Team 4 Testing Plan

Directory Structure

/TestAutomationTeam4

- /project
  - /src
    - NelderMeadTestDriver.java
    - ElapsedTimeTest.java
    - STEMTimeTest.java
  - /dependencies
    - automaticexperiment.jar
    - analysis.jar
- /scripts
  - /tests
    - runAllTests.sh
  - /build
    - build.sh * *Places .class files under /testCaseExecutables*
- /testCases
  - testCase1.txt
- /testCaseExecutables
  - NelderMeadTestDriver.class
  - ElapsedTimeTest.class
  - STEMTimeTest.class
- /temp
  - NelderMeadTestDriver_testnumber1_2020-11-04T22:24:07.txt
- /oracles * *May not use…*
- /docs
  - README.txt
- /reports
  - report_time_and_date.html

Testing Framework

runAllTests.sh under the directory "/scripts" will run all tests that rely on input from "/testCases." Each file under "/testCases" will contain csv-formatted inputs and oracles for each test case. Both the inputs and oracles for each test case are passed into the main methods of a driver specified, again, within the test case file. In other words, these scripts will execute the Java classes (our drivers), passing in the inputs and oracles from the test case files to their main methods, and these classes in turn write the results of the tests they run to a file under "/temp" with strings in the format:

> **Test Case X has failed/passed.**
>
> **Expected output: [Oracle]**
>
> **Computed result: [result of test case]**

The text files are then combined to make a html document.

The script runAllTests.sh, for example, writes the test results in the above format to a file titled "runAllTestsResults.txt". Once all tests have had their results written to this file, the results of the tests will then be displayed in the default browser.

For the testing framework to run, the .java files containing drivers for our test cases will need to be compiled under /project with the necessary .jar files produced by our local STEM build. The results of this compilation will be placed under "/testCaseExecutables". The above process will be handled by a script "build.sh". In other words, this script will be able to produce executable test cases without the user having to build the entirety of STEM. Instead, all they will need is the .jar files which contain the classes required for our test cases to run.

Test Cases

- **org.eclipse.stem.analysis.automaticexperiment.NelderMeadAlgorithm.java**

Method	Inputs	Oracles
execute () **SimplexFunction (-3x + x^2 – y -xy + y^2)**	**0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .5, .5, SampleFunction2**	**2, 1, -4**
execute () **SimplexFunction (-3x + x^2 – y -xy + y^2)**	**0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .1, .1, SampleFunction2**	**2, 1, -4**
execute () **SimplexFunction (-3x + x^2 – y -xy + y^2)**	**0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.6, 1.2, .5, .5, SampleFunction2**	**2, 1, -4**
execute () **SimplexFunction (-4x + x^2 – y -xy + y^2)**	**0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.8, 1.2, .5, .5, SampleFunction**	**3, 2, -7**

execute () SimplexFunction (-4x + x^2 – y -xy + y^2)	0, 0.0, 9999999.0, 1, 0.0, 9999999.0, 1.6, 1.2, .1, .1, SampleFunction	3, 2, -7

- org.eclipse.stem.analysis.automaticexperiment.ElapsedTimeTest.java

testAddincrement_long ()	[currentTime]	currentTime + 1day

- org.eclipse.stem.analysis.automaticexperiment.STEMTimeTest.java

testGetElapsedMilliseconds()	40	Valid, 3456000000

Resource Constraints

STEM will require the user to have an internet connection to install open-jdk-8 and a tool called “txt2html” which we use to help create our reports. The memory required to install our testing framework will be minimal, likely within the range of a few megabytes.

Dependencies

One of the main issues we are facing running our code in the terminal is the sheer number of dependencies required to actually run many parts of the code. Finding parts of the code that don't have an astronomical amount of dependencies proved interesting as some of the required packages to run code were needed for code that was extended several times. This made actually finding all of the required dependencies/packages the most frustrating part of trying to get some pieces of code to work. To avoid this issue we are going to have to be more selective in how we choose code to make into unit tests. The code we actually select must have limited dependencies to streamline that aspect of the test creation and allow us to test the code more thoroughly.

For the user to not have to deal with such dependencies, we simply take the .jar files produced by the full STEM build required to run our test cases and place them within our testing framework so that our test cases and compile with them on the user’s machine. This enables us to make changes to our test cases without having to worry about building the entirety of STEM with all of it’s dependencies.