Method Selection and Planning

Hannah Thompson
Kyla Kirilov
Ben Hayter-Dalgliesh
Matthew Graham
Callum MacDonald
Chak Chiu Tsang

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’.*

ADD A WORK BREAKDOWN - in progress as PlantUML file.

ADD DELIVERABLES

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 map for the game, Tiled appears to be the easiest software to use. Due to Tiled being a free and flexible level editor, this was enough for us to decide on Tiled being the designated map editor. Many reasons can justify why Tiled is the most suitable, starting with ease of use with Tile’s intuitive drag-and-drop functionality. Tiled is also proved to be very compatible with our project as LibGDX has a plethora of libraries to handle the .tmx file that Tiled provides its maps in. Overall, Tiled is robust with countless features like multiple layers, custom properties, tileset animations, and object grouping, among others.

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 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.

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-X'*).

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 in our project. We also made 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 pushed implementation back 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. During week 2, where 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.

After requirements elicitation had been concluded, we held another meeting to plan the implementation and architecture going forwards. 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. 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 stalled briefly, but since there were dependencies on the code, this approach worked well for our team.

Weeks 5 and 6 of the project focused on completing the code, making sure it fulfilled the requirements and that the system worked as expected from start to end.

NEED THE WEEK 4/5/6 SNAPSHOT FOR THIS PART.