Kevin Guan
6/7/2015
CS362 Test Report

The overall Dominion project has been an interesting and unique experience. Beforehand, my debugging techniques was limited to only printf and assert and have never applied them to a large degree like Dominion. The initial stages of debugging Dominion was difficult due to the unfamiliarity with its many foreign functions and their behaviors. The same case also applied when I worked with other people's code. When I debug any foreign source code, I need time to obtain a rough understanding of its workings. Once I obtain familiarity with the major functions and their behaviors, the debugging process becomes easier. In the process, the testing of Dominion in the class has introduced numerous new concepts and bolstered my arsenal of debugging techniques.

One of these methods was the unit test. My first unit test was relatively simple and tested the newGame function. The function is responsible for allocating memory for the game to run on. If the game state returned null, then a warning message will notify the user that a bug is present. I also created unit tests for the functions updateCoins, buyCard, and isGameover. UpdateCoins' unit test checks the content of the player hands to ensure that they return the proper value. The buyCard unit test is used to ensure that the buy mechanic of Dominion is implemented, and that players do not exceed their purchase limits. The isGameover unit test is implemented to ensure that the win conditions of the game are functioning properly and are ending the games when appropriate. Three out of the four unit tests achieved close to 100% coverage of the affiliated functions. The buyCard unit test only achieved about 40% coverage, but coverage for this unit test can be improved by implementing more case scenarios into the unit test via if statements. Furthermore, I eventually referred back to these unit tests when I was random testing a full Dominion game. In one case, the first unit test was used to pinpoint a bug where several instances of game states failed to free after the win condition is met and bug the newGame function. In addition, four more unit tests were written for the individual card effects. My four unit tests that were based on Gardens, Village, Smithy and Salvager functioned properly for the most part. All of the unit tests, except for Salvager, achieved close to 100% coverage. Even though the Salvager unit test could have performed better on coverage, it found a bug where the card failed to discard after activation.

Another debug method was the random test. I created some random testers for two of Dominion's actions cards. The two cards are Village and Adventurer. The random tester for Village randomizes several factors within the Dominion game, such as number of players and number of cards in hand, to check for bugs. For this case, if Village is free of bugs, it will allow the any user to draw two cards, attain one action, discard the action card, and increase number of cards in discard pile by one in every randomized test case. The random tester for Adventurer follows a similar procedure except that Adventurer's card effect is different.

I soon created a larger random test which simulated full games of Dominion. The random tester generated similar randomized states as the individual card effect random tester. In a typical test scenario, once the initial randomized game state is prepared, the players' decisions during the buying phase and the action phase will be heavily dictated by the random

seed. The players would continue playing until the win conditions are met, and the game state will be freed to start another test case.

After simulating 10 full games of Dominion on zakrevsm's source code, the test case's coverage was 77.62% over 581 lines of code. When I increased the number of test cases to 100, the test case's coverage improved to 81.24% of 581 lines. These results are acceptable, considering that my tester skipped the Feast function for better flexibility when testing. The reason being that the Feast function is responsible for countless infinite loops throughout many test runs. Even when it does not trigger that obnoxious infinite loop, the function fails to to decrement players' treasure count on a consistent basis. Another buggy function that was malicious was Sea Hag. When a player activates Sea Hag, the action card can potentially eradicate the other players' decks, resulting either an infinite loop or a game of Dominion where players only skip turns. Sea Hag's bug is not as prevalent as Feast's bug, so it is usually not omitted during simulations; however, if the max test value is in a range beyond 100, there is high probability that the Sea Hag's bug will appear, and the function should be omitted. In addition, I found a bug in his Gardens function when players were able to activate the card in the simulations. After checking his Dominion source code, it appears that Gardens was one of the cards that he had chosen to refactored where it will return 0 instead of -1. In addition, this subtle change broke the initial nine action card selection where action cards can be chosen numerous times after Gardens' initial selection, and only Gardens would be chosen many times shortly afterwards, breaking the nine card limit in the process.

After simulating 100 full games of Dominion on whitewi's source code, the code had a coverage 78.18% of 573 lines. The issues with Feast and Sea Hag were also present in whitewi's source code; however, I suspect that the infinite loop associated with the Sea Hag bug was more prevalent here, since my testdominion random tester required many more attempts before it was able to successfully simulate 100 full games. Additionally, some bugs that were unique to his source code include Outpost usually does nothing upon activation, Embargo fails to add two treasure, and Treasure Map does not activate properly.

After reviewing the Dominion source code from my two classmates, zakrevsm and whitewi, I am significantly confident that the reliability of either implementations needs major improvement. For clarity, the majority, if not all, of the Dominion source code was already very buggy in the first place. Further, when everybody inserts more bugs into the source code, the probability that the reliability will improve is low. Ultimately, in order to improve the reliability, I believe it would be better to begin the implementation from scratch again rather than debugging all of its current issues. The source code is riddled with too many subtle bugs that require copious amounts of time to pinpoint and solve.