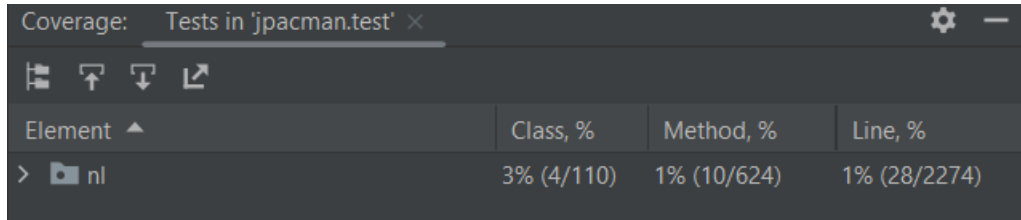


Fork Link: <https://github.com/TheBenKnee/jpacman>

Task 1 – 2.1:

Off of the bat the code coverage for 'test' directory tests for the jpacman repo looked like:



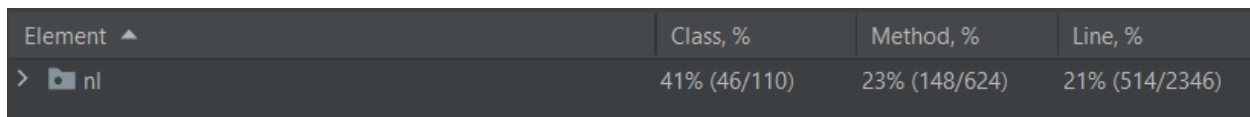
Coverage: Tests in 'jpacman.test' x

| Element ^ | Class, % | Method, % | Line, % |
|-----------|------------|-------------|--------------|
| > nl | 3% (4/110) | 1% (10/624) | 1% (28/2274) |

After adding the "Player" "isAlive()" test as well as the following:

| | | |
|----------------|------------------|---------------------|
| Benjamin Weber | MapParser | parseMap() |
| Benjamin Weber | EmptySprite | split() |
| Benjamin Weber | PlayerCollisions | playerVersusGhost() |

Test coverage was increased to:

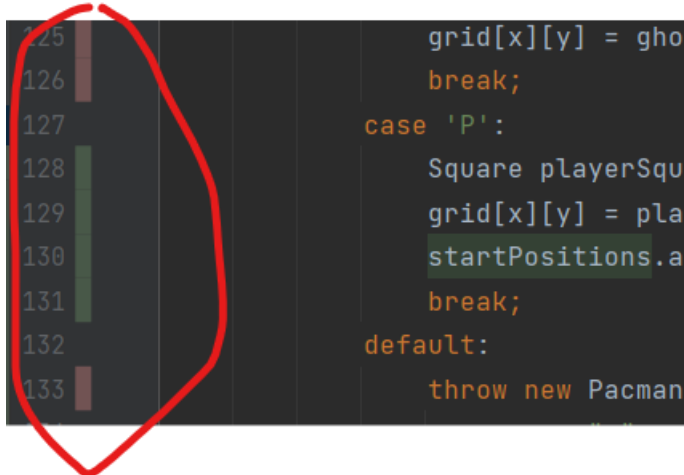


| Element ^ | Class, % | Method, % | Line, % |
|-----------|--------------|---------------|----------------|
| > nl | 41% (46/110) | 23% (148/624) | 21% (514/2346) |

Class coverage increased 38% and line coverage increased 20%. (Code snippets at end of doc).

Task 3

- The coverage results from JaCoCo were similar to that of the IntelliJ window with a bit of a decrease in code coverage. I would guess this would be due to the IntelliJ window not considering branch coverage.
- While the branch coverage is extremely useful as it reveals a proper test vs. an insufficient test, I also found that looking at the source code in IntelliJ after running the code coverage showed the branch/line by line coverage (the JaCoCo does show this information numerically/totaled up however, which is nice):



- I prefer the IntelliJ coverage window because in both situations where you want an in-depth summary and where you want a quick summary, IntelliJ provides a prettier and clearer interface. If you want a summary on a specific class or method, the drop-down format of the IntelliJ window mirrors that of the IntelliJ file manager in general making it much quicker to find specific files or classes in my opinion. Additionally, the visualization of branch coverage (as shown above) is extremely intuitive and helpful to show exactly where you need another test case or your test case is missing.

Code Snippets

PlayerAliveTest:

```
public class PlayerAliveTest {

    /**
     * The player under test.
     */
    private Player player;

    /**
     * Creates a player character to test on.
     */
    @BeforeEach
    void setUp()
    {
        PlayerFactory playerFactory = new PlayerFactory(new PacManSprites());
        player = playerFactory.createPacMan();
    }

    /**
     * Asserts that a unit has no square to start with.
     */
    @Test
    void isAlive() {
        // Remove the following placeholder:
        assertThat(player.isAlive());
    }
}
```

MapParserTest:

```
public class MapParserTest {

    /**
     * The MapParser under test.
     */
    private MapParser mapParser;

    /**
     * Creates a MapParser to test on.
     */
    @BeforeEach
    void setUp()
    {
        PacManSprites pmSprites = new PacManSprites();
        mapParser = new MapParser(
            new LevelFactory(pmSprites, new GhostFactory(pmSprites), new
DefaultPointCalculator()),
            new BoardFactory(pmSprites)
        );
    }

    /**
     * Asserts that an parseMap properly creates a level (determined if the
player position is registered).
     */
    @Test
    void parseMap()
    {
        Level newLevel = mapParser.parseMap(Lists.newArrayList("#P#", " # ",
"###"));
        assertThat(newLevel.isAnyPlayerAlive());
    }
}
```

EmptySpriteTest:

```
public class EmptySpriteTest {

    /**
     * The emptySprite under test.
     */
    private EmptySprite emptySprite;

    /**
     * Creates an emptySprite to test on.
     */
    @BeforeEach
    void setUp()
    {
        emptySprite = new EmptySprite();
    }

    /**
     * Asserts that an emptySprite properly returns its width.
     */
}
```

```

@Test
void split ()
{
    assertThat(emptySprite.split(0, 0, 1, 1) instanceof EmptySprite);
}
}

```

PlayerCollisionTest:

```

public class PlayerCollisionsTest {

    /**
     * The playerCollision under test.
     */
    private PlayerCollisions playerCollisions;

    /**
     * Creates an playerCollision to test on.
     */
    @BeforeEach
    void setUp ()
    {
        playerCollisions = new PlayerCollisions(new
DefaultPointCalculator());
    }

    /**
     * Asserts that an playerCollision properly kills a player when
playerVersusGhost triggered.
     */
    @Test
    void playerVersusGhost ()
    {
        Player player = new PlayerFactory(new
PacManSprites()).createPacMan();
        playerCollisions.playerVersusGhost (player, new GhostFactory(new
PacManSprites()).createBlinky());
        assertThat(!player.isAlive());
    }
}

```