Final Project Report

CS562 Applied Software Engineering

- Introduction

  In this project I use **QuickCheck** to test my dominion implementation and compare for my random test write by assignment 1.

  QuickCheck is a test tool originally for Haskell, it also have re- implementations version for other language such as C, C++, Java also Python. In this project I will use the Python version[2].

  In my directory */huangca/final/rand/* is my random test from assignment 1; */huangca/final/quick_check* is test program use **QuickCheck**.

- About Quick Check

  QuickCheck is automatic testing tool that help programmer do the testing more easily and more efficient. It is suitable test for each function, to test function definition.

  QuickCheck generate test case automatically basis on random testing. The idea of **QuickCheck** is " Random testing is most effective when the distribution of test data follows that of actual data ."[1] So the tester can not only control the distribution of test cases but also ensure that they satisfy arbitrarily complex invariants. As same as other test tool, **QuckCheck** can determine the test pass or not, the report of test server as checkable documentation.

  QuickCheck should be lightweight. It let **QuikcCheck** have good compatibility and non-embedability.

  The python version of **QuickCheck** is beta version, it completely work fine, but some important feature of **QuickCheck** still not implemented. It limit my test case design, so some of my test case still look like traditional way. And also still some known problem in this Python version such as memory management.

- Start to Test Case

  The Python version **QuickCheck** provide *pytest.mark.randomize* function for generating random test data and some options to restrict the test data. The syntax

will be

```
@pytest.mark.randomize(argname=type, **options)
```

For example, I want to write a test case for *buyCard*() function, I already the test of card choice in this range, I can write it like

```
@pytest.mark.randomize(c=int, min_num=7,max_num=16)
```

Python QuickCheck also provide *pytest.mark.parametrize* function to restrict some parameter. For example we know dominion is 2-4 players game, in some case I want to keep the number of player in reasonable range, so I can write

```
@pytest.mark.parametrize("p",[2,3,4])
```

This Python QuickCheck library provide *list_of* data structure to generating collections. For example in dominion game, kingdom card list is construct by 10 selected card, it could be write like

```
@pytest.mark.randomize(l=list_of(int, items=10),min_num=7,
max_num=16)
```

```
@pytest.mark.parametrize("x",[2,3,4])
@pytest.mark.parametrize("s",[2,3,4,5,6,7,8])
@pytest.mark.randomize(l=list_of(int, items=10),min_num=7, max_num=16)
def test_initialzeGame(x,l,s):
    g=d.initializeGame(x,l,s)
    if len(l)==len(set(l)):
        assert g==-1
```

Figure 1

Figure 1 is my part of my *initialzeGame*() function test case.

Test Result Review and Comparison

- **Bug report**:

  Most of bug I already found in assignment2 and some of bug is I leave them in my implementation intentional at assignment 1.Those bug I do not repeat here. However still have a few bug I did not found and surprised me. I pick one show it below.

```
state = <dominion_p.gameState instance at 0x02F67620>

    def endTurn(state):
>       state.discard[state.whoseTurn].extend(state.hand[state.whoseTurn])
E    KeyError: 3

dominion_p.py:214: KeyError
```

Figure 2

Explanation: KeyError means there are no definition when index equal 3. So I review the test, I find I did not pay attention. In this test the number of players is random 3, so the index of player should 0 to 2, but I forget to do the -1 in the *endTurn*() function. I did not found this bug at former assignments because in the first assignment I focus on correctly of function, I assert card in hand already move to discard deck, and *whoseTurn* attribute is change, but I miss the base thing.

- **Comparison**

  I will compare the code coverage and the performance.

  For code coverage

```
Name             Stmts   Miss  Cover   Missing
------------------------------------------------
buyCard_rand        18      0   100%
dominion_p         320     94    71%   59, 64, 69, 133, 135-137, 157, 159, 161, 171-176, 216, 218, 220, 227,
231-248, 253, 255, 257, 281-286, 292-296, 301-307, 313, 321-327, 333, 336, 345, 347-349, 351-353, 356-371, 377-
383, 396, 421
drawCard_rand       26      0   100%
getWinners_rand     31      0   100%
isGameOver_rand     14      0   100%
playCard_rand      229    104    55%   59-85, 117-132, 181-246
rand_test           20      0   100%
------------------------------------------------
TOTAL              658    198    70%
```

Figure 3

```
--------------- coverage: platform win32, python 2.7.9-final-0 ----------------
Name          Stmts   Miss  Cover
---------------------------------
dominion_p      319     38    88%
```

Figure 4

Figure 3 is the code coverage of use random test I done at assignment 1, we can see the coverage of dominion_p is 71%. Figure 4 is the code coverage use **QuickCheck** library we can see the coverage of dominion_p is 88%. The code coverage is increase 17%.

For performance, my own random test run 2000 test cases just spend 0.7 seconds, but the test use QuickCheck library run 1500 test cases use 3.2 seconds. So performance of QuickCheck is not so good.

Analyze: In my opinion, QuickCheck can get better code coverage is it benefit by QuickCheck suitable test case for each function and *pytest.mark.parametrize* function. This function actually is exhaustion, each value determine must be selected at least once. It is good for code coverage but it is bad for performance. Another reason of performance is, QuickCheck also is a plug of Python tool call *pytest*, it means when I run the test program it will run the *pytest* first.

- ## Feed Back for QuickCheck Library

  The Python **QuickCheck** library is amazing tool for testing, after all it still a beta version(version 0.8). So it still have some part unsatisfactory and some feature of **QuickCheck** still not implemented.

  Originally **QuickCheck** for Haskell have good data system. It has User-Defined Data Types system. But in the Python version that I use, this feature still on construct, it will limit test case writing. For example, we have two INT variable i1 and i2. I want to random i1 range 1 to 10, random i2 11 to 20. In **QuickCheck** should be two data type "i1 range 1to 10" and "i2 range 11 to 20". But in the Python version, do not have the User-Defined Data Types System, once I range INT 1 to 10, all INT variable will be restrict in 1 to 10. It provide an alternative solution use *pytest.mark.parametrize* function, but as I mention before this function actually is exhaustion, not a random function. So in some test case I still need to use the Python built in random function

  Memory management also is a unsatisfactory part.

```
# @pytest.mark.parametrize("x",[2,3,4])
# @pytest.mark.randomize(("c1", "int"),("c2", "int"),("c3", "int"),("c4", "int"),("c5", "int"),("c6",
"int"),("c7", "int"),("c8", "int"),("c9", "int"),("c10", "int"),("c11", "int"),min_num=0, max_num=10)
# def test_isGameOver(x,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11):
```

Figure 5

  Figure 5 is a piece of code I commend. It is test case for isGameOver() function, I define 11 random arguments represent 10 kingdom cards and province card. I commend this code because when I run this code, **QuickCheck** will report memory error. Through my testing, 10 random arguments is the maximum of number of arguments. I think that is not enough for testing.

- ## Conclusion

  In this project I use Python version QuickCheck library write a new test program to test my dominion implementation. This new test program has better code coverage but run a little bit slowly. Python QuickCheck is a good test tool that provide random test support to tester, but this Python version still have some problem.

References

1.Hughes,John.,Claessen,Koen.:QuickCheck:A Lightweight Tool for Random Testing of Haskell Programs. ACM SIGPLAN Notices. Volume 35 Issue 9. 268 - 279. ACM(2000)

2. https://bitbucket.org/pytest-dev/pytest-quickcheck