Lab 2 Testing Report

The results produced from JaCoCo are not similar to the report obtained from IntelliJ. Without any additional test files built/run, the coverage on JaCoCo for the *level* branch showed 67%. After running all of my test files, the JaCoCo coverage report only increased by 7%. Whereas in IntelliJ, it came from 3% to 36%. Both results are not similar due to large differences.

The visualization from JaCoCo on uncovered branches is helpful. It can show what exactly is uncovered and what percentage is already covered. IntelliJ is also useful for seeing what is uncovered. I do prefer JaCoCo's visualization because it has a deeper breakdown when it comes to how many lines, instructions, branches, etc., are missing. One thing I do like about the visualization of JaCoCo is how it highlights the instructions that are not covered. IntelliJ just puts a small indicator on the side of the code which can be easily missed.

Link to my forked repo: https://github.com/Dashtine/472-2023-G3-1

The next pages contain snippets of the coverage report of each test file added and also the code implemented to cause the coverage increase.

Note: The following report starts without any additional test files I have created. As this document continues, each coverage report shown will include test files that have been previously added.

Within the PlayerTest.java code, I created objects in order to declare a *Player* object. That was used to call *isAlive()* within the Player.java file. Before running the test, the coverage report showed 3%. After the file implementation, it increased to 16%.

3% classes, 1% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	3% (4/110)	1% (10/616)	1% (28/2266)
scoreplugins			
sprite			

Figure 1a: Coverage report before adding PlayerTest.java

```java
19  public class PlayerTest {
20
21      PacManSprites sprite_obj = new PacManSprites();
22      PlayerFactory pfact_obj = new PlayerFactory(sprite_obj);
23      Player thisPlayer = pfact_obj.createPacMan();
24
25      //isAlive() test
26      @Test
27      void isAlive_Test(){
28          if(thisPlayer.isAlive()){
29              System.out.println("thisPlayer is alive!");
30          }
31          else{
32              System.out.println("thisPlayer is dead!");
33          }
34      }
35  }
```
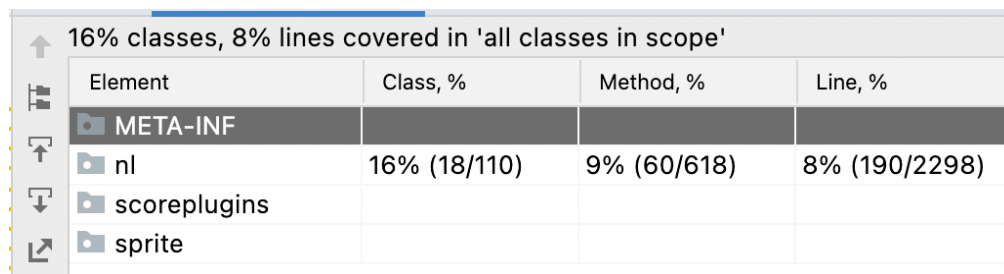
Figure 1b: Code snippet of PlayerTest.java

16% classes, 8% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	16% (18/110)	9% (60/618)	8% (190/2298)
scoreplugins			
sprite			

Figure 1c: Coverage report after adding PlayerTest.java

For my next test file, I invoked the DefaultPlayerInteractionMap constructor method. This increased the coverage report from 16% to 23%.
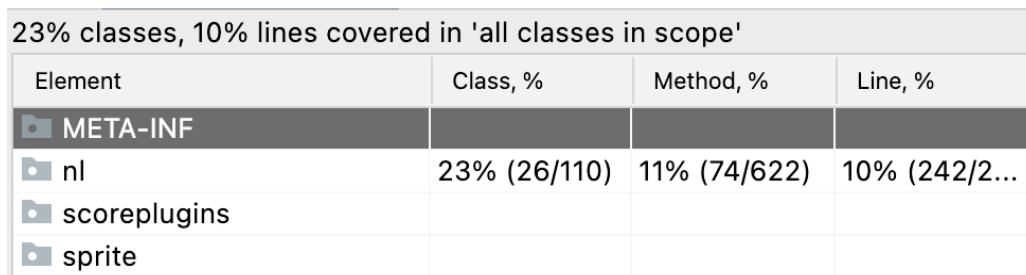
16% classes, 8% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	16% (18/110)	9% (60/618)	8% (190/2298)
scoreplugins			
sprite			

Figure 2a: Coverage report before adding DefaultPlayerInteractionMapTest.java

```java
public class DefaultPlayerInteractionMapTest {
    DefaultPointCalculator defpointcalc = new DefaultPointCalculator();

    @Test
    void DefaultPlayerInteractionMap_Test(){
        DefaultPlayerInteractionMap defaultPlayerInteractionMap = new DefaultPlayerInteractionMap(defpointcalc);

    }
}
```

Figure 2b: Code snipper of DefaultPlayerInteractionMapTest.java

23% classes, 10% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	23% (26/110)	11% (74/622)	10% (242/2...
scoreplugins			
sprite			

Figure 2c: Coverage report after adding DefaultPlayerInteractionMapTest.java

The next file I have added to test was a constructor method within MapParsing. For this, I had to create more objects from other classes in order to finally get the necessary parameters. This increased the coverage report from 23% to 30%.

23% classes, 10% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	23% (26/110)	11% (74/622)	10% (242/2...
scoreplugins			
sprite			

Figure 3a: Coverage report before adding MapParserTest.java

```java
public class MapParserTest {
    PacManSprites sprite_obj = new PacManSprites();
    PlayerFactory pfact_obj = new PlayerFactory(sprite_obj);
    Player thisPlayer = pfact_obj.createPacMan();

    //Variables for MapParser test
    GhostFactory ghost_fact = new GhostFactory(sprite_obj);
    DefaultPointCalculator df_point = new DefaultPointCalculator();
    LevelFactory level_factory = new LevelFactory(sprite_obj, ghost_fact, df_point );
    BoardFactory board_factory = new BoardFactory(sprite_obj);

    @Test
    void MapParsing_Test(){
        MapParser mapParser = new MapParser(level_factory, board_factory);
    }
}
```

Figure 3b: Code snippet of MapParserTest.java

30% classes, 11% lines covered in 'all classes in scope'			
Element	Class, %	Method, %	Line, %
META-INF			
nl	30% (34/110)	13% (84/622)	11% (276/2356)
scoreplugins			
sprite			

Figure 3c: Coverage report after adding MapParserTest.java

The last file I implemented to test out was LevelFactoryTest.java. This created a new LevelFactory object by invoking its constructor method. The coverage report came from 30% to 36%.

Element	Class, %	Method, %	Line, %
30% classes, 11% lines covered in 'all classes in scope'			
META-INF			
nl	30% (34/110)	13% (84/622)	11% (276/2356)
scoreplugins			
sprite			

Figure 4a: Coverage report before adding LevelFactory.java

```java
public class LevelFactoryTest {

    PacManSprites sprite_obj = new PacManSprites();
    GhostFactory ghost_fact = new GhostFactory(sprite_obj);
    DefaultPointCalculator df_point = new DefaultPointCalculator();

    @Test
    void LevelFactory_Test(){
        LevelFactory levelFactory = new LevelFactory(sprite_obj, ghost_fact, df_point);
        Ghost new_ghost = levelFactory.createGhost();
    }
}
```

Figure 4b: Code snippet of LevelFactory.java

Element	Class, %	Method, %	Line, %
36% classes, 13% lines covered in 'all classes in scope'			
META-INF			
nl	36% (40/110)	15% (98/622)	13% (318/2356)
scoreplugins			
sprite			

Figure 4c: Coverage report after adding LevelFactory.java