Richard Wendt
CS 362
6/7/2015

Test Report

Experience

Testing Dominion was definitely an interesting experience. One of the hardest parts about testing this game initially learning about what exactly my output was going to look like. I had to read about each card, and I needed to learn exactly how these cards worked. Once I got the hang of how the game worked, the next step was to write my unit tests.

After taking this class and learning more about testing, I can appreciate the work that goes into making a program run correctly. Being able to understand coverage was initially difficult for me, and I really struggled with testing in general for most of the class, and I am still not very confident in my ability to test. This is a skill that I believe should be brought up much earlier in the Oregon State curriculum, because it could definitely help with many of the programming classes that we are taking.

During the writing of my unit tests, I found out that many of the cards in dominion had bugs because they weren't outputting how they should have been. For instance, the remodel card was using a greater than symbol instead of a less than symbol in the initial for loop, and I had to go back and read about what that card did in order to figure out where exactly the problem was coming from. Learning the game was essential to each one of my tests because I needed to know what to look for if I had any problems. This was also essential when testing other people's code, because I needed to know why their code was buggy in certain places while my code seemed to be running fine. However, I was incredibly proud of my unit tests, because they allowed me to see the dominion code differently.

Writing a random test was initially difficult. I realized at this point I had no clue how the dominion code worked, and so my research began. At first I was just writing code with the intention of simulating some type of randomness, but as I learned on the dominion code worked, my code started to come together. One of the most interesting parts about this phase was seeing my testers in action. Initially, my random tests were seg faulting for seemingly no reason. I checked through the debugger, and everything seemed to be in order. What I found out was that testing more than 11 times was causing my program to seg fault, which means that my program was using too much memory if 12 or more tests were occurring. I had to break up the tests into three separate game states in order to get triple the amount of tests.

Overall, I thought this class was amazing. Learning how to randomly test a program will definitely help me with my future in programming, and I am really glad I learned how to effectively test a program. One of the most useful parts of the course was learning how to make useful unit tests, and I will definitely start creating unit tests for my future programs.

Richard Wendt
CS 362
6/7/2015

Coverage

For my randomtestadventurer, I only got 30.02% coverage, and my randomtestcard received 24.19%, but for each test, every line for adventurer and cutpurse was hit. Some lines were hit very minimally, while others were much more extensively, but I believe that I did a decent job at covering these two cards. As for my testdominion program, I received 73.93% coverage. In order to receive this type of coverage, I ran four different seeds of the program, each testing 20 different times that would change seeds based on the time. The coverage will change based off of this, but I am documenting this coverage based on these runs only. Three cards were not hit during these tests, and that is the baron card, the remodel card, and the curse card. In other runs, I would expect the baron card and the remodel card to be hit, but I did not write any tests for the curse card, so that card isn't being tested.

I also did not hit the fullDeckCount() function, the newGame function, or the kingdomCards function. If I were to rewrite this tester, I would write my tests to hopefully hit these functions further, and to see what their output would be and how it would have an effect on testing my program.

In summation, I believe I tested dominion pretty thoroughly with my random testers. Although I didn't receive complete coverage, I did hit the functions that I believe were essential to running the game, and using these testers, I was able to find bugs in my program, and was also able to see how the game made.

Reliability

Nguyehai

When testing this student's code, I received 70.19% coverage using every test I have. However, because of the way my testdominion code works, I was not able to always have the program compile completely. This is due to bugs in the students program; something is making the program crash. This is most likely because the student did not put breaks in his switch statement for cardEffect, and so, if I had an adventurer card, I would also use a council room card, and a feast card. If I didn't have those cards in my hand, then the program would crash. In addition, the remodel card was not working right, as my output for my unit test was not correct. After fixing these issues, my program would compile every time, and I was able to see improved results for all my tests. Overall, I would say this code is somewhat reliable, but could definitely be improved. If the student were to fix these issues in their code, and did some more extensive testing, this code could be used.

Richard Wendt
CS 362
6/7/2015
Rodrijos

When testing this student's code, I received between 70 and 80% coverage using every test I have. This person also had a problem with their remodel card and, like my other tests, as unable to hit the baron card. This is most likely a problem I have with my random testers. However, unlike the first student, this student's program seemed very reliable. I was having no problem hitting most of the cards in their program, and I was unable to detect any errors that would make my program crash. Overall, I think this student has a more reliable version of dominion; however, this student needs to fix some problems in their code first before considering releasing their code.