5. Continuous integration Summary

We chose to employ continuous integration to enable us to automate necessary steps such as running tests after changes to the main branch and building our project into an executable JAR. Employing CI allowed us to repeat these steps in a consistent fashion and reduce the need for personal involvement of team members.

Furthermore, the development of our code involved multiple group members merging code from their individual branches into the main branch. Automated test execution assured us that any issues resulting from erroneous merged code would be swiftly spotted and could be dealt with without creeping in silently into the codebase.

Both Windows and MacOS were used as development machines by various members. Encapsulating the build and test processes in a separate CI environment meant that inconsistencies related to a developer's environment would be guarded against.

Integration Infrastructure

We chose to use GitHub Actions for our CI infrastructure due to its ease of use, thanks to a marketplace of prewritten actions, and tight integration with GitHub, which we were already using to develop our code.

Two separate CI scripts were created:

- <u>build.yml</u> builds the JAR executable and makes it available for download under the Releases section of the repository
- test.yml validates the gradle wrapper and runs tests on the code

Separate scripts were created to visually notify us of the results of each process via the badges on the README displayed in the root of the repository. If any issues were spotted, the could be followed up on in detail by looking at the record of actions, which also held the record of the relevant commit responsible for the success or failure of the code at that stage.

Screenshots

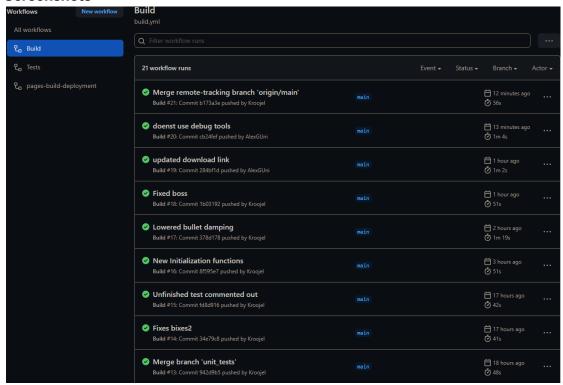


Fig 5.1: Build section

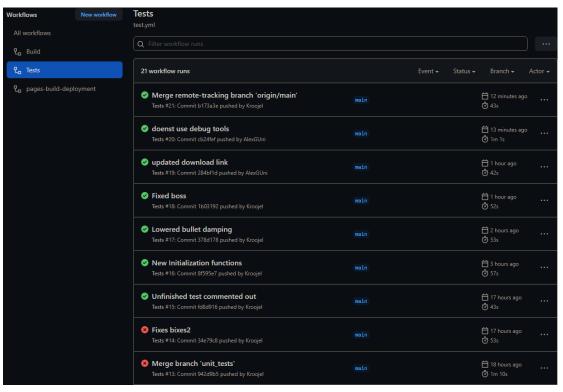


Fig 5.2: Test section