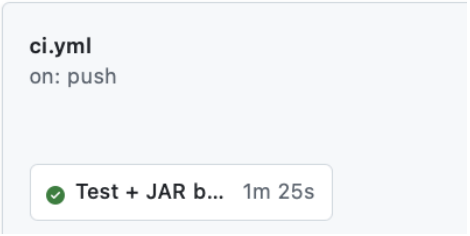
Continuous Integration (CI)

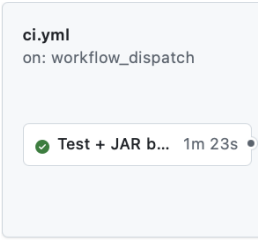
Cohort 2 - Group 7 - ‘Mikey and the Freemans’
Members: Louis Burdon, Varun Nayak, Alex Nevard, Sam Russell, Gaoman Zhu, Vivaan Kampani, Teva Geffen, George Overton

To support an iterative approach to software development, our team implemented a CI/CD pipeline to support our development process. As soon as any code gets pushed to a branch, a thorough process is followed to ensure that everything works as expected and intended. The pipeline was iteratively developed along with the code as we felt that it would be easier to adapt the pipeline configuration and logic along the way and support our development methodology at the time of writing the code rather than implementing it all as a rush towards the end. It made the debugging process far easier for us to ensure that the functionality worked as we wanted because the CI pipeline highlighted the tests that were failing for every push made to main and enabled us to quickly diagnose the fix to the problems. The pipeline takes the Java code for our project as the only input.

Triggers to our pipeline:

1) Push or pull request to main
2) When there is a manual trigger executed through the GitHub Actions tab on our repository.





There are **four outputs** in our pipeline after executing it:

1) **The JUnit report** - Having completed the automated testing, highlighting the result of each of the tests. There are a total of 123 tests for our project, but the pipeline runs it twice as a sanity check to see whether there is a bug within our code (or the test itself) and whether the first test run was just an anomalous result.
2) **JaCoCo report** - Following the completion of our test job, we then create a coverage report, highlighting the areas of our code which needs to be manually tested, making the testing process smoother and more efficient as the project gained complexity.
3) **Checkstyle report** - The test code and the code for the actual game get a style check executed on it to see if there are any violations associated with the code itself (things like indentation, inappropriate bracket use etc.).
4) **JAR file** - the game code that we have written gets compiled into a JAR executable which makes our game far easier to run and distribute.

Java is a platform independent language so the JAR file outputted from the pipeline will work on all operating systems and so, I felt there was no need to integrate matrix_os into the pipeline. We set up a basic environment using the latest version of Ubuntu and Java 17 (Temurin) to match our local development configuration. The OS converts the file paths for us upon download so there is no confusion related to using '/' for Linux and '\' for Windows operating systems.

The first part of the process is to run our automated tests, which are done in headless mode because then there is no need to create a graphical user interface for our program and then run our tests.

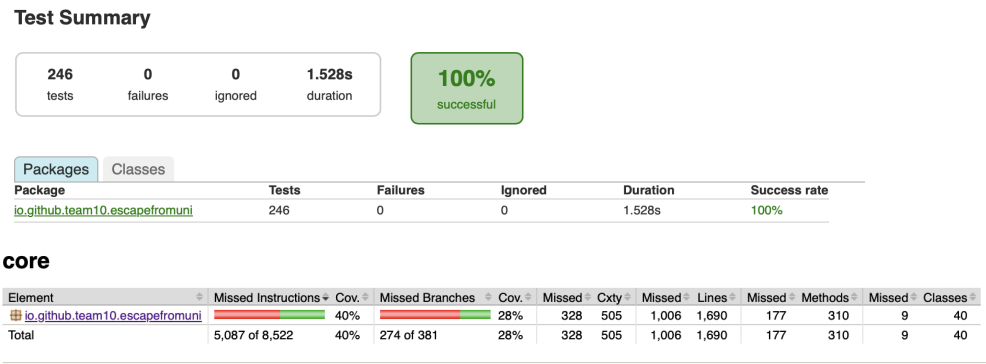
Pipeline steps:

1) Set up Gradle and make gradlew executable - Enables us to compile the java code to create the JAR file and run our tests and other operations in a stable environment. We initially had a 'validate Gradle wrapper' instruction which meant that for every operation we had to run associated with Gradle (tests, coverage, consistency reports), it meant that we had to use the –no-daemon flag to ensure that Gradle was still active and running and was taking a lot of time when it came to our pipeline being executed.

2) The next job is responsible for actually running all of our automated tests, running the test coverage report (which is dependent on the test job finishing prior to that), and then running the job responsible for the code consistency report. Following the creation of all of the reports, we then create the JAR file executable, which can be distributed.

3) Push the reports and the JAR file to GitHub - Once the reports have been created, we then create new directories in our existing workspace within the Website/docs/ folder and then use the GitHub Actions Bot to commit the reports and the JAR file to the repository using the same commands we would if we were doing it in the terminal. It means that we don't have to manually push the JAR file and the reports to the repository every time we execute our Actions workflow. **(references: links 1, 2, 4 in the bibliography on the next page).**

4) Output summary - We then output the summary of the jobs that were completed above in the Actions tab.



Test Summary

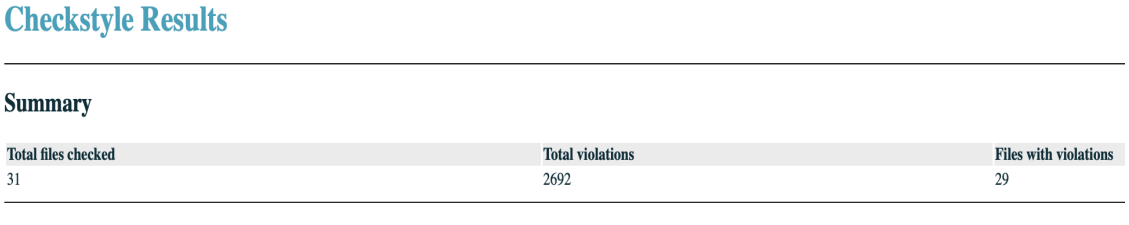
246	0	0	1.528s	100%
tests	failures	ignored	duration	successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
io.github.team10.escapefromuni	246	0	0	1.528s	100%

core

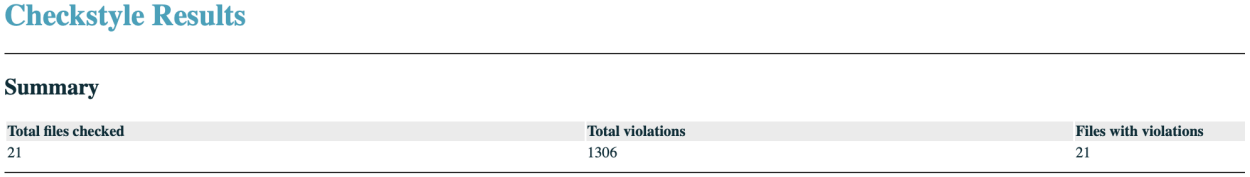
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
io.github.team10.escapefromuni		40%		28%	328	505	1,006	1,690	177	310	9	40
Total	5,087 of 8,522	40%	274 of 381	28%	328	505	1,006	1,690	177	310	9	40



Checkstyle Results

Summary

Total files checked	Total violations	Files with violations
31	2692	29



Checkstyle Results

Summary

Total files checked	Total violations	Files with violations
21	1306	21

Bibliography and references used as help for creating the pipeline

GitHub - actions/checkout: Action for checking out a repo. [online] GitHub. Available at: https://github.com/actions/checkout?tab=readme-ov-file
Gradle.org. (2018). The JaCoCo Plugin. [online] Available at: https://docs.gradle.org/current/userguide/jacoco_plugin.html.
Hombergs, T. (2018). Definitive Guide to the JaCoCo Gradle Plugin. [online] Reflectoring.io. Available at: https://reflectoring.io/jacoco/
GitHub. (2026). GitHub Push - GitHub Marketplace. [online] Available at: https://github.com/marketplace/actions/github-push?action=example&id=github-push-action
Graphite. (2026). How to use the checkout action in GitHub Actions. [online] Available at: https://graphite.com/guides/github-actions-checkout
GitHub. (2020). Push built files to repo using Actions · community · Discussion #26615. [online] Available at: https://github.com/orgs/community/discussions/26615
GitHub Docs. (2026). Workflow commands for GitHub Actions - GitHub Docs. [online] Available at: https://docs.github.com/en/actions/reference/workflows-and-actions/workflow-commands
GitHub Docs. (2025). Creating an example workflow - GitHub Docs. [online] Available at: https://docs.github.com/en/actions/tutorials/create-an-example-workflow.
GitHub Docs. (2025). Workflow syntax for GitHub Actions - GitHub Docs. [online] Available at: https://docs.github.com/en/actions/reference/workflows-and-actions/workflow-syntax.
GitHub Docs. (2026). Use GITHUB_TOKEN for authentication in workflows - GitHub Docs. [online] Available at: https://docs.github.com/en/actions/tutorials/authenticate-with-github_token?apiVersion=2022-11-28
GitHub Docs. (2026). Using custom workflows with GitHub Pages - GitHub Docs. [online] Available at: https://docs.github.com/en/pages/getting-started-with-github-pages/using-custom-workflows-with-github-pages