UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Continuous Integration
Cohort 2 - Group 13

TAKEN OVER FROM GROUP 16

Group Members:

Carys Hoile
Ivo Hadley
Shravani Baviskar
Alex Kleijwegt
Owen Codrai
Caner Cetinkaya
Haiqal Mohammad Nazli

Continuous Integration Methods and Approaches

For our project we implemented a continuous integration strategy based around ensuring simple development of the game and effective deployment of new features and updates. The jetbrains article on best practices for CI [1] helped guide our decision making during the CI process.

Firstly the ethos of 'commit early, commit often' meant that our goals were to always share our progress onto the main branch as soon as possible and avoid branches existing for too long. This meant that progress on the project was visible by everyone in the team and also meant our automated testing through CI could provide rapid and frequent feedback on the changes allowing us to catch errors and bugs quicker. Little and often commits also reduced the chance of big merges causing errors which may take a long time to resolve.

The 'keep the builds green' approach continues this idea by ensuring all automated tests are met with every commit. If errors are met, either they can be quickly resolved, or the commit can be reverted and the issue addressed locally before being re-committed.

The automated testing was the core component of our CI strategy. With each commit our automated tests would be run to ensure core gameplay elements hadn't been broken which helped us commit with confidence. We also added a test coverage report to the CI automations. This provided us with insight on the extent of the automated tests i.e. how much of the code was actually being tested. This allowed us to identify parts of the code that are missing tests and this is important as without tests, these areas could become potential sources of bugs or errors.

In addition, the coverage report is very useful should someone else decide to continue with the project in the future. Any new features they add to the game will be spotted by the coverage report and remind future developers on the project to continue implementing automated testing for their new features.

Our Continuous Integration Approach

Our actual CI approach echoed the ideas above. We focused on small, regular commits to the main branch to ensure steady progress and avoid the challenges that come with large branch commits.

Our automated CI was set up with 'Java CI with Gradle' inside the gradle.yml file. This provided a very simple and effective implementation of CI to our project that could handle automated building of the project and running of tests.

Our coverage report was implemented with jacoco, however as mentioned in the testing report, achieving 100% coverage was not a significant priority due to difficulties in how

Group 16 had originally developed the game. In some cases, game code and logic were removed from their classes and put into fake classes for the purposes of testing. However, the coverage report was still included as it allowed us to spot code which was testable but hadn't been tested, but more importantly it was included for future development of the game. Should another developer choose to continue working on the game in future, the coverage testing will inform them of missing tests on their new updates and features as well as potentially be usable as a complete coverage test if the code is refactored.

```yaml
      - name: Upload JaCoCo coverage report
        uses: actions/upload-artifact@v4.3.3
        with:
          name: jacoco-report
          path: Group-13-2/tests/build/reports/jacoco/test/html/
```

Automated upload of the JAR file was also included in the CI workflow allowing quick download of the latest JAR version from Github Actions.

```yaml
    - name: Upload JAR
      uses: actions/upload-artifact@v4.3.3
      with:
        name: game.jar
        path: Group-13-2/core/build/libs/
```

Finally an automated release job was added allowing quick creation of new releases whenever a new tag is created in the repository. This also allows us to create new releases straight from Intellij rather than having to manually create them within GitHub.

```yaml
  release:
    runs-on: ubuntu-latest
    needs: ["build"]
    if: startsWith(github.ref, 'refs/tags/')
    permissions:
      contents: write
    steps:
      - name: Download a Build Artifact
        uses: actions/download-artifact@v4.1.7
        with:
        #Name of the artifact to download. If unspecified, all artifacts for the run are downloaded.
          name: desktop-1.0.jar
      - name: Release
        uses: softprops/action-gh-release@v2
        with:
          files: desktop-1.0.jar
```

References:

https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-best-practices/