Continuous Integration

Methods and Approaches (and why they are appropriate)

Our chosen method for implementing our Continuous Integration was using Github Actions. We chose this method for many reasons, chief among them being how easily it integrates with our use of Github in the rest of the project, in addition to the large quantity of documentation available for the method. This saved us time organising our continuous integration between the development team, and used a platform that everyone was familiar with, reducing the likelihood of misuse or misunderstanding the continuous integration platform. Overall, using Github actions was the best option for our development team to use, due to their familiarity with the existing platform, and how our team was already working.

Github Actions allows us to easily and automatically test our codebase every time someone pushes an update to the Github repository, as well as giving us valuable feedback on how many tests have passed or failed, in addition to information on how much of our codebase is covered by our tests. This meant that the code being written was less likely to cause problems when merged, saving time in the development of the game, and making it less likely that time had to be taken fixing parts of the codebase that broke when merged. These features allowed us to implement our Continuous Integration efficiently and effectively, and minimised risks to slowing development time over the course of our project.

Our actual integration structure

Our continuous integration is built using Github Actions. It is triggered every time a merge is pushed to the main branch, and builds the whole project to check for any compilation errors. It also runs the j-unit tests, and if any fail, fails the build and sends a notification email