Method Selection and Planning

Software Development Methods

During our initial discussion about the SEPR project, we decided that we should choose a specific development methodology to work with as it would make sure that we know exactly how we are going to develop the game. There are several different development methodologies that are designed to cater for different development styles and group sizes, so we had to find the method that would work best for us.

During our research, we concluded that the best option for us would be to use the Agile method as it fits our situation perfectly. It is best used when the development is more customer-centered which, as we are producing this for customers and following their requirements, is perfect. It also “encourages teamwork, self-organization and accountability”[1] which is incredibly beneficial when working in a small group.

We also looked at several different models such as the waterfall method, but we found that it was too inflexible for us to be able to work with as it would require us to extensively plan our software before we started development [2]. As we are on a fairly short time frame, we thought that Agile methods would give us the most flexibility when working with changing requirements and any potential problems.

We have decided to adopt the Scrum implementation of Agile [3]. There are three main roles in this methodology. The first is the Product Owner. They set the project requirements and communicate the priorities to the team. Secondly, there is the Scrum Master. They are not a project leader in the traditional sense, but instead make sure that the Scrum methodology can be followed and that distractions are dealt with. Finally, we have the Development Team. This is a self-organising group that decides how to split up the tasks that have been given to them.

The actual methodology proceeds as follows. Firstly, there is the “Backlog” which is a list of all requirements for the project. It is changed throughout the project duration.

Work is organised around several short “sprints”. At the beginning of the sprint, the team meets to plan what they hope to achieve during the sprint. During the sprint, the development team meets regularly to track the progress that has been made toward the sprint goal. At the end of the sprint, the Backlog is updated accordingly, and the team reviews what went well, and what didn’t. After this, the next sprint begins.

During our project, between team meetings, we shall use two main methods of communication - Facebook Messenger, and Discord. Facebook Messenger is useful to quickly get in contact with other team members if a meeting needs to be quickly arranged or something needs to be quickly communicated. For more formal communications when team members have access to a laptop or desktop computer, Discord is more ideal. It allows for us to organise our communications into channels corresponding with the different assessment tasks.

We shall be programming this project in the Java programming language. To aid the development of a game, we have decided to use the “libgdx” game development framework [4]. This will allow us to conveniently handle the rendering of images, sound effects, and music.

As with any project of this nature, a version control system is important. Git was chosen by the team as it is the system that most members of the team were familiar with. A service to host our repository was needed. Github was chosen due to some members having used it before. Our university email addresses entitle us to free private repositories which is also an advantage.

While Github provides a good place for hosting our code repository, for documentation we have decided to use Google Drive and Google Docs. This allows for us to have a shared storage space where we can collaborate in real time on editing our documentation, as well as being a storage space for all sorts of miscellaneous files that don’t belong in the Git repository.

Team Organisation

To divide up the work between our team, we decided to first split into three pairs, and then assign tasks to those pairs instead of to individual people so that two people work on each part. This means that each pair is assigned twice as many tasks as each individual person would get, however working in pairs means each task is completed in half the time. This means the project is completed in the same overall time, but to a higher quality as the pair can help each other out and share ideas.

Exceptions to this pairs arrangement may be made depending on task priority – important tasks in the critical path may be assigned more team members in order to finish it quickly, before moving back to pairs to carry on with later tasks.

This pair system is very similar to pair programming, a software development technique that is part of the agile methodology which we will be using from assessments 2 onwards, in which a coder and a navigator work on writing code together. This will benefit us by producing higher quality code from the start, since there is an extra set of eyes looking out for mistakes in the code, meaning less time needs to be spent finding bugs and fixing code later.

For Assessment 1, the pairs initially assigned to each task were as follows:

- Requirements: Jordan Cameron, Spencer Atkins-Letten
- Architecture: Jiahao Wang, David Norman
- Risk assessment and mitigation: Ben Hassell, Tia Brigg

In addition, the following roles were assigned:

- Website maintenance: David Norman
- Meeting records: Jordan Cameron

Team members were assigned to method selection and planning once they had either finished or made substantial progress with their assigned areas.

For Assessment 2, we shall be following a similar pattern of work, with pairs initially assigned to various tasks, with opportunity for re-organisation of responsibilities as required. This re-organisation can take place at the planning at the beginning of each Scrum sprint.

Project Plan

Assessment 2 is what we prepared for in assessment 1 so we decided to follow the same management method initially, we split up each work package as detailed in the Gantt chart into groups of 2-3 people, we've had to change the plan a bit as it includes the first code implementation work package that everyone should be a part of, lasting for all three remaining tasks, however the other tasks such as updating assessment one deliverables, the testing and the concrete architecture can be done by splitting up. This assessment will be slow to start as it will be our first time learning new tools, but we've given ourselves a heaver intensity of work later on as seen in in the Gantt chart [5] to give us time to learn what we need to, from the beginning rather than completely as we go, giving us more efficient code we don't have to rework from the get go.

Assessment 3 is the shortest task, we will begin it as a group by reading through the documentation and code of the group who originally worked on it so we can understand what they have already made and what they planned which in turn will give us a better understanding on why they've coded it in the way they have, from this we can begin organising what work is remaining and split up into pairs to work on it, while each person codes we can have the other person asking why they've coded in a certain way and pointing out ways to make it more efficient. We will need to meet at a moderate frequency so we can go over what has been done and document it appropriately.

Assessment 4 is the longest of all the tasks, however we can follow a similar methodology from assessment 3, they are the most similar tasks even if the time frame is very different, aside from the Project Review Report which we can all give our opinions in as well, I'd say assessment 4 will require us to change our methodology to one which is useful for many people who have to work on the same tasks, and we divide up the work based on tasks undefined, such as GUI modification, implementations of any systems not included such as weather systems to make it a more complete product, this task will likely require longer but less frequent meetings, we can still work using previously used methodologies though, such as the use of sprints. Testing in every task however should be constant throughout the assessment, progressing systematically with the implementation in each one so any problems are immediately dealt with.

This Gantt Chart [5] provides an initial abstract overview of assessments 1 to 4, I've included shading with darker meaning a more intense workload, the tasks are split up into work packages for each of the four assessments, it keeps abstract so we are more focussed on getting each work package done within it's time frame rather than focussing on the smaller parts within it however for assessment 2 at least, the individual tasks are covered in detail by the critical path diagram below.

We created a Critical Path Diagram [6], this shows us how tasks depend on each other within the second assessment, showing us how to progress, though there are only 11 weeks total as shown in the Gantt chart, we can see that the total for the critical path is 24 weeks, this is because several of the tasks can begin and progress at the same time, one week tasks tend to be setting up URLs and such.

Bibliography

[1] Cprime. *What is Agile? What is Scrum?*, [Online] Available: https://www.cprime.com/resources/what-is-agile-what-is-scrum/ [Accessed: November 6th, 2018]
[2] TPX Manifesto. *Agile vs Waterfall: Comparing project management methodologies*, [Online] Available: https://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/ [Accessed: November 6th, 2018]
[3] Scrum Alliance. *Learn about Scrum*, [Online] Available: https://www.scrumalliance.org/learn-about-scrum [Accessed: November 6th, 2018]
[4] libgdx. *Goals and features*, [Online] Available: https://libgdx.badlogicgames.com/features.html [Accessed: November 6th, 2018]
[5] Gantt chart: https://davidjnorman.github.io/SEPR/assessment_1/GanttChart.pdf
[6] Critical path diagram: https://davidjnorman.github.io/SEPR/assessment_1/CriticalPath.pdf