Method selection and planning

Assessment 1

Engineering Methods

Engineering Methodology

We chose to use an agile software engineering methodology as this allowed us the most flexibility as the project is completed and allows for individuals to dynamically contribute to the progress of the system. The advantages of using agile are quick deployment, emphasis on collaboration and the ability to adapt to changing requirements. Quick deployment is important due to the short time scale of the project, collaboration is important to ensure that skills from all members of the team are being utilised to maximum effect and the ability to adapt as requirements change is useful as the features that we choose to implement may change as the project unfolds. The disadvantages of using agile are a smaller emphasis on testing, smaller emphasis on documentation and risk of burn-out. The smaller emphasis on testing can be mitigated by focusing on code quality and adhering to style guides and best practices. Burn-out can be reduced by allocating work based on team strengths, varying tasks that members do and working in pairs on sprints. These factors make agile the most suitable methodology for our project.

One of the alternatives that we considered was a waterfall methodology. An advantage of a waterfall approach is that it is easy to use and manage. It also provides definitive structure to each phase of the project and a greater level of documentation is required for each phase. However, changes to requirements cannot be easily accommodated due to the linear nature of the waterfall model, software development takes a back seat until requirements and design phases are completed and gathering accurate requirements before any prototypes have been made can be difficult. These disadvantages lead to waterfall being less suitable for our project.

Source Control

GitHub - We chose github for version control as it allows us to remotely collaborate with each other. It allows individual team members to pull code from a centralised repository to work on a section of code before pushing it to a side branch to be reviewed and merged with the main branch of code. This allows for developers to work on code without affecting the work of the other team members and code can be verified before committing it to the main branch, saving time by minimising errors in code.

We also use GitHub projects to keep track of progress throughout the development stage. This allows us to break down large tasks into issues, which can be assigned to individual team members and integrated with pull requests, which aligns with our agile development methodology.

An alternative that we considered for source control is Azure. Advantages of Azure are its integration with the Microsoft ecosystem including the VSCode IDE, which is a popular choice among developers. However as we are using Google Drive for some of our documentation and many of our team members are already familiar with GitHub that is the choice we went with.

Development Environment

We chose to use VSCode and IntelliJ in the implementation of our project. One reason for this is that both IDEs have support for Java compilation & debugging and version control through GitHub. IntelliJ has this built-in while VSCode has a selection of extensions that can be installed. As LibGDX projects generally use their own build scripts, the project itself is mostly agnostic to the IDE used, meaning group members can use their personal preference.

Team Communication

When deciding between software for team communication, the two popular options among our group were Slack and Discord. We chose to use Discord for this project as it is software every member of the group is familiar with. Additionally, we were able to use webhooks to create a channel that tracks GitHub activity - this is useful to see at a glance when issues are created or closed off, and when code reviews have been requested.

Frameworks

LibGDX - We chose libGDX as the framework for our game development. It comes with ample features for developing a 2D game. It is widely used and comes with extensive documentation, which means we can get started with a simple setup very quickly. Additionally, LibGDX is Apache 2 licensed, a highly permissive licence that allows the use of their code for any projects with no fees.

Team Organisation

Project Lead - Erin

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

Documentation - Erin/Sam

- Ensuring code documentation is complete
- Ensuring that documentation will allow new team to take over project
- Create developer README so that developers understand the source code
- Project documentation

Source Control / Project structure - Erin/Sam/Aidan

- 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
- Continuous infrastructure setup

Writeup - Aidan/Ryan/Zuhur/Kevin

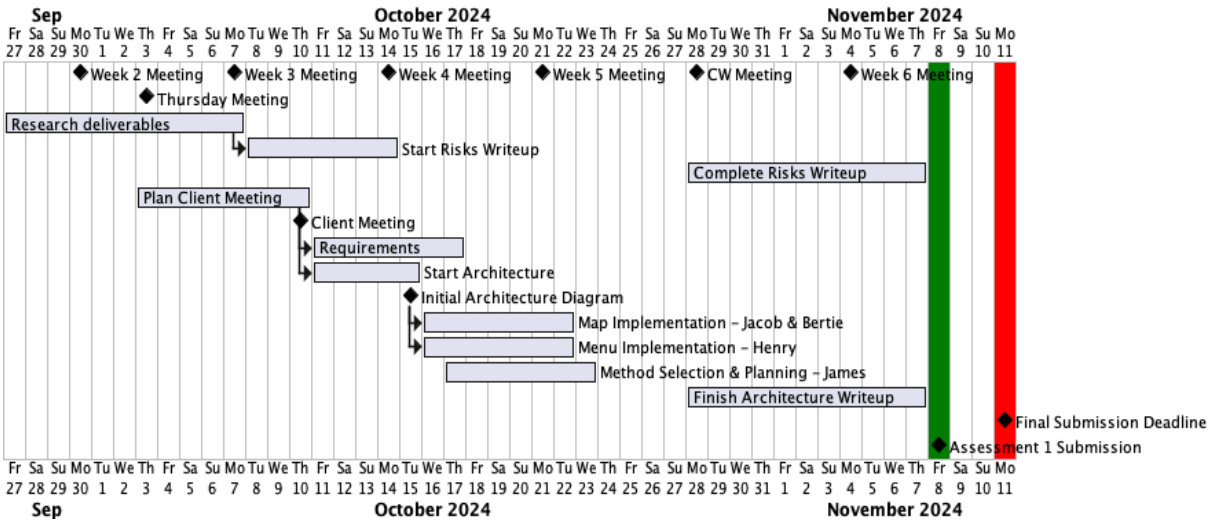
- Making sure the write up is complete
- Summarise team decisions to add to write up
- Proofreading
- Ensure that any methods/programs used have been added to the write up

Quality Assurance - Jack

- Ensure that code is high quality and readable
- Debugging code
- Ensure that game is fun and engaging
- Document known issues with code
- Creating automated unit tests
- Running manual tests

We chose to organise the team in this way to allow for clear definition of responsibilities with only small overlaps to avoid confusion as to who is responsible for certain tasks. 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 but have other members helping them.

Plan for Key Tasks and Deadlines (Assessment 1)



Note: Task - priority (Assignee)

Research Deliverables - High (All)
It is necessary for all members of the group to understand the deliverables to ensure that everyone is able to complete their sections correctly.
Plan Client Meeting - Medium (All)
Dependencies: Research Deliverables
The client meeting is to be planned to ensure that we use the time effectively and are able to elicit requirements from the client.
Start Risk Writeup - Medium (William)
Dependencies: Research Deliverables, Plan Client Meeting
The risk writeup will allow us to mitigate risk of failure throughout the project so that we ensure our success
Client Meeting - High (James)
Dependencies: Client Meeting Plan
The client meeting provides clarification on any confusions we had on the requirements and make sure whether a feature is necessary
Elicit Requirements - High (Will)
Dependencies: Client Meeting
Requirements write up allow us to manage requirements made by the clients thoroughly. As every demand is described in detail and assigned a priority, it allows us to easily plan and assign requirements to different teammates based on their strengths and preferences.
Start Architecture Design - High (Jacob & Bertie)
Dependencies: Elicit Requirements
The architecture plan allows us to decide on the themes of the game and the basic structure of the code. We focused on constructing a modulable structure as it provides easy modifications of codes and features, and allows team members to work on different parts of the code without a thorough knowledge of unrelated sections.

Create initial architecture diagram - High (Bertie)
Dependencies: Start Architecture Design
This will help us start work on implementation as if we don't have an architecture diagram we do not know how each class of our project should interact with each other. Without this, classes written by different members may not interact correctly.

Map implementation - High (Jacob & Bertie)
Dependencies: Create initial architecture diagram
The map will be made first as it is fundamental to the game and will contain all other entities that are added to the map.
Menu Implementation - High (Henry)
Dependencies: Map Implementation
The menu will be made after the map so that it is able to call for the map to be rendered once the play button is pressed
Method Selection & Planning - (James)
Dependencies: Elicit Requirements, Menu Implementation, Map Implementation
The method selection and planning stage of the write up will be left until the end of the project as we will be using github projects and google drive to keep track of team efforts
Complete Risks Write Up - High (Will)
Dependencies: Start risk write up
The risk writeup is to be completed at the end of the project as we want to be able to update the risk assessment if any issues come to light which were not previously anticipated.
Finish Architecture Write Up - High (Bertie)
Dependencies: Create initial architecture diagram
The architecture writeup will be completed at the end of the project as there may be tweaks made to method use and the way in which different classes may interact with each other.

Note: Due to the nature of these tasks and progress being dependent on completion of previous tasks, priorities are relatively high for all of these key tasks.

WEEKLY PLANS

How the plan changed throughout the project

Around 1 week into the implementation phase, Bertie joined the implementation side of the project to assist with writing the rendering code due to having previous experience with similar projects. This enabled us to move forward more quickly with development, meaning we would have sufficient time to prioritise code quality & documentation towards the end of the project. This led to Will taking over some of the writeup roles, particularly on the risks section to compensate for Bertie spending more time on implementation.

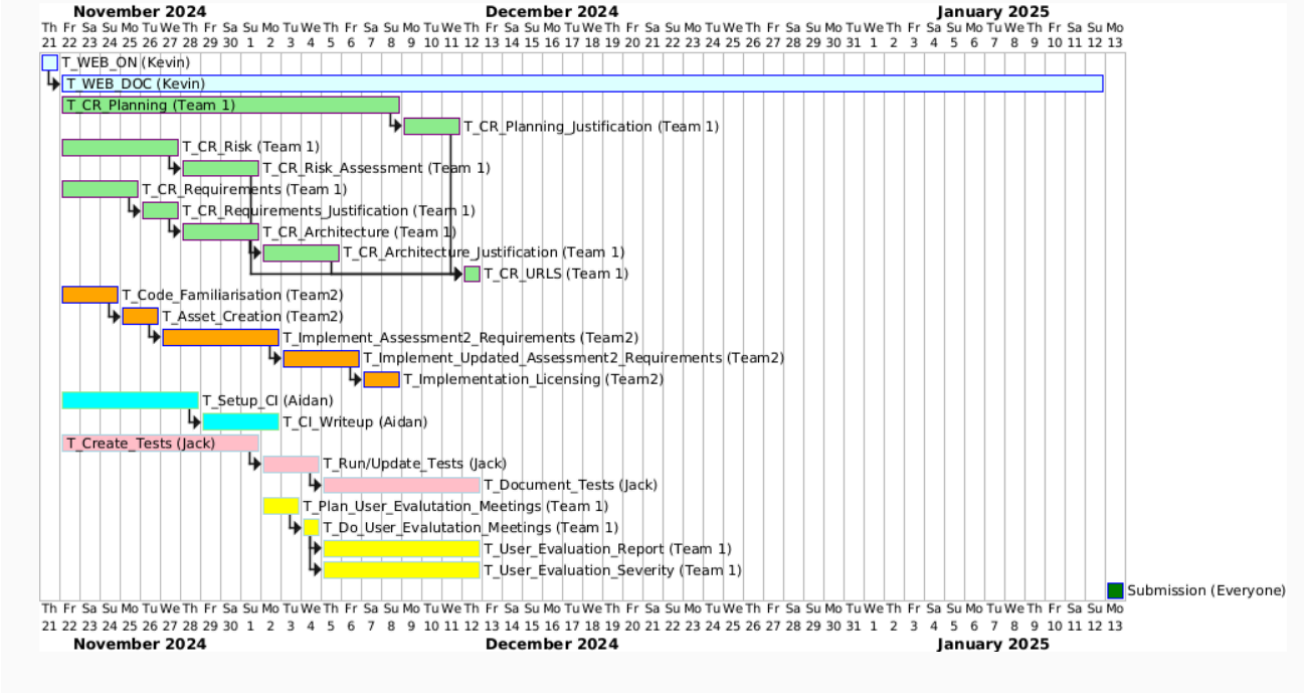
Overall, our efforts shifted towards implementation as the project progressed, with the most combined focus being towards the middle of the project - around weeks 4-5 and into Consolidation Week. It became clear that implementation would require significant effort to build a high quality product that can be easily extended and developed later. Towards the end of the project, Jacob took on a more supervisory role relating to development, with a particular focus on ensuring our code is compliant with javadoc conventions and the Google style guide.

Assessment 2

Plan for key tasks and Deadlines

The initial step in our planning process involved organising the project's deliverables into structured work packages, which we could then further break down into smaller, more manageable tasks to ensure efficient execution. To provide a visual representation and timeline of these tasks, we compiled them onto a Gantt chart, where we assigned dependencies and realistic deadlines to allow us to track progress effectively. To streamline our efforts, we split into two distinct groups, making each group responsible for specific work packages based on their priority level, as well as their complexity and the time allocated for their completion. Each group was also made responsible for dividing the workload equally among their group, ensuring a fair division of work for each task or section. Furthermore, some individuals were assigned their own smaller tasks to complete and overall this distribution strategy aimed to prevent overcrowding on smaller tasks, whilst ensuring an equal workload and good coverage on some of the more complex tasks.

Task dependencies are illustrated in the Gantt charts using arrows that connect one task to the next, indicating the sequence in which they must be completed for a successful end product. These arrows clearly demonstrate the relationships between certain tasks and show which need to be completed at a higher priority level, in order for several later tasks to be completed. As the project evolved, several iterations of these Gantt charts were created, each refining and making adjustments to the timeline when necessary. This resulted in the final Gantt chart shown below, as well as previous iterations that are visible on our website (**link**)



Team 1 - Aidan, Ryan, Zuhur, Kevin
Team 2 - Erin, Sam

How the plan changed throughout:

Towards the end of the second week of Assessment 2, we noticed a significant shift in the timeline of our project. The continuous integration aspect of the project, which we had allotted an extra week's worth of time to, was completed far quicker than expected. This unexpected change created some breathing room in our schedule but this was soon balanced out by some changes and delays in the implementation phase. The new requirements added to the project brief meant that the implementation process became more complex and time consuming than initially expected before the architecture and requirements had been completed. As a result, the implementation stage was going to extend beyond the deadlines we originally set, causing a knock on effect on subsequent tasks such as the user evaluation meetings. Because of these developments, we made the decision to adjust our timeline, as well as shifting some of our workforce from continuous integration to the physical implementation.

Below is the final Gantt chart after all of our adjustments had been made.