Continuous Integration Report

Cohort 2 Team 1 (Assessment 1)
Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Cohort 2 Team 2 (Assessment 2)
Bader Albeadeeni
Dan Hemsley
Jennifer Bryant
Mathilde Couturier-Dale
Oliver Elliott
Rosie-Mae Connolly
William Mutch

Continuous Integration Report

Continuous Integration, for our project, meant a few things. Arguably the most important was making frequent integrations that all functioned as working prototypes of the game. Each of the people working on the code had their own branch that they could commit changes to that were separate from each other and the main branch. Each person would create a pull request daily, or as often as possible to merge the changes they made with the main branch. We then made sure that someone else was in charge of approving the pull request and resolving the conflicts so that there was a second pair of eyes to look for errors or mistakes. Each time someone pushed or made a pull request, automated tests would run to ensure that the code could be built.

These CI methods suited our project for multiple reasons. The first is that we had a team of people working on the code, so CI helped prevent conflicts when merging. We were developing a game, so catching any changes that broke the game as early as possible was crucial. And we utilised sprints, which were well suited to the fast feedback that CI provides. CI was also useful in making sure that the project was built for different operating systems.

Therefore, through Continuous Integration, we saw fewer integration issues, a more stable main branch, faster debugging, more efficient communication and more confidence in each prototype.

We set up a few tools to help us along with the continuous integration. One of these was GitHub Actions. We set up the yml file so that it triggers every time that we push, pull request or push a version number tag (v0.0.0) to main. Then it would build the code for different operating systems, Linux, Windows and macOS. This job uses a matrix strategy, meaning the same steps are run on each of them. The yml file also generates artifacts, such as a JaCoCo coverage report for each operating system, so we can see how well our tests are working. Then the Dependency Submission job is performed, which uses Gradle to generate a dependency graph. This dependency graph is then used by GitHub to detect vulnerable libraries, suggest updates and track dependency changes over time.

We also wrote unit tests for all of the classes, which meant separating the logic from the rendering and the input of the classes so that they could be tested. Things like sprites and sounds and rendering cannot be tested by unit tests. We set these up using Github actions to make these run whenever there was a push or a pull request. In the JaCoCo report, it states that we have a 91% coverage of the tests. We used this throughout development to find untested areas.

Upon reflection, we began the coding before we had set up the continuous integration, and set up some parts of it quite late on in the development. This meant that the continuous integration was not fully present all the way through and we felt this impact as we set it up. If we were to do this again, we would make sure that we planned ahead and have the continuous integration set up from the start, to help with the development process.

References

Continuous Integration Report

Garcia-Dominguez,A.(2025).Continuous Integration. Department of Computer Science, University of York