Andrew Tolvstad
2016-06-07

Testing Dominion

Testing dominion has been a very interesting experience. I had no idea just how horrible a single program could be. I must give my congratulations to whoever wrote the original code snippets; they are the worst I have ever seen in my entire CS career.

Testing was subsequently difficult: because the code was so fragmented and (yet somehow also) very interdependent, it was difficult to create meaningful tests for many things, and I ended up changing the structure of several parts of the code.

Because the code was largely in one gigantic file (dominion.c), code coverage percentages largely became meaningless at the file level. I ended up splitting my code into other files, such as one file to contain all the cards. I also tried my best to keep everything organized, but since everything was kind of in one big folder and the assignments had mandatory locations for some files, it was very difficult to maintain organization while keeping the individual pieces manageable.

While writing my unit tests and random tests, I chose not to use the game creation and initialization functions in dominion.c, as I was afraid that any problems in their code could propagate through to my testing and obfuscate bugs, or create the illusion of a bug that does not exist in the code I was testing.

Repository 1: Phillels

For my first testing repository, I chose phillels. I started my testing with the random Adventurer test that phillels created in a previous assignment. The random adventurer test covered 100% of the 16 statements in adventurer.c on the seeds I used. The test was intermittently failing with certain seeds, leading me to the conclusion that a bug existed either in the implementation of Adventurer or the test, causing the count of the previously empty hand to be incongruent with the number of treasure cards drawn into it during the turn.

The random tester for embargo also showed failing tests: gcov showed the test covers 86% to 100% of the embargo case in dominion.c. Some digging showed that the reason the test was failing was not due to there being a bug in the embargo card code, but instead that the functionality of coins had been left unfixed from the original codebase, where it was terribly broken and inconsistent across different functions.

Since the gameState can only hold one coin count at a time, the function playCard must handle the tracking of coins completely within a turn. Unfortunately, it appears that the card functions and the code for handling coins were written by different people and were not

designed to work together, so the values are not interpreted the same way through the function calls. This portion of the code simply needs to be rewritten from the ground up.

The final bug I found was in sea_hag. The existing test covered a varying amount of the card's code in dominion.c, but after 100 runs with different seeds it approached 86% of the 7 lines. It was very easy to uncover the problem with this card: just looking at it revealed that the code was not only accidentally double-decrementing something instead of using n-1, but was also using post-decrement instead of pre-decrement, causing a complete failure of the intended behavior.

I do not feel that this first codebase is reliable at all, since there are no changes to the core functionality of the Dominion code and the original codebase was used as a canonical example of unreliable code. The code coverage of dominion.c from the unit tests and random tests is only 2.23%, but this was to be expected, as the tests wisely do not use any of the dominion.h functions to test the functionality of the cards. If they had used it, that would introduce both obfuscatory complexity and confounding variables into the tests.

Repository 2: Leightle

My second repository is Leightle's. Out of the unit, card, and random tests, two failed: Village and buyCard. Village's failure had to do with the number of actions left, and buyCard was complaining about phase restriction.

Starting with the Village card: the test for Village intermittently registered a difference between the expected final value of numActions and the actual value at the end of the turn. Being very simple, the source code for the card's action did not seem to be at fault, and the tester does not perform any actions between measurements other than playing the card itself, so there is probably a bug in playCard. This test covers 100% of the 3 lines in village.

The buyCard function should logically be limited to the times in the game when it is applicable to buy cards; that is, during the 'buy' phase. The unit test provided shows that the buyCard function does not return an error code on the other two phases. The test does not appear to fail on any of the other aspects,

Upon looking at the source (which comes from the original codebase), there does not appear to have been any attempt to handle entrance while a different phase is active. In fact, the card does one worse: it forcefully changes the phase of the game to be the 'buy' phase if the purchase is possible, even if such a change does not make sense at the current time.

With all of the unit, random, and card tests, Leightle's testing covers 36.2% of dominion.c. This is significantly higher than Phillels' because Leightle's testing used the game initialization functions in dominion.c to create a full instance of a game, rather than initializing the memory separately and manually.

Leightle's code appears to have more passing tests than Phillels', but that may just be due to preexisting problems being hit or avoided by the unit/card/random tests. Leightle has also reindented dominion.c, which in my experience has made it easier to see stupid mistakes like misinterpreting what block operations are being done in. Finally, Leightle's code had very few compiler warnings, which means that it's at least syntactically sound, and there are fewer possibilities for mistakes to be obfuscated by unexpected behavior in the executable.