ENG1 Method Selection and Planning

Cohort 3 - Group 28

“Team 28”

Muhammed Salahudheen
Joel McBride
Jamie Rogers
Maciek Zaweracz
Rhys Yeaxlee
Alex Spencer
Alex Firth

We approached this project with a strong emphasis on collaboration, teamwork and efficiency. We wanted to make sure that we worked as a team and communicated well at all times so that the project could be done to a high standard. We did this by using social media platforms such as Discord to organise meetings and discuss any ideas and thoughts anyone may have. We also agreed on a weekly team meeting, so that we can ensure that everyone was on the same page and so that we all stayed organised. The weekly meetings are useful in making sure that everyone is doing their part and also be a place where we can support each other and give each other ideas and inspiration.

Due to the dynamic nature of game development and the changing requirements that emerged from customer meetings, we have chosen to use an agile software development methodology. This allowed us to incorporate feedback into the development and streamline the process. We focussed on pair programming where each section of code was assigned to a group member and reviewed by another team member, enhancing code quality and modularity through the sharing of ideas. Additionally, new functionality was implemented in new GitHub branches resulting in small prototypes that could be independently developed (and later tested), ensuring that our main codebase remained stable. This method of isolated development aligned with Agile's principle of continuous development and integration.

When selecting what software we should use to implement the project, we all researched and shared our findings. We discussed main java libraries such as Libgdx, FXGL, Slick2D etc. and came to a conclusion that LibGDX was the best one. This is mainly because of the vast amount of resources and information that was available on it. LibGDX has cross-platform support meaning that all of us, regardless of the operating system we are using, are able to code if needed. LibGDX is known for its high performance as it has a low-level access to hardware acceleration features.

The active and large community LibGDX has made it very appealing to us, as this meant that it was easy for us, as developers, to take inspiration from, learn and troubleshoot issues. Furthermore, LibGDX has a wide range of features made for game development made for features such as, graphics rendering, input handling, audio playback and more. Alternatively, we could have chosen FXGL, Slick2D and other libraries but we chose not to. This is because FXGL and Slick2D does not offer the same level of performance as LibGDX and has a limited cross-platform support which may have compromised any other future developments that may occur.

FXGL is not as feature rich as LibGDX meaning that FXGL does not provide the same level of flexibility. Many developers find themselves needing to implement custom solutions and optimizations which wouldn't be very beginner friendly hence why we did not choose FXGL. Also, after doing a little more research, we found that FXGL has a steeper learning curve compared to other frameworks in java. This would make it challenging for us to get started with game development as newcomers. Slick2D on the other hand, has some performance issues as it may not leverage hardware acceleration effectively resulting in lower frame rates and a less smooth gaming experience for consumers.

Slick2D has a lack of modernization and moderation meaning that its architecture and design may not be as modern or flexible as newer frameworks. This is due to the lack of development which has slowed down significantly over the years. It has fewer updates and contributions to the community and its community is not as active or large as other frameworks.

Overall Slick2D and FXGL may still be suitable for smaller-scale game projects, which in our case would work, however the lack of documentation, tutorials and community support means that troubleshooting issues and learning the framework would be harder resulting us concluding that LibGDX was the best to use.

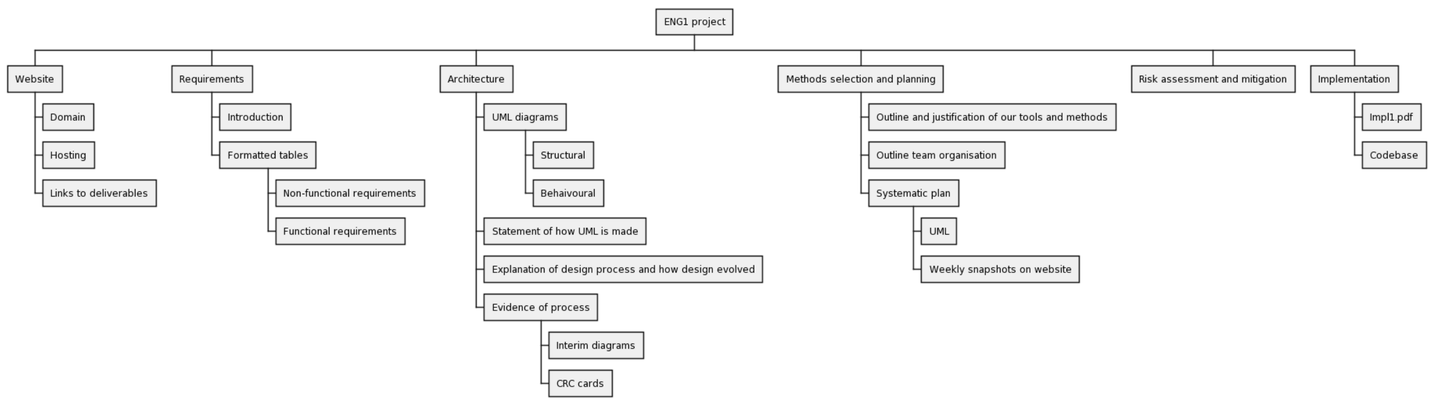
We used google docs for team collaboration for several reasons such as real-time collaboration, cloud-based accessibility, etc. Allowing real-time collaboration means that the whole team can edit and read a document simultaneously while changes are being made without needing for manual merging of changes. One of the many conveniences of google docs is that it has cloud-based accessibility meaning that team members can access documents from anywhere with an internet connection. This allows us to work remotely while being able to keep track of progress others may have made. One of the most important and impressive features of google docs is the version history. Google docs automatically saves version history, allowing us to track changes, revert to previous versions and be able to see who made what edits. This feature provides transparency and accountability as well as security and safety in terms of not losing the work. For example, if someone accidentally deleted something important, then we can easily retrieve it by using version history.

GitHub is built on top of Git and Git is an open-source version control system. A distributed version control system means that the entire codebase and history is available on every developer's computer which allows for easy branching and merging. Branching is what makes Git really powerful. Branching is where multiple create separate copies of their code and merge their changes into the main copy; in other words, you can “branch” your code out and merge your code into the main “branch”. Branches are meant to be temporary and it is a good habit to delete branches when work is completed. GitHub is a company that offers a cloud-based repository hosting service, essentially it makes it easier for developers to use Git for version control and collaboration. GitHub provides a centralised repository for storing code similar to google docs making it easily accessible to all team members. This ensures that everyone is working from the same codebase and reduces risk of code conflicts and divergence. Github’s pull request feature allows teams to propose changes to the codebase, which can then be reviewed, discussed and verified by other team members before being committed and merged. This promotes knowledge sharing and good code quality by facilitating and allowing peer reviews and feedback. Github provides robust issue tracking and project management tools allowing teams to organise their work effectively and due to the popularity of GitHub, there is a wide range of third-party tools and services for features such as code analysis, project management and more to supplement good organisation and communication between team members. The popularity of GitHub also means that GitHub is host to one of the largest communities and open-source projects, providing opportunities for knowledge sharing and networking. Furthermore, GitHub provides built–in support for documentation and wikis, allowing team members to document code while working which promotes good practices and habits and makes the code easier to understand for others. All of these reasons helped us decide Github as the right platform for us to work on our project as well as some team members already being familiar with it.

We also used Jamboard for some of our brainstorming and making of CRC cards and more. This is because just like google docs, Jamboard is a good tool for real-time collaboration and it has good integration with google docs making it perfect for our project and workspace.

For development tools, we chose to use a mixture of Visual Studio Code / Code - OSS, and Intellij IDEA. The main game implementation work was done in Intellij to make use of the inspection, refactoring, and build tools, while extra edits and work on the website were done with vscode. Both these tools have git integration built in, and some team members also used GitHub desktop for graphical source control assistance while some team members used git on the command line within their IDE.

Using gradle as the build system was an easy choice since it’s recommended for libGDX projects by the engine’s wiki, and the project creation tools automatically generates gradle build scripts. Personally, I don’t like the dynamic build script system with the Groovy SDK, I prefer static well-standardised build config systems like Python’s pyproject.toml or Rust’s cargo.toml. However, we didn’t need to edit the build scripts much, just to add tasks for formatting the code and packing textures into a texture atlas. The IDEs we used offer good integration with gradle while it can also be used easily from a shell which makes future CI/CD convenient.


ENG1 project
Website
Domain
Hosting
Links to deliverables
Requirements
Introduction
Formatted tables
Non-functional requirements
Functional requirements
Architecture
UML diagrams
Structural
Behaivoural
Statement of how UML is made
Explanation of design process and how design evolved
Evidence of process
Interim diagrams
CRC cards
Methods selection and planning
Outline and justification of our tools and methods
Outline team organisation
Systematic plan
UML
Weekly snapshots on website
Risk assessment and mitigation
Implementation
Impl1.pdf
Codebase

Team Organisation

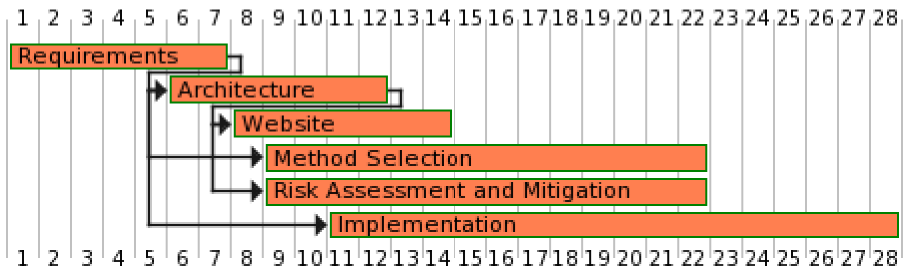
Our team organisation was pretty free and agile where we consistently did all parts of the project, and regrouped to see the product we had made, then add on feedback we gave each other. This was, no one was spending a lot of time creating a massive file which in the end was not what we needed, instead, by breaking down the project into smaller manageable parts, we were able to stay consistent and work at a good pace. We split the work amongst each other, with people who were more proficient with coding and git working on the implementation while others worked on the documentations. This way, people unfamiliar with git could slowly watch and learn and get used to it by the time we got to the second part of the assessment.

The success of our team was largely due to our dedication to honest and consistent communication. Through stand-up meetings, planned check-ins, and the usage of a messaging app like Slack, we created channels for communication where we shared information, resolved conflicts, and preserved unity within the team. Good communication promoted a culture of openness and responsibility, which made cooperation easier and reduced miscommunication or delays.

Adaptability also played a huge role in the success of our project so far. Each member of the team would help out with other parts of the project when needed. This helped to foster a creative work environment where everyone's ideas and opinions were heard. This approach was taken as every team member had specific qualities which were useful in all parts of the project.

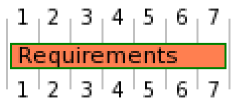
Planning snapshots on the website: https://team28.tech/2024/03/20/plan-process.html

Ideal plan



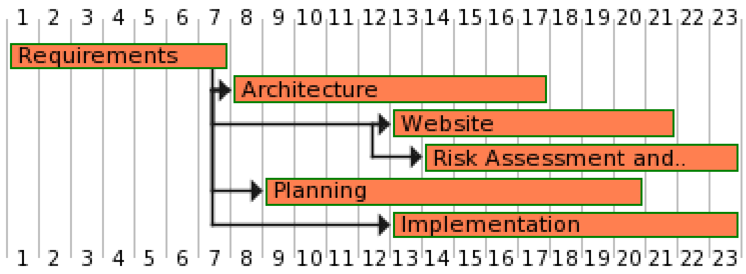
At first we had the idea that every key task would have dependencies on each other meaning we couldn't move onto the next task until we completed the one before. We didn't necessarily have anything figured out in terms of allocating work, we all worked on the requirements at first and getting used to working with each other whilst discovering each person's strengths and weaknesses along with overall personalities.

Week 1:



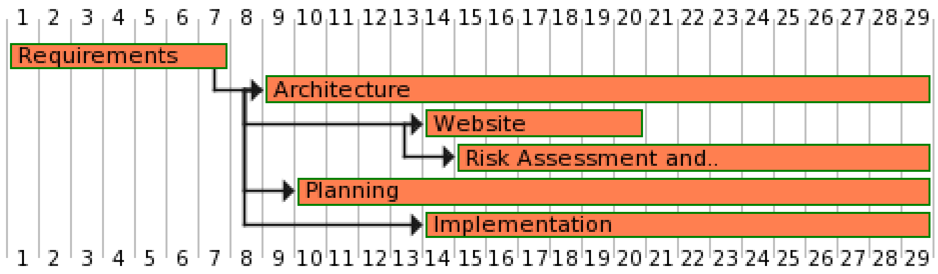
We worked on the requirements during the first week as we still hadn't decided the groups individual roles, therefore we all thought about the project in our own ways while some of us worked on the requirements.

Week 2:



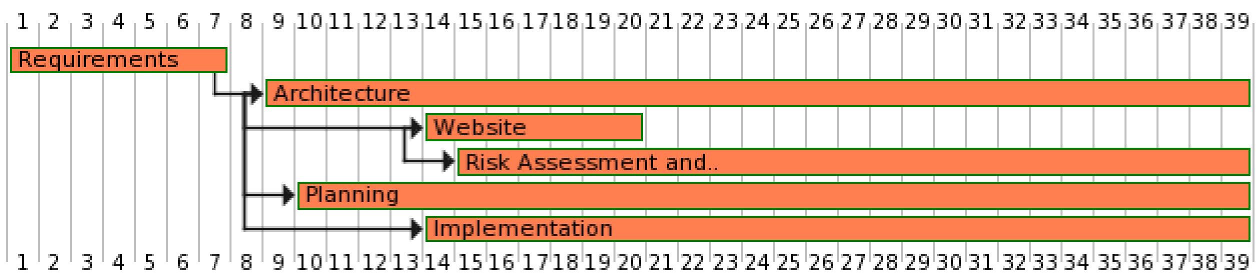
At this point, we decided that we would allocate questions/key tasks to everyone to create a better structure amongst ourselves relying on each persons given strengths. Architecture, Risk management and Implementation were all split and each involved 2 people, However Planning and the Website was done by an individual person. For every section, we thought it would be of our best interest to pace it according to the weekly lectures and what was introduced to us and slowly implementing and building our work so that we don't burn ourselves out but also we didn't want to spend too much time writing and getting ahead of ourselves incase we implement things differently or create documents that could become obsolete.

Week 3:



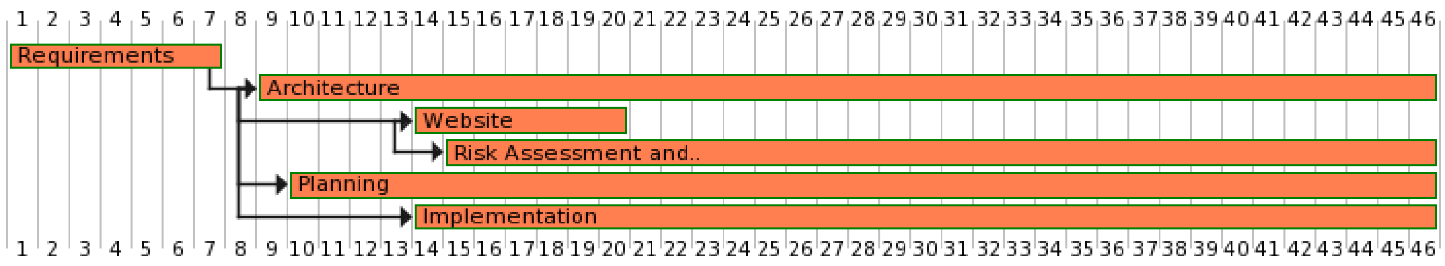
During week 3, we continued our Implementation, Architecture, Risk assessment and Planning. All of these required a great amount of time and consistent communication as we constantly had to add new things according to what we were working on. For example, to continue our planning, we had to discuss our future plans as well as discuss how everyone was doing with their task, on the other hand we would constantly discover new risks that we would then have to add onto our risk assessment and mitigation.

Week 4:



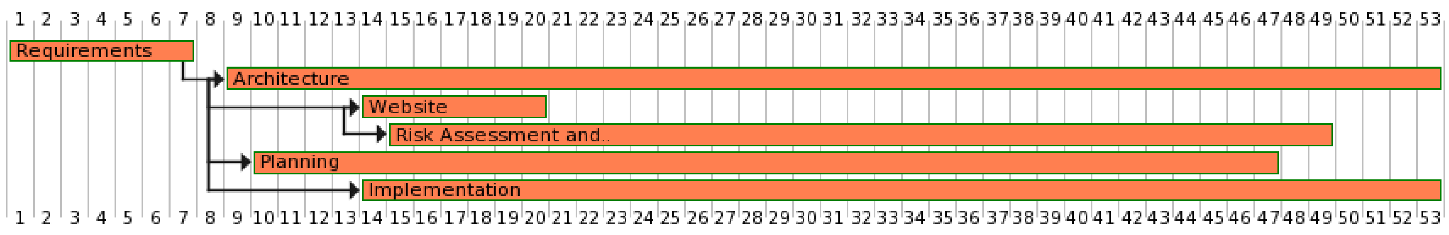
During this week, we discovered that the architecture team was slightly struggling with ideas therefore, during our weekly team meeting, we spent some time discussing and helping each other in order to get back on track with our original plan.

Week 5:



We are still working on architecture, risk assessment and mitigation and implementation. Risk assessment and planning is mostly completed with just small additions here and there according to further team implementation and work being done.

Week 6:



We have everyone doing implementation or parts of it such as implementing code or simply commenting the code. We're doing our final touches for submission and ensuring every section is completed to a good quality