Continuous Integration Report

Galin Dzhumakov

Aisyah Firoz Khan

Faran Lane

Samuel Nicholson

Jack Polson

Alana Witten

Part a)

Continuous integration is a practice where developers create pipelines that automates the integration of code changes in a project. This allows developers to frequently merge the changes in code into the main repository and then check the changes in code with builds and tests. This practice helps to reduce the time taken for developers to check the quality of their code as manually coordinating and communicating with each other while contributing to their end products may cause complex miscommunication and be costly to the project.

The continuous integration method that we chose for our project is Github Actions as it is integrated into GitHub, which is our main platform for this project. The inputs for this pipeline are the java source code files such as the core, desktop and tests folders, the Gradle builds and LibGDX library dependencies. There are several outputs that will be produced from this execution which is a report stating the code coverage, several .jar files, report about bugs and a report about the code style and formats. The output is generated as Java packages and will be uploaded in the build as artefacts that can be downloaded and used to help us in testing and debugging the pull requests before it is merged.

The workflow is created to be triggered when there is a commit and push to the main branch. For example, when a code or a file is committed to the main branch or any other branch specified in the workflow file such as the TestBranch: branch consisting of the unittests. The pipeline will run the build Gradle along with other tools in the workflow to examine the code and produce reports regarding the output of the build. Other than that, the workflow is also triggered when a pull request is created by checking whether the pull request is safe enough for the main. Ideally, continuous integration should have been set up around the start of the Assessment 2 and each member to commit regularly to the main branch but due delays it has been set up late. This causes some issues and problems to arise when branches are merged and pushed to the main branch. However, we managed to solve the issues and get the continuous integration to work fine in our game repository.

Part b)

There are several tests and builds that we have set up for this project. The first one is an automated build that checks the push and pull request to the main branch. Next, a test coverage tool is implemented in the workflow. This is to check how much is the test coverage for the code written by our group. A report containing the details of this test is uploaded as an artefact named JaCoCoTestReport. We also added a binary file that can be downloaded for every version of the implementation. This enables us to check and test certain versions to check and find room for improvement of the code.

We also needed to have an automated build that checks for the code style and formatting is important hence we used CheckStyle. As we are coding with Java, we want the code format to be consistent with Google Java Style Guide so the Checkstyle will release a report consisting of errors, warnings about the code that has violated the rules in the guide above. A clear description of the error or warning is given along with the location and resource of the code with the type in Checkstyle problem.

Lastly, the last build implemented in the workflow pipeline is a build to ensure that the game can be played on multiple operating systems. The three operating systems are Linux, Windows and Mac. The workflow uses the matrix strategy that enables Github Actions to reuse the same template for different values of variables such as for downloading and renaming the artefacts of different operating systems. All of the different operating systems will have the JaCoCo report and binary artefacts uploaded in GitHub Actions which allows users to access a specific version of the software.