Joey Baldwin, Eric Hofesmann, Marge Marshall

CSCI 362 Software Engineering

October 23, 2016

Chapter 3: Automated Testing Framework

Experiences:

Our team communicated remotely through Slack and ended up meeting on Sunday afternoons and evenings. Our work style was that of working in parallel on different aspects of the project and communicating on any problems we ran into to see if other group members had solutions. We ran into some snags with using the python unit testing framework, but ultimately decided to continue using it, as it provided us with experience using a standardized unit testing framework and would accomplish our goals for this assignment.

How-To Documentation:

Requirements: Linux OS, Python 2.7

1) Clone GitHub Repository:

git clone https://github.com/CSCI-362-01-2016/Cygnus-Inter-Anates

2) Change directory to scripts

cd Cygnus-Inter-Anates/scripts/

3) Execute runAllTests.sh to run the framework and open the html file in the default browser.

./runAllTests.sh

Architectural Design:

The testing framework is built around python's standard testing suite, unittest. Two python scripts create our automated framework: runTests.py and jythonMusicTestCase.py. These scripts read the testCase.txt files.

Test cases are stored in text files ranging from testCase00.txt to testCase24.txt. More can be added at any time. The structure of these can be seen in the section below entitled “Test Cases”.

JythonMusicTestCase.py contains two classes, JythonMusicTestCase and CustomResults which inherit from unittest’s TestCase class and Results class respectively.

JythonMusicTestCase class inputs the contents of the testCase.txt files and imports the required file dependencies to run the test’s intended method. These methods will be executed depending on the assertion provided. This class contains multiple assertions where the intended assertion is dictated in the testCase.txt files. These assertions are either testequal, testalmostequal, or testexception.

CustomResults is a class that will compile the results of the assertions along with the inputs. These are then used to generate an html file using a method within this class. One CustomResults object is created for all test cases and will store all assertion outputs.

runTests.py contains one class, called SetupTests, which creates a testSuite object that will contain all jythonMusicTestCase objects. It also contains a method that parses the test case text files and extracts the information. The suite is run in the main of this file, thus if it this program is executed, it will generate an html file of the test results.

File Structure:

/Cygnus-Inter-Anates

- /docs
  - README.txt
- /project
  - /src
    - Jython, jMusic, jsyn, and imgscalr source code
    - gui.py
    - music.py
    - image.py
    - timer.py
  - /bin
    - Compiled files
- /scripts

jython.sh

runTests.py

runAllTests.sh

/temp

/testCases

testCase00.txt

testCase01.txt

...

/testCasesExecutable

jythonMusicTestCase.py

/reports

report.html

Test Cases:

The template for our test cases is as follows, with a description below:

<test case id>
<description of function's purpose>
<module name>
<function name>
<input to the function>
<test type of expected output>
<expected output>
The test case id is designated with two digits, and starts with the first test case at 00.
The description is a plain English explanation of what the function does.
The module name is the module or file in which the function is found.
The function name is the name of the function being tested.
The input is the current value or collection of parameters being passed to the function.
The test type dictates what type of assertion will be used to determine if the actual output passes or not. This is given in the form of the name of the test functions, such as testexception or testequals. The expected output is what the function should return, given the input parameters.

Current test cases:

00
calculate the frequency value of a given midi pitch
music

noteToFreq
69
testalmostequals
440.0

01
calculate the frequency value of a given midi pitch
music
noteToFreq
48
testalmostequals
130.81

02
calculate the frequency value of a given midi pitch
music
noteToFreq
72
testalmostequals
523.25

03
calculate the frequency value of a given midi pitch
music
noteToFreq
1
testalmostequals
8.66

04
calculate the frequency value of a given midi pitch
music
noteToFreq
127
testalmostequals
12543.85

05
Create range of floats
music
frange
1.0, 5.0, 0.5
Testequals
[1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5]