**CS 1699 - DELIVERABLE 2: Unit Testing and Code Coverage**

**for Coffee Maker Quest 1.0**

Mike Byrne & Jamie Rossi

mjb187@pitt.edu

wjr18@pitt.edu

https://github.com/wjrossi/CS1699

CS 1699: Software Testing

Bill Laboon

February 17th, 2015

**DESCRIPTION OF ISSUES**

Perhaps the most universal issue in testing is simply figuring out what needs to be tested. In this assignment in particular, this proved difficult because we had not authored the code and were not familiar with it. For most tests, we attempted to provide at least one “normal use” case and at least one edge case; for example, a normal use case for House.generateRooms(int) would constitute a small positive number, while an edge case for that same function would be zero.

However, the nature of many functions such as the ones found in the Room class made even coming up with edge cases; most of the Room class functions simply return the value of a private class-level boolean variable. That leaves an obvious testing set of the case when the variable is true and the case when the variable is false. We were uncertain whether or not there should be additional cases to truly test the code. One suggestion might be to test the function when the class level variable is null, but since these variables are set in the constructor, trying to set them to null would result in code that cannot be compiled due to its errors. Perhaps a mock could assist us here, but then we had to consider if testing these functions in this way was even necessary, as it would be impossible to change the private variable from anywhere but the class level and the variable is guaranteed to have a valid value due to it being set in the constructor.

Additionally, this issue of accessibility came up often while devising tests. Again in the case of the Room class, there is a Room.getDescription() function that returns a description generated from a set list of adjectives using a pseudo-random private static class-level integer. At first glance, it’s obvious that two Room objects can have the same description if this variable is not incremented after use. However, that incrementing code is in the function, and since the variable is static and private, there seemingly cannot be a case where the variable is decremented to cancel out the increment and cause an issue. We decided then that attempting to circumvent these protections with a mock would be unnecessary because we would be testing for an incredibly unlikely scenario outside of something like concurrent accesses.

<paragraph about void functions>

Note to self: add Room test that tries to increment above-mentioned variable to an absurd number

**FAILED TESTS**

<tests that failed and thoughts on why>

**EXECUTED UNIT TESTS**

<image>

**CODE COVERAGE**

<image>