Final Project Part A
CS 362, 05/08/2016

Team Members:
Alisha Crawley-Davis: crawleya@oregonstate.edu
Brandon Swanson: swansonb@oregonstate.edu
Sara Sakamoto: sakamosa@oregonstate.edu

Table of Contents

testIsValid() Function Explanation

The testIsValid() function has two definitions, one accepts an array of testObjects (an array of subarrays of ResultPair objects that will be used to combinatorially construct URLs) and an options flag. There is an unparameterized testIsValid function that invokes the testing functionality with testing data defined in the testing file as “testUrlParts”.
The function begins by asserting the basic functionality of the validators isValid() function by asserting that it returns true on two “Ground Truth” urls, “http://www.google.com” and “http://www.google.com/” .
The outer loop of this test function iterates while there are still remaining possible combinations of URL parts to generate by combining the substring parts (ResultPair.item) in testObjects using the incrementTestPartsIndex() function.
On each iteration of the loop and possible combinations of ResultPairs a url string is generated by appending the substring parts together and an expected result is generated as an bitwise & operation of all of the expected validities of the ResultPairs (ResultPair.valid). In other words if any of the parts of the url are expected to be invalid then the entire URL is expected to be invalid, and the URL is only expected to be valid if and only if all of the parts are expected to be valid.
Using the generated url string the UrlValidator isValid() function is called and the result is recorded. It is then asserted that the result of this function call matches the expected with the generated URL passes as a message to the assertEquals function to indicate the url that this assertion failed on.

If verbose options are enabled (printStatus = true) then the result of each assertion is output as a single character with '.' for pass and 'x' for fail as well as the combination of ResultPairs used if enabled.

Total number of URLs being Tested

The function is testing a total of **31,920** or **31,922** urls. This number of tests is the result of the combinatorial testing performed by generating all possible combinations of the substring sets contained within the array testUrlParts. There are 8 available Scheme strings (the ninth scheme is a blank string "" and is not tested), and 19 Authority strings, 7 Port strings, 10 Path strings and 3 Query strings. The total number of urls that can be formed by combining these susbstrings is 8 * 19 * 7 * 10 * 3 = 31,920. If you include the two assert statements at the beginning of the function, the total number of urls tested would be 31,922

How isValidTest() builds URLs

The code builds the url in pieces, testing whether each piece is valid. Each piece must be valid in order for the url as a whole to be valid. The parts of the url that are tested are the scheme, the authority, the port, the path, and the query (<scheme>://<authority><path>?<query> where the port is separated out from the authority). For example, in the url http://www.google.com:80/test1?action=view, the scheme is "http://", the authority is "www.google.com", the port is ":80", the path is "/test1", and the query is "action=view".

Example of valid and invalid URLs being tested

An example of a valid url being tested by the testIsValid method is http://www.google.com:80/test1?action=view. This is valid because all parts are valid.

An example of an invalid url being tested by the testIsValid method is http://www.google.com:-1/test1?action=view. This is invalid because the port ":-1" is invalid (even though the other parts of the url are valid).

Real World Tests vs Dominion Unit Tests and Card Tests

It is not very different from the unit tests and card tests we wrote. In both cases were are given a defined set of rules with which to compare our test results. In Dominion, we created a scenario including the number of players, cards in their hand and cards in the deck. We used the game rules to compare the state of the game after a function was called or card dealt against the expected state. In the testIsValid() function we have a well defined set of rules governing valid

URLs. We generate a URL which we know to be either valid or invalid depending on the validity of the component parts. We test the return value from testIsValid to ensure it matches our expectations.

It shows that real world tests are based on the same concepts we have been learning and using in this class and that they do not need to be much more complex or complicated than the work we have already accomplished.