Test Report

In this report we discuss how we tested our game. We will start with explaining how often we tested our game. Hereafter, we will explain what kind of testing we have done. Lastly, we will show the results of the tests.

Frequency of testing

In this section we will discuss how frequently we tested our game. In the first sprint we started developing the game rapidly, even though we had made a careful planning. As a result, we forgot to write any test cases: we only briefly tested everything by running the game.

After this first sprint, it was clear that we had to test while developing. We had to abide more by the SCRUM methods, as well. We made sure to push a feature only when the relevant test was done and we set out to add the tests which we skipped in sprint one. We feel like our testing behaviour was better during this second sprint, albeit still not perfect.

Kinds of testing

We used both functional and static to test the various aspects of the game. Functional testing means that we tested the game by launching it and observing what happens. We used this technique extensively during the first sprint and it made us catch many bugs, especially in the moving of the tiles.

As discussed, we started writing all the missing tests in the second sprint. This resulted in us using static testing more. We caught less bugs, but we got confident in the under the hood code of the game as we weren't exactly sure it worked.

Results

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

1. Our line coverage is 75.9%.
2. Our branch coverage is 82.8%.
3. Our instruction coverage is 78.8%.

While this is a decent result, some classes are not at all covered by the JUnit tests: GameRenderer, GameScreen, ButtonHandler and TwentyFourtyGame. The GameRenderer and GameScreen classes are used to draw and control the graphical user interface and as such these can not be tested.

ButtonHandler can not be tested because everything inside it is static and there are no setters to inject mock objects to verify behaviour. Lastly, the TwentyFourtGame class is not tested because it is simple enough to test virtually: as long as a screen pops up correctly, it works.

One major problem we had during testing is that Devhub cannot initialize any graphical user interface on its server, because our library uses OpenGL. Any test we tried to write for classes that required on a display or an OpenGL call would cause a build failure. We worked around this by using a headless backend for our library, created specifically for this issue.

As discusses, our coverage results are all above 75%. To conclude this document, it only remains us to say that we are confident that our game is tested more than sufficient.