Tanner Cecchetti
CS 362

Tarantula

Testing with Statistics

For part of my final project, I chose to create a Tarantula tester (`tarantula.py`). This is the basic usage:

```
$ cd tarantula
$ python tarantula.py <optional: num tests> <optional: random seed>
```

Executing these commands will generate the following:

```
./tests-passed : a directory containing all gcov output of passing test runs
./tests-failed : a directory containing all gcov output of failing test runs
./tarantula-report.txt : a file containing the suspicion of each line in dominion.c
```

The report looks something like the following (depending on your input, of course):

```
====================================================================
======================== TARANTULA REPORT ==========================
====================================================================
============First column represents a graph of suspicion============
=======(0-10 # for percent suspicion, dashes for not executed)======
====================================================================
- - - - -  ln 17 never executed
#####      ln 45 executed when test failed 34/100. suspicion 0.5000.
########## ln 741 executed when test failed 34/34. suspicion 1.0000.
```

Of course that's only a tiny sample of the output, but that should give the basic idea of what my program does. It creates a "graph" (if you can even call it that) of the suspicion, as well as listing its numerical value. It also recognizes when code is never executed, and displays dashes next to it.

My program works by running `testdominion` with random seeds and checking the exit value. If the exit value is non-zero (i.e. something went wrong), then we move the gcov output of dominion.c to the `tests-failed` folder and continue to the next text. Conversely, if it passes, we again move the gcov output, but this time it goes to the `tests-passed` folder.

One major limitation of this program is that when a `testdominion` run enters an infinite loop and "times out," it gets sent `SIGKILL`. This interferes with gcov's data collection, and thus, it can't be used to determine which lines are at fault. This essentially means my program is not helpful in catching errors that lead to infinite loops. Since that's the primary weakness of `dominion.c`, it makes it extremely difficult to actually catch bugs using this method.

To combat this, I introduced a “bug” into dominion.c on my own, allowing me to test my Tarantula implementation more easily. Very simply, I just added an exit statement within one of the card effects:

```
int cardEffectSalvager(int handPos, int currentPlayer, int choice1, struct gameState *state)
{
    //+1 buy
    state->numBuys++;

    if (choice1)
    {
        //gain coins equal to trashed card
        state->coins = state->coins + getCost( handCard(choice1, state) );
        //trash card
        discardCard(choice1, currentPlayer, state, 1);

        // introduce an error
        exit(-1);
    }
```

This error is easily identified with my Tarantula implementation using only 100 test runs. The `exit` statement as well as the couple lines before it have suspicion 1.0:

```
########## ln 739 executed when test failed 34/34. suspicion 1.0000.
########## ln 741 executed when test failed 34/34. suspicion 1.0000.
########## ln 744 executed when test failed 34/34. suspicion 1.0000.
```

Additionally, the line responsible for executing this function (i.e. `cardEffectSalvager`) is also marked as highly suspicious:

```
########## ln 1219 executed when test failed 34/35. suspicion 0.9851.
```

Note that on this line, however, it has a slightly lower suspicion because the exit is only called if choice1 has the correct value. Neat! :D

It’s also worth noting that my program ignores all lines which gcov marks with a dash. These are lines that can’t be executed, and so they really aren’t worth paying attention to. If a line is reported as “not executed,” that means it’s dead code.