Bryan Liauw

CS362

Test Report: Dominion

Dominion, for me, is an unfamiliar game. I had trouble understanding the concept of the game because it is very unlike most board games or card games that I know. In a way, it feels like playing a trading card game with every card available to you. Most people consider Dominion to be its own genre of game, calling it a “deck-building” game, which I feel is a very fitting genre. While it presents a pretty tough challenge for me personally, I learn to deal with it as the testing goes on. Although it is required on the first assignment to understand the game, I honestly can only tell you the backbone of the game. I can understand the goals, concept and the phases of the game. But cards’ effect, cards’ interaction with the game and the more subtle things are not within my realm of understanding. Furthermore, there were some misconceptions that I made; One of it is that at first, I thought the player’s turn alternate in every phase instead of a player doing all the phases before giving up his/her turn. As the term goes, I also researched more about Dominion, especially how the cards interact. I even read strategy guides to understand the subtle complexities this game provides.

The code has several problems. The main problem, I feel, is that the code is not well modularized. The code is too congested in one particular function that it is hard to understand. I feel that the card effect function should be spread out into several functions and it would make it easier to understand. I also feel that instead of an enum, the cards could be implemented as object with their own respective functions. This will make the code easier to understand and easier to debug. Furthermore it will also solve the aforementioned problem. There are also several functions that I feel are redundant and can be removed if the cards are implemented as objects, such as get cost.

My dominion code does have several bugs that can be considered game breaking.

One of the aforementioned bugs appeared during the mandatory random testing of adventurer. My adventurer card code may discard more card than necessary. This number can sometimes results in negative number of hands in some cases and therefore a very large amount of cards in discard pile that may exceeds the card in deck. The numbers of cards discarded usually vary under different circumstances. Sometimes if the cards are close to each other it will discard the cards almost equal to the expected discard. However, drawing the card usually does not present any problems. After fixing it, it now discards the card as it is drawn if it is not a treasure card because I feel making a temporary hand is both unnecessary and might create a bug from there.

Another bug that I found is that playing 2 consecutive tributes can result in unpredictable behavior. I do not know what causes it but I think it happens because the player do not have enough cards to show and ends up breaking it.

Another bug I faced is that sometimes the scoring is very weird. Instead of a positive number, it sometimes goes all the way to negative numbers. This should never happen in a real game because there is no way to score below 0. Furthermore, whenever this bug appears, the game seems to end much faster than it should. When I'm comparing it with others with the same random seed, I tend to finish the game earlier if negative scoring happens. I think it happens because Baron often causes negative number of estate supply. I find that Baron gain the estate card and then decrement the supply number again. It often causes negative number of estate left and will promptly end the game earlier and produce weird scoring.

A bug that might happen to is that some cards are not discarded at times after usage and only discarded later on. I'm not sure if this is intended but I feel that changing it might be important in case there would be implementations of different action cards that allow actions that may reuse cards in discard pile. Also, this allows some weird mechanic that allows the player to discard the card that is currently used as an action for one of the cost of the action. This may results in unpredictable behavior although I have not yet encountered it.

The coverage of the game is pretty high. In the random game testing, using some seed, there could be a good coverage while some seed with cards that are not as complex and less players usually have lower coverage. Usually it could range from about 65% to 80%. The coverage with the unit testing are rather low- around 20 to 25%, but it is to be expected considering that the mechanic or card to test does represent only a small portion of the game. Through this coverage, we can see that the tests are almost comprehensive, meaning it most likely has every area covered in finding potential bugs that user might face. While there are more subtle bugs that can only be reproduced if certain cards are played in conjunction of another, I feel that such bug is going to be irrelevant and that the code can be considered reliable.

I also tested two of my classmates' Dominion implementation: Evan Schaler and Adam Puckette.

In Evan's code, I found similar problem to mine. The negative scores, however, are more often found. In most seed of my random testing of playing the whole game, I find that his code will often finish faster and with negative scores. His code also breaks in similar way as in mine. I find out that most of his code is pretty untouched and I feel that it is pretty similar to mine. There are some tests that did not find any bugs using his code as compared to mine. In a way, I feel that Evan's code serve as a good template for me to base my work on because it feels like I can compare my code and his easily. Adam's code is also pretty similar to Evan's. In a way I does the exact same thing to Adam's that I did in Evan's.