Method Selection and Planning

Cohort 1, Team 1:

Connor Blenkinsop
David Luncan
Emily Webb
Muhidin Muhidin(MO)
Sam Laljee
Sooyeon Lee

Method

When choosing a methodology to follow we focused on finding one that matched the size, time constraints and flexibility of the project. We researched and discussed both Plan-Driven and Agile Methods.

Plan-driven methods:

This traditional methodology requires detailed initial planning to guide the project and often has extensive documentation. It is a structured incremental method that tends to work better for longer term projects. We researched Ration Unified Process, a methodology that focuses on tools and OO techniques. It prioritises modelling software visually via UML.

Due to the amount of planning necessary, plan-driven methods need a stable environment with little to no change in requirements. We didn't choose this method as we knew our requirements would change as we reached assessment 2, and therefore we needed flexibility.

Agile Methods:

Agile software development is iterative meaning requirements and solutions evolve over the lifetime of the project. This makes agile projects adaptive and suitable in situations like ours where requirements might change. Because of this it is important to work closely with the customer, producing frequent deliveries of software. We chose to use an Agile method because they are successful for small teams and flexible in case of unforeseen change.

Feature-driven development (FDD):

We chose to use feature-driven development as the framework for our project. FDD is a structured agile methodology that focuses on breaking features down into simple and achievable goals that are then used to measure the progress of the project.

We compared FDD to Scrum, preferring it's focus on features rather than delivery. Using FDD would not require us to meet as frequently as Scrum, where teams typically meet every day, which was unnecessary and too time consuming for our purposes. FDD is more appropriate as important information is communicated through documentation. We chose not to use XP as we needed to be able to model our software architecture.

We compiled our features based on the customer meeting and product brief, and broke them down into the following:

- Game map
- Playable ship
    - Movement
    - Earn XP
    - Earn plunder
- Colleges
    - Combat
        - Fire at college
        - Increasing accuracy with proximity
        - Destroy or capture
- Shop
    - Spend plunder
- Instructions
- End screen

Tools

Communication:

We chose to use **Facebook Messenger** as our main method of communication. This allowed us to send links, discuss our availability and update each other on our progress between meetings. A group chat was preferential over private messaging as everyone was notified and kept in the loop. When we were unable to meet in person, for example over the winter break or when in isolation, we used **Zoom** to ensure we could still interact synchronously even when we weren't physically together. This meant we could communicate more effectively than over Messenger and keep each other accountable. We chose these tools because we all had access to and familiarity with them.

Documentation:

We decided **Google Drive** would be the best place to store our files as unlike an asynchronous service like Microsoft Word, we can all access and work on the files at the same time. Since Google Drive is cloud based, we were also less likely to lose our work. Another benefit is the ability to store all the different types of files we need including docs, sheets, and third-party tools. This is also where we kept our meeting log that kept a record of what was discussed so that anyone not present could catch up.

Implementation:

Following the product brief, the implementation was required to be written in Java. We considered various development frameworks such as Unity but ultimately, we chose to use **LibGDX** to implement this, as it is well-documented and the platform we had the most experience with. It is suitable for this project as it offers a vast range of features and is cross-platform, supporting MacOS, Linux and Windows – the operating systems we were required to cater for.

Version control:

GitHub is a cloud based decentralized version control system that prevents conflicts when multiple people are working on the same code due to the branching feature. This makes it suitable for collaborative development. We discussed using Apache SVN as it is easier to use however, GitHub offers more features and encourages branching. In addition to this it was already familiar to us as many of us had used GitHub before.

UML diagram:

We thought **draw.io** would be the most suitable tool for creating our architecture diagrams as it is compatible with Google Drive and has a free trial. Its built-in shape library keeps creating diagrams quick and simple.

Gantt chart:

Lucidchart is the tool we used to create our Gantt charts, giving us the rough timeline we hoped to follow. It allowed for more collaboration than draw.io as each member of the team could access the chart simultaneously.

Team Organisation

During our initial meetings, we tried to evaluate each of our strengths and past experiences to decide who should take on which roles. This involved taking personality tests which also helped us to get to know each other better.

We decided to have two main developers who would implement most of the game, with support from the rest of the team if needed. This was in order to keep everything simple and avoid having too many inputs and ideas complicating the process. It also meant smaller developers' meetings could be organized separate from the rest of the team if needed, that were not reliant on everyone being available. Connor and Muhidin were allocated as developers since they were the most confident in Java, having previous experience programming 2D games.

We also decided to have a lead software architect, Sam, who would work closely with the developers to design the game.

One of the early decisions we had to make was whether to appoint a team leader, responsible for allocating work and making sure everything was on track. A hierarchical system can be beneficial in a collaborative environment as someone is responsible for keeping others accountable. However, we found that our team functioned best with shared leadership. We assigned a person to take responsibility for each deliverable to ensure it was completed on time. This more democratic approach meant no one person was responsible for the entire project and decisions were made collectively with little conflict.

Our team met every Thursday during the ENG1 session with an optional Monday meeting if needed. We found that we were largely able to work on our sections separately, coming together to check in and ask any questions we had for each other.

Project Plan

Our higher level plan was first established on Aut/8, on our second meeting as a team. This was developed based on the table of deliverables, each of the 6 deliverables becoming a main task. We found that the mark breakdown for the individual deliverables to be a useful way to separate larger chunks of the project into smaller and more manageable, modular components. Our approach was to assign one or two people to each main task. This had the following aims:

1. Having 'section experts': since team members only need to focus on one or two sections, they have the time to better understand their respective area with greater depth
2. Concurrency: sections that do not depend on each other can be done simultaneously, and non-mission-critical tasks have the flexibility of being done in any order

This is an outline of the main tasks and their respective subtasks:

1. Website
2. Requirements
   a. Writeup
   b. Tables
3. Architecture
   a. Planning
   b. Writeup and Explanation
4. Method Selection and Planning
   a. Software choices
   b. Writeup and Explanation
   c. Project Plan
5. Risk Assessment and Mitigation
   a. Writeup
   b. Tables
6. Implementation
   a. Skeleton Implementation: movement
   b. Adding Colleges
   c. College Combat

The amount of time and number of people assigned for each of the sections was determined by their respective weight in marks. Since all these sections had a direct impact on the final result, our priority planning was not based on the perceived importance of any task but rather, its urgency. An outline of the initial plan, stating the 6 main tasks, their intended start date and latest possible end date, as well as their 'owners' can be see in the Gantt chart below:

	29/11/21	06/12/21	13/12/21	20/12/21	27/12/21	03/01/22	10/01/22	17/01/22	24/01/22
Website - 3								🟫	
Requirements - 20	🟧	🟧	🟨	🟨		🟨			
Architecture - 22		🟦	🟦	🟦					
Method selection and planning - 10		🟩	🟩	🟩		🟩			
Risk assessment and mitigation - 10		🟪	🟪	🟪		🟪			
Implementation - 25				🟫	🟫	🟫	🟫	🟫	
				🟥	🟥	🟥	🟥	🟥	

Method Selection and Planning - Eng1 Team 1

Legend:												
	🟥	Connor	🟧	David	🟦	Sam	🟩	Emily	🟪	Soo	🟫	Mo

This was only a rough outline, with the purpose of establishing a critical path: the sequence of dependent tasks (one cannot be started before the one before is completed) that amounts to the quickest possible time of completion. The series of events in our critical path was Requirements -> Method Selection [choosing appropriate software] ->Architecture [basic planning phase] -> Implementation -> Website Completion.

Our main obstacle when planning this project was everyone's availability over the winter break. As time was of the essence, and the dependencies laid out above had to be respected, this would eventually play part in what task each member took on. For example, David would be away a lot during the holiday, so his main area of responsibility were Requirements - the first piece in the critical path, while Connor had availability over a large period of time, so implementation was handed to him. Role assignment was based on a mixture of preference in regards to both the type of work, and availability, to create a system in which everyone is both happy and comfortable with what they are doing.

Looking ahead, we also planned for when inevitably, we have misjudged time allocations.. Having a 22 mark task being assigned to only one person may seem potentially excessive and unfair. To plan for this, and also any other problems regarding people having too much or too little work, we put in place three tools for a more efficient and dynamic collaboration: firstly, as some tasks are smaller than others, and also some are scheduled to be completed before others, extra 'manpower' will always be available with at least one other able to jump on an assist on a task. We wanted to do it this way so that we don't overcrowd any areas with too many people - stepping on each other's toes - but while still ensuring work is evenly distributed. Secondly, as seen by the lighter 'blocks' in our chart above, we scheduled in buffer times, to allow the project to still be completed on time even if certain sections were to overrun. To add to this flexibility, for each of the more written based tasks, we allowed another 'optional' potential week after the section was finished, just to focus on formatting and displaying the information correctly as per the subtasks stated in the Product Brief. Lastly, we also planned to end a week and a half before the actual deadline so that the project is safely completed.

This flexibility proved to be a crucial product of our planning, that we relied on heavily especially in the weeks of 10/01 and 17/01. While the writeup tasks were already coming along nicely before the start of the winter holidays, the architecture and implementation - the bulk of the marks - had yet to be started. Due to all members being away for more time than anticipated, and then the winter exam period in the week of 03/01, this hadn't advanced by our first meeting of the new year, on Monday 10th of Jan.

In this meeting we realised how soon the previously very distant feeling deadline really was. This proved to be a great motivating factor especially in the department of Implementation. By the week of 17/01 most of the writeups had been fully completed and more hands were available in case the implementation needed more help. This was, in the end, not necessary. The only difference versus the plan chartered above was that the one free week left blank, ended up taking over the weeks either side of it - this also meant that the project was only 'officially' finished on a meeting on Monday 31st. In retrospect, more detailed planning of the implementation, with milestones and deadline dates for subtasks may have prevented the slight rush in the week's before.