Continuous Integration

C3 Group 6
Team WHNI

Aaratrika Das
Arkoday Roychowdhury
El Foster
Hanna Pieniazek
Lewis Keat
Matthew Baghomian
Matthew Ford
Subham Magar

Summary of Continuous Integration Approaches/Methodologies

We have utilised a variety of Continuous Integration (CI) techniques in our project. These are designed to ensure that development of the software can progress without hindrances or conflicts. A summary is given below.

Repository
We used a shared code repository to ensure that all implemented code was readily available for all members of the team. A repository allows contributors to work on the same project with minimal conflict, so long as it is maintained correctly. Additional branches can be created, changed, and merged back into the main branch to allow for synchronous development, though these should be short-lived.

Automated Build
We created an environment to simplify the build process of the game. By utilising this technique, contributors do not have to manually build the game, ensuring that they are not wasting time and resources on the same repeated action. We also created automated testing within the build, allowing us to quickly identify and handle errors in the software.

Minimise Build Time
We also ensured that the automated build would not take an excessive amount of time to finish, as this could mean that problems are not discovered before another commit, and therefore another build, is made. In our case, we decided that the build should take at most 10 minutes to complete in the worst case. This was achieved through a CI technique called pipelining, which is a method of dividing the build process into multiple different processes. The first build process must be kept short, while subsequent builds can run for longer. To achieve this, the first build should only contain the necessary tests for the program to function; subsequent builds can test alternative cases that may take longer, but are not required for other contributors to begin working on this build.

Automated Deployment
We created scripts to allow for automated deployment of the game. This meant that the latest release was always readily available in a clearly identifiable location, and ensured that releases did not need to be created manually.

Consistent Builds on All Platforms
In addition to automated builds, we also ensured that all commits & changes to the project were tested on all of our target platforms: Windows, MacOS and Linux. This ensures that there is no change in execution between different contributors' devices, and creates a central designated machine to use as a reference for the current state of the project.

Test in a Cloned Environment
To avoid making any changes to the main branch, testing was completed in a separate environment to the main codebase. This was updated frequently to ensure that tests for new/changed features could be created as soon as possible.

Continuous Integration Structure

The specific software/environments used to implement the Continuous Integration methodologies listed above are detailed below.

Repository - Github
We have utilised Github to host our repository, due to its various methods of streamlining the process and its ease of use and integration. Creating pull requests and committing code is simple and efficient, and anyone can access the latest version of the software by simply clicking a link.

Automated Build & Release - Github Actions, Gradle, YAML
Github provides a straightforward automation framework through Github Actions, which allows for a variety of methods to facilitate this automation. In particular, we have used YAML (Yet Another Markup Language) to create automated Gradle builds for our project.

Github Actions hosts CI servers simulating different operating systems, and allows scripts to be run on them. Therefore, we could test each new commit on every operating system as soon as the commit was made, allowing us to identify any platform-specific errors quickly.

Automated Deployment - Release Script
Github Actions also allows us to automatically deploy a new release when a commit is made with a specific tag; this tag must be in the format dictated by semantic versioning: “vX.Y.Z”. X, Y and Z are each digits between zero and nine; X represents major version change, Y represents minor version change, and Z represents a patch where a bug has been fixed.

Minimise Build Time - Github Actions Pipelining
To implement pipelining, we designated the build and release processes as separate jobs in Github Actions; the release job is only executed if the relevant commit has a version tag, and cannot run until the build process is completed. This means that contributors do not have to wait for the release process to finish before they can begin development on the most recent build.

Test in a Cloned Environment - Testing Branch
A separate branch was created in the Github repository that branches from the main codebase to facilitate internal testing. As described in the summary, this was updated frequently to ensure that new features from the main project were received; tests could then be written for these new features without affecting the main project. Once test writing was completed, the testing branch would be merged into the main branch.