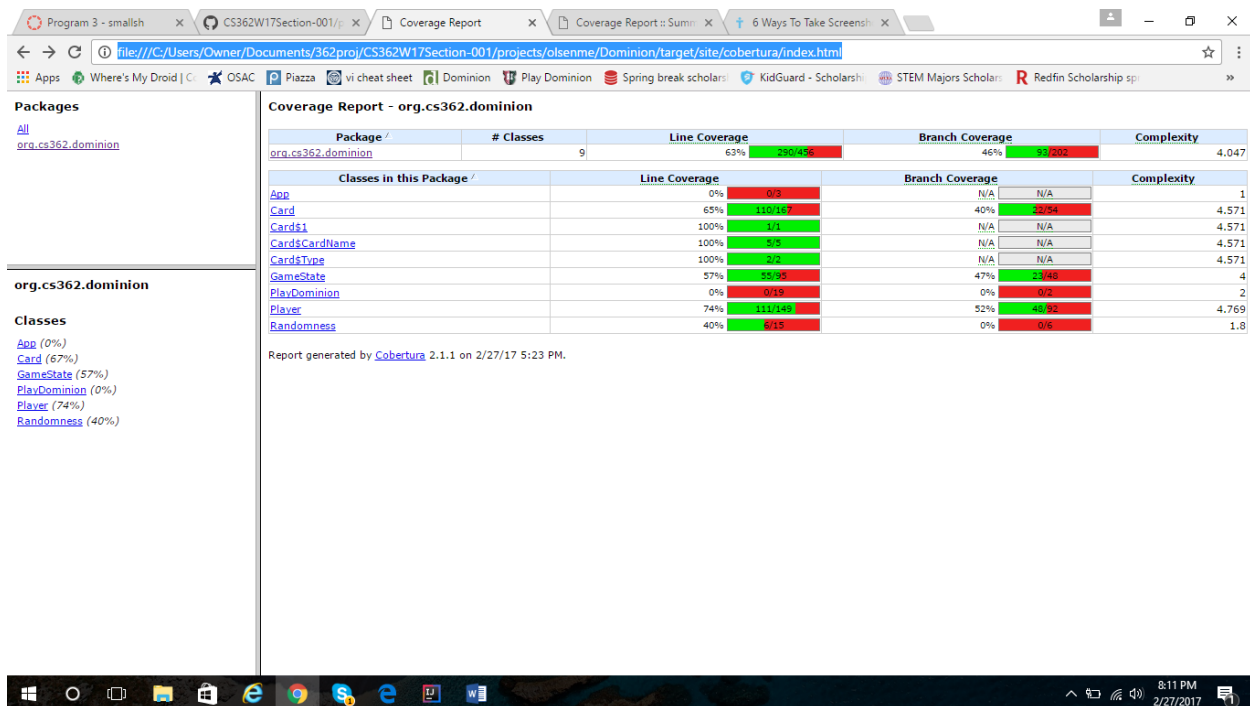


Meagan Olsen

Assignment2

i. Evosuite Tool

1. This tool covered the Dominion code relatively well a 63% line coverage over the entire package. This number varies significantly from the IntelliJ code coverage tool that I used in assignment 1, which had a little under 40 percent line coverage. This difference can be explained by the fact that, in assignment 1, we used manual unit testing, and so I was not able to generate as many tests as this tool did for me, resulting in a lower overall code coverage. Whereas in assignment 1, I was only able to generate successful unit tests for just the implementation of the 13 kingdom cards, this tool allowed me to generate unit tests for all classes in the Dominion code. As this graph shows, even my card class line coverage is significantly higher at a 65 percent line coverage in comparison to my line coverage for this class in Assignment 1, which was a little under 50 percent for this class in particular.



2. A couple of my bugs were related to the rules of the game, so this tool did not help catch those bugs such as two players being able to score the same amount of points or the feast card adding coins instead of cards. Since this tool simply generates tests for the code that is written, and these bugs do not cause any sort of interruption in the flow of the game, these tests, like mine, did not catch these bugs. In terms of my problem with the play() method in the GameState class, the tool did not help because I did not run tests for my PlayDominion class. Out of my 100 tests run, I only had one failure. This was in the randomness test. In terms of catching my bugs, I would have to say that this didn't do the greatest job. My tests did a better job of catching my known bugs.

3. I had trouble integrating this tool with Maven. Even after my tests were fully generated, I was still not able to get my tests to run successfully using the maven tool. I was constantly running into an issue with the version of the plugins that maven needed to run my code. In terms of generating my tests, I had problems with finding documentation that worked for configuring my maven pom file for the project.

4. I found that this tool is not very compatible with Maven. I thought that the results of this tool were decent for the amount of tests that were automatically produced. The unit tests that it generated were somewhat similar to my unit tests, except that I was able to gear my own unit tests a little more towards catching my bugs because I knew about them. In terms of time budget, I thought this tool took a long time to configure although it did produce decent and unbiased results.

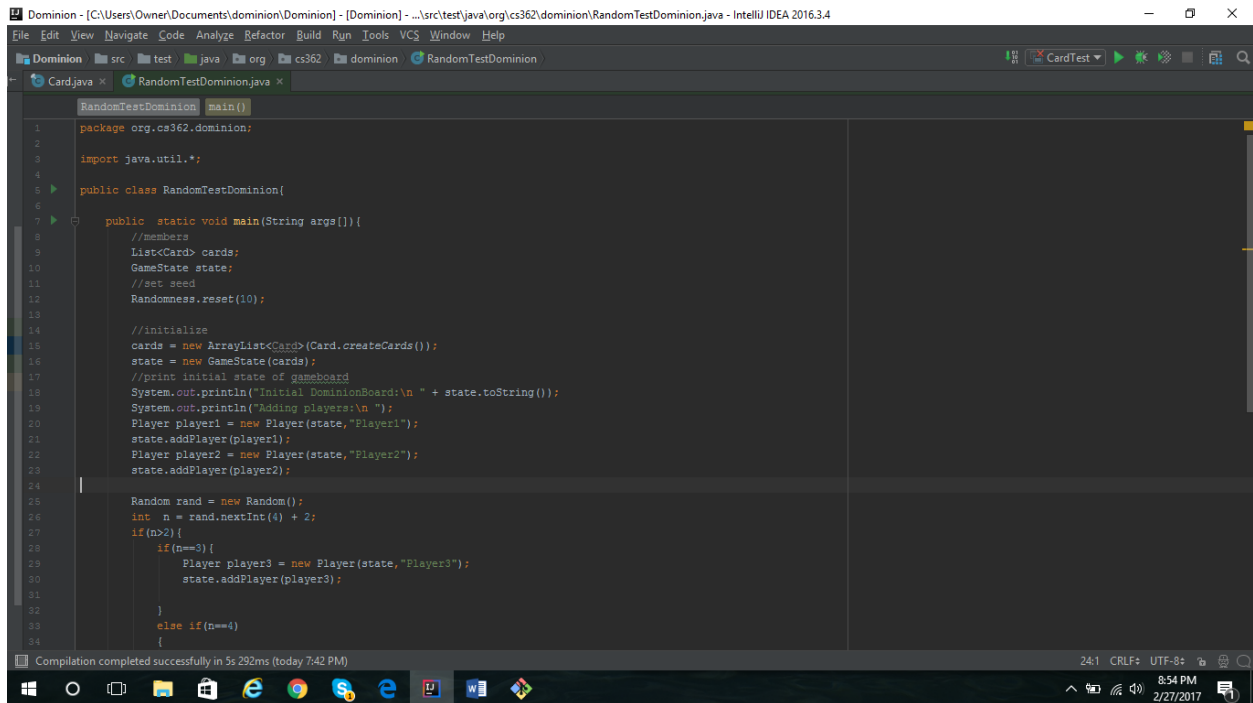
5. To be honest, I didn't look at many other tools before selecting evosuite. A classmate suggested PiTest, but then I found out that that was not a random tester. I was initially going to use Randoop, but decided to go with Evosuite for consistency with the group that I was working with.

6. This tool is useful because it creates a large variety of tests in a short amount of time. In the 12 minutes that it took to generate my tests, Maven showed that it attempted to run 100 tests total. It had good breadth in that it covered all parts of my code much more quickly than I could have done if I had wrote my own tests by hand. It was valuable in that, not knowing my bugs, made the tests themselves less biased than the tests that I created.

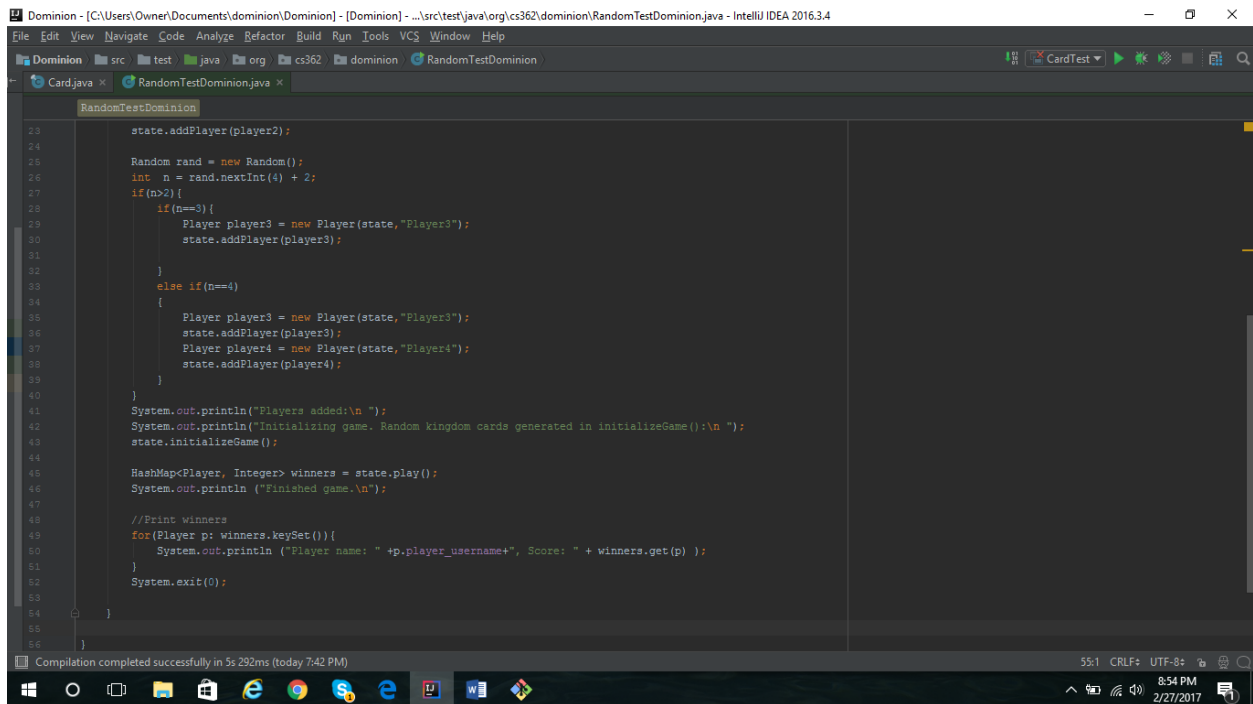
This tool is open source, which has both negative and positive aspects to it. Because it is open source, for example, I was able to research and find a wide variety of answers to problems that I ran into on common sites like stakoverflow. A negative aspect was that the tests that it generated had some buggy code in them that were unique to evosuite. And I had to remove large parts of the code manually to get it to run.

In terms of the amount of tests generated for the short amount of time it took, I think this would be a valuable tool for many industry projects although which projects I would use it on would be very specific. For example, I don't know if I would use this tool alone on a very delicate system because it is very easy for the tests to misinterpret meaning in the code because it doesn't have that intelligence. Since this tool is open source, I think it has good future prospects although it needs to be more compatible other common software tools like Maven. However, being that it is relatively new, I think it has room for development.

iii. Random tester is posted under Dominion folder in test folder. It is named RandomTestDominion.Java as specified although I have included snapshots here as well.



```
1 package org.cs362.dominion;
2
3 import java.util.*;
4
5 public class RandomTestDominion{
6
7     public static void main(String args[]){
8         //members
9         List<Card> cards;
10        GameState state;
11        //set seed
12        Randomness.reset(10);
13
14        //initialize
15        cards = new ArrayList<Card>(Card.createCards());
16        state = new GameState(cards);
17        //print initial state of gameboard
18        System.out.println("Initial DominionBoard:\n" + state.toString());
19        System.out.println("Adding players:\n ");
20        Player player1 = new Player(state,"Player1");
21        state.addPlayer(player1);
22        Player player2 = new Player(state,"Player2");
23        state.addPlayer(player2);
24
25        Random rand = new Random();
26        int n = rand.nextInt(4) + 2;
27        if(n>2){
28            if(n==3){
29                Player player3 = new Player(state,"Player3");
30                state.addPlayer(player3);
31            }
32            else if(n==4)
33            {
34
```



```
35                Player player3 = new Player(state,"Player3");
36                state.addPlayer(player3);
37                Player player4 = new Player(state,"Player4");
38                state.addPlayer(player4);
39            }
40        }
41        System.out.println("Players added:\n ");
42        System.out.println("Initializing game. Random kingdom cards generated in initializeGame():\n ");
43        state.initializeGame();
44
45        HashMap<Player, Integer> winners = state.play();
46        System.out.println ("Finished game.\n");
47
48        //Print winners
49        for(Player p: winners.keySet()){
50            System.out.println ("Player name: " +p.player_username+" , Score: " + winners.get(p) );
51        }
52        System.exit(0);
53    }
54 }
55
56 }
```

iii. I would identify a bug by first creating a source file which utilizes every top level API necessary to functionality of my application, in this case Dominion, attempting to reveal which parts exactly of my code do prove to be functioning correctly while also attempting to incorporate non top level code as well. One of my bugs that I documented in Assignment 1 were actually found through the process of using a debugger. When I ran my application and found out that the output was not as it should have been, I was able to go back through and examine the values of variables through a debugger, which also

helped me to narrow down where the actual bug was. I found that one of my initial bugs was a simple error in which I was iterating through a loop more times than I thought.