CS 362 Final Test Report

Nathan Murrow

June 11, 2012

1 The Tests

1.1 Structure of a Test Case

Each test case takes an initial game state, performs a particular operation that we wish to test, and outputs the final game state. For this project the main features that we want to test are the buyCard function and the Adventurer card effect. Because there are so many sample implementations of dominion to choose from, I decided that differential testing would be enlightening, so the test suite compares the outputs of each implementation to each other.

1.2 Length of Test Cases

In time, each test case is a single execution of a function, which takes a split second. In space, the test case takes a couple dozen lines of C code to set up the initial state and several more actually test and show the results for that initial state (although the setup scripts written in python are substantially longer; these scripts automate copying, compiling, and indexing the results of each person's implementation). In number, only several hundred cases are tested because there is a strict time limit of five minutes on execution time for this assignment.

1.3 Method for Choosing Inputs

The input states are chosen randomly, with some boundary restrictions in order to make sure the all relevant types of test cases are covered. Generally, values are kept to what might be expected to appear in an actual game of Dominion. However, in the case of Adventurer testing, for example, the deck sizes are kept intentionally small to allow for the situation where the deck must be reshuffled. This also allows the tester to run more quickly because it does not need to assign values to or search through a large number of cards.

1.4 Specification Method Used

Because this test suite uses differential testing, specifications do not need to be clearly defined to demonstrate the existence of bugs. For generating statistics, the test suite assumes that the most common output is the one that adheres to specifications but it is up to the testing person to determine whether the specifications are being met or if this is either a coincidence or a matter of miscommunication of specifications. I feel that this is another strength of differential testing; it can expose vague specifications even where there are no concrete specifications.

1.5 How to Handle Segfaults/Crashes

Because the tester works by opening subprocesses to test, segfaults in the tested code do not crash the test suite. Instead, the tester will notice that it did not receive any resulting data from this test case and will treat it as a failure. Failures are never acknowledged as meeting specifications.

1.6 Development History of the Test System

The test system began with basic unit tests using random inputs and a simplified set of specifications. I decided a more robust test system was needed, and I was really interested in using differential testing, so I rewrote the test driver to report changes in game state without asserting whether they were correct or not. I then wrote a python script to automate copying and compiling each user's dominion code. It seemed useful enough to include in the testing suite, so I made it a primary feature and added output indexing and statistics reporting. The script will automatically select and report (usually) representative test cases based on who failed and who succeeded.

1.7 Thoroughness of Testing

The tester only covers BuyCard and Adventurer. These are of interest because each student was required to implement them, so the implementations are likely to be different. Other functions come from a common source, so differential testing is not likely to produce interesting results. As such they were omitted from these tests even though it would be simple to include them. This resulted in 8% to 34% code coverage, depending on implementation.

1.7.1 Remaining Risks of Failure

The primary risks of failure are bugs that are common to all implementations tested. Differential testing will only find differences between implementations; it won't find a common bug. This risk can be eliminated by having access to an implementation that is known to be correct, but this is not the case for this project.

Although I have not experienced it, my test suite is theoretically susceptible to infinite loops in testing code. I considered altering the tester to set a timeout on execution, but didn't since this did not become a problem.

Another interesting thing to note is the disparity between Adventurer and BuyCard tests. The BuyCard reports generated by this program show which implementations are good or bad clearly and why. However, it seems that either no one can agree on an Adventurer specification, or everyone failed their implementation in radically different ways. Differential reports work best when there is one clearly correct implementation to compare against, but in the case of Adventurer it is not possible to determine which implementation is correct automatically.

2 The Testing Process

2.1 Communications with Partners

Communications with partners was sparse. A few bug reports did change hands, and in the case of my own reports, all of the information is available in the repo, but overall students have been more interested in finding bugs than fixing them (and fixing bugs makes them more difficult to find!).

2.2 Example Bug Reports

My script generates bug reports as a string of named values indicating the values of various relevant variables in the game state. Names are attached to a set of values to indicate the resulting state for that user's implementation. The initial state and the most popular state are also clearly labeled, as well as how many occurrences of this type of test case there have been.

INITIAL : c15 nP1 wT0 sC8 nB0 $3 dC234
RESULTS
Majority (6): c15 nP1 wT0 sC8 nB0 $3 dC234
wheeleri : c15 nP1 wT0 sC8 nB-1 $-1 dC235
murrown : c15 nP1 wT0 sC7 nB-1 $-1 dC235
Occurrences : 203/500

This is a BuyCard test case. Each series of values, from left to right: card to buy, number of players, whose turn it is, supply count of the relevant card, number of buys left, number of coins, and discard count. As you can see, the majority of implementations agree that there should be no change in state, probably because the number of buys is zero, or 3 coins is not enough to buy card

15. However, for murrown and wheeleri, somehow the purchase is made anyway, resulting in negative values and a new card to the discard pile. This kind of situation, where wheeleri and murrown disagree with the majority, occurs in 203/500 cases, so it's kind of a serious problem.

INITIAL : nP1 wT0 nA3 $1 pC148 ddC13 dkC9 hC7 hP2 d$0
RESULTS
Majority (5): Failure
westb : nP1 wT0 nA2 $0 pC148 ddC13 dkC9 hC7 hP2
adamsmic : nP1 wT0 nA2 $0 pC149 ddC22 dkC9 hC6 hP2
brookjon : nP1 wT0 nA2 $0 pC148 ddC0 dkC12 hC17 hP2
Occurrences : 155/500

This is a test for Adventurer. What's interesting about this case is that there is no treasure in either the deck or the discard (as indicated by "d$0"), so Adventurer will never find two treasure cards. As you can see, five out of eight implementations do not account for this case and result in a SEGFAULT. The other three implementations will still execute, but sadly, they all result in different states so further analysis is required to determine which, if any, is correct. From here, it appears that *westb* is most correct, because the number of cards in hand ("hC7") do not change.

In addition to this information on test cases, the tester will also indicate which implementations were the most robust overall.

Successes out of 500 tests:
murrown : 270
adamsmic : 500
brookjon : 230
kropfb : 440
westb : 203
vanbeeks : 500
wheeleri : 0
mcelfrec : 230
adamsmic and *vanbeeks* have the most agreeable BuyCard implementations, so it might be handy to look at them in particular for future comparisons.

2.3 Revisions to Tested Code

It seems that few revisions to the code have been made. The results of my tester are generally the same now as when I created it. It is likely that there is no interest in fixing the code for the reasons mentioned above.

2.4 Regression Strategy

Because there have been so few revisions, there has not been much opportunity for regression testing. However, once a clear specification is established, differential testing will be very effective at detecting subtle differences in code unless many people make the same subtle change at the same time. Thus, this system will be effective for regression testing. By keeping back-ups of old implementations, it can even be used to compare revisions against "last known good" implementation instead of each other.

2.5 Quality of Code Initially

The code at the start of this class was a mess and intentionally so. This made testing more fun and interesting. Of course, not all bugs were introduced by students this term. Some bugs are in the common code base, and these are the most difficult to find.

2.6 Quality of Code After Testing

Because of the lack of interest in revising code, the quality of the code remains largely unchanged. My tests generate approximately the same number of failure cases now as they did then.

2.7 Final Status of Bugs

As I predicted before, the bugs will not get fixed before the term is over. Their official status is WONTFIX.

3 Future Plans

3.1 Reflections

If I had more time I would consider differential testing in a larger scope. This could include creating more unit tests or attempting to test actual game executions. Either way, code coverage would increase, which is relatively low at this time. Of course, the effectiveness of differential testing is dubious here because each implementation shares a common code base. This is an issue that would ultimately need to be dealt with, so alternative testing methods would be considered.

3.2 Effectiveness of Tools Used

I mostly wrote my own tools and infrastructure, but I did use gcov to determine code coverage. It did its job and provided very detailed statistical info that may be useful for debugging, but mostly it just informed me that my tester will only find bugs in 20% of the code (however, it would not be difficult to bring that number up with more unit tests).

3.3 Infrastructure Needs

I don't know if tools already existed that did what I needed to program. It didn't take too long, but the most tedious part was getting everyone's code to compile and link correctly automatically. Of course, I needed gcov, and I doubt I would be able to program that in a few weeks. I mentioned before that my tester is weak to infinite loops. It would be nice to have an easy way to place timeouts on each test case but some cursory research suggests that this is not as easy a task as it sounds; the common solution is to use threads.

3.4 Automated Testing vs. Unit Tests vs. Code Inspection

Automated testing is a natural response to the repetitive tasks required in testing software. In my opinion, automated testing is very easy, but its usefulness might only extend to discovering bugs, not necessarily finding or executing them. With a sophisticated automated testing system, this might be less of an issue but of course that would be more difficult to program, and the real world might not have luxuries like many different implementations to use in differential testing. Ultimately, someone is going to have to inspect the code to debug it, but it seems that automated testing can help reduce the amount of inspection that needs to occur. I like that, and would like to avoid doing a lot of code inspection.

4 Additional Information

4.1 Code Coverage

File 'murrown.c'
Lines executed:26.23% of 530

File 'adamsmic.c'
Lines executed:21.19% of 538

File 'brookjon.c'
Lines executed:23.78% of 534

File 'kropfb.c'
Lines executed:34.18% of 550

File 'westb.c'
Lines executed:15.72% of 528

File 'vanbeeks.c'
Lines executed:17.96% of 529

File 'wheeleri.c'
Lines executed:7.77% of 528

File 'mcelfrec.c'
Lines executed:11.20% of 527

4.2 Full Test Data

```
------------------------------------------------------------
ADVENTURER TESTING
------------------------------------------------------------

INITIAL    : nP1 wT0 nA3 $1 pC148 ddC13 dkC9 hC7 hP2 d$0
RESULTS
Majority (5): Failure
westb      : nP1 wT0 nA2 $0 pC148 ddC13 dkC9 hC7 hP2
adamsmic   : nP1 wT0 nA2 $0 pC149 ddC22 dkC9 hC6 hP2
brookjon   : nP1 wT0 nA2 $0 pC148 ddC0 dkC12 hC17 hP2
Occurrences : 155/500

INITIAL    : nP3 wT2 nA4 $1 pC384 ddC14 dkC18 hC19 hP1 d$5
RESULTS
Majority (2): Failure
westb      : nP3 wT2 nA3 $0 pC384 ddC14 dkC18 hC19 hP1
adamsmic   : nP3 wT2 nA3 $1 pC385 ddC23 dkC16 hC20 hP1
murrown    : nP3 wT2 nA3 $4 pC393 ddC14 dkC7 hC21 hP1
brookjon   : nP3 wT2 nA3 $4 pC384 ddC0 dkC13 hC38 hP1
kropfb     : nP3 wT2 nA3 $3 pC385 ddC14 dkC7 hC20 hP1
```

```
vanbeeks   : nP3 wT2 nA3 $4 pC384 ddC23 dkC7 hC21 hP1
Occurrences : 158/500

INITIAL    : nP1 wT0 nA4 $8 pC334 ddC6 dkC10 hC19 hP12 d$1
RESULTS
Majority (4): Failure
westb      : nP1 wT0 nA3 $0 pC334 ddC6 dkC10 hC19 hP12
adamsmic   : nP1 wT0 nA3 $0 pC335 ddC15 dkC9 hC19 hP12
brookjon   : nP1 wT0 nA3 $2 pC334 ddC0 dkC5 hC30 hP12
vanbeeks   : nP1 wT0 nA3 $4 pC334 ddC1999 dkC-1985 hC21 hP12
Occurrences : 18/500

INITIAL    : nP1 wT0 nA0 $3 pC210 ddC17 dkC9 hC19 hP9 d$1
RESULTS
Majority (8): nP1 wT0 nA0 $3 pC210 ddC17 dkC9 hC19 hP9
Occurrences : 50/500

INITIAL    : nP1 wT0 nA2 $0 pC177 ddC9 dkC19 hC13 hP1 d$3
RESULTS
Majority (3): Failure
westb      : nP1 wT0 nA1 $0 pC177 ddC9 dkC19 hC13 hP1
adamsmic   : nP1 wT0 nA1 $0 pC178 ddC27 dkC18 hC13 hP1
murrown    : nP1 wT0 nA1 $5 pC196 ddC0 dkC7 hC15 hP1
brookjon   : nP1 wT0 nA1 $2 pC177 ddC0 dkC8 hC33 hP1
kropfb     : nP1 wT0 nA1 $3 pC178 ddC0 dkC7 hC14 hP1
Occurrences : 113/500

INITIAL    : nP1 wT0 nA7 $7 pC359 ddC17 dkC10 hC6 hP0 d$6
RESULTS
Majority (2): nP1 wT0 nA6 $3 pC359 ddC17 dkC8 hC8 hP0
westb      : nP1 wT0 nA6 $0 pC359 ddC17 dkC10 hC6 hP0
adamsmic   : nP1 wT0 nA6 $3 pC360 ddC17 dkC8 hC7 hP0
wheeleri   : Failure
mcelfrec   : Failure
brookjon   : nP1 wT0 nA6 $5 pC359 ddC0 dkC16 hC17 hP0
kropfb     : nP1 wT0 nA6 $0 pC360 ddC17 dkC8 hC7 hP0
Occurrences : 6/500

Successes out of 500 tests:
murrown    : 56
adamsmic   : 50
brookjon   : 50
kropfb     : 50
westb      : 50
vanbeeks   : 56
wheeleri   : 50
mcelfrec   : 50

------------------------------------------------------------
BUYCARD TESTING
------------------------------------------------------------
```

```
INITIAL    : c15 nP2 wT1 sC4 nB2 $6 dC61
RESULTS
Majority (4): c15 nP2 wT1 sC3 nB1 $2 dC62
westb      : c15 nP2 wT1 sC3 nB2 $6 dC62
wheeleri   : c15 nP2 wT1 sC4 nB1 $2 dC62
mcelfrec   : c15 nP2 wT1 sC2 nB1 $2 dC63
brookjon   : c15 nP2 wT1 sC4 nB1 $6 dC62
Occurrences : 237/500

INITIAL    : c15 nP1 wT0 sC8 nB0 $3 dC234
RESULTS
Majority (6): c15 nP1 wT0 sC8 nB0 $3 dC234
wheeleri   : c15 nP1 wT0 sC8 nB-1 $-1 dC235
murrown    : c15 nP1 wT0 sC7 nB-1 $-1 dC235
Occurrences : 203/500

INITIAL    : c7 nP3 wT0 sC2 nB1 $8 dC280
RESULTS
Majority (3): c7 nP3 wT0 sC1 nB0 $2 dC281
westb      : c7 nP3 wT0 sC1 nB1 $8 dC281
wheeleri   : c7 nP3 wT0 sC2 nB0 $2 dC281
mcelfrec   : c7 nP3 wT0 sC0 nB0 $2 dC282
brookjon   : c7 nP3 wT0 sC2 nB0 $8 dC281
kropfb     : Failure
Occurrences : 33/500

INITIAL    : c16 nP3 wT2 sC8 nB0 $4 dC28
RESULTS
Majority (4): c16 nP3 wT2 sC8 nB0 $4 dC28
westb      : c16 nP3 wT2 sC7 nB0 $4 dC29
wheeleri   : c16 nP3 wT2 sC8 nB-1 $1 dC29
murrown    : c16 nP3 wT2 sC7 nB-1 $1 dC29
kropfb     : c16 nP3 wT2 sC7 nB-1 $1 dC29
Occurrences : 27/500

Successes out of 500 tests:
murrown    : 270
adamsmic   : 500
brookjon   : 230
kropfb     : 440
westb      : 203
vanbeeks   : 500
wheeleri   : 0
mcelfrec   : 230

------------------------------------------------------------
GCOV INFO
------------------------------------------------------------

File 'murrown.c'
```

```
Lines executed:26.23% of 530
Branches executed:25.18% of 409
Taken at least once:18.34% of 409
Calls executed:11.96% of 92
murrown.c:creating 'murrown.c.gcov'

File 'adamsmic.c'
Lines executed:21.19% of 538
Branches executed:22.52% of 413
Taken at least once:15.74% of 413
Calls executed:11.58% of 95
adamsmic.c:creating 'adamsmic.c.gcov'

File 'brookjon.c'
Lines executed:23.78% of 534
Branches executed:24.11% of 419
Taken at least once:17.66% of 419
Calls executed:13.68% of 95
brookjon.c:creating 'brookjon.c.gcov'

File 'kropfb.c'
Lines executed:34.18% of 550
Branches executed:29.50% of 417
Taken at least once:22.06% of 417
Calls executed:21.43% of 98
kropfb.c:creating 'kropfb.c.gcov'

File 'westb.c'
Lines executed:15.72% of 528
Branches executed:19.80% of 409
Taken at least once:11.98% of 409
Calls executed:12.37% of 97
westb.c:creating 'westb.c.gcov'

File 'vanbeeks.c'
Lines executed:17.96% of 529
Branches executed:21.65% of 411
Taken at least once:14.84% of 411
Calls executed:9.78% of 92
vanbeeks.c:creating 'vanbeeks.c.gcov'

File 'wheeleri.c'
Lines executed:7.77% of 528
Branches executed:7.82% of 409
Taken at least once:7.09% of 409
Calls executed:1.10% of 91
wheeleri.c:creating 'wheeleri.c.gcov'

File 'mcelfrec.c'
Lines executed:11.20% of 527
Branches executed:10.86% of 405
```

```
Taken at least once:9.38% of 405
Calls executed:5.26% of 95
mcelfrec.c:creating 'mcelfrec.c.gcov'
```