**Requirements to run scene:**

To see if browser supports at least webgl v1

http://webglreport.com/?v=1

But im adding some extension libraries, so may need to support webgl v2 as well

http://webglreport.com/?v=2

Using global variable to enable/disable testing

All of the webgl testing/monitoring tools are only for webgl v1 without extensions, so can’t use them. Have to create something myself

**How I used to test**

Had game aspects, so test table was easy for those, now most of those aspects are gone. Need to unit test code instead of acceptance tests.

**Things I can’t test**

Testing rendering output is infeasible, this is essentially testing the WebGL implementation itself, rather than my application. This is the same as writing a C function, and trying to test that the compiler converts it into the correct assembly language calls.

Another infeasible option is writing my own implementation of the WebGL API, to see all the calls made to the graphics hardware are correct, having correct parameters etc.

**Things I can test**

* The data I give to WebGL/my shaders to get displayed on the screen.
* possible shader tests if have time, GLSL Unit
* buffer checks
* user input does what it should, camera, UI changing values
* terrain collision checks
* vertex creation
* Cross browser, firefox/chrome testing
* The link above, that all features of the scene work
* WebGL console errors
* Memory usage - See how rocks affect this etc
* FPS tests - see how rocks effect this etc
* Other language/library comparisons, three.js/engine, etc
* Compare on different computers

**Options for Unit Testing**

**Option 1: Make everything public**

So you access the objects properties in separate ‘Tester’ classes

Then can completely separate test/implementation code

Having everything public is very dangerous

**Option 2: Duplicate entire project and have ‘test’ and ‘release’ versions**

‘test’ version, with all public functions.

And a release function with regular public/private functions.

**Option 3: Keep everything as it is, but add a setter/getter for all private variables**

This way you can create objects and test them in other classes, but might as well make everything public, also bloats code

**Option 4: Have implementation and testing code within same class, split up into functions.**

Have private testing functions within their respective classes, code is clean but test code duplicated

**Option 5: Use a Tester class having generic testing functions, called from implementation code,**

This minimizes test code duplication, but heavily bloats the implementation code with lots of parameters to call the tester method.

Also, not everything can be put into a generic testing function, so testing code would be split between, the implementation file, and the generic testing file.

Have whole bunch of setup tests, and perhaps some in render

When testing finished, compared scene loading time with tests vs without