

# Continuous Integration

---

Cohort 1, Team5:

Lewis Mcshane

Adam Howard

Morgan Davis

Road Gibbons

Zijun Zou

Muhidin Muhidin

For our continuous integration method we decided to use GitHub Actions, due to the fact that our repository for the implementation is hosted on GitHub. Therefore, using GitHub Actions allows us to easily check the code for errors every time new code is pushed into the repository. Furthermore, this means that we won't have to use an external software, making it more efficient to check and set up, because we can access it straight from the GitHub repository.

Moreover, this is quite appropriate for the project, since GitHub Actions has a built in template for Java with Gradle. Java and Gradle are the frameworks that we use to develop the game, so this means it's a lot easier to set up and edit. Also, we can edit the .yml file directly from GitHub which makes it easier to manage the infrastructure of the continuous integration.



The screenshot displays a GitHub Actions workflow run interface. At the top, a summary bar indicates the workflow was 'Triggered via push 6 hours ago' by user 'Roan-Gibbons' on the 'main' branch, with a commit hash '2df55d6'. The status is 'Success', the total duration is '32s', and there are no artifacts. Below this, the workflow file 'Build Cl.yml' is shown, triggered 'on: push'. A single job named 'build' is listed with a green checkmark icon and a duration of '18s'. At the bottom right of the job list, there are icons for cloning the workflow file, zooming out, and zooming in.

Triggered via push 6 hours ago	Status	Total duration	Artifacts
Roan-Gibbons pushed 2df55d6 main	Success	32s	—

  

**Build Cl.yml**  
on: push

Job	Status	Duration
build	Success	18s

However, we couldn't find a way to make the continuous integration .yml jobs function properly with the LibGDX library. Therefore, we used continuous integration to test the build every time the code is pushed or pulled from the repository. Then, we can use the unit testing to ensure that the code works on a micro level, and we can use the continuous integration to test the code on a macro level.

This would allow us to easily differentiate between the lower level classes and the build as a whole when testing for errors.