Nathan Tollbom

CS 362

Alex Groce

My Experience Testing Dominion

Overall, I had a rather interesting experience testing Dominion in this class. It's a game in and of itself that I play and enjoy on my own, and it was pretty cool to be working on a programmed version of it. That said, over the course of the class, I discovered quite rapidly that the implementation of the game in this particular case was less than stellar. For example, I found a bug in the base code that leads to an infinite loop if a certain kingdom card is played with the right contents of the hand, deck, and discard piles.

As for testing the code of my classmates, I tested the code of the following usernames: fengzi, harderg, and pederson. These were chosen at random from the list just by spinning my scrollwheel. Overall, their results were very similar to my own. 60-70% or so code coverage using the random game generator I was running over 20 or so games. I ran the tests multiple times with different seeds, which produced the different code coverage values. I can conclude that their code was no more reliable or working than my own, which is likely due to the fact that we all started with the same code base and made negligible changes to it. All the users I tested focused more on writing tests for the code and completing the assignments for the class rather than working on creating a working implementation of Dominion, not unlike myself. I would consider their code, as well as my own, as no more reliable than the code we were given at the start of the class, as it did not really change. Looking back, it would be interesting to compare our current implementations to the base implementation we started with and see if there was any significant difference.

The most interesting part of this class for me was the random testing we did in Assignment 3. Namely, the testing of the adventurer card. I had only really looked in depth at the functions of the cards that I had reworked, so the failures in my adventurer random testing came as a bit of a surprise at first (though the fact that it failed didn't surprise me). This in turn inspired me to take a hard look at the code in question. I found what appeared to be a very obvious infinite loop bug, which I proceeded to check with a modification to a unit test file. As I suspected, the program ran but never stopped, from an infinite loop. This was due to the fact that it would continue to look for treasure, even when no treasure was present to find (the loop runs while drawntreasure < 2). So with contents similar to this:

Hand

```
[0]: adventurer      [1]: copper    [2]: copper    [3]: copper    [4]: copper
```

Deck

[0]: estate   [1]: village

Discard

[0]: estate   [1]: village   [2]: duchy   [3]: seahag   [4]: smithy

Playing the adventurer results in all the cards being drawn into the temp hand and the loop continuing, with nothing happening, as cards can no longer be drawn. Finding this bug in my own code led me to check for it in the other users' code that I checked, and since none of them really changed it (one had refactored it out), I was able to give them feedback about it. This is a pretty sneaky bug because it's not one that would normally be seen in game of dominion, since there's usually more than enough treasure in a deck that is actually constructed by a player for the card to always pull out a couple treasures. The only situations I could think of where this bug might be critical in actual play would be in a game where a player had a deck minimization strategy, or a very aggressive attack card game. Deck minimization is a strategy where the player uses cards like the Chapel (trash up to four cards from your hand) to trash out all of their low value cards (low value being cards that are basic that are not powerful enough to get the endgame, game winning cards - provinces, gold, etc). The gist of the strategy involves trashing all of your starting cards and having a small deck that is very consistent, since you'll be getting all of the same cards more or less every turn. You spend your initial turns buying silver and the action cards to trash your deck, and cards like Labratory (+1 action, +2 cards) to be able to cycle through your deck. A deck like this might be down to 6 or 7 cards at some point, having 2-3 action cards, maybe a victory card, and a few valuable treasures. Playing an adventurer in such a deck under the right conditions would, in this implementation of the game, cause the game to crash via the infinite loop. In the very aggressive case, the presence of strong attack cards in the game that trash treasures from players' decks (like Thief or Pirate Ship, which is not in this implemenation) could lead to a situation where a player with an adventurer does not have enough treasure in his or her deck to retrieve with the adventurer. This would then lead to the infinite loop crash I described earlier.

I also confirmed the presence of this bug through my tarantula testing. The unit test I had written earlier was modified to have random data (but still not enough treasure to make the loop complete). The data then confirmed that the loop was very suspicious.

Overall, it was rather interesting to be writing tests for an existing piece of code rather than working on the code itself (though this was an option). This was completely different than every other CS class I've taken so far, though we were required to have unit tests for our development in CS 361. As a Mechanical Engineering major with a CS minor, I doubt I'll ever really be a code tester, but the experience was pretty enlightening nonetheless.