CONTINUOUS INTEGRATION REPORT

GROUP 5 - BITCRUSHED BOB

Maryam Mathews
Joseph Hinde
Jacob Mace
Will Aon
Zathia Jacquesson-Ahmad
Bulganchimeg Munkhjargal
Evan Weston

CI Pipeline:

The CI pipeline our project follows uses github actions with gradle to build, test and produce executables of the project. One of the main Continuous integration principles that we wanted to uphold was ease of access to the executable for the whole team at all stages of development. So our pipeline creates .jar files for each operating system so no matter what system the team uses they can test the current stage of the project. These jar files are displayed at the bottom of the github actions tab as artifacts as shown in *figure 1.* As well as the .jar files the CI pipeline also gives a test coverage report with each deployment, this allows the team to know what parts of the game and game systems need to be tested by hand and which are covered in the unit tests written.

As well as producing executable files for all stages of development and the test reports, our CI pipeline will also present an obvious tick or cross depending on whether the tests for the project that are run through gradle pass or fail. This allows full transparency between the team as anyone can see what tests are currently passing and failing on the main branch of the project.

The project in of itself is quite small in scale so the CI pipelines features are made to work for a small scale project. Due to this size we only need a small CI pipeline that is focused on fast feedback and deployments. In this fashion a single github actions pipeline is more than sufficient and it only needs to have 2 runners, one on windows and one on linux. We don't need any complex functionality like artifact repositories or distributed caching. So the implementation that exists is more than enough for the scale of our project.

Inputs:

The actions are triggered on any push or pull to the main branch from any branch. We chose this input triggering the actions as pushing to minor branches may not be useful as the branch might not be fully functioning, whereas branches are only pushed to main when they have started to be tested and work enough to be manually tested further.
The other input we have is for when we want to produce a release executable, this input is waiting for a branch with tags on it in the form v0-9.0-9.0-9. When the main branch is given this type of tag, the action will build the 3 OSs executables and trigger a release with these executables.

Outputs:

As previously mentioned, one of our main outputs are the multiple jar files produced for each different OS that the game can run on. As well as this we thought the test coverage report was helpful so that is also outputted as an artifact of the action using Jacoco. Finally each time the pipeline is triggered we also submit a dependency submission to github which in turn updates the dependency graph of our repository. This isn't a direct output of the pipeline but helps us keep track of and make sure that our dependencies are working properly and don't have any security issues.

Implementation of CI pipeline:

The solution starts with the setting up of the inputs to the pipeline, this is showing that it activates on a push to the main branch, a pull request to the main branch and if the main branch is ever given the tag v0-9.0-9.0-9, this is shown in *figure 2.*
Following this it uses the matrix strategy to declare the OSs that the project will run and build on, this being Ubuntu and Windows. The matrix strategy for the build job means that it will in parallel perform all of the build steps on a runner in each OS. So on each OS the following is performed:

1) We start by setting up java and gradle using the setup gradle actions
2) Then we run the command to build the project and run gradle on it. This will build and test the project as well as running Jacoco to produce a test coverage report. It will also produce the jar for the game.
3) We then upload the .jar file as an artifact from each OS with the OS name in the title
4) As well as the .jar files we upload the produced Jacoco test report from each OS as another artifact
5) We can then produce a dependency submission
   This sets up java again and generates the dependency graph for submission. The dependency submission allows us to track our dependencies and have alerts if there are any clashes.
6) Finally we can Assign the action for if a release tag is given to the main branch, this is only run after build is finished and just places all of the executables in the release:
   The release part shown is only the start, but for windows and mac is the same as ubuntu

Appendix:

Figure 1: