FROM: Miles Van de Wetering

TO: Alex Groce

DATE: June 6 2016

SUBJECT: Software Engineering – CS 362 – Dominion Test Report (vandewmi, gibsonri)

Summary

For the final portion of this project, I compared my code (vandewmi) and testing results against Rikki Gibson's (gibsonri). Unfortunately, neither of our dominion implementations are anywhere near ready for deployment. In this report is a summary of (known) failing test cases – there are likely many more, but in my opinion these failures should be addressed before more testing occurs. This is because many of these failures are likely due to errors imbedded deeply in dominion.c, rather than 'shallow' errors existing only in, for example, card effect functions. I discuss the code coverage achieved by our current test cases, and the implications of this level of coverage for future testing efforts. I discuss random and mutation testing efforts and their relevance to improving the code base, and lastly I make holistic recommendations for the redesign of the code in order to improve testability and reliability moving forward.

Unit Tests

Most unit tests are currently failing. I have implemented unit tests for the initializeGame function, the isGameOver function, the buyCard function, Great Hall card, the Outpost card, the Smithy card, and the Minion card. The results are summarized in Table 1. It is not difficult to see that the nature and consequences of each of our observed errors are extremely varied, and it is unlikely that these errors are localized in a small portion of the code. These unit tests execute about 40% total of the dominion code, which is a good start. Unfortunately, much of this coverage is concentrated in simplistic functions – it is likely that the remainder of dominion.c contains many, many more errors. In order to ensure a correctly functioning dominion implementation for deployment, I likely need unit tests for every card function as well as each helper function. At this time I only have unit tests for four of the card functions and four of the helper functions.

Random Testing & Mutation Testing

In addition to prescriptive unit tests, I have also developed a number of random testing scenarios for dominion. These random tests resulted in more overall coverage of the dominion code, about 30% per test (though these tests do overlap). Unfortunately, several of these tests are unable to serve as excellent indicators of very many bugs, since several of the cards tested have catastrophic errors resulting in segfaults in both my own code and Rikki's, generally rendering these tests unusable. It is unlikely that these cards are testable by unit or random tests at this time – a minified failing test case should be generated in each case to help determine what is causing the segfaults. Interestingly, the adventurer card contains a segfault but only in apparently very specific situations. I have not yet determined which exact parameters cause this card to fail, but doing so should be relatively straightforward. Overall, I think that random testing will be a good way to ensure that code runs in all situations, but I don't believe that they are the best way forward in light of my proposed code redesign – they don't help us identify where or what each bug really is.

I have also implemented mutation testing in order to ensure that the kind of unit tests I have designed are sufficient. I tested against the Minion card, which in my own code has been factored out into the cards.c file. This card is currently passing the greatest proportion (99%) of its unit test, which is why I chose it for mutation testing. After generating 15 random mutations, every single mutation failed to pass the unit test. I believe that this is a strong indicator that the current unit test design is robust, and can easily be extended to safely test each and every card effect.

Whole-Game Comparison Testing

Where I have neither the time nor the interest to implement detailed unit tests or random tests for a given portion of the dominion code, I think that the whole-game comparison testing that I have implemented will be of use. This allows me to identify places where the code in one implementation differs from the code in another. While this will not help with identifying *consistent* incorrect behavior, most of the incorrect behavior in dominion is not consistent. Thus, by examining differences in behavior I may identify weak points in both implementations that are likely to need investigation and perhaps a redesign.

Recommendations

I think that the first priority moving forward should be to factor out every card effect into its own function. This makes minimizing failing test cases much, much faster, and will likely improve the maintainability of our code. Next, the core of dominion.c should be redesigned in order to reach a cohesive approach to public and private functionality. The update_coins function is a prime example of this – this function sets the state->coins to zero, and is often called after card effects which themselves modify the state->coins variable. Abstracting these global variables away is urgently needed, and will likely reveal many inconsistencies and bugs in the existing code base. My second recommendation is that each player in the game be represented in their own structure, rather than having their information scattered across the gameState struct in several arrays. This would allow card functionality to concisely operate on a single player at a time, without worrying about accidentally indexing into the wrong player and causing detrimental side effects.

All in all, both of these implementations are completely unstable and unreliable. I believe that a full redesign is in order no matter which code base is chosen as a starting point. Both seem to be subject to many of the same primary failings; more fine-grained bug analysis is unlikely to be of much use until the larger problem is addressed.

Table 1

Student	Card/Function	Pass/Fail	Reason for Failure
Rikki	Fail	Pass	Invalid kingdom card allowed into the game
Rikki	isGameOver	Pass	
Rikki	buyCard	Pass	
Rikki	endTurn	Pass	
Rikki	Great Hall	Pass	
Rikki	Outpost	Fail	Player doesn't get a second turn appropriately, they draw an incorrect number of cards for their second turn, and they never stop getting more turns.
Rikki	Minion	Fail	Player's don't discard their hands correctly, they don't draw the correct number of cards, adds gold when they shouldn't, and doesn't add gold when they should.
Rikki	Smithy	Fail	Card doesn't get discarded appropriately
Miles	Fail	Pass	Invalid kingdom card allowed into the game
Miles	isGameOver	Pass	Player doesn't get a second turn appropriately, they draw an incorrect number of cards for their second turn, and they never stop getting more turns.
Miles	buyCard	Pass	
Miles	endTurn	Pass	
Miles	Great Hall	Pass	
Miles	Minion	Pass	
Miles	Smithy	Fail	Card doesn't get discarded appropriately