UNIVERSITY OF YORK
ENG1 – TEAM 18

Continuous Integration Report

ROB ANDERSON
MORGAN ELKINS
ZIAD ISMAILI ALAOUI
SAM KNIGHT
SCOTT REDSHAW
GEORGE TONNERO

5.a - Continuous Integration Methods

Badges

GitHub badges were chosen to be included in CI as they allow anyone at a glance to see the current status of the code and check if the most up to date version is correct. Meaning that another developer can quickly see what is and isn't working with the project and then take steps to either fix that code, or start from a previous commit where the badge shows that it is working.
Another advantage of using badges is that they increase the readability of the repository, as they capture important metrics about the code, such as if all the tests have passed and the code has compiled correctly.

Building and Testing

Using GitHub actions, the code which is committed can be automatically tested and built using Gradle. This means that each time a commit or merge to main occurs the code is tested with unit testing (more on that in the testing section) and then compiled. This ensures that each time the code is updated, it can be automatically checked if the new code has broken any existing features as well as making sure that there are no syntax errors.
This is useful as it means that once someone has submitted their code, it can be automatically checked to see if it works correctly, and if it doesn't then it lets them know what is wrong, and typically a fix for it. It also means when someone in the group goes to make new changes to the code, they can see if that code is in a working state or not. Lastly, it gives immediate feedback on the code so someone can see if anything has been broken by their change and then fix it; this saves a significant amount of time bug fixing.

Linting

Linting was chosen to be included in the CI as it allows for the style of the project to stay consistent throughout the development as the linter runs automatically (using GitHub actions) on the committed code and then tells the committer what changes need to be made. This makes sure that there is a consistent coding style which is one of the requirements for the implementation.
It also helps to identify bugs and memory leaks, which can be found in the logs of GitHub actions. Overall linting allows for the automation of a repetitive task of checking if the code style is consistent, so allows group members to use that time to work on more important things.

Artefact Uploading

As a byproduct of the building and testing of the code, the unit test report and .jar for the game are created during this process and temporarily stored in GitHub actions. This means that they can be uploaded for the user to have access to after the testing process has finished. As well as those, the JavaDocs are also automatically created which are then uploaded in the same way. Then after the job has finished, they can then be downloaded to be used in the website.
This is useful as it means that the task of getting an up to date version of the unit testing report, JavaDocs and the game file is automated and is all in one place which saves time.

5.b - Continuous Integration Infrastructure

README.md

Java CI with Gradle passing | Lint Code Base passing

```yaml
name: Java CI with Gradle

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ${{matrix.os}}
    strategy:
      matrix:
        os: [ubuntu-latest,macos-latest,windows-latest]

    steps:
    - name: checkout
      uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'
        cache: gradle

    - name: Build with Gradle
      run: gradle build
```

Badges

At the top of our README file we have workflow status badges which automatically update and state whether the last commit to the main branch passes all of the tests and the code compiled correctly. The badges update live with the current push so are always up to date and appear green with the text “passing” if the CI task has produced no errors.

Building and Testing

In the gradle.yml file is where the code is automatically tested and compiled. To do this GitHub actions are used to complete these jobs on their servers, so each task is performed with no user input.The code is tested and compiled on Ubuntu, macOS and windows which helps cover the requirement of the game being able to run on multiple platforms automatically.

Linting

To make sure that the code which has been written and is to be pulled into main is stylistically consistent with the rest of the code, a linter workflow is applied to the code. This will produce an error message of what is wrong with the code in the action to help the user to format the code in the correct way. (Note: the flag `DISABLE_ERRORS: true` is enabled as the specific linter which we used produced an error even when correct, but it’s still used for finding where errors are.)
The code for this can be found here: linter.yml

Artefact Uploading

In the gradle.yml code, once the testing and compiling has completed, those files are able to be uploaded. This means the creation of the .jar, test report and javadocs can be created automatically and can then be downloaded and used. This is useful as these files can then be used on the website to display testing and documentation data and it means it’s always upto date.

```yaml
- uses: actions/upload-artifact@v2
  with:
    name: Tests
    path: ./core/build/reports/tests/test

- uses: actions/upload-artifact@v2
  with:
    name: Jar file
    path: ./desktop/build/libs

- uses: actions/upload-artifact@v2
  with:
    name: javadocs
    path: ./core/build/docs/javadoc
```