Testing Program

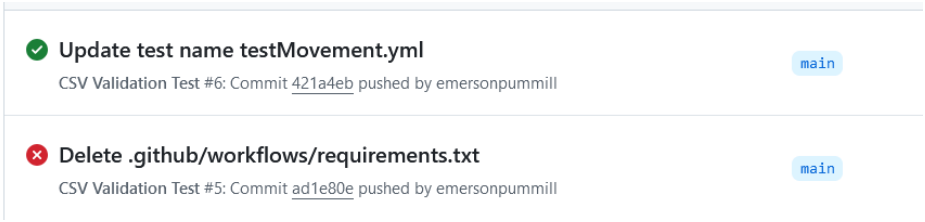
To ensure the proper function of our program, we created a testing file named “tests.py”. This uses black box testing by analyzing our movement file.

Each row of the movement file corresponds to an iteration of the grid. This means that the coordinates of every goat are documented in each row, so the entire row can be tested for border and goat collisions.

The program looks in each row for goats who have identical coordinates or coordinates at a border, according to the specified grid size. It documents these, if found, and displays them.

CI/CD Testing

Our team used GitHub’s built-in CI/CD for testing. To do this, we created a YML file and placed it in our “workflows” folder. This code ran our “tests.py” program each time an update was made to the GitHub, the results of which could be seen in the “actions” tab in GitHub. When a goat or border collision was detected in the data, the program exited with code “1”. Through the CI/CD testing in GitHub, an exit code of “1” would flag a failed test, and an exit code of “0” would flag a success. The image below shows the action page, with an indication that the test ran successfully upon the most recent update:



d

Thankfully, the black box testing was enough to verify that our goats were behaving as expected, so our team did not implement unit tests of any sort.

13x13 Grid with 61 Goats Results

First Attempt:

5397 iterations in 26:26 minutes.

By 4000 iterations, only one goat remained.

Second Attempt:

6363 iterations in 31:04 minutes.

By 3000 iterations, only 4 goats remained.