

Introduction:

I used Eclipse and maven through terminal for this project. For most of the coverage I used EclEmma, a coverage tool easily imbedded in Eclipse that implements Jacoco. For composite and final coverage I used Cobertura, through Maven. For all of the coverage the columns, from left to right, are class name, instruction/line coverage, and branch coverage. Classes that are not relevant are omitted except in the case of Cobertura and main Player, which should be ignored. I apologize for any confusion.

Search Based Tool:

For this assignment I used Evosuite, one of the mentioned free tools. I used Evosuite because I understand search based testing the most, and it seemed the most interesting. Most of the problems that I faced in implementing Evosuite involved working with the syntax. If I were to suggest changes I would suggest that it relies less on specialized libraries. If it instead built off of junit or existing java.util libraries. I'm not sure how possible that is, but for Evosuite to work I needed to change some of the framework files, and of course I had some issues with Maven.
















From Evosuite's imbedded coverage measurement I got coverage of 51%.

```
* EvoSuite 1.0.3
Total number of classes in the project: 7
Number of classes in the project that are testable: 7
Number of generated test suites: 4
Overall coverage: 0.51
```

Here are my results when ran alongside my unit tests, using Cobertura:

Player	97%	<div><div></div></div> 192/197	93%	<div><div></div></div> 93/100
Card	94%	<div><div></div></div> 184/194	86%	<div><div></div></div> 64/74
GameState	91%	<div><div></div></div> 190/208	87%	<div><div></div></div> 141/161
mainPlayer	86%	<div><div></div></div> 13/15	100%	<div><div></div></div> 2/2
Randomness	81%	<div><div></div></div> 13/16	100%	<div><div></div></div> 6/6

The coverage has improved from just unit tests. My coverage with EclEmma, just unit tests:

 Card	 76%	 72%
 GameState	 72%	 66%
 Player	 72%	 79%
 PlayDominion	 0%	 0%
 Randomness	 33%	 0%

The improvements are clear, though it may be skewed as I used multiple coverage tools. A clearer table below:

Class Name	Branch Coverage Unit	Branch Coverage Unit & SBST
Card	72%	86%
GameState	66%	87%
Player	79%	93%
Randomness	0%	100%

The table shows that all classes tracked were improved with SBST, and in the case of the randomness class it improved from 0% to 100%. This is because I didn't test randomness in my unit tests, but Evosuite was able to come up with tests.

From Fraser and Arcuri, Evosuite uses evolution search based testing with mutation methods for evolution. Unlike other evolution search based test generators which generate individual tests, Evosuite generates test suites, making it more effective for automated analysis.

As the generated tests didn't have the coverage of my unit tests, I'd say that for a small program like Dominion automated testing might be superfluous. If I were testing a larger program or system automated testing would be a definite tool, even if I used it alongside unit tests. Overall I had no complaints with using Evosuite, other than setup difficulties mentioned earlier. The time it took to generate my tests was always less than 15 minutes, using 2 cores. As a tool I think that Evosuite was very effective, and now that I know how to implement it I will be likely to use it again.

Random Dominion Test:

I didn't find it too hard to write the random tester. I decided to use a random seed so as to increase the variation. I also ran 40 games, varying from 2-4 players, thinking that this might increase my coverage.

Here is the coverage output, with EcEmma and 10 games, with seed of 10.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
Card		75%		71%
GameState		72%		66%
Player		73%		81%
PlayDominion		0%		0%
Randomness		33%		0%

This is the coverage output, with EclEmma and 40 games, with random seed.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
GameState		69%		65%
Card		75%		62%
Player		74%		88%
PlayDominion		0%		0%
Randomness		33%		0%

This coverage output is with EclEmma with 100 games, and a constant seed.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
Card		76%		72%
GameState		72%		66%
Player		72%		79%
PlayDominion		0%		0%
Randomness		33%		0%

I realized that changing the seed for repeat calls has very little effect, and my actually harm randomization, which would effect the coverage. There is also an upper limit to the coverage, as a few functions are not called during normal functioning, like toString and clone functions.

In my random dominion game I didn't use any asserts to check game states, instead I have it printing out the results and I haven't seen any errors except the one mentioned below.

Bugs Found:

A bug that I found was after implementing the multiple players, I realized through static analysis that the available cards were set to 13, so I could test all the cards without restarting the GameState. This turned problematic as the rules of Dominion require a random sample of 10 cards for each play through. I found this bug by running the RandomTestDominion, and receiving error where the program tried to print or manipulate a card that wasn't in the GameState. My solution was to just select the 10 random cards, but the result was failure in many of my unit tests, as well as the generated tests. After further analysis, I solved the problem by adding a integer parameter to the initialize game function, allowing me to explicitly set the amount of cards, 10 for real games but the full 13 for testing.

Works Cited

Fraser, Gordon, and Andrea Arcuri. "[EvoSuite](#)." Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11 (2011): n. pag. Web. 27 Feb. 2017.