Method selection and planning

C3 Group 6
Team WHNI

Aaratrika Das
Arkoday Roychowdhury
El Foster
Hanna Pieniazek
Lewis Keat
Matthew Baghomian
Matthew Ford
Subham Magar

Development tools:
The team will be using IntelliJ IDE to develop in Java code; this means that the contributors tasked with writing code will be able to collaborate with each other, sharing information about keybinds, extensions, and other capabilities; this streamlines the development process, rather than if each developer was using other IDEs. IntelliJ includes a built-in formatter that provides many useful features, such as automatic bracket formatting. For this reason, contributors feel that a third-party formatter will be unnecessary, and would bloat load times. Additionally, we will utilise PlantUML for architectural modelling. This tool integrates directly with IntelliJ, allowing us to create structural and behavioural diagrams such as Sequence and Class diagrams using code.

Software Methodologies:
We will be using the Agile methodology for our project. This methodology uses an iterative project management and it divides development into small, manageable cycles known as sprints. This should give us continuous development, regular feedback and incremental improvements throughout the entire project. In addition to this, after every sprint we could reflect as a team to figure out improvements that were needed and make the necessary adjustments. Overall, this will ensure that team members are aware of the current status of the project at all times, which facilitates effective team communication and simplifies task allocation.

Code-specific collaboration:
For collaborating with code, the team will be using Git via GitHub (as the platform to hold the code, discussions and revision history). We decided to use Git because it is an industry-standard method of version control, and displays how the game code changes with the collaborators. There are an abundance of guides and documentation online on how to use Git, ensuring that all collaborators understand how to utilise it effectively.

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

There will also be a .gitignore file 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:
For storing deliverables and other files, we have decided to use Google Docs and Google Drive. We chose this as all members were familiar with the tools, and each of our university Google accounts has access to 2TB of storage space, which is sufficient for this project.

We will also be using WhatsApp as the primary informal communication platform; we chose this as all members possessed an acceptable level of familiarity with it.

Project planning tools:
For project planning, we have also decided to use Github issues; this facilitates various forms of discussions. Alternative tools for this that were considered include Jira, Trello and Asana; since most of our members are not familiar with these tools, but most are familiar with Github, we decided that Github would be the simplest to use, and keeping things to just one platform reduces overhead for the development and documentation teams.

Map 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, full-featured level editor that supports flexible object layers, and is extensible with JavaScript. It is widely supported by many game development frameworks, such as Unity, Godot and LibGDX. We chose to use Tiled because of its user-friendly interface, wide-ranged support and ease of integration.

CI setup:
For Continuous Integration, we will use Github Actions. The main 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 will use it to automatically run builds and tests whenever a commit is made. It will also upload binaries, from which the game can be run on the respective platforms.

Some alternatives considered 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 denoted as 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 will 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.

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 for Assessment 1:

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)

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).
- T010 depends on T005 and T008
- T011 depends on all prior tasks.

Priorities (MoSCoW)

- Must have: T001–T009, T011 (meets A1 brief: one of each event, timer up to 5 minutes with pause, counters, executable JAR, and website with links).
- Should have: T010 (multimodal cues, attribution); simple victory screen polish.
- Could have: Optional marketing link/QR event if time permits (de-scoped if it risks Musts).

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 the 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.

Systematic plan for Assessment 2:

Milestones and dates (A2 due Mon 12 Jan 2026):

- Iteration 7 (17-24 Nov): Planning changes, events.
- Iteration 8 (25 Nov-1 Dec): Leaderboard, refactoring, events.
- Iteration 9 (2-8 Dec): Unit testing, leaderboard, map and art updates.
- Iteration 10 (9-15 Dec): Unit testing.
- Iteration 11 (16 Dec - 4 Jan): Continuous integration, art updates, unit testing.
- Iteration 12 (5 Jan - 11 Jan): Polishing, pdfs, website

Key tasks with dates, priorities, dependencies:

T012 - Review old documents, make necessary changes
- Start: 17 Nov • Finish: 24 Nov
- Dependencies: T001 - T011 • Output: Change2.pdf and all the updated deliverables

T013 - Clone repository and refactor code
- Start: 17 Nov • Finish: 24 Nov
- Dependencies: T004 - T008, T010 • Output: New repository on Github, refactored code

T014 - Additional events for Assessment 2
- Start: 24 Nov • Finish: 7 Dec
- Dependencies: T013 • Output: 4 additional negative events, 2 additional positive events, 2 additional hidden events

T015 - Achievement System
- Start: 24 Nov • Finish: 7 Dec
- Dependencies: T013 (map loader) • Output: Achievement system and achievements

T016 - Leaderboard functionality
- Start: 1 Dec • Finish: 7 Dec
- Dependencies: T013 • Output: Leaderboard class and methods, ability to save score

T017 - Create new map
- Start: 1 Dec • Finish: 14 Dec
- Dependencies: T013 • Output: New maze with improved graphics

T018 - Implement CI techniques
- Start: 8 Dec • Finish: 11 Jan
- Dependencies: T013 • Output: CI2.pdf, Github Actions, automated builds and release

T019 - Create new screens and edit existing
- Start: 8 Dec • Finish: 11 Jan
- Dependencies: T013 • Output: Leaderboard screen, updated screens

T020 - Carry out user evaluation
- Start: 8 Dec • Finish: 15 Dec

- Dependencies: T013 - T019 • Output: Eval2.pdf, user evaluation data

T021 - User evaluation code changes, music and sound effects
- Start: 15 Dec • Finish: 11 Jan
- Dependencies: T018 • Output: Updated code after reflections from user evaluation

T022 - Create and run tests
- Start: 8 Dec • Finish: 11 Jan
- Dependencies: T013-T019 • Output: Test2.pdf, automated tests in Github Actions

T023 - Website and deliverables
- Start: 5 Jan • Finish: 11 Jan
- Dependencies: all • Output: Website with links to PDFs, JAR, repo; final checks of PDFs

Dependencies summary:

- T0012 depends on T001-T011 (all the previous documents from assessment 1).
- T0013 depends on T004 - T008, T010 (all code & repository tasks)
- T014 - T019 all depend on T013
- T020 depends on T013 - T019 (all new code features)
- T021 depends on T020 (user evaluation)
- T023 depends on all prior tasks

Priorities (MoSCoW)
- Must have: T012–T020, T022 - T023 (refactored deliverables & code, events, leaderboard, achievements, CI, user evaluation carried out)
- Should have: T021 (user evaluation code changes)

How the plan evolved:

Very little deviation from the original plan was required. The only notable change was the events taking slightly longer than expected, and so an extra week was allocated; however, these were finished in time for user evaluation.