# Project Review

## Approach to team management

We decided to use agile-scrum methodology [1] with bi-weekly meetings and weekly sprints to give us a dynamic and flexible approach to software development. During our meetings, we discussed how the project was progressing and assigned tasks to each other. These tasks made up the sprint, and lasted up to a week. Our meetings allowed us to adapt our workload based on our current situation, or when necessary to help deal with new risks and requirements that arose.

Our first few meetings consisted of general discussion about what needed to be done - often this meant our meetings went off-track and we would spend time discussing ideas and tasks that weren't a priority. We noticed communication issues when key decisions were being made (Risk ID:11), as several team members were ill-informed of changes to parts of the project and were unable to adapt their own sections quickly enough. To solve these problems we decided a team leader was needed to help manage the project – this role involved planning meetings and ensuring enough progress was being made. After our first assessment, the team leader found the workload was too high, so this lead to us having two team leaders who alternated between each assessment.

Even with a team leader we found "a lack of communication" (Risk ID:11) and "arguments within the team" (Risk ID: 10) to still be an issue. In accordance with the mitigations in the risk assessment document, we decide to use GitHub issues [2] to assign members to tasks and have them feedback their progress in the comments, this made key decisions easy to find and debate.

We began to learn each other's strengths and weaknesses, and assigned tasks to each other accordingly. Initially, this meant team members with more technical experience took charge of the code, whereas other members took on more documentation – this meant some members didn't have a good understanding of key design decisions and lead to productivity loss. To solve this we engaged in pair programming to get every member up to speed with the code, and spread documentation and programming across the team.

As the project progressed, we faced dissatisfaction within the team about how the project was being managed by the team leader. The leader was assigning tasks to individuals and checking up on progress to ensure things got done, however this approach meant that some members weren't happy as they would have preferred more flexibility and team consensus on key issues.

We had a discussion and realised we hadn't clearly agreed the scope of the team leader role, and individuals were getting frustrated as everyone wanted the best for the team. We realised we weren't updating GitHub [2] Projects frequently enough, meaning it was hard to keep track of how well the project was progressing. This led the team leader to assigning tasks to individuals so progress would get made.

For the last two assessments we agreed to update GitHub more often, so that we could all keep track of project progress. We found that approach to be more effective, however some members still thought the team leader had too much control. Therefore we decided to replace the team leader with a meeting leader who would set an agenda for meetings, and we left project management to the entire team.

Despite our best efforts, issues were found with our new team structure. Although some members proactively picked up and completed whatever bits of work had to be done, some others did less work than in previous assessments due to the lack of structure and control. Over the holidays we aimed to meet at least once to ensure progress was being made. This worked for us during the Christmas holiday for Assessment 2, however over Easter this did not happen (Risk ID:11) due to a lack of leadership. If we had to do another assessment, we'd probably reinstate a leader with a different job. The leader would ensure progress is being made and set up meetings and their agendas, but not assign tasks to individuals unless there were serious problems, in which case the entire team would be involved.

## Software engineering tools and methods

From the start of the project, we chose several tools to use for software development, some of which were more effective for us than others. This document discusses how these tools evolved over the course of the project, but for overall details on tools and the justification of their use, please see the methods document.

**Programming Tools:**
We chose **Java** [3] and the **LibGDX** [4] library for development purposes, and continued to use it for all our assessments because we found them easy to work with and well documented. They featured everything we needed to create the game. Throughout the assessments we had the option of selecting a **C#** with **Unity** project, however we decided to stick with Java because we felt those projects had a more maintainable structure, and we were familiar with the technologies used.

The team decided to standardise on **IntelliJ** [5] throughout all of our assessments. An advantage of IntelliJ is its ability to produce an accurate UML (Unified Modeling Language) diagram based on our code for us, saving us the time of having to manually do this as we did in first assessment by using **draw.io** [6] as described in the SEPR [7] lecture on UML.

**Testing:**
We used test driven development (where appropriate) writing unit tests using the **JUnit** [8] library with **CircleCI** [9] to run our unit tests whenever a commit was made to GitHub. The usage of unit tests and CircleCI helped limit the occurrence of bugs in our code and mitigated “Our software doesn’t work as intended” (Risk ID: 5). This saved a lot of time that would have otherwise been spent finding and fixing bugs in the project and found them to work well together. This was a factor in choosing a Java project, as we could take our existing knowledge and apply advanced testing tooling to other teams’ projects. We later added JaCoCO to run test coverage on our code to motivate us to improve our code.

**Collaboration Tools:**
Initially we used **Facebook Messenger** [10] as our main communication tool as all members used it frequently, however we soon found that things were getting lost in one big thread, so we decided to switch to **Slack** [11]. During the development process, our team found Slack useful as it was synced to our GitHub repository to give us all updates, however throughout the process we often found ourselves reverting to Messenger for convenience, as notifications weren’t always reliable.

For remote scrum meetings we used a tool called **Join.me** [12], we found this very effective for meetings during the Christmas holiday period, and for remote demonstrations and pair programming. This helped prevent (Risk ID:11) from occurring over the holidays.

We used **GitHub** [2] as our code repository, it allowed us to intuitively keep track of code changes, and collaborate on the same piece of code. One risk that is common with code collaboration tools is “Conflicts in Git. Different members changing the same code” (Risk ID: 4) - to mitigate this we avoided working on the same file at any one time by separating tasks up, we were not aware of any tool that would remove any of these issues fully. Despite this, we found using GitHub very effective and used the surrounding functionality like projects and issues extensively. The ability to enforce a code review before merging branches proved essential for catching issues, and the integration with CircleCI meant that we were always aware of testing.

**Task Management**

To help manage the project, the Projects and issue features of GitHub were used to track of project progress and assigned to tasks to individuals. We found it ineffective to start with, because we often forgot to keep it up to date, however for the final assessments, we strived to update GitHub which proved useful as we felt the benefit of being able to see who was doing what and the project status in one place.

We conclude that if we were to do the project again, we would use the same tools again as they helped us effectively achieve the project goals. Throughout the project, we found it useful to evaluate how well our tools were working for us, and what we could do differently to improve.

## Bibliography

[1] B.Gupta, 2012. [Online]. Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf. [Accessed: 27-Apr-2017].

[2]"Build software better, together", GitHub, 2017. [Online]. Available: https://github.com/ . [Accessed: 27- Apr- 2017].

[3]"java.com: Java + You", *Java.com*, 2017. [Online]. Available: https://www.java.com/en /. [Accessed: 06- Apr- 2017].

[4] "libgdx", Libgdx.badlogicgames.com, 2017. [Online]. Available: https://libgdx.badlogicgames.com/ . [Accessed: 06- Apr- 2017].

[5]"IntelliJ IDEA the Java IDE", *JetBrains*, 2017. [Online]. Available: https://www.jetbrains.com/idea/ . [Accessed: 06- Apr- 2017].

[6] “Draw.io”, jgraph, 2017. [Online]. Available: https://draw.io/  [Accessed: 03- Jan- 2017]

[7] "SEPR", Vle.york.ac.uk, 2017. [Online]. Available: https://vle.york.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_81580_1&content_id=_2374047_1&mode=reset . [Accessed: 06- Apr- 2017].

[8] "JUnit - About", Junit.org, 2017. [Online]. Available: http://junit.org/junit4/ . [Accessed: 06- Apr- 2017].

[9] "Continuous Integration and Delivery", CircleCI, 2017. [Online]. Available: https://circleci.com/. [Accessed: 06- Apr- 2017].

[10]"Messenger", Facebook, 2017. [Online]. Available: https://en-gb.messenger.com/. [Accessed: 06- Apr- 2017].

[11]"Slack: Where work happens", Slack, 2017. [Online]. Available: https://slack.com/. [Accessed: 06- Apr- 2017].

[12]"Free Screen Sharing, Online Meetings & Web Conferencing | join.me", Join.me, 2017. [Online]. Available: https://www.join.me/. [Accessed: 06- Apr- 2017].