Project Reflections - Group 16

Team

Working in a team is slightly different to working alone due to a few reasons. Two of the most notable things are the increased need for a working version control system, and good code documentation. When working alone, you are familiar with all code, but when working in teams, you will not write all the code by yourself. Requiring every team member to know all the implementation details is not efficient, and not necessary. Therefore, code documentation is a good way to communicate and decrease the time to understand the code.

Another thing that is different is that when working in a group, you have a larger set of skills because people are good at different things. We utilised this by always being there on skype to ask each other questions and by having at least one other team member check through the code you had written. This way every member got useful input on their work.

When working in groups you also have to be very clear about who does what, since you don't want two people doing the same job. That requires good communication between the group members, which we accomplished by using skype frequently. The weekly sprint planning meeting also helped a lot, through it we all knew what everyone had to do each week, and what had been done. It was very time consuming, but it was still worth it even though we could have been more efficient while doing it.

Documents

Writing documents in groups can be tricky since it tends to be a bit inefficient at times. By using google docs all team members could read and write the documents together simultaneously without being at the same place, and by using skype we also had an easy way of discussing things even if we weren't in the same room (which we often were). As mentioned above, inefficiency might be a con here, but this method produces a higher quality output in return.

Software Engineering

None of our team members had any previous experience of Scrum as we started on this project, but the fact that it required less documentation than the waterfall approach appealed to us. And it turned out great. Having a sprint every week kept the spirit high, and the work got done. It was of perfect length, since most things didn't need more than a week to get finished. As we used Scrum we also should have had daily scrum meetings. That worked the first week. But since we usually focused on the math course during the early days of the week it became kind of pointless to do it during those days. It never really worked to schedule them during the weekends either, so most of the time the only formal meetings we had were during thursday and friday. This wasn't a big problem though, because we had constant contact via skype, and so everyone knew what the others were working on all the time. We are sure that if we did work 5

full days a week like normal employees, then the meetings would have been a big help.

Difficulties/Time Spending

Frameworks

In the early stage of the project, we decided to use two different frameworks: AndEngine and Artemis Entity Framework.

AndEngine is a game engine which we used for the rendering of the game and the collision checking algorithm for the collision detection in the game. It seemed promising, and it really is. But there is a big con; it has no documentation whatsoever. As it is quite popular, there are a lot of tutorials, which unfortunately mostly are written for an older version of the framework. That means that there is a slight difference between what the tutorials tell you to do, and what you really have write. To do things which are not included in any tutorial, you are required to go and read the source code manually, which of course takes a lot more time than reading the code documentation as mentioned earlier.

Artemis Entity Framework is a framework for the entity component pattern. In opposite to AndEngine, Artemis actually has code documentation which makes it a lot easier to use the framework. Although there was documentation, we still had to read up on how the entity component pattern works, as it's very different from the MVC pattern. After having base knowledge, it was easy to start developing our game as the entity component pattern works really well for game development.

If we were to do the same project again, we would spend more time to find a more well-documented game engine as developing with AndEngine was mostly about wasting hours of looking into AndEngines undocumented source code.

Audio

We originally intended to have different music in the game and in the menu. But as we found a nice track, we decided to not continue to spend time going through licenses while looking for more tracks. This was a good decision as we could spend more time on the code instead.

Graphics

Almost all graphics in the game is done by us so we didn't have to worry about finding good pictures with suitable licenses. Creating the graphics did take some time though.

Map Design

Instead of spending time to create multiple maps or a mapcreator (with a maploader in the app), we decided to spend time to make one good map. The reason for this decision was because we think that quality is better than quantity, one good map with a lot of features is better than many empty maps. We're very happy with our decision and with how the map turned out, even though a mapcreator and multiple maps would have been very nice.

Code Coverage

From the beginning, we wanted to have a tool to check our code coverage. We quickly tried the eclipse plugin EclEmma but it wasn't compatible with android testing. We also spent some time trying to include EclEmma into the build.xml file but we never managed to get it to run the tests.

Tests

During the project, we have partially used test-driven development, i.e. we have created the tests before the actual source files. This method is good because it quickly helped us find bugs in the project.