Reflection document

Refl. Team

Working in a team has worked really well, for the most part. Git and scrum in combination is without a doubt a very effective work environment.
There were, however, times when you wanted to ask someone about something in their code, but they were not present or online. This resulted in certain problems being put on hiatus for extended amounts of time.
We have wasted more time than necessary arguing about design choices, but someone was always there to eventually break up the argument. You always keep the fact that you are working in a team in the back of your head, so you make sure that your classes are as usable as possible for you teammates to work with.

Refl. Doc

For our documents we have almost entirely been using Google Docs (now also known as Drive). It is a very convenient tool for this kind of collaboration, since several people can write on the same documents at the same time. This way all four of us got involved in the documentation. We made sure to check our old documentation at regular intervals by letting them remain in the product backlog (with their occasional visits to the sprint backlog).
For creating the design diagram we used a web service called Cacoo. In Cacoo you can, much like in Google Docs, work several at the same time.
We have also heavily used GitHub's features, such as opening issues and reporting bugs.

Refl. SE

As previously mentioned, we believe that git and scrum are a very powerful combination and it worked really well. Specifying 10 traditional requirements was completely redundant and unnecessary after already having written user stories, which is a method we much preferred.

Refl. Coverage

We originally intended to use coverage tools like ECLEmma, but it was a real bother to get working.
We eventually decided to settle for common sense. If we had more time, we would gladly have spent some of it on getting coverage tools with Ant working, but for now we've had so much to do anyway and we still don't really understand how to get Ant working.

Refl. Problems, hurdles and things we would have done differently

We chose an Android game engine called AndEngine to use for our project. We *deeply* regret choosing this engine for a number of different reasons. The engine had essentially no support for the strict MVC model which we decided to use. This resulted in some ugly and complicated solutions using listeners to make the view oblivious to the model.
It also forced us to manually build a "sprite atlas", resulting in us making changes to already existing sprites every time a new sprite was added.
The engine just seems full of terrible design decisions in general.
Things might have been better if we adapted a different package structure (not MVC), but in all honesty, we want to put the blame on the engine.

If we would start the project over again, we would probably try to search for another engine for android.

Even though we had a lot of problems with the engine, if no other choice was available, we would probably still have to use this engine, mainly because none of us have sufficient experience with openGL. Thus the engine more serves as an interface to use graphics, rather than a full fledged game engine.

One of the biggest problems with this engine was the fact that it contained no Javadoc whatsoever. It seemed that all the programmers using this engine relied on the forum dedicated to it, which was something we unfortunately discovered too late. Though the forum DID give us some of the answers, most of it was outdated and didn't work with the current version of the engine. The result of this was that we spent a lot of time figuring out what even the simplest methods did, time that just a little bit of Javadoc would reduce to nearly nothing, time that could otherwise have been spent on actually improving the application.