CS 1632 - DELIVERABLE 2: Unit Testing

https://github.com/mjb236/CS1632

Michael Bowen

Introduction

One of the biggest challenges I faced when trying to figure out exactly how to link up the different classes. I tried to develop the program in as much of an object-oriented fashion as possible. I even used inheritance, which honestly, hasn’t come up much for me so often due to the last few semesters focusing on C programming. The main issue was trying to figure out how the Locations and Roads would interact with each other. The thing is Roads link to Locations and Locations link to Roads. I am actually not very happy with how I solved the issue – both Locations and Roads (well, their subclasses Avenues and Streets) have references to the other type of object, which leads to a convoluted mess of references. I think I may have over-engineered the problem in the attempt to adhere to object orientation. It would have been much simpler to just leave Roads out entirely and imitate their behavior when trying to leave a location. Try not to judge the leaky/lack of abstraction too harshly.

The above issue also led to some problems with testing the Driver class. The Driver class’s move() method is where the meat of the program rests. This is where the various classes’ functionalities meet. When I first wrote the move() method, it simply increased the Driver’s number of moves. But when actually adding in the core logic of the movement, rewriting the tests for that function was a real hassle. The move() method calls functions from Location and Road, which rely on what those functions return, so I had to set up a lot of stubs just to make sure the function was incrementing the number of moves. I’m not very satisfied with how this actually works in the code, but refactoring would require more time.

As I was writing methods and tests, I actually ran into some trouble coming up with ways to incorporate method stubbing. The concept makes sense, but it seemed like the way in which I was writing the methods did not quite lend themselves to stubbing. Most of the methods are setters and getters and don’t actually rely on other class methods. I tried to come at the project from a somewhat TDD direction, or at least an approximation. I would write tests and functions concurrently, but my discipline in this respect unfortunately waned as time passed.

Almost as if guided by fate, I did something really dumb that actually helped me meet the quota of method-stubbing test. I was looking into the Random Java class and my eyes must have completely glazed over and I did not notice the function that would return an integer up to a certain range. So, I wrote a wrapper class that would simply return a random number from zero to the provided upper bound minus one. I left this in the project simply because I had already written some stub tests using the RandomNumber class and figured the point of the exercises was to demonstrate the use of stubbing, not what you actually used them to replace.

Screenshot

