Frederick Kontur (konturf@oregonstate.edu)
CS 362, Tarantula Extra Credit

Isolating Bug Using Tarantula

As suggested in one of the lectures, I wrote my tarantula code using Python. This was a good learning experience since I have only used Python once before, so I was able to practice my skills with this important language.

Using the Python tarantula code, I ran my random number tester for my Mine card. I had to make the following alterations to my random number tester to make it compatible with tarantula:

- Only have it return "Correct Result" or "Incorrect Result" for the test, rather than a description of the test. This is so that it is easier for tarantula to identify which tests found bugs and which did not.
- Rather than have the iterations of the random tester take place in the tester, set up the random tester to only do one iteration. The tarantula code itself controls how many iterations are done by iteratively calling the random tester and supplying a different seed each time.

With that set-up, I had the tarantula code do 2,000 iterations of the random tester, analyzing the results of the gcov output to see which lines were run for each of the tests. It also read the output of the test to identify whether it found a bug or not. I was surprised at how long it took tarantula to run. In retrospect, this made sense because it was essentially running the tester and creating and scanning the results of the gcov output file each time.

I had tarantula determine the suspiciousness of each of the lines in my dominion.c code based on the formulas in this paper. I then analyzed the results in several ways. At first, I had the code output some user-chosen number of the most suspicious lines. I commented out that portion of my code because it was not as useful as I was hoping when I could not see where in the code the suspicious lines occurred. What I found more useful was to output the suspiciousness score of each line along with the line text itself to an Excel file. I then used conditional formatting to color code based on the suspiciousness score. A PDF copy of that Excel file can be found in my Extra Credit folder with the filename TarantRes.pdf. The color coding is as follows:

- Pink is for suspiciousness of 0.95 or greater
- Orange is for suspiciousness greater than 0.80 and less than 0.95
- Yellow is for suspiciousness greater than 0.50 and less than 0.80
- Green is for suspiciousness greater than 0 and less than 0.50
- White is for lines that were not covered, which are denoted by a suspiciousness score of -1

From previous tests, I knew one of the bugs for Mine was that players could choose non-Treasure cards from the supply. I found evidence for this in the getCost() function, where many of the non-Treasure cards had high suspiciousness scores. Based on the mineEffect() function color-coding, it seemed that if a player gained a card, it would result in a bug. This is a somewhat misleading result of the random

tester, where there was no effort to prefer treasure cards to be chosen from the supply, so the vast majority of the tests had players choosing invalid non-Treasure cards. The task is also made more difficult because the bug in this case is the absence of code to check if the player's desired card is a non-Treasure card, as opposed to an already-existing line with an error. I did this on purpose to see how effective Tarantula would be for this situation which would presumably be more difficult for debugging.

However, despite the difficulty, it is fairly easy to make the following conclusions about the bug that is occurring for the Mine card:

- It occurs when a player tries to gain a card
- It seems to preferentially involve non-Treasure cards being chosen

Based on that, it is fairly straightforward to conclude that Mine is allowing players to illegally gain a non-Treasure card.

One other error that was identified by Tarantula was when a player tried to choose a card that was not in the supply. Since the gainCard() function does not allow the card to be chosen in that case, I suspect the error is in the fact that mineEffect does not alert the calling function that an invalid card that was not in the supply was chosen by returning -1.