**MARS: Test Plan**

**Test Types**

* **Unit Testing**

Unit tests will be written as we complete different Epic cycles of development to reflect new features implemented in our codes behavior. This suite of tests will be included within our repository for the convenience of our developers so that they may ensure that new code they write does not negatively affect any previously established passing code behavior. These unit test will range from testing expected function return behavior to proto-user acceptance testing wherein we establish expected experience behavior as a requirement to pass the build. We are implementing TravisCI within our develop repository in order to complete continuous integration testing as we proceed to build out the systems functionality. This service will test each version-control-committed build across our entire suite of unit tests and indicate via email as well as build information displayed within GitHub itself to every developer in order to ensure that each future feature we implement does not negatively affect any previously established passing code behavior, especially when merging our feature branches to our development branch and our development branch with our main project branch.

* **Integration Testing**

As of yet, integration testing is not an essential priority as our dependencies are being managed by Maven to allow for instant development compatibility across systems and there are no hard-set hardware restriction requirements in which we will need to consider aside from that of desiring to improve performance (ie limited ram and processor cycle speed) as development progresses.

However, we will have tests, while appearing just like their related unit tests, which test the ability for our program to run successfully using our intended, large (2GB) GeoTIFF of Mars’ surface. This/these tests are to be considered integration tests as they are not included in our unit test suite in order to test functionality, but rather are to be manually run within an environment to test if our code’s integration on a specific machine is capable of successfully running as expected.

* **Performance Testing**

We do not yet have any performance data as the state of our code has only just reached a point wherein our classes are fully interacting with one and other. Performance data will be tracked in the future when iterations of our path-finding algorithm are complete and we would like to see how our changes affect the efficiency of our algorithm when dealing with varying sizes of map files.

**Test Authoring**

All developers will be responsible for writing tests relevant to their changes when developing their code. Dean Moser will be responsible for overseeing these tests and complete tests for corner cases during interactions between classes.

**Performing Tests**

TravisCI is the main tool completing the test suite and will run every time we:

* Commit to a feature branch; on said feature branch
* Perform a pull request to merge a feature with the dev branch; on the proposed merged-code version
* Perform a pull request to merge the dev branch with the main branch; on the proposed merged-code version

**Tracking Results**

We will be tracking performance results within our documentation and tracking its history by updating it per release within GitHub.

Any expected but “to-be-changed” behavior will additionally be noted within our documentation and version controlled. JIRA bug tickets will be created for each of these “to-be-changed” features in order to account for their existence when moving through future development cycles.

We will be using TravisCI build history in order to track our unit testing results as development progresses. TravisCI also has a convenient way to embed a build’s passing or failing status within the readme file that is automatically displayed on GitHub and we will be utilizing this feature in order to indicate our development and main branches’ test-suite status.

**Potential Difficulties**

The GeoTIFF will be the hardest to test because of its 2GB file. It is hard to run multiple tests on the same GeoTIFF because it takes forever to run. With respect to this challenge, any unit tests written to test our “map” file behavior will be run with a smaller, more manageable file that is able to be stored within our repository, so that we ensure TravisCI’s ability to run these tests as well as any developer who would like to ensure that a behavior works without necessarily needing to test it with a large file that will take much longer to run.

**Test Epics**

**CLI Testing 11/27**

user input – separate into input grabber and handler functions so that handlers may be unit-tested

- map path (<string>)

Throw exception if relative path file cannot be opened (may not exist or incorrect path)

? Throw exception if relative path file is of incompatible file type

- max slope (<double>)

Throw exception if non-number (automatically convert if given number pseudo-type)

Throw exception if out of range 0.0-90.0

- start coordinates (<int> <int>)

Throw exception if either entry not int

Throw exception if either entry out of range <x-range> <y-range>

- end coordinates (<int> <int>)

Throw exception if either entry not int

Throw exception if either entry out of range <x-range> <y-range>