Random Testing:

My adventurer random test has a coverage of 24.69%, which isn't impressive. I expected the coverage for this card to be higher, as I even have code to make the deck empty in an effort to increase coverage. It may be that I just didn't think of the correct ways to test the card, resulting in such a low coverage. My test revolves around running 24 tests, each with a random seed that will test the Adventurer card. I assumed this would result in a coverage of 70% or greater, but it appears I was mistaken.

My Randomtestcard1 tested my Smithy. However, my 'bug' was a game-breaking bug (the Smithy didn't discard itself). This resulted in the coverage for my test to be 18.07%, as the test could never actually get a satisfactory response. The Randomtestcard1.out reflects this, with thousands of lines repeating that it failed due to discard failure and failure to draw correctly. This second failure is a result of the card not discarding from hand, and thus the hand is not the appropriate size when my hand-check is called.

My Randomtestcard2 tested my Village and resulted in a 100% coverage. It checked every action that the Village Card is supposed to do, and verifies if the action is what is expected. If the Village card acted out of what was expected, it fails, otherwise it passes.

Code Coverage:

As stated above, my coverages were 24.69%, 18.07%, and 100% for Adventurer, Smithy and Village respectively. For Adventurer, looking through the code I can theorize the coverage is so low because I only run 24 different tests on it. If I ran more, the coverage would surely increase, as the chances for untested scenarios would increase. I use rand() to ensure randomness, so that is not the issue. It could be that I didn't force any failure states, and so the Adventurer card could still be buggy in unexpected ways, and I just didn't account for them.

The Smithy coverage, as explained above, is because my bug is game-breaking and forces the test to always result in a failure (there is no way for the random tester to reach a success state). As such, it is hard to say what to change in the tester to increase coverage. I ran more tests on this one vs my adventurer test (1000000 vs 24), so it is not the sample size. Again, I didn't force any different failure states, but the Smithy card was built to fail regardless. If the card worked correctly, I would be able to test the correct acceptance and failure cases, but testing a broken card is a bit different.

The Village coverage was 100%. It was hard trying to think of all possible ways the village card could be rejected, but it was easy for accepting it, as then it just works as intended. The test didn't run too long, about 2-2.5 minutes.

Unit vs Random:

Between these tests and the ones done in Assignment 3, the random tests preformed better on average. Of course, the Smithy card was built to fail, but the other cards had higher coverages for the random tests than for unit test. The random tests had more error detection than the unit tests, and a greater number of random tests were run. Each unit test was run only once, and each only verified one or a few acceptance scenarios. This resulted in the 21.29% coverage; not ideal but far from terrible. The random tests had more opportunities to test, and while the coverages are about the same, they have a wider

range of tests and thus a better representation of how well the card interacts with the other game components.