UNIVERSITY OF YORK
DEPARTMENT OF COMPUTER SCIENCE

Continuous Integration Engineering 1 - Assessment 2

Group 2, Cohort 1 ("*The JVMs*")

Ben Hatch
Charlotte Sharp
Dan Nicholson
Ethan Andres
Freddie Higham
Joe Silva
Rosie Kern

CI Methods and Approaches

Throughout the continued implementation of the project, we placed an importance on following good CI practices, including but not limited to; daily/regular commits, automated testing and builds that are quick.

We maintained a single repository for our game, and branches made were short-lived to allow for the development of different areas without conflict. These were regularly updated from main if the work was not completed yet and the work done did not clash with work being developed on other branches.

Taking this approach allowed us to continue work on features separately, while still ensuring regular integration of our code to help prevent merge conflicts. This suited our workflow, as often branches would be created for different aspects of the codebase, i.e. writing tests for previously developed code, and the storage of UML diagrams, and reduced overhead in organizing branches, while ensuring that most code was regularly integrated, maintaining a single source of truth.

While commits were not necessarily made daily, due to fluctuations in work done on the project/schedules, commits were made regularly when needed. Each commit would be checked for formatting, against automated tests, and a build would be generated. This approach worked well as when judging merge requests it was easy to see to a level whether the code was good to be merged. This also prevented the breaking of some code by new changes, as it enabled us quickly to see which new changes had failed a test.

Having a build generated also made it easy to ensure that there were no dependency issues as the build was made on a CI server, it also made it much easier to find the most recent build, which was very helpful for when running user evaluations.

We had one CI pipeline which was triggered on each push/pull request. The pipeline was input with the code of the latest commit on that branch, and output a testing coverage report, and a build, if the code passed the formatting check and automated tests.

CI Infrastructure

For this project, we used Github Actions YAML for our CI pipeline to run our format check, automated tests, and to generate a build and a test coverage report. This was run on every push to any branch, and any time a pull request was created. The pipeline would run on Ubuntu in JDK 11. The format check was spotless in Gradle, which enabled it to be applied with one button by the developer before pushing.

The automated tests were developed with JUnit and Mockito. Our test report was generated by Jacoco. Our automated tests and test report generation were both ran via Gradle, which also generated our build file. Both the test report and build JAR were then uploaded as artifacts attached to each commit.