Assessment 2: Brownfield Development

Method Selection and Planning

Cohort 3 Group 9

Oliver Barden

Connor Burns

Sam Goff

Milap Keshwala

Lewis Mitchell

Jeevan Singh Kang

Harry Thomas

Titas Vaidila

Software engineering methods

Our team followed a scrum inspired, iterative and collaborative software engineering approach. Development was organized into weekly informal sprints where we combined implementation with regular communication and documentation to make sure everyone was always up-to-date. At our weekly meetups in person, we first reviewed from last week's sprint and adapted if certain tasks were taking longer to complete than initially expected. Then we identified and prioritized which tasks we should complete and in which order and then those tasks were assigned to the necessary team members. This allowed us to be flexible throughout the project. To support this we used a range of development tools and collaboration tools, each chosen to fit the project's needs and the team's preferences.

Development tools:
The team will be using Visual Studio code to develop in Java code, this means that the people who are writing the actual code will be able to collaborate with each other, sharing information about keybinds, extensions, and other capabilities streamlining the development process, rather than if each developer was using either other IDEs or just plain text editors. Vs code has core features such as built-in Git integration which is extremely helpful for the team members who are less familiar with Git. VS Code has Gradle support which is crucial since our Java game engine is libGDX which is built using Gradle. Most team members have previously used this before in past modules, so we felt an IDE that we were all comfortable with would be vital to the success of the project. The team members responsible for code development did not feel like using a third-party formatter would be useful, and it would just implement mild setup delays.

Game development framework
We used libGDX for this, which is an open source framework which supports both 2D and 3D games. Since our project is a 2D game, libGDX was a strong fit, further strengthened by its ecosystem. It provides many official and third-party extensions such as Ashley (entity framework) and Box2D (physics engine) which will be key to help develop this project. LibGDX has a large and active community with many resources that will help with learning and debug errors. We looked at using LWJGL which LibGDX was built upon. LWJGL would offer lower level access to hardware. We ended up using LibGDX due to it being more widely used, more suited for our project, and the steep learning curve that LWJGL has.

Code-specific collaboration:
For collaborating with code, the team will be using Git, and GitHub (as the platform to hold the code, discussions and revision history). The reason we have decided to use Git, is that it is an industry-standard way to keep track of version control, and see the progression of how the game code changes with the collaborators. There is an abundance of guides and documentation online on how to use Git, including all of its commands, etc.

From there, we have decided to (specified in the CONTRIBUTING.md file in the repository) use pull requests and branches (a mechanism that allows contributors to propose changes to a project through in their own isolated branches of code, which can then be reviewed and discussed before being merged into the main codebase) to have a structured workflow that allows each change to be isolated and examined by every collaborator before being merged into the main codebase.

There will also be a .gitignore file (a file which excludes certain files from being tracked by git, and therefore uploaded to the repository) to keep the repository clean and exclude any build logs, caches, etc that are not needed when someone is browsing the code.

Non-code collaboration:
One collaboration medium that is not for code, we have decided to use Google Docs and Google Drive. This decision was made with the main reason being familiarity with the tools - everyone on the team has used these tools before, and each of our university Google accounts has access to 2TB of storage space, which is plenty for this kind of work. We have split it into multiple files that we each have access to and can edit, and the main people working on each deliverable are responsible for working on the documents. Google Drive is simply used to hold all of these files that we work on.

We will also be using WhatsApp as the primary “informal” communication platform, and once again, the biggest factor drawing us to choose this platform was everyone’s familiarity with it, and how standardised across the world it is.

Level development:
For the map development, to design and plan each of the levels for the game, we use a software called “Tiled”. It is a free, open source and full-featured level editor, that supports flexible object layers, is extensible with JavaScript, and is widely supported by many game development frameworks, such as Unity, Godot, as well as lots of languages like C, C#, Go, Java (what we are using for the game), Python, Rust, and many others.

We chose to use Tiled since it is more popular than other widely used and free alternatives, such as LDtx, but that is also a free and open source 2D level editor that is widely used and supported.

Definitions and change control:
A change is done when it is approved by the programming team, and CI is accepted on Linux, macOS and Windows and a JAR runs locally.

We implement the desktop game with Java 17, libGDX and Gradle. Java 17 is the mandated language level and provides a portable, cross-platform runtime. libGDX is a lightweight 2D engine with a mature LWJGL3 desktop backend, tiled-map support, text rendering and audio, which helps us target low-spec laptops and meet the client’s cross-platform requirement. Gradle offers fast builds as well. An alternative to Gradle is Maven, which is a robust build tool as well.

Team approach to organisation:

We operate without a formal leader, however for certain complex and time consuming tasks we designated a leader who would manage and organise that task to make sure it gets completed on time and with quality. This way of organising our team suits this project since we need to split our work load evenly and we all have around the same amount of experience in working on projects so it doesn’t make sense for someone to lead the group. At the start we assigned work based on preference and skills (e.g., changelog, coding).

Throughout the project we have had multiple different tasks being carried out by small subgroups of our team. This is to ensure that progress is being made simultaneously on several different aspects of the project at once. In most cases at least two people were assigned to a specific task to add redundancy - if one team member is unavailable for whatever reason, there will usually be someone else to pick up the slack and continue working.

We made sure that during our meeting everyone pitched in and that we were always aware of the tasks which needed to be completed for that week along with who is responsible for these tasks. We would also discuss if a certain task was incomplete and what steps should be taken to address this. Small decisions are made within the relevant sub-team (e.g., the two programmers) to avoid blocking others; big decisions (direction, scope, trade-offs) are discussed biweekly in a whole-team meeting with updates/demos and decided by vote. Day-to-day communication and quick questions are handled asynchronously on WhatsApp.

This approach fits the team and project: everyone expressed satisfaction with the setup.It is using tools familiar to everyone (GitHub, Google Docs/Drive, WhatsApp) to reduce overhead and suit mixed student schedules. Pairing in fragile areas and occasional role rotation mitigate single-point-of-failure risks highlighted in our risk assessment. If anyone needs history, we have a version history for each file through google docs and google drive, and for the code, there is a commit history in the repository.

Systematic plan:

Milestones and dates (A1 due Mon 10 Nov 2025):

- Iteration 1 (1–7 Oct): Risk register baseline; requirements skeleton; website scaffolded.
- Iteration 2 (8–14 Oct): Requirements expanded; architecture baseline. ● Iteration 3 (15–21 Oct): Architecture refined; risk register matured. ● Iteration 4 (22–28 Oct): Repo scaffold (Java 17, Gradle, libGDX); CI setup. ● Iteration 5 (29 Oct–4 Nov): Map pipeline (Tiled) and art; event system skeleton. ● Iteration 6 (5–10 Nov): Timer/pause/counters; polish, packaging, PDFs and website.

Key tasks with dates, priorities, dependencies:

T001 - Risk assessment and mitigation
- Start: 1 Oct • Finish: 28 Oct
- Dependencies: none • Output: Risk1.pdf, risk register (IDs, likelihood/impact, mitigation)

T002 - Requirements elicitation and specification
- Start: 2 Oct • Finish: 7 Nov
- Dependencies: none • Output: Req1.pdf with UR/FR/NFR IDs and acceptance criteria

T003 - Architecture (structural + behavioural, traceability)
- Start: 9 Oct • Finish: 7 Nov
- Dependencies: T002 (req IDs) • Output: Arch1.pdf with Doc

T004 - Tooling and CI (Java 17, Gradle, libGDX)
- Start: 22 Oct • Finish: 3 Nov
- Dependencies: none • Output: Initial code in the github repository

T005 - Map pipeline and art (Tiled + layers)
- Start: 29 Oct • Finish: 7 Nov
- Dependencies: T004 (project scaffold) • Output: maze assets and layers

T006 - Event system skeleton (triggers/effects)
- Start: 29 Oct • Finish: 4 Nov
- Dependencies: T005 (map loader) • Output: Triggering mechanism, effects interface

T007 - Implement required A1 events (1 negative, 1 positive, 1 hidden)
- Start: 30 Oct • Finish: 7 Nov
- Dependencies: T006 • Output: Three working events, counters incremented

T008 - Timer, pause and counters UI
- Start: 22 Oct • Finish: 5 Nov
- Dependencies: T004 (loop/input), T006 (event outcomes for counters) • Output: 5-minute timer with pause, counters overlay

T009 - Packaging
- Start: 1 Nov • Finish: 6 Nov
- Dependencies: T004, T008 • Output: Single executable JAR, validated on Windows/macOS/Linux

T010 - Accessibility and attribution
- Start: 1 Nov • Finish: 7 Nov
- Dependencies: T005–T008 • Output: Multimodal cues (text + visual/audio), Assets/ATTRIBUTION.md

T011 - Website and deliverables
- Start: 1 Oct • Finish: 9 - 10 Nov
- Dependencies: all prior • Output: Website with links to PDFs, JAR, repo; final PDFs (Req1.pdf, Arch1.pdf, Plan1.pdf, Risk1.pdf, Impl1.pdf)

Milestones and dates(A2 due Mon 12th Jan 2026):
- Iteration 1 (17th -23rd Nov): Establish changelog and documentation method, update requirements for A2, set plan.
- Iteration 2 (24-30 Nov): Rework architecture, set up CI workflow and testing.
- Iteration 3 (1-7 Dec): Implement requirements/architecture changes, create user evaluation process/doc, begin risk and plan updates.
- Iteration 4 (08-14 Dec): User evaluation session and complete doc, update architecture/plan to reflect feedback.
- Iteration 5 (15-21 Dec): Begin implementation refresh from evaluation.
- Iteration 6 (29-04 Jan): Complete all final reports.
- Iteration 7 (05-11 Jan): Final polish, package, PDF for all deliverables, create presentation.

Key tasks with dates, priorities, dependencies:

T012 - Change to requirements doc.

- Start: 20th Nov
- Finish 23rd Nov
- Dependencies: none
- Output: Req2.pdf

T013 - Change to architecture doc.

- Start: 24th Nov
- Finish: 14th Dec
- Dependencies: T012
- Output: Arch2.pdf

T014 - Change to risk doc.

- Start: 24th Nov
- Finish: 23rd Dec
- Dependencies: T012
- Output: Risk2.pdf

T015 - Change to plan doc.

- Start: 24th Nov
- Finish …
- Dependencies: none
- Output: Plan2.pdf

T016 - Add CI.

- Start: 24th Nov
- Finish: 30th Nov
- Dependencies: none
- Output: CI code in github repository, CI.pdf

T017 - Implement requirements/architecture changes.

- Start: 1st Dec
- Finish: …
- Dependencies: T012,T013,T016
- Output: Leaderboard, achievements, events coded

T018 - Testing.

- Start: 30th Nov
- Finish: …
- Dependencies: T017
- Output: Tests for all relevant classes

T019 - User evaluation.

- Start: 16th Dec
- Finish: 6th Jan
- Dependencies: All prior
- Output: User evaluation.pdf

T020 - Website and deliverables.

- Start: ...
- Finish: …
- Dependencies: All
- Output: Website with links to all PDFs, JAR, repo

T021 - Presentation.

- Start: …
- Finish: …
- Dependencies: All

- Output: Yetti.pptx

Dependencies summary:

- T003 depends on T002 (traceability from requirements).
- T006 depends on T005 (map loader and layers).
- T007 depends on T006 (event framework).
- T008 depends on T004 (input/game loop) and T006 (event outcomes).
- T009 depends on T004 and T008 (build + artifact).
- T011 depends on all prior tasks.
- T013 and T014 depend on T012 (traceability from requirements).
- T017 depends on T012, T013, T016 (requirements, architecture and continuous integration).
- T018 depends on T017 (Code implementation).
- T019, T020 and T021 depend on all prior tasks.

How the plan evolved:

Our initial plan front-loaded discovery and documentation in the first two weeks, so things such as risk register, meeting with the client to create the requirements skeleton, and a website scaffold. As requirements solidified, we produced a baseline architecture (structural first, then behavioural), and scheduled weekly snapshots to capture changes.

After scaffolding the libGDX project (Java 17 with Gradle), we learned that integrating Tiled maps and event triggers would take longer than estimated because map loading and collision layers needed more iteration. We pulled timer/input work forward and pushed the event-system skeleton back by one week. This change was recorded as a plan update in the next snapshot.

The requirements grew to include main menu and settings - we retained these in the spec for completeness but re-classified them as should/could for A1. We also tightened the wording of hidden events and added acceptance criteria, which improved testability and reduced work later.

Accessibility and licensing choices also evolved. We initially considered richer audio narration, but shifted to simpler multimodal cues (clear text plus parallel visual/audio signals) to avoid asset/licensing risk and fit the timebox

Resourcing adjustments were needed when availability fluctuated. We concentrated map work with one owner and split scoring/events, while pairing on fragile areas (input handling, collisions) and small refactors (e.g., extracting InputHelper) were scheduled inside tasks to reduce future integration risk without creating separate milestones.

Overall, the plan for Assessment 1 stayed within the original milestones with two notable adjustments: the event-system skeleton slipped by a few days and was recovered by trimming non-essential polish. Weekly progress on the website should reflect these changes with brief rationales.

For Assessment 2, the initial plan began focusing on changing the relevant Assessment 1 documents, starting with requirements and risk ,as well as setting up continuous integration, in order to be prepared for completing the architecture and implementation. We quickly realized that it would be beneficial to start implementation and testing as soon as possible and so gradually added tests and more events so we would not fall behind.

After carrying out user evaluation, the feedback showed several implementation issues such as text overlapping and events that were difficult to understand. So fixing these became a priority.

Once we got feedback for Assessment 1 we decided to go back and add or change to the documentation, which added requirements. Therefore we made sure these requirements were implemented before carrying out new implementation.

Overall, the plan for Assessment 2 also stayed within the original Milestones except for 2 slight setbacks as outlined above.