Continuous Integration

Cohort 2 Team 1

Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Continuous integration method and approach

What continuous integration means in this project
Continuous integration is a practice in development which automatically verifies changes when they are integrated into a shared repository. When new code is added, the project is checked to confirm that it still builds correctly and that any existing functionality has not been broken (by running all implemented code tests).

How the team integrates work
The team works on a shared GitHub repository. Each coder will create individual features or fixes on separate branches. When the code is ready, the coder will submit a pull request to the main branch on GitHub.

Why automated builds and tests are essential
Automated builds and tests are particularly important for this project as there are multiple contributors and a codebase which is continually expanding. Without continuous integration, errors such as syntax errors or broken game logic could go undetected until later in the development. Continuous integration provides immediate feedback, so issues can be fixed before future branches depend on faulty code.

Why this approach is appropriate
This continuous integration approach is appropriate for the scale of the project, as it is lightweight, easy to maintain and provides feedback efficiently. This allows us to focus on development.

CI infrastructure actually implemented
We implemented continuous integration using GitHub Actions. GitHub Actions allows us to define automated workflows which are stored in the repository itself using configuration files. Our project is built using Gradle and the continuous integration pipeline runs on a virtual machine provided by GitHub, using Java 17. Since GitHub builds and runs the tests on a clean VM, this ensures the project works regardless of the developer's local environment.

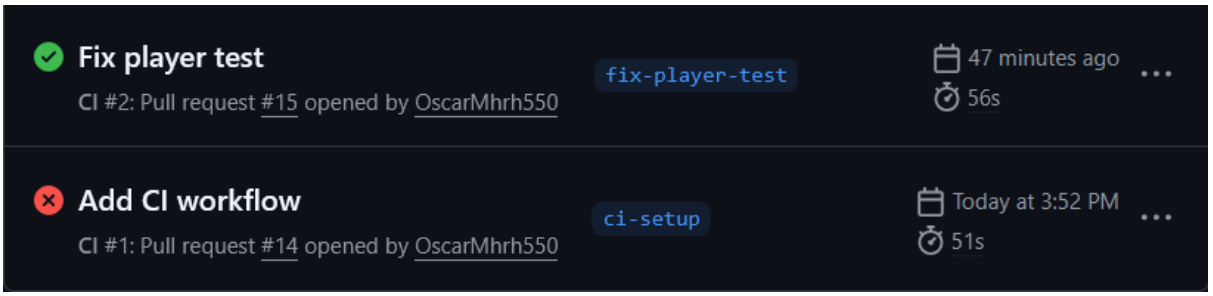
CI workflow integration
The continuous integration workflow is defined in a YAML config file in .github/workflows/ci.yml. It is configured to run automatically when someone submits a pull request to the main branch. This ensures that all changes are verified before being integrated.

The workflow does the following steps:

- Downloads the code in the repo and checks it
- Sets up the required Java environment
- Executes "./gradlew clean build", which compiles the project and runs all automated tests

Why we made these choices
We catch errors early on by ensuring all changes are verified before being integrated. The single Gradle command makes it simple for team members to contribute, allowing them to repeatedly validate the project using the singular Gradle command. The CI pipeline runs in a fresh environment which performs a clean build. This means the results are not influenced by previous outputs from builds, ensuring that any failures reflect real code issues.

What is checked by the CI workflow
When the continuous integration workflow runs, the project is compiled and all automated tests are executed. These tests make sure that the game logic is correct. For our project, examples include player logic, the collision checker and inventory logic. If the project does not compile or any logic fails, the workflow fails. Below is shown how successes and failures appear on GitHub:



Feedback and enforcement
As the results of the workflow are shown directly on the pull request, it is a good indication of whether the pull request can be safely merged or not. If the workflow fails, GitHub will show which step failed, which allows the team to fix the issue more efficiently. It also ensures that broken code does not get merged by accident to the main branch.

Why this infrastructure is sufficient
This infrastructure provides all the functionality required for our project. It automatically verifies integrations, provides fast feedback and enforces that our builds are correct. On top of this, it is lightweight, not overly complex and easy to maintain. This makes it a reliable tool to ensure safety of the main branch and reduce the risk of problems later on due to integrating broken code.