Part 5. Continuous Integration

Team 5 | Team Pending

Thom Robinson
Iris Scoffin
Ayman Elsayed
Annice Headman
Izaac Threlfall
Yihong Zhao

Part 5 - Continuous Integration

Part 5a - CI Summary

There are three main areas we used CI (Continuous Integration) for:

1. A quick check that the code compiles properly on commit

   This is useful to ensure that all code in the repository is at least basically functional. It is run whenever code is pushed to the repository. This does result in a large amount of runs however it is a useful check.

2. Running tests on pull requests to main

   Running the included tests ensures that the code in the main branch passes all tests. Furthermore, this can help find issues created by merge conflicts before they become obvious. This action is run whenever a pull request is opened to merge with main or code is pushed to main.

3. Creating and tagging releases

   This helps us create .jar files based on the latest code in the main branch. Furthermore, GitHub also automatically lists the commits made since the last release allowing us to see what changes we have/are missing from any jar file. This action is executed manually.

Part 5b - CI Infrastructure

For convenience, we used Github Actions. Github Actions is not only built into Github, but also provides a large library of predefined steps to allow easy customization, for example, we used the official Gradle actions to ensure Gradle was set up correctly for each run. Actions are run on both manual triggers and automatically when code is pushed to the repository.

The first part of all tests is the same - pulling the repository and setting up Java and Gradle. The following is the workflow that does this. We used the default actions for checkout and java setup and the official Gradle actions for setting Gradle up.

```
- uses: actions/checkout@v2

- uses: actions/setup-java@v2
  with:
  java-version: '11'
  distribution: 'adopt'

- name: Validate Gradle wrapper
  uses: gradle/wrapper-validation-action@v1

- name: Setup Gradle
  uses: gradle/gradle-build-action@v2
```

The next stage is to run Gradlew in some manner. For build and release, this is `Gradlew build`, for test it is `gradlew test`.

The following build Gradlew action creates a jar file. In the build workflow it leaves this file be, in the release workflow it uses the following to tag it as a release.

```
- uses: "marvinpinto/action-automatic-releases@latest"
  with:
    repo_token: "${{ secrets.GITHUB_TOKEN }}"
    automatic_release_tag: "latest"
    title: "${{ github.event.inputs.title }}"
    files: ./desktop/build/libs/desktop-1.0.jar
```

In the test workflow we used the following action to create a test report in Github, allowing us more detail on what isn't working.

```
- name: Publish Test Report
  uses: mikepenz/action-junit-report@v3
  if: always() # always run even if the previous step fails
  with:
    report_paths: '**/build/test-results/test/TEST-*.xml'
```