Team Verbosity
Phaser Final Report

Winslow DiBona

Table of Contents

- Introduction……………………………………………………………………………..2
- Phaser Testing Framework…………………………………………………………..2
- Team Evaluation…………………………………………………………………….4
- Conclusion…………………………………………………………………………...5

Introduction

For this project we chose to create an automated testing framework for the open source Phaser project (https://github.com/photonstorm/phaser). Initially we were developing an automated testing framework for the open source sql-cipher project (https://github.com/sqlcipher/sqlcipher) but changed focus half way through the semester after having difficulty with generating a framework around such a large project. Phaser is the most popular HTML gaming framework hosted on Github and can easily be run on all browsers and can be ported to mobile devices with the use of third party tools. Phaser is written in Javascript and uses WebGL and PIXI.js for rendering HTML Canvases in the browser.

Phaser Testing Framework

Since Phaser is written in Javascript and comes packaged in a single Phaser.js build file we decided to build out testing framework using Node.js (https://nodejs.org/en/). Node.js is a Javascript runtime utilizing Chrome's V8 Javascript engine and comes packaged with npm (node package manager). Npm is used for integrating two node modules. 'fs' is used for basic file system access and reading and writing data to files. 'open' is used for opening a test report html file displaying the results of the testing framework.

The main files/folders of the testing framework are

- /project
  - /bin
  - /src/phaser/build/phaser.js
- /reports

    - testReport.html
- /scripts
    - runAllTests.js
- /testCases
- /testCasesExecutables

The project folder contains the source code from the phaser github repository. Inside /project/arc/phaser/build is a file called phaser.js which is the build file for the phaser project. The /reports folder contains the testReport.html file which is used for displaying the results of the tests after completion. The /scripts folder contains the runAllTests.js file which is used for executing the frameowork.

The /testCases folder contains 25 .txt files which provide info on each test case. Each .txt file contains the following properties

- filename
- number
- requirement
- component
- method
- inputs
- outcomes

Each .txt file is formatted in JSON format and the files are read into the runAllTests.js script as JSON objects for easy execution of the test. The /testCasesExecutables folder contains 25 .js files which are used for the actual execution of the individual test cases.

A basic breakdown of how the framework carries out the tasks. The runAllTests.js file is invoked from the /TestAutomation folder using '> node ./scripts/runAllTests.js'. The main script will then read in the 25 .txt files from the /testCases folder as JSON objects and pass each of those off to a driver function that will invoke each test case. Each test case is given the inputs from the .txt file and it's output is compared with the 'outcome' property. If they match then the test case passes and if not then the test case fails. Once all 25 test cases have finished executing the results are put into an HTML table and saved to the /reports/testReport.html file. The testReport.html file will open automatically in your computer's default browser.

There are five faults injected into test cases 3, 11, 14, 15, 20. The faults undermines the weak type system of javascript by changing the expected types of values used in the functions being tested. For example test case #3 calculates the average of a collection of numbers. To do this it uses a 'sum' variable to hold the aggregated sum of all the numbers. For this function 'sum' was initialized to 0 so any numbers added on would result in the expected outcome. We changed 'sum' initialization to 'var sum = “”'. This caused the numbers to be added to a string resulting in a string containing the collection of number put into the function.

Team Evaluation

We are pleased with the results of the project and will definitely be able to apply a lot of the knowledge learned along the way into a professional career. While we wanted to test some of the graphical components of the Phaser project (collision detection, gui updates, etc.) it was too difficult to create individual tests around these components as they had to be rendered in a browser which drastically increased the completion time of the testing framework. Switching which open source project we were working on halfway through the semester definitely forced a lot of work to be done in the second half of the semester but we were able to catch up thankfully.

Conclusion

Overall the project turned out to be a success and I am very happy with what I learned over the course of the semester. As for how the project assignment could be improved as a whole my only suggestion is providing examples possibly of past years work for students to be able to know what is expected for each of their deliverables. My reasoning for this is that it seemed like in some of the presentations throughout the semester several groups would be confused about a similar problem, not necessarily with their testing framework, but with the way they design their project because they were confused on requirements.