Arthur Liou
CS362

Assignment 5

Teammate's Code Being Used:
https://github.com/wangch7/CS362-W2019/tree/wangch7-assignment-2/projects/wangch7/dominion

- Grab one of your teammate's code and put it in a separate folder within your folder. The name of the folder should start with the ONID of the teammate whose code you are testing. For instance, if you chose to test robertwen/dominion, create the directory projects/yourONID/roberwenDominion/.
    - https://github.com/artliou/CS362-W2019/tree/931989226-assignment-5
    - Done - "wangch7Dominion/dominion" - https://github.com/artliou/CS362-W2019/tree/931989226-assignment-5/projects/lioua/wangch7Dominion
- Add your tests to this directory and run your tests (the unit and random tests from assignment-3 and assignment-4) to test the code written or modified by your teammate that you copied. You can use your makefile to run the tests or you can run them one by one. Please note that you "may" need to change your tests in order to match the refactored code, depending on how your teammate refactored the code as far as method names, the parameter passed, etc. This will also help indicate how maintainable your tests are. (25 points)
    - Done / added
- Find and report at least 2 bugs and document these as bug reports in the Assignment-5.pdf file, under a section called Bug-Reports. Describe in detail about the bugs; the cause, how you found them, etc. o Note: In case your teammate already fixed the bugs and you cannot find any, use the version they submitted for assignment 2. (25 points)
    - See Bug Reports section
- Write a test report, in a section called Test Report, describing your experience testing your teammates code. Document in detail, including code coverage information, and your view of the reliability of the Dominion code of your teammate. (25 points)
    - See Test Reports section
- Document the process of identifying and fixing the bugs in your teammates code and describe the code changes under a section called Debugging. Show how you used a debugger (e.g., GDB, or another debugger tool) to understand and debug your teammates code. Submit any files or logs that you generate through the debugging process as part of your submission. (25 points)
    - See Debugging section

Bug Reports

1) In running my cardtest1.c, which tested the Smithy card, I found a bug where the user didn't draw a sufficient amount of cards. They drew 2 instead of 3. The cause for the bug, on L688, is that `for (i = 0; i < 2; i++)` only draws 2 cards instead of 3. During my Debugging section, I fixed this code

Testing output: hand count = 6, expected = 7 && deck count = 3, expected = 2.

2) In running my cardtest2.c, which tested the Adventurer Card, I found a bug where the user didn't receive the coin count expected.
Testing Output: Card Test 2A: Receive at least 2 Coins. Coin count = 4, minimum expected = 6
The cause for the bug is that `for (i = 0; i < 2; i++)` only draws 2 cards instead of 3. During my Debugging section, I fixed this code on L688.
3) In running my cardtest3.c, which tested the Village Card, I found a bug in my test, where I didn't account for a discarded card and only the drawn card.

Testing Output: Card Test 3A: +1 Card. hand count = 5, expected = 6

4) In running my cardtest4.c, which tested the Remodel Card, I found a bug where the user did not discard the "Remodel"ed card and so had an extra card in hand. During my Debugging section, I fixed this code on L786.

Testing Output: Card Test 4A: Hand Count -2. hand count = 4, expected = 3

5) Separately, outside of my tests, I saw that countcil_roomRefactor on L703 was incorrect. It was drawing 6 cards instead of the correct 4.

Test Report

I migrated my makefile commands for running my unit, card, and random tests. I left the unit/card and random tests make commands separate, as the random tests tested thousands of scenarios, and it would be easier to check separate the visualization of the two outputs. I also grabbed my tests from HW3 and HW4 and placed them in the "wangch7Dominion/dominion" folder I created according to the instructions.

From there, I ran my tests on Chu's (my teammate) code. After running the tests and securing the two .out files, I took a look at the code at Chu had refactored from HW2. She had 5 refactored functions for: Adventurer, Smithy, Council Room, Mine, Remodel. From there, I went on to identifying and then fixing the bugs in Chu's code (see Debugging section)

Code Coverage:

For my unit/cards tests, they covered 29.05% of the lines from dominion.c and around 36.69% for branch coverage. A strange thing I noticed was that the two percentages for lines and branch coverage for dominion.c were the same for all 8 tests, even though each test tested different cards.

For my random tests, they did cover 100% of their respective cards, but only branch coverage was only 37% of the dominion.c code

Reliability: Beyond the introduced errors in Chu's code, her dominon.c and the resulting functions/code seem to be reliable.

Debugging

Process of Identification: When running my tests as noted above, I looked for the specific outputs and error messages my tests returned. In addition, I took a closer look at the code that was refactored and compared the refactor to my knowledge of Dominion.

Fixing the Bugs: Once I identified the bugs, I went back to the function and/or block of code that contained the bug and fixed it. I did this for the bugs mentioned above. From there, I also reviewed the logic after my fixes to make sure the cards / functions now worked as intended.

Debugger / Logs / Submission: The two .out files will be attached to this HW submission. I also attached my launch.json I used from VSCode when debugging dominion. From running

these three test sources & my knowledge of how my tests were constructed & my now expert knowledge of dominion, I was able to understand and debug my teammate's code. The key ways I was able to review and identify the bugs have been illustrated above.