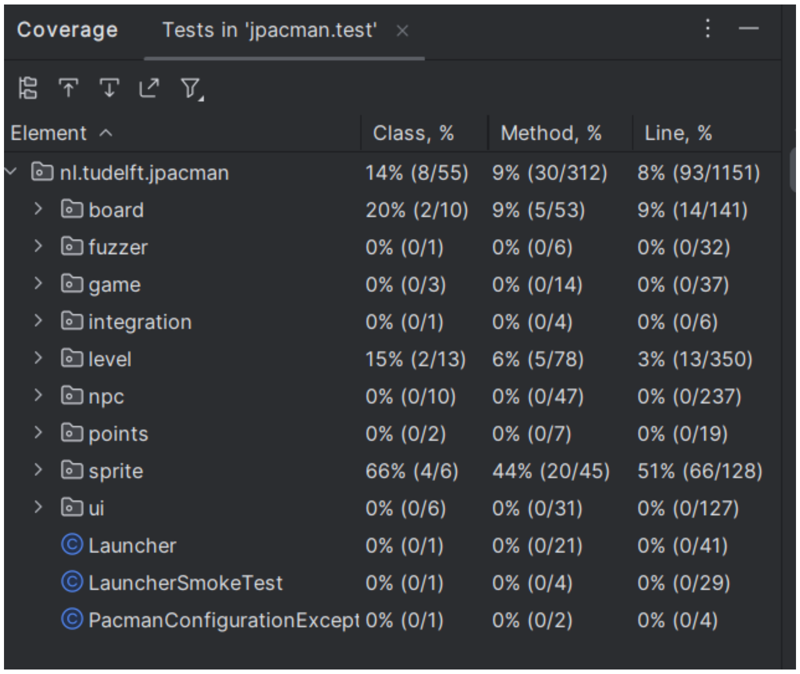
Anthony Valdez
CS472

Link to Forked Repo: https://github.com/SibeW/group8_CS472Project/tree/as2

Task 2.1

Before adding any other tests (Task 2.1), the jpacman test folder had a test coverage of 14% for the class, less than 10% for the methods, and 8% line coverage. The complete detailed results can be seen below.



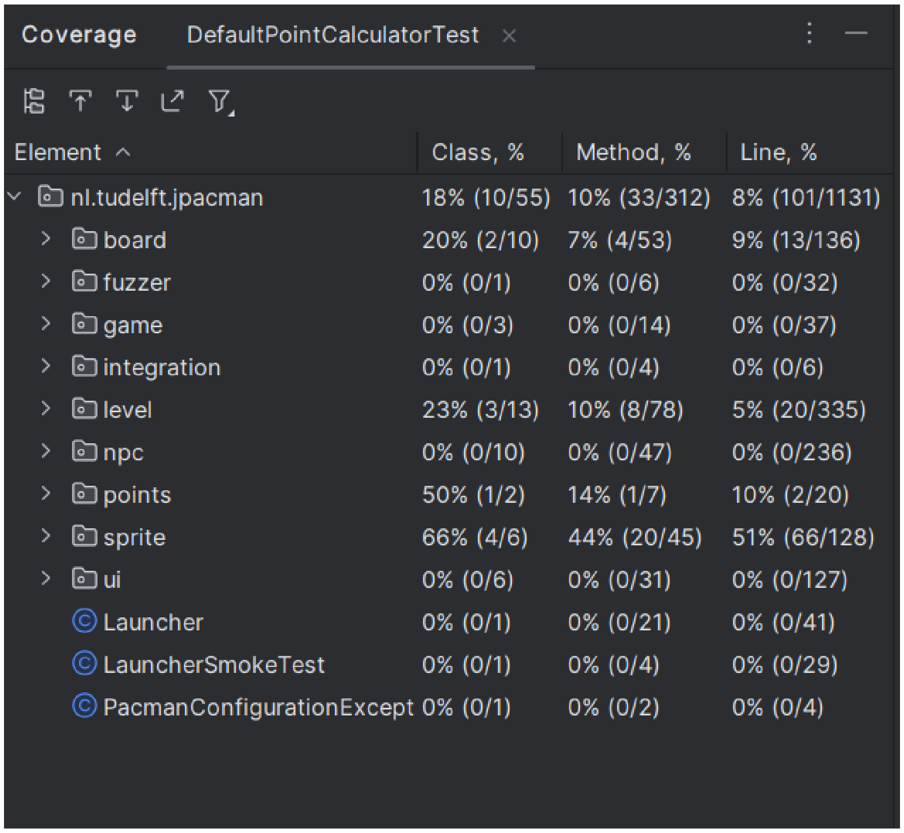
Coverage — Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	14% (8/55)	9% (30/312)	8% (93/1151)
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	15% (2/13)	6% (5/78)	3% (13/350)
npc	0% (0/10)	0% (0/47)	0% (0/237)
points	0% (0/2)	0% (0/7)	0% (0/19)
sprite	66% (4/6)	44% (20/45)	51% (66/128)
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)
PacmanConfigurationExcept	0% (0/1)	0% (0/2)	0% (0/4)

Figure 1: Test Coverage Pre-Task 2.1

Anthony Valdez
CS472

For this test, I initialized a new DefaultPointCalculator object, and created a new player that has zero points, or score. The DefaultPointCalculator method "consumedAPellet," takes in a Player object and adds new points to the Player depending on the pellet consumed. Hence, the test checks to see if the Player's score increases from '0' to '1' as it should.
After implementing a test for the DefaultPointCalculator, we get the following results below. A code snippet is also included for reference.



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	18% (10/55)	10% (33/312)	8% (101/1131)
board	20% (2/10)	7% (4/53)	9% (13/136)
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)	10% (8/78)	5% (20/335)
npc	0% (0/10)	0% (0/47)	0% (0/236)
points	50% (1/2)	14% (1/7)	10% (2/20)
sprite	66% (4/6)	44% (20/45)	51% (66/128)
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)
PacmanConfigurationExcept	0% (0/1)	0% (0/2)	0% (0/4)

Figure 2: After implementing test for src/main/java/nl/tudelft/jpacman/points/DefaultPointCalculator.java: consumedAPellet

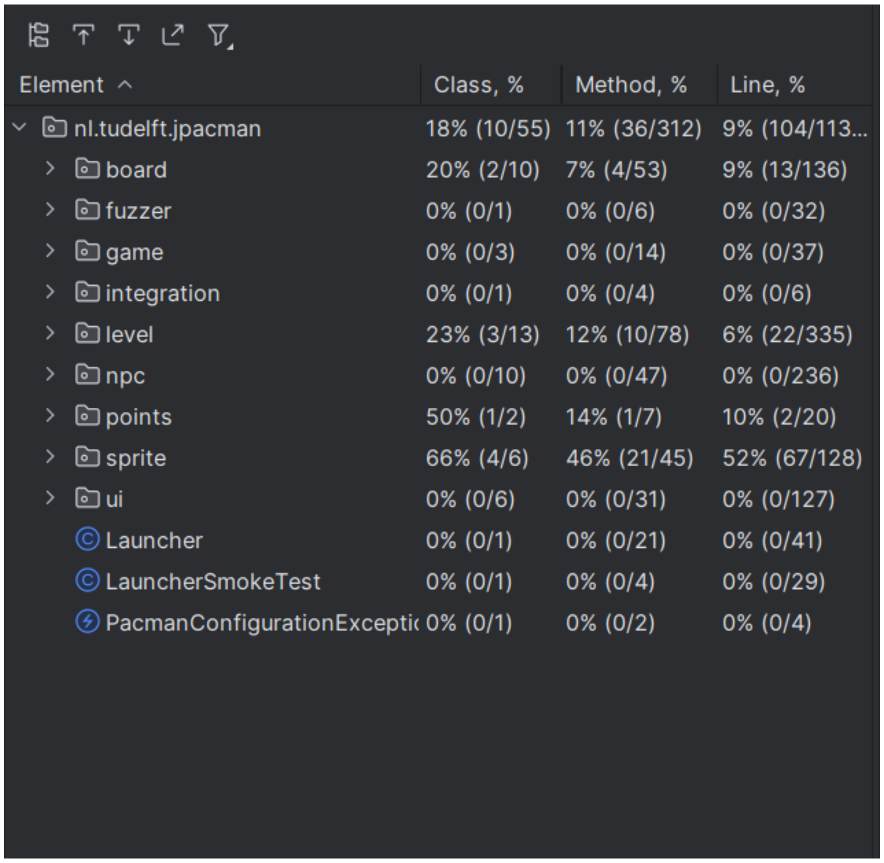
```java
public class DefaultPointCalculatorTest {

    private DefaultPointCalculator DPC = new DefaultPointCalculator();

    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
    private Player ThePlayer = Factory.createPacMan();
    private EmptySprite TheSprite = new EmptySprite();
    @Test
    void testConsumedAPellet(){
        DPC.consumedAPellet(ThePlayer,new Pellet(1,TheSprite));
        assertThat(ThePlayer.getScore()).isEqualTo(1);
    }
}
```

Anthony Valdez
CS472

For this test, I initialized a new Pellet object, set its score/value to 1. This test was mainly to ensure that the Pellet constructor was/is working properly.
After implementing a test for the Pellet class, we get the following results below. A code snippet is also included for reference.



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	18% (10/55)	11% (36/312)	9% (104/113...
board	20% (2/10)	7% (4/53)	9% (13/136)
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)	12% (10/78)	6% (22/335)
npc	0% (0/10)	0% (0/47)	0% (0/236)
points	50% (1/2)	14% (1/7)	10% (2/20)
sprite	66% (4/6)	46% (21/45)	52% (67/128)
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)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)

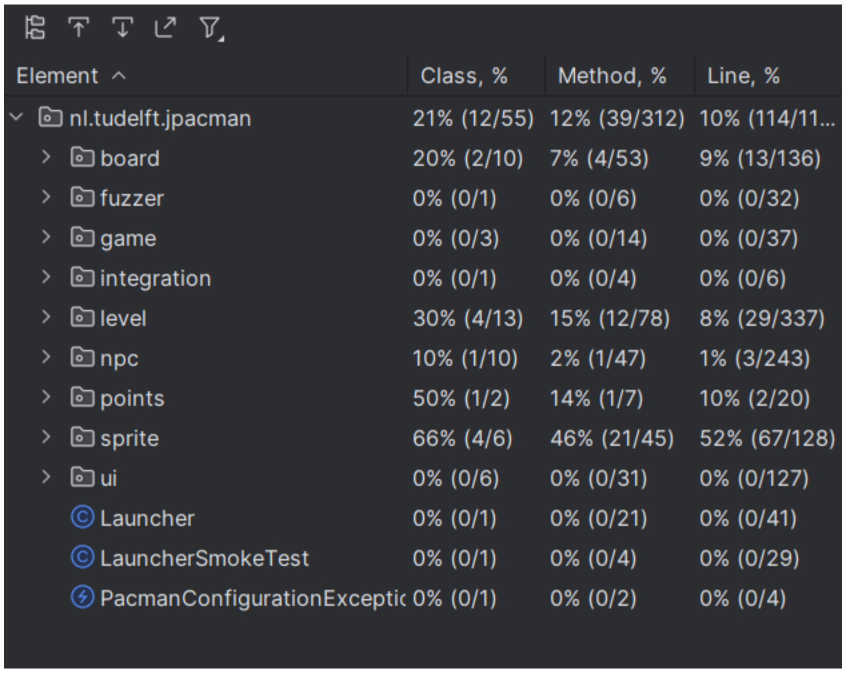
Figure 3: After implementing test for src/main/java/nl/tudelft/jpacman/level/Pellet.java Pellet

```
public class PelletTest {

    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private Pellet ThePellet = new Pellet(1,SPRITE_STORE.getPelletSprite());
    @Test
    void testPelletConstructor(){
        assertThat(ThePellet.getValue()).isEqualTo(1);

assertThat(ThePellet.getSprite()).isEqualTo(SPRITE_STORE.getPelletSprite());
    }
}
```

For this test, I initialized a new LevelFactory object, and tested its method "CreatePellet". The LevelFactory method "CreatePellet," creates a new Pellet object sets its score to '10' and sets the Pellet's Sprite to a PelletSprite. Hence, the test checks to see if the Pellet's score is equal to '10' and also checks if the Sprite was set to PelletSprite as well.
After implementing a test for the LevelFactory, we get the following results below. A code snippet is also included for reference.



Element	Class, %	Method, %	Line, %
nl.tudelft.jpacman	21% (12/55)	12% (39/312)	10% (114/11...
board	20% (2/10)	7% (4/53)	9% (13/136)
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	30% (4/13)	15% (12/78)	8% (29/337)
npc	10% (1/10)	2% (1/47)	1% (3/243)
points	50% (1/2)	14% (1/7)	10% (2/20)
sprite	66% (4/6)	46% (21/45)	52% (67/128)
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)
PacmanConfigurationExcepti	0% (0/1)	0% (0/2)	0% (0/4)

Figure 4 After implementing test for src/main/java/nl/tudelft/jpacman/level/LevelFactory.java createPellet

```
public class LevelFactoryTest {
    private DefaultPointCalculator DPC = new DefaultPointCalculator();

    private static final PacManSprites SPRITE_STORE = new PacManSprites();
    private GhostFactory GstFactory = new GhostFactory(SPRITE_STORE);
    private LevelFactory LvlFactory = new
LevelFactory(SPRITE_STORE,GstFactory,DPC);
    @Test
    void testCreatePellet(){
        Pellet testPellet = LvlFactory.createPellet();
        assertThat(testPellet.getValue()).isEqualTo(10);

assertThat(testPellet.getSprite()).isEqualTo(SPRITE_STORE.getPelletSprite());
    }
}
```

Anthony Valdez
CS472

Conclusion:

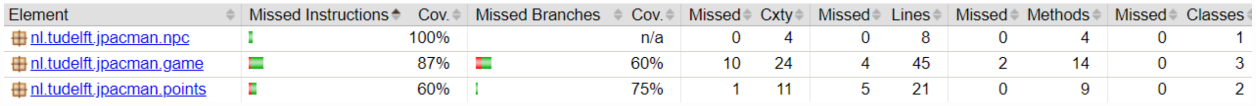
The final result shows that after implementing tests for three separate methods, the test coverage has improved dramatically. As stated before, the jpacman test folder had a test coverage of 14% for the class, less than 10% for the methods, and 8% line coverage. Now, the class coverage is over 20% and all of the coverage columns for jpacman now equate to **at least 10%.** The most dramatic increase was the points package, going from **0% to 50%** in the class column.

Anthony Valdez
CS472

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?

No, the JaCoCo coverage results are different than the ones I got from IntelliJ.



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.npc	🟩	100%		n/a	0	4	0	8	0	4	0	1
nl.tudelft.jpacman.game	🟥🟩	87%	🟥🟩	60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.points	🟥🟩	60%	🟩	75%	1	11	5	21	0	9	0	2

For instance, the coverage percent for .game is higher than the test coverage IntelliJ showed, which is at 0%.

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

Yes, the branches helps to visualize which methods or packages need more attention. Whereas, the IntelliJ coverage results does not help to visualize uncovered branches.

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

I prefer IntelliJ since I do not have to open a new window in a browser in order to see my results. IntelliJ's coverage window is easier to use and also shows which specific project I am performing tests on.