Continuous Integration a)

To carry out our continuous integration, we are using **GitHub Actions**. We decided to use this CI solution as it fits our project perfectly. It is built into GitHub, which we are already using for all of our repositories as well as project management. This allows CI to be integrated with all aspects of our project, such as displaying on our project dashboard. GitHub actions also features a wide variety of community built open-source tasks which can be ran within our workflows, this fits our project greatly as we have needs for features which aren't within the scope of default tasks. For example converting deliverables from markdown to PDF. Due to these reasons, GitHub actions greatly increases our productivity and was the best CI solution for us to use.

GitHub Actions is configured using .yml files stored under `/.github/workflows`, with the exception of the GitHub Pages workflows, which are handled by a GitHub bot. In order to ensure that our continuous integration is always working as intended, we have a designated CI manager, who applies fixes or develops new workflows as needed. This helps to keep our repositories stable.

- On our game repository, a test workflow is carried out when a pull request is made. This is to ensure that the code to be merged passes all tests and has no errors. If it passes the tests, this means the code is functional may be merged, which will then cause further workflows to be triggered, which will deploy test reports, JavaDocs and an executable to the website repository. This keeps the code on the main branch as well as website data correct and up to date.
- On our documentation repository, whenever a new change is successfully merged into the main branch continuous integration is triggered. This will cause all of the relevant markdown files in the repository to be committed to the website repository. This ensures all of the project deliverables on the website are up to date at all times, as required.
- On our website repository we have two separate workflows. Whenever any changes are made to the repository, the GitHub Pages workflow will always be triggered. This workflow runs `jekyll build` on the repository and deploys the results to the online website. This ensures the website is always online and up to date as required. If changes are made involving the `/markdowns` directory, a workflow will additionally trigger to convert markdown files to pdf files. This ensures that all deliverables are available in PDF format as required.

Continuous Integration b)

In each of our repositories we added a continuous integration workflow to ensure that the risk of things like merging issues causing delays (R19 in risk assessment) and additional work from manually creating pdfs and uploading to the website meaning the team has to do more work than predicted (R14 risk in risk assessment)

yorkpirates2 (game) repository

Continuous integration occurs in two ways in our game repository

1. If a pull request is made, the `test workflow is run
    - This is to ensure that the act of merging the implementation branch (different for each part of the game being worked on) and the main branch, particularly the resolution of conflicts, has not caused

any of the game to break.

2. If a pull request is **successfully** merged to main, the `test`, `build` and `documentation` work flows are all run.
   - This generates the test report, jar file and documentation pages to put on the website and the new build is put to main.

`test` workflow

Tasks: **Builds** the project using gradle. The workflow will **test** the project, but continue if it errors. This will **generate** a test report. The workflow will then **deploy** the test report to the website2 repository. Finally, the workflow will **run the tests** again, but fail upon error.

This ensures that if there are any errors in testing, it still generates a test report for us to see, and also alerts us to the error during the pull request. This means that we can then fix the issue before the game is published to main branch.

`build` workflow

Tasks: **Builds** the project using Gradle. Uses the dist command to **generate a jar** file then finally **Deploys** the jar file to the website2 repository

This allows us to automatically place the latest jar onto the website.

`documentation`

Tasks: **Builds** the project using Gradle. Uses the javadoc command to **generate** documentation as html. It then **deploys** the documentation to the website2 repository.

This means that all of the JavaDocs in our code are viewable on the website to allow for future developers to inspect but also to allow the rest of the team to view them easily

documentation2 repository

If a pull request or push to the main branch in the documentation repository is made, `deploy` is ran.

`deploy` workflow

Deploys the markdown files to the website2 repository. This allows us to keep the websites documentation files up to date.

website2 repository

All continuous integration in this repository occurs on a push to the main branch.
`pages-build-deployment` will always be ran, however `convert` will only be ran if the push includes markdown files in the `/markdowns` directory.

`convert` workflow

Tasks: **Converts** all of the files in the `/markdowns` directory into pdf files. Then **commits** the pdf files to the `/pdfs` directory in the repository.

This allows our documentation to be presented in both markdown and pdf format.

`pages-build-deployment` workflow

Tasks: **Builds** the Jekyll website. **Deploys** the built site to GitHub pages.

This allows our website to constantly be up to date with all the latest files.