17-654/17-754: Analysis of Software Artifacts
Jonathan Aldrich, Nels Beckman, Kevin Bierhoff
Assignment 3 (Programming): Test Case Generation

Due Tuesday, February 13, 10:30am

Turn in a zip file named <username>-17654-A3.zip, where username is your Andrew id, electronically through Blackboard. The zip file should contain the following files:

- *Answers.txt*
- Your modified test files, *OthelloBoardAgitarTest.java*, and *OptionsAgitarTest.java*.

Note: *This assignment is* ***not*** *to be done in teams, even though it builds upon assignment 2 which was done in teams.*

100 points

Assignment Objectives:

- Run a unit test generator.
- Gain an understanding of the benefits and drawbacks of automated unit test generation.
- Gain a better understanding of testing against a contract.

Part 1 (20 Points)

In this section, you will run an automated unit test generation tool. The concept of automated unit test generation is a relatively new one, and therefore there are relatively few good, free tools. Fortunately, Agitar Software has a free trial of their unit test generation software that can be used via the Internet. The first part of this assignment is installing this tool, generating unit tests and examining the coverage that they provide. Before you can use the tool, you must register.

Registration and Installation:

1. Go to http://www.junitfactory.com/request-invite.jsp
2. Fill in your email address, your first name and your last name. You should receive an email giving you an address to visit.
3. When you receive the email, it will direct you back the homepage. You should click on "download," which will allow you to "accept the invitation," and will give you instructions on installing the Eclipse plugin.

Generating and Running Unit Tests:

4. Once you have installed the plugin, go to the "BoardGame" project that you should still have from assignment two.
5. Automatically generate unit tests for the three classes, "OthelloBoard," "Options," and "OthelloMove," by right-clicking on those classes in the package explorer, selecting "Agitator," and then selecting "Generate Tests."
6. Run these tests, by right-clicking on the "edu.calpoly.csc435.othello" package, selecting "Agitar | Find tests for edu.calpoly.csc435.othello," and then selecting

"Run" on the ensuing "Find Tests" dialog box.

Task: Measuring Code Coverage:

7. Open the "Coverage Summary" view by going to the "Window" menu and selecting "Show View | Other | Agitar | Coverage Summary."
8. For each of the three classes, OthelloBoard, OthelloMove and Options, determine the coverage percentage of the automatically generated unit tests. Compare these values to the coverage percentage of the unit tests your group created in assignment 2 for the three mentioned classes. Record all six coverage values in *answers.txt*.

Part 2 (60 points)

In this section you will judge the ability of the automatically generated unit tests to find bugs in the Othello program. We will do this by pointing out three specific, existing bugs in the Othello program.

Bug A: Options Cloning

The "clone" method of the "Options" class does not perform a proper deep copy of the weights as it should. (See line 53 of Options.java.) This is due to the fact that the "mWeights" field of the new Options object is assigned the "mWeights" field of the original object. In reality, the copyWeights method should have been called. There is another problem: In the "getWeights" method (See line 76 of Options.java) the 'r' and 'c' indexes of the "mWeights" array have been put in the incorrect order.

1. In the OptionsAgitarTest.java file, a method testClone() was automatically generated to test the clone() method. Yet, testClone() passes when run even though the clone() method contains a bug. Why does this test not find the bug?
2. Change the testClone() method so that it fails when run on this bug but succeeds when the bug has been fixed. Turn in the entire OptionsAgitarTest.java file.

Bug B: Board mValue Invariant

In class we talked about unit testing against a class's or a method's contract. As it turns out, the OthelloBoard class has an invariant that is not properly preserved. The value of the "mValue" field should always be set to the [sum of all white moves multiplied by the weight of those board locations] minus the [sum of all black moves multiplied by the weight of those board locations]. We have included brackets to make the order of operations explicit. Note that the "setOptions" methods of the "OthelloBoard" class *do not* preserve this invariant. If we are changing the weights of the squares on the board, the "mValue" must be adjusted accordingly. These methods violate their contract!

1. In the OthelloBoardAgitarTest.java file, there is a automatically generated method testSetOptions() that tests setOptions(). Why does this test not find the invariant violation bug?
2. Change the testSetOptions() method so that it fails on the buggy code but succeeds when this bug has been fixed. Turn in the entire OthelloBoardAgitarTest.java file.

Note: If fixing these bugs makes the assignment easier, go ahead and fix them. We will be running your modified tests on a working code base

Bug C: Board Comparison Bug
The "compareTo" method (See line 211 of OthelloBoard.java) does not fulfill its contract. According to the comments, the sort order of the four example boards should be BADC, but when testing this board we find that this is not the case.

1. Again, there is an automatically generated test method, testCompareTo(), in the OthelloBoardAgitarTest.java file. Why does this test not find the bug in the sorting order?
2. Change the "testCompareTo()" method so that it fails on the buggy version of this code but succeeds when the bug has been fixed. Turn in the entire OthelloBoardAgitarTest.java file (

Note: Don't turn OthelloBoardAgitarTest.java in twice, just once with the new tests for bug B and bug C.

Part 3: Final Questions (20 points)
Please answer the following questions in *answers.txt.*

a) Now that you have seen some automatically generated unit tests, what is your general impression of their ability to find existing bugs in an application?
b) Why do you think that this is the case? (*Hint: Remember that the purpose of unit testing is to test a method or class against its contract.)*
c) Can you think of certain situations where automatically generated unit tests would be particularly helpful?