Math, but More - Test Plan

Unit Testing

We will be testing running unit tests for each individual applet that we will be running. Unit testing refers to each applet because each applet has its own purpose and requires individual testing, which is included below.

Word Problems Applet

Inputs expected: integers floats or strings that fit the nature of the problem at hand; restrictions are yet to be determined

Outputs expected: a text blurb that confirms or denies the user has answered the question correctly

We will test the handling of text responses versus numeric responses, as well as a response in decimal form versus integer form. We will also test for special characters. We will also test that the formulas for the given word problems to be sure that they are accepting the correct answers. A successful pass of this test would be indicated by only accepting the correct integer input and reject any inappropriate input.

Back-Mapping/Homography Applet

Inputs expected: a .jpg that is within an expected resolution that will be listed on the screen and is yet to be determined

Outputs expected: a image that has been altered to fit the dimensions of another other so it looks 3 dimensional

We will test the use of .jpg images and there conversion to matrix form. We will also test matrix transformations that would result in invalid images (setting a pixel value below 0 as well as setting it above 255). We will also test the use of drag and drop library that we will be incorporating. These tests will involve using the same drag and drop tool multiple times as well as handling for dragging items off screen. This test will pass successfully when a picture is able to be modified by a user.

Transformation Matrices Applet

Inputs expected: 3 inputs: scale:(integer or float 1/10 to 10), rotation: (integer or float -360 to 360), and translation: (integer -1000 to 1000) default will be given if none are recieved and no other inputs will be expected

Outputs expected: an image in the interface that reflects the changes the user has implemented

For the transformation matrices we will test for invalid input (using text instead of numbers). We will also test rotating beyond 360 degrees as well as negative rotation. We will also test translating beyond the limits of the screen to make sure that the image does not disappear. We will also test the ability to reset the image to the start view. A successful test will pass when the inputs will accept decimal inputs and can properly handle the screen size.

Programmable Robot Maze Applet

Inputs expected: position given by a mouse click

Outputs expected: a changed position given by input and events in the applet may affect the nature of the user's avatar (see details below)

For the programmable robot maze we will test if the user given code is handled so that it won't break the site. We will test that the code is only handled by the robot and that any errors that occur with the user code will be displayed to the user. We will also test unexpected movement of the robot (i.e. running off the map, moving backwards through the start, just trying to go through the wall). The programmable robot maze will pass the unit test when it can properly handle code injection as well as properly being able to move in the given units.

Angles.yum Applet

Inputs expected: position given by a mouse click

Outputs expected: a changed position given by input and events in the applet may affect the nature of the user's avatar (see details below)

For Angles.yum we will test the addition of angles that are being added when a snake 'eats' an angle. We will also test the collisions (i.e. the snake colliding with the angle, with itself, with a wall, etc.). The other unit testing will be to see if there is proper mouse control in game (i.e. the snake will follow while the mouse is moving, it will stop if it is not moving). Unit testing will be successful when the proper addition and subtraction of angles are handled as well. Success is also contingent on the proper handling of collisions.

Negation Reflex Applet

Inputs expected: integers or floats in the range of (-100, 100); will not handle other inputs

Outputs expected: and integer or float depending on the problem or an error string

The Negation Reflex Applet is expecting user input so we will test the use of text versus numbers. We will also test decimal versus integers and expected outcomes. This test is simpler due to the simplicity of the functionality. Successful testing will be concluded when the code can handle 100 correct subtractions, 100 false subtractions, and verify the correct input type.

Integration Testing

For integration testing we will have a strong focus on integrating the programmable robot maze with the react applet package. This integration is important because it requires a complicated user input and needs to be integrated into the existing react code base. This part of the integration test will be successful if a user is able to navigate to the maze and return to the main page with the other applets. This integration applies to the other applets as well. The other integration testing is dependent on all the navigation buttons functioning and navigating to the correct pages. So successful integration testing will run when all the applets can be navigated through properly.

System Testing

System testing will test that each applet can be accessed and used. It means that each applet will take a given input and an output will be given. Success will be achieved when every output has the expected result.

Regression Testing

Our regression testing will be handled through github and the developed test suite for our system testing. Before our code is merged into the master branch, it will be ran against the test suite to ensure that everything works according to specification. This will include that no existing code will break when new code is introduced. Successful regression testing will occur when a full system test is runned and passed according to the expected outputs.

User Testing

We will be doing a first wave of usability testing on our application from November 5th to the 10th. Usability testing will involve using our targeted age group to test the app and give us feedback on the feel and if everything makes sense in how it was set up. We will also include a second wave of usability testing from December 3rd the 10th. These user tests will allow us to make the app easier to use for the given test group.

Performance and Load Testing

Load testing is contingent on the AWS server we are using. Due to the rapid nature of this project and the short time we have, we will not do any load tests. Our plan as of now does not include a lot of web traffic. We can plan to expand this test if the project continues after the class is complete. Performance testing will be included for applets that may need heavy computing. As of now the only applets of concern are the homography and transformation matrices applets. We will evaluate the performance based off of feel at the completion of each applet. If a consensus is reached that the applet is running below expectations we will incorporate a timer function into our test suite to evaluate that each applet is able to run in adequate time.

Testing Schedule

As previously stated in the regression testing section above, our tests will be ran against the code every time there is going to be a push or merge to the master branch. This will allow us to know that every time code is added it will function properly with the existing code base and that the code being added works as expected.