Continuous Integration

Cohort 1, Group 6 - M6

Members:

Mir Baydemir
Adam Fraulo
Azib Hj Abu Akmar
Agata Pittarello
Esther Scanlon
Jazz Stubbins
Sonia Szetela

Overview of CI Architecture

Our project utilizes Github Actions as the backbone of its Continuous Integration (CI) Architecture, designed to rigorously enforce code quality and automate the release lifecycle. We have implemented Three primary pipelines to allow fast software delivery and a robust development cycle.

Continuous Integration Pipelines

1. Build and Test Pipeline
    - Trigger: Push/Pull request and any push of Tag version following v#.#.# pattern.
    - Inputs: Source code, Temurin JDK 17, Gradle build tool, OS system matrix, and Gradle arguments.
    - Outputs: Core test report, JUnit xml file, and JaCoCo Coverage Report.
    - Description: This stage uses a multi-OS matrix to compile the Java source code and run the automated unit testing using Gradle. It ensures multi-platform compatibility and that quality metrics have been taken before any artifacts are stored.

2. Artifact Archiving Pipeline
    - Trigger: Runs automatically after the Build and Test pipeline succeeds.”If: always()” on the test reports ensure that even if the first pipeline failed, the Core-Test-Report and JUnit xml artifacts are uploaded for debugging and analysis.
    - Inputs: JAR file in the lwjgl3/build/libs/ directory, Test documentations, and JaCoCo coverage report that are generated by the previous pipeline.
    - Outputs: Github workflow artifacts.
    - Description: This stage saves the documents generated during the Build and Test pipeline. Since Github Actions runner does not preserve the outputs form the earlier pipeline, this stage is critical for saving the executable game files, test documentation and quality reports, allowing further inspection by developers and to be used in the Release pipeline.

3. Release Pipeline
    - Trigger: Git tag Push and the Successful completion of the Build and Test Pipeline.
    - Inputs: Platform specific executable JARs downloaded from the Github Actions storage via actions/download-artifact.
    - Outputs: Formal Github release page containing three downloadable assets for each operating system.
    - Description: This stage acts as the "Continuous Delivery” component of CI pipeline, and uploads them as a public release. The implementation of “needs: [‘build’]” line in the release ensures that it only runs with the succession of the Build and Test pipeline. Since it requires the successful completion of the said pipeline, it ensures that users can access only the stable, and tested version of the game without needing to run the source code themselves.

Continuous Implementation using YML

1. Multi-OS Build Strategy
   The pipeline is implemented with a multi-os matrix, which verifies that the code compiles and runs successfully on each platform. “Fail-fast: false” allows the workflow to continue executing parallel jobs even if a specific platform encounters an error.

```
build:
    strategy:
        fail-fast: false
        matrix:
            os: [ubuntu-latest, windows-latest, macos-latest]
    runs-on: ${{ matrix.os }}
```

2. Automated Build and Testing
   The Build and Test pipeline is implemented by a single gradle command that compiles the source code, allowing the code to be tested using UnitTest and JaCoCo.

```
steps:
-   uses: actions/checkout@v5
-   name: Set up JDK 17
    uses: actions/setup-java@v5
    with:
        java-version: '17'
        distribution: 'temurin'
-   name: Build with gradle
    Uses: gradle/actions/setup-gradle@417ae3...
    With:  {arguments: build jacocoTestReport}
```

3. Conditional Artifact Archiving
   The pipeline distinguishes between the product artifacts and diagnostic artifacts using conditional logic. The use of “if: always()” logic ensures Core-test-report and the JUnitXML documentation is uploaded even with the failure of the Build and Test pipeline.

```
-   name: Upload core test results (XML)
    if: always()
    uses: actions/upload-artifact@v4
    with:
        name: core-test-results-xml-${{ matrix.os }}
        path: core/build/test-results/test
```

4. Controlled Release
   A successful release is gated by two conditional triggers: a successful Build and Test exit and a manual tag push by the development team. This ensures that the released software has gone through extensive testing and satisfies all quality metrics across every os in the multi-os matrix, thus guaranteeing the end-user a stable software.

```
release:
    runs-on: ubuntu-latest
    needs: [ 'build' ]
    if: startsWith(github.ref, 'refs/tags/')
```