**Unit Tests**

/\*Comments inside of unit testing and what i want the individual unit tests to do\*/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter an invalid school or a school that doesn’t exist

Enter an invalid classroom/ teachers name

Catching these errors and then calling CanRegister() to assess whether it is available or not to register

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CanRegister(){

/\* Checks if people can register properly\*/

/\*also checks that once registered they can login with the registration information\*/

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter a negative amount to donate

Enter a ridiculously large number (unfathomable)

Enter nothing and continue with transaction

After doing the following each time calling CanDonate() to ensure the module works as intended

CanDonate(){

/\* Checks to see if donating works \*/

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PlayGames(){

/\* ensures the games are up and on the site (cannot check if playable from code as i know how YET)\*/

}

CheckProgress(){

/\*checks the progress of an individual child (rng’ed)\*/

}

CheckLeaderboards(){

/\* Check the leaderboards inspect top, inspect bottom, check for someone individually \*/

}

Database(){

/\* Does it fill properly, does it take information\*/

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter invalid (inappropriate) username

Enter

After entering invalid values into our profile, checking Profile() to see if we can still fill out profile with valid names successfully.

Profile(){

/\* Does the profile work as intended, can we fill the profile and it fills correctly \*/

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Integration Test - Bottom Up**

PlayGames

CheckProgess

CheckLeaderboards

CanDonate

Profile

The purpose of integration testing is to expose the faults in the interaction between integrated units, and it does this by testing individual units as combined group. For example, when we test our checkLeaderboards() class alone, it may work fine, but will it work straight from the profile of our user? Or right from the end of a game played? This is what integration testing tests.

For our integration testing, we will test a series of classes to ensure they work in every scenario.

We will test to see if the following works in context:

* Playing games from the profile screen
* Donating from the profile screen and at every screen
* Checking progress from the profile screen OR after playing a game
* Checking leaderboards from the profile screen OR after playing a game

Also similarly in a reverse order:

* Going to the profile screen from the leaderboards, progress, or donation screen

**Regression Test Procedure**

Regression testing is required when we make a change to our existing code or program and we need to ensure that the recent change does not affect any of our existing features.

Most importantly, we will prioritize our test cases depending on critical and frequently used functionalities. Doing so, we will make our most used functions a priority and this will reduce the overall testing time spent.

To make this test, let’s say that we made an upgrade to our PlayGames() class. Now, we not only need to test the new code itself, but we need to go back and test it while integrated, just like we did in our integration testing. This thorough testing will ensure that the new code not only works by itself, but works in any integrated context. Performing regression tests will help our programs to work cohesively and will prevent a change in a certain spot from introducing problems to other parts of the program.

Our Regression Test Procedure looks like this:

* Plan which changes need to be made to the program
* Analyze test cases and prioritize which functions are the most crucial that need changes
* Determine which other components of the program that might be impacted by this change
* Evaluate test cases and determine the ones that fit our situation best
* Schedule the time needed for the test and begin testing

**System Test Procedure**

After performing our unit tests and integration tests, it is important for us to test the entire system for errors and bugs. In addition, we should be testing all integrated games/functions, as well as every input into the system. To test the system as a whole, we need to make sure our requirements and expectations of the program are clear. While testing the system, having a clear picture of how the program will be used and knowing what issues it can face are crucial to know.

Our simple system testing plan looks something like this:

* Create a test plan
* Create a clear test script and test cases
* Perform the system test script and test cases
* Report any bugs and re-test once they are fixed
* Regression test to validate the impact of the changed code
* Repeat the testing cycle until system is ready to be deployed