Continuous Integration 2

Cohort 1, Group 6

Group Members:

Hussain Alhabib

Ellen Matthews

Minnie Poon

Jason Ruan

Daniel Smith

Owen Smith

Our Chosen Continuous Integration Methods and Approaches

Our chosen Continuous Integration approach utilises GitHub Actions, a CI tool well-suited for GitHub-hosted projects, like ours, due to its seamless repository integration and YAML-based customisable workflow. We did consider other options such as Jenkins, CircleCI, and TravisCI, but GitHub Actions stood out for its simplicity, version-controlled workflows, and lack of dependency on external services. This choice aligns with our objective of creating a robust and maintainable CI framework that ensures consistent builds, high code quality, and efficient development cycles.

Key Features of Our CI Pipeline

1. **Multi-Platform Testing:**
   The CI pipeline is configured to run across multiple operating systems - Ubuntu, Windows, and macOS - to ensure cross-platform compatibility for our LibGDX project. This is crucial for maintaining consistent functionality, as LibGDX inherently supports multiple platforms, and it is a key project requirement.
2. **Trigger Mechanism:**
   The pipeline is triggered when the following occurs:
   - A commit is pushed to the master branch
   - A pull request is made against the master branch

   This setup ensures that all changes undergo thorough validation before being integrated into the main code.
3. **The Pipelines Stages:**
   1. **Environment Setup:**
      - Checks the code
      - Configures Java 17 with Gradle caching
      - Caches Gradle dependencies specific to each OS
   2. **Build and Analysis:**
      - Builds the project using Gradle
      - Runs Checkstyle for code quality
      - Executes unit tests and generates a code coverage report using JaCoCo
   3. **Artifact Management:**
      - Uploads Checkstyle and JaCoCo reports for review
      - Compiles the project into an executable JAR using Shadow JAR
      - Lists the JAR files for verification
      - Uploads the final JAR as an artifact for distribution or deployment

Why is this approach appropriate?

Our CI methodology is appropriate for our project as it supports rapid development cycles, facilitates early detection of issues, and maintains high code quality standards. By validating changes across multiple OSs, we ensure compatibility for our cross-platform built with LibGDX. Furthermore, GitHub Actions simplifies the process by tightly integrating workflows with our repository.

This approach effectively supports our project's goals of maintaining a seamless development workflow and deliverable a stable, high-quality product across multiple operating systems.

Our Continuous Integration Infrastructure

Our continuous integration (CI) infrastructure is implemented using GitHub Actions, integrated into our GitHub repository under the `.github/workflows` directory. The primary workflow file, `ci.yaml`, automates the CI process, including the build and test stages across multiple operating systems.

Triggers

The CI workflow is triggered on every push to the master branch and on pull requests targeting the master branch. This ensures that all changes undergo validation before being merged, maintaining code integrity.

```
on:
 push:
   branches: [ master ]
 pull_request:
   branches: [ master ]
```

Job Matrix

A matrix strategy is used to execute jobs in parallel on Ubuntu, Windows, and macOS, with all jobs configured to use Java 17 as per our project's requirements.

```
strategy:
 matrix:
   os: [ubuntu- latest, windows- latest, macos- latest]
   java: [17]
```

Workflow Steps

1. **Checkout Repository:** The workflow uses `actions/checkout@v3` to clone the repository.
2. **Set-up JDK:** The Java Development Kit is configured using `actions/setup-java@v3` with the Temurin distribution and Gradle caching enabled.
3. **Cache Gradle Packages:** Gradle dependencies and wrappers are cached using `actions/cache@v3` to reduce build times.
4. **Grant Execute Permission:** On Unix-based systems, the Gradle wrapper script is granted execute permissions to ensure it properly executes.
5. **Build and Test:** The project is compiled and unit tests are run using `./gradlew build` and `./gradlew test`.

Benefits

This CI setup automates the verification of our LibGDX project across multiple environments, ensuring that our game remains stable and functional across all targeted platforms. By integrating CI directly within GitHub using GitHub Actions, we maintain an efficient development workflow, allowing for continuous improvement and fast iteration.