Continuous Integration

“Lucky” Team 13

Team 13

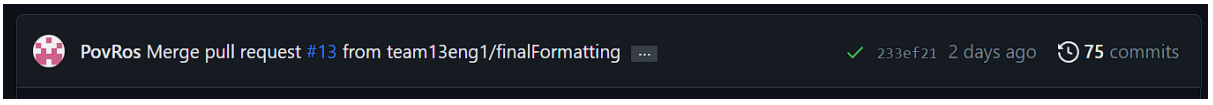
Yuxin Wu
Bailey Findlay
Elizabeth Edwards
Chenxi Wu
Alex McRobie
Yihong Zhao

Continuous Integration

Methods and Approaches

For our approach to continuous integration we needed a version control system. After gaining experience with GitHub from our previous assignment we decided to continue using it as we could have a single source repository, allowing us to easily locate where everything related to implementation was. GitHub also allowed us to separate tasks onto different branches which were then integrated into main so we could work on the project code simultaneously. For example we had short lived branches which through pull requests were integrated into main. Examples of this include branches for implementing power ups, saving and the pause screen which were all merged with main once development on that branch was completed. This way different requirements could be worked on both independently and simultaneously with the most up to date version of the code as any new branches would be created from main which was frequently being updated.

A large part of continuous integration is automating building and testing so the developers do not have to do it manually either through locally running the Gradle build tool with automated Junit tests or by running it through GitHub actions. We automated our building and testing by using GitHub actions as it integrates well with GitHub which is where we already store our code and although it is not as flexible as a standalone CI server such as Jenkins, it is sufficient for small and medium projects. Using GitHub actions enables us to run the Gradle build action on every commit and pull request. This allows the developer to see if their latest changes have passed all of the tests through email notifications as well as every commit being annotated with a tick or cross. The notifications of failing builds show the developer that their changes require fixing.



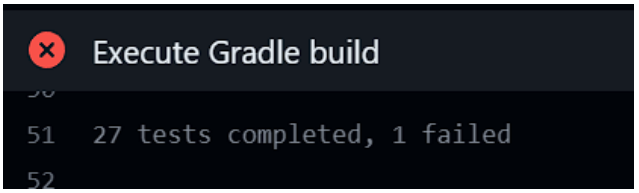
In continuous integration it is good practice to designate an integration machine as this way there is a reference for whether the game is able to run and pass the tests. We used GitHub actions for our designated integration machine as we are able to automatically run the build and tests on multiple operating systems for changes made by anyone on our team using a clean checkout of the code, allowing us to cover multiple scenarios at once. Since our game is cross platform, being able to test on supported operating systems is prudent.

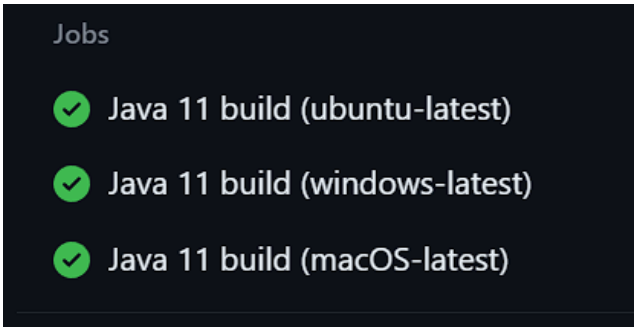
We decided that continuous delivery and deployment was not needed for our project as we only needed to provide one release of the game. Therefore due to time constraints we believed it was beyond the scope of this project to include it as development time would be better spent elsewhere.

Continuous Integration Infrastructure

We used GitHub for our version control for the previous assessment which we decided to continue for this assessment. This allowed us to use GitHub actions, which is a workflow automation tool that can be used to implement continuous integration. GitHub actions was appropriate for our project as it enabled us to automate the building and testing of the code whenever the code was pushed or pull requests to main were made. This is set up through the Github actions YAML using the “on” property to define that the automation should run when pushes and pull requests are made to the main branch. We only ran this automation on the main branch as this was where any final implementations of code were merged. Therefore, developers should be made aware if their changes do not pass tests or prevent the game from building.

```yaml
name: Java CI with Gradle
on:
  push:
    branches: [ 'main' ]
  pull_request:
    branches: [ 'main' ]
  workflow_dispatch:

jobs:
  build:
    name: Java 11 build
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macOS-latest]

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 11
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '11'

      - name: Allow execute permissions for gradlew
        run: chmod +x gradlew

      - name: Build with Gradle
        uses: gradle/gradle-build-action@v2
        with:
          arguments: build
```





We configured GitHub actions for our repository to build and test the game across multiple operating systems as our users and developers may be using different operating systems so we want to ensure the game can be built and pass the tests on popular operating systems. First JDK 11 is set up for each operating system and the main branch of our repository is checked out. We used JDK 11 as this was the JDK version specified in the project brief. Next the official Gradle GitHub action is run with the ‘build’ argument. We used the build argument as it compiles our code, runs our tests and builds an executable jar file which is exactly what we require. Since the tests are run at the same time as the build, the build will not complete if any of the tests fail, allowing developers to receive feedback for any failing tests. They will also be notified if all the tests pass as the build will have completed.