Test Report

In this document we will explain how we tested our game. We will start with explaining how often we tested our game. Hereafter, we will explain what kinds of testing we have done. Lastly, we will present the results of the testing.

Frequency of testing

In this section we will discuss how frequently we tested our game. Because of the major rewrite we have done, alot of our tests needed refactoring as well. Sometimes we had to share code between one another and in cases like this, we just disabled our tests (by commenting them completely). However, we made sure to test our new code as early as possible. Luckily, most of our classes remained the same and as such we could reuse all our tests.

One big issue, however, was the fact that some classes now were Actors by themselves (an Actor is a LibGDX class). This means that they require a texture upon instantiation, something that is not available during headless testing on DevHub. It took us a few days to figure out how to solve this, which is why these classes took longer than the rest to be tested.

A similar case happened with the AssetHandler, which now uses a Skin to easily manage assets. The JSON file it requires was not readable in headless mode (for reasons still unknown), so this too took quite a while to figure out.

As the deadline came closer, we started feeling more and more pressure. This resulted in Jente rushing with Jochem to implement all the functionality and Piet chasing after them, testing all the code they produced.

Lastly, it remains us to say that most of our code was indeed tested, albeit visually, before pushing - so we were somewhat sure it actually worked, before sharing.

Kinds of testing

As touched briefly in the previous section, we made use of two types of testing this sprint: visual testing and unit testing. With visual testing we mean that we launched the game and observed what happened. We used this testing technique mostly to ensure our code somewhat worked when the pressure to implement things was high.

Unit tests were made when the pressure was off. Again, it did not catch that many bugs but it did make us feel confident that our code works.

Results

We have used EclEmma to measure our coverage by three different metrics:

1. Our line coverage is 72.1%.
2. Our branch coverage is 69.4%.
3. Our instruction coverage is 72.1%.

While these results are less than what we achieved with the first deliverable, we feel that we have tested our code extensively. The parts of the code we have not covered is almost only code related to drawing or code that requires OpenGl, which both are impossible to test on DevHub. These pieces are visually tested, and we are pretty sure they work.

To conclude this document it only remains us to say that, again, we are confident that our game is properly and sufficiently tested.