CI Report

Team 10

Gergely Gal

Ben Leaver

Utkarsh Singh

Ali Tekin

Hux Stevenson

Faras Hbal

Chihun Park

a) CI methods and why they fit this project

For DebugThugs, the team uses GitHub Actions as the main continuous integration system. Every time someone pushes to the master or main branch, or opens a pull request into those branches, GitHub spins up a workflow that checks the project automatically. The workflow builds the project with Gradle, runs the JUnit tests, and generates code coverage with JaCoCo, so the team quickly sees if a change breaks the build or reduces test coverage.

The pipeline also enforces code style using Checkstyle. Reports are generated for each module and stored as artifacts the team can download and review, which helps keep the codebase consistent as multiple people work on it. Because the project is a LibGDX game expected to run on different platforms, the workflow uses a matrix to run on Ubuntu, Windows and macOS in parallel, which catches platform-specific issues early.

On top of that, there is basic release automation. When it is pushed, GitHub Actions builds distribution JARs for each platform and creates a GitHub release with those files attached. This removes a lot of manual work from packaging the game and makes it easier to ship consistent builds. Overall, this combination of automated builds, tests, style checks, coverage and releases suits a student game project well: it keeps the main branch stable, supports collaboration, and gives fast feedback when something goes wrong.

b) CI infrastructure

The CI setup is entirely hosted by GitHub's own runners, so there is no separate server to manage. The workflow file (gradle.yml) defines two jobs: build, which runs on Ubuntu, Windows and macOS, and release, which runs on Ubuntu and only triggers for version tags. Each build job checks out the repo, installs Temurin JDK 17, and configures Gradle via the official Gradle setup action.

On Linux runners, the workflow installs xvfb so LibGDX code that depends on a display can run in a headless X server, which is important for automated testing of a graphical game. The main build step runs ./gradlew build jacocoTestReport jacocoRootReport, which compiles the game, runs tests and produces coverage reports. After that, the workflow uploads Checkstyle reports and JaCoCo reports as artifacts and also builds a distribution JAR in the lwjgl3 module and uploads it so team members can download a tested build directly from the workflow run.

The release job depends on the build job finishing successfully. It downloads all the build artifacts from each operating system and then uses a GitHub Action to create a release for the tag, attaching the JARs for each platform. Permissions are kept tight: the build job has read-only access to the repository contents, while the release job gets write access only so it can publish releases. This setup gives the project a reasonably complete CI pipeline (covering build, test, quality checks and packaging) without needing any extra infrastructure beyond GitHub itself.