Debugging dominion in CS 362 has been a fun and interesting experience. I learned a lot about what it means to write effective tests for software, how to look for and mitigate bugs, the importance of coverage, and an overall better approach to writing software. I actually learned a lot more than I thought I would from this class.

The first task was to refactor a few cards from case statements into their own functions. I felt like this part was pretty easy. The toughest thing was figuring out what all the variables did and how to deal with them if there were not in the cardeffect function directly. Eventually though I figured them out, and introduced bugs like the instruction say. Ironically, later on I forgot about one of these bugs and found it by testing in the final project.

Testing my code was very interesting. For the card a unit tests, I tried to test the minimal amount of code possible to it would be very easy to identify a bug. While the coverage of these tests were small within the whole of dominion.c, they showed how good coverage was within either the card or the function's code. This was a good example of how even if overall coverage of program is low, it can still be very helpful. In the end, I had pretty good coverage, at least around 60-70%, and I was also able to identify many bugs in the code.

Random testing was the most interesting and fun part of this class for me personally. We needed to write random tests for the adventurer card and another card of our choosing. This seemed to be the best way to test and found bugs. If anything running more tests in general is better because you get to test many different inputs and you get coverage of hopefully the same code many times. This way it's also easy to see which lines were ran with certain inputs and which tests failed with certain inputs, and compare those two. Writing the random tests were relatively simple.

Like the first assignment, the most difficult part was figuring out what all the variables did and how to create conditions to play the adventure or other card and get results. It took a lot of combing through code, looking at other students implementations and messing around with gameState to figure out exactly how to set up the random test. But I eventually figured it out and from there, I generated random inputs to actually run the tests. The other part was to look at what each card actually does so that I could test for correct output. Depending on the card this could be difficult, but adventurer and the card I selected were decently easy to test.

The fourth assignment was far more difficult. I had to take the random testing and instead of applying it to one card or function, I now had to make the tests play a full game of dominion as well. This proved to be very difficult. While I had the random generation of inputs now figured out from the third assignment, figuring out the game logic for a full game of dominion took a long time. I had to research exactly what happens in a turn for dominion, even though I've played the game a few times before. Once I figured out how the turn goes, I had to simulate it.

First, I had a random seed, and initialized all the variables that I needed to, such as number of players, and randomly generated cards to be in the pool of cards to buy. Then I made a loop that ran until a flag was set to signify the game was over. Basic game logic. Then I randomly chose a card for the player to play for their play phase, then randomly decided on a card to buy for their buy phase, ran endTurn(), and rand a check to see if the game was over. Iran into some problems with the buy phase; sometimes the player wouldn't have enough coins to buy the card chosen by the random number, but they needed to buy cards so the game would progress. To fix this, I created a loop that would keep generating a different random card to buy until they could buy a card or they tried every card. Finally, to actually see the effect of each turn, I printed out the useful member variables of the gameState before and after the turn to see if the cards did what was expected.

For this final project, to find the bugs for part 1, I mostly looked around in different classmate's code until I saw a card that had a wrong return. Then I would write a test to see if first, my own code passed the test, and if their code failed the test. This was pretty simple and I enjoyed bug hunting. Next, for my own bug, I decided to use my test of a game of dominion I used in the fourth assignment. I got the output from it and looked through the various cards and how the gameState changed after playing the cards. I compared that with the card effects I found on the games wiki. I found a bug where the village card was adding more actions then it should. I looked in the code and then I remembered I had introduced that bug in the first assignment when refactoring the card's code. I thought it was ironic that I found my own bug out of all that could be in dominion.

Finally, for the tarantula code, I decided to write a small python script. I wrote a test for the minion card that prints a 1 if failed, and a 0 if passing. The script then takes all the fails, looks at gcov and sees what lines they run. Once it runs through all the tests, it prints out all the line numbers from the failed tests, and then I ran a uniq -c | sort on them to see which lines were the most suspicious. I had a few small bugs in the python, but it showed that the minion card failed virtually every time it was ran, which is true but in the end it didn't really help me narrow down the buggy code.

Overall, this class taught me a lot about debugging, testing and thinking like a tester. I really enjoyed it and I'm sure to use these concepts in the future