Updated Requirements Justification

In the elicitation and negotiation part of the requirements document, extra detail was added about the process of the interview and the actual elicitation of requirements. This is because in the original document we did not go into enough detail about the process itself, now there is a description of the process and more details about what happened at the interview. An incorrect argument was also removed to make the document more accurate.

In the requirements table itself most of the changes we made during this assessment were to the Non-Functional Requirements (NFRs). In the original document they were imprecise and failed to discuss usability and performance of the game. Constraint-like NFRs were removed from the table and replaced with NFRs that were more specific to the user story and much more focused on usability and playability.

Finally a discussion of the risks associated with the requirements was added so as they must be taken into consideration when requirements are being viewed.

Updated Methods and Tools

In Assessment 1, we discussed how we would be using the Scrum methodology [1] to manage our group working and keep everybody up to date with the features which are being implemented. We kept to this method during Assessment 2, however we increased the length of the sprints from around 4-5 days to about 2 weeks. We did this mainly because it fit in naturally with the way we were implementing the game, starting with the back-end where we built up the entities using Test Driven Development followed by the development of the front end using the libgdx library.

We also discussed how we would be using eXtreme Programming principles [1] as the main approach to our development of the game. We stuck to many of these principles, although some of them were not used as often as we thought they would be. An example of this was Pair Programming, which was used on occasion but not too often. While programming in pairs was a good way to write some high quality code, it decreased the the productivity of the group to a degree as it decreased the speed in which features were completed. We found it best to employ pair programming when we were fairly stuck implementing a feature.

Test Driven Development [1, 2] worked very well in developing back-end parts of the game, as it provided us with suites of tests which we could run regularly to check if any code had been broken by newly added features. There were parts of the development where we couldn't apply the technique however, mainly when developing the front-end. This was testing how pleasing the user interface looks to the user isn't something that can be tested using unit tests and such like. Therefore we had to adopt a different approach to testing and implementing the GUI. We chose to use a mixture of play testing and trial and error to test the GUI.

We've kept using many of the tools we spoke about in Assessment 1, as well as adding new tools and removing ones we didn't need. Git and GitHub are still being used as our version control tools and Slack [3] is still being used for team communication along with Facebook Messenger. We stopped using the Trello dashboard to manage current tasks because the GitHub updates sent to everybody via our Slack channel which told us about commits, merges, etc made the Trello dashboard redundant and unnecessary. We made use of the Tiled software [4] during this assessment to build the map for our game, as it was easy to use and easy to integrate with our game library of choice, libgdx [5]. All unit testing was carried out using JUnit [6] which worked fantastically with Eclipse. Finally to draw our UML diagram, we used LucidChart [7] as it was made freely available to us via our Google Drive.

Bibliography

[1] I. Sommervile, Software Engineering 10th ed, Harlow, United Kingdom: Pearson Education, 2016

[2] K. Beck, Test-driven development: By example, 1st ed. Boston, MA: Addison-Wesley Educational Publishers, 2002.

[3] Slack, "Slack: Where work happens," Slack. [Online]. Available: https://slack.com/. Accessed: Jan. 23, 2017.

[4] T. Lindeijer, "Tiled map editor," Tiled Map Editor, 2008. [Online]. Available: http://www.mapeditor.org/. Accessed: Jan. 23, 2017.

[5] "Libgdx," 2013. [Online]. Available: https://libgdx.badlogicgames.com/. Accessed: Jan. 23, 2017.

[6] JUnit. [Online]. Available: http://junit.org/junit4/. Accessed: Jan. 23, 2017.

[7] Lucidchart, Lucidchart, 2017. [Online]. Available: https://www.lucidchart.com/. Accessed: Jan. 23, 2017.

Changes to risk documentation and approach

A description of how we identified risks has been added to the document, this allows the reader to understand the process and allows us to demonstrate the completeness of our set of risks.

Risks now have an ID so they can easily be referenced throughout the documentation easily. Risks also have ownership so they can be managed by the appropriate individual and this allows the team to be more organised and deal with problems more efficiently.

Certain risks in the original document were too specific and overlapped with each other, these risks have been removed and replaced with broader risks that cover all of them at once, this means that we will know exactly which part of the table to consult if a problem occurs, which we couldn't do before due to the ambiguity that was present.

The table has now been colour coded and organised in terms of overall risk, this makes it easy for the reader to see which risks are the most important and which need the least attention.

We have not changed our approach however as it has continued to work well throughout assessment 2, minor problems such as team members missing meetings and problems with certain tools being used have been easily overcome by consulting our risk management table. And other more major risks, such as loss of data, have been avoided by consulting the Risk mitigation column of the table.