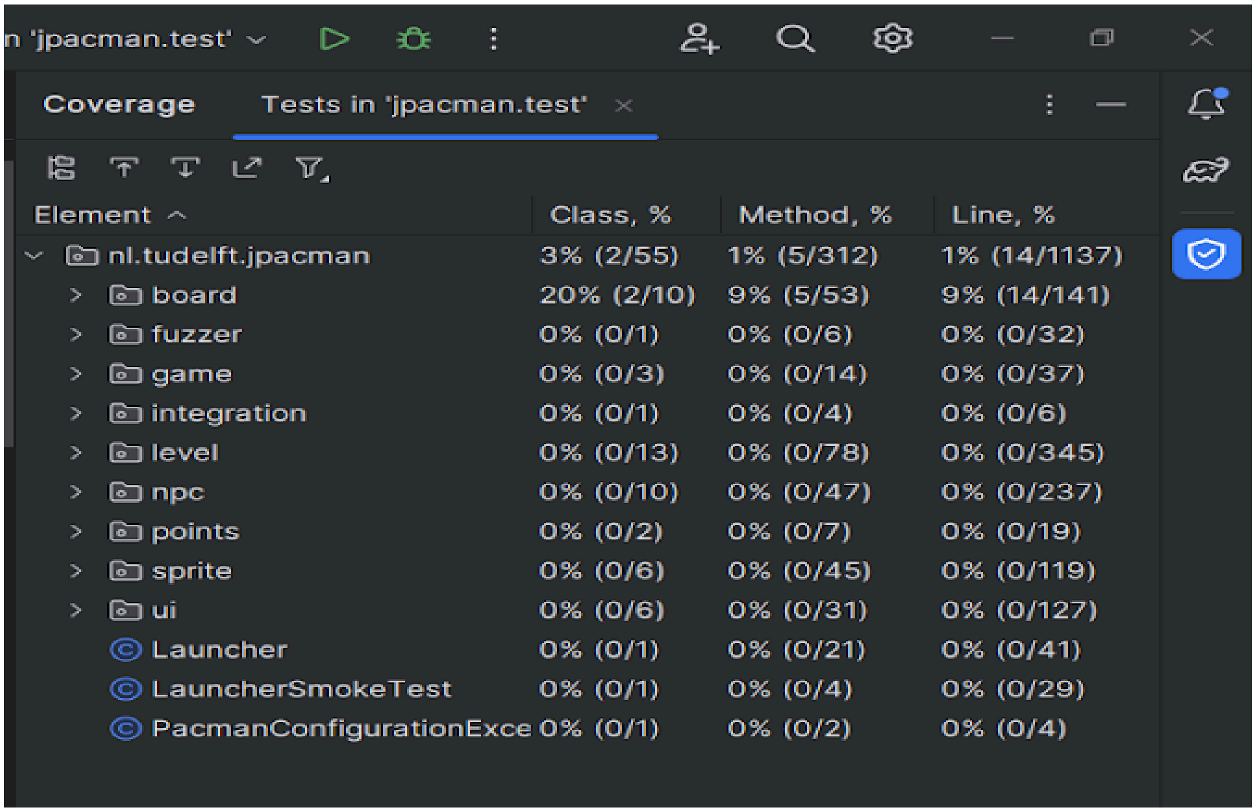
Kelly Esquejo

CS 472

Dynamic Analysis

Task 1 - JPacman Test Coverage:



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/10)	9% (5/53)	9% (14/141)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	0% (0/13)	0% (0/78)	0% (0/345)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	0% (0/6)	0% (0/45)	0% (0/119)
ui	0% (0/6)	0% (0/31)	0% (0/127)
Launcher	0% (0/1)	0% (0/21)	0% (0/41)
LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
PacmanConfigurationExce	0% (0/1)	0% (0/2)	0% (0/4)

The image above is the initial run of jpacman [test] with Coverage. The percentages indicate that it is a low test coverage, which is not good, because there are barely any tests for the methods.

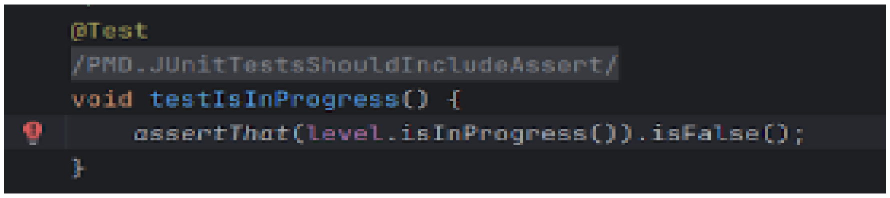
TASK 2 - Increasing Coverage on JPacman:

Below are the three methods I choose for task 2. Since there are over 300 methods in jpacman, the change of percentage for the coverage on jpacman after testing three methods will not be noticeable compared to having a test for all of the methods for one class, such as Player or Level.

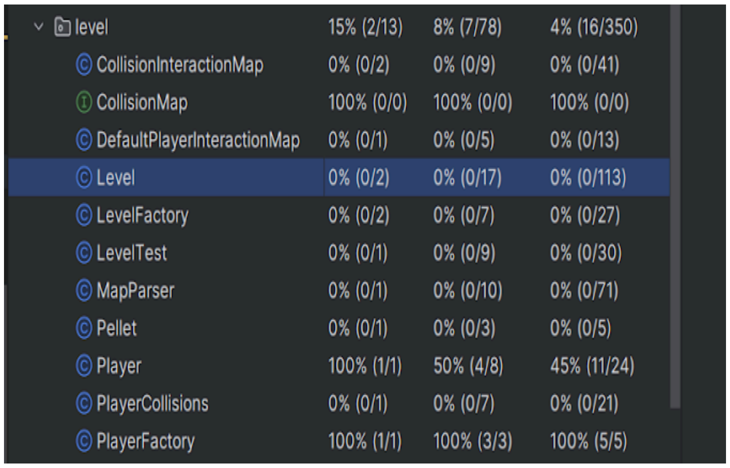
src/main/java/nl/tudelft/jpacman/level/Level.isInProgress
src/main/java/nl/tudelft/jpacman/level/Level.isAnyPlayerAlive
src/main/java/nl/tudelft/jpacman/level/Player.addPoints

Before each test, testIsInProgress and testIsAnyPlayerAlive, the set up consisted of instantiating board, ghosts, square, and collisionmap objects for the Level constructor. Because the game had not started, isInprogress should return false. Furthermore, since no player has been registered, isAnyPlayerAlive should also return false.

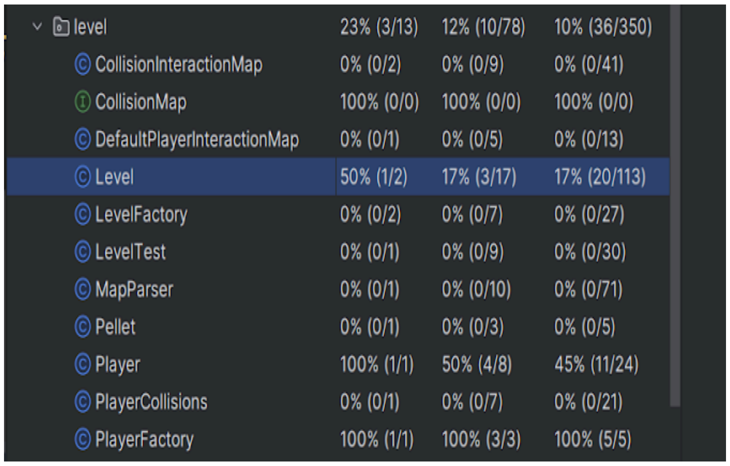
Level.isInProgress

- **CODE SNIPPET:**
  - 

```
@Test
/PMD.JUnitTestsShouldIncludeAssert/
void testIsInProgress() {
    assertThat(level.isInProgress()).isFalse();
}
```

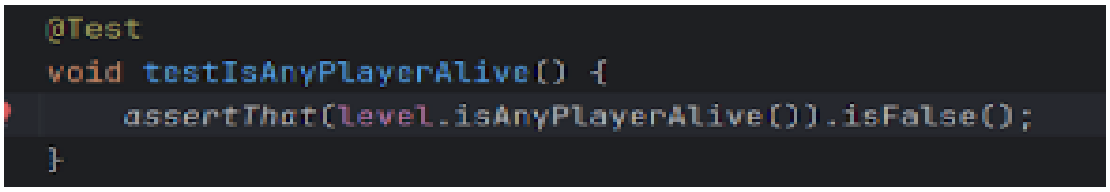


level	15% (2/13)	8% (7/78)	4% (16/350)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)

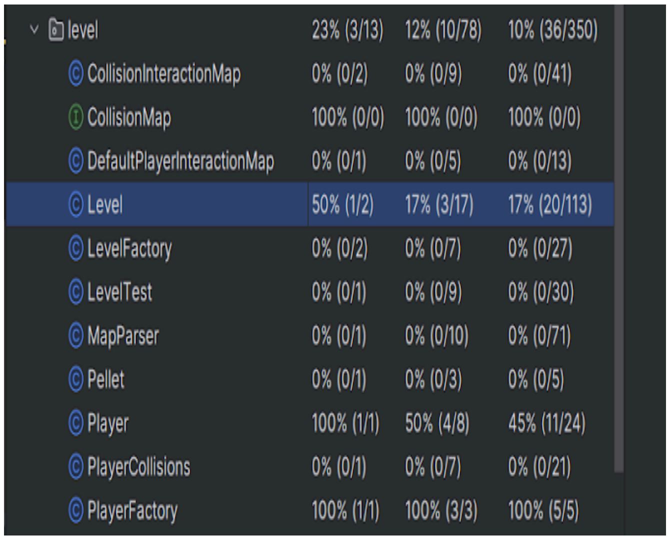


level	23% (3/13)	12% (10/78)	10% (36/350)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	17% (3/17)	17% (20/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)

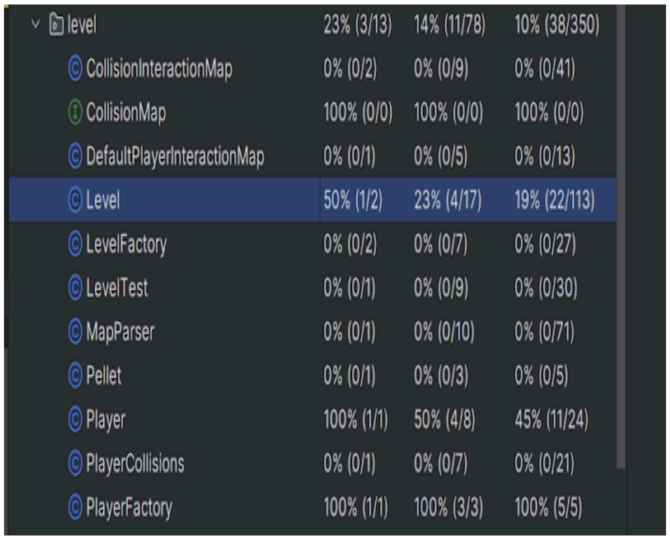
Level.isAnyPlayerAlive

- **CODE SNIPPET:**
    - 

```
@Test
void testIsAnyPlayerAlive() {
    assertThat(level.isAnyPlayerAlive()).isFalse();
}
```



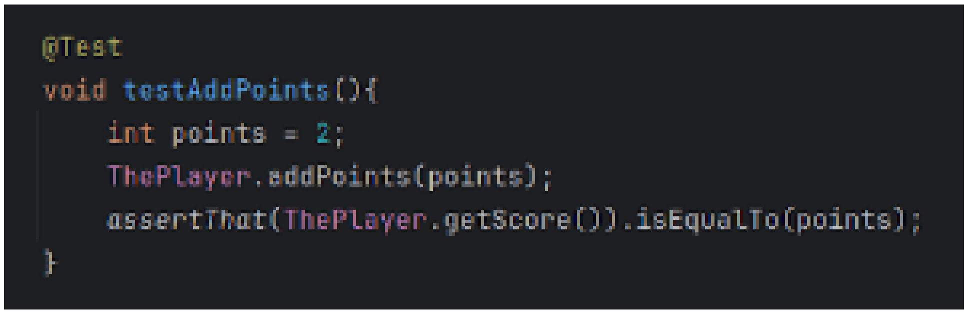
level	23% (3/13)	12% (10/78)	10% (36/350)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	17% (3/17)	17% (20/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)



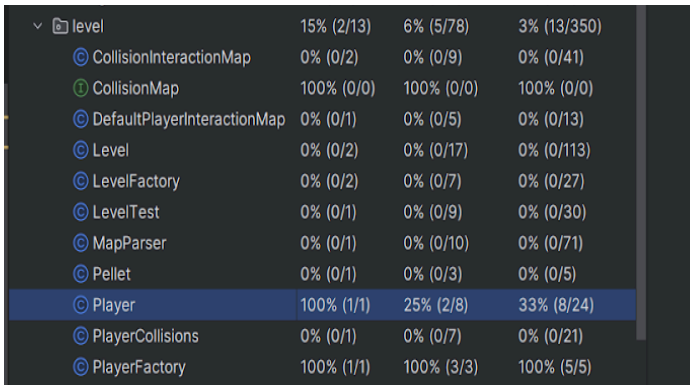
level	23% (3/13)	14% (11/78)	10% (38/350)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	50% (1/2)	23% (4/17)	19% (22/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)

For testAddPoints, it also was not complicated as the first test, testAlive, had created the class PlayerTest and had instantiated the objects required for testing. To test addPoints, I passed an int of 2 to the method and using getScore(), I was able to test if it was equivalent to the points expected.

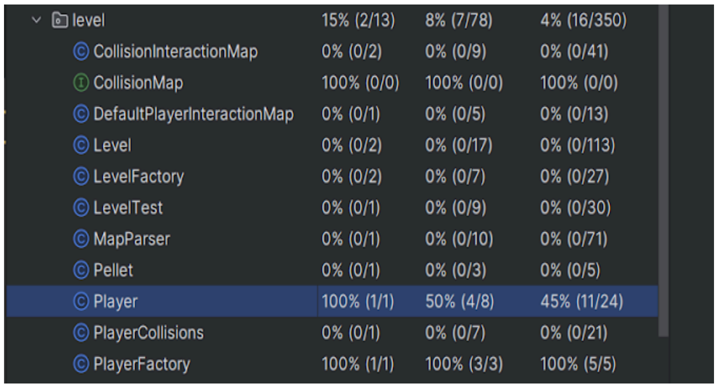
Player.addPoints

- **CODE SNIPPET:**
  - 

```
@Test
void testAddPoints(){
    int points = 2;
    ThePlayer.addPoints(points);
    assertThat(ThePlayer.getScore()).isEqualTo(points);
}
```

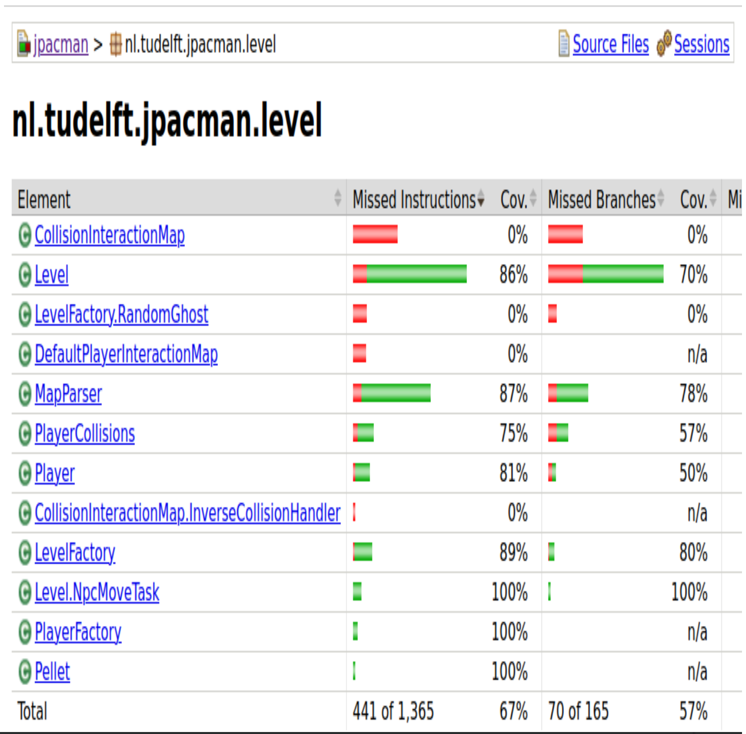


level	15% (2/13)	6% (5/78)	3% (13/350)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractionMap	0% (0/1)	0% (0/5)	0% (0/13)
Level	0% (0/2)	0% (0/17)	0% (0/113)
LevelFactory	0% (0/2)	0% (0/7)	0% (0/27)
LevelTest	0% (0/1)	0% (0/9)	0% (0/30)
MapParser	0% (0/1)	0% (0/10)	0% (0/71)
Pellet	0% (0/1)	0% (0/3)	0% (0/5)
Player	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)



level	15% (2/13)	8% (7/78)	4% (16/350)
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	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)

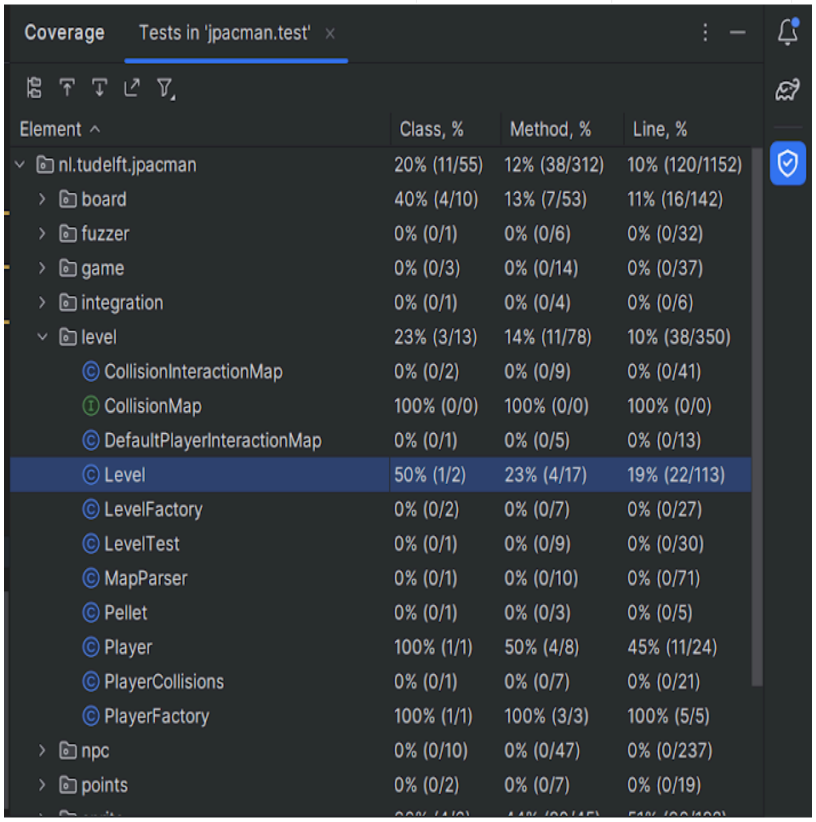
Task 3 - JaCoCo Report on JPacman:



jpacman > nl.tudelft.jpacman.level   Source Files Sessions

nl.tudelft.jpacman.level

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Mi
CollisionInteractionMap	🟥	0%	🟥	0%	
Level	🟥🟩	86%	🟥🟩	70%	
LevelFactory.RandomGhost	🟥	0%	🟥	0%	
DefaultPlayerInteractionMap	🟥	0%		n/a	
MapParser	🟥🟩	87%	🟩	78%	
PlayerCollisions	🟩	75%	🟥🟩	57%	
Player	🟩	81%	🟥🟩	50%	
CollisionInteractionMap.InverseCollisionHandler	🟥	0%		n/a	
LevelFactory	🟩	89%	🟩	80%	
Level.NpcMoveTask	🟩	100%	🟩	100%	
PlayerFactory	🟩	100%		n/a	
Pellet	🟩	100%		n/a	
Total	441 of 1,365	67%	70 of 165	57%	



Coverage   Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	20% (11/55)	12% (38/312)	10% (120/1152)
board	40% (4/10)	13% (7/53)	11% (16/142)
fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
game	0% (0/3)	0% (0/14)	0% (0/37)
integration	0% (0/1)	0% (0/4)	0% (0/6)
level	23% (3/13)	14% (11/78)	10% (38/350)
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)	23% (4/17)	19% (22/113)
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	100% (1/1)	50% (4/8)	45% (11/24)
PlayerCollisions	0% (0/1)	0% (0/7)	0% (0/21)
PlayerFactory	100% (1/1)	100% (3/3)	100% (5/5)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)

Intellij's coverage results check if the classes, methods, and lines are being tested. While JaCoCo's coverage results check the difference of tested instructions or branches over total instructions or branches of an element.

I did not find JaCoCo's source code visualization on uncovered branches more helpful than IntelliJ.

I preferred working with IntelliJ's coverage window rather than JaCoCo's report. The coverage window, combined with IntelliJ's visual of what lines were covered or not, the red or green indicators next to the line numbers, was a better experience for me as a new unit tester.

Link to my fork repository:
https://github.com/esquek1/SeniorDesign.git