CS 362 Test Report: Eric Stevens

In this class we were testing the implementation of a software version of the Dominion board game. To do this we utilized several different tool. Unit testing was used to test specific aspects and situations of individual functions. Random testing was used to try to get at edge cases and to possibly find hidden bugs in places where they were not expected to be. We attempted to test individual functions to make sure small scale implementations were working correctly as well as system wide tests in order to check for seamless system integration. Gcov was utilized to try to get an idea of exactly how much of either a single function or of the entire program was being tested.

Finally I implemented a version of tarantula. This was very inspiring for me. To see how to manipulate test program outputs and coverage collections and organize them in a way that could give one further insight into the philosophical durability of their testing scheme was inspirational to me. I felt that it was unfortunate that the dominon.c code was given to us in such bad shape that it would have taken far too much work for me to be able to piece it into a working version. I feel that tarantula is an incredibly valuable tool for testing large pieces of code with isolated bugs. But when there are so many different bugs in so many different places in a piece of code that is very integrated, tarantula seems to throw a lot of false positives. I hope to try the tarantula method again, maybe on a project that I work on independently.

ZHENGZH(onid)

Onid: zhengzh tested the Dominion implementation with unit testing, random testing, and coverage. He used unit testing to cover the smithy card. He used unit testing to cover the great hall card. He used unit testing to cover the outpost card. He also used unit testing to cover the embargo card. He also used the unit testing method to cover a series of functions that were not cards. He used unit testing to cover the supply count function. He used unit testing to cover the is game over function. He used unit testing to cover the kingdom cards function. And he also used unit testing to cover the end turn function. As well as unit testing, zhengzh used random testing to test the implementation of the code. He used random testing to cover the adventurer card. He used random testing to cover the smithy card. And he used random testing to cover the great hall card.

Through the testing, zhengzh was able to achieve fairly high coverage. When putting all of the test coverage together he was able to achieve the following coverages by function:

43 % of dominion.c was covered.

100% of compare() was covered.

95% of initializeGame() was covered.

100% of shuffle was covered.

92% of playcard()was covered.

46% of buycard()was covered.

100% of handcard()was covered.

100% of supplyCount was covered.

100% of whosTurn()was covered.

100% of endTurn() was covered.

100% of isGameOver was covered.

100% of drawCard() was covered.

10% of getCost() was covered.

100% of playsmithy() was covered.

100% of playgreathall() was covered.

18% of cardEffect() was covered.

93% of dicardCard() was covered.

82% of updateCoins() was covered.

and 77% of gainCard() was covered.

For the functions in the above the 90% mark I think it is fair to say that those are very well covered. It is difficult to know the value of those tests. The coverage is only useful if the test that come with that coverage are catching the bugs when they occur. In this instance most of the tests had 100% passing rate. He was able to catch several bugs. This leads me to believe that for the functions that had tests written and had high coverage percentages, it is safe to assume that those functions and cards will work as intended.

PEDERSON(onid)

Onid: pederson tested the Dominion implementation with unit testing, random testing, and coverage. He used unit testing to cover the steward card. He used unit testing to cover the cut purse card. He used unit testing to cover the outpost card. He also used unit testing to cover the embargo card. He also used the unit testing method to cover a series of functions that were not cards. He used unit testing to cover the score for function. He used unit testing to cover the is whos turn function. He used unit testing to cover the kingdom cards function. And he also used unit testing to cover the get winner function. As well as unit testing, pederson used random testing to test the implementation of the code. He used random testing to cover the adventurer card. He used random testing to cover the steward card. And he used random testing to cover the great hall card.

It is difficult to tell how much coverage pederson was able to get. His coverage results are not tagged by function and therefore, each time the coverage is run, we only get a small glimpse into what the value of that particular instance for the dominion.c file.

For instance:

When running random test 1 the dominion.c coverage was 17%.

When running random test 2 the dominion.c coverage was 17%.

When running random test adventurer the dominion.c coverage was 12%.

He does not appear to be analyzing the coverage for any of his unit tests. There does not seem to be a way for him to know whether he is full covering the functions that he is testing. Much of the small percentages he is getting is likely to be peripheral coverage of other helper functions.

Due to this, I would not be satisfied with an assertion that this code is reliable. He does catch failures in dominion.c but who knows how many bugs are being missed.

Conclusion

I would not trust any of the code that any of the members of the class have been testing. This code was given to us in a mutilated state and with the amount of time we had it would have been very difficult to even come up with a solid comprehensive sweet, let alone to work out all of the bugs. That being said, I learned about a set of valuable tools to implement to check my codes proper functionality along with a way of thinking that I think will make me a better software writer.