Continuous Integration

5.a) Methods and Approaches

Frequent Integration

This refers to regularly merging code changes when working in teams into a shared repository. This process will enable the team to identify and fix issues early, reducing the likelihood of costly errors and improving the quality of the final product. By continuously integrating small code changes, our team can also improve their ability to collaborate, share knowledge and stay aligned with requirements.

This will be an appropriate method for our project as frequent integration is a key component of Scrum, which emphasises iterative and collaborative approaches to software development. Overall, it can lead to more efficient, high-quality software development processes.

Automated Build

To ensure all commits to the main branch don't break anything we use an automated build. As a team we discussed this and decided it was essential as it will allow builds to be done automatically, saving time when implementing features, as any push to the main branch will trigger a build action. Also using the automated build we can run tests automatically ensuring changing and adding features doesn't break any test and if they do the bug can be found easily.

This will be appropriate for a project as we have a few programmers working on the code and if one makes a commit causing the game to be unable to build, progress on the game will be impossible until that bug is fixed. With automated build and testing we can ensure that every time code is pushed that the game will be able to run and feedback is given in a timely manner to the person pushing it.

Code Style

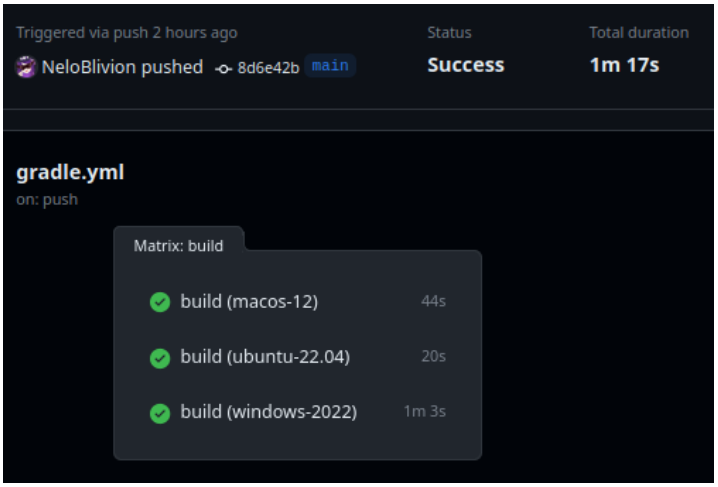
We also discussed having a code style to stick to, this will make the code consistent, readable and make it more maintainable. Code style checks can also be done automatically along with the build and testing so will be easy to implement and increase the quality of our code. This in turn will speed up development and new features will be added in seamlessly.

5. b) Continuous integration infrastructure

Frequent integration

Here on github, you can see regular commits ensuring that progress is made on the game. When a new feature or rework takes place developers are encouraged to create a new branch and then merge to main after it functions as intended. This is important as many people can now add features at the same time, increasing workflow and less merge errors. Some care must be taken when merging branches to main as this may break some other aspect of the game, however using automated build and testing this shouldn't be too big of an issue.

Automated Build



Using github action we were able to create an automated build gradle.yml file. This ensures that each commit the game will be able to build. Here is an example of everything passing. As you can see it is configured so it can check if it builds on all platforms. If something were to fail the person pushing would be notified and they can fix the issues.

In addition to this I would be good to have action for automatic tests that run on code that has been changed as well as creating a .jar file for each different platform. We have been unable to implement this though at this time however it would make production of our code easier.

Code Style

The code style we chose to use Google's Java Code Style, this was one we were most familiar as was used by the previous team. We used camelCase for variable names and capitalised the first letter if it was a class name. We also used descriptive names of classes and added on the class they inherited from such as PlayScreen inherits from Screen ect. Also no line breaks before open brackets, or closing brackets.