Method Selection and Planning

Team name: Endeavour (Team 28)

Team Members: Sarah Berry, Finley Brown, Yupeng Di, William Griffiths, Kurtas Joksas, Fraser Masson

Method Selection and Planning

Introduction

This document will outline the software engineering methodologies and tools used throughout the project, in addition to the efforts made in organisation and planning.

Development and Collaboration Tools

The following tools were utilised throughout the project to aid in collaboration and development:

- **Discord**
  This was used by all our team members as the primary method of communication – meetings, discussions and sharing of small resources was all performed using this platform.
  We heavily utilised features such as screen sharing and voice channels to collaborate and discuss in real time during development.
- **GitHub**
  Centralised place for all code – allows everyone in our team to view and contribute to the development of the project (source code). Branches and versioning were utilised heavily to ensure collaborative efforts were constructive and not destructive. GitHub Desktop was also used to minimise teaching necessary to those unfamiliar with the command line utility.
- **Tiled Map Editor**
  Map editor designed specifically for development of video game maps. Free and easy to use, allows the use of efficient editing tools to minimise development time such as stamp, fill, and layers.
- **Libgdx**
  Cross platform Java game development framework. Well documented used for game design specifically.
- **Java**
  Object oriented programming language used for the project. Easily made cross platform and easy to develop with, alongside compatibility with libgdx made this ideal for our project.
- **Neovim / Visual Studio Code**
  Development tools used to write and run the project's source code, using Gradle to build the project. Offers features such as syntax highlighting, folder organisation, debugging tools and other programming specific features which make development faster compared to a traditional text editor.
- **Google Suite (Drive, Docs, Slides, Gmail)**
  Google applications such as Google Docs, Drive and Gmail were used for file sharing and development of documentation. Allows for real time collaboration on documents meaning files could be stored in the cloud and accessed by any member of the team.
- **Adobe Suite**
  Adobe applications such as Photoshop and Illustrator were used to develop and edit graphics for the game, such as sprites, logos, and other visual matter.

Before deciding on the above tools and software, we also considered alternatives:

- **Unity**
  Allows for development of top down 2D games and is also well documented, but for the scale and complexity of our project this would have been overkill (examples include 3D models and lighting engines, which we would not be using in the project). In addition, this engine uses C# which not all our team are familiar with.
- **Microsoft Teams/Zoom**
  Other software for communication were also considered early in the project's development but features such as text channels for discussions and voice calls without a URL make Discord more suited to purpose. Discord also offers screenshare which is easy to use and easily accessible.
- **Lwjgl**
  Another game focused library for Java – however we chose libgdx over this as it offers functions necessary for our project that lwjgl does not. This saves us time in development as we can use pre-programmed and well tested code in our project.

Software Engineering Methodologies

Our team worked in a scrum methodology throughout the development process – this involved weekly sprints where work was completed, and then discussions and new tasks were assigned at the end of each mini-sprint. The length of the sprints was limited due to the timeframe given to us for the project. This helped us to keep on track with our progress, communicate more often, and support others working on different parts of the project. In addition, the flexibility of this approach allowed us to reorganise and succeed when behind on certain timeframes/deadlines internally.

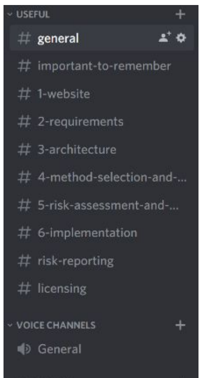
For the second project where new requirements were added, agile methodology was utilised, specifically a scrum approach. we divided the project into sprints, getting together for a daily meeting reporting what each person had accomplished and what the plan for the rest of the day would be.

We were each handed a specific scrum role, which remained flexible throughout the project, sticking to roles when it felt necessary and vice versa. Finley was the scrum master, Sarah was the product owner and the rest of the team were development members. As the product owner, Sarah was responsible for providing clear direction and deciding which tasks required the most focus at any given time. Finlay was responsible for keeping everyone on task and aiding in Sarah’s decision making regarding the prioritisation of tasks. Everyone else performs the tasks at hand making sure to follow the values decided on by the scrum master and product owner.
We decided on this approach as we wanted to stay flexible around the changing requirements of the project. There were constantly changing factors due to possible new game ideas we’d brainstormed from the users requirements. For example we changed our ideas for powers from the player having a shield that means health does not decrease to just increasing the player's max health. Due to our agile method planning this was a seamless transition and did not take a heavy toll on time management.

Team organisation approach

Our team approach has been mainly focused around daily meetings (calls) with the team, involving discussion about the progress made during the day, decisions about what needs to be done next, and who will be assigned to aforementioned tasks. Our method for deciding this has always been to ask each member what section of the project they’d be interested in doing or what part they feel most confident about, then splitting the team of 6 into 3 groups of 2, these small units allow us to break down each problem into smaller tasks that can be tackled easier and simultaneously, making efficient use of time.

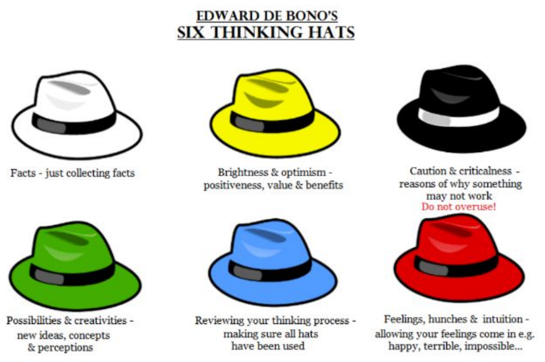
By having meetings consistently, it is easier to identify issues that team members are having with the tasks they have been assigned and people stay up to date with other aspects of the project. Almost any problem faced would be dealt by the next meeting which means we were never stuck on any problem for longer than a day.
We make sure to never have only one person working on a section at any given time, as this could put uncertainty and unnecessary pressure on the task the individual is doing. Having members who are the most confident on certain sections means that our efficiency is increased, and the quality of work is improved - ultimately leading to a higher quality project.
If there is a problem that occurs from any one of the 3 team groups, they will report back to the main discord group (online chat) and we will work as a team to figure out how to solve the problem - whether that is to switch one of the team group members out for another or to simply provide a piece of advice – this depends on the problem presented. We constantly and consistently criticise each other's code and documentation as it allows us to spot any mistakes and listen to other invaluable ideas from other group members; improving our project.



Within our Discord server, we have different channels allocated to discrete sections of our project – these proved valuable as it allowed us to have discussions in isolation of the other mini-teams and meant that we stayed focused on the task at hand. To maintain work-life balance, we also opted to include a #memes channel which allows for non-project related matters to be posted – this helped us build a bond early in the project.

We also had additional channels such as #general, #important-to-remember and #licensing for other miscellaneous topics and discussions during our meeting calls.

During the meetings, we also referred to the six thinking hats thought of by Edward De Bono. These allowed us to keep the discussion focused, gather useful information from other team members, and logically consider if certain proposals are feasible. Using this system ensures that we make the 2-hour timeframe of our usual weekly meeting as efficient and to the point as possible.


EDWARD DE BONO'S
SIX THINKING HATS
Facts - just collecting facts
Brightness & optimism - positiveness, value & benefits
Caution & criticalness - reasons of why something may not work
Do not overuse!
Possibilities & creativities - new ideas, concepts & perceptions
Reviewing your thinking process - making sure all hats have been used
Feelings, hunches & intuition - allowing your feelings come in e.g. happy, terrible, impossible...

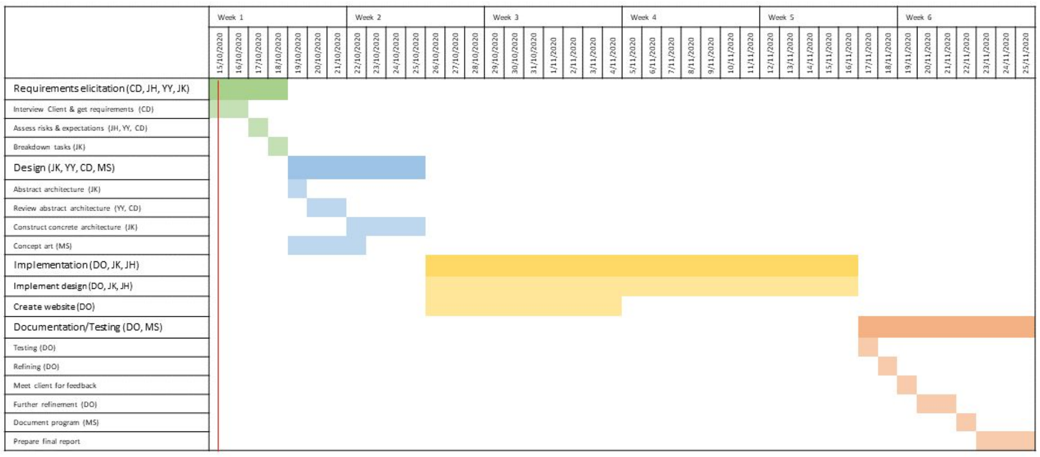
Systematic Project Plan

Our project plan was mostly linear – people handled different areas during the project. The responsibilities of these were split up based on their strengths and the benefits they could bring to that specific area. This is more efficient than a model where everyone does equal chunks of the overall project, as more focus can be given to each part and not everyone having a diverse and equal skill set makes this logistically difficult.

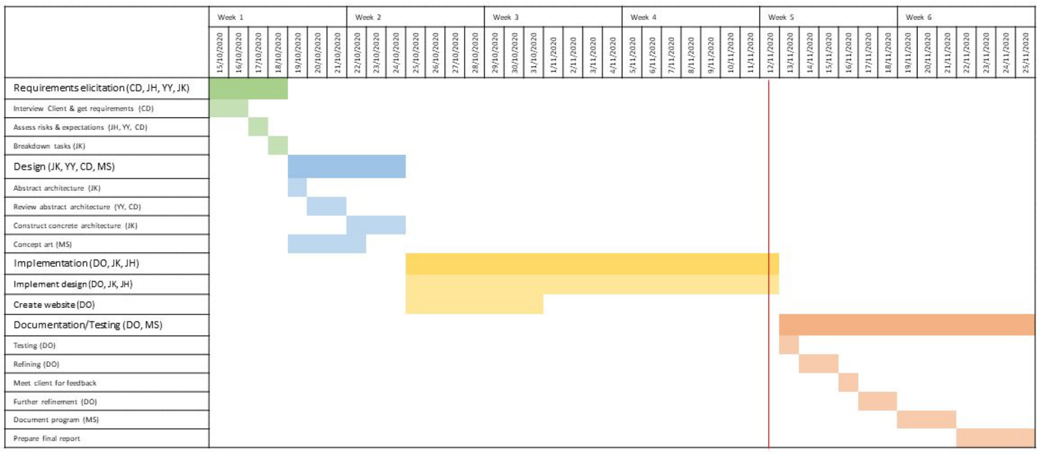
For this reason, there were different combinations of people working on different sections. People missing from the current section would be working on cleaning up the previous sections and bringing new views and interpretations to the current work.
The critical path was decided based on what made the most sense: Requirements → Design → Implementation → Documentation → Testing/Feedback. We believed in this model, each section depended on the completion of the last, and to jump ahead would cause avoidable issues.

It was decided to use a Gantt chart to plan responsibilities. This was done on account of their ease of modification and readability. It was updated each week with new times and responsibilities before images of each week were taken at the end of the project when creating this document.
The plan saw the phases of the project split up such that implementation took up the most time, due to the ambitious nature of the project. Requirements, design, and testing were planned to take up just shy of a week each, with Implementation predicted to take about 3 weeks. As the project went on, the sections turned out to take a day or two more than predicted, however this was not a concerning issue as we had planned for issues like this to arise. For example, halfway through implementation, notable issues were occurring that saw the timeline increased for implementation, however it was brought back down to the expected time. Overall, the prediction was not far off, and we were successful in completing the required work in the timeframes we allocated ourselves. Shown below are the Gantt charts for the six weeks of our project:

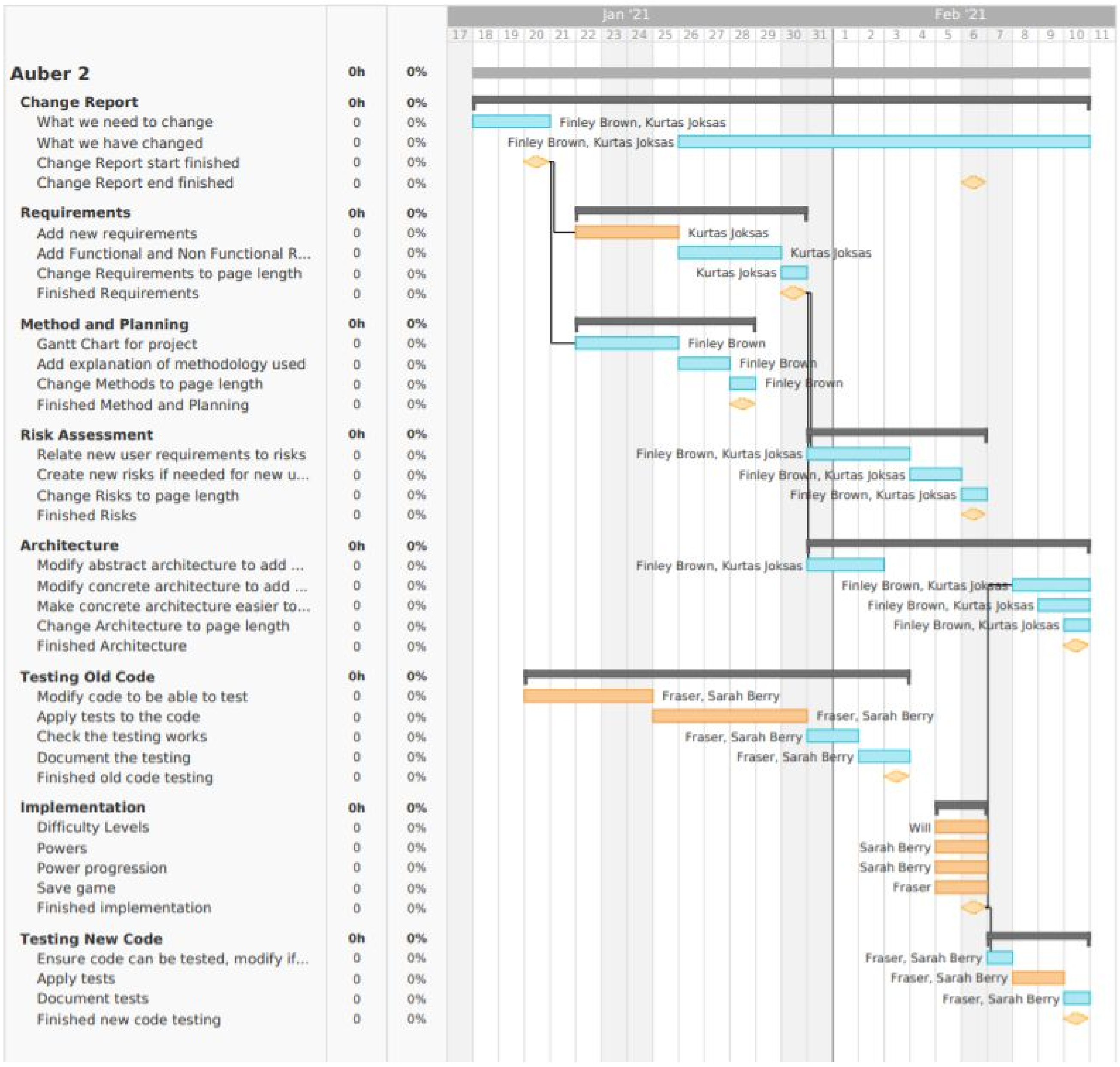
Week 1 (15/10/20 – 21/10/20)- initial gantt chart



Week 5 (12/11/20 – 18/11/20)- final gantt chart



This plan did not change much however nearing the end of the project it became clear that there should be more time spent on the documentation and testing rather than the implementation.



Auber 2	0h	0%
Change Report	0h	0%
What we need to change	0	0%
What we have changed	0	0%
Change Report start finished	0	0%
Change Report end finished	0	0%
Requirements	0h	0%
Add new requirements	0	0%
Add Functional and Non Functional R...	0	0%
Change Requirements to page length	0	0%
Finished Requirements	0	0%
Method and Planning	0h	0%
Gantt Chart for project	0	0%
Add explanation of methodology used	0	0%
Change Methods to page length	0	0%
Finished Method and Planning	0	0%
Risk Assessment	0h	0%
Relate new user requirements to risks	0	0%
Create new risks if needed for new u...	0	0%
Change Risks to page length	0	0%
Finished Risks	0	0%
Architecture	0h	0%
Modify abstract architecture to add ...	0	0%
Modify concrete architecture to add ...	0	0%
Make concrete architecture easier to...	0	0%
Change Architecture to page length	0	0%
Finished Architecture	0	0%
Testing Old Code	0h	0%
Modify code to be able to test	0	0%
Apply tests to the code	0	0%
Check the testing works	0	0%
Document the testing	0	0%
Finished old code testing	0	0%
Implementation	0h	0%
Difficulty Levels	0	0%
Powers	0	0%
Power progression	0	0%
Save game	0	0%
Finished implementation	0	0%
Testing New Code	0h	0%
Ensure code can be tested, modify if...	0	0%
Apply tests	0	0%
Document tests	0	0%
Finished new code testing	0	0%

This specific gantt chart did not need to change at any point, this is due to the project not changing much and as we had experience with the prior project it was easier to estimate how much time would be required for each section.

The orange bars are sections we prioritised due to their importance in this section of the project, the connecting lines are dependencies of the project e.g. we can only do concrete architecture if we have finished the implementation.

We had no more space for the critical path so have linked it below.

https://github.com/jonWadman/auber2/blob/master/docs/pdfs/Critical%20Path.pdf