Continuous Integration

Group 21
Generic Games

Josh Thomas
Andrew Palombo
Oscar Gunn
Scarlet Desorgher
Immanuel Ghaly
Madeleine Nielsen

Summary of Continuous Integration Method and Approach

We elected to use a Continuous Integration ‘CI’ approach that is basic (on push/pull request) and partially distributed. We chose a basic approach, that executes the CI pipelines only on push/pull request operations, as our software runs in the Java Runtime Environment on desktops and does not make use of many OS services that would differ across platforms, meaning that building and testing on different platforms would is very unlikely yield different outcomes. Additionally, we did not expect many push/pull request operations to be carried out on a daily basis, so an approach such as scheduled CI is not necessary.

The approach is partially distributed as in addition to the main ‘build’ pipeline, that builds, tests and produces reports and a jar file, there is a ‘macos-build’ and a ‘windows-build’ that do not run tests or produce reports but do produce an easy to install executable. Below is a summary of the CI pipelines used, what they produce and why these are appropriate for the project.

The main ‘build’ outputs the following:

- Test report: to review JUnit tests.
- Executable jar file: to quickly run the game for testing purposes without needing to carry out a build operation locally.
- Checkstyle report: to review the project’s conformance with the Google Java Style Guide for easy readability amongst various programmers.
- Jacoco code coverage report: to verify unit tests’ extensiveness and to limit untested code.

The ‘windows-build’ outputs the following:

- Executable .exe file: a Java application image that contains all dependencies necessary to run the compiled class files (also included). Appropriate for the project as the system needs to be easy to install and run on desktops that may not have the necessary version of Java installed.

The ‘macos-build’ outputs the following:

- Executable .pkg file: a package of the compressed files to install the .app in the Applications folder. Can be run on any machine and works in the same way as above. Apt for the same reasons as above.

Note: input for all pipelines is the latest version of the shared repo with latest changes integrated. Pipelines are all triggered by push/pull request operations.

These outputs are commonly needed and having them produced every time a push operation occurs prompts developers to ensure that code conforms to the expected standards at all times and this prevents a deviation from these that becomes ever more difficult to rectify if otherwise left until the end. Additionally it ensures that code compiles correctly, produces no errors in standardised environments (Ubuntu, MacOS and Windows), not just on a developers local environment, (as the system will be run on many machines).

Report of Continuous Integration Infrastructure

We are hosting our project on GitHub, therefore, it is most appropriate to use GitHub Actions for our CI approach. This is achieved through a YAML file that instructs Actions of what to do, explained below (please view '.github/workflows/main.yml' alongside this).

Lines 3-7 tell Actions to perform this workflow whenever a push or pull request to the main branch occurs.

Lines 10-56 consist of the code to build, test and generate reports of the project in an Ubunto environment.

> **Lines 12-15** setup this job.
>
> **Lines 18-23** setup the appropriate Java Development Kit for this job.
>
> **Lines 25-26** clean the build environment, execute the build and run tests (according to the build.gradle file this also triggers the Jacoco report generation).
>
> **Lines 28-38** upload the test and Jacoco reports.
>
> **Lines 40-47** build the desktop project and upload the JAR file for this.
>
> **Lines 49-56** generator the Checkstyle report for the project and upload it.

Lines 58-104 setup the MacOS environment, build the application for MacOS and package it for distribution in an executable installer.

> **Line 59** sets up this job.
>
> **Lines 61-66** setup the appropriate Java Development Kit for this job.
>
> **Lines 68-69** build the desktop project in the MacOS environment.
>
> **Lines 71-90** decode GitHub secrets that are then used as certificates to verify the authenticity of the application, according to Apple's security requirements. These certificates are then installed to a keychain and files necessary to generate the executable are moved to the same directory. Only now can Jpackage be run to create this executable.
>
> **Lines 92-96** upload the executable generated.
>
> **Lines 98-104** send the uploaded executable to GitHub releases if the pushed commit has a version number tag (indicating release version).

Lines 106-139

> **Line 107** sets up this job.
>
> **Lines 108-114** setup the appropriate Java Development Kit for this job.
>
> **Lines 116-117** build the desktop project in the Windows environment.
>
> **Lines 119-125** move all files necessary for executable generation to the same directory and use Jpackage to generate this executable.
>
> **Lines 127-131** upload the executable generated.
>
> **Lines 133-139** send the uploaded executable to GitHub releases if the pushed commit has a version number tag (indicating release version).