Continuous Integration

Hannah Thompson
Kyla Kirilov
Ben Hayter-Dalgliesh
Matthew Graham
Callum MacDonald
Chak Chiu Tsang

1.1 Continuous Integration Methods and Approaches

In our project, we utilise continuous integration by incorporating Martin Fowler's 'Practices of Continuous Integration' [1]. The methods we applied and their specific use in our project are outlined below:

Put everything in a version-controlled mainline: We set up a single source repository in GitHub to contain all source code, assets, libraries, configuration information and any other files needed to build the game besides the final executable file (including this would indicate that our project could not reliably recreate builds). GitHub maintains a version-controlled structure for the project, which will prevent serious errors from occuring during integration. The structure of the GitHub repository makes it clear which branch is currently being worked on, with the mainline being called '*main'*. All the files within the repository are in text format, as this makes it possible to easily track the differences between them.

Automate the build / Every push to the mainline should trigger a build: The build process is automated to avoid human error when compiling, moving files and loading data. To facilitate this, instructions for the build will be included in the repository, and we will use GitHub Actions to construct workflows that automatically build the project every time a change is committed. If the build fails, this will tell us that the error was in the latest commit, which will help us to resolve it quickly.

Make the build self-testing: As manual testing would significantly slow down the continuous integration approach, the build process includes automated testing. When each team member commits a change to the mainline, they should also commit a test to verify it. A failed test indicates the error occurred in this integration. It is also important to frequently test dependencies on third-party modules, as updates to these modules could make the code behave in unexpected ways.

Everyone pushes commits to the mainline daily: The key rule within our team is that every member must commit to the mainline daily (as long as it passes the build and test), and they must fetch from the mainline whenever they begin to work on a new feature. Enforcing frequent commits keeps everyone up-to-date, preventing issues arising from multiple people changing the same part of the code.

Fix broken builds immediately: If a build in our project fails, it becomes the highest priority objective to fix it. While this doesn't require all team members to work on the fix, no further commits should be made until either the issue is resolved, or the code has been rolled back to a previous working build. This way, some members can fix the issue while others can continue working on the mainline.

Keep the build fast: A project that takes a long time to build would both negate the aim of continuous integration (immediate feedback) but also waste the developer's limited time. As defined by Martin Fowler [1], and in order to make significant progress while frequently committing and with automated building and testing, our builds should not take more than 10 minutes.

Automate deployment: Automated deployment makes it easy to deploy the latest version of the game and allows us to rollback to a previous working state if errors occur. We will use GitHub Releases alongside our repository to track the version number and version control logs for each release.

The use of these continuous integration practices in our project aligns well with our agile approach by enabling us to rapidly integrate the results of our separate sprints. Identifying integration issues early-on in the development process reduces the likelihood of larger problems emerging later on, and gives us quick, real-time feedback through the automated building and testing of our code.

2.2 Continuous Integration Infrastructure

We used the techniques described above to inform planning and decisions about our approach to continuous integration. The actual infrastructure of our continuous integration pipeline is described in detail below:

Overview:

We ensured that every push (at least once per active team member per day) was handled by GitHub Actions to automate the build, testing and deployment. This confirms that every push to the mainline is automatically built and tested as well as generating a dependency graph for our project - the workflow for this process in the YML file is as follows, and the YML can be found on our website under ‘Assessment 2’.

1. **Pipeline Configuration**
   The pipeline is triggered on every push to the main branch and on pull requests from the main branch. This step takes the source code from within the repository and the configuration files for building and testing, and checks out the latest code from the repository. It then sets up the JDK using Amazon Corretto (the same as is used on our local machines) and sets up Gradle, ensuring that it is executable. Finally, the build is run which generates JaCoCo test reports.

2. **Dependency Submission Pipeline - Generates Dependency Graph**
   The pipeline runs after the build and test pipeline and generates a dependency graph. It takes the source code from the GitHub repository and checks out the latest code, setting up the JDK as before. Finally, it generates and submits a dependency graph to enable Dependabot alerts, and outputs the graph.

   Dependabot [2] is a service provided by GitHub that automatically and continuously scans the repository for dependencies between files. It checks any dependencies found against a known database (GitHub Advisory Database) which contains information about security vulnerabilities. If one such vulnerability is found, Dependabot generates an alert within the GitHub platform. This is particularly useful in our project as it helps to ensure the security of our project automatically, so that we don’t have to manually check for vulnerabilities after each push/pull to the mainline.

3. **Generating Test Reports**
   We added further instructions to the YML file to automatically generate test reports within the repository after each trigger (push/pull to the mainline). This ensures that any changes to the mainline are quickly reported on, such that if errors occur, a team member can easily locate the source of the error, which tests it failed, and determine how it can be corrected. These test reports can be viewed on our website, under ‘Assessment 2’.

We backed this pipeline up with other good practices, including enforcing regular pushes to the mainline (at least once per active member per day) and ensuring that as much as possible, pushes were made to the mainline rather than to branches that would later need to be merged.

References:

[1] M. Fowler. (2024, Jan. 18). *Continuous Integration* [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration [Accessed 28/04/2024].

[2] “Dependabot,” *GitHub*. https://github.com/dependabot [Accessed 19/05/2024].