Method Selection and Planning

James Smith
Amanda Ling
Fran Medland
Hannah Vas
James Kellett
Malik Tremain
Mischa Zaynchkovsky

1.1 Software Engineering Methods

Our team is adopting an agile approach to software development because of its flexible and adaptive nature. An iterative approach allows for regular reassessment of our actions in response to changing requirements and priorities.

By aligning our use of agile methodology with the timetabled practical sessions provided by the department, we . In order to have completed the deliverables by the six-week deadline, we treated each week as a ‘sprint’ and held meetings twice a week to discuss the progress and outcomes of that sprint.

In the first week, we held the project’s ‘kickoff meeting’, in which we addressed the briefing, discussed ideas and allocated resources, as detailed below in *‘2.1 Approach to Team Organisation’*. In this meeting, we discussed the sequence of tasks involved in each deliverable. A high-level view of our work breakdown structure can be found on our website under *‘Method Planning and Selection, Fig. X’,* and is elaborated on in section 3.1 of this document.

From here onwards, we will meet in-person on Tuesdays to check-in with the progress of the sprint, and on Fridays to define objectives for the next week’s sprint. Tasks from the work breakdown will be assigned to either individuals or sub-teams. A record of these meetings can be found on our website under *‘Method Planning and Selection - Meeting Minutes’.*

We decided to adapt and to try and work similarly to the previous group; we had a small kickoff meeting, where we discussed ideas, divided the workload, and set targets to meet. From this a work breakdown structure was created alongside an initial Gantt chart to outline our expected progress and to keep track of the timeline we had given ourselves. From then on we split into our smaller teams and assigned targets for our ‘sprints’ which we determined to be a week long, like the other team had. As we had our kickoff meeting on a Monday, the day that we could all meet, we decided that our ‘sprints would start on Monday, with a check in meeting on Fridays to stick to the bi-weekly team meetings.
The agile working method, which we had previously used and followed scrum to be able to check in with the whole team whilst also letting us use the benefits of splitting into our smaller groups.

1.2 Development and Collaboration Tools

We have chosen to use IntelliJ IDEA as our team’s IDE as we find it to have the most appealing, refined interface, as well as being customisable. Compared to other IDEs (such as Eclipse) IntelliJ has better support for GitHub desktop and Gradle builds, which are discussed below.

Originally, we explored the functionality of Eclipse - however, we found it was incompatible with GitHub desktop. When cloning a repository that has LibGDX imported, all of the classes are not recognised as a type; therefore, we chose to switch to IntelliJ to mitigate this issue.

To facilitate reliable version control for the duration of our project, we unanimously decided to use GitHub’s desktop application, which we installed onto our local machines and connected to our individual GitHub accounts. An alternative to GitHub desktop is the Git command line interface, or interacting with the repository directly through the IDE using Personal Access Tokens (PAT). However, these processes are cumbersome and time-consuming, which makes it difficult for all members of our team to follow the same process and avoid user errors that threaten the reliability of our version control system.

Consequently, we are using GitHub desktop to take advantage of its visual interface and user-friendly nature, which will be particularly beneficial to the members of our team who have no previous experience using Git.

Furthermore, our decision to employ LibGDX as our Java game development framework stems from its robust capabilities and intuitive implementation. Through our initial research, we concluded that LibGDX offers straightforward concepts, exemplified by the “render()” and “dispose()” methods that have clear functionality. Additionally, it is extremely well-documented and has a comprehensive Wiki page that boasts support from setting up a development environment to adding in-game music. Alongside LibGDX, we are using Gradle - a build automation tool for software development. This aids us in compiling, linking and packaging the code into a single application that runs on various operating systems. Gradle can be seamlessly integrated into IntelliJ IDEA with the help of extensions, which makes our adapted IDE an ideal environment in which to create our project.

For building the game map, Tiled appears to be the easiest software to use. Due to Tiled’s free and flexible interface, we decided to use it as our designated map editor and builder. There are many reasons that can justify why Tiled is suitable, starting with ease of use of its intuitive drag-and-drop functionality to build maps in sections. Tiled also proved to be very compatible with our project as LibGDX has a plethora of libraries to handle the .tmx file that maps are generated in. Overall, Tiled is robust, with countless useful features: multiple layers, custom properties, tileset animations, and object grouping, among others.

As a team we decided to use the same Collaboration and development tools, as this was how the game was originally developed and worked well, as well as it being familiar to the development team. The only addition is that we used a Discord server to communicate and hold online meetings as these were easier for us all to attend, allowing us to have a full team meeting and also splitting into smaller groups and communicating in specific circumstances with only the people who were working on the same part of the project. This helped streamline our communication allowing people to easily find help and solutions to problems along the way.

On further research of other collaborative tools, specifically if there was a good alternative to Git, as this doesn’t seem to have been looked into, we had a look at switching to YouTrack, due to the integration with IntelliJ, the IDE that we decided to use. It has similar capabilities to Git, and is from the same software suite as IntelliJ IDEA, so the integration between them may be better than the integration with Git. For the small scale and fast paced project that this is, the new unfamiliar software would pose a greater issue to learn compared to the integration benefits that it may give. Git also has a built-in testing functionality; which was useful for the unit testing done on the code during the project. Alongside this built in functionality we decided on using JUnit and JaCoCo for our unit testing, as it was the easiest and fastest way to include testing whilst still integrating well within the software that we had previously used.

2.1 Approach to Team Organisation

Our team chose to divide its members into two departments in order to accommodate the workload described in the project briefing.

The development team, made up of Kyla Kirilov, Ben Hayter-Dalgliesh, and Matthew Graham, is responsible for writing the code for our system. Managed by Kyla, this team is building the game from the elicited requirements, and meets regularly to negotiate and develop new ideas. The team works horizontally on different sections of the code, and collaborates with Kyla to integrate them into the system.

The documentation team, consisting of Hannah Thompson, Callum MacDonald and Chak Chiu Tsang, is assigned to write up the required documentation for the project. Each member of this team is responsible for maintaining and updating one or more of the deliverables, such that the division of marks is equal. For every deliverable, the team meets to review changes and evaluate the progress, with all members of the team adding their own relevant contributions.

The teams meet twice a week, both online and in-person, to discuss and demonstrate development and make necessary updates to the documentation, such as reviewing the risk assessment and auditing the requirements. Both teams display the progress they have made in that week, and collaborate to solve problems with the code or documentation.

This approach plays both to our individual strengths - ensuring that those more proficient in writing code or documentation are utilising their skills - and helps to avoid overcrowded collaboration on each deliverable. Similarly, it allows us to divide the workload evenly between all team members, taking into account the weighting of each task and deliverable.

Our approached team organisation was to assign everyone tasks, or assign smaller groups tasks to ensure that everything was completed in time. To do this at our kickoff meeting we spent a while equally dividing the work, this went faster than the previous time and everyone knew from the beginning what their specific tasks were, this ensured a far faster and more efficient start to the project. Most people had more than one task assigned to them, but to ensure the workload was manageable, the second of these tasks was typically reliant on a task that was set to be completed before their task would be, or was not reliant on anything at all. This helped reduce issues of someone having to wait on someone else's tasks to be complete before they could start theirs.

The main issue with this approach is that it can be difficult to know what everyone is up to at any one time, or keep up to date with any difficulties, to mitigate this we found that as part of our regular meetings a check in with how everyone was getting on and what they were focusing on. Another issue we found last time was that at the end bringing everyone's individual work together, and ensuring the project looked cohesive as a whole. Knowing this could be an issue we had all the documents stored in one location that was available to all, and held a long meeting to go through everything as a team before submission.

Due to this project having a clear split between documentation and the development of the game, we did split into 2 main focuses to be able to talk to others for help that are working on a similar part. This was mostly due to our division of work rather than a structured idea of splitting the team in half and the communication between people in our groups was more open rather than going through a specific

designated person, this seemed to work as everyone had a good idea of what everyone else was working on.

By having our bi-weekly meetings as a whole group, with a portion of the meeting at the beginning, being a check in of where everyone was up to, we managed to avoid waiting on the other half of the team to complete something by outlining what was needed for one group to be able to continue working and if that was not currently on track to complete, then plans would change to prioritise the portion of the work that would impact the other team members. This was a choice we made to ensure that everyone could always be working on something to do with the project.

3.1 Systematic Plan for the Project: Key Tasks and Dates

Our initial meeting produced a high-level work breakdown, which can be found on our website (*'Method Planning and Selection, Fig. 1'*). This is an overview of the components required for each deliverable, broken down into atomic tasks that can be completed by an individual or team. The deliverables, labelled D1-D6, correspond to section 3.3.1 of the ENG1 Team Assessment document. By taking each deliverable and breaking it into a series of smaller tasks, we were able to generate a systematic plan for the following weeks, taking into account the dependencies of each task.

From this diagram, we created an initial Gantt chart that lays out the key tasks and their ownership. This chart, which can be found on our website (*'Method Planning and Selection, Fig. 2'*) shows how the project will develop over the first week. Each week, we reviewed our progress during the previous week and updated the Gantt chart to plan for the next weeks' tasks. Weekly snapshots of the plan can be found on our website (*'Method Selection and Planning, Fig.2-6'*).

In week 1, we focused on setting up the collaborative tools discussed in the previous sections, as well as researching methods, assets, and techniques that would be useful for our project. We also created a prototype of the game, implementing only a sprite moving around a map. Due to issues with our IDE choice and integrating LibGDX, as well as waiting for a client meeting, we postponed implementation by a week to ensure that we were fully prepared to begin writing the code.

Our client meeting took place in week 3, which meant that our requirements could not be fully elicited until this point. Once the meeting was concluded, the requirements were defined and the process of writing the documentation was fully implemented. Therefore, during week 2, when we were unable to begin writing the code, the documentation team prepared the website, risk assessment, and architecture documents, while the development team prototyped simple modules and constructed the game map. Despite not being able to properly start the implementation, our development team was eager to become as comfortable as possible with the chosen IDE and game engine.

In week 3, the player animation and movement were completed, finalising the first prototype of the game. Some assets were selected this week, along with communicating concept designs for the game and its appearance. This week went very smoothly, and no issues arose.

In week 4, after the requirements elicitation had concluded, we held another meeting to plan the implementation and architecture going forward. The development team leader noticed that the pace of implementation was not up to speed and that certain individuals in the team were unsure of what needed to be done. To resolve this, we created a checklist of key features of the system, using the requirements as a starting point and decomposing them into smaller, more manageable tasks that included self-constructed deadlines. Since the workload at this point was heavy on development, the documentation team assisted by working on the code alongside the other deliverables. This meant that progress on documentation briefly stalled, but since there were dependencies on the code, this approach worked well for our team. This led to the completion of the map, energy bar, and activity counters.

In week 5, the project focused on completing the code, ensuring it met the requirements and that the system worked as expected from start to finish. This week, our aim was to complete map collisions, music/SFX, and all GUIs. Everything went according to plan, except for the loss of one GUI's code during the merging of code versions through GitHub Desktop. This setback was unexpected; however, it was quickly resolved by having one of the developers, who had already completed their tasks for the

week, redo the code. During this week, it appeared that the team member responsible for creating diagrams with PlantUML was struggling a bit, due to many of the diagrams being reliant on the code, which was only slowly coming together. This issue was addressed by the development team discussing plans for code that would specifically aid in the creation of diagrams. Overall, the architecture was more behind than other sections of the project, but this was simply resolved through some motivation.

Week 6 was when we made the finishing touches to everything, especially cleaning up the code and adding comments so that whoever may take over can understand everything clearly. Along with implementing a game end screen, a simple minigame to simulate studying was added, and we created our own eating sound effect for in-game use. After creating an executable for the game, the workload this week proved to be a bit too much for everyone. This didn't go according to plan, and we should've discussed how to even out the workload among everyone in the week leading up to submission.

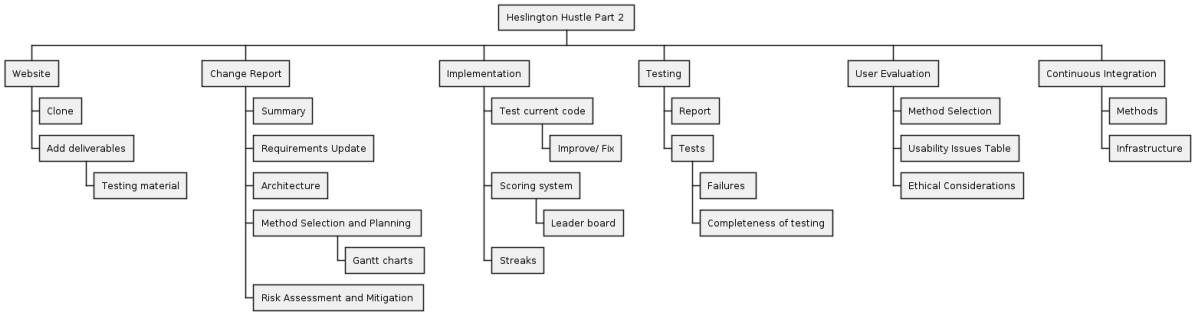
Finally all the deliverables, executables, and documentation were added to the website for submission.

After the kickoff meeting, we created an initial Gantt chart, this was to give us a plan to try and follow the key tasks, and labelled them with the ownership of that task, these could then be broken down into smaller subtasks for individuals to complete. After each weekly meeting these were then revised and updated to align with the new plans that we had discussed.
For simplicity, the main tasks were outlined first (labelled in the gantt charts as Light blue) and once that main task was about to be started, we then further outlined subtasks, this aligns with our agile method of working, as we decided to create a rough plan, however as we expect and will change our plans when and if needed, it seemed counterintuitive to fully outline tasks that clearly relied upon previous ones to be completed first, this also meant that at the beginning of each new main task the people who were assigned that task started by defining and assigning the new tasks from that, making adjustments as needed. This helped to keep everyone's number of tasks and things that they focused on at a manageable level at all times.

Systematic Project Plan Continued

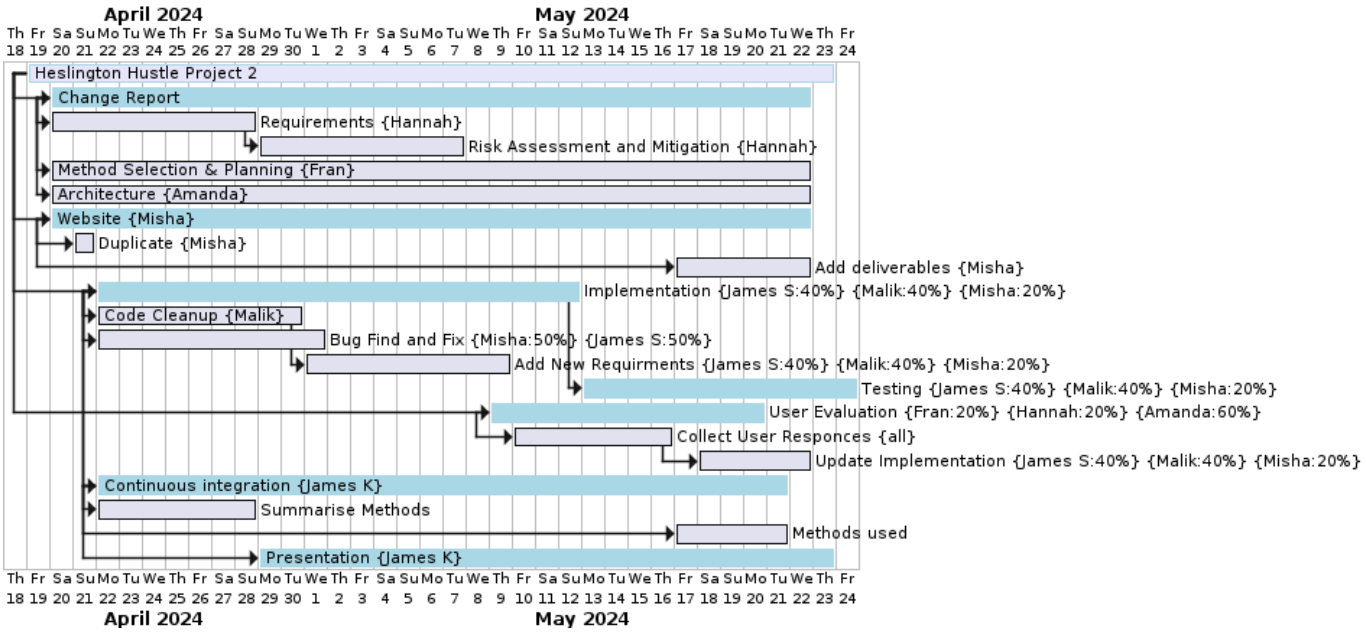
Kickoff Meeting

From our Kickoff meeting we created a Work breakdown structure to outline main tasks and key subtasks to complete



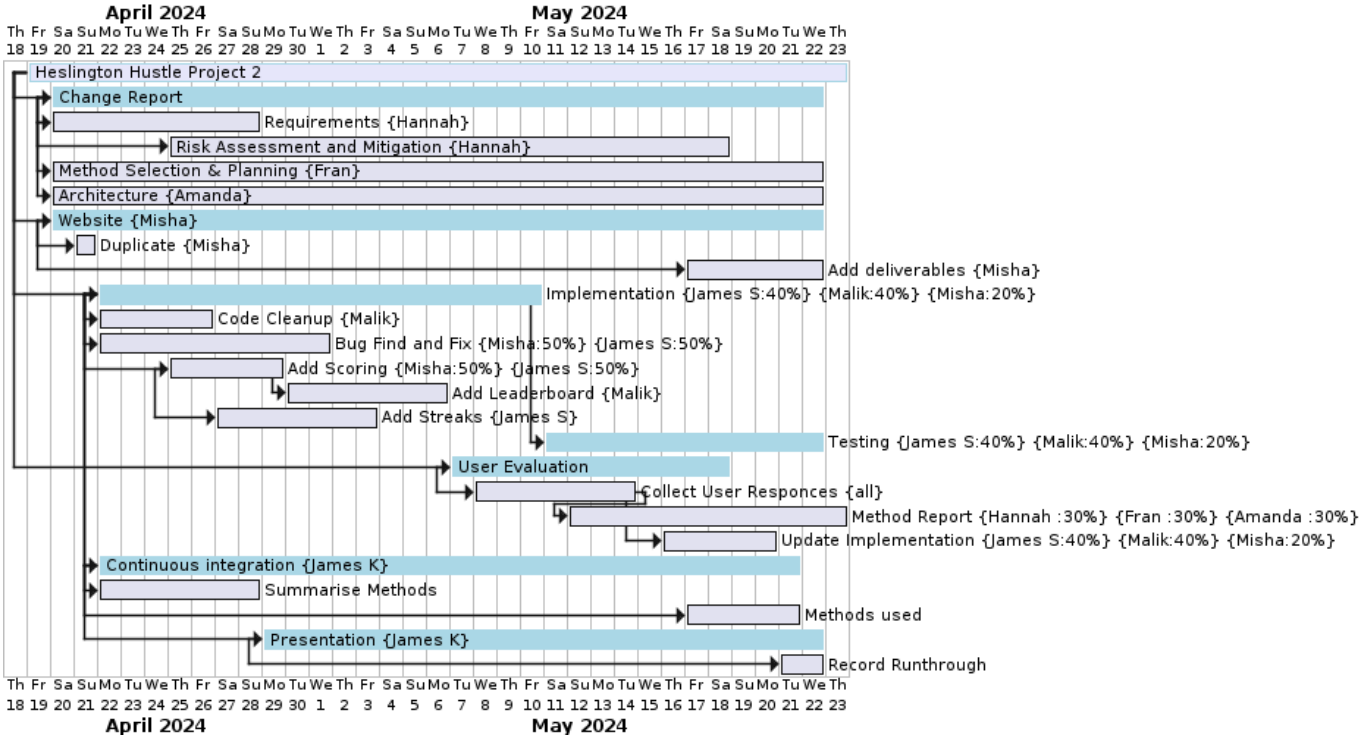
After this we assigned tasks based on the individual teams along with a rough timeline for the project to be completed in. The tasks for everyone before our Monday Meeting was to familiarise ourselves with our assigned tasks and get started on them if we were able to.

Week 7 (22/04/24-28/04/24)



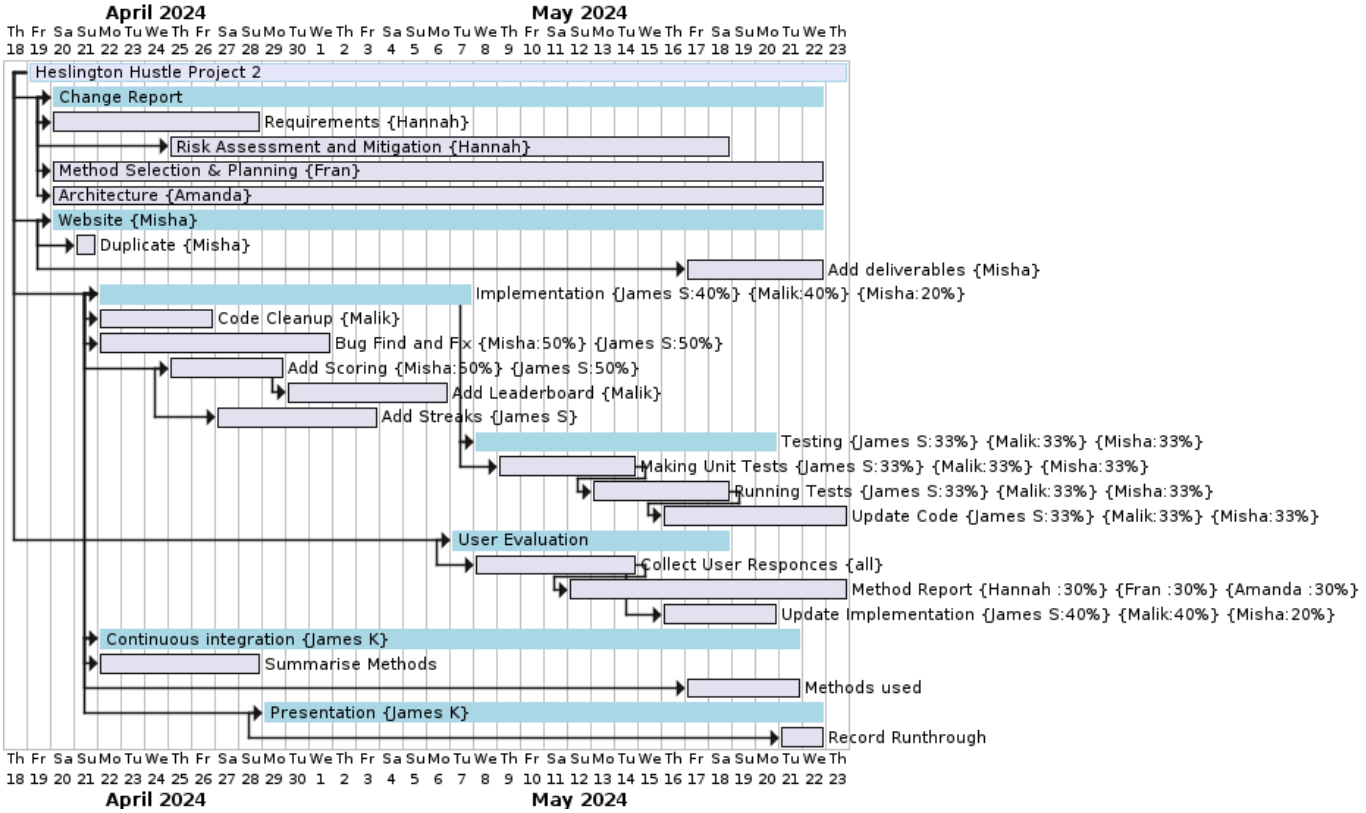
The main tasks to focus on was to understand the previous teams documentation and code, by spending some time doing this at the beginning (3 days) we hoped it would avoid confusion later on in the project. This was expected to take a couple days, and then focused on primarily updating the documentation and to start on the code, by testing to find and fix bugs and make it slightly more readable. After this had happened the implementation could start, which we aimed to get to by the end of the week. Alongside this, we aimed to update the documents that we could at the beginning and as continuously update the ones that could not be completed until nearer the end.

Week 8 (29/04 - 05/05)



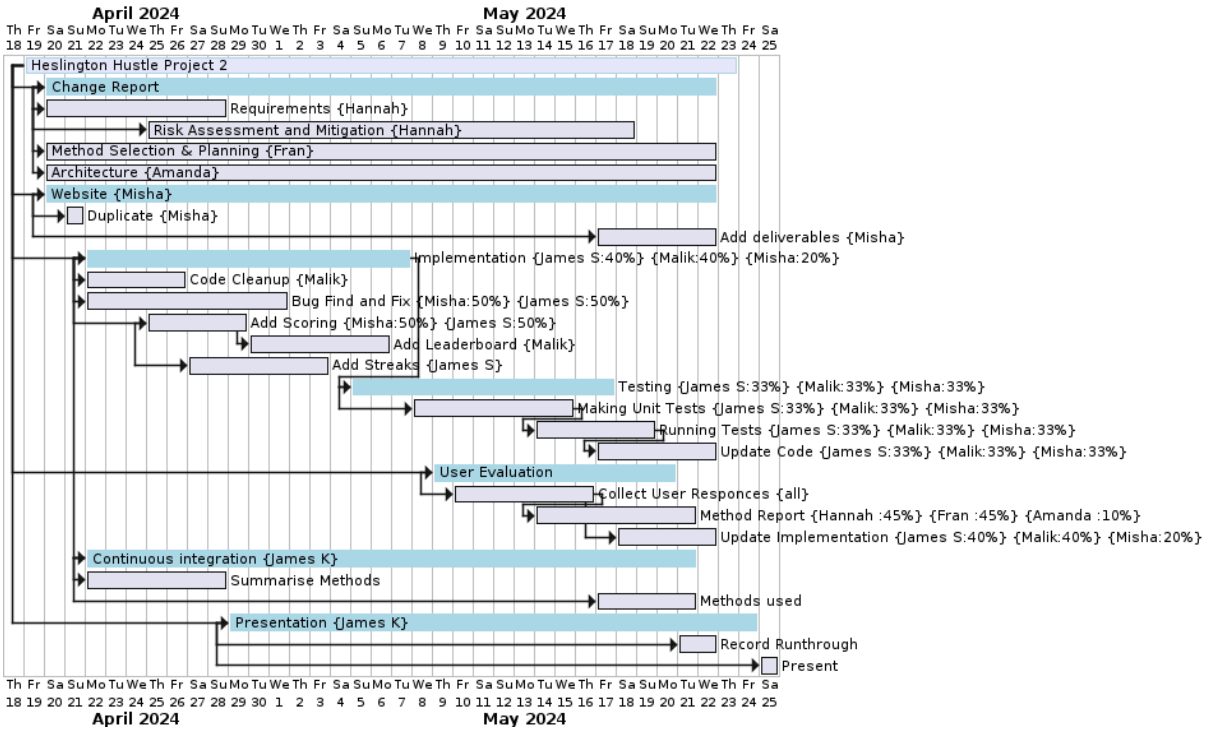
The code cleanup took a little less time than expected to be able to start on the implementation, which was also detailed in more specificity and divided amongst the people working on it, so that it could be started. This also meant that the timeline that we initially planned could be shortened to increase the amount of time spent testing the code afterwards, and other parts of the project do rely on the implementation being complete before they can be properly started. We also realised that similarly to the Method selection & planning and the architecture, the Risk assessment and mitigation document would need constant updating and revising throughout the project, so that was extended until the end.

Week 9 (06/05 - 12/05)



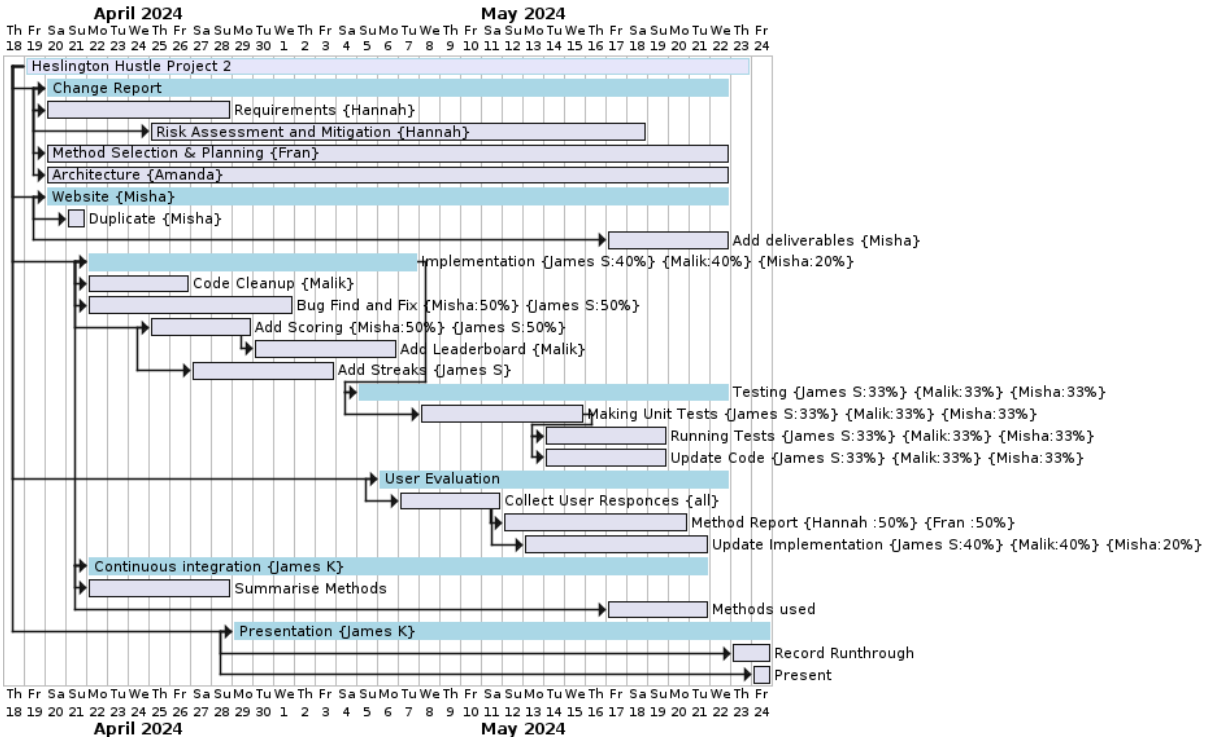
The implementation of the new requirements was completed, so whilst the unit testing was being started, it seemed beneficial to try and get started on the User testing, this would likely take a while as everyone has to find and be available to test the game with a person. After this we would need to analyse the results and start updating the implementation afterwards, with the implementation being shortened the User evaluation was brought forwards a little the previous week. This was a task that everyone had to collaborate and work together on with other people which made the timeline difficult to estimate; to mitigate this risk we tried to overestimate the length of time needed to complete, so that none would be held up by this being incomplete.

Week 10 (13/05 - 19/05)



As the main bulk of the implementation was now complete, with only small changes due to the user evaluation, which had been started to be implemented and the unit testing, the architecture diagrams could now be made and added. This allowed the documentation to stay on track and after a little miscommunication between the development team and documentation they were ready to add to the documents, by having our check in on the previous Friday we managed to see this problem and this allowed us to over the weekend prioritise the parts that let others continue their work, this did mean that the making of the unit tests and the starting of the running of the tests got slightly delayed, and took slightly longer than initially anticipated.

Week 11 (20/05 - 24/05)



The testing was extended due to what happened in the previous week on, this did mean that it ended alongside everything else, rather than a couple days earlier, this did mean that there wasn't much time for the writeup part, and the recording of a final runthrough was moved to the last day, this then was added to the presentation. To ensure all the code was done on time, we decided to work on updating the code alongside running the tests rather than after. Our Monday Meeting was a check in to see how everything was going, and reading through the Deliverables to ensure they were able to be added to the website, there were some small changes that needed to be added, before submission. As our typical Friday meeting would've been after submission, we moved this forward to have a final meeting to ensure that everything was ready to submit, but still leaving us enough time that if something needed to be altered or amended that that could happen. As we had planned for this throughout the week and the whole project, the final week was busy but not as hectic and was still a manageable amount of work for everyone.