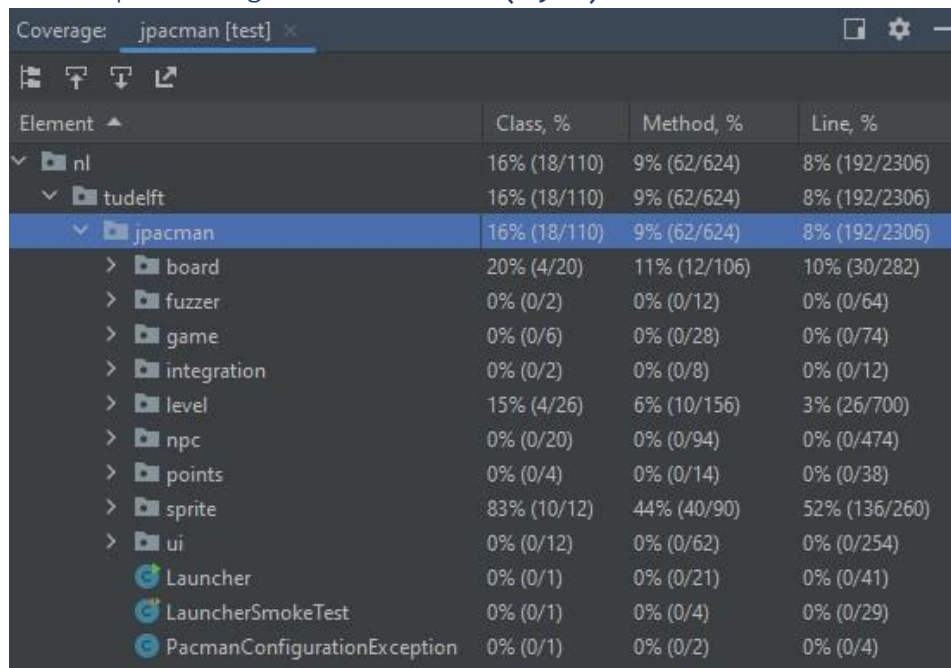
```
@Test
void testWest() {
    Direction west = Direction.valueOf( name: "WEST");
    assertThat(west.getDeltaX()).isEqualTo( expected: -1);
}
```

Before implementing unit test for EAST(-1, 0)



Element	Class, %	Method, %	Line, %
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)

After implementing unit test for EAST(1, 0)



Element	Class, %	Method, %	Line, %
nl	16% (18/110)	9% (62/624)	8% (192/2306)
tudelft	16% (18/110)	9% (62/624)	8% (192/2306)
jpacman	16% (18/110)	9% (62/624)	8% (192/2306)
board	20% (4/20)	11% (12/106)	10% (30/282)
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)

testEast() code snippet

```
@Test
void testEast() {
    Direction east = Direction.valueOf( name: "EAST");
    assertThat(east.getDeltaX()).isEqualTo( expected: 1);
}
```

Task 3: JaCoCo Report on JPacman

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

The coverage results from JaCoCo are not similar to the ones from IntelliJ because JaCoCo's report of the coverage only reports 268 methods while IntelliJ reports 624 methods. Other than the number of methods covered, JaCoCo reports missed instructions but also reports more coverage than IntelliJ.

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

I found the source code visualization from JaCoCo on uncovered branches useful because it shows a progress bar for each file or folder and the missing coverage.

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

I prefer JaCoCo's because it reports more about the code coverage, such as the visualization, the missing coverage, the total, and is much simpler to look at when you look at the coverage of each folder.

Link to Fork Repository

<https://github.com/cs472-team6/cs472-team6>