Continuous Integration Report

Team 1:
Megan Forrow
Jonathan Broster
Sophie Bailey
Alastair Johnston
Luiz Oliveria
Alina Merkulova
Tameem Abuhaqab

Methods and Approaches

Workflow

With a team size of 7, of which 3-4 are core developers, we decided to use a Feature Branch Workflow**.** This means that the 'Main' branch is always stable, controlled by builds and tests, and other branches are used for development of features and are merged via Pull Requests. On top of this, we have a code review policy where pull requests are reviewed and merged by the primary contributor/designated release manager, creating a Gatekeeper Workflow to minimise the risk of inexperienced members breaking the build.
We decided it was important to automatically build and test the project on each push to main (CI at push). The CI pipeline runs on pull requests before the merge, ensuring broken branches are caught early and don't reach main. We also decided to use a designated integration machine, making managing dataflow and backups much easier, whilst ensuring environmental consistency.

Testing & Validation

Our policy for testing and validation is continuous integration at push, so every push to main triggers an automated build (which also runs the tests). Having a successful build is more important than having successful tests, so if both fail on a commit to main, the immediate priority is to get the build working.
JaCoCo is used to ensure test coverage. However, this is a video game project where visual/gameplay mechanics are difficult to unit test, so we prioritise stability over high percentage coverage. This is automatically run along with the gradle build and uploads the report as an artifact of the build.

Releases

Because of the short-term nature of the project, there will be few releases, and these are done by pushing a special release tag (version number) upon milestone completion. Releases are triggered manually by pushing a version tag which the pipeline detects. Releases are available on the GitHub releases page, under a tag with the version number, the Jar file and source code attached.

Infrastructure We Used

Platform

Github Actions is used as the designated integration machine, automatically building and testing the project at each push to main. It was chosen because it is seamless to integrate with our version control (GitHub), and it ensures environmental consistency.

Pipeline

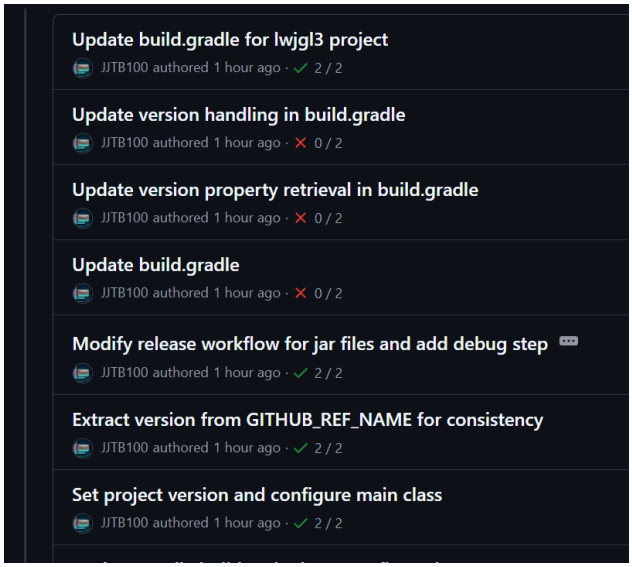
The pipeline configuration is defined by a gradle.yml file inside .github/workflows. This pipeline executes the following steps on a Ubuntu-latest machine:

1. Setup Gradle & JDKs, caching Gradle dependencies for speed of execution.

2. Build the Gradle project and run unit tests.
   run: ./gradlew clean build jacocoTestReport -Pversion=${{ env.VERSION }}
3. Generate artifacts, e.g. the executable .jar file and the JaCoCo report
4. Make a Release (if a version tag exists) and upload the .jar and the source code as assets.
   if: startsWith(github.ref, 'refs/tags/')
   with:
   files: Game/lwjgl3/build/libs/.jar*

Feedback & Releases

When it has finished building and testing, there is a visual status check (tick/cross besides commit title). This makes it clear to us that a test or a build fails, so we can readily organise a fix - the indicators are even integrated into the commit history, so it is simple to identify the specific failed build and roll back if necessary.



The release page contains the version number, source code, brief description and associated .jar file (which is named by the version number automatically):

GitHub also allows us to track pre-release and latest release versions.