Continuous Integration Report

Megan Miles
Alistair Foggin
Kajol Dodia
James Wild
Isselmou Boye
Matt Fitzpatrick

Part A

We implemented our Cl immediately as CI is an essential practice for software development projects, and it has helped to ensure that the code remains stable and reliable throughout our development process. It also helped to improve collaboration between team members, reducing development time.

We have integrated CI within Github where we have 2 different Github actions. We have it automatically unit test whenever a pull request is created or a commit is pushed to the main branch. Before a pull request can be merged, all unit tests have to pass. This is appropriate for our project as errors are quickly detected and can be fixed before changes in code become too different to integrate. The second bit of CI is for every commit to main,
the whole game is built and compiled into the resulting jar file.

It was appropriate to use GitHub for our project as other standalone automated tools, such as Jenkins, CircleCI, or Travis CI would be too much power for the code we have. Also, as we have already been using GitHub for our repository it is very convenient to use GitHub integrated CI for our small codebase.

Some of our methods for CI included rules that every team member followed. Such as for every new function created a new test would be made. Another method was that every team member would commit every afternoon when they were working on the code. This is to ensure that commits were frequent meaning that errors could quickly be identified.

As one of the original requirements of the game was to be able to play the game on the university computers. The game would need to run on multiple operating systems. So we ran the build on the different operating systems making sure it worked.

We only maintain a single repository so that every team member could see what was happening. We tried to work on the main branch as much as possible, any new branches should not be alive for not more than a few days. As if the branches were active for longer the branches will be harder to merge to the main branch.

If any errors were found we would fix any bugs as soon as possible. We would revert the changes so that we had more time to work on the bugs and other team members could still collaborate on the project.

All of our builds took less than 10 minutes thus our tests were fast enough to provide rapid feedback to the team. And there was no need to spend time and resources on parallelisation. This was advantageous for the project as time was a limited factor.

We did not use checkstyle as our code was too small and we wanted to focus more on the functionality. With more time and a greater scope of the project, we would have implemented it.
Overall, all our methods ensured that CI was carried out effectively to detect and address issues early in the irritative process, reducing the risk of larger problems occurring later on.

Part B

For one of the continuous integration pipelines we implemented it to ensure that all JUnit tests pass with Gradle which is carried out for every push or pull request that is completed on the main branch. It is given the permission to read the contents of files so the tests can be run and it is also given permission to write to say whether each test has passed or failed. It is then set up to run on the latest version of Windows, macOS and Ubuntu for Linux in case there are any inconsistencies of features in Java across the operating machines and then ensures that it is using JDK 11 as that is the version we used in our development. It then runs ‘Build with Gradle’ with the arguments test which is a function from the ‘build.gradle’ file in the tests folder to run the tests. After that it runs ‘Publish Test Report’ which creates a report with all the test results from the project to the file 'tests/build/test-results/test/TEST-*.xml', which will continue until all tests have been reported whether they have been successful or have failed.

For the other continuous integration pipeline we implemented it to build the JAR file of the game with Gradle every time we push onto the main branch. This pipeline only requires reading the contents so that the code for the game can be compiled, no writing is required. We then run it only on the latest version of Windows, as that is what most commonly will be used and the JAR file can then be used on any available operating system. It again uses JDK 11 as that was what we used in development to ensure consistency. Then it runs ‘Build with Gradle’ with the argument desktop:dist which is in the ‘build.gradle’ in the main folder and then uses the dist from the ‘build.gradle’ file in the desktop folder which creates the JAR file so it can be distributed. Afterwards the pipeline will run ‘Upload game JAR’ which uploads the build of the game to the path ‘desktop/build/libs/desktop-1.0.jar’ with the name ‘piazza-panic’.