Morgan Howard
CS362
Assignment 3 Write-up
April 30, 2017

Bugs

During the course of my unit testing, I found several bugs in the dominion code:

- endTurn() doesn't move the cards in the playing area to the discard pile, as dictated by the dominion rules. I noticed this bug when I was trying to compare the cards in the discard pile with the cards in the player's hand, cards the player played, and cards purchased.
- endTurn() doesn't have the current player draw 5 new cards at the end of the cleanup phase as dictated by the dominion rules. I noticed this bug when I was trying to test the business requirements of the cleanup phase.
- buyCard() relies on state->coins to determine purchasing power/ability. However, when buying a card, the treasure cards from your hand that are used to make that purchase don't actually leave the hand until endTurn(). The dominion rules state that treasure cards used to buy a card should be placed in the playing area, and then discarded from there at the end of the cleanup phase.
- card_smithy() has a bug I introduced during assignment 2, where it causes the player to draw 4 new cards instead of 3 cards.
- card_adventurer() has a bug that I noted in assignment 2 where it doesn't discard the adventurer card.
- discardCard() doesn't add to a card to the discard pile directly as you would expect, though this behavior is correct according to the rules (played cards go in the playing area). It should be renamed or refactored.
- card_village() has a bug that I introduced in assignment 2 where it adds 3 actions instead of 2.

Unit Testing

For the functions and cards that I covered with my unit tests, I got pretty good coverage (100% on all except the adventurer card, which was 60%). However, when I look at the total coverage on dominion.c, it was pretty low (20%). First off, 8 unit tests are not enough to cover even a small program like dominion. I would easily need 5 times as many tests, possibly more if I want to keep them small and targeted. Also, the dominion code can use a lot of fixing and refactoring. While fixing and refactoring might make more lines of code, they would make it easier to target business requirements and program functionality.

If I was to continue on with improving and testing dominion, I would start with finishing the refactoring of all the case statements in cardEffect(). Then, I would move on to the core functions that are used in a lot of places that represent basic game actions (draw, play, buy,

discard, shuffle, etc). For it to be considered well-covered, I'd imagine that it would need to at least be in the 80% range, if not the mid-90s.

Unit Testing Efforts

Overall, I found the process of unit-testing to be really interesting and enjoyable. I found the card tests to be the easiest to write, especially the cards that I had already refactored for assignment 2, mostly because I had already thought about what they were supposed to do. The function tests were harder to write, but were probably more interesting because you had to really dig into the other functions that were called to understand how they interrelated.

Once I selected which cards and functions I was going to test, my general process for unit testing consisted of reading the rules for those items, and writing out the "business requirements" for each one. The easiest requirements to write were the things that should change; the harder requirements were for the things which shouldn't change, which was generally a long list. Then, I read the source code to get a sense of how the code matched up to the requirements. Then, I thought about how I would functionally test each business requirement using the source code as is.

I prioritized function over coverage, though at the end of the process I had pretty good coverage for most of the items. Cardtest2 (card_adventurer), unittest2, and unittest3 had the lowest coverage percentages with 63%, 93%, and 95%, respectively. I was able to go back and get unittest2 and unittest3 to 100% coverage by adding some edge cases. For untittest2, I added a bad value to check if getCost() would return -1 for a non-existant card. For unittest3, I added a check to verify that endTurn() would follow player 1 with player 0. Card_adventurer() is lacking coverage for scenarios that would happen later in the game (e.g. having to shuffle the discard pile back into the deck so you have enough to draw from; getting more cards in circulation so it's less likely that you'll draw 3 treasures in a row; and discarding a temphand from the previous scenario).