Continuous Integration

Group 12

Keira Lauder
Amy Mason
Dan Ho
Sina Khosravi
Yibing Wang

Once individual programs/key features of our game have been tested individually, it was important to be integrated to create a complete system. The continuous integration methods have been appropriate for this project because it allowed each member of the team to own a new code change through to the release. It allowed us to work on an individual issue and/or requirement of the game without worrying whether it would integrate with the rest of the code from other members of the team.

A. We set up continuous integration using github actions. The setup builds the project and creates a jar file every time someone pushes to the main branch or submits a pull request to the main branch. It also runs all the junit tests and produces a test report displaying all the tests and whether they passed or failed, as well as producing a test coverage report using jacoco. The CI deploys these test reports as well as the jar file to our website. This was useful for us because it meant that we knew we always had a jar file for the latest version of the game, confirming it was building correctly. Having it deployed directly to our website was also very helpful as it saved us having to manually change the jar file download on our website every time someone made a small change to the code base.

B. The build.yml file can be found here
The ‘Build with Gradle’ step builds the project, producing a .jar file inside of ./lwjgl3/build/libs in our project. The ‘Deploy jar file to website’ step uses the Github Action ‘peaceiris/actions-gh-pages@v3’ to deploy the jar file created from the build to our website. It creates a ‘jar’ folder in our website repository if there isn’t already one, and adds the jar file to it, replacing it if one already exists.

   The test.yml file can be found here
   The ‘Run Unit Tests’ step builds the project using gradle, producing a ‘reports’ folder inside of ./core/build. This ‘reports’ folder contains a ‘tests’ folder containing the unit test results as well as a ‘jacoco’ folder containing a jacoco test coverage report. The ‘Deploy Test Reports to Website’ step uses the Gitub Action ‘peaceiris/actions-gh-pages@v3’ to deploy the test reports to the ‘testreports’ folder in our website repository. The test results report is produced automatically but in order to produce the jacoco report we added some extra code to ‘build.gradle’:

```
test {
  useJUnit()
  maxHeapSize = '1G'
  ignoreFailures(true);
  finalizedBy jacocoTestReport // report is always
generated after tests run
}

jacocoTestReport {}
```

The ‘finalizedBy jacocoTestReport’ line ensures that the ‘jacocotestReport’ task (which produces jacoco test report) is run every time the tests are run. The line ‘ignoreFailures(true)’ ensures that the project still builds even if some of the junit tests fail. This was necessary as some of our power up tests don’t always pass but we still needed to produce a working jar file.