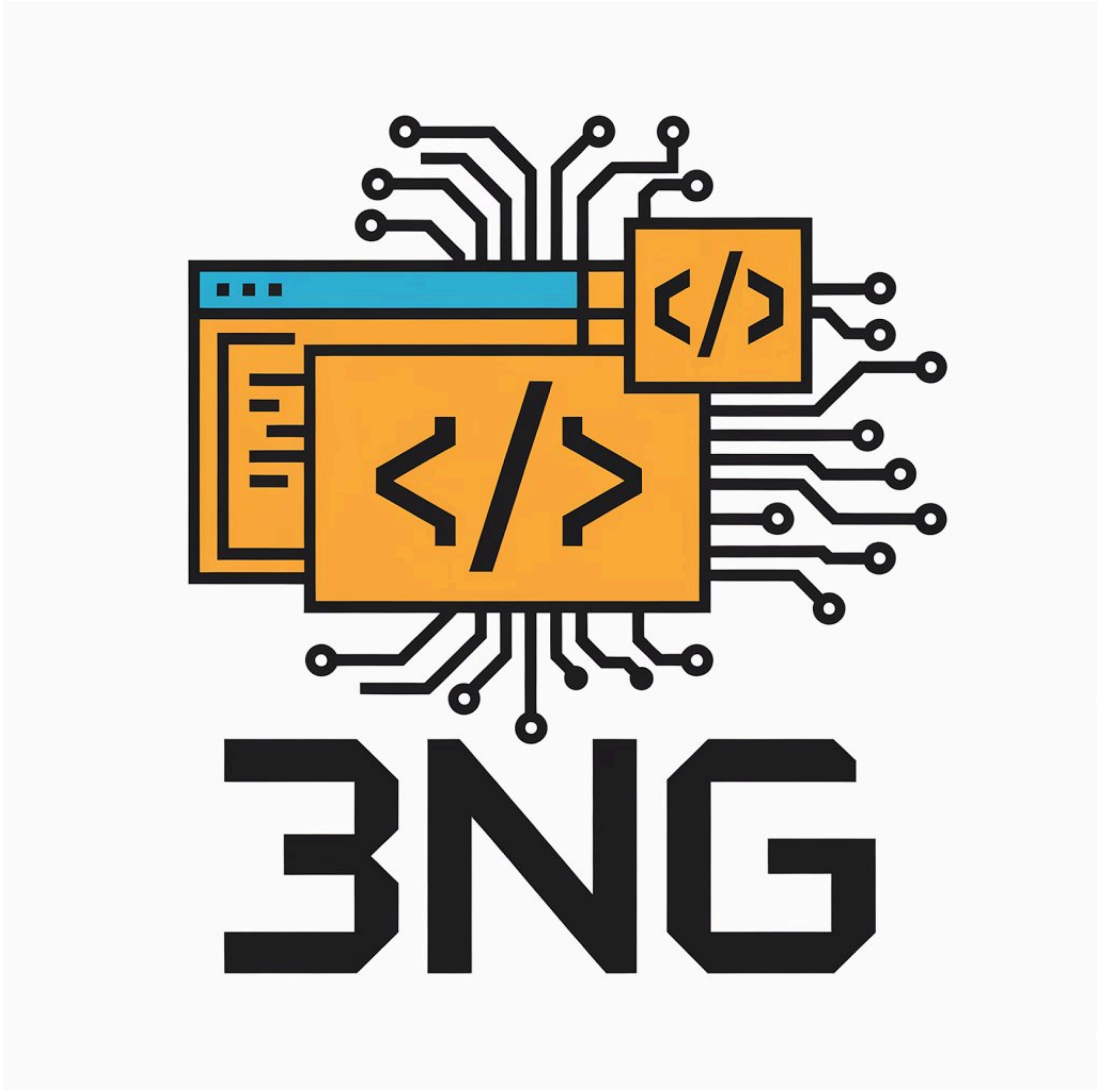
Continuous Integration Report

Cohort 4 Group 3

Team Members
Charles Fellows
Elif Gunes
Eve Davey
Hongrui Liu
Leo Owen-Burton
Tomi Olagunju
Will Bannerman



Part A: CI Methods and Approaches

1. Managing Risk and Complexity

Our continuous integration (CI) approach evolved from basic build verification in Assessment 1 to a more risk-focused integration strategy in Assessment 2. As outlined in the product brief, Assessment 2 introduces a substantial increase in system complexity, including multiple independent event types, a scoring system, achievements, and a leaderboard. These features increased the complexity of the game's internal state machine and interactions between components.

To manage this risk, the team adopted CI practices that support parallel development and early detection of integration issues. Team members were able to work concurrently on separate functional requirements (for example, patrol-based enemy behaviour [FR_E_PATROL] or bonus interaction events [FR_INTERACTION_BONUS]) while relying on CI to continuously verify that their changes did not introduce logical or build-breaking errors when merged into the shared codebase. This reduced the likelihood of late-stage integration problems and ensured that interactions between components behaved as expected.

2. Automated Integration Methods

Following recommended CI practices, all changes were integrated frequently into a single shared repository. Commits were kept small and focused [1], making integration issues easier to identify and resolve. Each integration was automatically verified using an automated build pipeline, ensuring that the project compiled successfully and that existing functionality remained intact. This aligns with the CI principle that every commit should be tested on a clean integration machine before being merged, avoiding environment-specific issues like if someone claimed "this distribution works on my machine, why not yours?"

3. Appropriateness

The adopted approach is particularly well suited to the requirements of 'Escape from Uni' with a small design team behind it; the system combines event-based mechanics, time-driven gameplay and non-trivial scoring logic. The product brief specifies that the player's score depends on both time-to-escape, combined with interactions with several events. Automated builds and tests ran on our CI infrastructure help verify that changes to event logic do not have unintended consequences for score calculations, or other features.

The game also operates under strict real-time constraints, with a maximum duration of 5 minutes and the ability to pause at any time. These timing-related features are vulnerable to regression, as small changes can easily disrupt intended functionality. CI provides rapid feedback when changes break these core mechanics, allowing issues to be detected and resolved early in the development lifecycle.

Finally, the game will be marketed towards a casual audience and evaluated by a stakeholder acting as a prospective customer. Thus it is essential that the main branch of the source repository remains in a playable state, free from potentially fatal issues such as inaccessible maze areas or unintentional event triggers.

Part B: CI Infrastructure

1. Overview of our CI platform

We used GitHub Actions to implement our CI workflow, which is tightly coupled with the project's GitHub repository, and also mirrors the workflow demonstrated in the CI lecture. The configuration is defined using a YAML workflow file located in the .github/workflows/ directory. GitHub Actions was chosen as it provides a hosted integration machine, requires no additional server configuration, and offers clear visibility of build status for all team members.

The CI workflow is automatically triggered based on:

- Pushes to the main branch.
- Pull requests targeting the main branch.

This supports validation both before and after integration into the main codebase, preventing incompatible changes from accumulating.

2. Build Environment

Each CI workflow executes on a clean, Linux-based virtual machine packaged with GitHub Actions. It begins by checking out a fresh copy of the repository and installing any required dependencies or runtime components during setup. This allows a consistent execution environment across all integrations. The core CI job executes an automated build process that compiles the project and runs all available automated tests. This prevents compilation errors or existing functionality from breaking when new changes are introduced. If the build or test process fails at any step, the workflow is marked as failed, and the associated commit or pull request is flagged in GitHub.

3. Artifacts, Visibility and Feedback

Build results can be viewed through GitHub's Actions interface; GitHun also annotates commits and pull requests with pass/fail indicators, allowing team members to inspect logs and diagnose issues with the latest attempted changes. Where applicable, build outputs such as executable files or reports can be uploaded as CI artifacts. This allows team members and stakeholders to download and run a known working version of the game without requiring a local build, which helps with test and demos throughout the project.

[1] M. Fowler, *Continuous Integration*, MartinFowler.com, 2006. https://martinfowler.com/articles/continuousIntegration.html