Luay Alshawi
CS362
Spring 2016

Throughout the quarter, we have been debugging dominion's implantation in C starting from the basic unit tests and ending with more appropriate industrial approach for debugging. I believe I learned the fundamentals of debugging in this class, by applying what I learn into a written code that can be executed to find bugs and error in the dominion's game. We started learning how to use a manual testing and we learned that the cost of this approach is expensive compare to the automated testing. Also, one thing I learned is the stages of testing which are unit testing, integration testing and system testing. Started from the second assignment we were able to write unit tests to find bugs in dominion. Eventually, this report will consist of different sections to discuss the testing process during this class.

By working on the unit tests I started struggling on how to find bugs and do unit testing in order to find bugs. After I finished my first unit test I was able to understand more of the game's api so I can do more of unit testing and test the logic for the game. For assert, I used the same function that was provided on the github repo's example and I think it helped me to focus more on unit testing and avoid off topic work such as worrying about assert. Overall, testing the api function helped me to fear less of the long code, because of this I am comfortable changing the code behavior, also, the same goes for cards unit test I think cardeffect's function should be shortened and refactored for easier debugging. Also, by writing my unit tests for the function fullDeckCount I realized the function may not be the cause for a bug but actually helped me to show that the cards I supply to initialize the game with, turned out, some of them goes missing even though I do supply the card's name to the function but return a count of 0. The bug could be caused by initializeGame or fullDeckCount does not do correct count for a given card.

Moreover, during the class the gcov tool got introduced to us as a tool to collect coverage on certain programs. The tool is helpful and helped me to understand whether the code that I wrote for my unit tests are covering the functions that I am testing or not. However, the coverage of gcov for my unit tests start from 16% up to 26%. The lowest gcov percentage is the unit tests for initializeGame and that because there are not many cases to check compare to the other functions. However, the best coverage is testing the function council_roomCard which was refactored from the assignment 1.

On the other hand, finding a differential of the dominion by running my classmates code and compare it with my code using a python script results to an interesting difference. The first difference is the coverage; my classmate's dominion is always 1 percent higher. However, comparing the scores for each player during the game my dominion is less error in terms of score. I think this not a bad idea to test and see other dominions with small changes and able to see the overall changes. The generated difference file is not easy to follow since there are so many outputs. Overall, the differences are significantly huge and because of that it's not easy to say which one has a correct implementation of dominion.

Comparing my code of dominion with my classmate jiangzh I see that there are two bugs in his domenion.c. the first bug is in the council room function. The function supposed to draw 4 cards when the council room card played in the game. Also, by running my implementation of tarantula the script was able to detect the lines that causes the unit tests to fail. In fact, by running tarantula.py I was able to see the suspicious lines which actually point to jiangzh's council_room implementation. Moreover, there is another bug in village implementation that causes the number of actions to increase by three while the number of actions should be increased by two when the card used during the game. The dominion

Luay Alshawi
CS362
Spring 2016

code of jiangzh can be reliable if the bugs mentioned earlier can be fixed. On the other hand, the dominion code of harderg consist only of one bug based on my testing of his dominion. The only bug that I found in harderg's code is located in the embargo's method. the bug in this function is that each time the card gets played it would add one coin to the player's coins while it should add two coins. Overall, both of my classmates have a reliable implementation of demonian after they commit a fix to their bugs.

Last but not least, for part three of the final project I decided to implement the tarantula formula using python. Before implementing the formula, I had to modify the testdom.c and rename it to tarantulatestdom.c in order to apply the tarantula on the program. The reason I decided to use testdom as the base for the testing phase for tarantula is because testdom plays the game randomly and it would be a great approach to apply the tarantula formula on it to locate the most failing percentage lines on dominion.c. The tarantula formula proved to me that it is helpful in my case since I am using it on a game that is running randomly that buys and plays card. Also, the formula helped me to locate the lines where my testdom is failing, even when running testdom there won't be any odd behavior on the output. However, behind the scenes testdom actually try to play a card while the conditions for this such card are missing therefore the testdom will continue running but at some point at least one player would not play a card since there is a failure when playing or buying a card.

Overall, I think dominion is a good choice for this class to apply what we learn to the dominion's implementation. The rule of the game is a straight forward and can be easily implemented if it's designed very well at the beginning. Otherwise, the game will consume a lot of time in order to debug the game. Obviously, having a program that is written by someone else and try to test and refactor the code turn out to be a fun experience and it would give us a similar experience based on the industries out in the real world.