HW7 Feedback

NOTE: Even if you didn't get marked down, this is very useful to read

MVC Quality

- **M/V:** Model contains functionality of the view/controller
  - Typically, any class that exists primarily to store data is part of the model even if it doesn't contain much logic. For example, a class that represents a single point on campus and has only simple accessor methods would be part of the model.
  - Subtle point: because the view/controller shouldn't know how data is stored, it shouldn't supply filenames to model. It's fine to pass in filenames to make the model more flexible, but ideally this should be done in an external main class

- **V/C:** View contains functionality of the controller (for those students who make an effort to separate the view and controller.
  - The View is responsible for all output to the user. This means that there should never be something like printlns in the Controller code.

- **Documentation Encapsulation:** The model's public documentation, class name, and method names should not expose the internal fields. In particular, the client should not know that the model represents data with a graph, as this is an implementation detail. You could imagine trying a new approach to path-finding (e.g. calling the Google Maps API) in which the model no longer uses a graph.

- **Graph Encapsulation**: Like documentation encapsulation, except the view actually gets a reference to the Graph or to Vertex/Node or Edge objects that are part of the Graph ADT. This is worse than simply mentioning the graph in the documentation because eliminating the graph will now break the code.
  - **Returning Path as Nodes/Edges:** Path is returned a list of node or edge data types.

- Please come talk to us if you don't understand why you were marked off.

Other Code Quality

- **Reusing Dijkstra's algorithm:** Most people either duplicated Dijsktra's algorithm from HW6 or called their MarvelPaths2 application. As mentioned after HW6, ideally your pathfinder should be in a reusable generic pathfinder class:
  - Duplicating code is just a bad idea: if you make an optimization or fix a bug, you have to change it in both places.
  - MarvelPaths2 was presumably written specifically for the Marvel application and not intended to be reused elsewhere. If you decided to change something in MarvelPaths2, you shouldn't have to expect that it would break some other, unrelated application.

- **Changed Graph ADT:** Altered Graph, Node, or Edge ADT to application specific use. Your graph ADT should still be completely reusable.

- **Dynamic instance fields as Static:** Declaring instance fields as static when they shouldn't be. If a field changes over the duration of the program, that means it has state and usually should be an instance field. Another way of thinking about this is, could you have two objects of the same type that don't and shouldn't share the same state? (e.g., two graphs built from different data sources?) If so, that state should go in an instance field so each object can keep track of it separately. (A number of students wrote Views/Controllers that were completely static, which only works because the console interacts with the static 'System.out' and 'System.in' resources. In these cases, we were more lenient about static instance fields.)

- **JD** (Javadoc): Missing class or method comments or Javadoc tags used incorrectly

Test Quality

- **JUnit tests / test coverage:** The specs asked for JUnit tests for each model class. We deducted points if some classes were missing JUnit tests without a commenting justifying their omission (but less so if you had comprehensive specification tests). There seemed to be some confusion in that previous assignments asked you to focus on writing test scripts rather than JUnit test classes. The distinction was that the test scripts for past assignments were more or less still testing a single module (your graph, MarvelPaths, etc.), so they were still unit tests. In this assignment, the test scripts were testing an entire application – view, controller, model, the whole shebang. The fact that they invoke main to launch an entire application, rather than calling methods on an ADT, makes them more of integration tests.

  In hindsight, we should have clarified this distinction. We did also account for your overall test coverage (both implementation *and* specification tests), however, so that your grade would reflect the overall time and thought you put into testing.

- **Test Coverage**: Does not cover an adequate domain of inputs.

- **Test Documentation**: Non-descriptive test names or comments.

- **Test Granularity**: Tests do not represent coverage of a single domain. Each test (a method or a script file) should be short and test one specific case.

- **Test Naming**: Name of test does not indicate the condition being tested.