Method selection and planning

Cohort 3 Team 7 (Yetti)

Bak, Bartek
Burberry, Katelyn
Collins, Lucy
Ganley, Joe
Keegan, Josh
Kirkham, Katharine
Mayall, Daniel
Sawdon, Theo

Development tools:
The team will be using IntelliJ IDEs 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. IntelliJ includes a rudimentary built-in formatter, that will for example indent the code correctly on save. 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.

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. Each commit also follows conventional commit naming, such as “feat“, “fix”, “docs”.

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.

Project planning tools:
For project planning, we have also decided to use Github issues, they are not just for issues, but can have all forms of discussions. Alternative tools for this include Jira, Trello and Asana, and we looked at them, and while they had free plans, either nobody or only a few people on the team ever used them, whereas most are familiar with Github, and keeping things to just one platform reduces overhead for the development and documentation teams.

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 that 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.

CI setup:
For continuous integration, we use Github Actions. A big reason for this once again is that it is integrated into the platform that we are already using for code hosting, issue tracking, discussions, etc. We have learned in one of the lectures how to set up basic Actions, so we will use it to run builds, and checks to make sure it compiles and runs on the platforms outlined in the requirements, get the compiled jar file, upload it as an artifact and create releases with the Jars attached. Some alternatives we considered to use were GitLab CI/CD (it is powerful and YAML-based, and great if the repo is on GitLab, but our repo is on Github and using it adds friction (mirrors/tokens) and an extra ui to manage) and self-hosted Jenkins (we would have full control but we would have to provision servers, plugins, backups, etc).

Definitions and change control:
To keep work predictable, an issue is Ready when it satisfies a requirement from the list of requirements derived from the initial (or subsequent) client meetings. A change is done when it is approved by the programming team, and CI is accepting on Linux, macOS and Windows and a JAR runs locally. If any issues arise, they are opened on GitHub issues to be addressed.

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. At the start we assigned work based on preference and skills (e.g., requirements, risk assessment, coding). GitHub is our single source of truth: all work flows through Issues and PRs with labels and assignees (no templates, no requirement IDs). The main branch is protected; merges for the game code go via reviewed PRs, while low-risk repository admin/docs changes (e.g., CONTRIBUTING.md) can go directly to main. 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 minimises ceremony while enforcing quality with things like branch protection, code review and visible ownership on Issues/PRs, and 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 • Output: Website with links to PDFs, JAR, repo; final PDFs (Req1.pdf, Arch1.pdf, Plan1.pdf, Risk1.pdf, Impl1.pdf)

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.

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, 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 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.