Continuous Integration

Mario

GROUP 28

Joseph Frankland

Anna Singleton

Saj Hoque

Leif Kemp

Shi (Lucy) Li

Hugo Kwok

<u>Part A</u>

Continuous Integration plays a huge role in improving the quality and validity of our code whilst also improving the efficiency of the process as a whole. Before researching the different possibilities of CI infrastructures and which would be most appropriate for our project, we first planned our methods and approaches towards it and took many considerations towards this decision.

Our approach to Continuous Integration automates 2 important things, which are automated testing and automated building. The next thing was to consider how often these automated tasks would be performed. When adding features or making changes to code we followed a process of committing to other branches from which we would make pull requests into master. We would not directly commit to master, unless it appealed to our rule of very small commits which usually only included fixing typos or cleaning code commits. Once a pull request had been received, our Continuous Integration would come into act, performing a CI test on the proposed change whilst also testing it on different platforms - specifically Windows, Linux and Mac OS. We decided as a group that this approach would be most beneficial and endurable if we kept these CI tests limited to when making pull requests as this ensures we are being efficient and not wasting time with unnecessary CI tests and also is reliable because of our previous approach towards making changes to our code (avoiding committing directly to master).

These CI tests came to be quite significant in finding issues that were not obvious. For example, when performing a commit on the entity ship unit test, it failed the ci test - this failure did not come up on the developers system however did on master suggesting it would not work on anyone else’s system which would be deemed impractical. Preventing issues like these are where Continuous Integration proves to keep our code reliable and prevent errors from going by unnoticed. The results from these CI tests are closely looked at when they are passed as errors - this is seen as when a CI test fails, it will just upload a standard error output, also known as a stderr. This will allow us to take any action appropriate or amend any changes made to ensure the validity of our code.

There are other methods of checking the status of our CI tests which are usually presented as reports given by your chosen CI infrastructure. This approach would be better suited to overviewing the results on a broader view, whereas the stderr would be more appropriate when working side by side with the implementation. In terms of approaching automated building, we established a methodology of automatically building new releases of the game and putting them onto the releases tab on our github - this would be concluded once either a merge or small push to master was accomplished with all CI tests passing. This approach was appropriate as it allowed us to keep up automation of the project whilst allowing several versions of the game to be built and released that were fully functional due to the CI tests. It also gave us a form of reverting back to previous releases provided there is a significant change that causes us to do so.

Part B

For the actual implementation of our continuous integration we decided to use GitHub Actions. This was due to the fact that we housed our project on GitHub and we came to the conclusion that it would be very easy to set this up due to being familiar with the GUI. The comfort of having our code repository and CI in the same place discards the need for an external site for our CI - this feature would allow us to integrate our testing and builds directly into our GitHub Repository, providing us with the ability to manage CI activities in the same place as all our other repository related features like pull requests and issues. Other tools were considered, such as jenkins and circleCI, however, these tools were excess to the smaller requirements of testing for this project, having features that we would deem to be unnecessary and distracting for the level of our project and hence were swiftly dismissed as a possibility for our choice of CI. Furthermore, setting these two tools up would require a much longer amount of time and would harm progress in other areas of the project. So, overall the ease of use and set up and how it fit all testing requirements were the most important factors on what we chose to use as our CI.

We set GitHub actions to perform two key tasks, automated testing and automated builds. These actions are set up by a couple of files in the workflow folder of the repository. One file to set up how the automated testing works and one to set up how the automatic build happens. The first yml file is the one that automates the unit tests for the project. The first part of the file specifies when to do the next part of the task, in our implementation it is when there is either a pull or push request to master. When this condition is met, the rest of the file will be executed. The first step in this is to set up the tests on three different operating systems, these being Windows, Linux and MacOS. Once this has been done jdk11 and Gradle will be set up so that the tests can actually be run. Getting the tests from the test folder in the repo is the next stage, with these the unit tests will be run on all the three previously mentioned operating systems. Finally, once all the tests have been run a test report will be generated and output to a directory within the tests folder and under the build you're trying to test. There's a similar process for the automatic building of the game. Once pushed to master jdk11 and Gradle will be used to generate a jar file of the game. Using this jar file the build will be put on the side of the repo so people can easily download the latest version of the game. If a test fails the pull request is unsuccessful it will come up as a failed check on the GitHub page, as well as this it will output the stderr so you can more effectively debug your code and solve the problem quickly, rather than having to change things slowly and testing one by one.