Method Selection and Planning

Oliver Dixon
George Cranton
Praj Dethekar
Denys Sova
Shivan Ramharry
Albara Shoukri
Rafael Duarte

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’.*

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.

Following the handover, the usage of these tools (IntelliJ, GitHub, Gradle, and LibGDX) was continued given the coupling of the codebase to the ecosystem defined by Team 25. However, for the local management of version control, we unanimously decided in an early group meeting to prefer the integrated IntelliJ Git features over the GitHub Desktop program, due to our familiarity with the various processes from Assessment 1 work. The GitHub-specific features integrated into IntelliJ for managing Actions, Issues, and Pull Requests also proved incredibly useful for change-tracking/-management and continuous integration purposes.

Although we continued to use the Tiled software for graphical creation and editing of TMX tile-map files, we did not need to do any substantial work other than fixing minor issues with the inherited assets (see https://github.com/ENG1-Group-23/A2-implementation/issues/10 for an exemplar).

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.

Following the hand-over, the development team consisted of Prajwal, Oliver, Denys, and George, given their prior experience (such as with Assessment 1); their responsibilities consisted solely on the development and thorough documentation of the Implementation code and testing strategy. It was the unanimous decision of Team 23 that the Development and Documentation should be undertaken by the same Team, as the programmers have the greatest knowledge of the code to be documented.

Other Assignment 2 deliverables were managed principally by Shivan, Albara, and Rafael, although were often heavily assisted by the Development and Documentation team on an ad hoc basis where intimate knowledge of implementation details was necessary.

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.

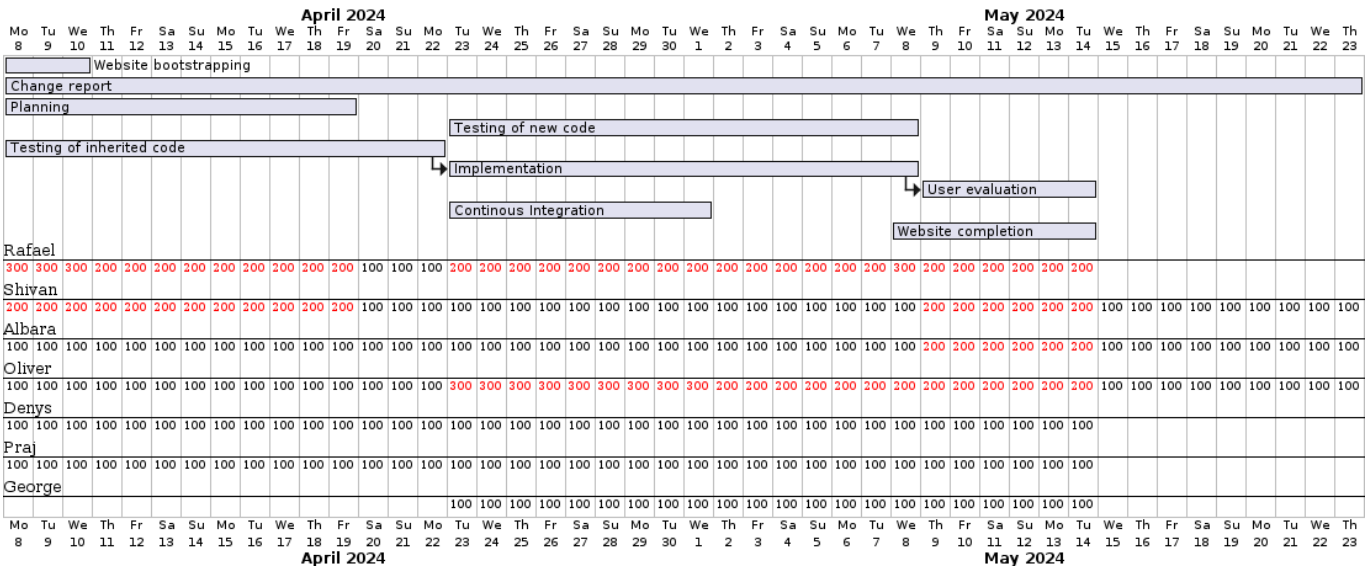
In week 5, the project focused on completing the code, ensuring it met the requirements and that the system worked as expected from start to finish. This week, our aim was to complete map collisions, music/SFX, and all GUIs. Everything went according to plan, except for the loss of one GUI's code during 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.

3.2 Gantt chart for Assessment 2 tasks:



On each day, the Gantt chart displays the utilisation of each team member: 100[%] indicates that the individual is assigned a singular task on the particular day; 200[%] indicates the assignment of two tasks; et cetera. Each task contains dependencies, which are listed below:

- *Website bootstrapping and completion:* the website can be bootstrapped at the first stage of the project, whereby the assigned team member (Rafael) clones the original website sources and prepares it for use by Team 23. The completion of the website is highly dependent upon all other deliverables, including the modification of Assessment 1 deliverables, as it is used as a hosting repository for each of the completed documents.
- *Planning:* Given the highly concurrent nature of the project (i.e., there are few task dependencies), the Planning section was not especially complex or demanding. A typical agile-like approach was used in discrete tasks, enabling the continual refinement of documentation throughout the entire Assessment 2 cycle.

- *Testing of inherited code:* Given that Team 25 did not write any automated testing, and did very limited documented manual testing, we invested a substantial amount of time into the construction of tests on the inherited code to ensure its suitability for fulfilling the requirements.
- *Implementation and testing of new code:* Once it was established that the inherited code was suitable to act as a basis for the Brownfield Development, the updated customer requirements could be considered for implementation. The development of a comprehensive unit-testing framework ran concurrently with the implementation of the required functionality.
- *Continuous Integration:* The CI component had very few dependencies beyond that of an initialised GitHub repository capable of despatching GitHub Actions requests to the build server. Oliver Dixon was able to complete this concurrently to the Implementation; continually testing the CI methods on early revisions of the revised codebase.
- *User evaluation:* The User Evaluation requires a full working game, hence could only occur after the Implementation was complete and ready to be shipped. At this stage, all feedback (coupled with the requisite consent forms and ethical procedure documentation) could be compiled and submitted to the ENG1 ethics controller (Christina) for review.