Test Plan

**Black-Box Testing**

For black-box testing, we created several test input files, passed them into our main program, and observed the resulting outputs. This part of the testing can be divided into the following three categories.

1. First we passed inputs with no errors (i.e. pairs of valid 5-letter words) and verified that either a word ladder would be printed out or a message notifying the user that a ladder did not exist. If something other than a word ladder or error message occurred, then our program failed this test.
2. Then we passed in inputs with errors (e.g. a missing argument, too many arguments, a 3-letter word, etc.), and we verified that an appropriate error message was printed to the console. Specifically, we were looking to see if our program handled exceptions gracefully (i.e. did not crash).
3. Finally we tried to make our program crash/fail by passing in corner cases and seeing how our program reacted. For example, we passed in the same word twice as the input argument and saw if the output was what we desired. In particular, the input word was supposed to be printed out twice indicating that a word ladder exists but the length of the ladder is zero.

**White-Box Testing**

For white-box testing, we deciding to look into to our recursive method, MakeLadder, that created the ladder solution and also our findDictWord method that created a list of candidate words in our recursive method.

To make sure our MakeLadder was creating appropriate solutions, we created two Junit test cases that would test the ladder solutions. The first one “InDictionaryTest” would verify that each word in the word ladder was in the dictionary. The second one “OneLetterTest” would verify that each word in the solution was one letter apart. For these test cases, the example of stone to money is setup, however we also tested a variety of strings.

We also looked into to the method findDictWord. This method is supposed to create a list of candidate words that are one letter apart from the inputted start word. It only creates a list of words that are different based on the given index we are looking at. We inputted an example of (stone, money, 0). This should create a list of words that are one letter away from stone but are only changed on the “0th” index, s. The way we wrote this method, the candidate words will also contain the given start word. So for stone at the 0 index, it would create a list with atone and stone. We created a Junit test case for findDictWord and tested a variety of other strings to make sure they were being created correctly.