Athelas Interview - Senior Software Development Engineer in Test (SDET)

Alejandro Avella

7-7-2024

The Wordle game has been popularized by the New York Times. Many people around the world play it on a daily basis. The object of the game is simple: Guess a five letter word in less than 5 attempts. For each guess, the user is given feedback using the following color code: “Gray” for a missed letter, “Yellow” for a good guess but in the wrong position and “Green” for a good guess in the correct position. For a sample game, go to the following page. The following sample game for the word “Apple” shows how the user wins in 5 attempts and each time he receives some feedback that helps him to solve the puzzle.

As part of the interview, the interview panel at Athelas asked 3 questions that are answered in this document.

1. Write unit tests for the golden path. Basic unit tests with 100% coverage of the code written. The number of tests is up to you given the time we have left.
    1. Golden path is **check_new_word**
    2. Prove we are 100% covered using index.html output from Pytest
    3. If you would like a different behavior— let that reflect in your test and let that test fail.

I found several problems with the code, I made the modifications in the file *wordle_interview_revised_code.py and described the changes in question 3*

Description on how to run code coverage for the wordle_interview.py.

Installation:

1. **Create a virtual environment**

```
$ python -m venv myvenv
$ source myvenv/bin/activate
```

2. **Install required packages**

```
$ pip install pytest
$ pip install pytest-cov
```

3. **Code Coverage command to be executed from the CLI (note: remove .py from file)**

```
$ pytest --cov=wordle_interview --cov-report=html
```

Reference:

1. Pytest-cov plugin: https://pytest-cov.readthedocs.io/en/latest/

Figures 1 and 2 show the Code Coverage after the Golden Path test cases for the function check_new_word have been added.

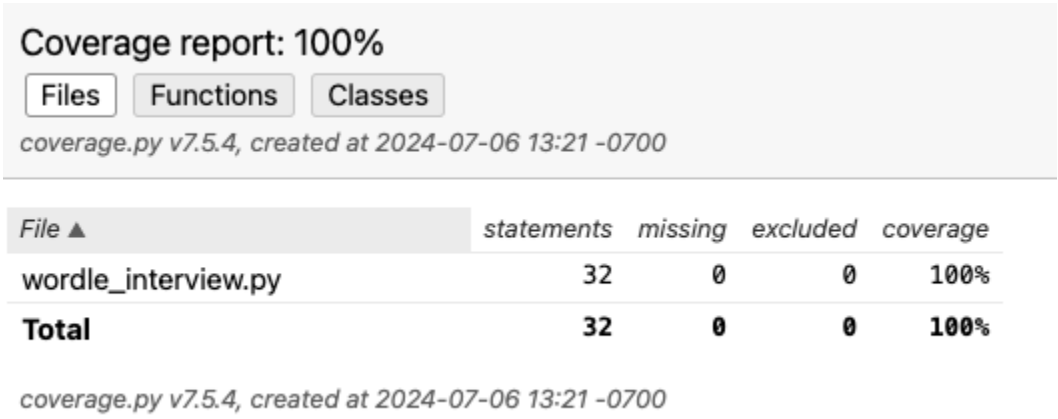

Figure 1 - Sample Code Coverage report

Coverage for **wordle_interview.py**: 100%

32 statements | 32 run | 0 missing | 0 excluded

« prev ^ index » next coverage.py v7.5.4, created at 2024-07-06 13:21 -0700

```
1  from collections import Counter
2  import copy
3
4
5  class WordleGame():
6
7          def __init__(self, answer):
8                  self.max_tries = 6
9                  self.answer = answer
10                 self.current_tries = 0
11
12                 self._internal_representation()
13
14         def _is_game_over(self):
15                 if self.current_tries > self.max_tries:
16                         return True
17                 return False
18
19         def _internal_representation(self):
20                 # create map of index or to char
21                 self.mappings = {}
22                 for i, char in enumerate(self.answer):
23                         self.mappings[i] = char
24
25                 self.counts = Counter(self.answer)
26                 return
27
28         def check_new_word(self, new_word):
29                 self.current_tries += 1
30
31                 if self._is_game_over() is True:
32                         return "Game is Over"
33
34                 deep_copy_counts = copy.deepcopy(self.counts)
35                 res = []
36                 for i, char in enumerate(new_word):
37                         if self.mappings[i] == char:
38                                 res.append("G")
39                         elif char in deep_copy_counts and deep_copy_counts[char] > 0:
40                                 res.append("Y")
41                         else:
42                                 res.append("W")
43                         deep_copy_counts[char] -= 1
44
45                 return "".join(res)
```

Figure 2 - Coverage report after running Golden Path test cases

2. How are you going to test this at scale?
    1. How are you going to test thousands of words and their expected output?
    2. **Please write tests with guesses and expected outputs**. The number of tests is up to you given the time we have left. Also, explain how you built this bank of words and why you chose to test these specific answer/guess combinations to test.
        1. Some example word categories:
            1. Anagrams, duplicate letter

One way to handle anagrams and duplicate letter words is create a comprehensive list of 5 letters for and check in the answer is the list. Otherwise, reject the answer immediately. I have provided a wordlist.txt file with valid words. However, I did not take that approach and added extra test cases. I don't think you need to test thousands of test cases to test scales, just find the different combinations of test cases that test different scenarios.

I identified key scenarios in order to determine the types of words that will effectively test the game logic. This includes anagrams, words with duplicate letters, words with no matching letters, and words that are exactly the same as the answer. Sett the file: ***test_wordle_interview_revised_code.py***

3. Is our current solution correct?
    1. If not, can you propose a better one?
        1. English/pseudo code explanation of better solution:
        2. Actual code:

Using **Github Copilot** coding assistant, I was able to pinpoint that he implementation of the **WordleGame** class has a few issues that affect its functionality:

1. **Incorrect Handling of Yellow (Y) Feedback**: The current logic decrements the count of a character even if it matches exactly (green). This could lead to incorrect "yellow" feedback when a character appears more times in the guess than in the answer.

Problem Identified: The issue here is that the character count is decremented even when a character is correctly matched (green). This could lead to a situation where a character that appears more times in the guess than in the answer might incorrectly receive a "yellow" feedback instead of "wrong" because it's count was reduced prematurely.

Impact: This affects the accuracy of feedback for guesses, potentially misleading the player about the presence of certain letters in the answer.

2. **Handling of Guesses Longer or Shorter than the Answer**: The method **check_new_word** does not handle cases where the length of **new_word** is different from the length of the answer. This could lead to index errors or incorrect feedback.

3. **Game Over Logic**: The method **_is_game_over** only checks if the number of tries is greater than the maximum allowed. It does not consider a scenario where the player wins by guessing the word correctly before reaching the maximum number of tries. The game over condition related to winning is implicitly handled within the check_new_word method through the feedback mechanism. When a player guesses the correct word, the check_new_word method generates a feedback string of "G" characters equal in length to the answer, indicating a correct guess for each letter. In this case, it could be considered to add the output “You Won!”

4. **Character Count Decrement Logic:** The logic to decrement the character count (**deep_copy_counts[char] -= 1**) is executed regardless of whether the character matches (green) or is present in the answer (yellow). This should only happen if the character is present in the answer but not in the correct position.

Problem Identified: This suggestion also identifies an issue with the decrement logic of character counts but focuses on the broader logic of when counts should be decremented. It points out that the decrement should only occur if the character is present in the answer but not in the correct position (yellow feedback), not just whenever a character matches or is present.

Impact: The core issue here is ensuring the game logic accurately reflects the rules of Wordle, specifically how guesses are evaluated. Incorrect decrementing could lead to both incorrect "yellow" feedback and potentially incorrect "wrong" feedback if the counts are not managed correctly.

Fixes:
To address these issues, the following adjustments were made:

1. Modify the **check_new_word** method to ensure it correctly handles the logic for green and yellow feedback, taking into account the number of times a character appears in both the guess and the answer.
2. Add logic to handle guesses that are not the same length as the answer.
3. Update the game over logic to consider a win condition.
4. Ensure that character counts are only decremented when appropriate, avoiding decrementing for green matches or when a character is not in the answer.

This revised version includes a **two-pass approach** for marking characters as green first and then handling yellow or wrong characters, ensuring accurate feedback based on the actual and guessed words.

How to make sure coverage is adequate?
On 7-8-2024, I had the interview with the Athela's engineer and he proposed a test case that I didn't add in the initial test suite and it was failing in the original implementation. I did not consider the case where there are words with double letters where the match is in the second letter. Consider the following test case:

```
def test_word_double_letter_words_where_the_match_is_in_the_second_letter(self):  #
This test case fails in the original implementation
   game = WordleGame("APPLE")
   result = game.check_new_word("HELLO")
   assert result == "WWYGW", "Expected result: 'WWYGW', Actual result:
{}".format(result)
```

This test case fails in the initial implementation as follows:

```
Expected :'WWYGW'
Actual   :'WYYGW'
```

And it works correctly in the revised code:

```
Expected :'WYYGW'
Actual   :'WYYGW'
```

Let's analyze why the test case fails:

1. Guess "H" as the first letter, it **is not** in the secret word, then feedback should be "W", which is correct.
2. Guess "E" as the second letter, it **is** in the secret word, then feedback should be "Y", **which is NOT CORRECT.**
3. Guess "L" as the third letter, it **is** in the secret word, then feedback should be "Y", which is correct.
4. Guess "L" as the fourth letter, it **is** in the secret word, then feedback should be "G", which is correct.
5. Guess "O" as the fifth letter, it **is not** in the secret word, then feedback should be "W", which is correct.

So, then how come I said that the code was fully tested and there was a case that was not considered. And not only that, but what other scenarios I might be missing. And please note that code coverage said that with test cases added for the revised suite there was 100% coverage and I was certifying the release. Not a good thing for the release! A escape to the field.

How is it possible to say that I tested the code completely? That is a good question! Should I test all possible 5-letter words that appear in the dictionary? Is that too much, or are we forced to do that? Notice that I thought I had everything covered and the interviewer was able to break my test plan with the word "Hello"!. How can we add all the test scenarios? I am not sure!

Appendix A:

In the **source code**, you can find the following files

1. wordle_interview.py - File provided during the interview
2. test_wordle_interview.py - Test file for question #1 (before the code change)
3. wordle_interview_revised_code.py - Revised code for question #3
4. test_wordle_interview_revised_code.py - Test cases for the revise code for question #3
5. htmlcov_original - index.html with code coverage with test cases added in question #1
6. htmlcov - index.html with code coverage with test cases added in question #3
7. fixes.diff - a file showing the differences implemented for question #3
8. fixes_colors.png - a file showing the differences for question #3 with colors