GoatGoatCar

Xinyue Guo

Julian Monticelli

Jason Naughton

CS 1632

Software Quality Assurance

Spring 2017

Deliverable 1:

Test Plan and Traceability Matrix

Introduction

Our first challenge was in understanding the actual underlying Monty Hall problem. The mechanism of the Monty Hall problem is, of course, unintuitive, and it was unfamiliar to us. We ran the program and found a much higher success rate at Switch. We wondered: was this a flaw in the program or a result intrinsic to the problem itself? After deliberating over the problem, we reached a sufficient understanding. In retrospect, this understanding wasn’t completely necessary to the nature of our testing, but we did find it to be helpful and worth the effort.

We encountered some difficulty with the ambiguity of the requirements. When considering possible inputs outside the “happy path” we noticed some potential inputs (and their expected results were not accommodated in the requirements. For example, should the program accept “0” and “1” as the first and second command line arguments, respectively? Though they are seemingly integer values, they can be stringified without much confusion. We decided that, yes, this was appropriate. What about: should the system accept two of the same arguments for the first and second arguments, i.e. “car” and “car”? We decided, no, this would not be appropriate, because this makes the whole problem nonsensical. We reached consensus on these decisions amongst ourselves, but they ought to be clarified with the rest of the program team. After we had agreed on these clarities, we wrote the related test cases to make sure the program complied.

We found that even simply considering edge cases encouraged us to consider this kind of potential ambiguity. During our process, we employed two steps for writing an edge test case. First, we had to decide whether there reasonably could be an edge value or a corner case (or multiple of such conditions) for each given requirement. Second, if so, we had to find it.

We did this by considering kinds of conceptual boxes of potential values, and considering the transitions between these. For example, we wrote a test for an edge case, TEST-ITERATION-DISTIBUTION-EDGE-CASE. We decided there was a possibility for an edge case because of the proportion distribution of the iterations: what if the number of iterations is not proportional to the number of threads? This question leads to the edge test case which runs the program with k\*n+(n-1) iterations and n threads. Because k\*n+(n-1) from the perspective of integers is the nearest to (k+1)\*n. Of course, (k+1)\*n+1 is another edge case in this situation, but we omitted this case here since the two cases are basically testing with the same idea. Another edge case, for example, is TEST-SMALL-NUM-WITH-EDGE-CASE for the FUN-SMALL-NUM. Again, we considered a sort-of conceptual box of the possible values, those less than 100, and another with those greater than 100. We wondered what should happen on the transition border between the two, namely on the value 100 itself. According to the requirement, the border value 100 should not issue a warning. So we wrote the corresponding test case, and we actually found a defect.

Test Plan

(1.)

IDENTIFIER: TEST-BASIC-CALC

TEST CASE: This test case will verify that the calculation of the Monty-Hall program is accurately being tested for. The values should approach one-third and two-thirds probability for staying and switching, respectively, for sufficiently large values.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “Car Goat 10000 4”.

POSTCONDITIONS: The program displays the chances of getting both the good and bad options had you switched or had you stayed, and the values are correctly displaying roughly one-third probability for staying and two-thirds probability for switching.

(2.)

IDENTIFIER: TEST-ITERATION-DISTRIBUTION

TEST CASE: This test case aims to confirm that the program allocates

iterations evenly per thread when the number of

iterations is proportional to the number of threads.

PRECONDITIONS: Command prompt window is opened and the current directory is

the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with the arguments “car goat 1000 5”

2. Run the program with the arguments “car goat 2000 5”

3. Run the program with the arguments “car goat 3000 5”

POSTCONDITIONS: Every time after each step, the program displays the chances of getting both the good and bad options had the user switched or had the user stayed in current iterations. After step 1, the program displays each thread did 200 iterations. After step 2, the program displays each thread did 400 iterations. After step 3, the program displays each thread did 600 iterations.

(3.)

IDENTIFIER: TEST-ITERATION-DISTIBUTION-EDGE-CASE-1

TEST CASE: This edge case test case aims to confirm that, when the number of iterations is not proportional to the number of threads, the number of iterations distributed per thread never has a difference of more than one compared to any other thread. Here we test with an edge case where, when the number of threads is n, the number of iterations is k\*n+(n-1) with k is a positive number.

PRECONDITIONS: Command prompt window is opened and the current directory is

the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with the argument “car goat 1004 5”

2. Run the program with the argument “car goat 1009 n”

3. Run the program with the argument “car goat 2004 n”

POSTCONDITIONS: Every time after each step, the program displays the chances of getting both the good and bad options had the user switched or had the user stayed in iterations. After step 1, the program displays the first 4 threads did 201 iterations, and the 5th thread did 200 iterations. After step 2, the program displays the first 4 threads did 202 iterations, and the 5th thread did 201 iterations. After step 3, the program displays the first 4 threads did 401 iterations, and the 5th thread did 400 iterations.

(4.)

IDENTIFIER: TEST-ITERATION-DISTIBUTION-WITH-EDGE-CASE-2

TEST CASE: This edge case test case aims to confirm that, when the number of processes exceeds the number of iterations, the number of iterations distributed per thread still never has a difference of more than one compared to any other thread.

PRECONDITIONS: Command prompt window is opened and the current directory is

the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat 101 102”.

POSTCONDITIONS: The program should prompt the user: “You have provided more

threads than iterations. There will be unused threads.

Continue? [y/n]”, and wait for input.

(5.)

IDENTIFIER: TEST-DISPLAY-RESULTS

TEST CASE: This test case will attempt to verify that the results are displayed accurately (to three decimal places) when given valid arguments.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat 1000 4”

POSTCONDITIONS: The program displays percentages of success and failure for each option, stay and switch, where the option is printed out and the percentages are precise to three decimal places.

(6.)

IDENTIFIER: TEST-ARGS-NUMBERS-WITH-MORE-THAN-REQUIRED

TEST CASE: This test case will attempt to verify that our program successfully ensures that we have no more than four arguments.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with 10 different arguments, “0 1 2 3 4 5 6 7 8 9 10”.

POSTCONDITIONS: The program issues a warning that states the four arguments it needs before exiting.

(7.)

IDENTIFIER: TEST-ARGS-NUMBERS-WITH-LESS-THAN-REQUIRED

TEST CASE: This edge case test case will attempt to verify that our program successfully ensures that we have exactly no less than four arguments.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with 3 arguments, “0 1 2"

POSTCONDITIONS: The program issues a warning that states the four arguments it needs before exiting.

(8.)

IDENTIFIER: TEST-ARGS-INVALID-IN-INTEGER-ARGS

TEST CASE: This test case will attempt to verify that the program tells the user if his or her input of integer parameter is valid.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat -1000 4”

POSTCONDITIONS: The program displays warnings telling the user that the provided input is not valid, that the number of iterations needs to be an integer great than 0.

(9.)

IDENTIFIER: TEST-ARGS-INVALID-IN-STRING-ARGS

TEST CASE: This test case will attempt to verify that the program warns the user that his or her input is invalid when the same argument is given as both the first and second value.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car car 1000 4”

POSTCONDITIONS: The program displays warnings telling the user that the provided input is not valid, that the bad and good value cannot be the same.

(10.)

IDENTIFIER: TEST-SMALL-NUM-BELOW-THE-BORDER

TEST CASE: This test case aims to test if the program will issue proper warning when the number of iterations the user choose is less than the recommended minimum, which is 100.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat 80 5”

POSTCONDITIONS: The program issues a warning saying “Recommended minimum number of times is 100. Continue?[y/n]” and waits for the feedback from the user before proceeding.

(11.)

IDENTIFIER: TEST-SMALL-NUM-ABOVE-THE-BORDER

TEST CASE: This edge case test case aims to confirm that the program should execute properly when exceeding the integer limit.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat 10000000000 100”

POSTCONDITIONS: The program displays the chances of getting both the good and bad options had the user switched or had the user stayed in 10000000000 iterations, with no warning about iteration size.

(12.)

IDENTIFIER: TEST-SMALL-NUM-WITH-EDGE-CASE

TEST CASE: This edge case test case aims to confirm that the program should not issue any warning when the number of iterations the user choose is no less than the recommended minimum, which is 100. Here we test with the edge value 100, which is the first integer that is no less than 100.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: Run the program with the arguments “car goat 100 5”

POSTCONDITIONS: The program displays the chances of getting both the good and bad options had the user switched or had the user stayed in 100 iterations, with no warning about iteration size.

(13.)

IDENTIFIER: TEST-SMALL-NUM-CONT-CASE-INSENSITIVE-YES

TEST CASE: This test case aims to test that, when the program is taking further instruction from the user to deal with the warning on minimum number of iterations, as long as the response entered is ‘y’, regardless of its case, the program should give the calculated output.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with the arguments “car goat 80 5” and wait until the warning for minimum iterations.

2. Enter “Y” and press “Enter”.

3. Run the program again with the argument “car goat 80 5” and wait until the warning for minimum iterations.

4. Enter “y” and press “Enter”

POSTCONDITIONS: The program displays the chances of getting both the good and bad options had the user switched or had the user stayed in 80 iterations after both Step 2 and 4.

(14.)

IDENTIFIER: TEST-SMALL-NUM-CONT-CASE-INSENSITIVE-NO

TEST CASE: This test case aims to test that, when the program is taking further instruction from the user to deal with the warning on minimum number of iterations, as long as the response entered is ‘n’, regardless of its case, the program should exit immediately.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with the arguments “car goat 80 5” and wait until the warning for minimum iterations.

2. Enter “N” and press “Enter”.

3. Run the program with the argument “car goat 80 5” and wait until the warning for minimum iterations.

4. Enter “n” and press “Enter”

POSTCONDITIONS: The program exits with words “Please retry with a higher number of times” after both Step 2 and 4.

(15.)

IDENTIFIER: TEST-SMALL-NUM-CONT-INVALID-RESPONSE

TEST CASE: This test case aims to test that, when the program is taking further instruction from the user to deal with the warning on minimum number of iterations, as long as the response entered is not ‘y’ or ‘n’, the program will keep asking the user for a valid selection.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with the arguments “car goat 80 5” and wait until the warning for minimum iterations shows up.

2. Enter “m” and press “Enter”.

3. Enter “d” and press “Enter”.

4. Enter “q” and press “Enter”.

5. Enter “n” and press “Enter”.

POSTCONDITIONS: The program displays “Sorry, I don’t know what m means!” after Step 2. The program displays “Sorry, I don’t know what d means!” after Step 3. The program displays “Sorry, I don’t know what q means!” after Step 4. The program exits with words “Please retry with a higher number of times” after Step 5.

(16.)

IDENTIFIER: TEST-NF-PERFORMANCE

TEST CASE: This test case aims to confirm that the program runs faster with a higher number of num\_threads given.

PRECONDITIONS: Command prompt window is opened and the current directory is the folder where GoatGoatCar.jar is saved.

EXECUTION STEPS: 1. Run the program with command line argument “car goat 4000 5”.

2. Benchmark the time used as t1.

3. Run the program with the arguments “car goat 4000 4000”

4. Benchmark the time used as t2

POSTCONDITIONS: t2 is smaller than t1

Traceability Matrix

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req/ Tst | (1) TEST- BASIC-CALC | (2,3,4) TEST-THREADS | (5) TEST-DISPLAY-RESULTS | (6,7) TEST-ARGS-NUMBER | (8,9) TEST-ARGS-INVALID | (10,11,12) TEST-SMALL-NUM | (13,14,15) TEST-SMALL-NUM-CONT | (16) TEST-NF-PERFORMANCE |
| (1) FUN-BASIC-CALC | x |  |  |  |  |  |  |  |
| (2) FUN-THREADS |  | x |  |  |  |  |  |  |
| (3) FUN-DISPLAY-RESULTS |  |  | x |  |  |  |  |  |
| (4) FUN-ARGS-NUMBER |  |  |  | x |  |  |  |  |
| (5) FUN-ARGS-INVALID |  |  |  |  | x |  |  |  |
| (6) FUN-SMALL-NUM |  |  |  |  |  | x |  |  |
| (7) FUN-SMALL-NUM-CONT |  |  |  |  |  |  | x |  |
| (8) NF-PERFORMANCE |  |  |  |  |  |  |  | x |

Defects

(1)

SUMMARY: High num\_times error

DESCRIPTION: When you enter above (231 – 1), it does not read the value properly. This is found in TEST-SMALL-NUM-ABOVE-THE-BORDER.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with the arguments “car goat 10000000000 100”

EXPECTED BEHAVIOR: Display the results of 10000000000 iterations of CarCarGoat.

OBSERVED BEHAVIOR: Displays that num\_times must be greater than 0.

(2)

SUMMARY: More threads don’t always make program faster

DESCRIPTION: The more threads the program runs with, the more time it takes to print out the number of iterations per thread, which will cause a delay in the run time of the program, and that is unexpected. Also, threads take extra time without the print statements, and only offer improvement up to a certain point, i.e. available amount of CPUs. This is found in TEST-NF-PERFORMANCE.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with command line argument “java -jar GoatGoatCar.jar car goat 4000 5”

3. Benchmark the time used as t1.

4. Run the program with command line argument “java -jar GoatGoatCar.jar car goat 4000 4000”

5. Benchmark the time used as t2

EXPECTED BEHAVIOR: t1 is less than t2

OBSERVED BEHAVIOR: t2 is noticeably longer than t1

(3)

SUMMARY: Program accepts the same value as success and failure flags

DESCRIPTION: The program will accept car for both “success” option and “failure” option, which is counterintuitive to the expectations of the program – how can one both win *and* lose for winning a car? This will make the problem itself nonsensical. This is found in TEST-ARGS-INVALID-IN-STRING-ARGS.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with java -jar GoatGoatCar.jar Car Car 1000 4

EXPECTED BEHAVIOR: Although not explicitly stated, it is implicit that it would throw some sort of error because the problem wouldn’t make sense.

OBSERVED BEHAVIOR: Runs as normal without warning the user that it makes no sense.

(4)

SUMMARY: small\_num border value is wrong

DESCRIPTION: The recommended minimum number for iterations is 100, so passing 100 as the iteration number should not arouse any warning, but the testing fact is the opposite. Found in TEST-SMALL-NUM-WITH-EDGE-CASE.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with command line argument “java -jar GoatGoatCar.jar car goat 100 5”

EXPECTED BEHAVIOR: The program displays the chances of getting both the good and bad options had the user switched or had the user stayed in 100 iterations.

OBSERVED BEHAVIOR: The program issues a warning saying “Recommended minimum number of times is 100. Continue?[y/n]” and waits for the feedback from the user.

(5)

SUMMARY: Uppercase “N” is not accepted as valid input on the continuation prompt.

DESCRIPTION: In accordance with the FUN-SMALL-NUM-CONT requirement, passing “N” as the input when prompted to “Continue?[y/n]” after entering an input less than 100 should terminate the program, but instead the program re-prompts. This is found in TEST-SMALL-NUM-CONT-CASE-INSENSITIVE-NO.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with command line argument “java -jar GoatGoatCar.jar car goat 99 4”

3. Enter “N” to the prompt

EXPECTED BEHAVIOR: The program prints “Please retry with a higher number of times” and terminates.

OBSERVED BEHAVIOR: The program prints “Sorry, I don't know what N means!” and re-prompts: “Continue?[y/n]”

(6)

SUMMARY: When the number of processes provided exceeds the number of iterations provided, the user should be warned that this will not make it any faster.

DESCRIPTION: The requirement states that more threads should make the program faster, but there cannot be any more division of labor than the number of iterations itself. If the user enters in more processes than iterations, he or she should be warned. This is found in TEST-ITERATION-DISTIBUTION-WITH-EDGE-CASE-2.

REPRODUCTION STEPS: 1. Open the command prompt window and change the directory to the folder where GoatGoatCar.jar is saved.

2. Run the program with command line argument “java -jar GoatGoatCar.jar car goat 101 102”

EXPECTED BEHAVIOR: The program should prompt the user: “You have provided more threads than iterations. There will be used threads.

Continue? [y/n]”, and wait for input.

OBSERVED BEHAVIOR: The program prints the results from the calculations, noting that the excess threads did no operations.