Continuous Integration2

Cohort 1, Team 1:

Connor Blenkinsop
David Luncan
Emily Webb
Muhidin Muhidin(MO)
Sam Laljee
Sooyeon Lee

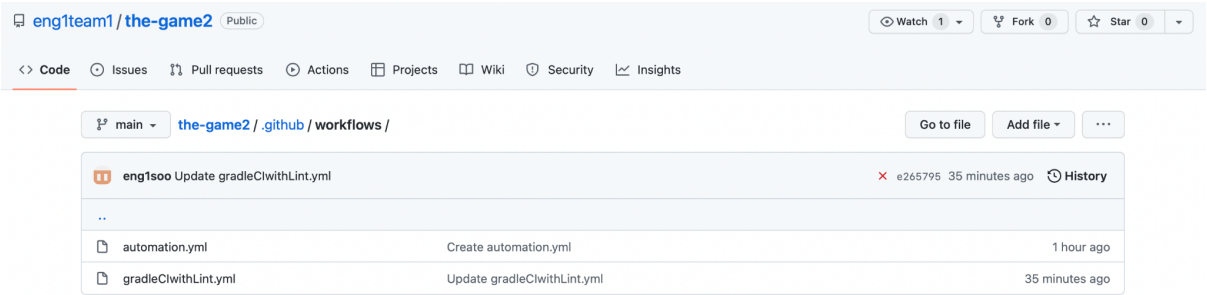
Methods and Approaches

Continuous integration is a software development practice that lets developers merge their code to a shared mainline frequently to minimise conflicts between codes. Instead of committing a major amount of work by multiple contributors at once to the code, committing smaller amounts of code frequently and sharing the progress can lead to increased cohesiveness and ensure the developers aren't working on a completely different page of a project. To achieve this, we took these methods and approaches.

- **Version Control System (VCS)**: We used Git, a distributed version control system to keep track of how the codes change. Using a Github repository, all the code was eventually committed to the main branch under the Git VCS.
- **Automation:** With Github Actions, we automated a few processes of continuous integration. Incorporating automation into workflow enhances the efficiency of team development effort and breaks down complicated tasks within the project. This will be explained in detail later on.
- **Testing & Code coverage**: Software testing is necessary for developers. The best continuous integration practice would be to automate the testing process so that whenever someone commits to the main code, the change would be tested automatically. But instead of including an automated testing scheme to CI, we carried out JUnit and manual testing. This method also informed us of our code coverage status and helped us ensure the code quality.
- **Code linting**: Linting is useful to check the code for its problems and errors. To prevent future contributors suffering from minor bugs and issues that would later be detected upon testing, we added code linting to our selected continuous integration tools.

Continuous Integration Report

The infrastructure of our continuous integration is based on GitHub Actions. Considering that the team's repository is hosted on GitHub, we decided that it would be the most accessible and efficient option for future development. We also considered GitLab CI tools, but comparing the benefits of both GitLab and GitHub CI, the latter seemed to be more intuitive for usage. GitHub Actions also support many templates for Java with Gradle, which are the frameworks we utilised for this game.



Main CI tasks were handled via gradleCIwithLint.yml file, which will perform continuous integration for Java with Gradle and code linting. For code linting, we incorporated GitHub Super-Linter locally to our project. It is a strong automation tool that deals with many languages including Java provided by GitHub's repository itself. We also created a workflow

space that can contain other kinds of basic automation tasks such as greeting and labelling to our CI for successful team projects in the future. Regarding the testing process, as mentioned earlier we did not create automated test build but instead explored other methods. We utilised JUnit to craft unit tests that manage small chunks of code and examined their code coverage. The exact details of code testing and coverage can be found in the Test2 report from the website.