

Repository:

<https://github.com/yaaacii/jpacman>

Task 1:

Before isAlive()

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)
Board	0% (0/1)	0% (0/7)	0% (0/17)
BoardFactory	0% (0/3)	0% (0/11)	0% (0/27)
BoardFactoryTest	0% (0/1)	0% (0/8)	0% (0/18)
BoardTest	0% (0/1)	0% (0/3)	0% (0/3)
Direction	100% (1/1)	75% (3/4)	90% (10/11)
Square	0% (0/1)	0% (0/8)	0% (0/23)
SquareTest	0% (0/1)	0% (0/4)	0% (0/13)
Unit	100% (1/1)	20% (2/10)	13% (4/29)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
JPacmanFuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/6)	0% (0/28)	0% (0/74)
Game	0% (0/1)	0% (0/7)	0% (0/24)
GameFactory	0% (0/1)	0% (0/3)	0% (0/4)
SinglePlayerGame	0% (0/1)	0% (0/4)	0% (0/9)
integration	0% (0/2)	0% (0/8)	0% (0/12)
StartupSystemTest	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/26)	0% (0/156)	0% (0/690)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	0% (0/1)	0% (0/8)	0% (0/20)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	0% (0/1)	0% (0/3)	0% (0/4)
npc	0% (0/20)	0% (0/94)	0% (0/472)

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()

Element	Class, %	Method, %	Line, %
nl	17% (20/112)	9% (62/628)	8% (200/2274)
tudelft	17% (20/112)	9% (62/628)	8% (200/2274)
jpacman	17% (20/112)	9% (62/628)	8% (200/2274)
board	20% (4/20)	7% (8/106)	9% (26/272)
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	21% (6/28)	8% (14/160)	5% (38/680)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/101)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/67)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	25% (2/8)	33% (8/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
PlayerTest	100% (1/1)	100% (2/2)	100% (6/6)
npc	0% (0/20)	0% (0/94)	0% (0/472)
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:

Test1:

Before updateObservers():

levelTest.java

```
import nl.tudelft.jpacman.boss.NpcGhast;
import nl.tudelft.jpacman.npc.ghost.Ghast;
import nl.tudelft.jpacman.points.PointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions;
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;

no usages
public class levelTest {
    // private Level level = mock(Level.class);
    // @Test
    // void testObservers(){
    //     level.updateObservers();
    //     verify(level).updateObservers();
    // }
}
```

Coverage: Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	20% (22/110)	12% (80/624)	10% (250/23...
tudelft	20% (22/110)	12% (80/624)	10% (250/23...
jpacman	20% (22/110)	12% (80/624)	10% (250/23...
board	20% (4/20)	13% (14/106)	12% (36/282)
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	30% (8/26)	15% (24/156)	9% (70/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	62% (5/8)	66% (16/24)
PlayerCollisions	100% (1/1)	42% (3/7)	35% (10/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/26...
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 updateObservers():

The screenshot shows an IDE with two panels. The left panel displays the code for `LevelTest.java`, which includes imports for `nl.tudelft.jpacman.npc.ghost`, `nl.tudelft.jpacman.points.PointCalculator`, `nl.tudelft.jpacman.sprite.PacmanSprite`, `org.junit.jupiter.api.BeforeEach`, `org.junit.jupiter.api.Test`, `org.assertj.core.api.Assertions`, and `org.mockito.Mockito`. The code defines a `LevelTest` class with a `private Level level = mock(Level.class)` and a `@Test` method `testObservers()` that calls `level.updateObservers()` and `verify(level).updateObservers()`.

The right panel shows a coverage report for 'Tests in 'jpacman.test''. The report is organized into a table with columns for Element, Class, %, Method, %, and Line, %.

Element	Class, %	Method, %	Line, %
nl	21% (24/110)	13% (82/624)	10% (252/23...)
tudelft	21% (24/110)	13% (82/624)	10% (252/23...)
jpacman	21% (24/110)	13% (82/624)	10% (252/23...)
board	20% (4/20)	13% (14/106)	12% (36/284)
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	38% (10/26)	16% (26/156)	10% (72/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	5% (1/17)	0% (1/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	62% (5/8)	66% (16/24)
PlayerCollisions	100% (1/1)	42% (3/7)	35% (10/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/26...)
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)

Test2:

Before PlayerVersusPellet():

The screenshot shows an IDE with two panels. The left panel displays the code for `PlayerCollisionTest.java`, which includes imports for `nl.tudelft.jpacman.npc.ghost`, `nl.tudelft.jpacman.points.PointCalculator`, `nl.tudelft.jpacman.sprite.PacmanSprite`, `org.junit.jupiter.api.BeforeEach`, `org.junit.jupiter.api.Test`, `org.assertj.core.api.Assertions`, and `org.mockito.Mockito`. The code defines a `PlayerCollisionTest` class with a `private Ghost theGhost = mock(Ghost.class)` and a `@Test` method `testPlayerVPellet()` that calls `ThePlayer.getScore()`, `ThePlayerCollision.playerVersusPellet()`, `verify(ThePellet, times(0)).leave()`, and `assertThat(ThePlayer.getScore() > 0)`.

The right panel shows a coverage report for 'Tests in 'jpacman.test''. The report is organized into a table with columns for Element, Class, %, Method, %, and Line, %.

Element	Class, %	Method, %	Line, %
nl	21% (24/110)	11% (74/624)	10% (236/23...)
tudelft	21% (24/110)	11% (74/624)	10% (236/23...)
jpacman	21% (24/110)	11% (74/624)	10% (236/23...)
board	20% (4/20)	9% (10/106)	9% (28/284)
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	38% (10/26)	14% (22/156)	8% (64/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	5% (1/17)	0% (1/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	50% (4/8)	62% (15/24)
PlayerCollisions	100% (1/1)	28% (2/7)	25% (7/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/26...)
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 PlayerVersusPellet():

The screenshot shows the IDE with the file `PlayerCollisionTest.java` open. The code includes a `private Ghost TheGhost = mock(Ghost.class);` line and two test methods: `testPlayerVPellet()` and `testPlayerVGhost()`. The `testPlayerVPellet()` method calls `ThePlayerCollision.playerVersusPellet()` and `verify(ThePellet, times(0)).leave`. The `testPlayerVGhost()` method calls `ThePlayerCollision.playerVersusGhost` and `assertThat(ThePlayer.isAlive()).isE`. The coverage table on the right shows the following data:

Element	Class, %	Method, %	Line, %
nl	21% (24/110)	13% (82/624)	10% (252/23...)
tudelft	21% (24/110)	13% (82/624)	10% (252/23...)
jpacman	21% (24/110)	13% (82/624)	10% (252/23...)
board	20% (4/20)	13% (14/106)	12% (36/284)
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	38% (10/26)	16% (26/156)	10% (72/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	5% (1/17)	0% (1/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	62% (5/8)	66% (16/24)
PlayerCollisions	100% (1/1)	42% (3/7)	35% (10/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/26...)
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)

Test 3:

Before playerVersusGhost():

The screenshot shows the IDE with the file `PlayerCollisionTest.java` open. The code includes a `private Ghost TheGhost = mock(Ghost.class);` line and two test methods: `testPlayerVPellet()` and `testPlayerVGhost()`. The `testPlayerVPellet()` method calls `ThePlayerCollision.playerVersusPellet()` and `verify(ThePellet, times(0)).leave`. The `testPlayerVGhost()` method calls `ThePlayerCollision.playerVersusGhost` and `assertThat(ThePlayer.isAlive()).isE`. The coverage table on the right shows the following data:

Element	Class, %	Method, %	Line, %
nl	21% (24/110)	11% (74/624)	9% (222/23...)
tudelft	21% (24/110)	11% (74/624)	9% (222/23...)
jpacman	21% (24/110)	11% (74/624)	9% (222/23...)
board	20% (4/20)	13% (14/106)	12% (36/284)
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	38% (10/26)	12% (20/156)	6% (50/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	5% (1/17)	0% (1/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	37% (3/8)	37% (9/24)
PlayerCollisions	100% (1/1)	28% (2/7)	21% (6/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
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 playerVersusGhost():

The screenshot displays an IDE with two main panels. The left panel shows code snippets for `NavigationTest.java` and `PlayerCollisionTest.java`. The right panel shows a coverage report for 'Tests in 'jpacman.test''.

Code Snippets:

```

1 usage
private final PointCalculator
2 usages
public PlayerCollisions ThePlayerCollis

1 usage
private Ghost TheGhost = mock(Ghost.cte

no usages
@Test
void testPlayerVPellet() {
    int beforeScore = ThePlayer.getScore
    ThePlayerCollision.playerVersusPellet
    verify(ThePellet, times(0)).leave
    assertThat(actual: ThePlayer.getScore

//
//
no usages
@Test
void testPlayerVGhost() {
    ThePlayerCollision.playerVersusGhos
    assertThat(ThePlayer.isAlive()).isE
    verify(ThePlayer, times(0)).setKi
}

```

Coverage Report: Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	21% (24/110)	13% (82/624)	10% (252/23...
tudelft	21% (24/110)	13% (82/624)	10% (252/23...
jpacman	21% (24/110)	13% (82/624)	10% (252/23...
board	20% (4/20)	13% (14/106)	12% (36/284)
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	38% (10/26)	16% (26/156)	10% (72/716)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	5% (1/17)	0% (1/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	100% (1/1)	33% (1/3)	66% (4/6)
Player	100% (1/1)	62% (5/8)	66% (16/24)
PlayerCollisions	100% (1/1)	42% (3/7)	35% (10/28)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/20)	0% (0/94)	0% (0/474)
points	0% (0/4)	0% (0/14)	0% (0/38)
sprite	83% (10/12)	46% (42/90)	55% (144/26...
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)

Code Snippets:

```
1 package nl.tudelft.jpacman.level;
2
3 import org.junit.jupiter.api.Test;
4
5 import static org.mockito.Mockito.mock;
6 import static org.mockito.Mockito.verify;
7
8 no usages
9 public class LevelTest {
10     2 usages
11     private Level level = mock(Level.class);
12
13     no usages
14     @Test
15     void testObservers(){
16         level.updateObservers();
17         verify(level).updateObservers();
18     }
19 }
```

nl.tudelft.jpacman.level.Level
void updateObservers()
Updates the observers about the state of this level.
jpacman.main

```
5 public class PlayerCollisionTest {
6     1 usage
7     private static final PacManSprites SPRITE_STORE = new PacManSprites();
8     1 usage
9     private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
10    5 usages
11    private Player ThePlayer = Factory.createPacMan();
12    1 usage
13    private static final EmptySprite emptySprite = new EmptySprite();
14    1 usage
15    public Pellet ThePellet = new Pellet( points: 1, emptySprite);
16    1 usage
17    private final PointCalculator pointCalc = mock(PointCalculator.class);
18    2 usages
19    public PlayerCollisions ThePlayerCollision = new PlayerCollisions(pointCalc);
20    1 usage
21    private Ghost TheGhost = mock(Ghost.class);
22    no usages
23    @Test
24    void testPlayerVPellet() {
25        int beforeScore = ThePlayer.getScore();
26        ThePlayerCollision.playerVersusPellet(ThePlayer, ThePellet);
27        // verify(ThePellet, times(0)).leaveSquare();
28        // assertTrue(actual: ThePlayer.getScore() > beforeScore);
29    }
30
31    no usages
32    @Test
33    void testPlayerVGhost() {
34        ThePlayerCollision.playerVersusGhost(ThePlayer, TheGhost);
35        assertTrue(ThePlayer.isAlive()).isEqualTo( expected: false);
36        // verify(ThePlayer, times(0)).setKiller(any());
37    }
38 }
```

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. /**
70.  * Actual case of player bumping into ghost or vice versa.
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.  * @param player
85.  *     The player involved in the collision.
86.  * @param pellet
87.  *     The pellet involved in the collision.
88.  */
89. public void playerVersusPellet(Player player, Pellet pellet) {
90.     pointCalculator.consumedAPellet(player, pellet);
91.     pellet.leavesSquare();
92. }
93.
94.
95. }
```