CS 1632 - DELIVERABLE 2
Unit Testing Ruby Rush

Zachary Whitney
Github Username: MisterZW
D2 URL: https://github.com/MisterZW/D2

Class Section: T TH 9:30

+2 ISSUE #12

D2 Retrospective:

Algorithmically, this deliverable was straightforward. We have lots of experience writing code to meet program specifications from the CS curriculum, so it was not especially difficult to write a reasonably clean and readable solution.

Rubocop linting added a bit of extra complexity, but most of its suggestions are things I already strive to accomplish. Short methods and lines, descriptive variable names, and documentation are all pretty automatic. The tricky parts were the things more specific to Ruby. For instance, preferring implicit returns, adding blank lines after explicit return guards, and using .zero? methods for testing against 0 were new considerations for me.

My code has 3 remaining Rubocop warnings and 2 of them are complaining about a form of using rescue which I don't fundamentally understand well enough to fix. The last is one I've violated intentionally (using an explicit multiple return). I think "return rubies, fake rubies" is more self-documenting than using an array as an implicit return type and then needing to refer to rubies as gems[0] and fake rubies as gems[1] strictly by ordinal in the calling context. I appreciated the more lax .yml file provided, as the default linting configuration read my if, else if, else if, else chain as unacceptable both in perceived and cyclomatic complexity even though all it did was fiddle with grammar in a string according to 4 possible cases.

Unsurprisingly, the most difficult part of Deliverable 2 was devising a sound and comprehensive unit test suite. I found Minitest pretty intuitive for unit testing on the whole. It took some time to reference documentation on what I wanted to accomplish, but nothing out of the ordinary. My problem was just that some of my source code was not written with testability as a paramount concern.

The prospector and args classes did an okay job with isolating pure functionality which made the tests easier to write. The location tests were harder, and I resorted to some nondeterministic iterative tests on random_location? because I was worried that mocking the randomization might mask boundary issues with the method under test, though I'm sure there's a better way to accomplish the same goal.

My worst problems with the testability were on the graph class. I think the main issues there are the hard-coded values and lack of dependency injection. It was essentially impossible to mock the dependencies the way it was written, so I had to test the class more holistically and less thoroughly than I would have liked. Even so, I think it was helpful to write the code all the wrong way because it made it much more clear why I should have done it differently in the first place.

Grading Notes:

- ruby_rush.rb is 20 lines or fewer and thus exempt from test coverage as per the deliverable requirements
- All other source files have associated tests which are called by all_tests.rb. The tests themselves are located in the /tests directory of the Github repo. This makes it possible to test all files with Rubocop by calling rubocop *.rb from the main directory without linting the test files themselves.
- Test run was executed with Ruby version 2.5.1 on Ubuntu Linux 16.04