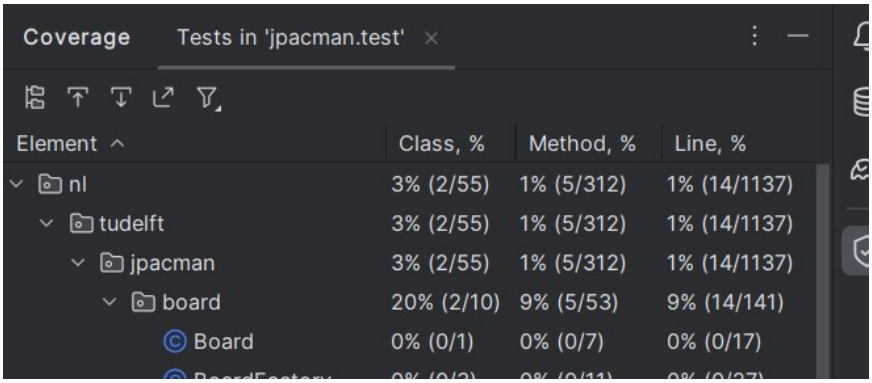
Repository Link: https://github.com/ian5924/CS472-Team

Task 2.1 Report

Before I wrote any methods I started out with 3% coverage.



Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	3% (2/55)	1% (5/312)	1% (14/1137)
tudelft	3% (2/55)	1% (5/312)	1% (14/1137)
jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
board	20% (2/10)	9% (5/53)	9% (14/141)
Board	0% (0/1)	0% (0/7)	0% (0/17)

I chose to do coverage for the playerVersusPellet method from the PlayerCollisions class. This involved creating a player object and a pellet object that could be used as parameters for the method. Then I created a PlayerCollisions object and invoked the playerVersusPellet method using the PlayerCollisions object. Then I asserted that the player was still alive which meant that they had the opportunity to get the point and continue playing. This brought me to 20% coverage.

```java
    //call the factory method to create a Player
    2 usages
    Player ThePlayer = newFactory.createPacMan();

    //make sprite obj
    1 usage
    Sprite newSprite;
    //instantiate pellet obj
    1 usage
    Pellet pelletObj = new Pellet(deafultPelletVal,newSprite);
    //instantiate pointCalculator obj
    no usages
    PointCalculator pointCalculator;

    //instantiate PlayerCollisions obj
    1 usage
    PlayerCollisions newCol = new PlayerCollisions(mock(PointCalculator.class));
    ;
    ian5924 *
    @Test
    void playerVersusPellet() {

        newCol.playerVersusPellet(ThePlayer, pelletObj);

        assertThat(ThePlayer.getKiller()).isEqualTo( expected: null);
    }
}
```

Coverage | Tests in 'jpacman.test'

Element	Class, %	Method, %	Line, %
nl	20% (11/55)	12% (38/312)	9% (113/1161)
tudelft	20% (11/55)	12% (38/312)	9% (113/1161)

I also chose to do coverage for the playerGhosts method from the PlayerCollisions class. This involved creating a player object and a ghostFactory. The ghostFactory allowed me to create a ghost object that was linked to Blinky the ghost. I used the PlayerCollisions object “newCol” which was set to a mock a scenario where the pointCalculator class is used. I then created a test where I invoked the playerVersusGhost method. Lastly, I asserted that the player's isAlive() status was set to false meaning that they did die. This raised coverage to 27%.

```java
    Player ThePlayer = newFactory.createPacMan();

    //new factory for ghost
    1 usage
    GhostFactory newGhostFact = new GhostFactory(ObjForPlayFac);

    //instantiate ghost obj
    1 usage
    Ghost ghostObj = newGhostFact.createBlinky();

    //instantiate PlayerCollisions obj
    1 usage
    PlayerCollisions newCol = new PlayerCollisions(mock(PointCalculator.class));

    ian5924 *
    @Test
    void playerVersusGhost() {

        newCol.playerVersusGhost(ThePlayer, ghostObj);
        //make sure player is not alive
        assertThat(ThePlayer.isAlive()).isEqualTo( expected: false);
    }
}
```

Coverage | Tests in 'jpacman.test'

Element	Class,...	Method, %	Line, %
nl	27% (15/55)	15% (49/3...	12% (148/1...
tudelft	27% (15/55)	15% (49/3...	12% (148/1...

Lastly, I created a test for the addPoints method. After creating a player object I checked the player's current score using getScore() and then invoked the addPoints method with the player object to add 10 points to the player's score. Next, I asserted that the score before adding the 10 points was less than the current score.

```java
    3 usages
    Player ThePlayer = newFactory.createPacMan();

    ian5924
    @Test
    void addPoints() {
        //points to be given
        int pointsToBeAdded = 10;
        //get current score
        int preScore = ThePlayer.getScore();
        //add points to player's scores
        ThePlayer.addPoints(pointsToBeAdded);

        assertThat( actual: preScore <ThePlayer.getScore() );

    }
}
```

Task 3 Report

The results are technically the same. However, they are presented in different ways. JaCoCo shows the element, missed instructions, missed branch coverage, and the associated data. On the other hand, IntelliJ will present coverage through the methods, classes, and lines. I did find the visualization helpful. It allowed me to better understand how much of an effect my unit testing has over specific parts of the file. The JaCoCo visualization detailed what exactly was uncovered. It matched what I thought would be uncovered due to my focus on three methods.
I prefer the JoCoCo report due to how it explains what was covered and uncovered. Additionally, I think it creates a more effective breakdown of the folders and files.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		68%		58%	72	155	102	344	20	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		77%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.board		86%		59%	43	93	2	110	0	40	0	7
nl.tudelft.jpacman.sprite		88%		62%	29	70	10	113	5	38	0	5
nl.tudelft.jpacman		69%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		60%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,201 of 4,694	74%	289 of 637	54%	289	590	226	1,039	50	268	6	47