Continuous Integration

For our continuous integration we used GitHub Actions to create a ‘Java CI with Gradle’. Setting up a CI workflow like we did allows us to automate testing, meaning that whenever there is a ‘push’ to the GitHub repository it automatically runs all the tests and reports on whether they have all passed or not. After each change to the code has been made, checking whether all the tests have run is crucial when working with an agile approach. Whenever anyone merged or made an update we made sure that they ran the tests again, therefore if they fail, we know what to change and what the problem is. For this project, as there are multiple people working on one set of code, having the automated tests means that everyone knows what state the code is in and what works and what doesn’t when they continue working on the project. This way, it's a continuous process of communication, so that less errors or faults get missed or overlooked, meaning there could result in a reduction of bugs.

Workflow Outline

We used Github actions as our continuous integration provider. To configure this we use a yaml file in the main repository, this file contains the configuration for the CI workflow and describes the steps needed to build, test, and log the project. The following is a breakdown of the commands used in the yaml file and their effect on the CI configuration we used.

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

Firstly we specify the events on which the CI workflow will run, initially we chose to run testing on push and pull requests to the main brance. This decision made the most sense as it allowed us to ensure all the main code was well tested, ready for deployment.

We next specify for Github Actions to run the CI pipeline on ubuntu-latest, this operating system made the most sense as its fast and supported by our game.

```
runs-on: ubuntu-latest
```

The next section of the code describes the steps taken upon running the CI workflow:

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

This section specifies two steps, the first checkouts the current version of the code, this ensures that the code that we are about to test is the latest version in the repo. Next we setup the JDK, this is needed for building and testing the project in the next steps.

```
- name: Build with Gradle
  uses: gradle/gradle-build-action@...
  with:
      arguments: build
- name: Test Project
  run: ./gradlew clean tests:test
```

This section first builds the project using Github Actions' gradle-build-action, it passes the build argument to specify building the project. It next runs the pre-compiled "gradlew" binary with the clean and tests:test command which specifies cleaning logs and running all tests outlined in the test project.

If all tests pass without error the workflow exits and a tick is displayed on the main repo, if any tests fail an X is displayed and we are able to click the link for specific details of which test failed and why.