5. Continuous Integration

Group Number: 20

Group Name: Gourdo Ramsay

Group Members:

Megan Bishop

Tommy George

Bartek Grudzinski

Davron Imamov

Nathan Sweeney

Lauren Waine

a)

CI Server

- The project is built on a CI server using GitHub actions. This happens on a pull or push request to the main branch. If it is successfully built (all the automated steps were successful), then a green tick will appear and if not a red cross.
- By using Github actions, we can ensure it builds consistently and not just on one person's machine. This can then be used as a reference to whether the new changes work as it is easy to see if the build passes or fails. It also helps identify broken builds and to fix them immediately as GitHub actions provides a command terminal detailing what went wrong.
- The CI build runs on Windows, Mac and Linux, if one of these builds fails then the entire build fails and the remaining ones aren't run. This ensures that our project meets the requirement CON_OS, which specifies that the game should run on all desktop operating systems.

Testing

- When the repository is built (on a push or pull request), automated tests are run. If these pass, then the build will pass, but otherwise it will fail and a red cross will appear along with the error on the command terminal detailing which test failed.
- This is vital to our project as we are using the agile method Extreme Programming (XP) [1], which relies upon testing to eliminate errors.

Code Coverage

- When the automated tests have passed, a test coverage report is generated by JaCoCo based on the tests present in the tests folder. This creates a downloadable artifact in GitHub actions for each of the operating systems.
- Code coverage can expose untested code that either isn't being tested and so could generate errors in the program or has become redundant [2, p.243-244].
- It also makes it easier to manage the code as more is incrementally added (a feature of XP [1]) as the code coverage metric will indicate what new areas will now require testing.

Code Style checks

- An automated style check report is generated. It goes through the code and checks if it follows the Google Java standards. If it doesn't, a warning is generated and the error is added to the report, specifying the file, the line and the rule broken.
- As multiple people are developing the code and writing tests for it, having consistent documentation and formatting will help everyone to understand it. We chose to follow the Google Java Style as this was used in our SOF2 module so everyone in the team should be familiar with it.

Version Release

- When the repository is built, a binary artefact of the generated .jar file (core-1.0.jar) is created for each operating system.
- This allows users or other developers to to quickly try out the game (or even previous versions by downloading from previous actions), making it simpler to gain feedback and better understand the customer's needs [1].
- Additionally, a more trusted version of the game is released via pushing a tag.

b)

YAML Workflow file : Runs GitHub actions.

ENG1-Team20-Assessment2/gradle.yml at main · CrimsonLeaves/ENG1-Team20-Assessment2 (github.com)

- This YAML file is used to set-up and run github actions. It runs on a push or a pull request to the main branch and runs the build on a matrix strategy i.e it runs the build on multiple operating systems, one at a time, iterating through a list of them (Ubuntu, Windows and MacOS).
- The file runs through a series of steps that make up the job. Firstly, actions/checkout@v3 fetches the repository, which by default is the most recent commit. Then actions/setup-java@v3 sets up java 11, as per the requirement CON_PROGRAMMING_LANGUAGE, caches dependencies and configures runners.
- Then gradle/gradle-build-action gets the gradle to build the JaCoCo testing report, which shows the code coverage of our tests.
- This report is then uploaded as an artifact for each of the operating systems so it is visible and can be downloaded.
- For each of the operating systems, using the matrix strategy, the created .jar file is retrieved and displayed as an artifact.
- A checkstyle report, which details warnings for incorrectly formatted docstrings, is also uploaded and displayed as an artifact.
- A release is added by pushing a new tag. It downloads the .jar file artifact created by the Ubuntu operating system and adds it as a part of the release.

Test build.gradle: Runs testing and creates testing reports.

ENG1-Team20-Assessment2/build.gradle at main · CrimsonLeaves/ENG1-Team20-Assessment2 (github.com)

- The JaCoCo plugin is applied, this will automatically create a JaCoCo code coverage report for all of the tests contained within the tests folder.
- Testing is added using JUnit. Its specific version is added in the dependencies and JUnit is specified to run the test task. This will then look for all methods labelled with @Test in the folder and run them.

Core build.gradle: Creates checkstyle report.

ENG1-Team20-Assessment2/build.gradle at main · CrimsonLeaves/ENG1-Team20-Assessment2 (github.com)

- Checkstyle and java-library plugin is applied and the version is declared in dependencies so checkstyle can be run.
- Checkstyle then runs style checks based on the Google Java rules, specified by ‘google_checks.xml’.
- By default it creates warnings on the console whenever these rules are broken.

Desktop build.gradle: Creates the .jar file.

CompletedDishStation Test · CrimsonLeaves/ENG1-Team20-Assessment2@ab10a44 (github.com)

- The ‘Java’ plugin is applied and the .jar file is created in the dist task.

References:
[1] GeeksforGeeks (2023, Feb. 15). Software Engineering | Extreme Programming (XP). geeksforgeeks.org. [Online]. Available: Software Engineering | Extreme Programming (XP) - GeeksforGeeks [Accessed: 10 March 2023].
[2] I.Sommerville, Software engineering Pearson Education, 2015