

Continuous Integration

Group 26

devCharles

Ross Holmes
Sabid Hossain
Thomas Pauline
Joel Paxman
Andrey Samoilov
Louis Warren

Method & Approach_(A)

Automate as much as possible

In our CI framework, the following is desired to be automated: builds, tests, integration and monitoring. Automating builds is crucial as not only is it one of the most time consuming tasks, it is also a requirement for the rest of the CI pipeline. Our team decided that in our CI approach, whenever a build was performed by the CI, a build artefact should be uploaded. This means that the latest build is always available should we need it for manual testing or evaluation.

Automating testing and integration is the most crucial part of our approach, as it means that we can automatically check that any new features that we have created have not broken any existing features or functionality. This saves a lot of time and reduces risk of human error, for example it makes it impossible to forget to run tests before finishing a feature.

Our continuous integration approach also needs to automate monitoring. Team members should be notified when things go wrong and kept up to date with progress. This can be done using automated coverage reports, which can be used to inform the team's decisions on whether the tests are complete and correct. A coverage report can be created everytime CI runs, with statistics such as line coverage per class and branching coverage per class.

Of course, sometimes automation can be unwanted and annoying, so there should always be an option not to run a CI workload if that is not desired for a valid reason. Therefore we believe that limiting the CI workflow to a specific scope rather than running it on any branch or event is more suitable.

Integrate frequently

CI should be designed such that code can be constantly integrated with the main branch. This reduces the complexity of the development workflow, since team members won't need to decide when to try to integrate. This is ideal since we will be able to automatically detect incompatibility and be aware of the issue earlier, allowing us to fix it before the feature is fully merged into the main branch. Integrating often also reduces risk of conflicts that can arise when developers work on the same codebase simultaneously, as well as encourages collaboration as developers are incentivized to communicate about the changes to the codebase.

Fix issues quickly

If the tests fail or an issue is identified the team should immediately divert their attention to fixing these issues. Otherwise, if failing tests are ignored, and features are built on top of code that doesn't work properly, then this will cause greater issues down the line and can amount to a significant problem that will take up development time and could have been avoided.

Enforcing a code style

To eliminate discussions about code style, it would be useful if the CI pipeline enforced a code style. This would help the codebase to remain consistent. Consistency in style is very important, as any code style is better than no code style when it comes to readability. This also partially removes the burden of learning the particular code style for the project, as it can be made to comply with the style rules automatically.

Targeting more than one OS

One of our User Requirements states our game should be able to run on any computer with any major OS, such as Windows, MacOS and Linux this means Multi-OS build testing is required. Our CI workflow should perform all build and test tasks on all three platforms so that we can make sure we can fulfil this requirement.

Keep builds fast

It is always beneficial to have a fast build, since it reduces time spent waiting on CI to run in between commits. We will consider ways to reduce build time when creating our continuous integration, although since our project is fairly limited in scope it shouldn't require a lot of optimisation.

Actual CI Infrastructure_(B)

GitHub Actions

GitHub Actions is the chosen CI framework, for both its feature set and ability to seamlessly integrate with the rest of our tools, as we use GitHub to store the repository and organise tasks. Our workflow consists of a format job and a build job. The build job targets Ubuntu, Windows, and MacOS. It compiles the project from the commit it was triggered by, and runs tests and creates a test report for each OS, specifying which tests passed, failed or were skipped. To make the action take less time, the code coverage report and artefact upload is only done for the Ubuntu version of the build job.

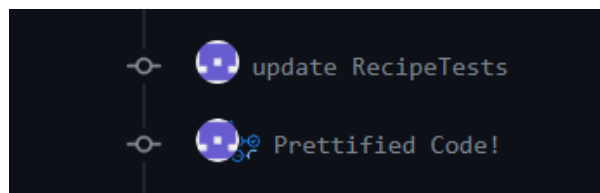
The CI build takes at most 3 minutes to run concurrently, with most jobs taking up less than 1.5 minutes. Total time can reach up to 5 minutes, but this does not matter as our project is not billed on Actions CPU time.

Automatic Integration

A branch was created for each feature or related feature group. The group agreed to keep each feature branch minimal in scope so that it would be easier to evaluate using CI tests, among other reasons. The CI workflow was set up to target any GitHub pull request and any direct commit on the main branch. Specifically, the GitHub action targeting the pull request would target the merge commit between the feature branch and main, therefore integrating automatically on every run. Direct commits to main were highly discouraged and branch protection rules were put in place on main so that all features were developed on pull requests. As a result, all features get to run through integration testing before being approved and merged. Merge conflicts were resolved in the feature branch semi-automatically using Github Pull Requests.




Automatic Formatting

The format job in the CI workflow runs the prettier code formatter on the codebase, and adds an extra commit with formatted code on top of the original author's commit. This ensures that the code style is consistent. The extra commit has been proven to be problematic sometimes, as team members can forget that it was added and push more changes, resulting in a merge conflict. Overall, it has been proven useful, but further improvements could be made.



Automatic Deployment

We have implemented a second workflow that is triggered by git tags. This workflow produces a release for all 3 platforms. This allows us to rapidly produce a new release by tagging a commit with a version number.

Artifacts		
Produced during runtime		
Name		Size
	piazza-panic_release_macos-latest	38.6 MB
	piazza-panic_release_ubuntu-latest	38.6 MB
	piazza-panic_release_windows-latest	38.6 MB