Cooper Gale

CS362

Final Project

Dominion Testing Experience

Up to this point in my collegiate career I had never taken any class that had more than a day or two that was dedicated to testing buggy code. I feel like I might have been better off taking this class earlier as it opened up a myriad of options for debugging effectively and making verified finished code projects. It was actually a refreshing break from the usual business we take care of in CS classes. Hunting through code to find errors is an entertaining puzzle.

The first section of class was becoming familiar with the game of dominion, which was one of the best tasks I've been assigned in college. Additionally we had to take five of the cards used in game within the code and refactor them into separate functions called by the main. Rather than just recreate them exactly we also added bugs to the cards for catching during testing down the line. The only crazy challenge we ended up with here was the seemingly random core-dumping problem, which never totally went away throughout the duration of our testing.

The second section was creating our first bug testing scripts. We were required to create unit tests, which look specifically into one portion of your code and cover than section extremely diligently. We did the same again with different cards in the game as well, also making four tests. This was a little difficult at first (and I now know not to use C assert to get meaningful output) but eventually I was able to create decent tests that don't crash everytime they see a little error, additionally I can see tests that will actually do good work while verifying code. The second portion of this section was learning how to use the gcov tool which is an extremely interesting tool for this class. The ability to look at a piece of code and know that you have actually exhaustively ran through a significant portion of it can give you much more confidence that your system is outputting as it should be. Running gcov with our little tests was a little bit disconcerting because we were able to very clearly see how little of our code was actually being executed even with all eight tests.

The third class assignment focused on random testers which are important in that they go out and run thousands of tests using random values to check many more times at a speed much

quicker than a human. This was probably my favorite portion of dominion testing because it was much different than anything I had done previously for bug correction. This one, being so different, took a little time to wrap my head around, and coming up with all the random variables that could be in play before the card was executed took a good deal of thought. These tests are only as good as the creators, and if we forgot one game state that could be random we were losing coverage and potentially not discovering bugs that might be hidden in our code. I looked through the gamestate struct and essentially took every variable out of that and assigned them random numbers to randomly check any of the conditions that might accrue during game time.

The fourth assignment was a random tester for the entire dominion.c code which was actually relatively easy to make compared to the random card tests we did before. Essentially we only had random number of players and a random game seed generated at the beginning. We also had to implement random set of kingdom cards to be used in game, this was to guarantee the best possible coverage of all the code we created. Interestingly enough I actually got less coverage on my dominion.c code than I did on my card random tests. I believe this is because the main code only ever plays smithy so much of the card data is never reached. To improve on this testing we would need to create a player algorithm that is a more intelligent ai with its purchase choices and strategy, however that was outside the scope of this class.

Classmate Testing Experience

During our final testing for this project I checked the code of my classmates Dillon Odle and Nolan Thompson for a code review.

Nolan's Code :

The code was well commented although I did have some confusing with his function names, such as card_test for his refactored code. With some small exceptions in naming conventions I believe it was easy code to follow. During code testing I found his code to perform mostly correctly, only catching one error about actions in his village card. I had no tests that actually ran coverage over my great_hall section of code so I notably never caught his second error until a line read in our final inspection for part three of this assignment.

Dillon's Code:

This was exceptionally readable code for me, everytime I was curious about a code implementation the explanation was a comment at hand. I had no qualms about the readability of this code. With Dillon's code I was actually able to catch all his errors because they ended up in his smithy and village cards for which I had card tests for. After correcting this errors (extra card given for smithy and extra action given for village) I was able to run his code against mine in random testing and get no difference in a diff output compare. I was actually quite happy with the fact that my testing programs were able to pinpoint both of the errors that he had induced earlier in the term.

Conclusion

I really enjoyed learning all the various testing techniques used to adequately test code. It was also quite satisfying to catch my classmates errors with the tests I had written without knowledge of their implemented bugs. This class changed my perspective on verification of code, because typically that is not something we do at OSU. I am happy that I elected to take this class as I believe this base level will give me a leg up in my future testing requirements.