Continuous Integration

Group 5 :
Callum Watson
Jack Guan
Craig Slomski
Yaseen Khan
Chase Mo
Kamrul Islam

5a. Approaches to CI:

After having a team meeting to discuss CI, we concluded some practices to ensure when conflicts occur, they can be handled as smoothly as possible These include:

Prioritising fixing broken builds: Upon encountering a broken build our process is to either:

- Revert the change causing the broken build and allocate resources to determining the cause and correcting it. This is essential when wanting to avoid halting any other implementation.
- Pushing a quick fix, during this time no other commits should be made. Only useful when the problem is already identified and can be corrected in a small amount of time (< 5 minutes)

Frequent committing: As part of continuous integration which is to integrate work frequently. If we decompose the implementation into smaller problems and commit them as we develop them, this ensures that if conflicts emerge when merging branches, that they can be resolved quickly. It also allows others to see our progress and catch up with what we are doing should a member become unavailable.

Single source repository: In our repository, we aim to keep everything that is linked together all in one organised place so teammates can track down what they need easily and make it easy to fork and clone the desired work. Repository for: UMLs and Architecture, Website, Active Game Repository.

Accessible latest executable: To encourage users to download and run the latest build, made the JAR available from just to clicks on our website. Typically this would be automated, however we made the decision to manually update this, in order to do any manual checks needed and not risk releasing an unintended buggy version to the users.

Build testing on an Integration Machine: Before releasing a build, ensuring the build has been tested on an integration machine to standardise the build works on everyone's machines. A couple of options were discussed and we opted for the second option:

- Using the University computers ensures that it can build on an clean independent device, however these are not always available to use
- Using GitHub actions to run a test build from a clean checkout of our code, available to everyone.

5a. Intended CI Pipeline:

Our ideal pipeline can be shown in the diagram [1].

BUILD: **Input:** Code submitted on GitHub **Output:** A build of the updated game.
Trigger: Pushing code to any branch (specifically in our 'Active Game Repository')

TEST: **Input:** Build from previous stage, latest Java unit tests **Output:** Testing report (regardless on if success/fail) **Trigger:** A successful build being created from the previous phase

MERGE: **Input:** Previous code available on the repository, new code submitted on GitHub
Output: Updated merged code, that passes all of our tests
Trigger: Successful completion of the previous phase

DELIVERY:

A successful completion of tests will trigger GitHub to merge the new code with existing code and automatically release the new version of the code into the repository (if it was committed to the main branch). However, regardless of the outcomes of the tests, it will still post a testing report as an artefact

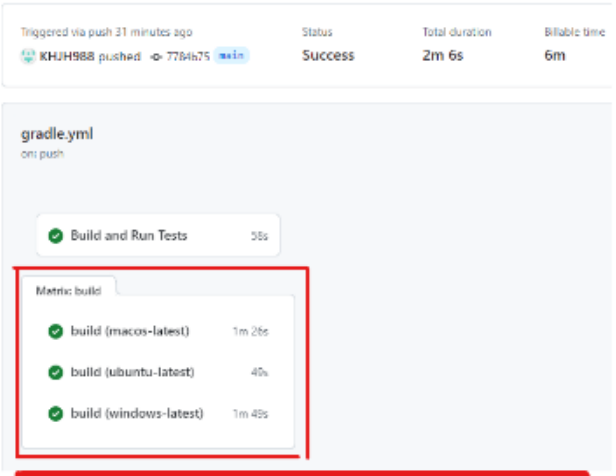
DEPLOYMENT:

Due to the website being in a separate repository, it will require additional steps to set it up to release onto the website. In addition, it is more suitable for us to deploy the latest production manually, as CI does not account for additional manual tests made by our team.

Criteria, we aim for this Continuous Integration process to:

- Use GitHub actions to streamline the implementation without the use of any additional software.
- The total time it takes to build and test the solution should be no longer than 5 minutes in order to provide fast responses to the developers.
- Clearly show the developers any failed tests with details on specifically which tests have failed, in order to easily locate and refactor the code necessary to pass all the tests.

5b. Evidence of implemented CI Pipeline and justification:

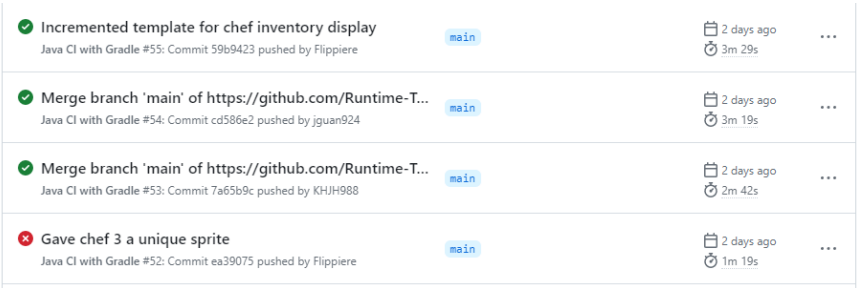


As intended we used GitHub actions to implement our CI Pipeline. Using YAML in order to create various components each explained below:

To ensure that our code meets stakeholder requirement FR_FLEXIBLE_OS we have implemented automatic building and testing across the latest Windows, Mac and Linux distribution provided by GitHub actions. Despite only requiring our game to work on 2 of these distributions, across all of our team, we use a combination of all these systems. Therefore, we want to ensure that it works across all of our development systems. This is achieved through the YAML code shown in Figure 1 inside the red box.

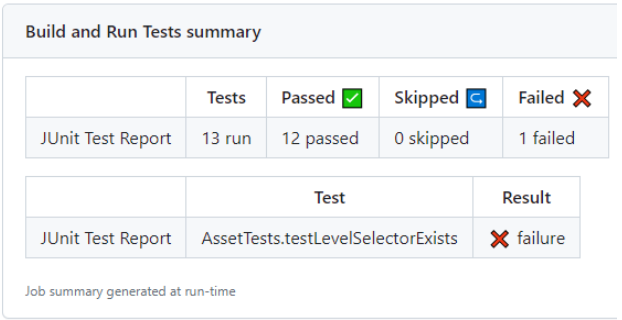
```yaml
jobs:
  build:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]

    runs-on: ${{matrix.os}}
```

With each github push, comes with a visual indication to whether all of the current tests have passed. A green tick for all passed and red for at least 1 failure. This is to ensure our developers investigate the source of the error and resolve it as soon as possible to prevent further refactoring. The tests are completed through github actions and the following YAML code highlighted in the green box.




```yaml
Test:
name: Build and Run Tests
runs-on: ubuntu-latest
steps:
  - name: Checkout Code
    uses: actions/checkout@v1
  - name: Build and Run Tests
    run: |
      cd PiazzaPanic1/PiazzaPanic1-main/
      gradle test
  - 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'
      detailed_summary: True
```

Our testing report ‘Build and Run Tests Summary’ informs us of the number of passed and failed tests. If any fail, it will return the name of the test that fails and any additional info. This allows us to locate and refactor necessary code in order for these tests to pass. The YAML Code is shown in blue. Here we use an external action available on the market place to produce this which is credited to in the licensing section.
There were attempts made to convert this table into an artefact, but it was possible with this external action. However, we agreed it was satisfactory, as the only downside is that you need to scroll to reach it.



Finally, our criteria outlined in the previous section have been met from this solution as we used Github to streamline the process testing and implementation process, each build/test takes under five minutes (ranging between 1-4 minutes) so the developers get fast feedback. Then lastly, we get an indication of when tests fail and specifically which ones go wrong, essential for locating errors.

References:

[1] G. Quaresma, “Do you know the CI/CD initiative? applying these concepts using gitlab CI/CD tool: The miners,” *The Miners* | *Codeminer42's Engineering Blog*, 12-Jan-2023. [Online]. Available: https://blog.codeminer42.com/do-you-know-the-ci-cd-initiative/. [Accessed: 21-Mar-2023].