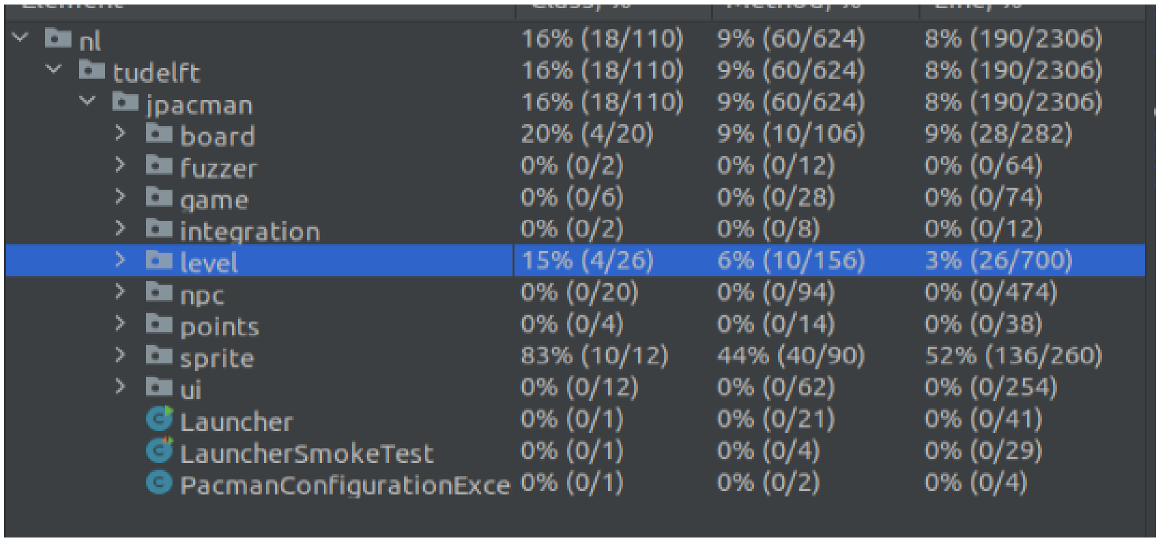
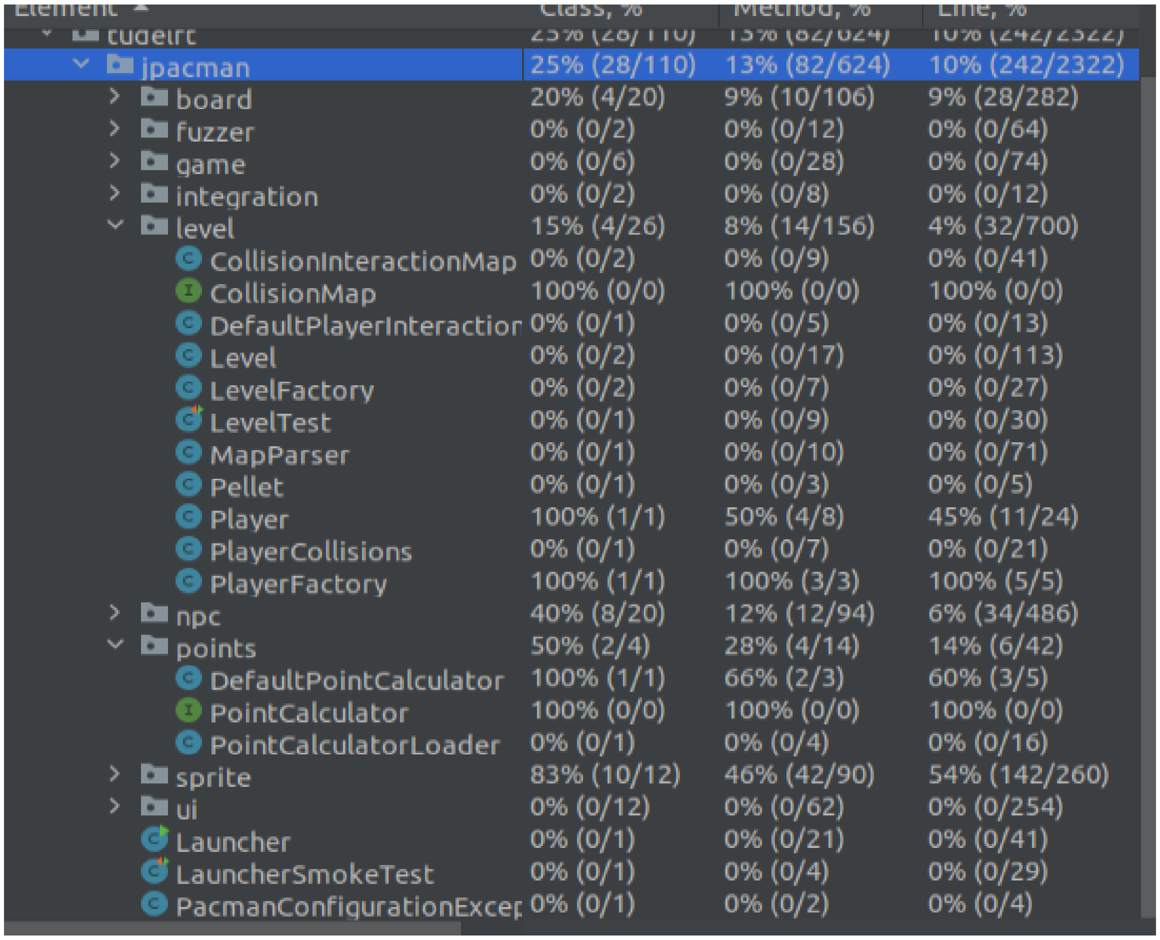
Test Coverage (before and after) - Task 2.1

The following screenshot is the coverage before I added my three method unit tests:



Element	Class, %	Method, %	Line, %
nl	16% (18/110)	9% (60/624)	8% (190/2306)
tudelft	16% (18/110)	9% (60/624)	8% (190/2306)
jpacman	16% (18/110)	9% (60/624)	8% (190/2306)
board	20% (4/20)	9% (10/106)	9% (28/282)
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	15% (4/26)	6% (10/156)	3% (26/700)
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)
PacmanConfigurationExce	0% (0/1)	0% (0/2)	0% (0/4)

The next screenshot is the coverage after I added my three method unit tests:



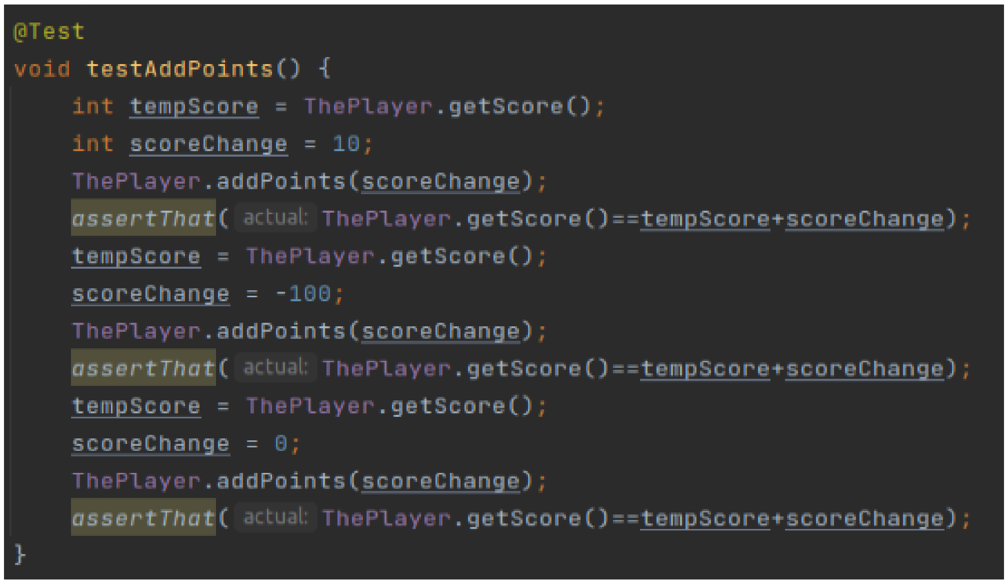
Element	Class, %	Method, %	Line, %
tudelft	25% (28/110)	13% (82/624)	10% (242/2322)
jpacman	25% (28/110)	13% (82/624)	10% (242/2322)
board	20% (4/20)	9% (10/106)	9% (28/282)
fuzzer	0% (0/2)	0% (0/12)	0% (0/64)
game	0% (0/6)	0% (0/28)	0% (0/74)
integration	0% (0/2)	0% (0/8)	0% (0/12)
level	15% (4/26)	8% (14/156)	4% (32/700)
CollisionInteractionMap	0% (0/2)	0% (0/9)	0% (0/41)
CollisionMap	100% (0/0)	100% (0/0)	100% (0/0)
DefaultPlayerInteractior	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)
npc	40% (8/20)	12% (12/94)	6% (34/486)
points	50% (2/4)	28% (4/14)	14% (6/42)
DefaultPointCalculator	100% (1/1)	66% (2/3)	60% (3/5)
PointCalculator	100% (0/0)	100% (0/0)	100% (0/0)
PointCalculatorLoader	0% (0/1)	0% (0/4)	0% (0/16)
sprite	83% (10/12)	46% (42/90)	54% (142/260)
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)
PacmanConfigurationExcep	0% (0/1)	0% (0/2)	0% (0/4)

Unit Tests - Task 2.1

My three unit tests were on the methods:

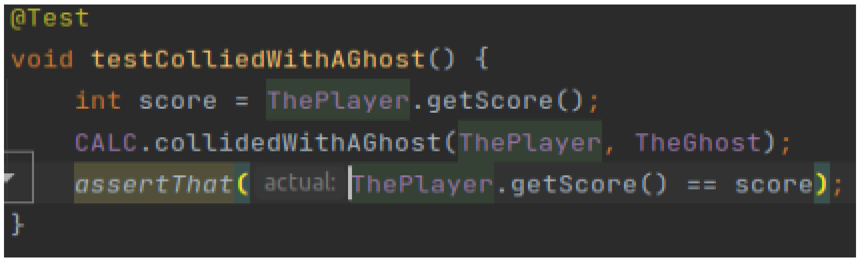
- level/player.addPoints
- points/DefaultPointCalculator.collidedWithAGhost
- points/DefaultPointCalculator.pacmanMoved

level/player.addPoints



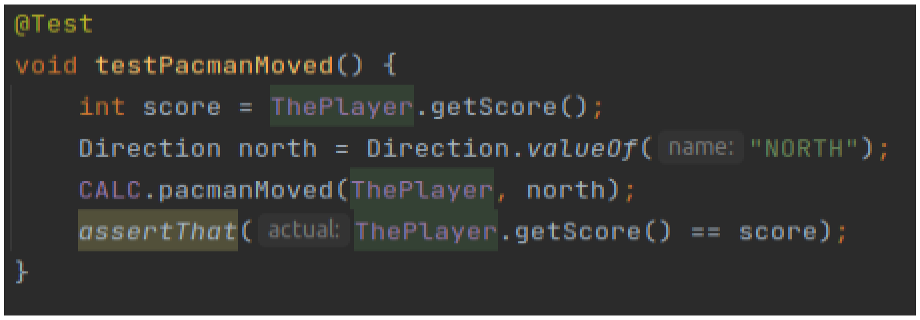
```
@Test
void testAddPoints() {
    int tempScore = ThePlayer.getScore();
    int scoreChange = 10;
    ThePlayer.addPoints(scoreChange);
    assertThat( actual: ThePlayer.getScore()==tempScore+scoreChange);
    tempScore = ThePlayer.getScore();
    scoreChange = -100;
    ThePlayer.addPoints(scoreChange);
    assertThat( actual: ThePlayer.getScore()==tempScore+scoreChange);
    tempScore = ThePlayer.getScore();
    scoreChange = 0;
    ThePlayer.addPoints(scoreChange);
    assertThat( actual: ThePlayer.getScore()==tempScore+scoreChange);
}
```

points/DefaultPointCalculator.collidedWithAGhost



```
@Test
void testColliedWithAGhost() {
    int score = ThePlayer.getScore();
    CALC.collidedWithAGhost(ThePlayer, TheGhost);
    assertThat( actual: ThePlayer.getScore() == score);
}
```

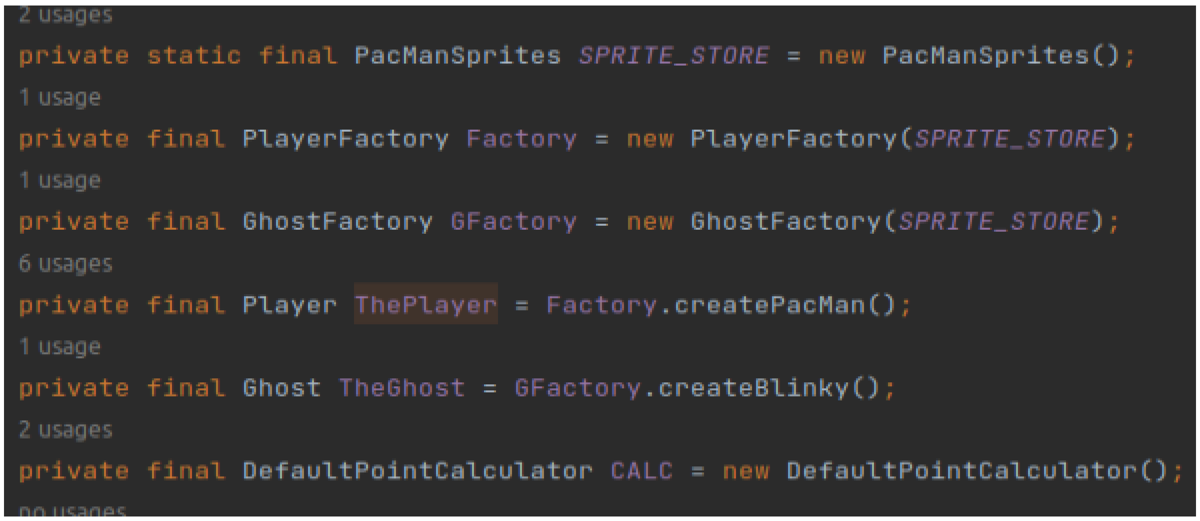
points/DefaultPointCalculator.pacmanMoved



```
@Test
void testPacmanMoved() {
    int score = ThePlayer.getScore();
    Direction north = Direction.valueOf( name: "NORTH");
    CALC.pacmanMoved(ThePlayer, north);
    assertThat( actual: ThePlayer.getScore() == score);
}
```

Unit Test Report - Task 2.1

Each unit test required that I create objects of classes like PacManSprites and GhostFactory in order to create Ghost, Player, and DefaultPointCalculator objects. The use of the ghost and player objects can be seen in the objects “The Player”, “TheGhost”, and CALC above. The creation of the prerequisite objects can be seen in the screenshots below.



```
2 usages
private static final PacManSprites SPRITE_STORE = new PacManSprites();
1 usage
private final PlayerFactory Factory = new PlayerFactory(SPRITE_STORE);
1 usage
private final GhostFactory GFactory = new GhostFactory(SPRITE_STORE);
6 usages
private final Player ThePlayer = Factory.createPacMan();
1 usage
private final Ghost TheGhost = GFactory.createBlinky();
2 usages
private final DefaultPointCalculator CALC = new DefaultPointCalculator();
no usages
```

These unit tests resulted in an increase in the Method, Line, and Class coverage (according to IntelliJ) for the entire project. The final increase is as follows:

+2% Method coverage in the level package
+1% Line coverage in the level package
+50% Class coverage in the points package
+28% Method coverage in the points package
+14% Line coverage in the points package

JaCoCo vs. IntelliJ - Task 3

Question 1:

The coverage results are similar between JaCoCo and IntelliJ, but JaCoCo formats the results differently. JaCoCo shows branch and instruction coverage missing, while IntelliJ shows class, method, and line coverage. These formatting differences result in JaCoCo stating that the project is missing 74% (1213/4694) instruction coverage, and IntelliJ stating that the project has 10% (242/2322) line coverage.

Question 2:

I found the source code visualization from JaCoCo on branches useful. I didn't create any unit tests on source code that had branches but I would have used JaCoCo if I had to double check my coverage.

Question 3:

I prefer the IntelliJ coverage report because it can be seen inside the same UI as the coding environment. I also think that it is easier to visualize the amount of coverage in terms of total coverage rather than coverage missing. The only thing I find more useful in JaCoCo is that JaCoCo highlights the tested source code.

https://github.com/GusEsplin/472-2023-G3