Continuous Integration

Team 16

CatepillaDevelopment

Yousif Al-Rufaye
Joe Fuller
William Gracie-Langrick
Jack Hardy
Bailey Uniacke
Ben Young

a) Continuous Integration Summary

In our project we used multiple methods of continuous integration to make sure that our code avoided errors, and if they were created, we were able to detect them and resolve them quickly after any changes were made. Our team used a version control system, where we hosted all of the code for our game. We made sure that the people working on the code were communicating frequently on what they were adding.

When working on new features, separate versions of the code repository were made. To give a safer environment for code changes and experimentation. We made sure that branches of the repository never got too big, or were too ahead of the main branch, and made sure to re-integrate our code frequently, where the repository copy was deemed acceptable. To catch and fix any small issues immediately before they eventually grew to be much more resource consuming to resolve, and prevent any potential issues from occurring during a large merge.

Testing constituted an important part of our continuous integration process. Testing was carried out both on individual repositories and also when making a merge to our main working repository, tests were made to ensure that the new features did not interfere with any changes. Most tests were automated so could be run quickly and after each change. However, some tests could only be checked manually so they were carried out during bigger changes and merges.

b) Continuous Integration Infrastructure

The infrastructure we used was mainly implemented using Github's online systems. Github implements git version control which allows for local and main repositories. Where the main is hosted and changes are committed to it. Each commit also contains a commit message which allows us to communicate with each other what changes have been made. And for users to see the information when it is applicable to them

Github also allows branches which act as separate versions of the code repository. These can easily be merged, and github will automatically detect some dependencies or problems with the merge and highlight them for you to make integration easier.

Automatic testing on merge was partially implemented by github actions on our repository. Github actions allows for certain files to be run when some action is made on the repository, such as a merge. We were able to get it to run a Gradle build, but it failed when attempting to run our tests as it could not detect a Gradle wrapper. This meant testing had to be run after merging instead, which was less productive.