Method Selection and Planning

ENG1 Team 9
1. Jacob Dicken
2. Bertie Cartwright
3. William Croft
4. James Dovener
5. Henry Chan

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.

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 artifact 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.

Git was our chosen version control system, as it is a very popular 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 to the main branch later, which fits well with the feature-driven development process we selected. Due to this, merge conflicts should not occur or be trivial to resolve.

We used GitHub as a way to have a remote repository, which allows for collaboration as well as maintaining a single source of truth for the project. 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.

We decided to use GitHub actions to support our CI processes. While we considered alternatives such as Jenkins we decided to use GitHub actions as it integrates seamlessly with our version control system and has extensive documentation that is intuitive to less experienced developers. While Jenkins is an older project with an extensive set of plugins and additional features, we concluded it would be more efficient to choose GitHub actions.

We placed an emphasis on using automated tools where possible across the project, particularly for processes such as version control, testing and CI as it is easy to verify that a task will be completed correctly by an automated system and we can reduce development time by performing repetitive tasks automatically.

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.

For team collaboration, we chose to use discord throughout the project. We selected this over alternatives such as Slack as all members of our team are familiar with the software, reducing the learning curve faced at the start of the project. Additionally, discord has support for features such as webhooks, which we used as part of our CI process.

The most popular choice of development environment among our team was Visual Studio Code, with one member choosing to use IntelliJ. We designed our project and build system to be IDE-agnostic, so the choice of IDE was largely up to the preferences of the individual team member.

Team Organisation

Project Lead, Implementation - Bertie

- Coordinating team (Arranging meetings)
- Overseeing that team members are completing their responsibilities
- Manage task board on github so that tasks aren't left out

Testing - Henry

- Oversee testing and ensure high coverage
- Write the test report
- Ensure all requirements are tested for

Source Control / Project structure, Implementation - Jacob

- Maintaining the github repo (branching and merging onto main branch)
- Make sure that tasks within the project are not missed
- Code reviews
- Contribute to weekly plans regarding implementation tasks

Deliverable Updates - Will

- Update the Risks & Requirements deliverables
- Oversee updates of other deliverables
- Ensure change management is tracked
- Contribute to the change report

Website, Testing - James

- Update the website and ensure all material is included before submission
- Work with Will & Bertie to ensure changes are recorded properly
- Implement tests and work with Henry to ensure code is testable

We chose to organise the team in this way to allow for clear definition of responsibilities, everyone should know who to approach first with a particular issue. This allows us to work more efficiently, by reducing time spent on delegating tasks. Roles were chosen based on our personal interests and skills to maximise participation and mitigate the risk of burnout. This approach is suitable for developing a small 2D game due to the relatively small scope of each part of the project, meaning one team member is able to lead each main area.

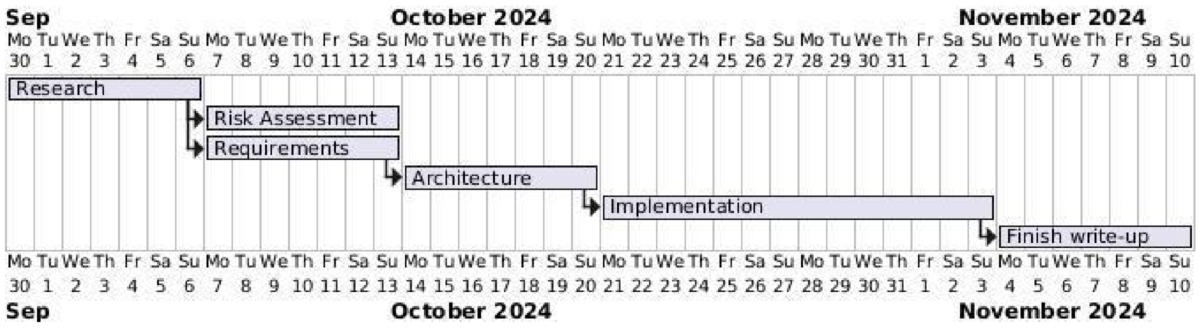
Project Plan

Week 1

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.

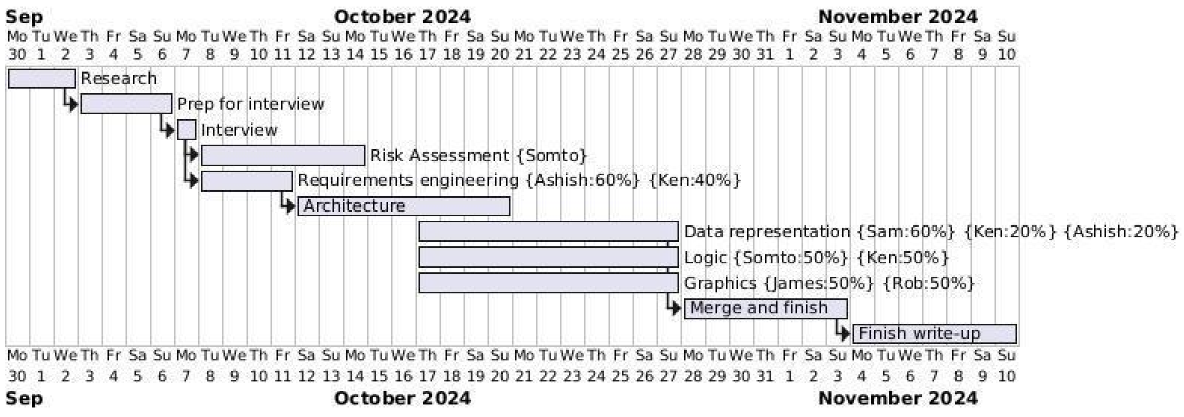
Week 2

By following the weeks assigned to each of the lecture slides, we can produce a rough plan.



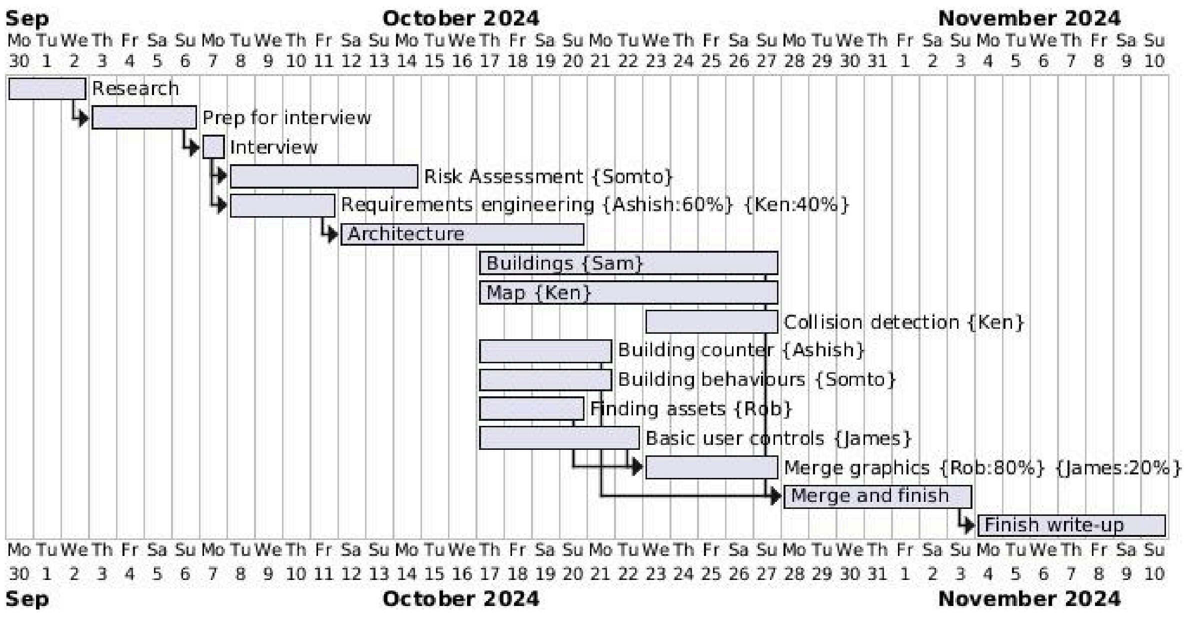
Week 3

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.



Week 4

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

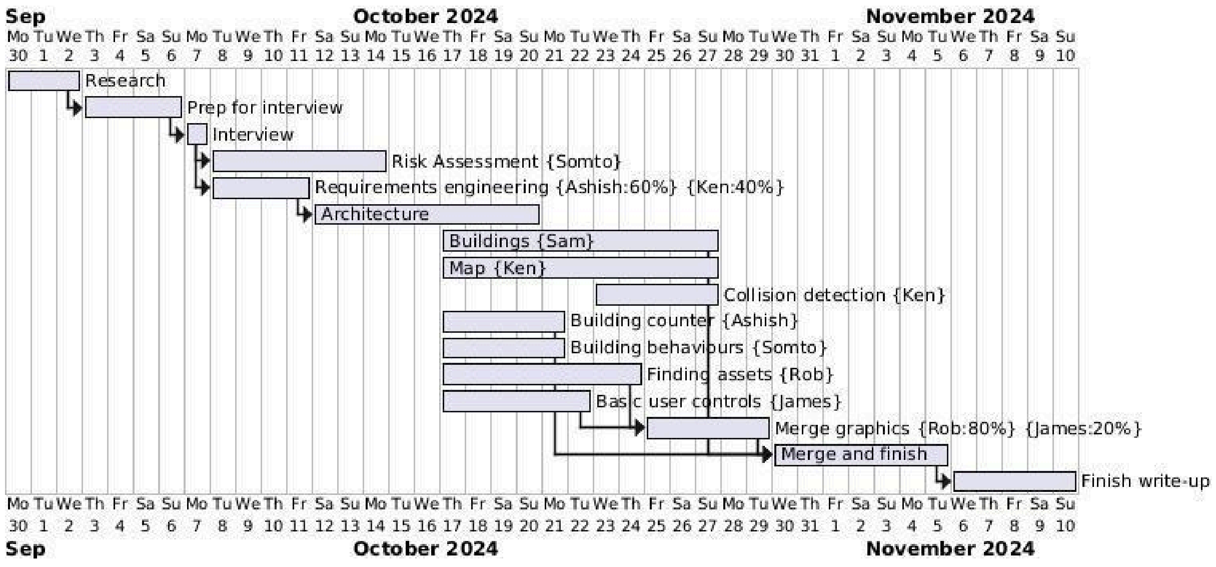
Finding assets took longer than expected which has pushed back merging however most of the rest of the plan has remained mostly the same owing to the lack of dependencies on other tasks in the implementation stage. The diagram is linked on the website (*Week5.png*).

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

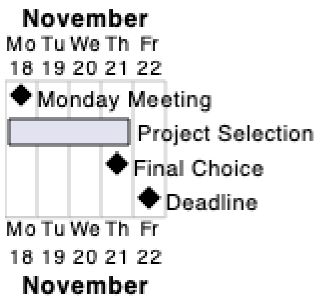
After reading week, we are on target and have merged most of the elements of the game together. With the team now focused on one task, it should be easier to remain on target.



Assessment 2

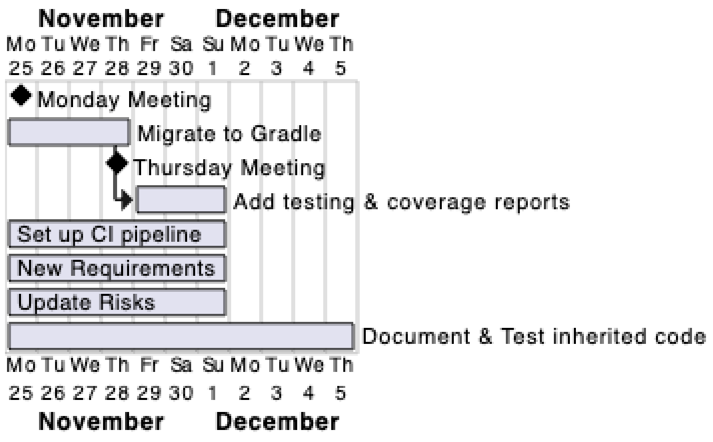
Week 8

This was the week of the team selection phase. In our Monday meeting we wrote a shortlist of the best projects to choose, and met again in our Thursday session to make a final decision.



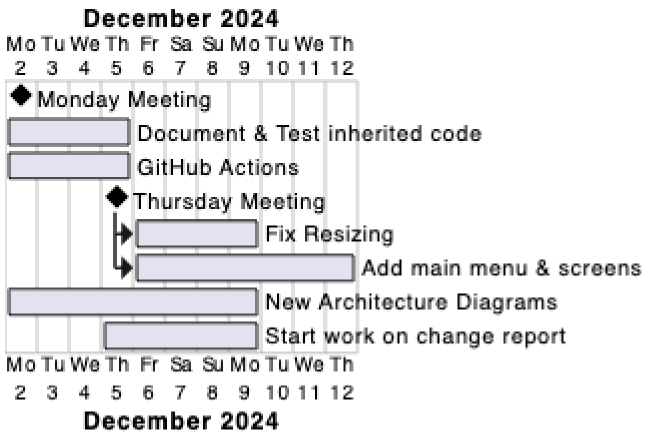
Week 9

After selecting team 11’s project, we started work on updating the deliverables and understanding the new codebase. We initially wanted to migrate to Gradle before setting up further CI tasks such as testing and coverage reports, however this proved too difficult - as a result we implemented the CI system with Maven as our build system.



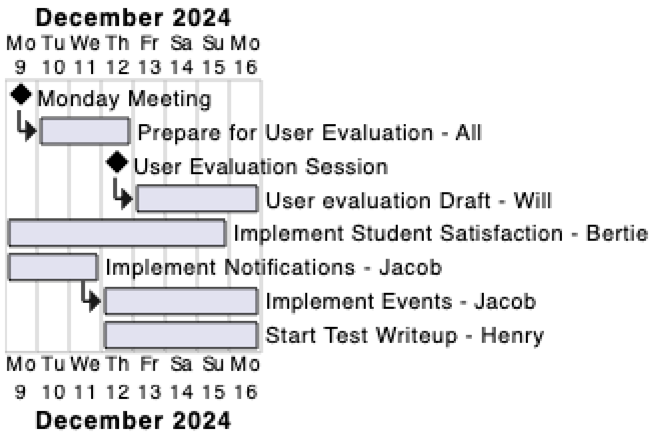
Week 10

Development is properly underway now, with focus split between the architecture, change report for the deliverables from last week, and the implementation of the additional features for assessment 2.



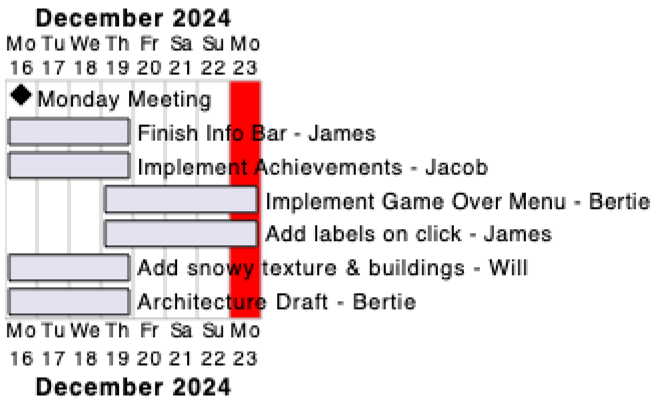
Week 11

During this week we prepared for the user evaluation session and continued with the implementation. We were slightly ahead of our plan with events, with these finished by the weekend.



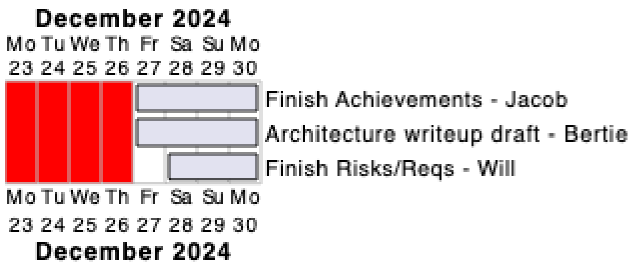
Week 12

This is the final week before taking a break over christmas, as such we wanted to get into a good position to resume development. We implemented the game over menu and achievements, then added some quality of life features based on our user evaluation results.

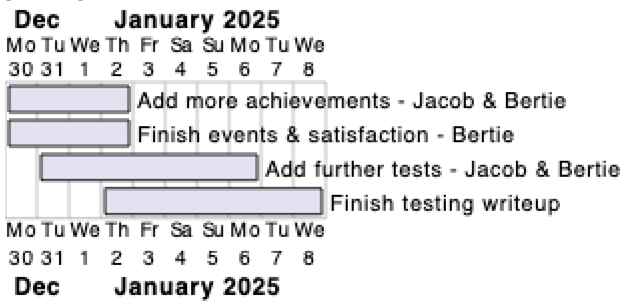


Week 13

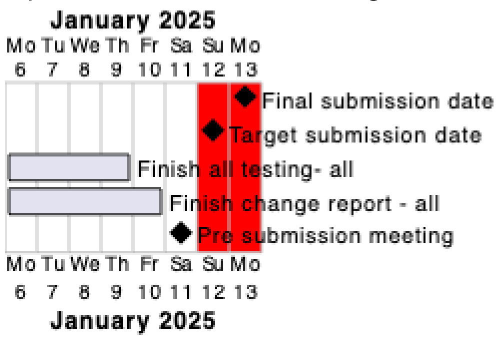
Very little of note over the Christmas week, we all agreed it would be best to take a break and work at a more relaxed pace over the new year, ready to finish things off in the last week before submission.



Week 14
Mostly focused on finishing off the key implementation tasks and getting the writeup sorted in preparation for the final goal. Fell slightly behind with achievements & tests but we were getting back on track towards the end of the week.



Week 15 (final week)
Final week before the submission deadline, with a goal of submitting the assignment on Sunday. Just a few final tasks remaining, including finishing off the drafts for the change report documents and adding the final tests.


January 2025
Mo Tu We Th Fr Sa Su Mo
Final submission date
Target submission date
Finish all testing- all
Finish change report - all
Pre submission meeting