Hangman Project

Table of Contents

[Test-driven Development (TDD) 1](#_Toc48833754)

[Development before beginning test cases 1](#_Toc48833755)

[Requirements 1](#_Toc48833756)

[Test Cases 1](#_Toc48833757)

[Refactoring 3](#_Toc48833758)

[Code Smells 3](#_Toc48833759)

# Test-driven Development (TDD)

This project has been built following a Test-Driven Development approach. This approach utilizes planned test cases that must be passed before continuing to the next test case. Throughout the project, code will be developed to pass each test case one after another, however it is likely that with the development of code for new test cases, older tests will have to be run again. When running older tests some issues may arise that indicate code smells. These will then be addressed and solutions developed for them.

## Development before beginning test cases

I want this application to utilize a UI for display and input, therefore before I begin developing to pass test cases, I decided to construct a blank Tkinter window.

## Requirements

The project test cases are derived from the following requirements.

1. One word will be generated randomly
2. Player will be presented with a number of blank spaces representing the missing letters the player needs to find.
3. If the player’s chosen letter exists in the answer, then all places in the answer where that letter appear will be revealed.
4. Every time the player guesses a letter wrong, the player’s life will be deducted.
5. The player must find the missing word before the player’s life becomes zero.

## Test Cases

The following is the series of test cases in order of their initial development. There is a likely chance that code will need to be refactored throughout development which will be documented here.

**Test Case: 1**

**Description:** Generates a random English word to Tkinter window.

**Test Steps:**

1. Generates a random word (initially printed in console)
2. Check if the generated word is a correct English word in lowercase
3. Word is displayed in Tkinter window

**Expected Result:** The word will be printed in console and displayed to Tkinter window.

(Screenshot of code and App)

**Actual Result:** The word displayed

**Pass / Fail:** Pass

**Refactoring Test Case 1**

**Test Case: 2**

**Description:** Make letters display hidden and show only Underlines

**Test Steps:**

1. Make the word invisible
2. Display an underscore for each letter

(Screenshot of code and App)

**Actual Result:** The word displayed

**Pay / Fail:** Fail

1. Make words invisible and display as underlines to the screen
   1. An underscore is shown for each and every letter
2. User can input letters 1 at a time to display

Only allow letters

Future work:

Base word length on difficulty

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Test Case | Expected Result | Actual Result | Pass / Fail | Additional Notes |
| 1 | Generate a random word | Correct word is generated and displayed correctly |  |  |  |
|  |  |  |  |  |  |

# Refactoring

## Code Smells

A code smell is a front-end / surface indication that usually corresponds to a deeper problem in the system. This section will clearly detail found code smells and the implementation of their fix.