Joseph Cronise

CS 362

Final project part 4

Throughout the course I have had the chance to develop my own test functions for multiple different aspects of the dominion code. At first I thought it would be similar to the J-unit tests we wrote in CS 361, and I was right, to an extent. The difference being that in that class we used the tests as the basis for creating brand new code, where as in this class we were developing tests for code that was already written. This lead to a significant change in how I approached the problem. I found that reading over every function and header file in order to get an idea of the overall picture was very helpful.

While I had trouble writing some of the code, especially in assignment 4, I felt this experience was very useful and has helped me to become a much better coder. I learned many lessons, not the least of which was “always check the header files” while it might sound silly this would have helped me greatly early on. Up until assignment 4 I never really paid attention to the interface file and didn’t realize how useful the functions contained within it could be as the basis for testing.

Classemate: phelpsmi

Total Coverage: Hard to tell precise coverage % due to using multiple test files. However it should be in the 50-60% range.

Note: in order to run his code I had to fix a compiler error in his dominion.c (though I believe it would have compiled if I had used flip instead of visual studios)

I started by running my cardtest1-4 and unittest1-4 files on his code. All came back with passing results. The cardtest files tested steward, smithy, embargo, and council. The unit test files tested to make sure that card costs were accurate, that the isgameover function worked correctly, that the whoseturn function worked correctly, and that the numhandcards function worked correctly. Finding no failing cases with these tests I moved on to my randomtestadventurer and randomtestcard files. These thoroughly test adventurer and steward. However both came back with all tests passing. That said I believe that if I had run this prior to fixing the error that kept it from compiling by using a less strict

complier such as on flip, I would have seen errors in the adventurer card. This is because he was using an uninitialized variable multiple times in the code. Visual studios won't allow this to compile, but, if memory serves, flip will. This means that the value of his temp variable would have been unknown, leading to multiple incorrect values and potentially game breaking bugs. I ended by running my testdominion.c file on his code. However, due to this code only testing the game by forcing the player to buy victory cards only, no bug were found.

Apart from the one issue mentioned above I found no major errors or bugs in his code and find it to be reliable based on what was tested. However due to the limited nature of my tests and the fact that I was only able to check that 11 cards worked correctly I cannot make any claims as to the reliability of the code as a whole. As stated above, prior to my fix I believe that playing the adventurer card using this implementation would lead to errors in the game such as incorrect discard amounts. These errors would be potentially game breaking.

Classemate: nguyehai

Total Coverage: Hard to tell precise coverage % due to using multiple test files. However it should be in the 50-60% range.

My approach to testing this code was the same as above. I first ran my card/unit test files, followed by my randomtestcard/adventurer files, before finally running my testdominin.c file. Unlike in my previous case, I found significantly more bugs in this case. Except I found that they would only appear in specific circumstances. What I found is that when I tested the cards individually they worked fine, however when certain cards were played as part of the overall game they did not perform correctly. Luckily, I had run into this kind of problem before in prior classes and new that, since the individual cards worked perfectly, and everyone was using the same basic code, the only place that this issue could arise is in the cardeffect switch statement. And when I looked there I noticed that every

refactored card that was now a function call was missing a break statement. This meant that whenever one of these cards was called the statements below it would also get called till it ran into a break or return statement.

With this bug in mind I cannot call this code reliable. There are multiple cards that, when played, will have the effect of playing other cards. This entirely breaks that game and makes in effectively unplayable. (though with a bit of editing this could be an interesting game mode).

Overall I feel my test code is inadequate to fully test the dominion game as I was never able to get my random card testing code in testdominion.c to work as I wanted and instead I had to settle for a simplified version. However I feel that the skills I learned in this class will be useful in future classes and I wish that some of the information from this course was touched upon much earlier (CS 162 maybe).