Report

3.1 Smoke Testing

3.1.1

The overall line coverage was found to be 82%. The `CollisionInteractionMap` and the `DefaultPlayerInteractionMap` both have 0% coverage. This is because `CollisionInteractionMap` is only used by `DefaultPlayerInteractionMap`, which, in turn, is never referred to by any classes that are instantiated during the smoke test.

We added the following error to the game: we replaced the direction in the `Game.move()` method to `Direction.NORTH`. The smoke test captured this with a failing test.

3.1.2

The `move()` method is covered. After commenting out the last line of the `move()` method, the smoke test generated the error:

```
org.junit.ComparisonFailure: expected:<[1]0> but was:<[]0> Expected :10 Actual :0
```

The trace reveals:

```
at nl.tudelft.jpacman.LauncherSmokeTest.smokeTest(LauncherSmokeTest.java:69)
```

which indicates that the test failed on line 69, which contains the code:

```
assertThat(player.getScore()).isEqualTo(10);
```

We can conclude that something must have gone wrong between the previous assertion (on line 65) and this one. The only line in between these two assertions in the smoke test is line 68:

```
game.move(player, Direction.EAST);
```

So, the smoke test tells us that we should investigate this method call further.

3.1.3

When we changed the `board.Direction.getDeltaX()` method, the smoke test failed in exactly the same way as in the previous exercise. So, while the smoke test *is* able to tell us that we should investigate the call to `game.move(player, Direction.EAST);`, it does not tell us whether the problem resides with the method itself or with the objects that are being passed *to* the method.

3.2 Feature-Based Testing

3.2.6

For the second `Given` statement in S2.4 it is hard to realise proper timing of movements, in order for the player to be next to a ghost. For the `When` statement in S2.5 it is hard to realise exact timing of movements, so that pacman has eaten all but one pellets, without colliding with a ghost.

3.2.8

All of the scenarios in User Story 3 pertain to ghosts, which are automatically moved around. Because of this, it is not possible to test the moves using `game.move()` as before. Moreover, Ghost itself is an abstract class, which means we would have to test the individual ghosts, thereby creating additional test cases. Furthermore, the Ghost classes do not have built-in methods to return the current square in the way that Player does. The situation is further complicated by the fact that the different Ghosts not only move differently from each other, but also differently depending on certain scenarios in the game.

3.3 Boundary Testing

3.3.9

See 3.3.9_domain_matrix.png

3.4 Understanding your tests

3.4.11

If the test methods in a class start with duplicate initialization code this can be moved into a common initialization method with the @Before annotation, because this method will be executed before every single execution of a test method. Also, when a method needs to be tested using multiple values as input (as in boundary testing for example),the test can be parameterized, which prevents writing same pieces of code around the input values.

3.4.12

Using clean instances of the class under test, is necessary for independence among tests. The impact that one test has on another should be minimized to be sure that when a test fails, it only fails because of that test.

3.4.13

The difference between `assertTrue(a == 1)` and `assertEquals(a, 1)` is that the `assertEquals`

gives a comparison of the expected value with the actual value, whereas `assertTrue` does not. So `assertEquals` is more useful, as it provides information that can be used debug a failing test.

3.4.14

One could make the argument that it is not necessary to test the private methods of `MapParser` because all of the end-to-end tests rely on a `Launcher` which makes use of `MapParser`. So, we would expect a faulty `MapParser` to yield failing end-to-end tests. However, it is also the case that a faulty `MapParser` may make it difficult to debug the failing end-to-end tests, without having isolated tests of the private methods of `MapParser` itself. Furthermore, a passing test does necessarily not guarantee anything if the test itself has faults. So, a passing end-to-end test does not neceaarily guarantee that `MapParser` would not fail an isolated test. In conclusion, it would probably be a good idea to test the private methods in isolation.

3.5

3.5.15

There is one warning that remains in IntelliJ. IntelliJ complains that `public class WithinBordersTest` can be private. We left this public because the comment in `ParameterizedAssignment` specifically states that it needs to be public.

In terms of the additional adequacy achieved thanks to our classes, we measured the new overall coverage to be 89%, with a line coverage of 84%. So, our efforts have contributed 2 percentage points to the overall line coverage, as compared to what we measured in question 3.1.1.

The continuous integration server confirmed that our builds worked properly in most cases. We generally tried to avoid successively failing builds on DevHub, as evidenced by the many green commits. We used Git very extensively. In general, we tried to make new branches for different exercises. This allowed us to divide the work effectively and gives a very clear record of what was done.