John Knopp
CS 362
June 9, 2015

Dominion Testing Report

Overview

This document will detail my testing procedures for the class implantation of Dominion. Testing methods including unit testing, random game state and random game testing will be discussed, as well as coverage and code reliability for Chris Martin's and Chad Kunde's versions of Dominion.

Unit testing

Function Testing

My unit testing comprised of testing both specific functions and implementations of cards found within the cardEffect function. The individual functions I tested for were compare, isGameOver, kingdomCards,, and updateCoins. These tests attempted to cover all statements/basic blocks of the functions. An example test of kingdomCards is shown below in Figure 1. These tests proved to find no bugs other than that no functions handled null pointer game structs and that updateCoins accepted a negative coin bonus, an impossibility within the game. These bugs were true of both Chris's and Chad's code. However, since all these are helper functions used within other functions that always pass along valid game structs and coin amounts, these bugs (if one can call them that) have a low priority and negligible severity.

```
int main (int argc, char** argv) {

    printf("***Testing kingdomCards***\n");
    printf("Entering random array...\n");
    int i;
    int r;
    int* initial_array = malloc(10 * sizeof(int));
    srand(time(NULL));


    for(i=0; i<10; i++){
        r = rand();
        initial_array[i] = r;
    }

    int* return_array = kingdomCards(initial_array[0], initial_array[1], initial_array[2],
            initial_array[3], initial_array[4], initial_array[5], initial_array[6], initial_array[7], initial_array[8], initial_array[9]);

    for(i=0; i<10; i++){
        assert(return_array[i] == initial_array[i]);
    }

    printf("All tests pass\n");

    return 0;

}
```

Figure 1: Unit Test for Kingdom Cards

Card Testing

Unit testing for specific cards proved to be more fruitful than that of function testing. Due to the time crunch of final's week I was unable to write unit tests for every single card, but was still able to find several bugs. My card tests were written for outpost, gardens, steward, council room, great_hall, village, and a special case of Baron, which will be discussed in detail later. These tests checked the game state following the play of the card against the cards intended effect. A simple case of the unit test for the garden card is shown below in Figure 2. In Chris's code I was able to find to find two considerable bugs. In his implementation of great_hall two cards were drawn instead of one.. The other bug I found was in the village card, where instead of 2 actions being added to the player's number of actions, only one is. This limits the player's number of turns in the action phase and would be rather frustrating. My card

tests for Chad's code were unable to locate any bugs. More unit tests should be written, as I suspect there may bugs within other cards.

```
int main () {
    printf("***Testing gardens***\n");
    printf("Testing whether gardens returns -1...\n");
    struct gameState G;
    int k[10] = {village, smithy, gardens, tribute, baron, adventurer, cutpurse, mine, embargo,
           outpost};

    initializeGame(2, k, 3, &G);

    G.handCount[0] = 3;
    G.hand[0][0] = gardens;

    int r = playCard(0, 1, 0, 0, &G);
    assert(r ==  -1);

    printf("Test passes\n");

    //playCard(0, 1, 0, 0, NULL);


}
```

Figure 2: Card Test for Garden

Random Testing

Random Game State Testing for Village and Adventurer

My random testing consisted of two different types of random testing including random testing of specific cards in random game states, and randomly playing through whole games of dominion. I randomly tested both the village and adventurer cards by putting the game into a random game state including a random number of players, and decks (discard, hand, and deck), playing the cards, and then checking to see if the cards had the intended result. The village test is shown in Figure 3.

```
for (i = 0; i < MAX_TESTS; i++) {

    printf("*Test %d*\n", i);
    players = 2 + (rand() % 3); //Set players between 2 and 4
    printf("players is %d\n", players);

    player = rand() % players;

    seed = rand();      //pick random seed

    int r = initializeGame(players, k, seed, G);    //initialize Gamestate
    assert(r == 0);
    G->whoseTurn = player;

    //Initiate valid state variables
    G->deckCount[player] = 10 + (rand() % 490); //Pick random deck size out of MAX DECK
    deckCount = G->deckCount[player];
    printf("deckCount is %d\n", deckCount);

    G->discardCount[player] = rand() % (deckCount);
    discardCount = G->discardCount[player];
    printf("discardCount is %d\n", discardCount);

    G->handCount[player] = rand() % MAX_HAND;
    handCount = G->handCount[player];
    printf("handCount is %d\n", handCount);
    numAction = G->numActions;


    r = cardEffect(village, 0, 0, 0, G, 0, 0);      //Run village card

    if(r != 0){
        failedTests++;
        printf("FAILED: cardEffect returned improper value");
    }

    if( handCount != G->handCount[player]){
        failedTests++;
        printf("FAILED: player did not draw a card and then discard one\n");
    }
```

Figure 3: Random Village Test

I ran this test 1000 times on both Chris's and Chad's code. This resulted in no failing cases (when accounting for Chris's error in the number of actions as discussed above) in either players Dominion implementations of village, but led to several failed test cases in both players versions of adventurer both involving treasures being added incorrectly, as shown in Figure 4. This was a very rare issue though, with less than 10/1000 cases of treasure being added incorrectly in both players code.

```
***Test 185***
players is 2
deckCount is 159
discardCount is 29
handCount is 422
PASSED
***Test 186***
players is 4
deckCount is 217
discardCount is 205
handCount is 40
PASSED
***Test 187***
players is 2
deckCount is 157
discardCount is 86
handCount is 297
PASSED
***Test 188***
players is 2
deckCount is 409
discardCount is 308
handCount is 186
PASSED
***Test 189***
players is 2
deckCount is 16
discardCount is 4
handCount is 347
FAILED: treasures not added
```

Figure 4: Treasures not being added in Random Testing

Random System Testing for Whole Games of Dominion

My random testing for entire games of dominion consisted of the following setup: choose a random number of players, and a random set of kingdom cards. When it was a player's turn, the player would look through their hand and play the first available kingdom card. They would then proceed to the buy phase where they would a random card that was in the price range for their given hand. This allowed for players to buy curses, which is not a great strategy but led to some very interesting final scores.. Each player would repeat this procedure until the isGameOver function yielded a true result. The test for this was to large for this document but can be seen in randomtester.c under my github directory of the class repository.

I ran my random tester on both Chris's and Chad's code and found an error with the kingdom card baron common to both of them. When baron was played, and the choice to gain 1 estate card was selected,

the estate supply pile would be depleted by 2, rather than 1. This was especially problematic when the supply count for the estate is at 1, as after the baron was played, the supply count would become -1. This is an invalid game state and especially becomes a problem when checking to see if the game is ended. This led me to write a specialized unit test putting the game in the situation just described to see if the baron was indeed the problem. This test is shown in Figure 5.

```
int main () {
    printf("***Testing baron***\n");
    int p = 0;
    struct gameState G;
    int k[10] = {salvager, smithy, gardens, tribute, baron, adventurer, cutpurse, mine, embargo,
            council_room};

    initializeGame(2, k, 3, &G);

    //set the supply count of estate to 1
    G.supplyCount[estate] = 1;

    //put baron in hand
    G.hand[0][0] = baron;


    printf("Testing Baron with 1 estate card in supply count...\n");

    //play card and choose 0 for choice 1 to gain an estate card
    playCard(0, 0, 0, 0, &G);
    printf("There are %d estates in supply after playing this card\n", G.supplyCount[estate]);
    if(G.supplyCount[estate] ==  0){
        printf("Test passes\n");
    }
    else{
        printf("Test fails\n");
    }



}
```

Figure 5: Testing whether Baron causes Estate count to fall to -1

After running this test, baron proved to be the problem. Specifically there was an extra decrement of the supply count located in the baron implementation. Removing this line solved the problem.

This was the primary bug my random tester found with Chad's code, but there were several more located in Chris's. Chris's Dominion implementation would get caught in a loop probably 1/8 times when a different seed was provided. I found this was the case when the feast card was played. I was unable to test the feast card any further due to time constraints but recommend a further investigation as to why this is happening, as it makes the game unplayable if the feast card is in the set.

Coverage

As shown in figure 1, all of my tests combined including just 1 random run through the entire game, results in about 60% code coverage. Since only 10 of the 17 kingdom cards are used, I suspect more runs of the random test would result in around 70-75% coverage. This is fine, but higher code coverage is desirable. I think the issue with my coverage is that, not all choices for each card are being covered,

some of them being fairly large branches of code. My random tester would need some improvements to increase the coverage for all choices.

```
File '../../martich2/dominion/dominion.c'
Lines executed:56.34% of 749
../../martich2/dominion/dominion.c:creating 'dominion.c.gcov'
```

Figure 6: Code Coverage for my Tests on Chris's Code

Reliability

Other than a few undesirable quirks in rare game play states, Chad's code worked very well. I could see users playing on it with very few issues. Chris's Code on the other hand, is unplayable if one desires to use either the village card due to its extra card drawing error, or the feast card, where an infinite loop causes the program to be exited manually without any information on the saved game state. I suspect the latter issue could be fixed in an hour or two with a debugger and should be addressed before letting users play his version of the game.