Continuous Integration

ENG1 Team 4

Isabella Miles
Maciek Racis
Sally Finnon
Sophia Eaves
Wrijurekh Mukherjee
Xavi Murtagh Molina

Methods & Approaches:

For our project, continuous integration was a key part of the process. Though the actual code infrastructure was only developed later due to time constraints, we used the principles and practices of continuous integration throughout development. Ensuring that we caught errors early was incredibly important, due to our changing requirements and strict time constraints. We also found that this method was useful for keeping communication open between the coding team, testing team and continuous integration team – preventing us from missing tasks, and decreasing the risk of intra-group arguments.

When we were provided with the new code for the second half of this project, we began to make use of these processes immediately. Iterations were made to the code regularly, with our coders committing to the main branch at least once on each day they were working. This allowed them to check each other's work, while also making it easier to keep the corresponding tests and continuous integration infrastructure up-to-date. These other groups, who made less frequent updates on the code, also made commits often while working, though this did not exactly correspond to the minimum of once per day. The regularity of the updates, including dated daily updates from our coders, can be seen clearly in the GitHub repository used for the second half of the project.

Once the formal infrastructure was added to the code, during the Christmas holiday period, we made use of this to further improve our processes. After each commit, we checked the report provided by GitHub actions, identifying errors in the code or tests and updating them where needed. This addition to our process removed the element of human error, as well as specific features of the coders' or testers' computers, which allowed us to be more precise and efficient throughout. Unfortunately, due to time constraints, we did not get to make full use of the JaCoCo reports or versioning systems provided in our infrastructure.

We also did one final test of our program, in the group meeting before submitting the entire project in the January exam period, ensuring everything was committed to the main branch where needed. This should allow us to remain on the same page for both the submission and the upcoming client presentation, as well as allow us to make updates to our documentation where needed.

Integration Infrastructure:

Our continuous integration infrastructure is similar to that recommended in the lecture – as such, this will be a brief report and focus on the changes or updates made to the lecture version of the code. These include updates to the versions of various systems and actions, as well as sections removed or altered due to practical limitations. It is also worth noting that, as timing issues caused our continuous integration to be implemented later, we had the benefit and drawbacks of a largely developed piece of software when creating it.

In order to ensure our code works on a variety of systems, we gave our continuous integration system the capacity to create virtual machines in various operating systems. Specifically, we use MacOS v14, Windows 2024 and Ubuntu 22.04, as these are more recent OS versions recommended by GitHub. The code then sets up both Java & Gradle on each of these machines, as those are used throughout the project, and builds in the game itself using Gradle. Some issues were encountered here due to Gradle not having access permissions by default, but those were corrected using a Git command on a local file, which was then pushed to the GitHub repository.

The primary purpose of continuous integration, of course, is to test our program in these various environments. Our code uses the tests developed by the testing team in the .idea file. These tests cover all the program's functionalities and specific requirements from the specification, and the continuous integration workflow does not succeed unless all tests are passed. The originally added environment, using Ubuntu 22.04, is the one on which both others rely – if it fails, both of the other sets of tests are cancelled immediately. Also in the Build part of this, we use JaCoCo to provide a written report on these tests and how they went, a useful resource for our coders and testers in determining what happened.

Our recommended infrastructure, provided in lectures, also included Checkstyle to ensure the code met specific style guidelines. Due to the difficulty of integrating and enforcing this late into a project, particularly when our code already has a clear and cohesive style, we chose to eliminate this section of the build altogether.

As well as building the game in virtual machines, the continuous integration code can also release the game with specific version tags. This allows notable changes in the code to be marked for our users or hypothetical clients, and allows us to get more rapid feedback.

This checking process is automated whenever something is pushed to the main branch, so there were no group systems in place for ensuring it was run and used. We were simply able to check the GitHub Actions tab after making changes, to check if the code was currently up to our testing standards.