Test Report CS362

Harrison Alan Kaiser

Summary of Findings

I had not played Dominion before this class, though it had been on my Amazon Wishlist for a time. Thus my first order of business was to purchase the game and play. In doing so I understood what the system was supposed to be and could better understand the concepts that needed to be applied within the program.

I believe this implementation of Dominion to be heavily bugged. Many bugs were game breaking and created unexpected or unpredictable game states which went against the rules of the game.

Unit testing found no errors but card tests revealed that some cards were not discarded. This also revealed to me the first major bug found which was that the cases in cardEffect were adding coins directly to the state instead of adding to the implemented bonus coin variable. This caused the coins to be lost upon the conclusion of playCard as it called updateCoins.

Another major bug brought to light was the playedCard array within the state struct. Whenever cards are played they are added to this but nowhere in any of the code is their functionality to return these cards to the discard pile where they belong. This is a major flaw which makes action cards one time use and they simply stack up in the unused array.

Creation of the tests for adventurer and steward were made easier with my discovery of the previous bugs discussed. I was able to assert these errors to be true in testing and modified my implementation to successfully execute and satisfy the proper game state upon being enacted.

One of the more difficult tasks with this code was creating a random tester which would complete without crashing or looping on a bugged card. The major issue causing this was that some choices on cards were not error checked and were used at times in ways which created unintended results. I corrected many of these but certain cards were turned off as it was apparent they were causing issues due to their implementation and needed to be addressed. As such the testing had identified the most important cards to be addressed in debugging.

I completed a code inspection with two of my classmates and this revealed another level of bugs I had not even considered. scoreFor was counting curses for garden scoring, and there was over a dozen unnecessary lines of code which could be replaced with a function fullDeckCount, which had a horrid naming convention. There was numerous times that a variable was declared and then only intermittently referenced later in the function. Player hands did not exist on other player turns. Many card implementations had simple bugs such as not adding to counts of using unwieldy data structures in the most inappropriate ways. Also the phase variable had been abandoned by the original developer who had become confused as to

its purpose. This leads to the idea that the game has no implementation in the current state of turn order which is supposed to include Action, Buy, Cleanup. Cleanup is entirely unaddressed and led to many of the oversights in the code.

I actually addressed and fixed a decent number of bugs within the implementation. The one I chose to debug based on a bug report was that player hands did not exist outside their turn. Many cards rely on this for their interaction and not having it made these cards have unintended or nullified affects. I addressed this issue along with some of the coins issues and I made sure every card returned error in proper places and all cards discarded to playedCards after use, though they still never return from this array to the player discard. Many people did not fix any bug beyond the minimum and this made testing for more hidden bugs difficult as the more obvious bugs, such as broken cards came to the forefront quickly during random testing runs.

Coverage and Comparison

Matthew Zakrevsky

The code is slightly more reliable than many I ran across, though all implementations still had massive amounts of unnecessary spacing and improper indentation which made the code difficult to read. Many of the bugs were the same as found within my own implementation, though a majority of them were unaddressed. This code is still very unreliable.

Unit/Card Tests: Lines Executed: 42.17% of 581

> There were over 1300 lines which means almost 45% of the lines brackets or spaces in addition to the fact that this amount of coverage isn't up to par.

Adventurer Card Test:

> Assertion Failure handCount was unexpected value. (due to no discard of Adventurer card itself) I do believe this could have been made easier if I had written my own assert in order to complete the test but with my prior knowledge of the specific bugs I knew exactly what was wrong upon reading the failing test log and every other one.

Random Card Test:

> Assertion Failure state->coins was unexpected value (due to updateCoins bug described earlier)

Random Tester:

> My random tester could not run due to the error in Embargo which was unaddressed in the code which caused infinite loops upon improper choices.

Andrew Gass

Much of the errors were untouched and the cards which were refactored brought strange bugs to the surface that required thought to properly understand. One such bug was where the parameters to the function were swapped causing the choices made to be wrong. There were numerous compiler warnings for unused variables. This code is worse than the original implementation.

Unit/Card Tests: Lines Executed: 41.14% of 581

> Again we have the same issues as with the previous evaluation.

Adventurer Card Test:

Assertion Failure handCount was unexpected value. (due to no discard of Adventurer card itself)

Random Card Test:

Assertion Failure state->coins was unexpected value (due to updateCoins bug described earlier)

Random Tester:

Upon an attempt to run the random tester it merely looped a debug prompt from Feast: “That card is too expensive!” This indicated the error returns had not been set as was done in my implementation.

Recommendation

I recommend an overhaul of the development approach to this Dominion code. An object oriented approach would benefit the game greatly in development, testing and readability. Code coverage was also difficult to achieve due to the disjointed nature of the implementation. Certain functions are almost entirely unused throughout the game. Some functions could be salvaged but the amount of time spent refactoring may outweigh the benefit of such actions and the risk of pulling bugs into the system is high.