Name: Jose A. Garcia

Date: 2/2/23 Class: CS 472

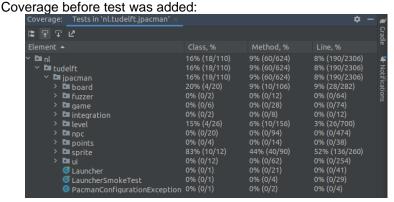
Testing lab

Task 2.1:

II.

III.

 $\underline{\text{Coverage with the 1}^{\text{ST}} \text{ test:}} \textbf{src/main/java/nl/tudelft/jpacman/level/LevelFactory.java@createPellet()}$



Coverage after test was added:

```
Element ▲ Class, % Method, % Line, %

➤ In | 25% (28/110) | 12% (80/624) | 10% (246/2330) |

➤ In | 25% (28/110) | 12% (80/624) | 10% (246/2330) |

➤ In | 12% (80/624) | 10% (246/2330) |

➤ In | 12% (80/624) | 10% (246/2330) |

➤ In | 12% (80/624) | 10% (246/2330) |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

➤ In | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 |

■ | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10/2 | 10
```

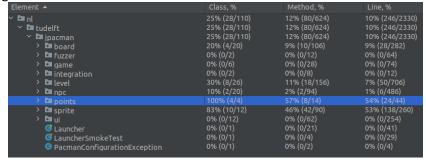
Test source code PelletTest.java

- a. Note that whenever I create a new pellet in the constructor a default value of 10 is set to its *value* attribute thus my thinking of testing it the way shown above.
- b. Also note that by adding this test my level class coverage incremented by 15%.and also my method coverage incremented by 5%.

Coverage with the 2nd test: src/main/java/nl/tudelft/jpacman/level/LevelFactory.java@createGhost()

 Please note I modified the source code to include a getter for ghost index to keep track of the number of ghosts incrementing

Coverage before test was added:

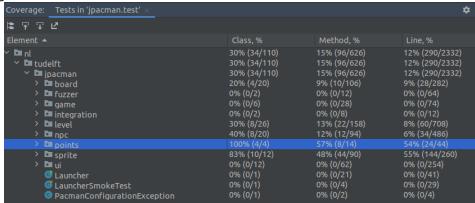


Coverage after test was added:

I.

II.

III.



Test source code GhostTest.java:

```
public class @hostTest{
// Create a levelFactory to create a pellet
// Create a PacmanSprites
2 usages
private final PacManSprites sprite_loader = new PacManSprites();
// Create a GhostFactory
1 usage
private @hostFactory ghFactory = new @hostFactory(sprite_loader);
// Create a pointCalculator
1 usage
private PointCalculatorLoader tempCalc = new PointCalculatorLoader();

// Create a level factory
2 usages
private LevelFactory Lvl_factory = new LevelFactory(sprite_loader, ghFactory, tempCalc.load());

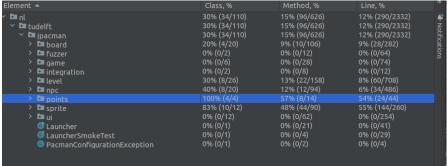
// Create a ghost for the level factory
no usages
private Ghost test_ghost = Lvl_factory.createGhost();

// If the ghostIndex is greater than -1(default value) that means that a ghost was created
no usages new*
@Test
void test@host() { assertThat(Lvl_factory.get@hostIndex()).is@reaterThan( other: -1); }
```

- a. For this test when creating a new ghost in the LevelFactory class there's an attribute named ghostIndex which keeps track of how many ghost exist in the current object. I simply check after creating one that the ghost index is greater than the default of -1.
- b. Also note that by doing this test my npc coverage in class increased from 10% -> 40%.

Coverage for the 3rd test: src/main/java/nl/tudelft/jpacman/board/BoardFactory@createBoard()

Coverage before test was added:

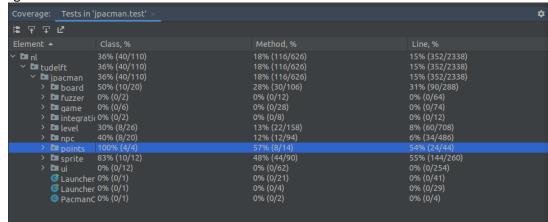


Coverage after test was added:

١.

II.

III.



Test source code BoardTest.java

```
public class BoardTest {
    // Create a sprite_store
    1 usage
    private static final PacManSprites sprite_store = new PacManSprites();
    // Create a board_factory
    1 usage
    private BoardFactory board_factory = new BoardFactory(sprite_store);
    // Create a Square object of 2d
    1 usage
    private Square newSqr[][]={
        {mock(Square.class)},
        {mock(Square.class)}};
    // Create a test board
    2 usages
    private Board test_board = board_factory.createBoard(newSqr);

    // If it was created correctly then the board size would exist 2d with a dimension greater than 0 no usages new *
    @Test
    void create_board_test(){
        assertThat(test_board.getWidth()).isGreaterThan( other: 0);
        assertThat(test_board.getWidth()).isGreaterThan( other: 0);
}
```

- a. For this test when creating a new board, the default width and height are never set so by immediately creating one of and checking its dimension is a good enough check.
- b. My board class package coverage incremented from 30% -> 50%

Task 3:

- I. Are the coverage results from Jacoco similar to the ones you got from IntelliJ in the last task? Why so or why not?
 - a. The coverage results from JaCoCo are different from the ones produced by IntelliJ. My thoughts for this are probably because JaCoCo also takes into account branches so thus needs a different technique to do so compared to IntelliJ.

i jpacman												Session
pacman												
Element \$	Missed Instructions \$	Cov. \$	Missed Branches	Cov. \$	Missed +	Cxty \$	Missed +	Lines	Missed +	Methods \$	Missed÷	Class
🖶 nl.tudelft.jpacman.level		67%		57%	74	156	104	345	21	70	4	
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	
🖶 nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	
<u> </u>		0%	=	0%	12	12	21	21	5	5	1	
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	
nl.tudelft.jpacman.sprite		86%		59%	30	70	11	113	5	38	0	
nl.tudelft.jpacman	_	69%	=	25%	12	30	18	52	6	24	1	
nl.tudelft.jpacman.points	1	60%	1	75%	1	11	5	21	0	9	0	
nl.tudelft.jpacman.game	=	87%	=	60%	10	24	4	45	2	14	0	
nl.tudelft.jpacman.npc	1	100%		n/a	0	4	0	8	0	4	0	
Total	1,213 of 4,697	74%	293 of 637	54%	293	591	229	1,040	51	269	6	

b.

- II. Did you find helpful the source code visualization from JaCoCo on uncovered branches?
 - a. Yes, I did. They added much more detail with including the missed branches section as well as the missed instructions. It's much more detailed than Intellij.
- III. Which visualization did you prefer and why? Intellij's coverage window or Jacoco's report?
 - a. Personally, I prefer IntelliJ's coverage as it's simpler to follow. However, it's pretty neat that JaCoCo includes so much detail in their report like possibly missed branches. I could make a case for both i.e) like using IntelliJ for the draft project and switching over to JaCoCo to catch all the missed branches.

Team Repo: https://github.com/justin-negron/SoftwareProductDesign

My Repo: https://github.com/soulos98/jpacman