Assessment 2: Brownfield Development - Method Selection and Planning

Cohort 1 Team 2

Team Members:

Trace Chinelle
Vidhi Chohan
Apollo Cowan
Siyuan Liu
Aryaman Marathe
Charlie Mason

Engineering Methods - Greenfield Development

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 led 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.

Engineering Methods - Brownfield Development

Engineering Methodology

We chose to use an agile methodology for this project for a few reasons. This methodology granted us a lot of flexibility which allowed group members to help each other more easily, which could reduce the potential for delays. The focus on collaboration and adaptability in this approach allowed team members to play to their strengths. Another advantage of agile is quick deployment which we knew would be very helpful for the short timeframe, especially as we expected work to decrease massively over the vacation period.

Although agile does have some disadvantages, such as the risk of burn out, we knew we would be able to combat this easily with regular check-ins with all members of the team. We considered the waterfall approach but eventually decided it would be unwise to use a restrictive methodology with such a short time frame. We were all familiar with the agile approach and knew that it was appropriate in this context because it had worked for the team we inherited the project from.

Source Control

We continued to use GitHub as the project's version control because we were all familiar with its features. We knew it would allow us to create different branches from the central repository so that different team members could tackle separate aspects of the project at the same time, before it is reviewed and then merged to the main branch of code, which complements our agile approach . Because of this, it made sense to keep the source control the same.

Development Environment and Framework

The project already had build scripts built in so team members were able to choose any IDE they wanted. Despite this, all members of the team ended up using IntelliJ because of its built-in compilation and debugging tools. This worked to our advantage as we found that it was much easier to help each other when we were all using the same IDE.

We continued to use LibGDX for the project's framework since it was already built into the project. It is very popular and therefore there are lots of resources available as well as extensive documentation.

Team Communication and Collaboration

We used Google Drive for document collaboration because Google Docs allows for real time editing and easy accessibility throughout the team. We already knew that it would be appropriate for use in this project since we used it in the Greenfield phase. We briefly considered also using Git for our non code documents as well but ultimately decided against this because Google Drive is much more user friendly for frequent use.

We used WhatsApp to communicate with each other mainly because we all already had it downloaded on our phones and had each other's contacts on there. We were able to communicate with each other much more easily than if we used an app like Discord or Slack because we checked it for messages frequently already.

Team Organisation - Greenfield Development

Project Lead - Bertie

- 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 - Henry

- 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 - Jacob

- 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

Writeup - Will

- 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 - James

- Ensure that code is high quality and readable
- Debugging code
- Ensure that game is fun and engaging
- Document known issues with code

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.

Team Organisation - Brownfield Development

1. Team Structure and Division of Work

To divide the work among our team, we first reviewed the deliverables together, considering the timeline that would be needed to complete them. Then we assigned tasks to each team member, outlining the responsibilities of each individual clearly in tables, which could be updated as we progressed. We opted to allow team members flexibility to move between tasks based on availability and interest, rather than rigid roles. Some deliverables were split between several group members such as Implementation that was completed by a sub-team of 4. Other deliverables, such as each section of the change report were assigned to a single person.

2. Justification for the Team's Organisational Approach

Using tables for task division ensures an even workload and clear progress tracking, that also facilitates the equitable distribution of tasks. Furthermore, by considering the project timeline before we assigned all of the tasks, we would be able to ensure that everyone would have a workload spread relatively evenly throughout the duration of the project.

Allowing flexibility would make it easier for team members to shift their focus to support each other where necessary without the worry that they would not be able to balance their workload because of it, reducing the likelihood of delays.This also made it so we could change the roles if needed as the project goes on, as members of the team could choose to do tasks that suited their preferences.

When dividing work we accounted for the benefits of a collaborative or individual approach for each task, aware that during the vacation period collaborative work would be very difficult. This is why we shared the responsibility of the implementation between 4 people, knowing it would be easy to break down into smaller tasks. This allowed them to claim these smaller tasks at their convenience, improving productivity overall, especially over the vacation period. Conversely most of the written deliverables were assigned to a single person so that they could plan and write consistently without having to keep anyone else updated on the finer details.

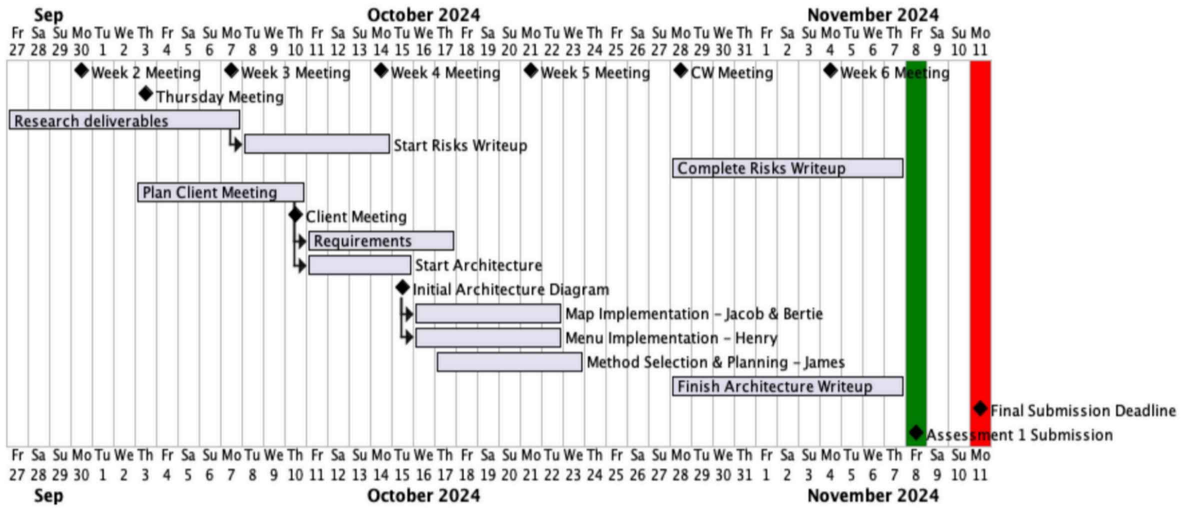
3. Coordination and Communication Practices

During the semester time we met twice weekly when possible so everyone could update the team on their progress. We were also able to check on the status of each deliverable regularly, using shared tables in Google Docs and updating each other on new developments in a group chat. This structure allows each member to stay informed on the overall project status while also focusing on their section.

4. Effectiveness of the Approach

Our approach supports accountability and independence while still gaining the benefits of collaborative work where possible. Assigning our tasks around the project timeline reduces the risk of team members getting overwhelmed by large amounts of work all at once.

Plan for Key Tasks and Deadlines - Greenfield Development



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.

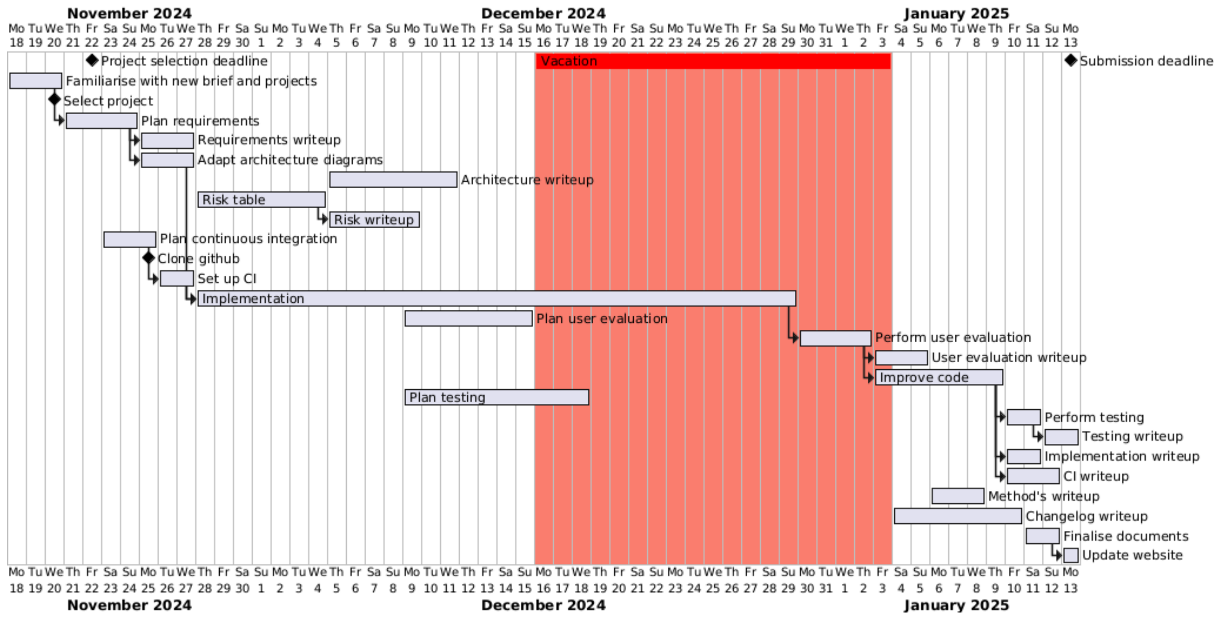
WEEKLY PLANS

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.

Plan for Key Tasks and Deadlines - Brownfield Development



Note: Task - priority (Assignee)

Familiarise with New Brief and Projects - High (All)
It is necessary for all members of the group to understand the deliverables to divide the work equitably among the team members. All team members also needed to review the available projects to choose to extend to make the most informed choice.

Select Project - High (All)

Dependencies: Familiarise with New Brief and Projects

All team members would be affected by the project selected so it needed to be a whole team decision.

Plan Continuous Integration - High (Charlie, Tracey)

Dependencies: Familiarise with New Brief and Projects, Select Project

It is necessary to plan the continuous integration approach before implementation can begin.

Plan Requirements - Medium (Apollo)

Dependencies: Familiarise with new brief and projects, Select project

Planning the requirements at this point will allow us to start updating the architecture without spending too much time writing the full requirements deliverable.

Requirements Writeup - Low (Apollo)

Dependencies: Plan requirements

The requirements deliverable can be written fully at this point since implementation cannot begin until the architecture is updated.

Adapt Architecture Diagrams - High (Charlie)

Dependencies: Plan requirements

Creating updated architecture diagrams will advise the structure of the program for implementation including how classes interact.

Clone GitHub - High (Charlie)
Dependencies: Select Project
A private copy of the chosen team's GitHub repository is needed to start making changes to the code
Set up Continuous Integration - High (Charlie, Tracey)
Dependencies: Plan Continuous Integration, Clone GitHub
The frameworks for continuous integration need to be set up before implementation can begin
Amend Risk Table - High (Siyuan, Charlie)
Dependencies: Familiarise with new brief and projects, Select project
It is necessary to amend this table as soon as possible to make sure the avoidance and mitigation plans are still relevant in case any risks occur.
Implementation - High (Apollo, Vidhi, Tracey, Siyuan)
Dependencies: Set up Continuous Integration, Adapt Architecture Diagrams
The implementation is made up of many smaller tasks that the implementation sub-team can do in a flexible order so the smaller tasks are not shown in this plan.
Architecture Writeup - Low (Charlie)
Dependencies: Amend Architecture Diagrams
The architecture deliverable can be written at this point because none of Charlie's other deliverables are in progress.
Risk Writeup - Medium (Siyuan)
Dependencies: Amend Risk Table
Writing the rest of the risk deliverable now will leave time in the future to amend it if risks emerge during the rest of the project that we didn't anticipate.
Plan User Evaluation - Medium (Apollo, Siyuan)
Dependencies: Plan Requirements, Adapt Architecture Diagrams
Whilst the implementation is still in progress, we can prepare for the user evaluation by preparing the observation checklist and template.
Plan Software Testing - Medium (Aryaman, Vidhi)
Dependencies: Requirements Writeup
Before implementation is complete, prepare for software testing by planning a test for each requirement.
Perform User Evaluation - High (Apollo)
Dependencies: Plan User Evaluation, Implementation
Performing a user evaluation will provide us with feedback to improve aspects of the game we may not have noticed ourselves.
User Evaluation Writeup - High (Siyuan)
Dependencies: Perform User Evaluation
This will allow us to see the user feedback in an easy to understand table format which we can refer to while improving the code.
Improve Code - High (Apollo, Vidhi, Tracey, Siyuan)
Dependencies: Perform User Evaluation
At this point we will be able to make small changes to the code which will improve the

ease of use for the users.

Changelog Writeup - Medium (Apollo, Siyuan, Charlie)

Dependencies: Requirements Writeup, Risk Writeup, Architecture Writeup, Method Selection and Planning Writeup

The changelog will be left until the end of the project as further changes may be made to the updated deliverables between being written and being submitted.

Method Selection and Planning Writeup - High (Apollo)

The method selection and planning writeup will be left until the end of the project because it needs to include information from the whole process.

Perform Software Testing - High (Aryaman, Vidhi)

Dependencies: Plan Software Testing, Improve Code

Software testing will be performed at the end of the project after all coding is completed so that we can have objective tests proving that our requirements have been met.

Implementation Writeup - High (Tracey)

Dependencies: Implementation, Improve code

The implementation writeup will be left until after all coding has been completed and tests have proved that everything is up to standard. This is because otherwise changes may be made to the code that affect the writeup.

Continuous Integration Writeup - High (Tracey, Charlie)

Dependencies: Improve Code

Once all code is written we will be able to summarise the infrastructure and methods we used throughout our implementation to monitor the continuous integration.

Finalise Documents - High (All)

Dependencies: Requirements Writeup, Architecture Writeup, Risk Writeup, User Evaluation Writeup, Changelog Writeup, Method Selection and Planning Writeup, Implementation Writeup, Continuous Integration Writeup, Software Testing Report

Before submission we will review and provide any necessary edits to all written deliverables, especially those that were initially written much earlier in the project.

Software Testing Report - High (Aryaman, Vidhi)

Dependencies: Perform software testing

Once all software testing has been completed we will be able to summarise all of the results into one report which is clear to understand.

Update Website - (Aryaman)

Dependencies: Improve Code, Finalise Documents

Before submission the final versions of all the deliverables including an executable Jar file will be uploaded to the website to keep everything in one easily accessible place.

WEEKLY PLANS

How the plan changed throughout the project

The plan for this project changed a few times throughout its duration. In the first week, when we were familiarising ourselves with the new product brief and new required deliverables, we drafted a rough timeline of the project tasks. The aim of this was to divide the deliverables among the team members in a way that not only splits the workload equitably between team members, but also spread each individual's work load relatively evenly across the available time. We were able to broadly adhere to this plan but the implementation took longer than anticipated, going through the vacation period. This meant that in the final few weeks everybody needed to increase their productivity to make up for the delays.

During implementation we regularly needed to reassess the priorities of the remaining elements to be programmed. We used a table in our shared Google Drive to keep track of the owner, dependencies and importance of each element. In this way the plan for the implementation was very flexible because a team member was able to assess the highest priority item independently and also choose one that more closely matched their abilities and interest.

In the final days of implementation three of the four in the implementation sub-team stepped back to focus on their other deliverables. This meant that one person was able to merge all ongoing branches into one project, and fix any bugs, much more easily, not having to communicate each change with all other team members.