Continuous Integration Report

Group 1 Cohort 1

Tom Haslam, Christopher Oulton, Charlie Piper, Shirin Sitara, Dillon Anthony, Kevin Thomas, Ella Daramola

Continuous Integration (CI) is a software development practice that involves automatically building, and testing, and deploying any code changes that are committed. This includes changes from multiple developers or branches which share a single repository as the builds and tests are run there. CI is a crucial component of our development process which makes sure our project is always deployable, functional, and stable. Below is a summary of our current CI implementation and the infrastructure we have set up for our project.

Continuous Integration Method and Approach:

We used Git as our version control system, because of its widespread adoption, and its capabilities in managing changes to code across multiple developers. And GitHub to host our repository, because of its collaboration features and CI integration. Our CI pipelines were defined and executed using custom YAML (.yml) configuration files within GitHub Actions.

In addition to utilising Github and Git for version control, we adhered to the practice of conventional commits. This is a convention for commit messages, making it easier to understand the purpose and scope of each change to the codebase. This is typically in the form of something like “feat: added Character selection screen” which translates to the character selection screen was added as a feature.

This setup allowed us to integrate our code changes seamlessly, automate builds, run tests and deploy our code directly from our GitHub repository (in the form of a releases page) along with a version history.

Key aspects of our CI approach include:

Automated Testing:

We have implemented a suite of automated tests, that should cover a high percentage of our code base. These include unit tests, integration tests and end-to-end tests. These tasks are executed automatically within the CI pipeline upon each commit to the main branch or a merge into the main branch, providing immediate feedback to developers.

Continuous Deployment:

The pipeline is also configured to automatically build and deploy successful builds to a GitHub releases page, allowing us to validate new features or fixes in a real-world setting before fully deploying the game. This ensures that only thoroughly tested and validated code is made into the final game. This also makes sure that when adding to the codebase, the code still successfully builds a valid game.

Our Continuous Integration Infrastructure

1. GitHub Repository: Our project's codebase is hosted on GitHub, enabling version control and collaboration among team members who work on features in separate branches, and then when a feature is successfully implemented it is then merged into the main branch.

2. Conventional Commits: We adopted conventional commits to standardise commit messages within our group. With this standardisation, each team member can easily understand the purpose and scope of each code change, making the commit history more comprehensible. This is typically in the form of something like "feat: added Character selection screen"

3. GitHub Actions: We utilised GitHub Actions as our CI/CD platform, configuring 2 automated workflows within a YAML file.

4. YAML Configuration: A custom YAML (.yml) configuration file defines our CI pipelines within GitHub Actions. This configuration allows for our automated building, testing, artifact generation and continuous deployment.

    -The first part of the file defines that this workflow is run on a push to main or a pull request to main.

    -Then the first job is to build which attempts to build the current libGDX game on 3 operating systems - Windows 2022 server, MacOS 12 and Ubuntu-22.04

    -This then builds the game on a Gradle wrapper

    -It then also creates JaCoCo coverage report as a workflow artefact

    -And creates a checkstyle report showing if it built correctly

    -And if it builds correctly, the release job is started which uploads the executable jar file under the releases page on our repository, alongside a small message that states features or fixes in the current release.

5. Automated Building and Testing + Artifacts: Once a commit is pushed to main or if a branch merges to main, the entire game attempts to build. If the game correctly builds, then another workflow is run which automatically runs all the tests for the game and generates a JaCoCo test report, which are archived for future reference per individual commit.

6. Continuous Deployment: Successful builds are automatically deployed to a GitHub releases page, allowing us to validate new features or fixes in a real-world setting before final deployment.