5. Continuous Integration Report

Part a)

Continuous Integration (CI) is a key part of any major software development process, and so it was critical for us to create a simple yet effective workflow for building, testing and deploying our project. It involves integrating code changes into a shared repository on a regular basis. The main goal of CI is to ensure that code changes are thoroughly tested and integrated into the main codebase as quickly and as frequently as possible.
CI is implemented in the development of our project using GitHub Actions. Our first approach of CI was setting up automated testing which involves setting up a testing framework to automatically run tests on all code changes made to the repository. Automated tests act as a safety net which prevents buggy builds released to users by mistake, ensuring that any code changes made do not break the build and that the game remains stable. Once the automated testing is successful, the code changes can be automatically deployed to the game server or build server, allowing stakeholders to access the latest version of the game. This is called Continuous Delivery (CD).
Code reviewing is done before merging code changes into the main repository, this can be done to ensure that the code changes conform to the project's coding standards and that there are no errors or bugs. Automated builds involve creating a build script that automatically builds the game from source code every time there is a change in the repository.
The input to our CI pipeline is the source code which is written in the IDE. When it is pushed to the repository, the CI pipeline is triggered. The build phase then takes this source code and compiles it into the final game based on the dependencies and assets used within each class. Then, the CI pipeline creates a .JAR executable file of the game, which is a playable copy of the game which does not require the source code or any time consuming compilation in order to run. This is particularly useful to us as it can be sent to stakeholders or be made available on our website for download, linking back to the concept of Continuous Development.
Then, in the next phase of the continuous integration pipeline, this executable version of the game is used in order to run automated tests, ensuring that the code which will be pushed to the repository is correct and does not produce any errors or incorrect behaviour. This is implemented using unit testing, looking at each method atomically and ensuring that its behaviour is correct. If all unit tests are passed, then the result is that the build has passed all tests, and may now be pushed. If not, then we are alerted that the build has failed, and we must take steps to fix the issue. A benefit of using unit tests is that the failing tests will be highlighted, speeding up the debugging process. This makes the overall development process much more efficient, which is critical in such a short time scale as this project.
These methods are appropriate for game development projects as they help to catch bugs early and ensure that the game remains stable. They also help to reduce the amount of time spent on manual testing and deployment, thereby increasing productivity and efficiency. Additionally, they help to ensure that all developers are working on the same version of the game, reducing the risk of version conflicts.

Part b)

We chose to implement CI using Github Actions, using the file gradle.yml once a commit has been pushed to the repository to build and test the code. This is evident with the on: push: instruction within the .yaml file. The Build job performs the following tasks sequentially: set up job, run actions, set up JDK 11, build with gradle, set up game JAR, post build with gradle, post setup JDK 11, post run actions and complete job.

The task which creates the JAR file is critical to us in implementing continuous deployment as it means we always have access to a JAR file, which we can then upload onto our website for any stakeholders to download and try. It also means that since this task was added to gradle.yml, we have a running log of all previous versions of the game as .JAR files available to be downloaded and compared. This is very useful to us in seeing how the game has evolved and learning from our previous approaches.

In our implementation, this job is called “build” and is triggered when a commit is pushed to the Western-Criminals/PiazzaPanic-2 repository.

The next job in our CI pipeline is the “test” job, which occurs after a successful build. The test job runs all available Junit unit tests available in the test/java folder, which is our test sources folder. The gradle.yml file draws the tests from here, runs them, and reports back the results. This has been invaluable to us in quickly finding and fixing errors, as the issue is reported as soon as it is found, meaning that we can fix it before finalising the push.

The final stage of the pipeline is to deploy the changes, merging them with the rest of the repository. This finalises our changes and makes them available to all other team members. Then, on our individual local machines, we may update our IDEs with the new changes by merging our branch with the main repository, and carry on editing the section of code we are working on. As previously mentioned, this helped to reduce version conflict.

With that, we have a full CI pipeline to build, test and deploy our code with minimal effort from us. This maximises code quality while reducing time spent on these tasks, giving us more time to focus on the code itself.

Additionally, we used another workflow within GitHub actions: pages-build-deployment, for our website. In this case, when a change is made to our website, the CI infrastructure builds the website, then deploys the website and reports its status. The first job in this pipeline is build, which builds the Github Pages website “western-criminals.github.io” and performs the appropriate checks. The next job is “report-build-status” which returns the status by using one of Githubs APIs. The final job in this pipeline is “deploy”, which deploys the completed website to github pages.