Testing can be both interesting and boring at the same time. I'll start with describing my experience with unit tests. Unit tests are painfully boring to write and they really don't cover much in complex functions (well at least mine didn't), but they're surprisingly not useless. I say this because for both my own code and a classmates, my super simple salvager unit test was able to find problems in both of ours. Granted, the problem was pretty major as the card didn't give the player money that it was supposed to, but it only takes one check and nearly any scenario to find that kind of problem. Still, I feel like random testing would just as easily find the problem while making me sleep better at night knowing that I didn't miss nearly every branch while testing. I would use unit tests in the real world for a quick and dirty "does this thing work at all" test and maybe to test very specific edge cases, but I wouldn't use it for extremely serious testing.

Writing good random tests was surprisingly hard. For my random Adventurer test, I was reasonably certain that the actual Adventurer code was correct, but my test kept failing. It turns out that it can be difficult to write a correct random tester just because of how many edge cases there are. Once it was written though, I felt very confident in it. It got 100% coverage after just a few runs, and would generate all kinds of weird inputs and they would all pass. I feel like this is a good way to thoroughly test a single function, but it would be fairly time consuming to write a random tester for every function. Each random test took me between half an hour and an hour to write, so if you wanted to write it for every card and every function then it would take multiple days. It's also fairly easy to output what went wrong when a random tester fails, like "Player 0 should have 6 cards in their hand, but they only have 5". This makes it easy to find the bugs you encounter with random testing.

Now onto whole game random testing and differential testing. On the surface, a whole game tester doesn't sound incredibly useful because it doesn't actually check anything. It just plays a random game and outputs which cards were played and bought. Using just that information, you couldn't tell if there are any bugs in the code. But in reality, I found quite a few bugs just by running the whole game tester by itself. I would frequently encounter seg faults and infinite loops that shouldn't be there, so I had to fix these before I could even think about differential testing. Add in some basic sanity checks like making sure no player has invalid cards in their hand at any point, and this was by far the most useful tool for finding bugs across the program. It probably won't find smaller, more nuanced bugs, but it is good for large things because you don't need to write new code for every function. My random tester had 98% coverage after 500 runs, so it was testing nearly everything. The unfortunate thing about this method was that isolating bugs took quite a bit longer than if I had a random test for a specific function. Most of the time it was just "SOMETHING WENT WRONG" and then it would take me 30 minutes to hunt down exactly what went wrong. I would have to figure out the quickest way to recreate the bug, then figure out a way to break just before the bug occurred so I could step through the code with a debugger to see exactly what was happening. Doing this multiple times was time consuming, but probably faster than writing out unit tests for every card and function. I'd say it's useful at the beginning of the testing phase when there are tons of bugs, but probably not later when the only bugs remaining are minor ones.

As for differential testing, it was kind of a failure in my opinion since both of the classmate's code that I tested had the same problems that mine originally had. Namely, my random tester couldn't play an entire game using their code without crashing or getting into an infinite loop. Obviously this shows flaws in their code, but it tells me nothing about mine unless theirs is extremely close to being 100% correct. I have no doubt that it could be useful, but it wasn't at all in this case. Differential testing should only be used at the end of the testing phase as the differences will probably be great at the beginning when there are tons of bugs.

The final thing that I have to talk about is mutation testing. I did my mutation testing using differential testing (so comparing the output of the mutant to the original), and in this case differential testing was much more effective than before. Unfortunately I wasn't able to check the mutants for equivalencies or make sure that the mutated lines were actually being executed, but even with that I got a 77% kill rate. I was hoping it would be a bit higher, but a lot of the mutants were just changing return values of functions where the return value didn't matter. I felt like my random tester was solid, but for the sake of time I only ran each mutant through 20 games. Considering that each game doesn't use all the cards, it meant that coverage was only 95% rather than 98% and some mutants that weren't killed probably would have been with more runs, as the mutations should have definitely affected some of the games. I think it would be interesting to see if a suite of random function tests would do better than differential testing for detecting mutants. I was thinking about doing this, but then I realized that it takes nearly an hour to write a good test for each card because writing correct random tests is difficult. Thus, it would have been a bit too time consuming, but my gut says that a complete suite of random tests would be better than random differential testing at detecting mutants.