# Developer documentation

**Written by Xuan Koo (20125282)**

# Program overview

Guess-My-Word is a fun game that involves users guessing a 5-letter word that is selected randomly for each play. The catch is that users only get 6 attempts per round before they lose, but for each guess they get hints on how close their guess is based on coloured tiles that indicates if they guessed the correct letter, correct position or both.

# Algorithm (function) overview

There’s quite a lot of functions involved in this program, which I will attempt to break down in the following sections. Here’s just a rough overview of each:

1. **help()**: This just gives players the instructions to play the game.
2. **get\_player\_name():** This prompts player to enter their name before playing the game so the game could appear more personable.
3. **game\_stats(player\_name, attempt, target, is\_correct):** This function is to record the game statistics after the game ends with relevant information like attempts taken to win/lose the game, player’s name etc.
4. **play()**: This is the main loop of the game where player will keep making guesses until they either win/lose. This incorporates a lot of the other functions.
5. **get\_valid\_words(file\_path=ALL\_WORDS)**: Retrieves the list of valid words from the all\_words,txt from the word bank.
6. **get\_target\_word(file\_path=TARGET\_WORDS)**: Retrieves a random target word from the target\_words,txt from the word bank.
7. **ask\_for\_guess(valid\_words)**: Asks the player to input a guess and validates it against the list of valid words.
8. **score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT):** Scores the guess based on its similarity to the target word.
9. **is\_correct(score)**: Checks if the guess is correct based on the score.
10. **format\_score(guess, score)**: This makes the score output from the *“score\_guess”* function appear “pretty” and easier to understand through the form of coloured tiles.
11. **main(test=False)**: The main function of the program in which the game runs.

# Algorithm (function) pseudo code

1. **Help():**

Print "Welcome to the Guess-My-Word! The objective of the game is to guess a 5-letter word with as little attempts as possible, you have 6 attempts."

Print "🟩 - Indicates a correct letter in the wrong position"

Print "🟨 - Indicates a correct letter in the correct position"

Print "⬜ - Indicates a letter not in the target word"

1. **get\_player\_name():**

Ask player to input name

Read input from player

If input is not blank:

Set player name as the input

Else:

Set player name to default “anonymous”

Return player’s name

1. **game\_stats(player\_name, attempt, target, is\_correct):**

If is\_correct is True:

Set result to “Win”

Else:

Set result to “Lose”

Create statistics txt with the following format:  
“Player:” player\_name

“Target Word:” target

“Attempts:” attempt

“Result:” result

Open the file statistics.txt and append.

Write statistics into the file

Close file once done.

1. **play():**

Display game instructions.

Set word\_of\_the\_day to a randomly chosen word from the target words list

Feed the list of valid words that’s available as “guess words” for players to input from the word bank

For each attempt in range(MAX\_ATTEMPTS=6):

Display the current attempt number

Ask the player to input a guess word

Make sure that the guess is a valid 5-letter word

Score the guess against the word of the day

# function score\_guess (guess word, target word):

Intialise empty list to store scores

# Do a for loop for each letter in the guess word

For (each character in guess word) and (each character in target word):

if guess word = target word 100%:

Break

Elif guess letter matches any target letter in the right position:

return 🟩(2)

Elif guess letter matches target letter but in wrong position:

return 🟨(1)

Else:

Return ⬜(0)

If the guess is correct (all letters in the right position):

Display a congratulatory message

Exit the loop

If the loop completes without a correct guess:

Display a message indicating the player has run out of attempts

Reveal the word of the day

1. **get\_valid\_words(file\_path=ALL\_WORDS)**:

Open file

Read file

Split file into list of words

Close file

Return list of words

1. **get\_target\_word(file\_path=TARGET\_WORDS)**:

Open file

Read file

Split file into list of words

Close file

Choose random word from list and return that word.

1. **ask\_for\_guess(valid\_words)**:

Loop:

Ask player to input guess

Read input an convert to lower case

If guess is in the list of valid words:

Return guess

Else:

Print message telling player to enter VALID 5-letter word.

1. **score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT):**

Create empty list to store the score.

For each character position i in the guess:

Let guess\_char be the character at position i in the guess.

Let target\_char be the character at position i in the target\_word.

If guess\_char matches target\_char:

Append EXACT to the score list.

Else if guess\_char is in target\_word:

Append MISSPLACED to the score list.

Else:

Append MISS to the score list.

Return the score list.

1. **is\_correct(score)**:

If score is equal to 2,2,2,2,2

Return True

Else:

Return False

1. **format\_score(guess, score)**:

Create an empty string to store the formatted guess.

Create an empty string to store the formatted score.

For each character g in the guess:

Append g followed by a space to the formatted guess.

For each score s in the score:

If s is EXACT:

Append '🟩 ' to the formatted score.

Else if s is MISSPLACED:

Append '🟨 ' to the formatted score.

Else:

Append '⬜ ' to the formatted score.

Return the formatted guess and formatted score.

1. **main(test=False):**

If test mode is enabled:

Run doctests for the module

Otherwise, start the game by calling the play() function

If \_\_name\_\_ == '\_\_main\_\_':

Call the main() function

# Algorithm (function) flow chart

**help()**:

Not applicable for a flowchart

**get\_player\_name():**

**A diagram of a computer game

Description automatically generated**

**game\_stats(player\_name, attempt, target, is\_correct):**

**A diagram of a function

Description automatically generated**

**play()**:

A diagram of a diagram

Description automatically generated

**get\_valid\_words(file\_path=ALL\_WORDS)**:

Not really applicable, it’s pretty straightforward and doesn’t have diverging paths/decisions. Hope it’s okay if it’s not included.

**get\_target\_word(file\_path=TARGET\_WORDS)**:

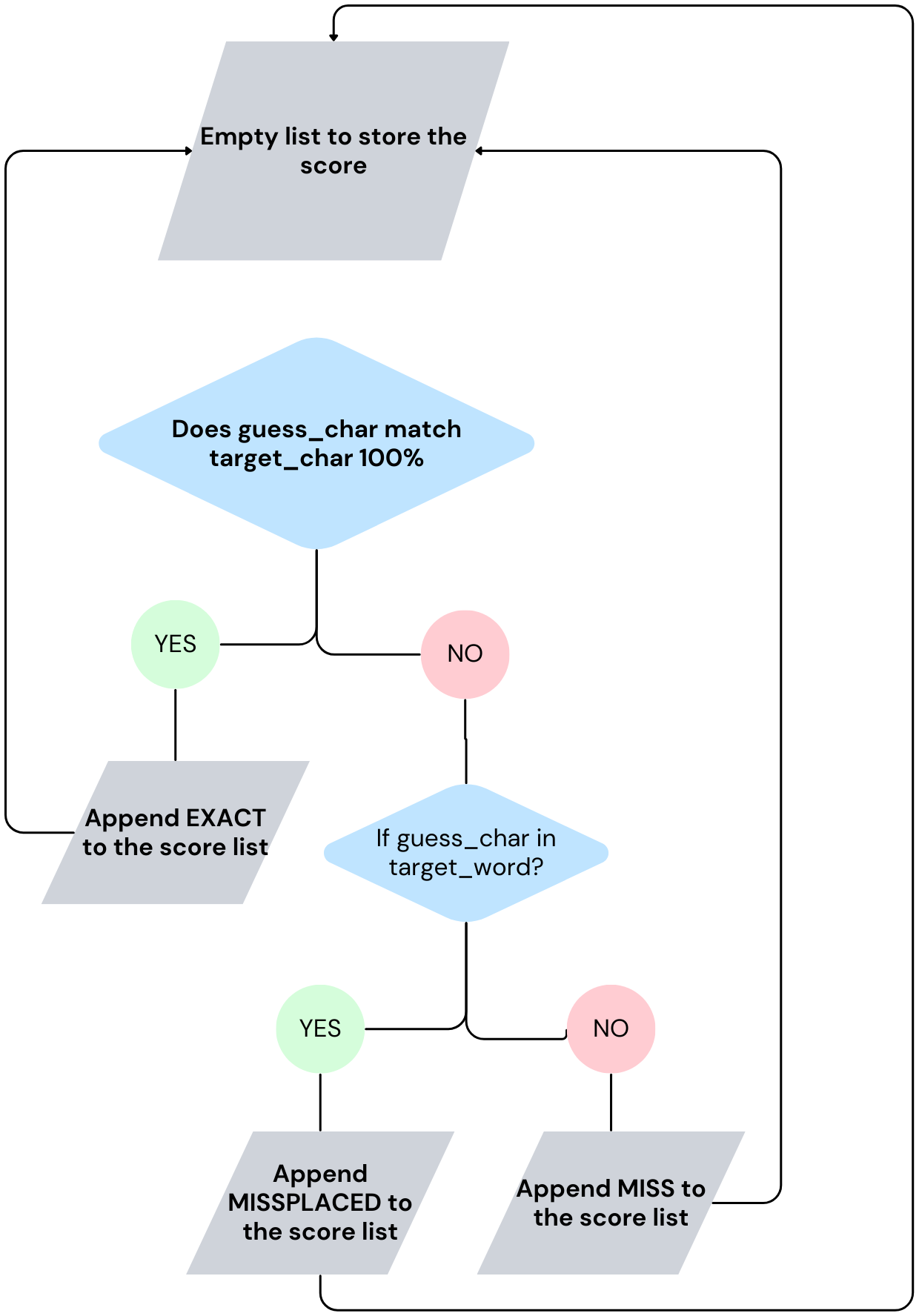
Not really applicable, it’s pretty straightforward and doesn’t have diverging paths/decisions. Hope it’s okay if it’s not included.

**ask\_for\_guess(valid\_words)**:

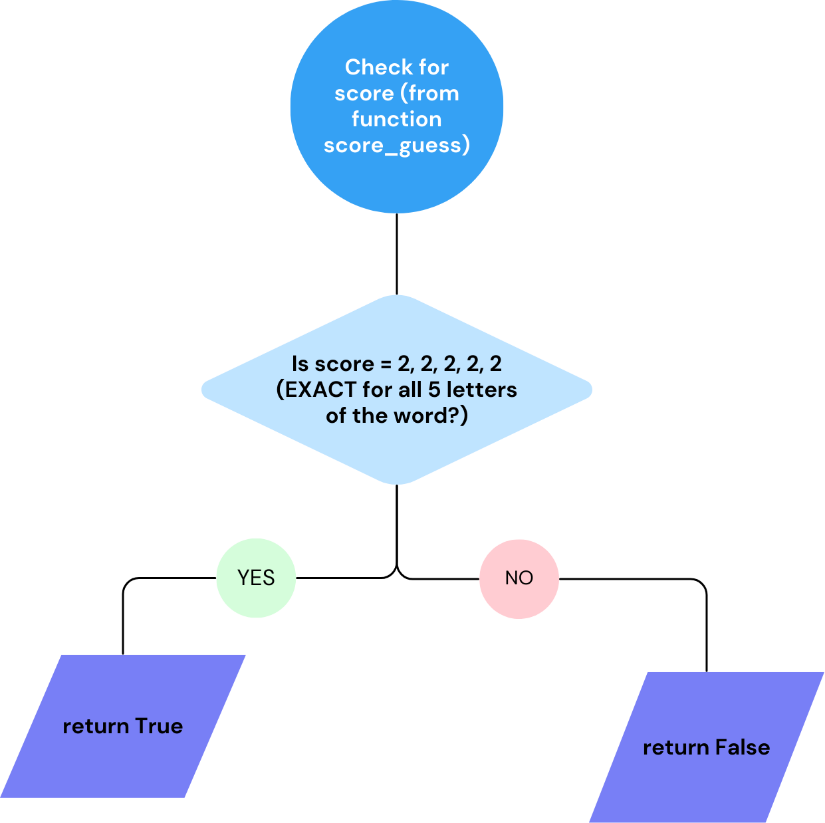
A diagram of a diagram

Description automatically generated

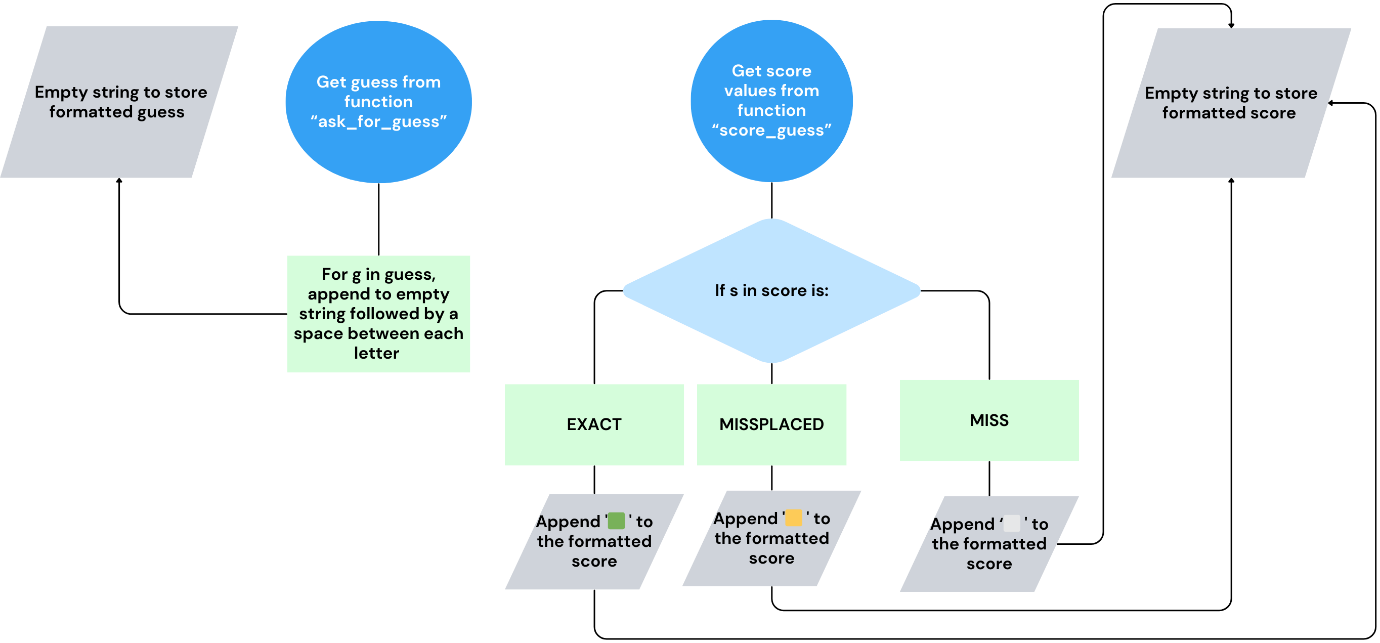
**score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT):**

****

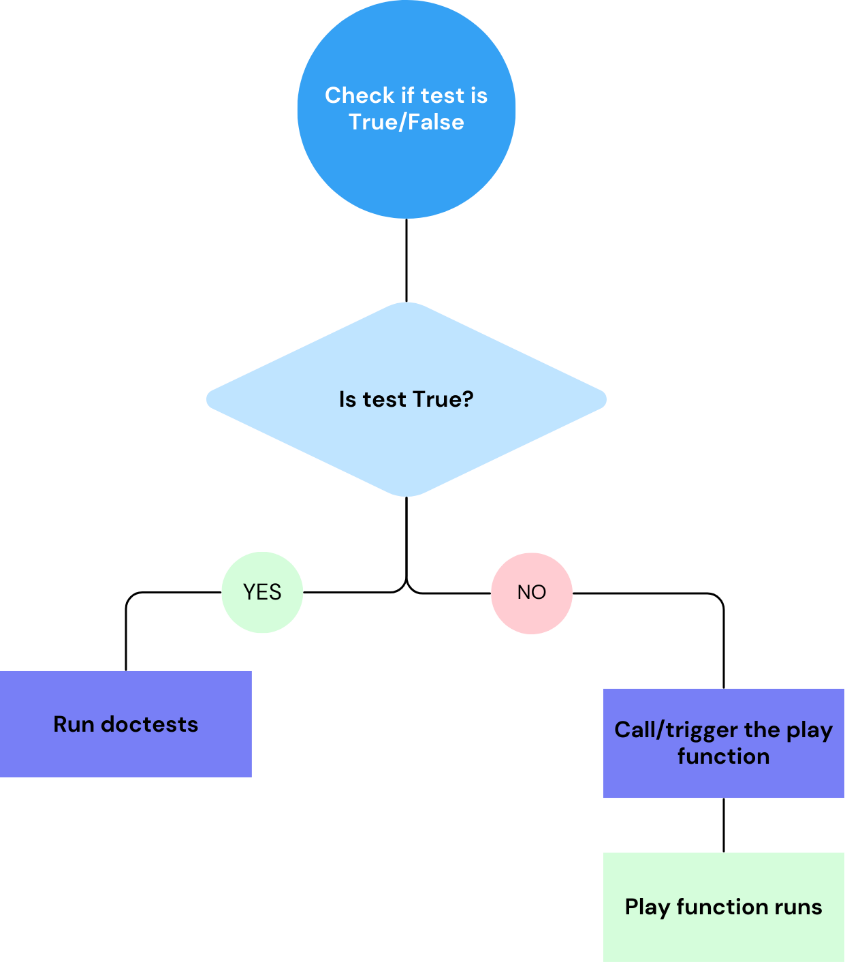
**is\_correct(score)**:



**format\_score(guess, score)**:

****

**main(test=False):**



# v. Algorithm (function) test data

I’ve established three different test cases in test\_cases.py and will list them down here:

1. **Hello/world**

Target word= world

Guess word= hello

Expected output: ⬜ ⬜ 🟨 🟩 🟨

1. **Ankle/apple**

Target word= ankle

Guess word= apple

Expected output: 🟩 ⬜ ⬜ 🟩 🟩

1. **Batch/right**

Target word= ankle

Guess word= apple

Expected output: ⬜ ⬜ ⬜ 🟨 🟨

# vi. Algorithm (function) coding

Please refer to the codes in the wordle.py for the full details:

[wordleproject/wordle.py at main · 20125282/wordleproject (github.com)](https://github.com/20125282/wordleproject/blob/main/wordle.py)

# vii. Program testing

A screenshot of a computer

Description automatically generated

This is an example of the function I’ve used for the 1st test case:

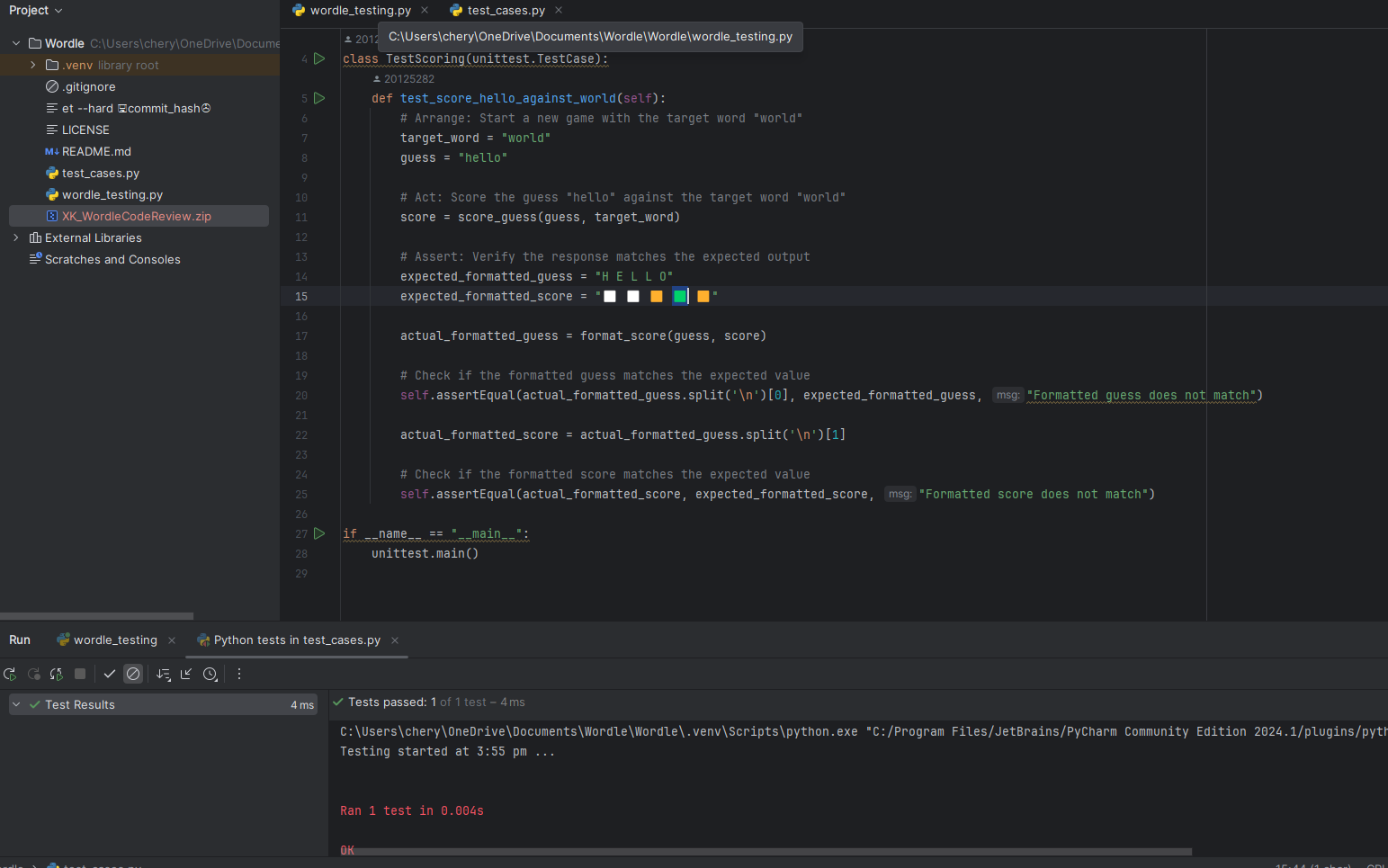
def test\_score\_hello\_against\_world(self):  
 # Arrange: Start a new game with the target word "world"  
 target\_word = "world"  
 guess = "hello"  
  
 # Act: Score the guess "hello" against the target word "world"  
 score = score\_guess(guess, target\_word)  
  
 # Assert: Verify the response matches the expected output  
 expected\_formatted\_guess = "H E L L O"  
 expected\_formatted\_score = "⬜ ⬜ 🟨 🟩 🟨"  
  
 actual\_formatted\_guess = format\_score(guess, score)  
  
 # Check if the formatted guess matches the expected value  
 self.assertEqual(actual\_formatted\_guess.split('\n')[0], expected\_formatted\_guess, "Formatted guess does not match")  
  
 actual\_formatted\_score = actual\_formatted\_guess.split('\n')[1]  
  
 # Check if the formatted score matches the expected value  
 self.assertEqual(actual\_formatted\_score, expected\_formatted\_score, "Formatted score does not match")

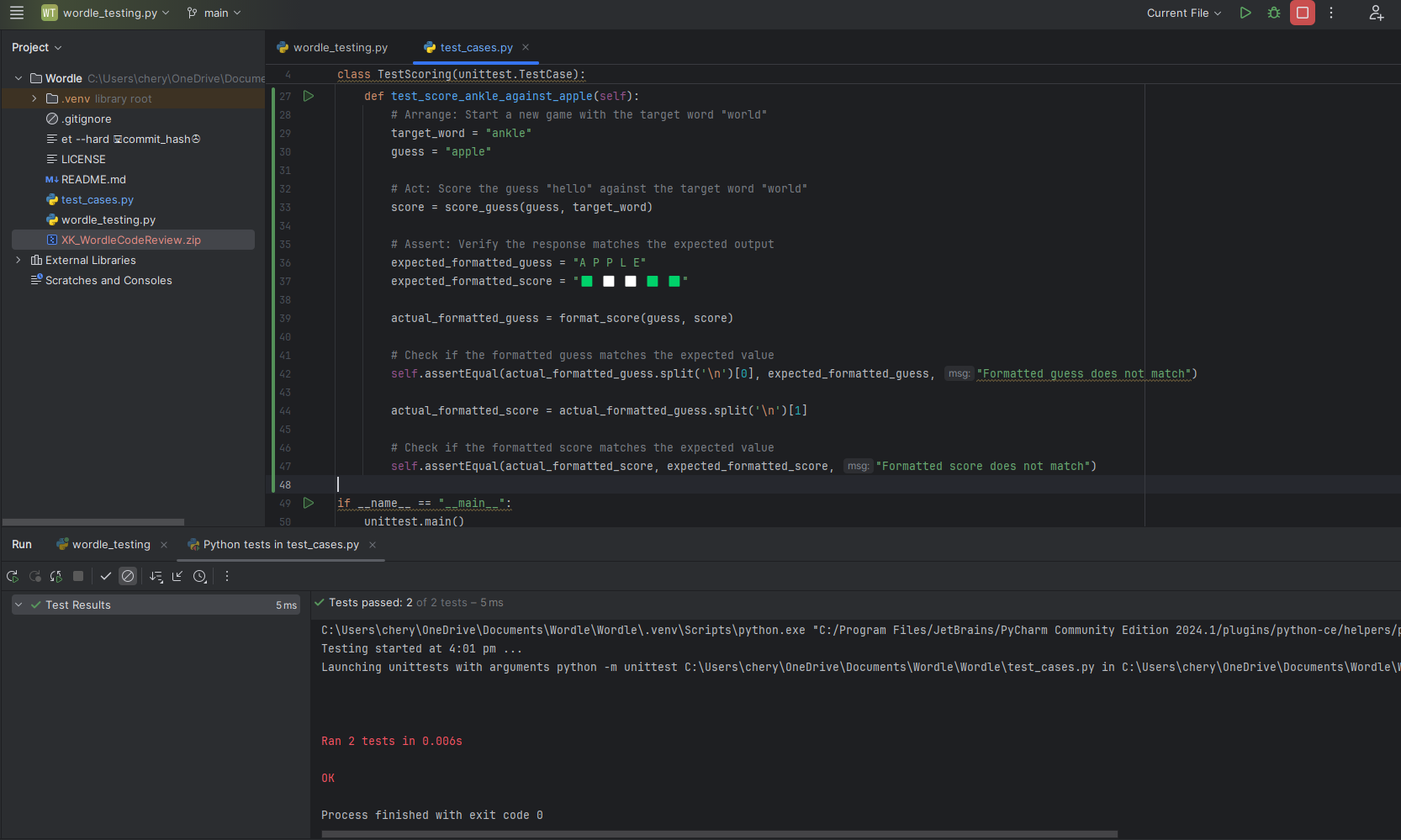
This is the base of the rest of test case 2 and 3. The only difference is renaming the function accordingly, then changing the following:

* Change the target word
* Change the guess word
* Change the expected formatted guess
* Change the expected formatted score

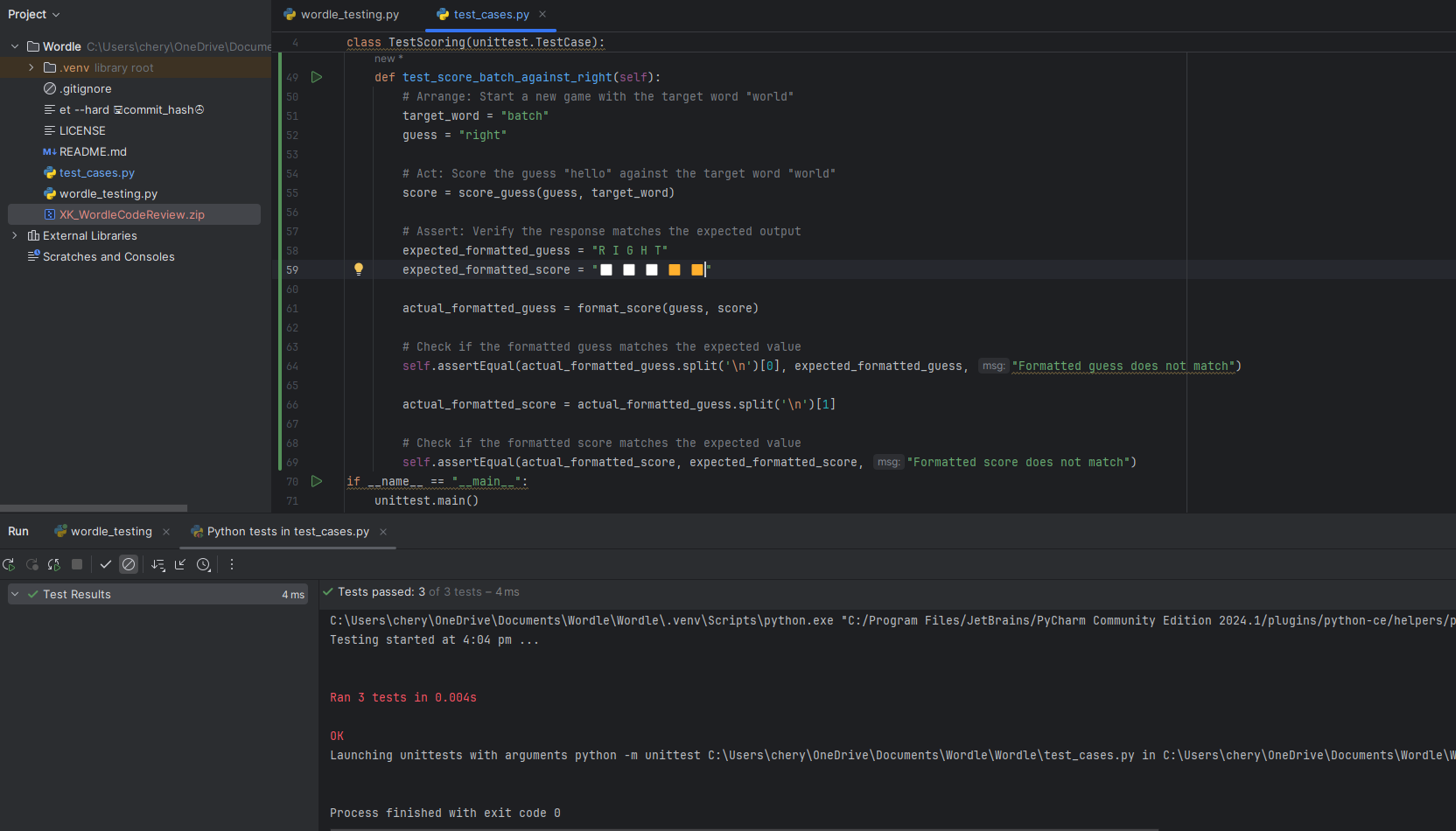
It should work when once you run it.

This is test case 1 (hello/world) and the subsequent result:



This is test case 2 (ankle/apple) and the subsequent result:  


This is test case 3 (batch/right) and the subsequent result:



Here’s the full program:

import unittest  
from wordle\_testing import format\_score, score\_guess  
from wordle\_testing import MISS, MISSPLACED, EXACT  
  
class TestScoring(unittest.TestCase):  
 def test\_score\_hello\_against\_world(self):  
 # Arrange: Start a new game with the target word "world"  
 target\_word = "world"  
 guess = "hello"  
  
 # Act: Score the guess "hello" against the target word "world"  
 score = score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT)  
  
 # Assert: Verify the response matches the expected output  
 expected\_formatted\_guess = "h e l l o"  
 expected\_formatted\_score = "⬜ ⬜ 🟨 🟩 🟨"  
  
 actual\_formatted\_guess = format\_score(guess, score, EXACT, MISSPLACED)  
  
 # Check if the formatted guess matches the expected value  
 self.assertEqual(actual\_formatted\_guess.split('\n')[0], expected\_formatted\_guess, "Formatted guess does not match")  
  
 actual\_formatted\_score = actual\_formatted\_guess.split('\n')[1]  
  
 # Check if the formatted score matches the expected value  
 self.assertEqual(actual\_formatted\_score, expected\_formatted\_score, "Formatted score does not match")  
  
 def test\_score\_ankle\_against\_apple(self):  
 # Arrange: Start a new game with the target word "world"  
 target\_word = "ankle"  
 guess = "apple"  
  
 # Act: Score the guess "hello" against the target word "world"  
 score = score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT)  
  
 # Assert: Verify the response matches the expected output  
 expected\_formatted\_guess = "a p p l e"  
 expected\_formatted\_score = "🟩 ⬜ ⬜ 🟩 🟩"  
  
 actual\_formatted\_guess = format\_score(guess, score, EXACT, MISSPLACED)  
  
 # Check if the formatted guess matches the expected value  
 self.assertEqual(actual\_formatted\_guess.split('\n')[0], expected\_formatted\_guess, "Formatted guess does not match")  
  
 actual\_formatted\_score = actual\_formatted\_guess.split('\n')[1]  
  
 # Check if the formatted score matches the expected value  
 self.assertEqual(actual\_formatted\_score, expected\_formatted\_score, "Formatted score does not match")  
  
 def test\_score\_batch\_against\_right(self):  
 # Arrange: Start a new game with the target word "world"  
 target\_word = "batch"  
 guess = "right"  
  
 # Act: Score the guess "hello" against the target word "world"  
 score = score\_guess(guess, target\_word, MISS, MISSPLACED, EXACT)  
  
 # Assert: Verify the response matches the expected output  
 expected\_formatted\_guess = "r i g h t"  
 expected\_formatted\_score = "⬜ ⬜ ⬜ 🟨 🟨"  
  
 actual\_formatted\_guess = format\_score(guess, score, EXACT, MISSPLACED)  
  
 # Check if the formatted guess matches the expected value  
 self.assertEqual(actual\_formatted\_guess.split('\n')[0], expected\_formatted\_guess, "Formatted guess does not match")  
  
 actual\_formatted\_score = actual\_formatted\_guess.split('\n')[1]  
  
 # Check if the formatted score matches the expected value  
 self.assertEqual(actual\_formatted\_score, expected\_formatted\_score, "Formatted score does not match")  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

This is the result of the testing, all test cases passed:

A screenshot of a computer

Description automatically generated

# ix. Documentation using docstrings

Docstrings for each functions in the program:

**def get\_player\_name():**

"""

Prompt the player to enter their name. If no name is provided, default to 'anonymous'.

Returns: str: The player's name.

"""

**def game\_stats(player\_name, attempt, target, is\_correct):**

""" Log game statistics to a file. """

**def help():**

"""Provides instructions on how to play the game/listing rules.

"""

**def is\_correct(score):**

"""Checks if the score is entirely correct and returns True if it is

>>> is\_correct((🟩 🟩 🟩 🟩 🟩))

True"""

**def get\_valid\_words(file\_path=ALL\_WORDS):**

"""returns a list containing all valid words which could be entered as guesses. Any words outside of this list would be an invalid guess.

"""

**def get\_target\_word(file\_path=TARGET\_WORDS):**

"""Picks a random word from a file of words, this is to be used as the target word for the game.

"""

**def ask\_for\_guess(valid\_words):**

"""This is where we ask user for their input (guesses). This input will be validated against the list of valid words.

"""

**def score\_guess(guess, target\_word):**

"""

Score the guess against the target word by comparing each letter of guess word to letter of target word and returns tuples:

- 0 : Letter not found in the target word.

- 1 : Letter found in the target word but in the wrong position.

- 2 : Letter found in the target word and in the correct position.

"""

**def format\_score(guess, score):**

"""Formats a guess with a given score as output to the terminal.

This is my output based on the value of the tuples:

* 0: ⬜
* 1: 🟨
* 2: 🟩

"""

**def main(test=False):**

"""Main function to start the game"""

**def play():**

"""

This function is where the main flow of the game gets executed, such as selecting a word of the day, prompting the player for guesses, scoring the guesses, and formatting the output of said guesses.

"""