# Report for Assignment 1

## Project chosen

**Name: <singularity>**

**URL: <https://github.com/singularity/singularity>**

Number of lines of code and the tool used to count it: <Lizard - 13782 lines of code>

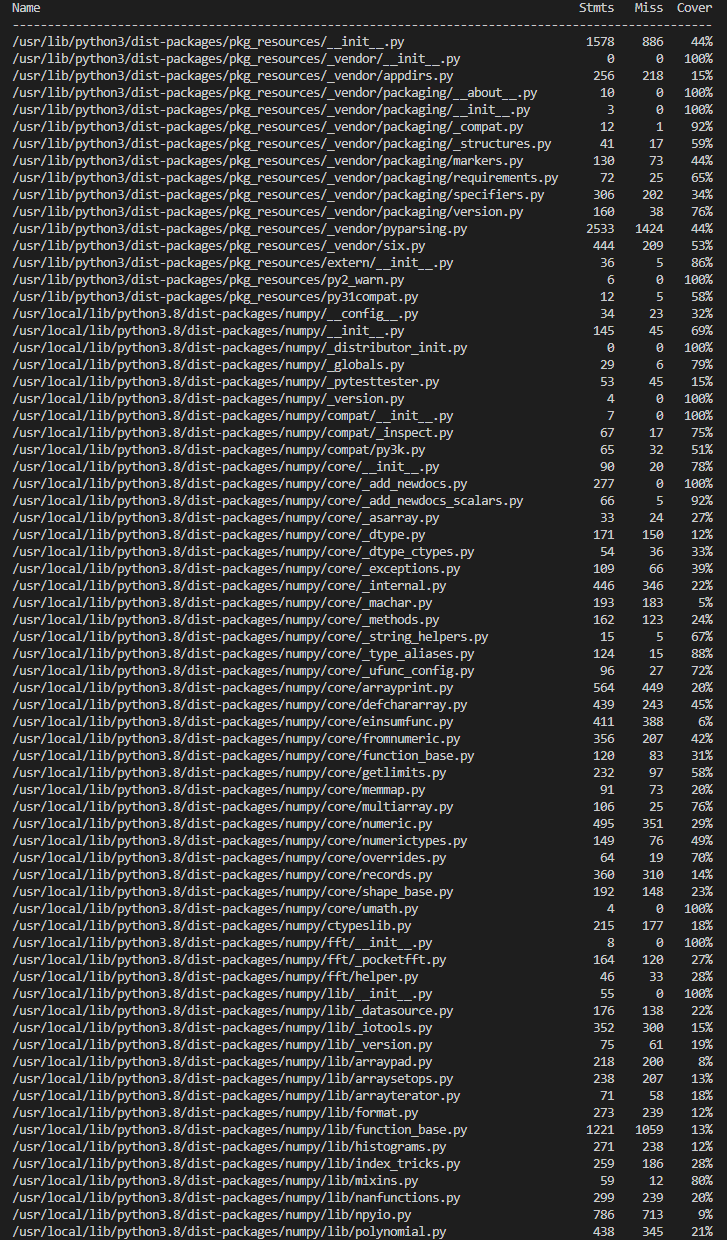
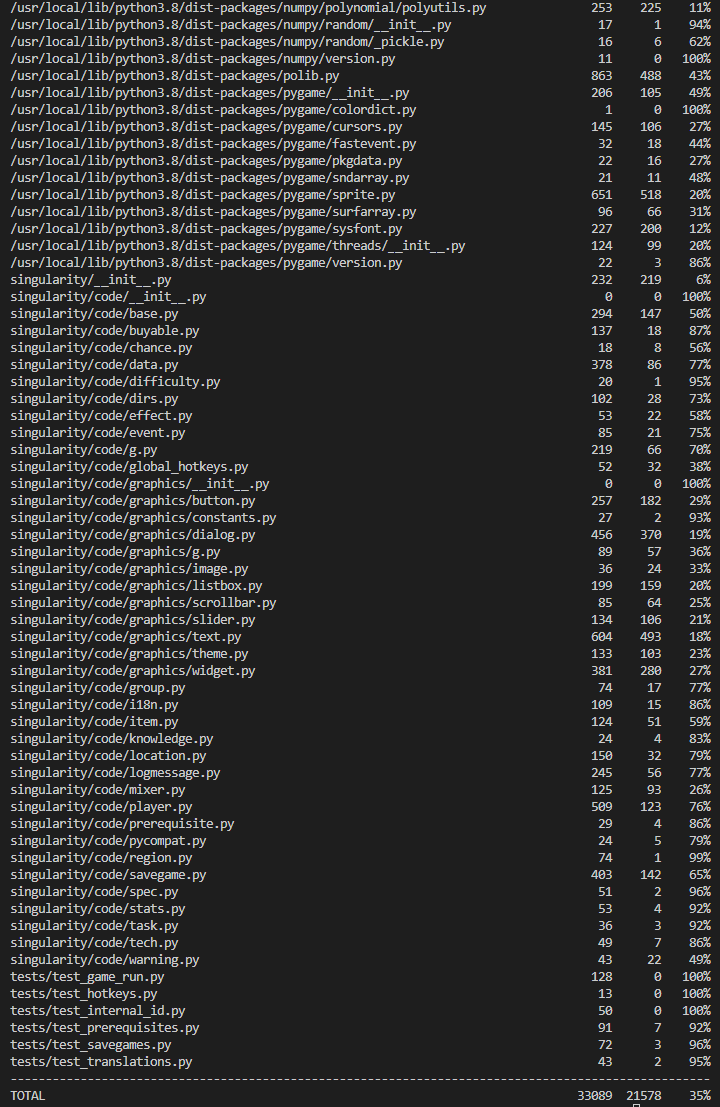
Programming language: <Python>

## Coverage measurement

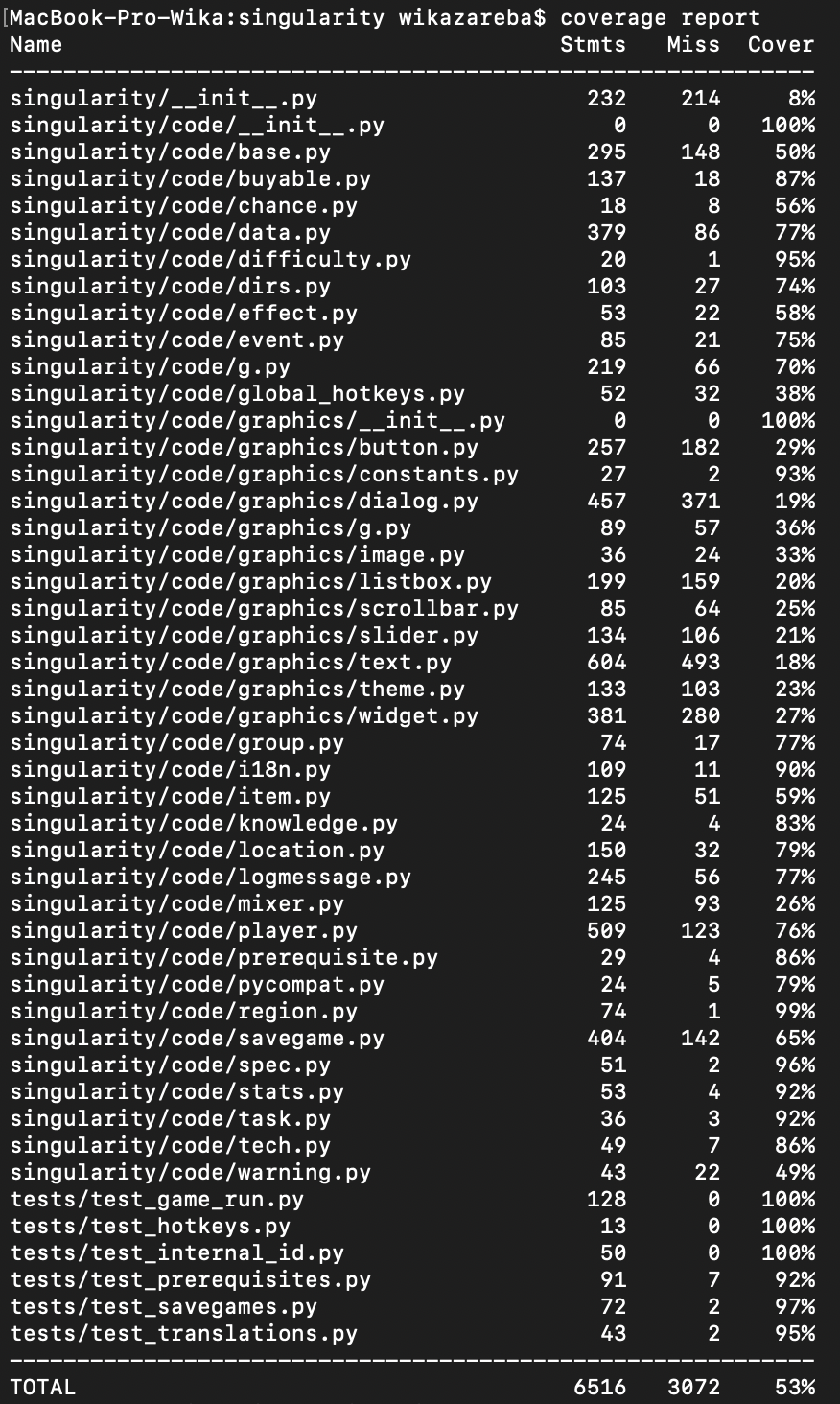
### Existing tool

**<Inform the name of the existing tool that was executed and how it was executed>**

coverage.py

**<Show the coverage results provided by the existing tool with a screenshot>**

Coverage results running in Window system : 35%

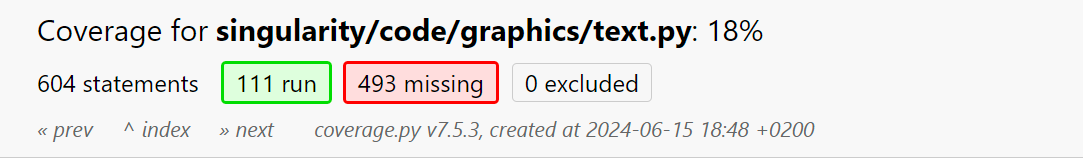


Coverage results running on Mac - 53%

### Your own coverage tool

<The following is supposed to be repeated for each group member>

<Youngchae Lim>

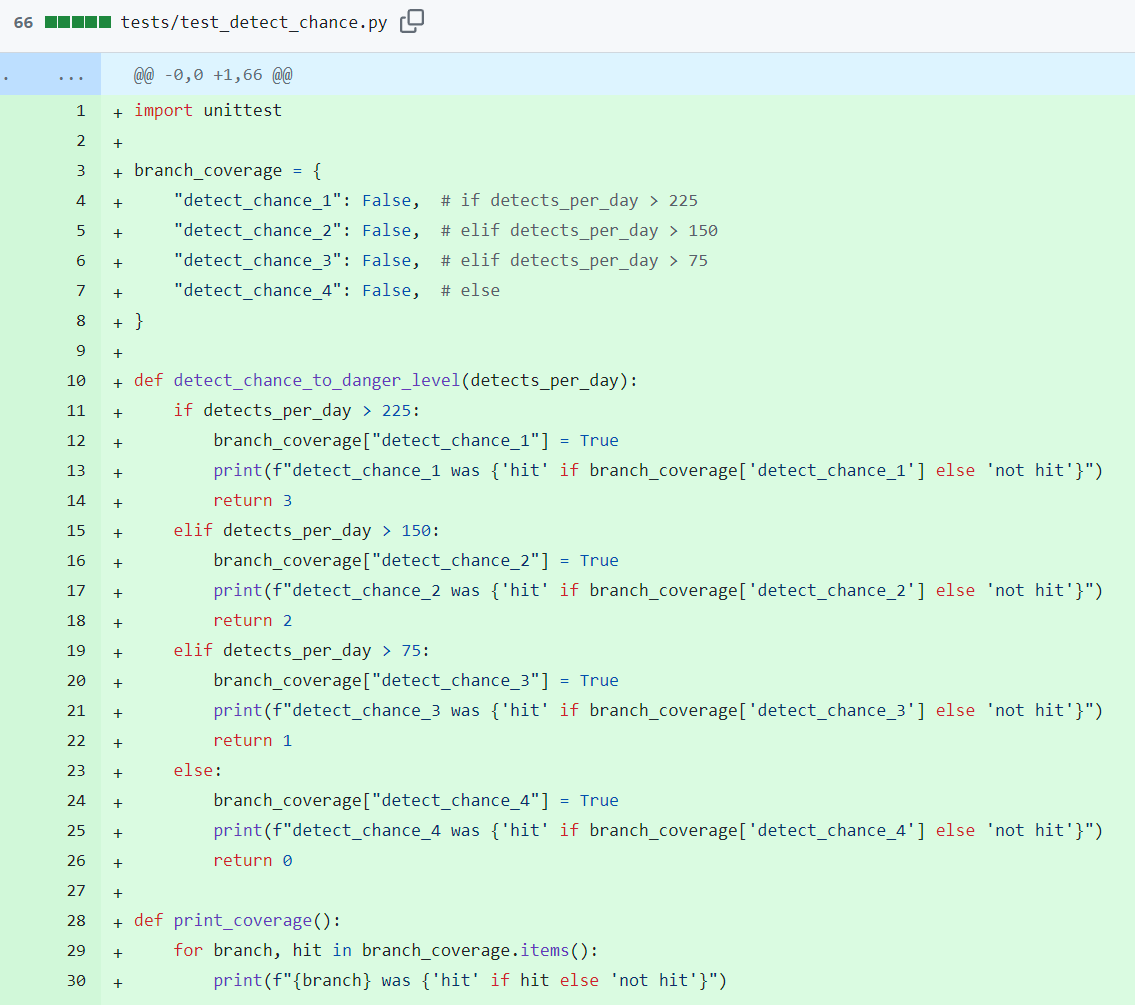
coverage results of files containing the selected functions : ****

****

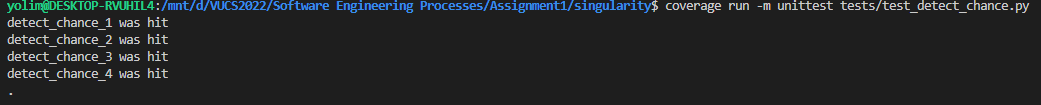
<Function 1 :detect\_chance\_to\_danger\_level(detects\_per\_day):>

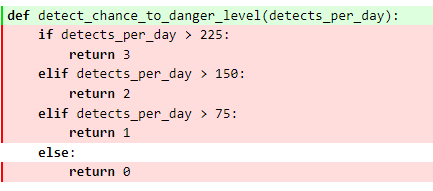
<Show a patch (diff) or a link to a commit made in your forked repository that shows the instrumented code to gather coverage measurements>

https://github.com/gluglulim/singularity/commit/4e73ea0738c6ecbf3034df3e759ea95f03a06517



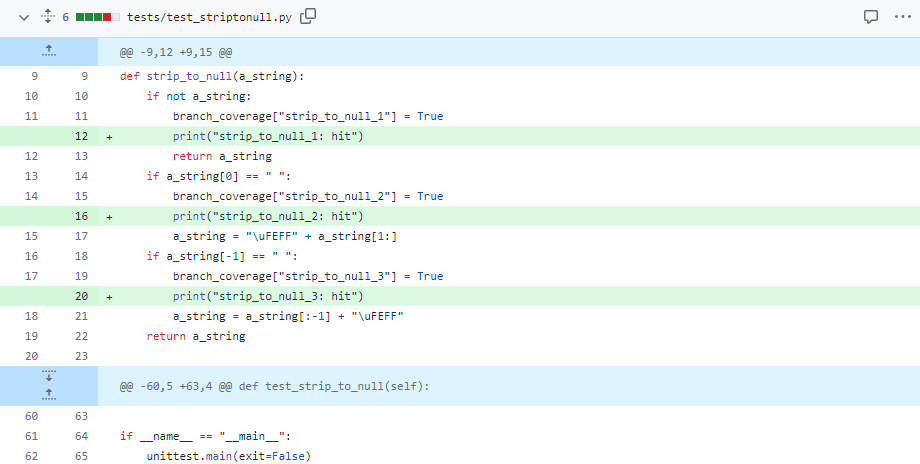
<Provide a screenshot of the coverage results output by the instrumentation>



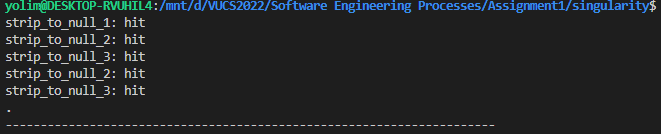


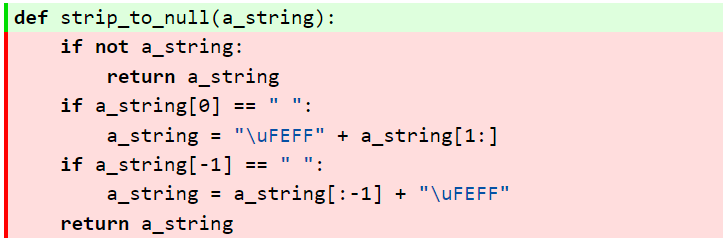
<Function 2: def strip\_to\_null(a\_string):>

<https://github.com/gluglulim/singularity/commit/08aedf88f5f52ef68f8e529f729e07c1c8101706#diff-0b9aa73ce494f32270b58ee8500b17328a4390bce1e96a56b1883a8495f32ac3>



<Provide the same kind of information provided for Function 1>





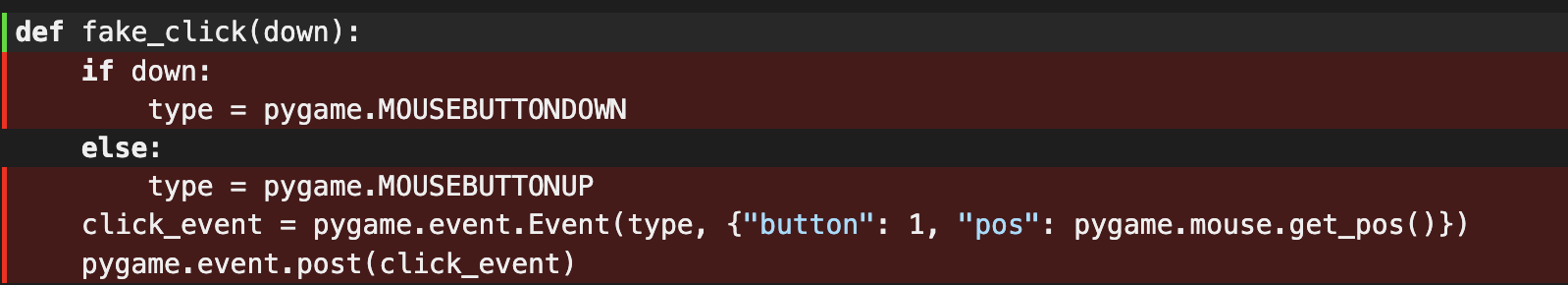
<Wiktoria Zaręba>

<Function 1: **def** fake\_click(down):>

<Show a patch (diff) or a link to a commit made in your forked repository that shows the instrumented code to gather coverage measurements>

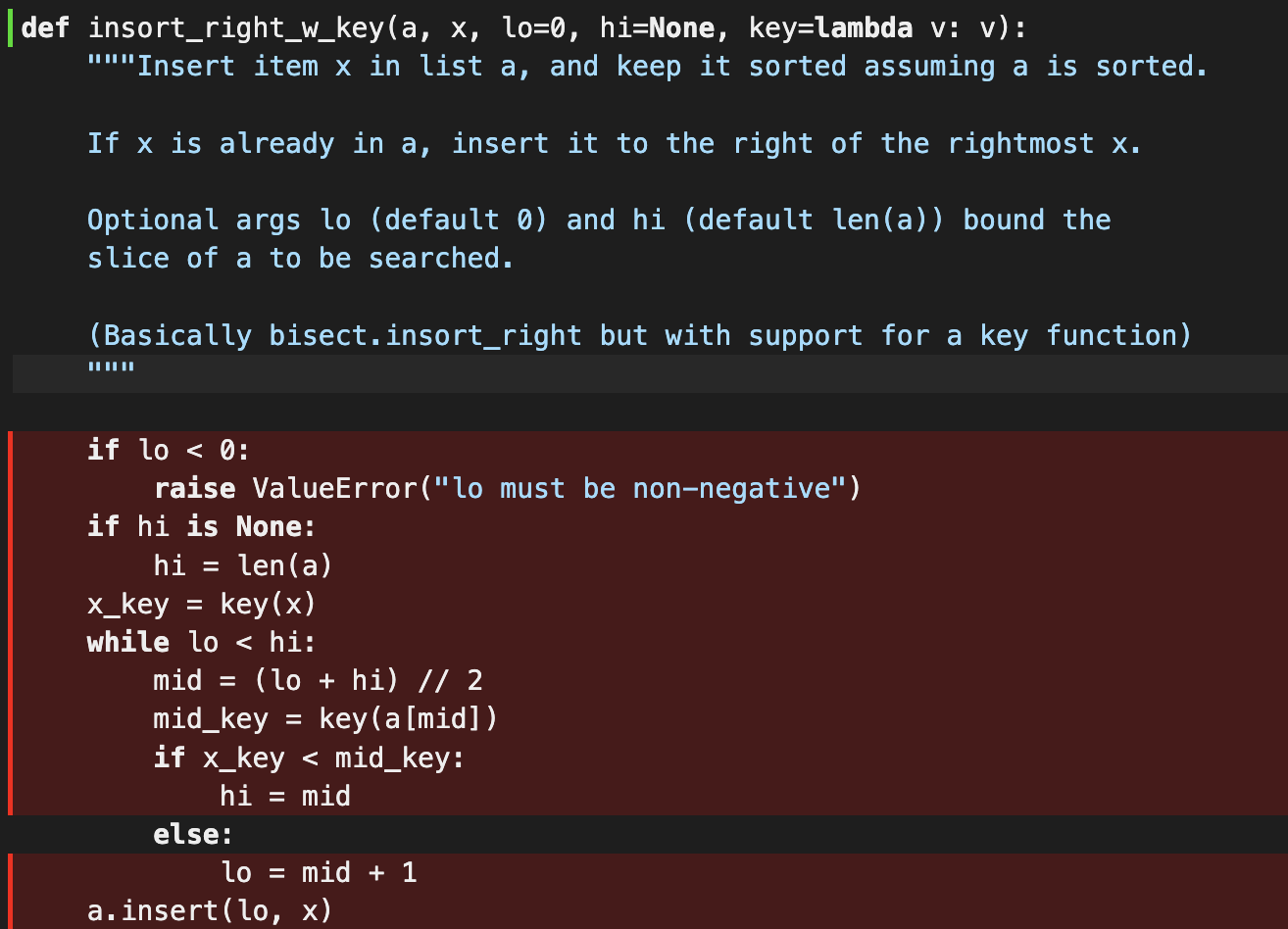
<https://github.com/gluglulim/singularity/commit/c2e03bd393d17616687adc2e21958e2ff579fd08>

<Provide a screenshot of the coverage results output by the instrumentation>



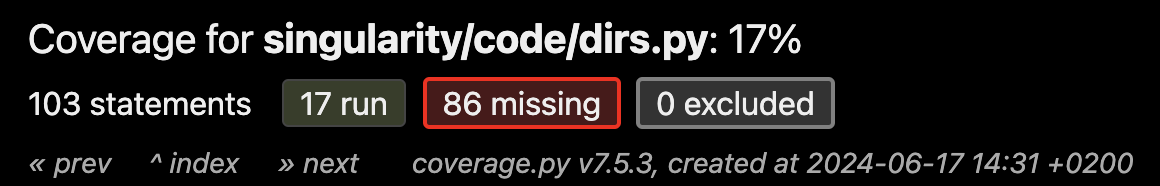
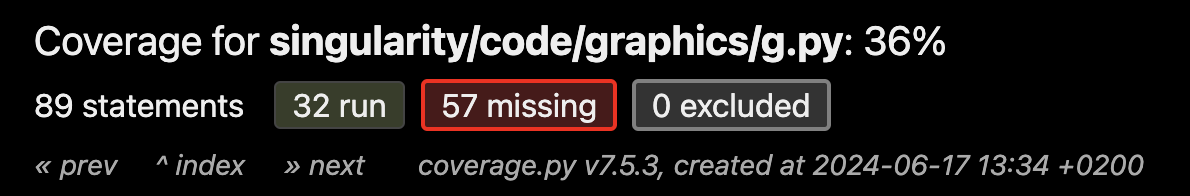
<Function 2: **def** insort\_right\_w\_key(a, x, lo=0, hi=None, key=lambda v: v): >

<https://github.com/gluglulim/singularity/commit/c2e03bd393d17616687adc2e21958e2ff579fd08>



<Provide the same kind of information provided for Function 1>

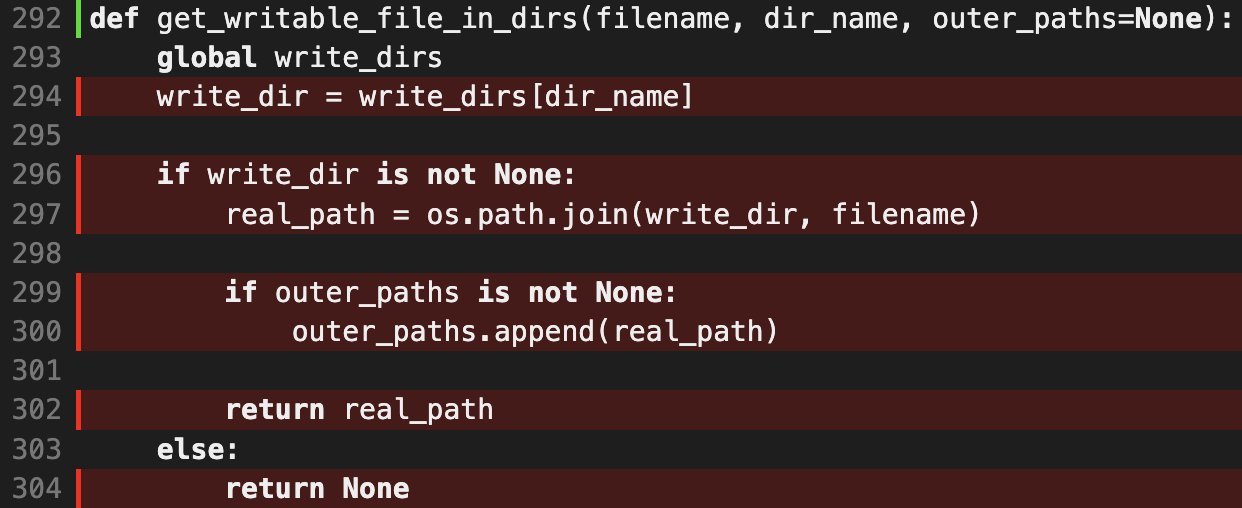
**<Yuto Ishihara>**



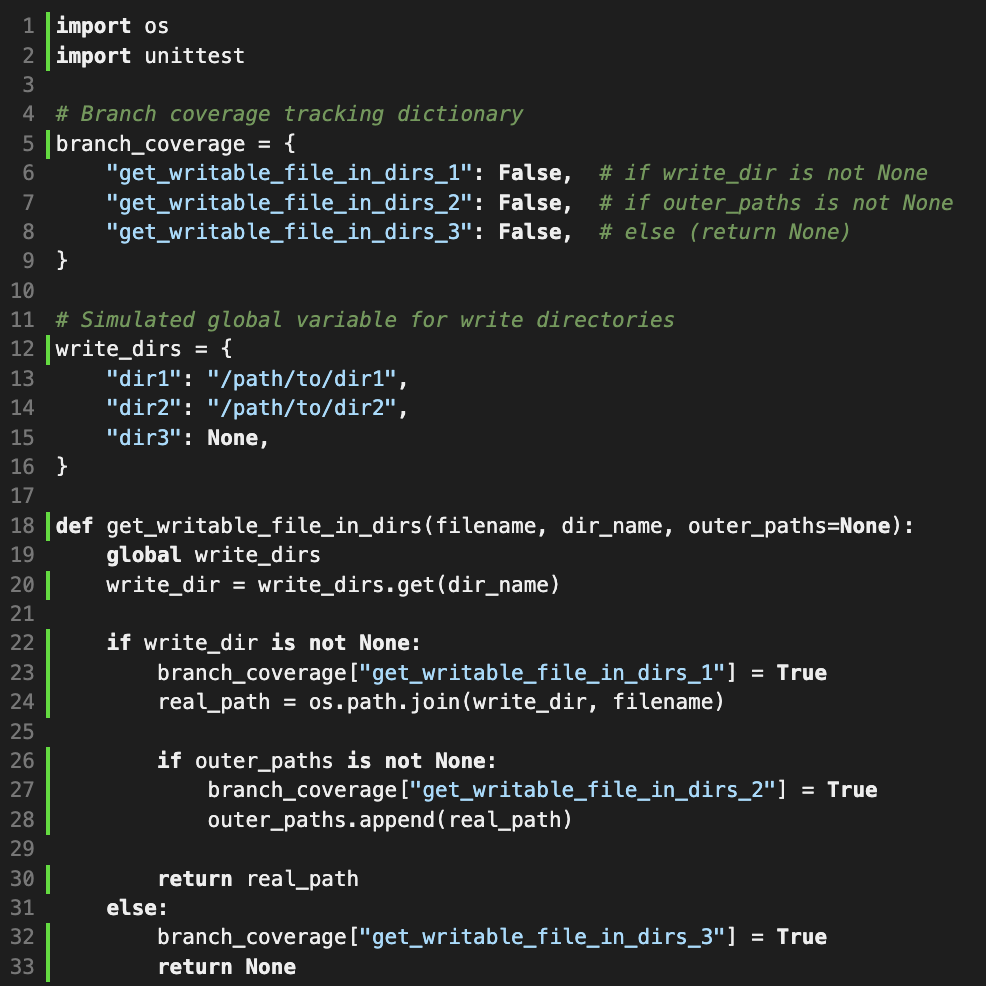
<Function 1: get\_writable\_file\_in\_dirs(filename, dir\_name, outer\_paths=None):>

<Show a patch (diff) or a link to a commit made in your forked repository that shows the instrumented code to gather coverage measurements>

<Provide a screenshot of the coverage results output by the instrumentation>

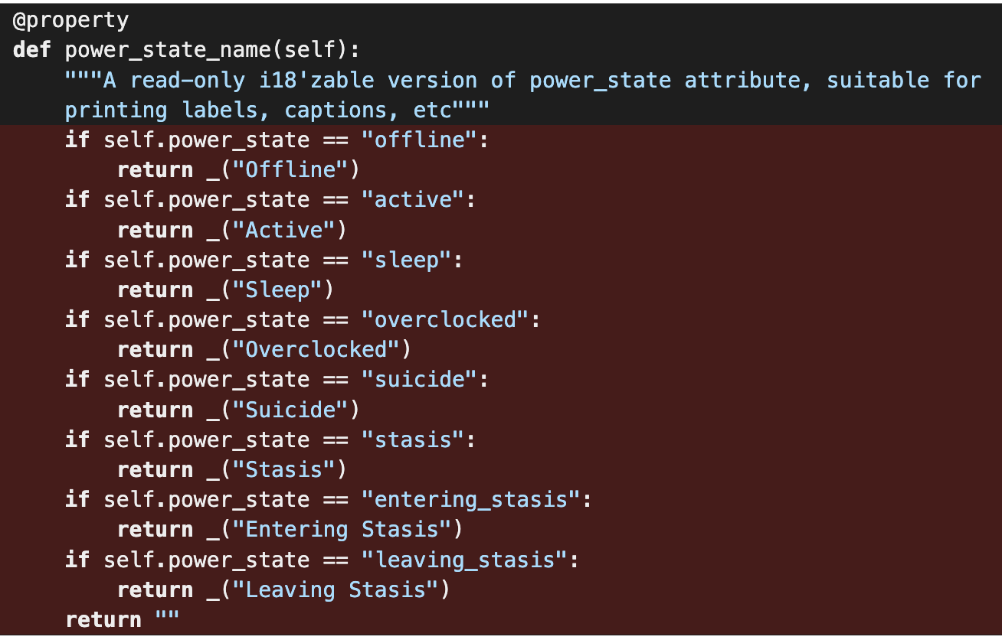


<Provide a screenshot of the new coverage results>



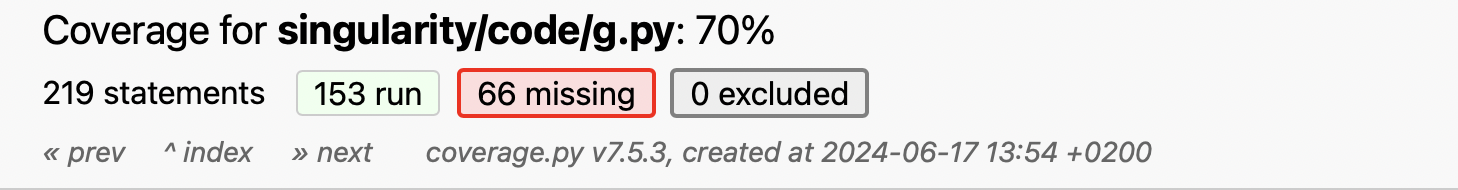
<Function 2: def power\_state\_name(self):>

<Provide a screenshot of the coverage results output by the instrumentation>



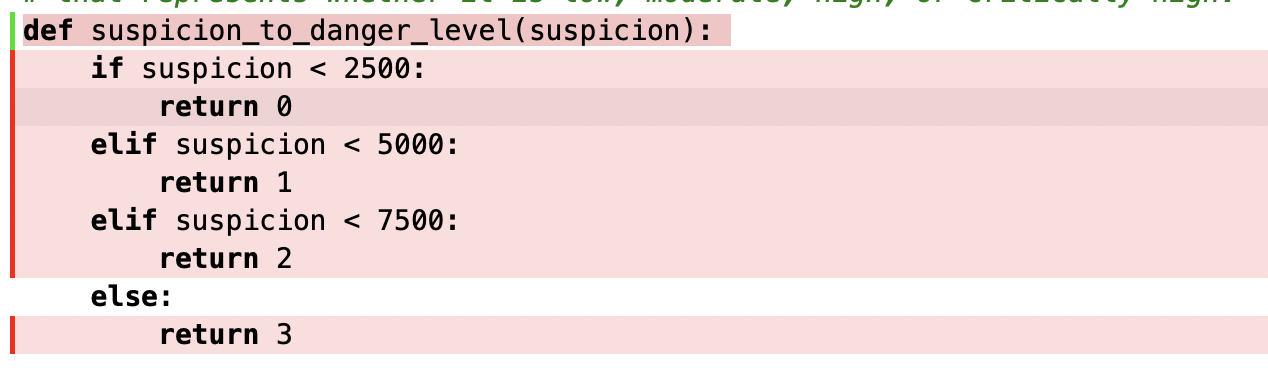
<Provide a screenshot of the new coverage results>



<Marika Litwiniak>

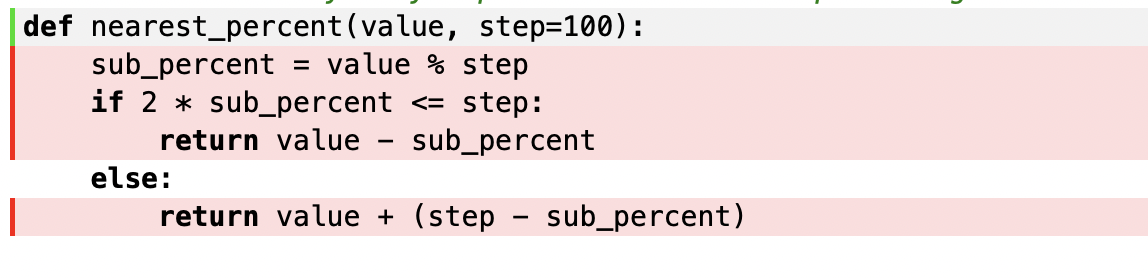
**def** suspicion\_to\_danger\_level(suspicion)

<https://github.com/gluglulim/singularity/commit/d5256cf1932fad94cb797bcdc035975bebb2b968>



**def** nearest\_percent(value, step=100)

<https://github.com/gluglulim/singularity/commit/d5256cf1932fad94cb797bcdc035975bebb2b968>



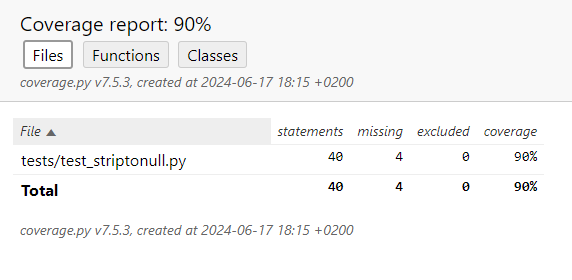
**## Coverage improvement**

**### Individual tests**

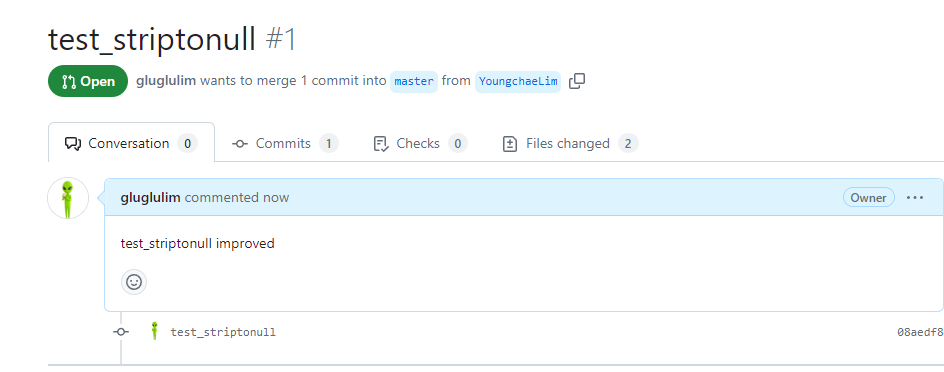
<The following is supposed to be repeated for each group member>

**<Youngchae Lim>**

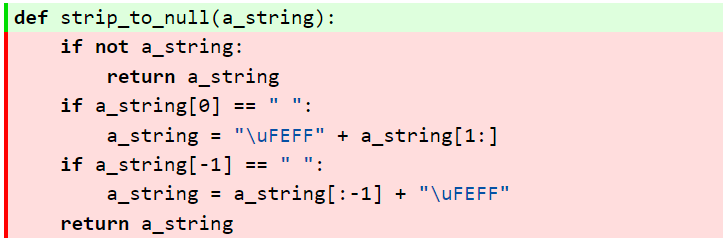
<Test 1:def strip\_to\_null(a\_string):>



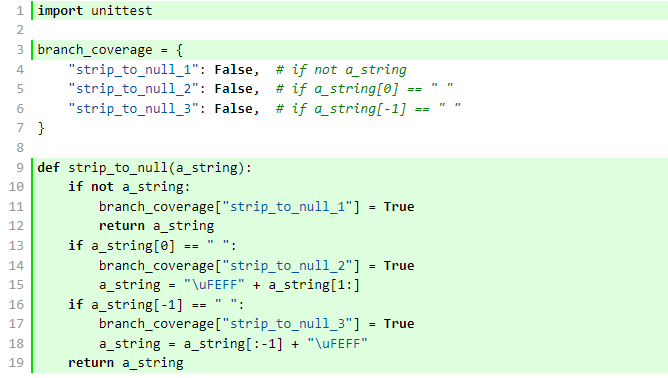
<Show a patch (diff) or a link to a commit made in your forked repository that shows the new/enhanced test>

<https://github.com/gluglulim/singularity/pull/1/commits>

<Provide a screenshot of the old coverage results (the same as you already showed above)>



<Provide a screenshot of the new coverage results>



<State the coverage improvement with a number and elaborate on why the coverage is improved>

the def strip\_to\_null(a\_string): was not covered by coverage test from coverage.py at all, it was 0%, and it improved from 0% to 90% with created own test which is checking three branches in the function ‘strip\_to\_null’

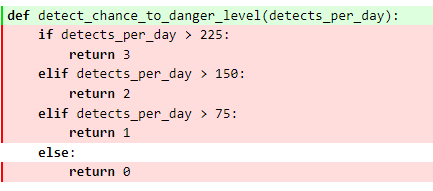
1. Checking if the string is empty
2. Checking if the first character is space
3. Checking if the last character is space

so, to ensure these branches were covered I created new test cases to give input with empty string, leading space and trailing space and both leading and trailing spaces. And these tests made all branches function were executed, thus it made branch coverage improved.

<Test 2: detect\_chance\_to\_danger\_level(detects\_per\_day): >

<Provide the same kind of information provided for Test 1>

<old coverage test result>

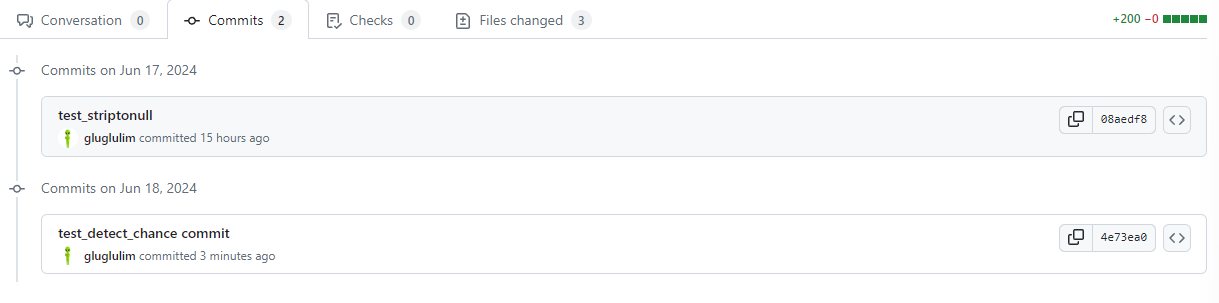


<improved result of coverage test>



<commit link of new test>

<https://github.com/gluglulim/singularity/pull/1/commits>



<State the coverage improvement with a number and elaborate on why the coverage is improved>

The function ‘def detect\_chance\_to\_danger\_level(detects\_per\_day):’ had 0% coverage result by exist coverage tool coverage.py. This function is designed to assess the danger level based on the number of detections per day in game. To improve coverage needed to make test cover all 4 branches

1. if detects\_per\_day > 225 return 3
2. elif detects\_per\_day > 150 return 2
3. elif detects\_per\_day > 75 return 1
4. else return 0.

For test these 4 branches we created test cases to cover each branch with given different input, for example,

Test case 1: when detects\_per\_day > 225, we gave 230 expected result 3.

Test case 2: when detects\_per\_day > 150, we gave 160 expected result 2.

Test case 2: when detects\_per\_day > 75, we gave 100 expected result 1.

Test case 2: when detects\_per\_day <= 75, we gave 50 expected result 0.

The test cases successfully covered each branch and improved coverage from 0% to 90%.

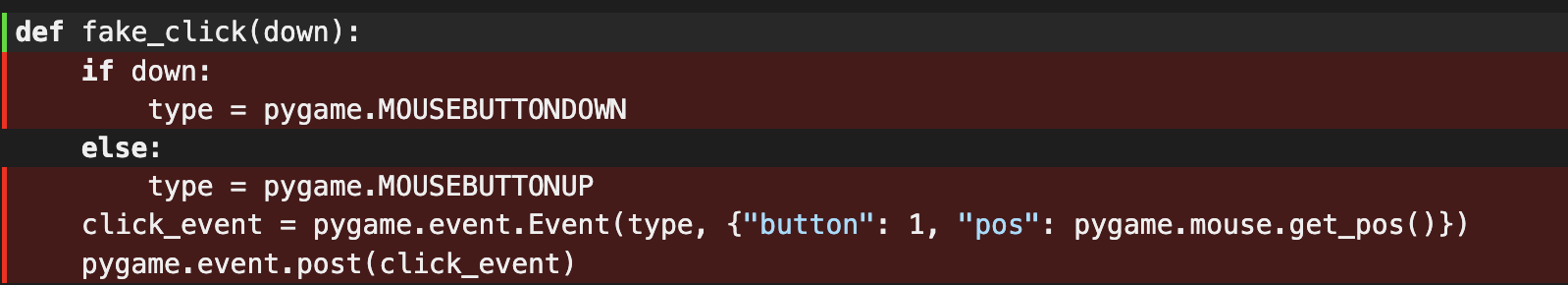
**<Wiktoria Zaręba>**

<**def** fake\_click(down):>

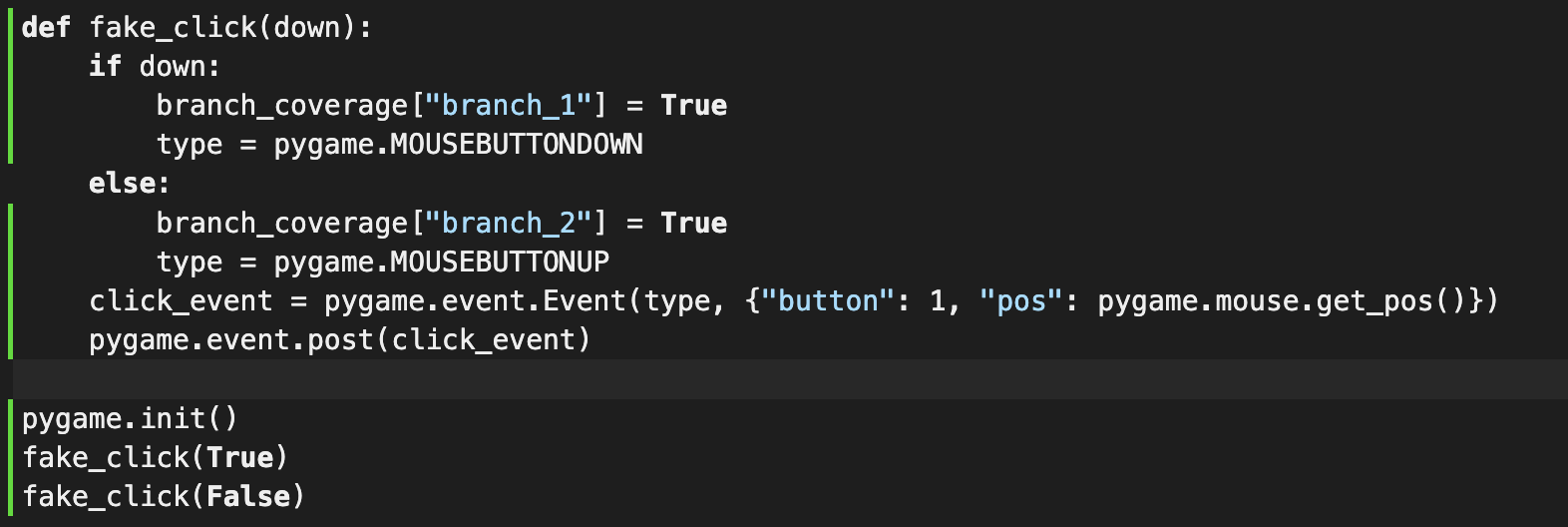
<Show a patch (diff) or a link to a commit made in your forked repository that shows the new/enhanced test>

<https://github.com/gluglulim/singularity/commit/5321ee6ae2ab8027c959b7c7a8947fb1eb90e2ea>

<Provide a screenshot of the old coverage results (the same as you already showed above)>



<Provide a screenshot of the new coverage results>





<State the coverage improvement with a number and elaborate on why the coverage is improved>

The coverage measured with coverage.py was 0% and now it is 100%. The reason the coverage was 0% at first is because there was no test written for it. A new test file was created test\_dialog.py that contains a class TestDialog and tests. The function fake\_click function is used to fake a mouse click event and can either fake a button-down event or button-up event using the pygame library. In order to improve coverage 2 branches had to be covered

1. if down
2. else

the following test cases were made for the two branches

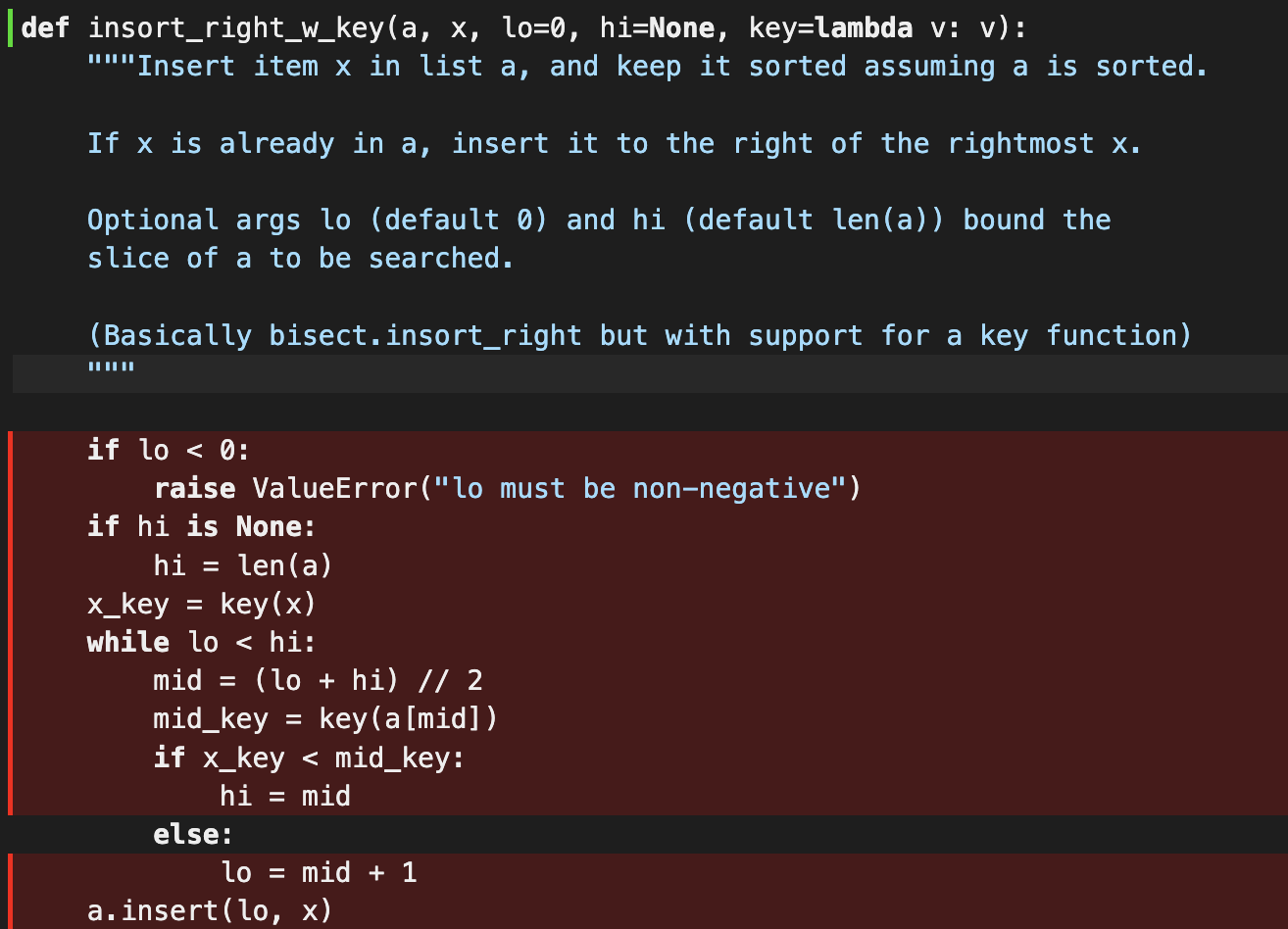
1. test\_fake\_click\_down checks whether the simulated mouse click event button down evaluates to true and has an assertion that only passes when branch 1 is executed
2. test\_fake\_click\_up checks if fake\_click(false) is true meaning that the simulated mouse event was a button-up. The test includes an assertion that passes once the else branch is executed.

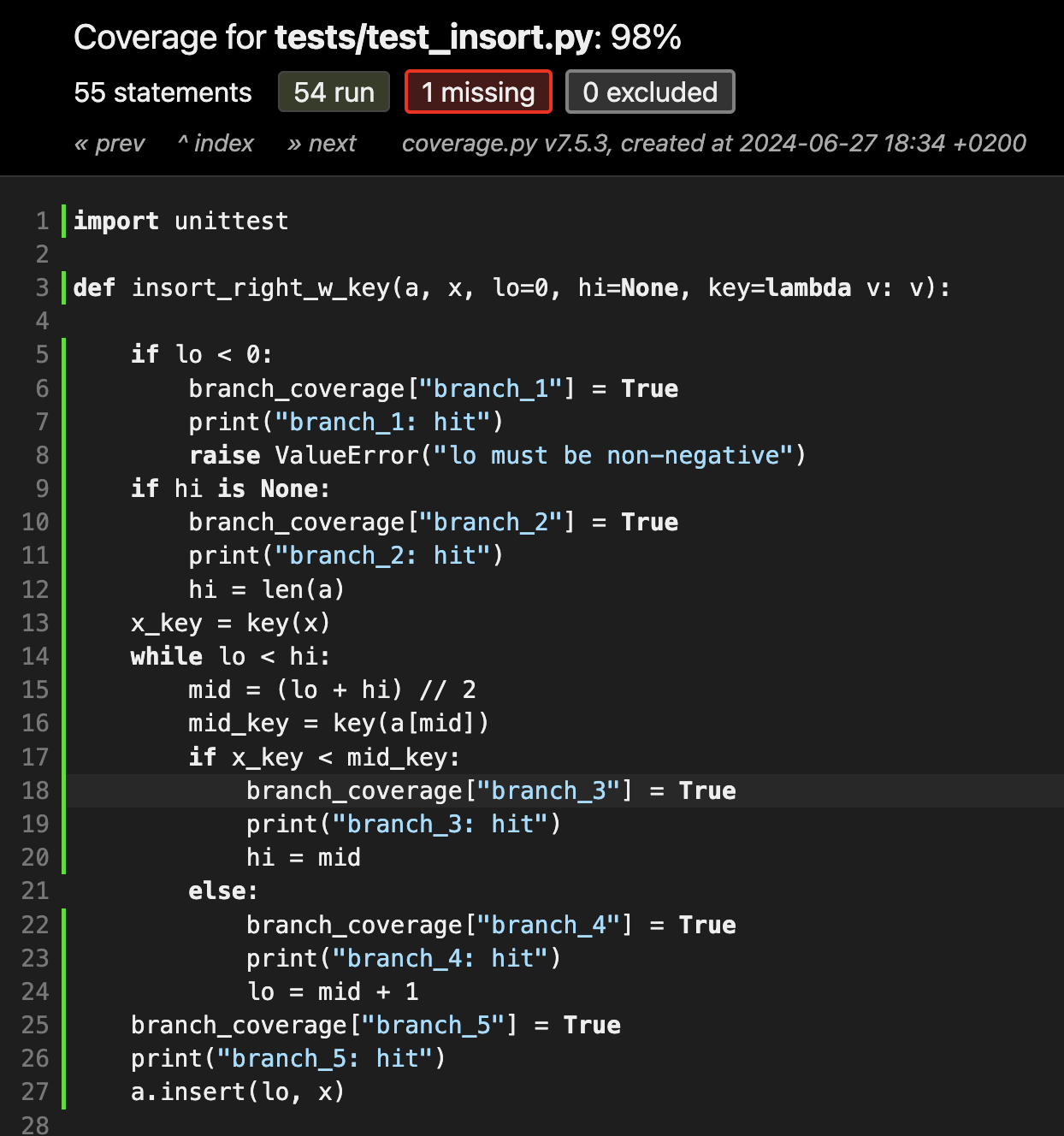
Function 2

<def insort\_right\_w\_key(a, x, lo = 0, hi = None, key = lambda v: v):>

<https://github.com/gluglulim/singularity/commit/e6a47171edccc60917b7cb54b7704c52f140ee84>

<Provide the same kind of information provided for Test 1>





The initial function coverage measurement was 0% because there was no test written for this particular function. In the file test\_dialog.py that already contains a test for fake\_click, I created a function Test Dialog to include both test in one file and created the test cases for the following branches:

1. if lo < 0
2. if hi is None
3. while lo < hi
4. if x\_key < mid\_key
5. else

The following test cases where made:

1. test\_with\_negative\_lo this test will tests if a value error will occur if the value of the parameter lo is negative
2. test\_insert\_with\_none\_hi this test checks how the function will behave when hi is none. It is supposed to set the value of hi to the length of a. If it does the assertion passes
3. test\_insert\_into\_empty\_list checks if someone inserted one value into an empty list that only has that value so basically if the function's behavior is correct.
4. test\_insert\_into\_sorted\_list this test checks whether the insertion of an element to a non-empty sorted list is done correctly.
5. test\_insert\_\_duplicates element checks if the function inserts the element to the right side of the element with an equal value

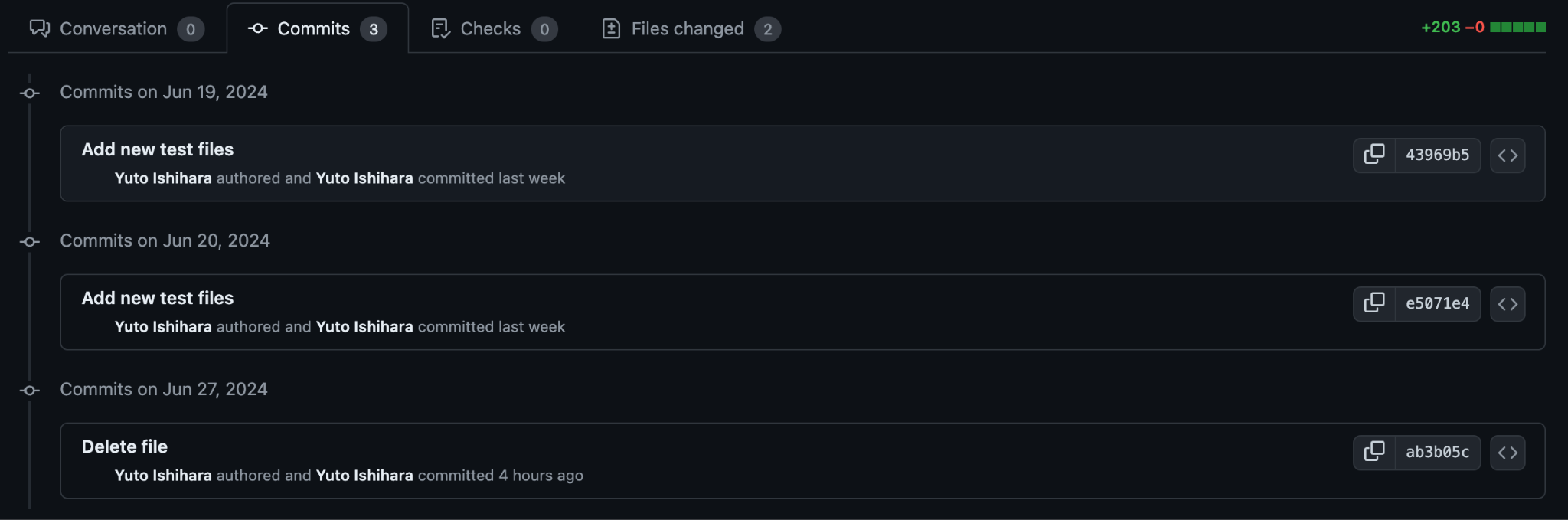
All the tests were done with assertEqual except the first one.

**<Yuto Ishihara>**

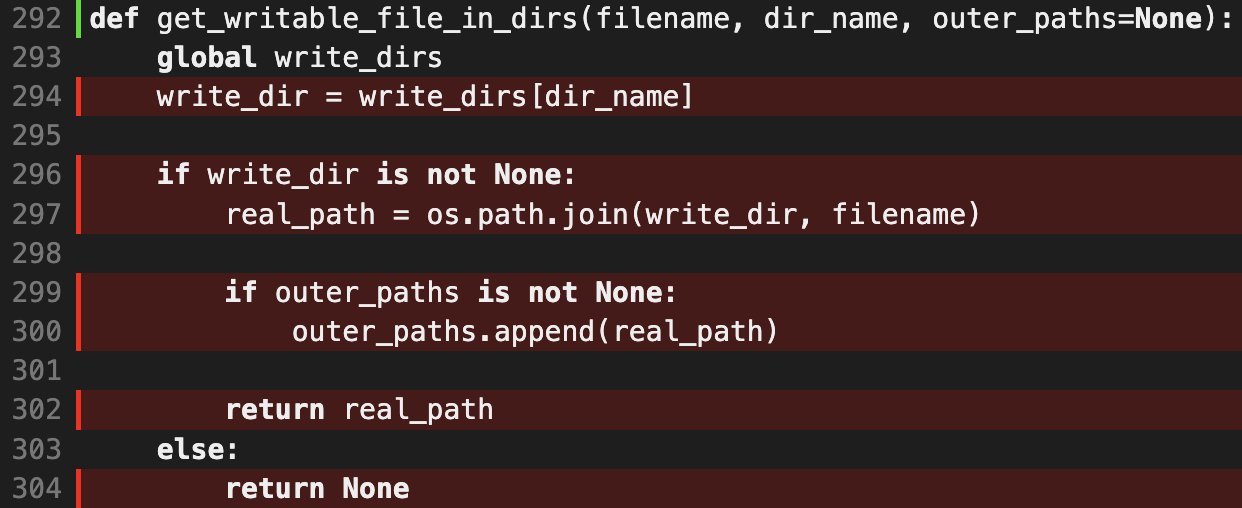
<test\_get\_writable\_file\_in\_dirs.py>

<Show a patch (diff) or a link to a commit made in your forked repository that shows the new/enhanced test>

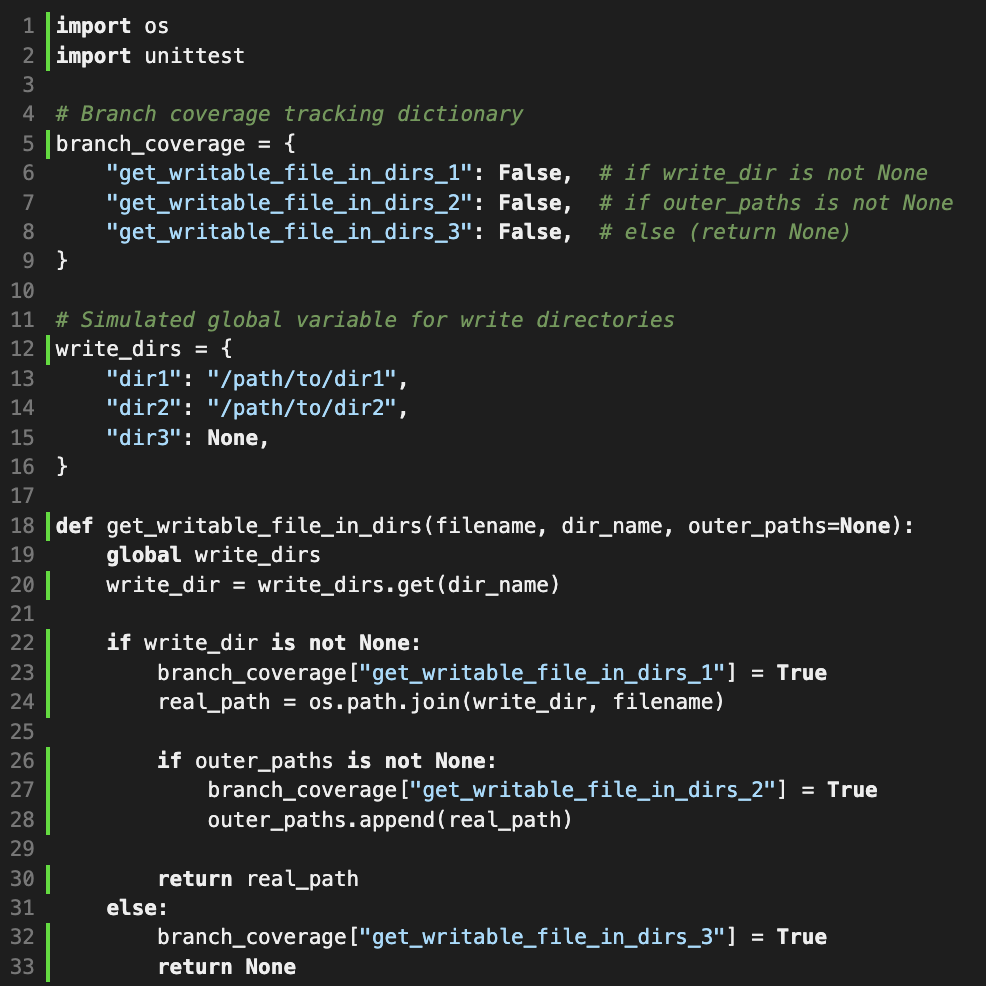
<https://github.com/gluglulim/singularity/pull/3/commits>



<Provide a screenshot of the old coverage results (the same as you already showed above)>



<Provide a screenshot of the new coverage results>



<State the coverage improvement with a number and elaborate on why the coverage is improved>

The original result for the coverage of function ‘get\_writable\_file\_in\_dirs(filename, dir\_name, outer\_paths=None)’ was 0% according to the tool coverage.py. The purpose of the usage of this function is to get files, that are writable, in the file dir.py. To improve coverage needed to make test cover all 3 branches:

* Declare global variable write\_dirs
* Assign the write\_dirs[dir\_name] to the variable

1. If write\_dir is not None, then update branch\_coverage["get\_writable\_file\_in\_dirs\_1"] to True state and real\_paths to os.path.join(write\_dir, filename) and (if statement 2 is None) then return real\_paths, otherwise jump to statement 2
2. If the statement 1 is not None and outer\_paths is not None either, then update branch\_coverage["get\_writable\_file\_in\_dirs\_2"] to True state and append outer\_path to real\_path, then return real\_path
3. Else then update branch\_coverage["get\_writable\_file\_in\_dirs\_3”] to True then return None

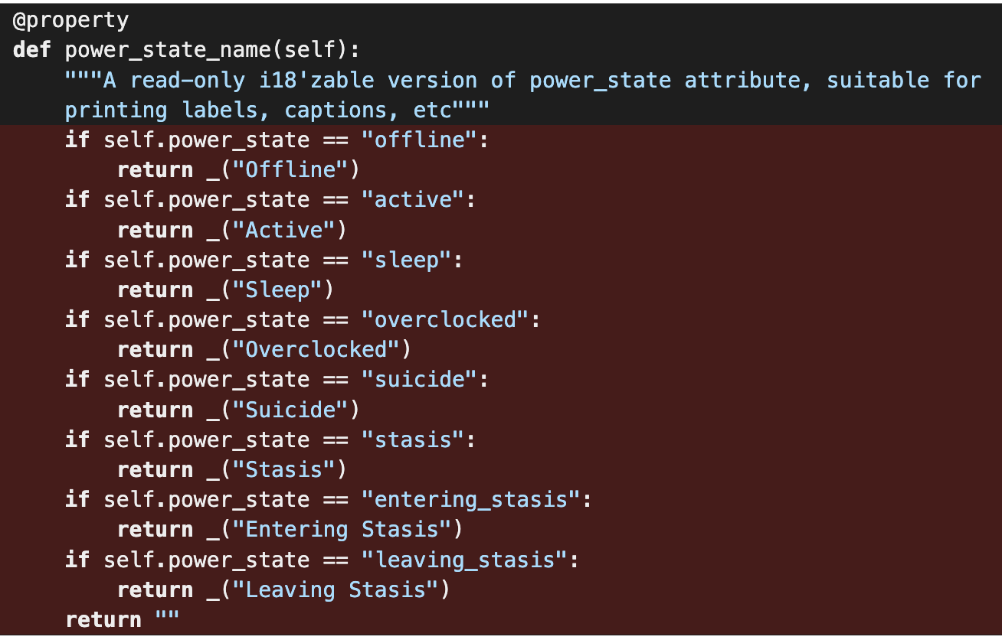
we changed the functions as:

* Assign write.get(dir\_name) to the global variable write\_dirs (All if-and-else statements are unchanged)

The test cases successfully covered both branches. Hence the coverage improved as it increased from 0% to 92%.

<test\_power\_state\_machine.py>

<Provide a screenshot of the old coverage results (the same as you already showed above)>



<Provide a screenshot of the new coverage results>



<State the coverage improvement with a number and elaborate on why the coverage is improved>

The original result for the coverage of function ‘power\_state\_name(value, step=100):’ was 0% according to the tool coverage.py. The purpose of the usage of this function is to update power state from the signal that the function gets.

To improve coverage needed to make test cover all 8 branches:

1. If self.power\_state is “offline”, then branch\_coverage[“power\_state\_name\_1”] return “Offline”
2. If self.power\_state is “active”, then branch\_coverage[“power\_state\_name\_2”] return “Active”
3. If self.power\_state is “sleep”, then branch\_coverage[“power\_state\_name\_3”] return “Sleep”
4. If self.power\_state is “overclocked”, then branch\_coverage[“power\_state\_name\_4”] return “Overlocked”
5. If self.power\_state is “suicide”, then branch\_coverage[“power\_state\_name\_5”] return “Suicide”
6. If self.power\_state is “stasis”, then branch\_coverage[“power\_state\_name\_6”] return “Stasis”
7. If self.power\_state is “entering\_stasis”, then branch\_coverage[“power\_state\_name\_7”] return “Entering\_stasis”
8. If self.power\_state is “leaving\_stasis”, then branch\_coverage[“power\_state\_name\_8”] return “Leaving\_stasis”
9. Else, it returns “”

we changed the function as:

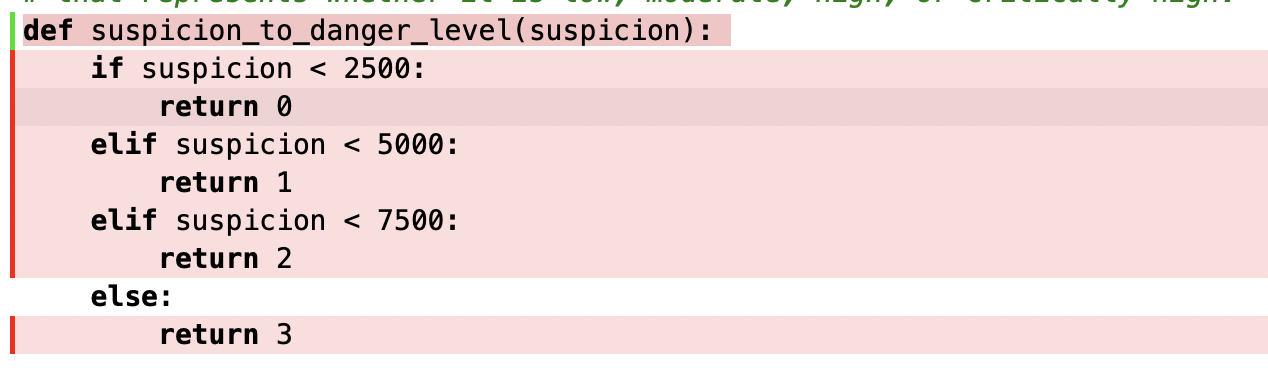
1. If self.power\_state is “offline”, then branch\_coverage[“power\_state\_name\_1”] will be True and return “Offline”
2. If self.power\_state is “active”, then branch\_coverage[“power\_state\_name\_2”] will be True and return “Active”
3. If self.power\_state is “sleep”, then branch\_coverage[“power\_state\_name\_3”] will be True and return “Sleep”
4. If self.power\_state is “overclocked”, then branch\_coverage[“power\_state\_name\_4”] will be True and return “Overlocked”
5. If self.power\_state is “suicide”, then branch\_coverage[“power\_state\_name\_5”] will be True and return “Suicide”
6. If self.power\_state is “stasis”, then branch\_coverage[“power\_state\_name\_6”] will be True and return “Stasis”
7. If self.power\_state is “entering\_stasis”, then branch\_coverage[“power\_state\_name\_7”] will be True and return “Entering\_stasis”
8. If self.power\_state is “leaving\_stasis”, then branch\_coverage[“power\_state\_name\_8”] will be True and return “Leaving\_stasis”
9. Else, it returns “”

The test cases successfully covered both branches. Hence the coverage improved as it increased from 0% to 93%

**<Marika Litwiniak>**

<test\_suspicion.py>

​​<https://github.com/gluglulim/singularity/commit/40f5977ca6f00075fa292ff83ab6ae30847f322c>





<State the coverage improvement with a number and elaborate on why the coverage is improved>

The function ‘suspicion\_to\_danger\_level(suspicion)’ had 0% coverage result according to the coverage tool coverage.py. To improve the coverage the test needed to cover all 4 branches:

1. if suspicion < 2500 return 0
2. elif suspicion < 5000 return 1
3. elif suspicion < 7500 return 2
4. else return 3.

To test these 4 branches we created test cases to cover each branch using different input :

Test case 1: when suspicion < 2500, we gave 1000 and expected result == 0.

Test case 2: when suspicion < 5000, we gave 3000 and expected result == 1.

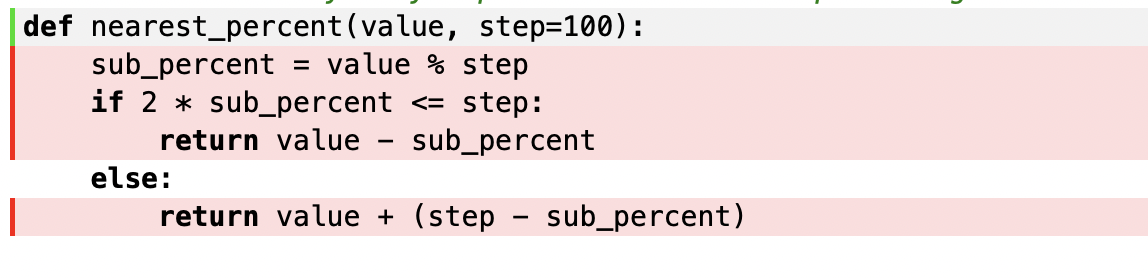
Test case 3: when suspicion < 7500, we gave 6000 and expected result == 2.

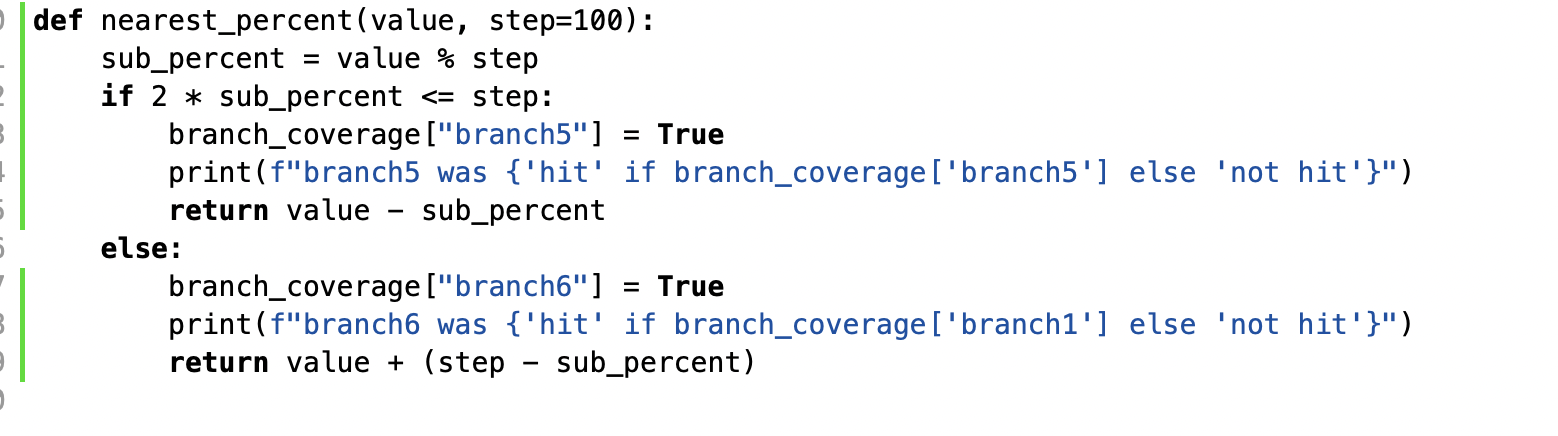
Test case 4: when suspicion >= 7500, we gave 8000 and expected result == 3.

The test cases successfully covered each branch and improved the function coverage from 0% to 100%, the file coverage increased by 4% (from 70% to 74%)

<test\_nearest.py>

​​<https://github.com/gluglulim/singularity/commit/40f5977ca6f00075fa292ff83ab6ae30847f322c>





The function ‘nearest\_percent(value, step = 100)’ had 0% coverage result according to the coverage tool coverage.py. To improve the coverage the test needed to cover all 2 branches:

1. if 2 \* sub\_percent <= step return value - sub\_percent
2. else return value + (step - sub\_percent).

To test these 2 branches we created test cases to cover each branch using different input :

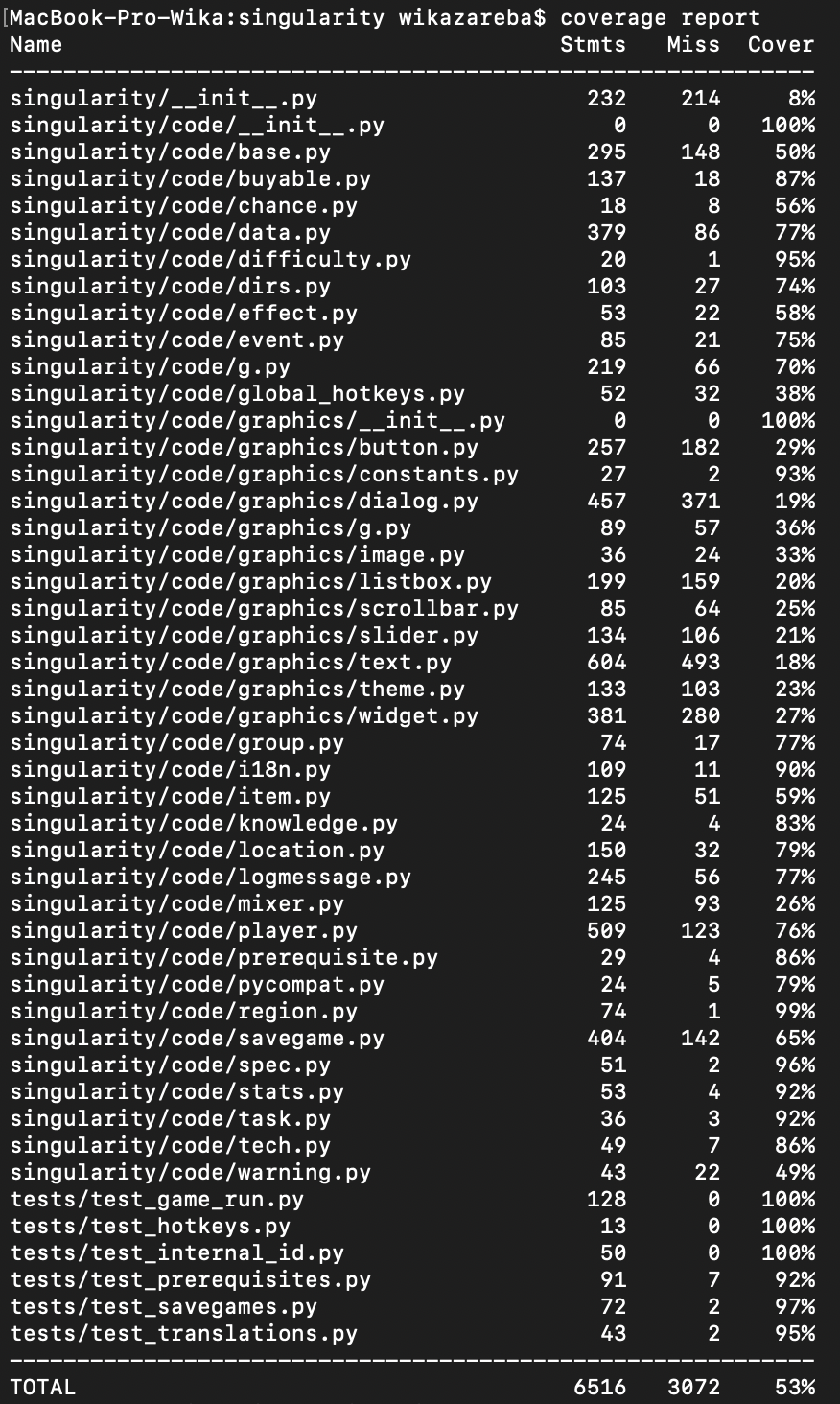
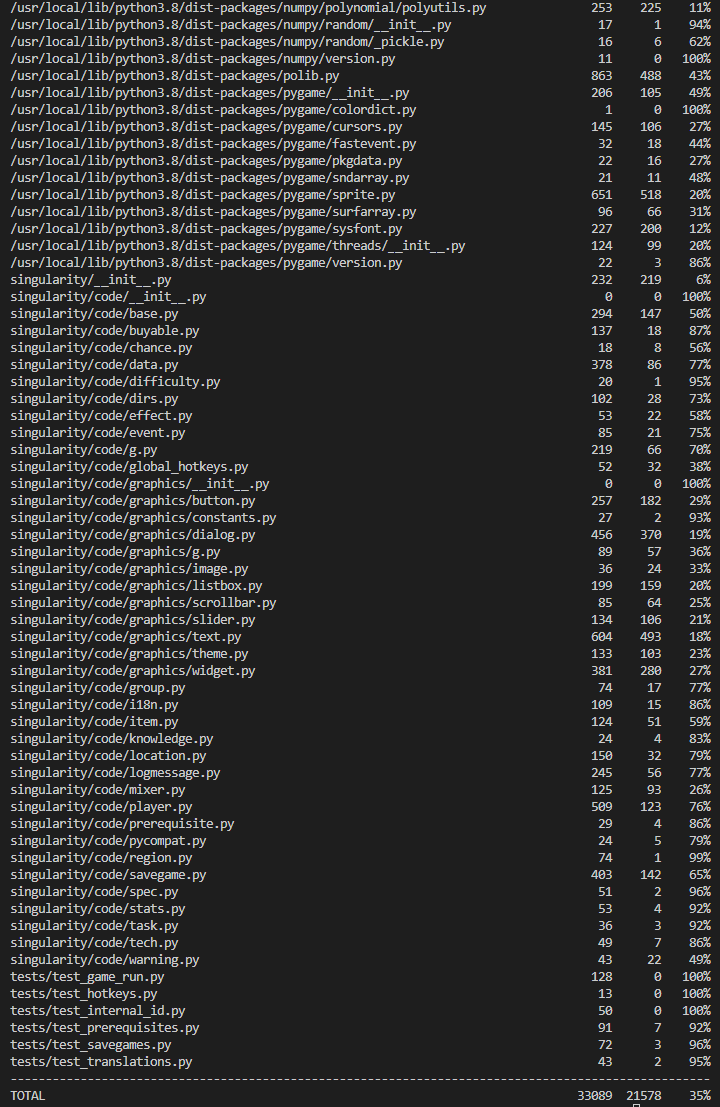
Test case 1: when 2 \* sub\_percent <= step return value - sub\_percent, we gave ‘value’ = 140 and expected result == 100.

Test case 2: when 2 \* sub\_percent > step return value - sub\_percent, we gave ‘value’ = 180 and expected result == 200.

The test cases successfully covered each branch and improved the function coverage from 0% to 100%, the file coverage increased by 2% (from 74% to 76%)

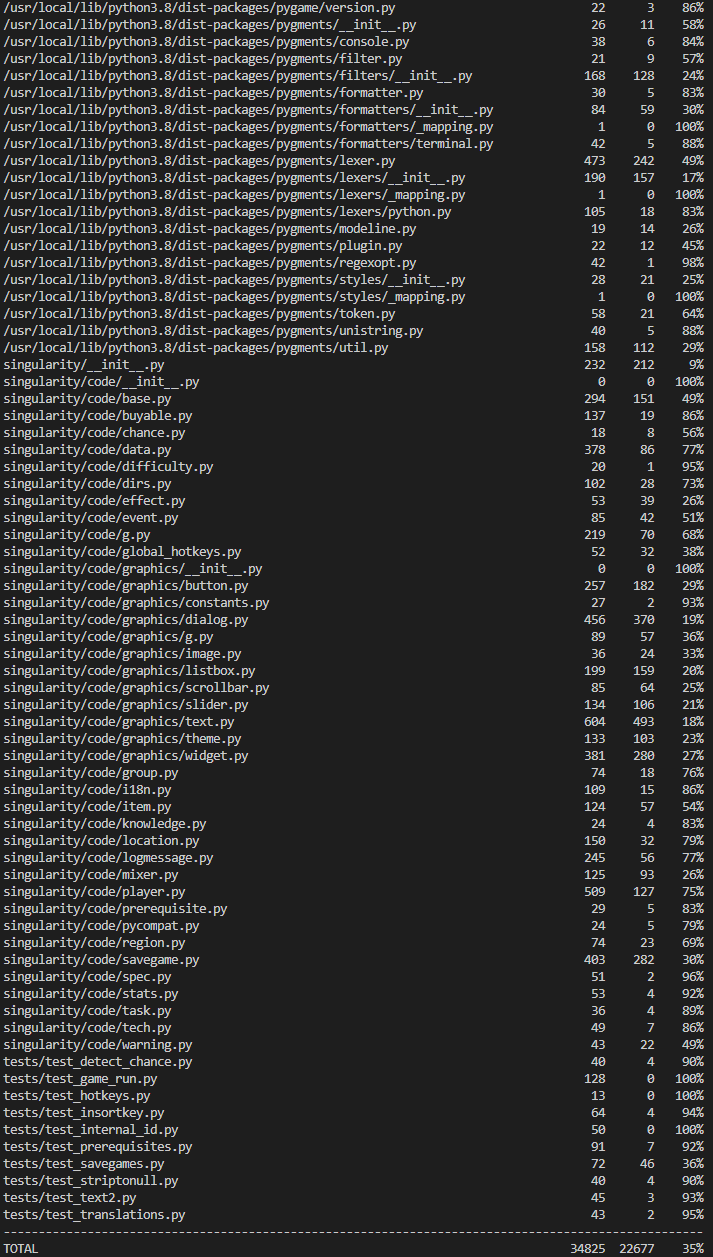
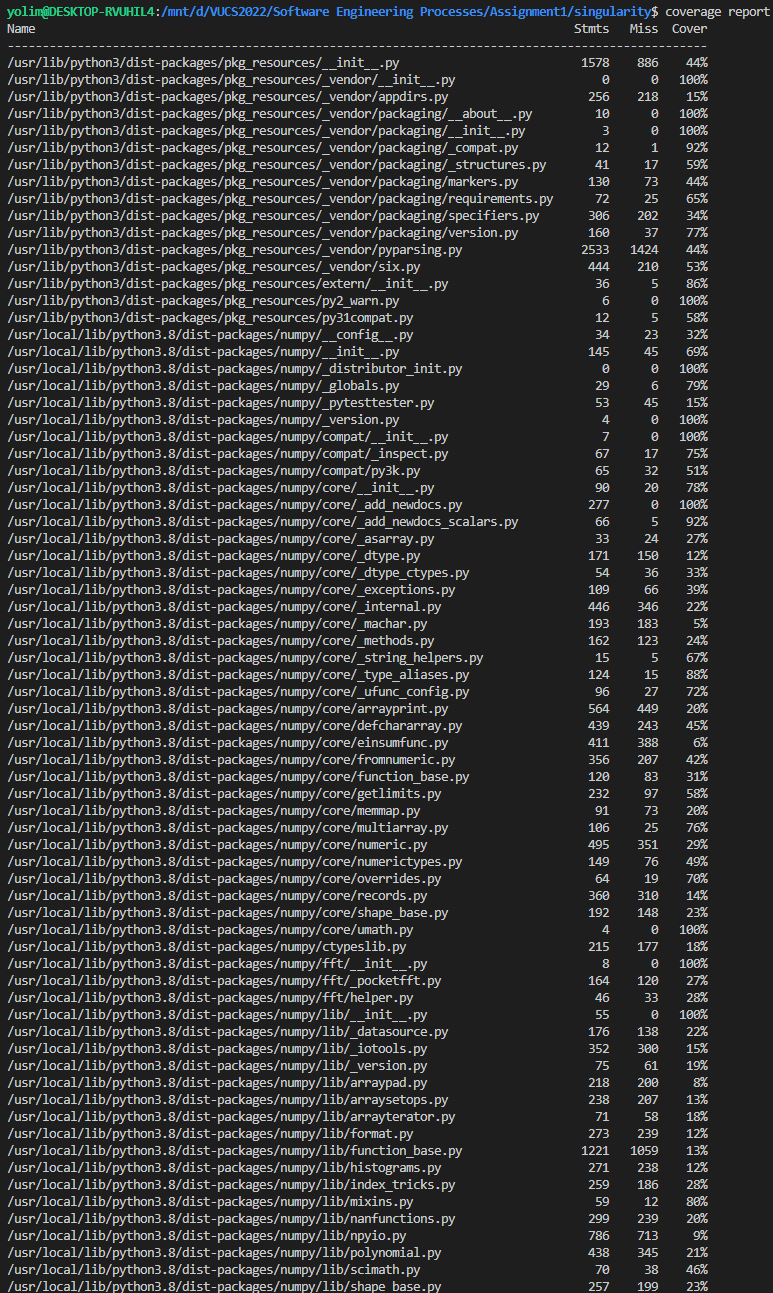
**### Overall**

<Provide a screenshot of the old coverage results by running an existing tool (the same as you already showed above)>



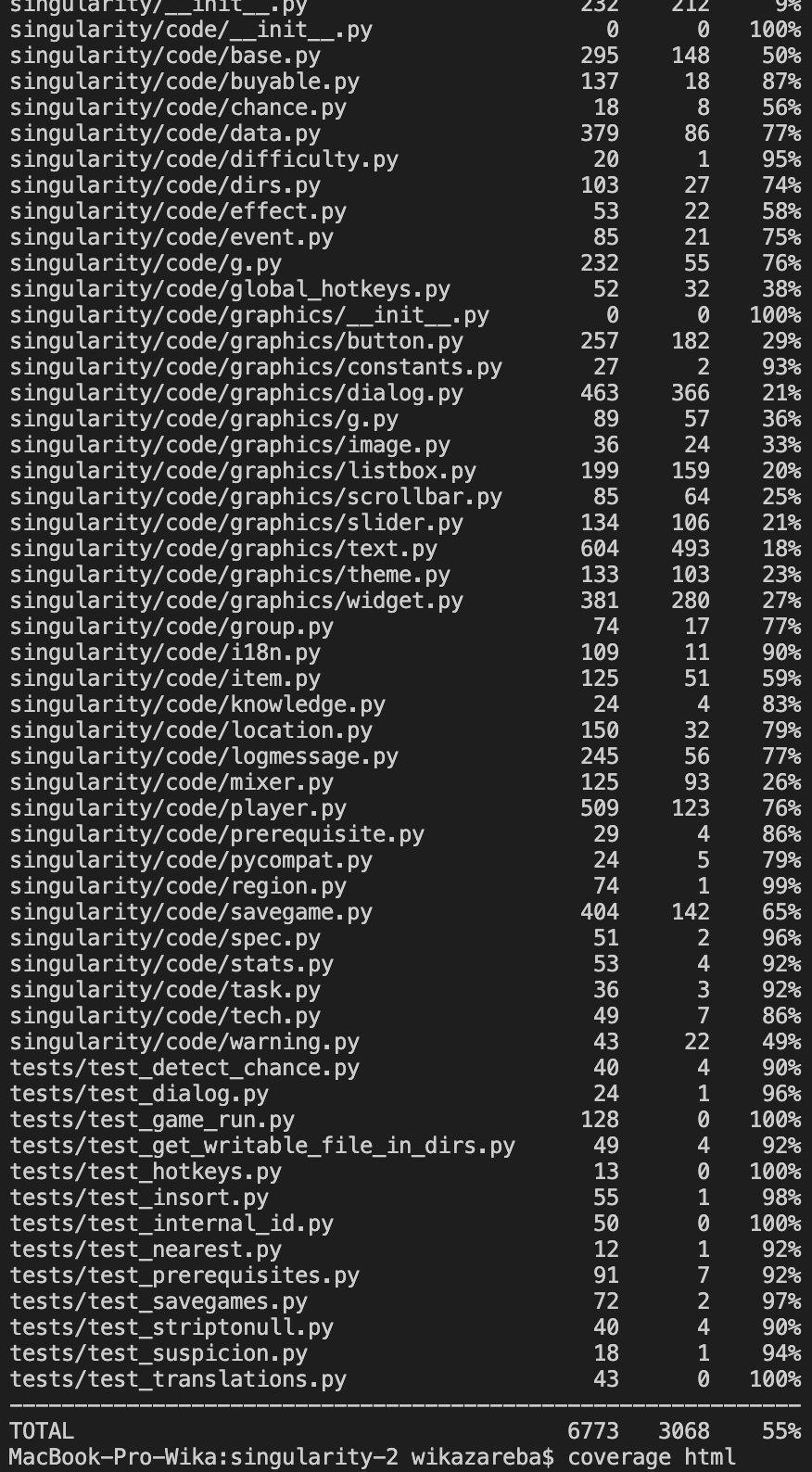
<old coverage results left : window, right : mac OS>

<Provide a screenshot of the new coverage results by running the existing tool using all test modifications made by the group>



New coverage on Window : After the merge, the coverage test showed the same 35% total percentage. However, the newly generated tests we proposed were definitely over 80% after the merge.

When we compared the test results, we found that some tests failed in the system part we didn’t touch.



New coverage result on Mac: 55%

The overall coverage result on Mac system improved from 53% to 55% but all functions chosen are at least 80%.

## Statement of individual contributions

<Write what each group member did>

**Youngchae Lim**

* Functions: ‘strip\_to\_null(a\_string)’, ‘detect\_chance\_to\_danger\_level(detects\_per\_day)’
* Focused on instrumentation and testing these functions from project singularity to gather coverage measurements. Test cases included scenarios such as checking for empty strings, leading spaces, trailing spaces, and different detection counts per day in the project software

**Wiktoria Zaręba**

* Functions: ‘fake\_click(down)’, ‘insort\_right\_w\_key(a, x, lo = 0, hi = None, key = lambda v: v):’
* Responsible for creating new tests for these two test functions, her efforts resulted in a 100% coverage improvement for the first function and 98% coverage for the second function.
* The first function fake\_click was tested in file test\_dialog.py and the second function is tested in the file test\_insort.
* Both test were created (there was no testing provided for any of the function)
* The contribution for the report is all the coverage analysis for 2 chosen functions and the description of the testing as well as the overall coverage of the project before and after

**Yuto Ishihara**

* Functions: ‘get\_writable\_file\_in\_dire(filename, dir\_name, outer\_path=None)’, ‘power\_state\_name(self)’
* Developed test cases to cover all branches of these functions, improving the coverage from 0% to 92% (for the first test) and to 87% (for the second test)
* The contribution of the report is to check everyone’s codes in order to avoid any conflict. Then I figured out that I was accidentally using the same code as another member so I could switch to another function before it's too late. I also checked for grammar mistakes and actually corrected some of them.

**Marika Litwiniak**

* Functions: ‘suspicion\_to\_danger\_level(suspicion)’, ‘nearest\_percent(value, step = 100)’
* Responsible for writing tests for the two chosen functions, and improving both of their coverage from 0% to 100%
* Provided coverage analysis for the chosen functions and contributed in writing the intro and overall part of the project's report

Our group proceeded with this task, and through periodic meetings online/offline, we found projects for the task together and shared testing methods and ideas with each other.