Continuous Integration

Cohort 3 Team 4

Kiran Kang

Hannah Rooke

Ben Slater

Abualhassan Alrady

Cassie Dalrymple

Charles MacLeod

Dash Ratcliffe

Harley Donger

6 - Continuous Integration

6a - Methods

We decided from the beginning that a simple system, containing all the infrastructure within one file and only using a few jobs, would be easiest to maintain and expand. This proved to be the right approach, as when we later updated it with a summary of the passing tests it was very simple to add the extra step.

We made test running a priority, so that we gained the benefits of testing every time we pushed a commit, and could confirm that the tests were passing on a third-party machine to eliminate any issues with specific software versions causing inconsistencies.

Cross-platform compatibility is essential for making the game available to our user - the requirement NFR_SYSTEM_RESTRICTIONS specifies that the game shall be available on as many operating systems as possible. Using Java as the language guarantees compatibility on any system with the JVM installed, and LibGDX is cross-platform. Since NFR_SYSTEM_RESTRICTIONS also specifies the game should be easy to run, we use Packr to bundle everything together, so the final product functions as a native binary. This removes friction, allowing the user to run it in the same way they would any other app, with no need to install requirements or understand different java versions.

There are two pipelines, one for building and testing, and the other for release. Both pipelines are triggered by pushing a commit to Github. The build and test pipeline runs whenever commits are pushed to any branch, in order to detect any changes breaking the build or failing tests early. We have emphasised running a build and tests locally before pushing to ensure the minimum of bugs make it to the source repository.
When pushing with a version tag, the native binaries for all three OSes will be added to a new Github release, making it much easier for an end-user to access.

The pipelines take as input the assets, source code and gradle build configuration. All of the relevant materials are kept in one repository to ensure they can be easily found and builds can be made with as few complications as possible.

They output built native binaries for Windows, Mac and Linux, containing everything needed to function, including a copy of the JDK in case the user doesn't have it installed. Also produced are the built jars from the compilation process, which can be downloaded and tested to ensure that it has produced a functioning artifact. Also attached is the Checkstyle report, as a HTML document. Integrating Checkstyle helped to standardise formatting, settled any discussion over the correct style and created smaller, more meaningful diffs between branches, making merges easier.

6b - Infrastructure

We chose Github Actions for our CI infrastructure for a few main reasons. Primarily, we were already using Github as our version control system, bug tracker and project management software, so using their CI system seemed the simplest way to ensure everything was in one place and easy to find. It also made it easy to deploy changes (simply storing and updating the `gradle.yml` in the git repository itself).

However, we did consider another CI, Jenkins. It had some important advantages:

- It is very flexible and has a huge plugin library, so we would definitely be able to create the right workflow for this project.
- It's decoupled from a VCS, so if we decided to move away from Github, we would be able to take it with us.

Unfortunately, it was overkill for our use and would have required a lot of specialised work (pipelines are written in Groovy, a new language for all of us) for not much gain. A simpler solution was the better choice.

Having settled on Actions, we implemented the cross-platform compilation and testing using the matrix strategy for variables:

```yaml
jobs:
  build:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
    runs-on: ${{ matrix.os }}
```

Figure 1 - The declaration of the OS matrix

We used this to repeatedly run the same steps on multiple operating systems. This is more flexible than copy-and-pasting the same code because it removes the need to update code in three places rather than one as well as it being easy to expand at a later date if we needed to add more possible configurations. Running the same job, with the same steps, on a different OS means that we can guarantee cross-compatibility.

When pushing with a tag, the release will only be created if the tag is a "version tag", i.e. it matches a particular format. The regex we use for this is "`v[0-9]+.[0-9]+.[0-9]+`". This means that we won't accidentally create a release on pushing any other tag, which lets us still have the freedom of tags for organising the codebase and ensures any releases are purposeful.

When using Actions, the security implications of pinning your CI pipeline to whichever code is associated with a tag which can be reassigned by the action's author is a serious security issue and could be the cause of a supply-chain attack. As such, we use commit hashes whenever we reference an action to ensure we have a safe version, with a comment showing the version to ensure readability.

```yaml
- uses: actions/checkout@8e8c483db84b4bee98b60c0593521ed34d9990e8 #v6.0.1
- name: Set up JDK 17
  uses: actions/setup-java@f2beeb24e141e01a676f977032f5a29d81c9e27e #v5.1.0
```

Figure 2 - The commit hashes

Bibliography

[1] M. Fowler, “Continuous Integration,” *martinfowler.com*, Jan. 18, 2024. https://martinfowler.com/articles/continuousIntegration.html (accessed Dec. 29, 2025).

[2] M. Rehkopf, “What is Continuous Integration | Atlassian,” *Atlassian*, 2019. https://www.atlassian.com/continuous-delivery/continuous-integration (accessed Dec. 29, 2025).

[3] Y. Morgenstern, “Distributing Java Desktop & Android projects with Github Actions,” *Medium*, Oct. 30, 2021. https://yairm210.medium.com/ci-cd-for-libgdx-projects-with-github-actions-f94830654629 (accessed Dec. 29, 2025).

[4] uoy-cs-eng1, “GitHub - uoy-cs-eng1/sample-gradle-ci: Sample project for continuous integration of a Gradle Java library project,” *GitHub*, Dec. 09, 2022. https://github.com/uoy-cs-eng1/sample-gradle-ci (accessed Dec. 29, 2025).

[5] S. Dalvai, “libGDX Android upload - GitHub Marketplace,” *GitHub*, 2025. https://github.com/marketplace/actions/libgdx-android-upload (accessed Dec. 29, 2025).

[6] R. Skoberg, “GitHub - rafaskb/awesome-libgdx: 🎮 📝 A curated list of libGDX resources to help developers make awesome games.,” *GitHub*, Dec. 03, 2024. https://github.com/rafaskb/awesome-libgdx (accessed Dec. 29, 2025).

[7] B. Thomas, “GitHub - bthomas2622/libgdx-github-actions-demo: GitHub Actions with LibGDX game,” *GitHub*, 2025. https://github.com/bthomas2622/libgdx-github-actions-demo (accessed Dec. 29, 2025).

[8] Jenkins, “Jenkins,” *Jenkins*, 2023. https://www.jenkins.io/ (accessed Dec. 29, 2025).

[9] “GitLab CI vs. GitHub Actions: a Complete Comparison in 2025,” *Bytebase*, Apr. 2025. https://www.bytebase.com/blog/gitlab-ci-vs-github-actions/ (accessed Dec. 29, 2025).

[10] “Get started with GitLab CI/CD | GitLab Docs,” *Gitlab.com*, 2025. https://docs.gitlab.com/ci/ (accessed Dec. 29, 2025).

[11] T. Preston-Werner, “Semantic Versioning 2.0.0,” *Semantic Versioning*. https://semver.org/ (accessed Dec. 29, 2025).

[12] J. Nielsen, “Severity Ratings for Usability Problems,” *Nielsen Norman Group*, 1994. https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/ (accessed Dec. 30, 2025).

[13] libgdx, “GitHub - libgdx/packr: Packages your JAR, assets and a JVM for distribution on Windows, Linux and Mac OS X,” *GitHub*, Mar. 29, 2021. https://github.com/libgdx/packr (accessed Jan. 01, 2026).