# Continuous Integration Report

**Assessment 2**


## Team 10: Hard G For GIFs

Dragos Stoican

Rhys Milling

Samuel Plane

Quentin Rothman

Bowen Lyu

For our game repo there are three different pipelines. A build release pipeline, test pipeline, and documentation generator.

For the test pipeline. It runs our unit tests which it does on every commit to our main branch and for every pull request. This allows us to quickly check and ensure that the added code passes our automated tests and lets us decide on merging pull requests easier. It also produces a HTML report that is available.

The build pipeline produces a distributable build of the game available to download through GitHub's website and runs on every commit on main. This is great since it allows us to play test the game without having to have a development environment setup and ensures that we're always on the latest version.

Finally, the doc generating pipeline runs the Javadoc tool to produce HTML documentation for the Java code. It runs on every commit on the main branch. We implemented this as it helps people understand implementation documentation very clearly as it is formatted very well, with good accessibility, and doesn't require a development environment to browse.

For the website, this was more simpler since it only had to serve a static website, no compiling. Since all work was done on the main branch we only deployed CI on every commit to it. This produced an accessible website on the internet. This was appropriate as it allowed us to focus only on producing content for the website instead of going through the hassle of setting up a server and automating website updates to it.

For our website we used GitHub Pages to serve our website. This action was automated to be performed for every commit applied to the main branch. This was configured to make the entire repository contents navigatable

For our game we used GitHub Actions which built and tested our Java with Gradle. This was triggered on every commit to the main branch and on every merge request. This verified that the application was buildable and passed our automatic tests. The feedback was available either through the dedicated “Actions” tab on the GitHub repository or through the commit history on the repo with the result displayed as a tick for passing the tests and compiling. This has been very beneficial as when the CI does fail it gives a full log of where it failed, either a test or a compilation error.

We choose GitHub Actions since it is very easy and powerful to create continuously integrating pipelines. There is also a large amount of community-made tools available already which has also been beneficial for development.

Furthermore, it provides a build log for every action. Quite helpful as it allowed us to see exactly what commands were executed and their outcomes. So if a build failed a certain test we would be able to identify which one specifically had failed. Having the testing integrated with CI allowed team members to have a simpler local development setup, not having to add new config files. Also, ensuring that we didn’t run into the “works on my machine problem” as each build was done in a new VM everytime which provided a clean environment.