Group 21
Team 21

Continuous integration

Doaa Doukh
Surbhi Lahoria
Sean Abong
Peter Beck
Isaac Ohara
James Cretney
Lloyd Newton

6a. Methodology

Our CI methodology revolves around using GitHub actions with Gradle to ensure that our Java game is stable and that any errors when integrating new code are identified as quickly as possible. This is advantageous in team projects where multiple developers are collaborating and simultaneously working on overlapping parts of the code and are frequently committing changes to the main branch or merging from feature branches.

We used automated workflows as GitHub actions allows us to automate our software build and testing processes every time a change is pushed or pulled into the main branch as well as the testing-branch. This setup ensures all new code is automatically verified and tested against the Junit tests we made which provide continuous feedback and maintain code quality. We used Gradle for build automation as it's powerful and efficient and Gradle Wrapper, gradlew, which ensures consistent Gradle versions across all development and CI environments, eliminating compatibility issues.

The workflows use 'actions/setup-java' to configure the JDK environment and the JDK is set to version 11 as required by our project. The 'gradle/actions/setup-gradle' is used to configure Gradle and './gradlew build' command is used to compile the code and ensure it builds and that any compilation errors are caught early. The '/gradlew test' command is used to run JUnit tests, verifying the functionality of the code and the '/gradlew jacocoTestReport' is used to generate code coverage reports with JaCoCo which are then uploaded as artefacts for review so we can identify any untested code areas or issues with the code..

With this approach we receive rapid and immediate feedback on each push and pull which are organised in a way that we can easily identify specific issues in which we can solve quickly without disrupting the rest of the implementation process. It also ensures that any code integrated into the main branch has been thoroughly tested, which reduces bugs and disruption to the main branch. Overall using Gradle and GitHub actions is an efficient and easy way to test our project and fulfil all the requirements.

6b. Integration Infrastructure

Our first continuous integration infrastructure we set up was the Java CI with Gradle which is triggered when there is a push or pull request to the main branch. This focuses on any inputs to the main branch and was the first thing we set up after forking the project from the other group. We configured the workflow to use JDK 11 as our environment and use Gradle to automate the build and testing process. The output is a compiled JAR file resulting from the build process and a JaCoCo generated code coverage report. We had a gradle.yml file and a gradle-testingbranch.yml for both workflows respectively.

For the Gradle package workflow, the pipeline is activated when any code is pushed either to the main or testing-branch and any pull request targeting the main branch. The build and test execution ensures that once the project is built and tests are run a JaCoCo coverage report is also made and saved. Both pipelines ensure a consistent build and test environment and ensure code quality is maintained. Since we used branching, this helps identify any errors before merging to main or otherwise.

For testing, we created JUnit tests as part of the automated testing to validate the functionality of our game such as energy levels, day tracking and leaderboard functionalities so by integrating this into the CI pipeline, if a test fails we are aware of this as GitHub actions provide logs which will help the development team identify the issues.