# Method Selection and Planning

Group Number: **Cohort 1, Group 11**
Group Name: **Y111 Studios**

**Group Members:**

1. James Hutchinson
2. Somto Igweonu
3. Robert Kisloski
4. Sam Knight
5. Kenneth Kok
6. Ashish Kumar

This document outlines the planning phase of the development process of our game, highlighting the engineering methods and the development & collaboration tools that we have utilised to help us work effectively as a team to achieve our goals. This also includes our project plan and how this plan changed over time as we made progress.

## Software Engineering Methods and Development/ Collaboration Tools

The main software engineering method is feature-driven development, where the goal is to follow the requirements and split the implementation into separate features that can be developed simultaneously. This style of development promotes quick results and emphasises good object-oriented design by aiming for each feature to contain objects designed to be flexible and applicable over a wide range of use cases. This integrates well with the Java programming language, which is a strongly object-oriented language.

The main development tool used was the Maven project management tool, which has a number of features that make the development process easier. The main feature that was beneficial was the project build automation, which can be defined as an XML file known as the Project Object Manifest (POM) file. This allows the build process to be modified using premade plugins, which allows for the final JAR file to be self-contained and portable.

One of the requirements for this project was for all dependencies and resources to be included in the final JAR. This was achieved by using the resource and shade plugins. The resource plugin includes the resource in the final JAR file, which allows them to be referenced using the internal files, allowing the JAR to be portable. Further, the shade plugin is used to add all of the dependencies to the JAR file so that the JAR is fully portable. This includes all of the native libraries that the LibGDX library depends on. This is resolved by Maven as the transitive dependencies are resolved by the build system.

Another benefit of using Maven was dependency management, as the Maven repository is very large and contains many popular frameworks, such as JUnit and LibGDX. The addition of dependencies was also very easy as they can be added by specifying each dependency's group and artefact IDs', followed by the version of the dependency to use. Another benefit of this approach is that each dependency has a scope to be used in, which defaults to the compile scope (available in all phases of the build lifecycle) but can be limited to only be available during testing or, even more restrictive, only limited to at compilation. These scopes can be used to limit the size of the final JAR file as some dependencies used, such as Lombok do not need to be included in the final JAR but are needed at compile time for the compiler to correctly generate the code from the annotations.

In terms of versioning, Maven allows a version to be set for the project inside the POM file. During this stage of development, where a stable API is yet to be established, the default "0.1.0" version was used to denote that the project is still in the early stages of development.

For a version control system, Git was used. This was selected as it is a very common version control tool for software projects, is very well documented, and has many tutorials available for some of the more complex features. One of the other main features is branches, where a separate branch can be split off from a specific point and merged back in later to the main branch, which fits well with the feature-driven development process we selected. Due to this, merge conflicts should not occur or be trivial to resolve.

Expanding upon Git, we used GitHub as a way to have a remote repository, which allows for collaboration as well as maintaining a consistent state for the project among the team. GitHub also has additional tools that are built upon Git, such as pull requests and the code review system. Issues are also a feature implemented by Github, which allows each feature to be given an owner and allows them to be linked to a pull request that can be closed when the pull request has been merged. This allows for easy collaboration on the project as each team member can clone the remote repository, make local changes, and push them up onto a branch, which can be merged later and reviewed by others. This review process is important as it allows another to request changes to the code, which helps in enforcing a consistent code style across the code base.

These tools are a good fit for this project as they automate repetitive processes, which reduces the risk of errors occurring due to incorrectly applying the build process, or by pushing breaking code to the repository.

Some alternatives that were considered were GitLab in place of GitHub. This was not selected as GitHub has a simplified UI and a larger community making it easier to adopt than GitLab. The ease of use and familiarity that comes with using GitHub makes it a more suitable platform to use than GitLab.

Another alternative tool that was considered was Gradle as a project management tool over Maven. Maven was selected over Gradle as it has a simpler way to configure build processes by using a predefined lifecycle and uses its large number of stable plugins to simplify build tasks, such as shading. The configuration of the project can also be done in an XML file, which provides a familiar and easy-to-read configuration as it is more declarative than Gradle. While Maven can be slower than Gradle as it lacks in-built caching and parallelism, these can be added using the plugins, making this downside less impactful.

The other software engineering method that was considered was using the agile methodology. This is an iterative approach and is similar to feature-driven development in many ways, but due to the short time frame of the project, the idea of using sprints did not seem to fit well with the time frame of the project. This entire section of development could be considered one sprint as feature-driven development is effectively the internal methodology of one sprint applied over the entire time frame of the project. Since the list of requirements was defined early on and the architecture was understood, using feature-driven development seemed like a better fit as it focused on each component of the final system, resulting in more flexible code being made.

## **Team Organisation**

Our team was split into three sections: graphics, data representation, and logic. Each section had roughly two people working on the planning, implementation, and write-up of that particular section with there being no leader of each one owing to the small number of people in each section. This worked well to encourage cooperation within the sections whilst also allowing for swift progress to be made as two separate tasks could progress at even rates rather than one person having to constantly stop their task to oversee the progress of the other. This also worked well with Git version control as the two people could work on separate branches and review the other's merge requests.

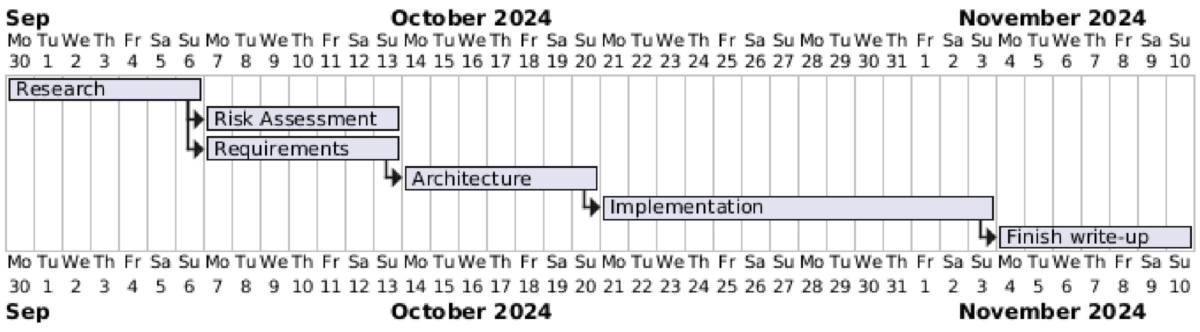
The team was split into those three sections as they are very distinct parts of the game meaning each could make progress without waiting for the others, making the process faster, and there weren't likely to be many conflicts when the sections were merged. Each section also required a different knowledge and skill set meaning people could choose a section best suited for their abilities and it meant that no one had to have the full depth of knowledge of each section which sped up the research process.

Any work that lay outside of the defined sections was split as evenly as possible between all members of the team with Sam, being the most experienced with game development and also the libGDX framework, taking charge of the final decisions after hearing everyone's opinions and also of laying out the timeline of when each section should aim to achieve a particular target for. This meant that each person was still held accountable for their work which served to maintain the quality of the game as well as ensure it would be ready in time for the deadline provided whilst allowing people the freedom to experiment with what was a new framework for many which was important as, given the creative liberty provided by the requirements, people had to feel confident in trying out new ideas. This was where it was also important that each section had at least two people as, when feeding back to Sam, there was always at least one other person with the same knowledge of that section who could act as a second opinion.

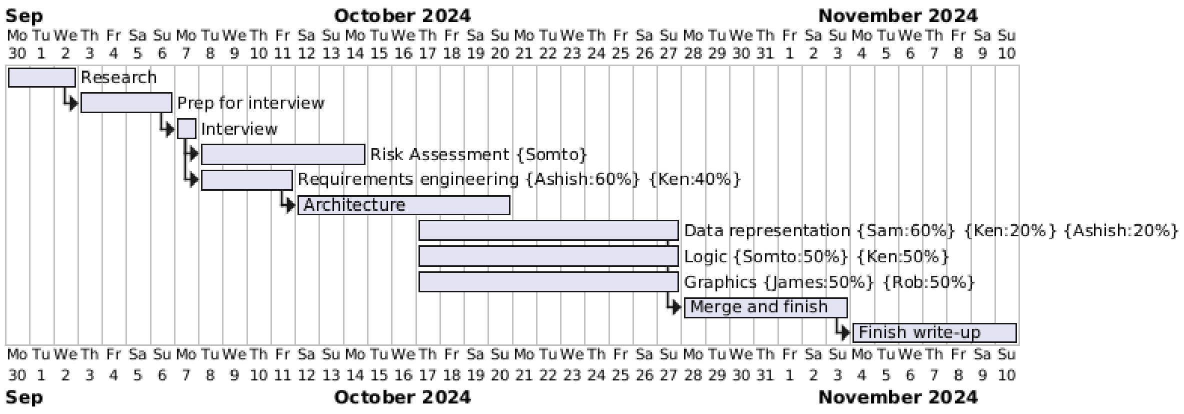
**<u>Project Plan</u>**

<u>Week 1</u>
The project has just started. Most of the time is spent getting to know each other and understanding the project. As such, there is no concrete plan.

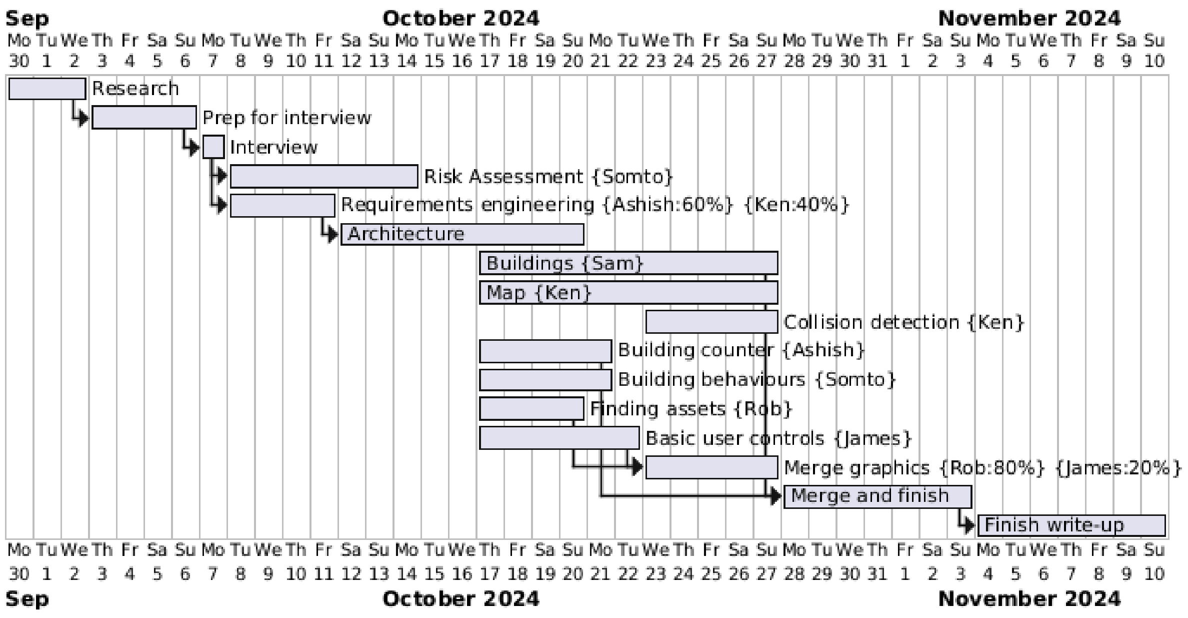
<u>Week 2</u>
By following the weeks assigned to each of the lecture slides, we can produce a rough plan.



<u>Week 3</u>
We now have our customer meeting booked and a rough idea of how the implementation work will be divided. Based on this, we can start to divide tasks between people. There is also some overlap as some risks will only be realised once the architecture is explored and some changes will be made to the architecture plan once the implementation is explored.



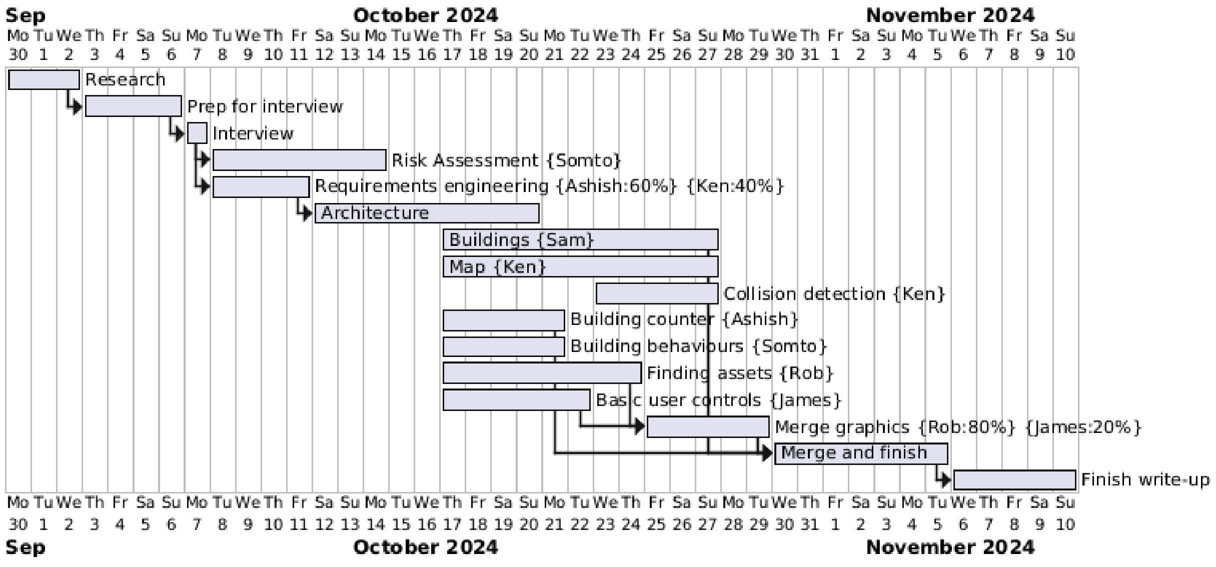
<u>Week 4</u>
Having started defining the architecture and thinking ahead to the implementation, we can now further divide it into a series of more specific tasks.



Week 5
Finding assets took longer than expected which has pushed back merging however most of the rest of the plan has remained mostly the same owing to the lack of dependencies on other tasks in the implementation stage. The diagram is linked on the website (*Week5.png*).

Week 6
With lower levels of in-person contact due to reading week, it is even more important that we follow the plan therefore we have worked hard to stay on target meaning the plan for this week is unchanged from last week. The diagram is linked on the website (*Week6.png*).

Week 7
After reading week, we are on target and have merged most of the elements of the game together. With the team now focused on one task, it should be easier to remain on target.


Sep
October 2024
November 2024
Research
Prep for interview
Interview
Risk Assessment {Somto}
Requirements engineering {Ashish:60%} {Ken:40%}
Architecture
Buildings {Sam}
Map {Ken}
Collision detection {Ken}
Building counter {Ashish}
Building behaviours {Somto}
Finding assets {Rob}
Basic user controls {James}
Merge graphics {Rob:80%} {James:20%}
Merge and finish
Finish write-up