CS 1632 - DELIVERABLE 2: Unit Testing CitySim9006

Sumanyu Gupta

sumanyugupta

<https://github.com/sumanyugupta/CS1632_Deliverable2>

Introduction

To start this project, I first sat down and tried to sketch out the initial classes and methods I planned on using. At first, I did not have a large number of methods or more than 2 classes. Instead, I tried to stuff all the code into giant methods in one class. My biggest priority was to first get the program working and then focusing on altering the code if I needed to. After I got the program running, I could began testing and I soon realized that it would have saved me more time and effort had I tried to follow the methodology of Test-Driven-Development from the start. It proved difficult to test the few methods I had and even the initial tests I wrote were repetitive. In order to increase my testing capacity and make my program more object-oriented, I began sprouting methods from the previously giant functions and created a City class to store and manipulate the streets represented on the map of Oakland.

From beginning with two classes and one helper method, I was able to divide my program into four classes and twelve methods at the end of my development stage. I also began writing pseudocode for tests that I would be creating for the requirements before I actually wrote the code for the new methods and classes. Using the pseudocode as a blueprint turned out to assist me in saving more time and being more cognizant of the required edge cases needed to be tested as well.

The most difficult challenge I faced the process of unit testing was to understand and implement Mocks and Stubs. I looked at lecture notes, YouTube videos, and other resources before finally understanding how to write tests using the aforementioned tools. Additionally, the hardest thing to simulate was running the program with random values generated based on the seed value passed in by the user. Since Ruby does not really have a maximum value for an Integer, it proved difficult to write a test for this situation because it was not easy to write a simple edge case. Also, another challenge I faced was to figure out multiple equivalence classes for a given method. On more than one occasion, the output of a method was only dependent on multiple different single values. For example, moving from one location to the next was a binary decision. If 0, I would move to one place. If 1, I would move to the other. Since the seed passed in by the user did not change the actual number (0 or 1) generated, it was difficult to describe how I partitioned those classes. Finally, it took me many repeated attempts to properly write tests simulating actual runs of the program in order to test item values based on different seeds “passed” in by the user.

In conclusion, while this project provided lots of challenges and was difficult to test at times, I learned and implemented the important concepts of Stubbing and Mocking, began writing programs following a more TDD fashion, and realized the importance of developing a variety of unit tests that covered unique cases.

