CS 1632 – DELIVERABLE 2:

Unit Testing

CitySim9003

<https://github.com/kingsman142/CS1632/tree/master/Deliverable2>

James Hahn

Introduction

**Writing the Code**

The first step to this project was thinking of a strategy to tackle all of the requirements of the program. Initially, this was extremely easy. I went with an object-oriented design, with my main class along with a Driver class. The game/simulation worked perfectly fine. However, it proved difficult to test. There were only about four different methods (albeit with good design and short enough length). Obviously, the program had to be split into smaller components to be worthwhile testing. So, after some time, the program went from one main class, a helper class, and four methods to a main class, two helper classes, and thirteen methods. To further increase the object-oriented design of the program, the City class was created to generate new locations and directions for the Driver class. Although my first code layout for the program had maximum security and excellent object-oriented design, an issue for this deliverable was that it needed to be further split to fulfill the test requirements (number of mocks and stubs).

**Testing the Code**

Then came the most important part of the deliverable, the unit testing. These proved to be tiresome and painful to implement as the project was split up (as mentioned above), as more tests were needed than perhaps could have been needed. The main issue that repeatedly appeared was handling random movement. The drivers were required to move in random directions, so a challenge to figure out was how to stub the randomness of the program. This was the first obstacle that I encountered.

The next major issue that popped up with unit tests was handling every single kind of input. For example, if a getNewLocation(String currentLocation, Random rand) method is being written, and a null String object is passed in, what should be returned? A null String, empty String, or the currentLocation String for the new location? As a developer, deciding the output given special cases proves to be difficult. Many different paths can be taken depending on how the method is used by other methods.

Lastly, a concern that developed was making sure the program was testable without being prone to low security. As mentioned earlier, this program required splitting up to help in the development of tests. Accompanying that came methods that required arguments to be passed in. Obviously, if arguments are passed in, that means the end-user can easily screw the method up. Allowing more arguments to be passed in turns out to be a bad practice because if a special case is not covered, the entire program breaks. However, if arguments cannot be passed into the method at all, the program proves difficult to be tested.