PRT582 – Software Unit Testing Report

Project: Hangman Game using TDD and Automated Unit Testing Tool in Python

Department of Information Technology, Charles Darwin University

Sydney Campus

Student Name: Krishna Sapkota

Student Id: S395795

Unit Coordinator: Dr. Charles Yeo

Professor: Dr. Muhammad Rana

Date: 06/09/2025

Introduction

1.1 Project Summary

Hangman, a classic word-guessing game, requires players to guess letters one at a time to reveal a hidden word or phrase. This project implements Hangman in Python and follows the Test-Driven Development (TDD) approach by checking for correctness and robustness with automated unit tests (Beck, 2002).

The game consists of:

- Two levels of difficulty - Basic (single word) and Intermediate (phrases).
- Life-based play, where the player has a maximum of 7 lives.
- Hints - players can use a limited number of hints to reveal letters.
- Timer - There are 15 seconds for each guess. If the time runs out, the game is over.
- GUI - A GUI was created using Tkinter, which features a drawing of a hangman, a display of the word, wrong guesses, and score (Van Rossum & Drake, 2009).

1.2 Goals and Objectives

The main goals of the project are:

- Implement the Hangman game in Python, while following good design principles that allows for clean, modular code (Van Rossum & Drake, 2009).
- Apply TDD principles - write unit tests before coding feature to establish functional operations (Beck, 2002).
- Incorporate a GUI with visual feedback for correct and incorrect guesses.
- Implement a timer which will end the game when it reached zero.
- Document the development process and provide evidence of the use of unit testing automated tools.

1.3 Rationalization of Programming Language

The programming language chosen for this project was Python for the following reasons:

- Simple and understandable syntax allowing fast development (Van Rossum & Drake, 2009).
- Extensive standard libraries (e.g. Tkinter, for GUI, unittest, for Unit Testing) (Python Software Foundation, 2024).
- Excellent support for factorial programming and modular software design.

- Widely adopted in academia and software engineering providing much greater ease of demonstrating TDD. (Beck, 2002).

1.4 Automated Unit Testing Tool

This project uses the unit test module that comes built-in with Python for automated testing (Python Software Foundation, 2024).

Reasons to use unit test:

- Allows automated checking of each game functionality.
- Facilitates TDD by allowing for test cases to be written before the functionality is coded (Beck, 2002).
- Creates a clean report of tests that can easily be added to documentation.

Scope of tests:

- Correct letter guesses will correctly update the game state.
- Incorrect letter guesses will reduce lives correctly.
- Hints will show letters and deduct one life.
- Game will correctly identify when the win/loss condition has been met (Krekel et al., 2024).

1.5 Features Implemented

1. Selected word/phrase from a list of dictionaries.
2. GUI showing:
    - Letters masked by underscores (_)
    - Incorrect letter guesses
    - Hangman character (head, torso, arms, legs)
    - Lives as hearts
    - Players score
3. Timer per guess (15 second timer)
4. Hints for revealing letters.
5. Game loop:
    - Will run until a player win, loses or exits.
6. Play again option provided at end of game.

2. Process

2.1 Development Methodology

The Hangman game was developed based on a process called "Test-Driven Development" or "TDD." The TDD process includes these basic steps:

1. Write a Test Case - Describe expected behavior of the feature.
2. Run the Test - Must initially fail as the appropriate functionality has not been implemented.
3. Implement Functionality - Write the code so that it passes the test.
4. Refactor - Change the code to be "better" while keeping the test passing.
5. Repeat - Continue for all features needed to have a fully functioning game.

2.2 Tools and Environment

- Programming Language: Python 3.13
- IDE: Visual Studio Code / PyCharm
- Unit Testing Framework: unit test
- GUI Library: Tkinter
- Version Control: Git/GitHub

2.3 Implementation of Features and Test Cases

2.3.1 Game initialization

Goal: To start a new game with the correct number of lives remaining, guessed letters equal to zero, and a valid word/phrase that is appropriate for the selected level.

Test Cases:

Test Case ID	Description	Input	Expected Result	Actual Result	Status
TC001	Initialize basic level game	Level = Basic	lives = 7, guessed_letters = {}	lives = 7, guessed_letters = {}	Pass
TC002	Initialize intermediate level game	Level = Intermediate	lives = 7, guessed_letters = {}	lives = 7, guessed_letters = {}	pass

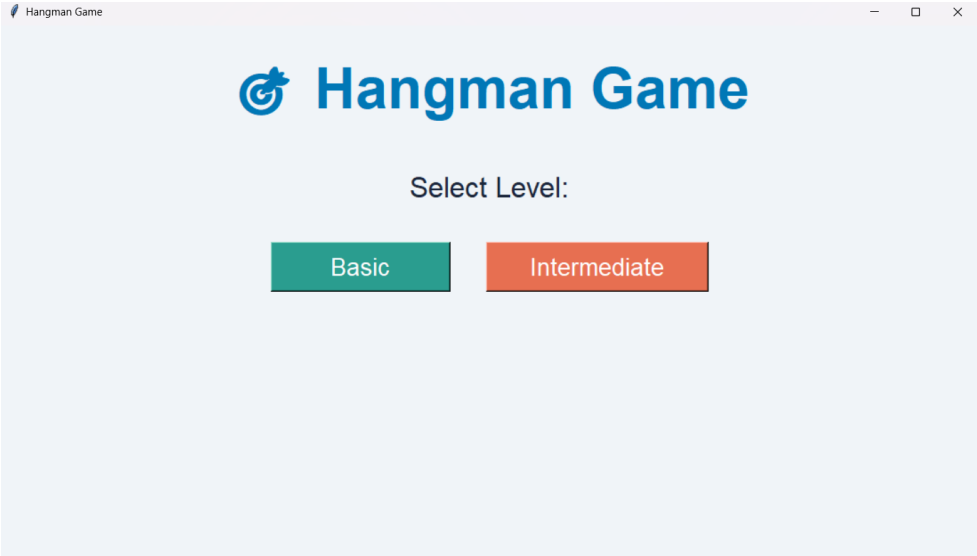

Fig 1 : Splash screen of the game

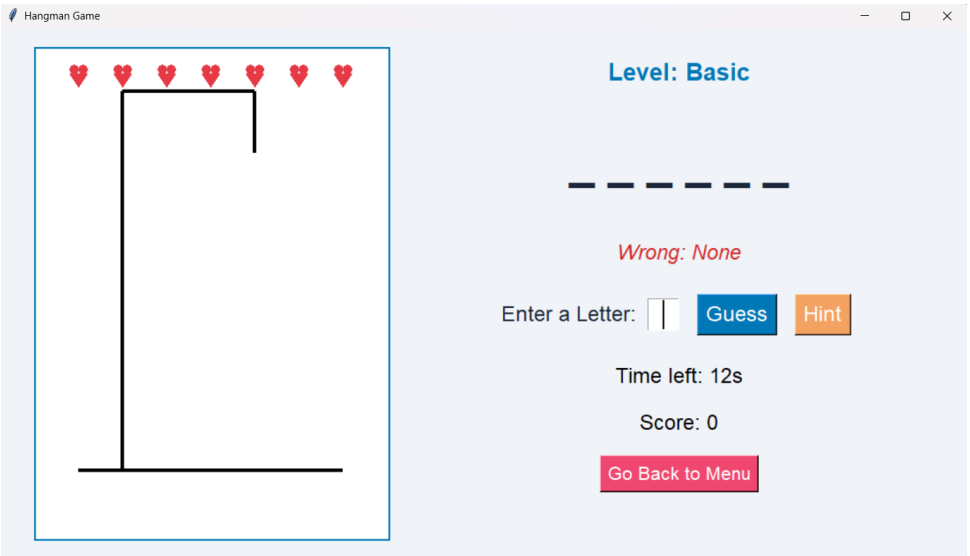

Fig 2 : Basic level of the game

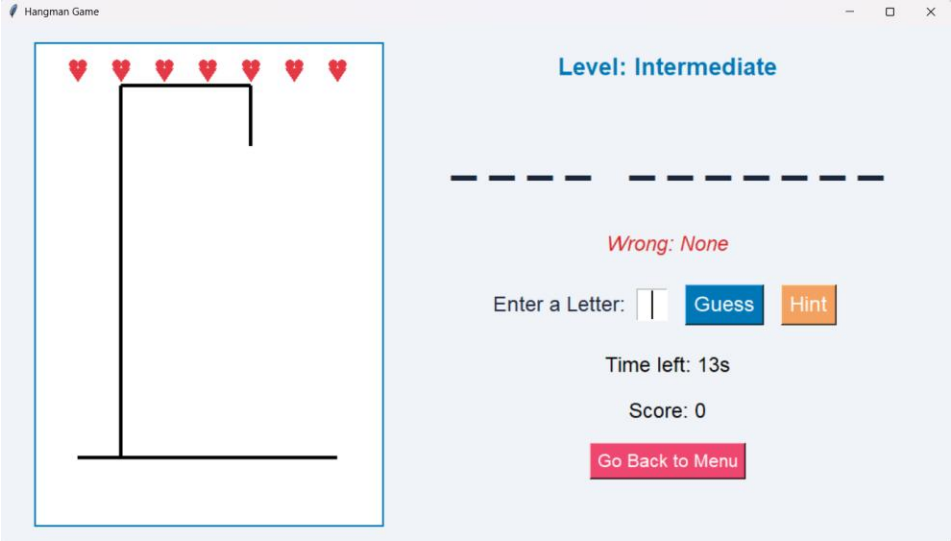

Fig 3 : intermediate level of the game

2.3.2 Letter Guessing

- **Objective**: Correctly update game state for player guesses.

Test Case:

Test Case ID	Description	Input	Expected Result	Actual Result	Status
TC003	Correct letter guess	Letter = "B"	Letter revealed in masked word	Masked word updated with "B"	Pass
TC004	Wrong letter guess	Letter = "A"	lives decremented by 1, wrong_letters updated	lives = 6, wrong_letters = ["A"]	Pass
TC005	Repeated correct letter	Letter = "B"	No change in lives	lives = 6, masked word unchanged	Pass
TC006	Repeated wrong letter	Letter = "A"	No further lives deducted	lives = 6, wrong_letters unchanged	Pass
TC007	Invalid input	Letter = "1"	Error message pop up, Game terminates	Error message in dialogue box popped up	Pass

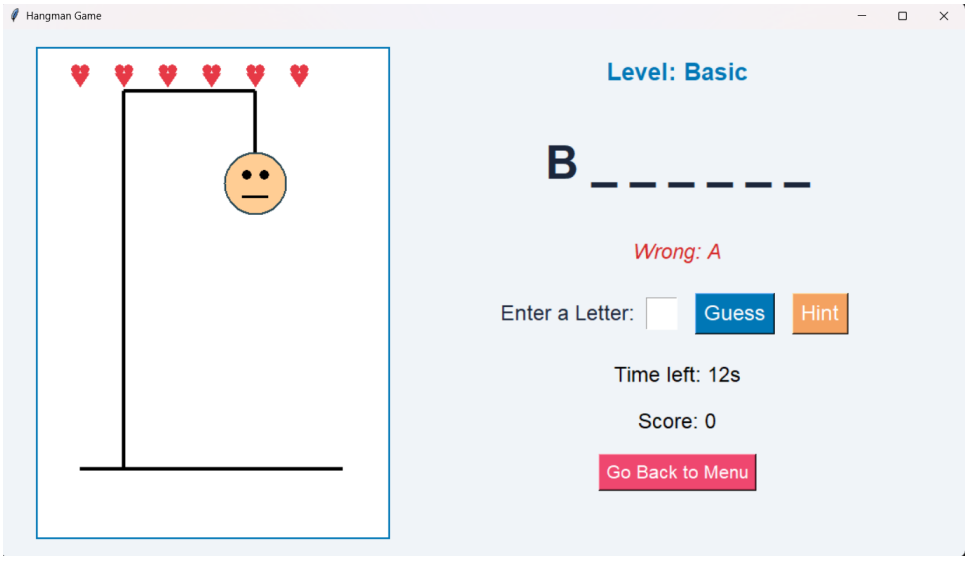

Fig 4 : correct letter B revealed in Masked word

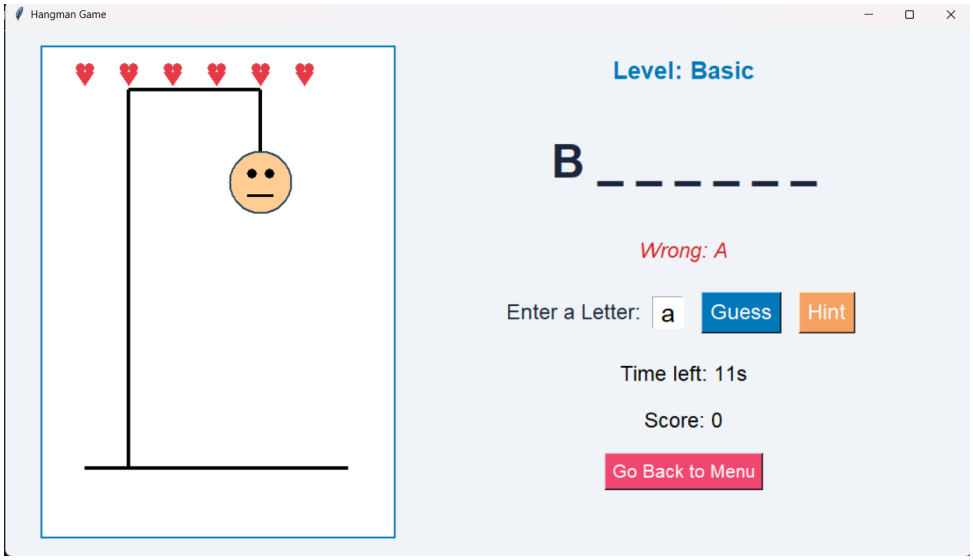

Fig 5: incorrect letter A inputed second time in the Placeholder but no lives deducted

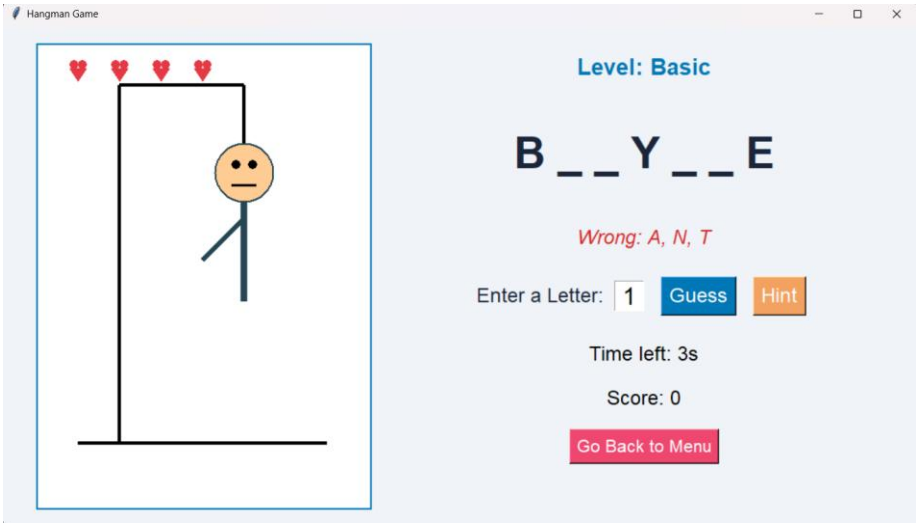

Fig 6 : incorrect character 1 typed in the Placeholder

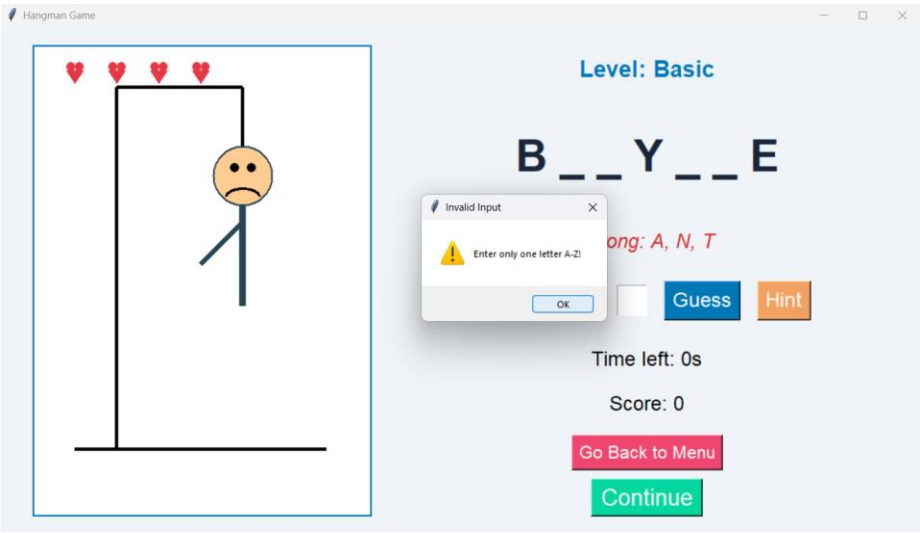

Fig 7: Incorrect message displayed after putting the numeric value

2.3.3 Hint Functionality

- **Objective**: Reveal a random letter, decrementing one life and tracking hints used.

Test Case:

Test Case ID	Description	Input	Expected Result	Actual Result	Status

TC008	Use hint with hints left and lives > 1	Hint requested	Random letter revealed, lives decremented by 1, hints_used incremented	Letter "E" revealed, lives = 5, hints_used = 1	Pass
TC009	Use hint with no hints left or lives ≤ 1	Hint requested	Error message shows	Dialogue box popped up with proper error message	pass

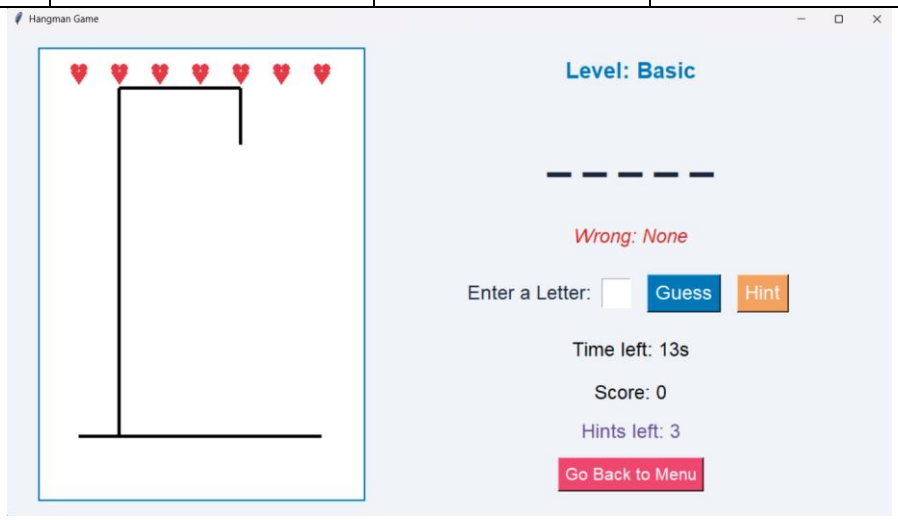

Fig 8: Hint count shows 3

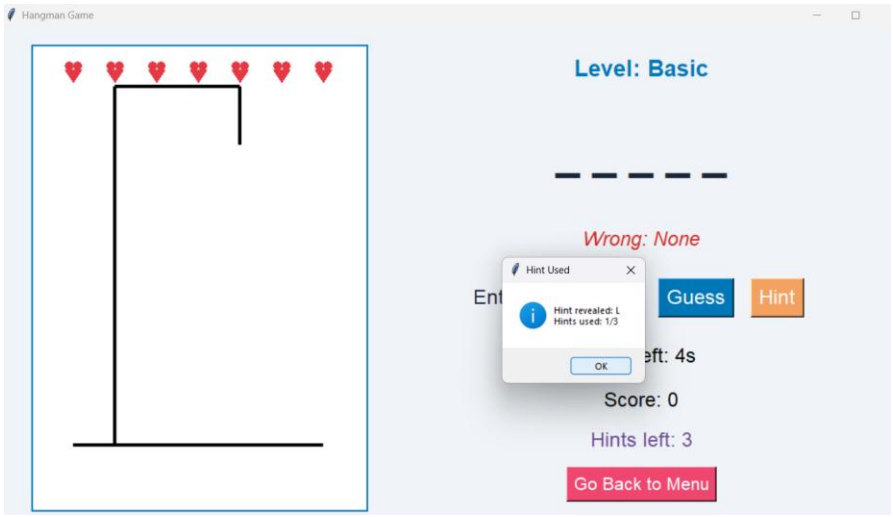

Fig 9: Hint Used and the letter L shows in the dialogue box

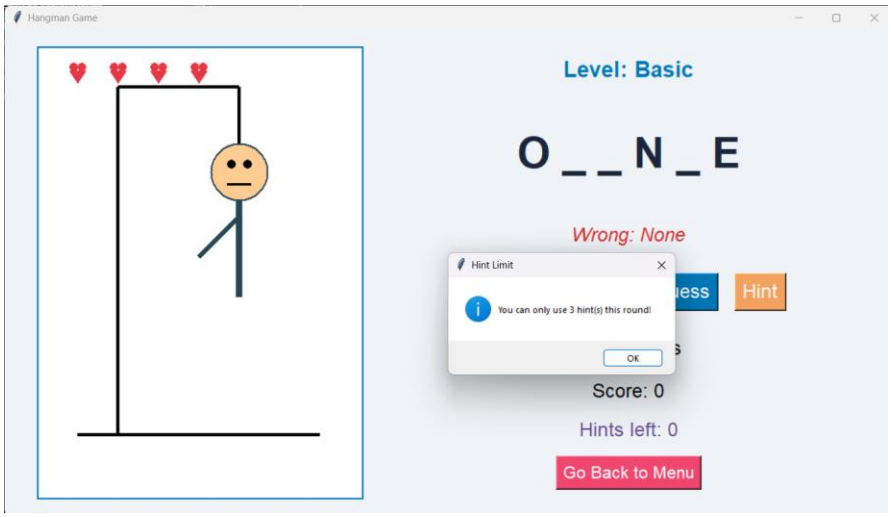

Fig 10: All hints used and the hint count is 0 and when user try another hint the error popped up

2.3.4 Win/Loss Conditions

- **Objective**: Detect when the game has been won or lost.

Test Case ID	Description	Input	Expected Result	Actual Result	Status
TC010	Player guesses all letters	All letters guessed	won = True, game stops, victory message	won = True, victory message shown	Passed
TC011	Player loses all lives	lives reach 0	lost = True, game stops, game over message	lost = True, game over message shown	Passed
TC012	Timer reaches 0	15s elapsed	lost = True, game stops, game over message	lost = True, timer expired message shown	Passed

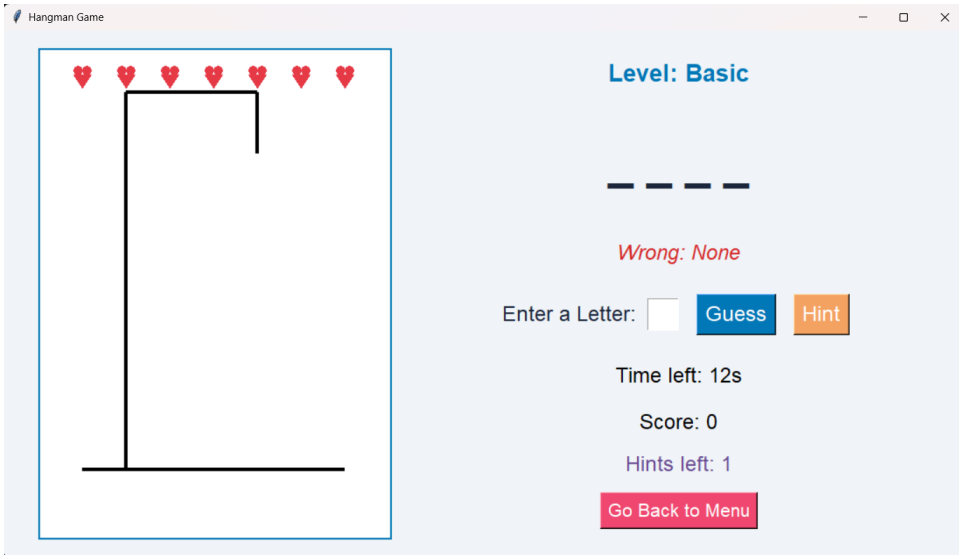

Fig 11: Player started guessing the word with full lives

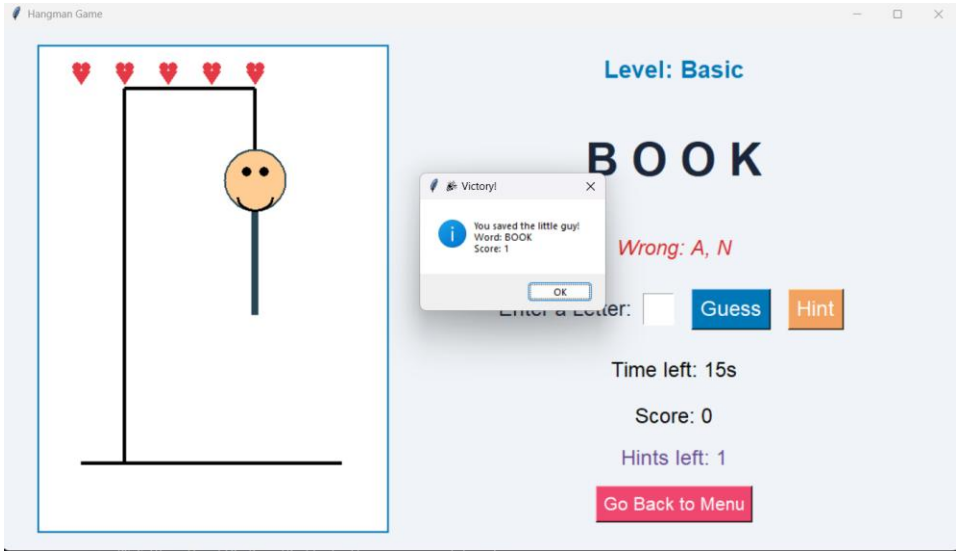

Fig 12: Player guessed the full word and the victory message popped up

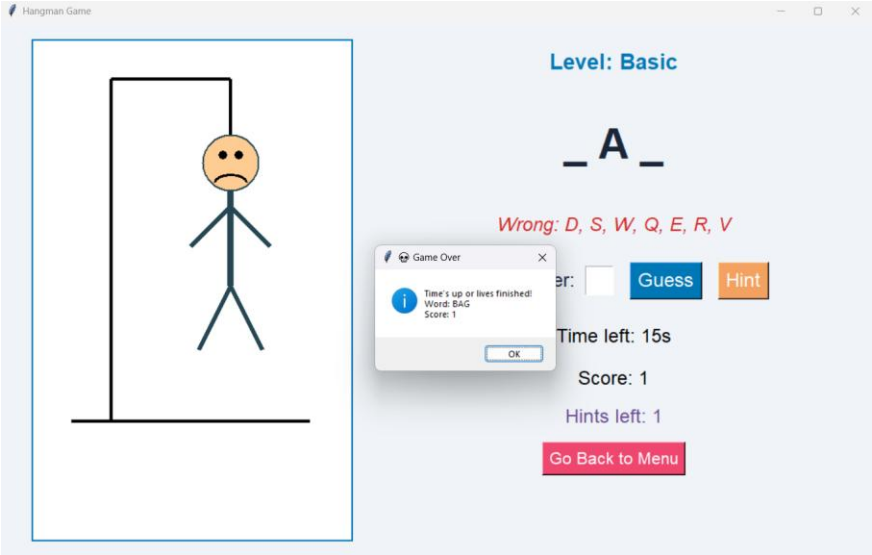

Fig 13: Player lost all lives and the game finished message popped up

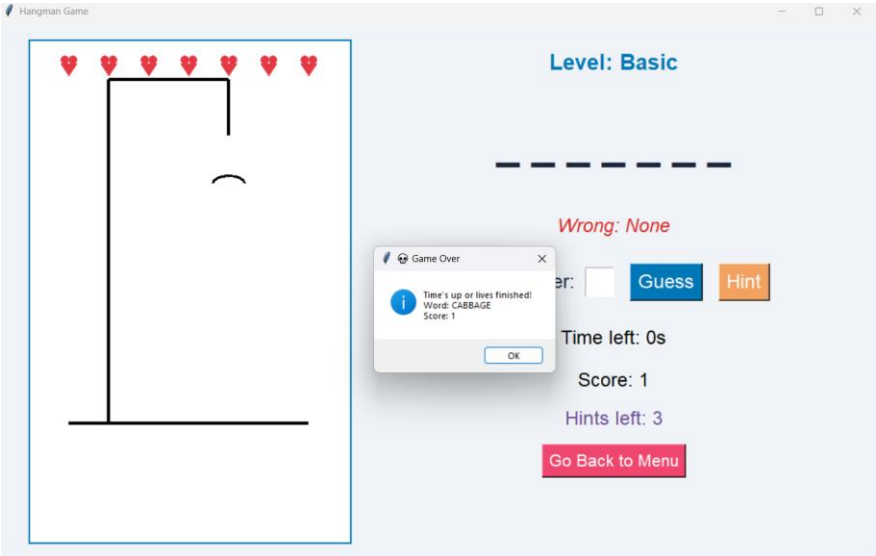

Fig 14: Player got the times up message in the dialogue box

2.3.5 Timer Functionality

- **Objective**: 15-second countdown for each guess. Game ends if time reaches 0.

Test Cases:

Test Case ID	Description	Input	Expected Result	Actual Result	Status
TC013	Timer countdown	Game running	Timer decrements 1 every second	Timer decremented correctly	Passed
TC014	Timer reaches 0	15s elapsed without input	Times up message pop up, lives deducted	Message popped up and lives deducted	Passed

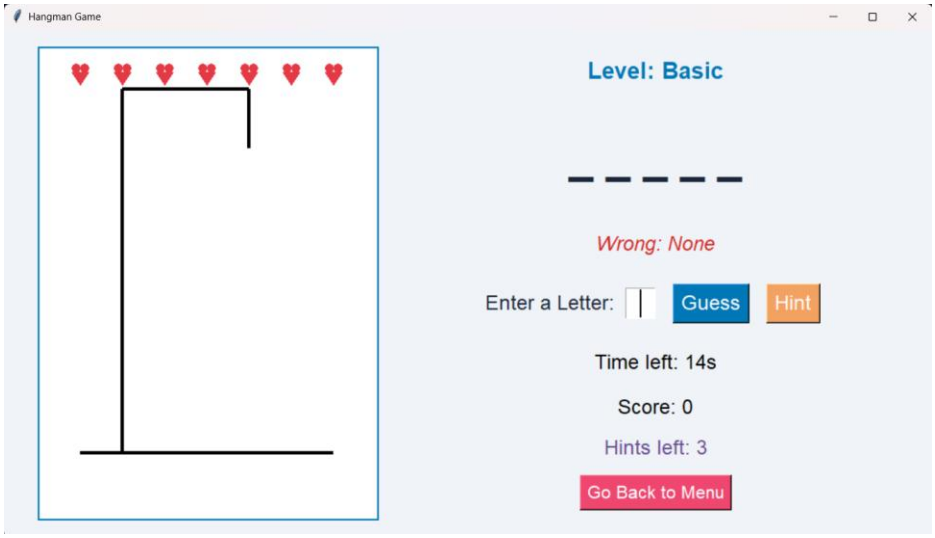

Fig 15: The timer countdown started from the begining

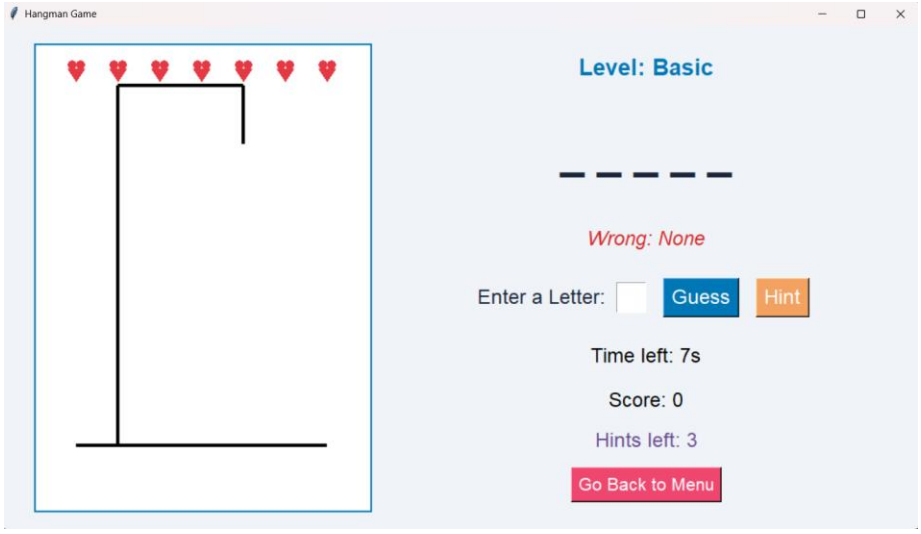

Fig 16: Timer reached 7sec and decrementing by 1 sec

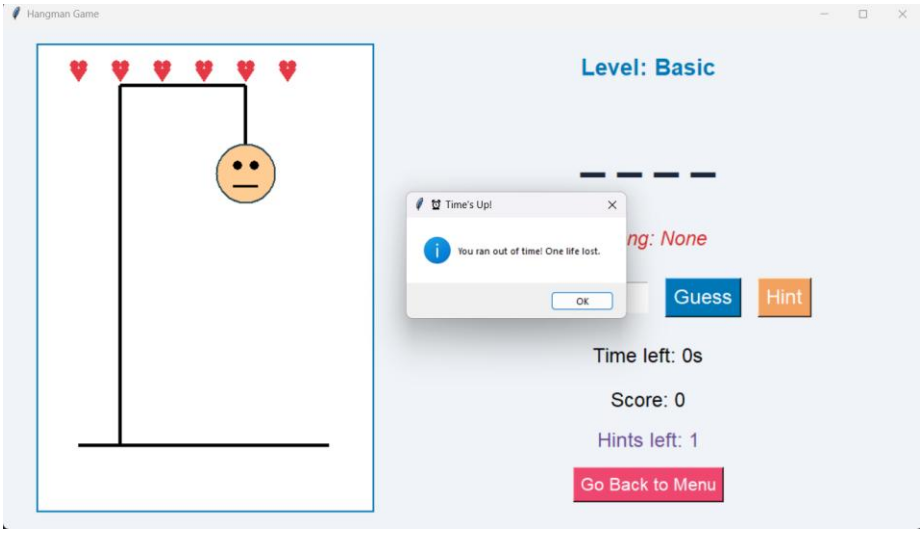

Fig 17: Timer ended and the desired message popped up

2.3.6 GUI Elements

Goal: Visuals offered for improved player experience

Elements Added:

- Beginning screen with level selection
- Game canvas with hangman drawing
- Masked word and guessed letters
- Display wrong letters
- Lives displayed with hearts
- Score Label
- Play again button

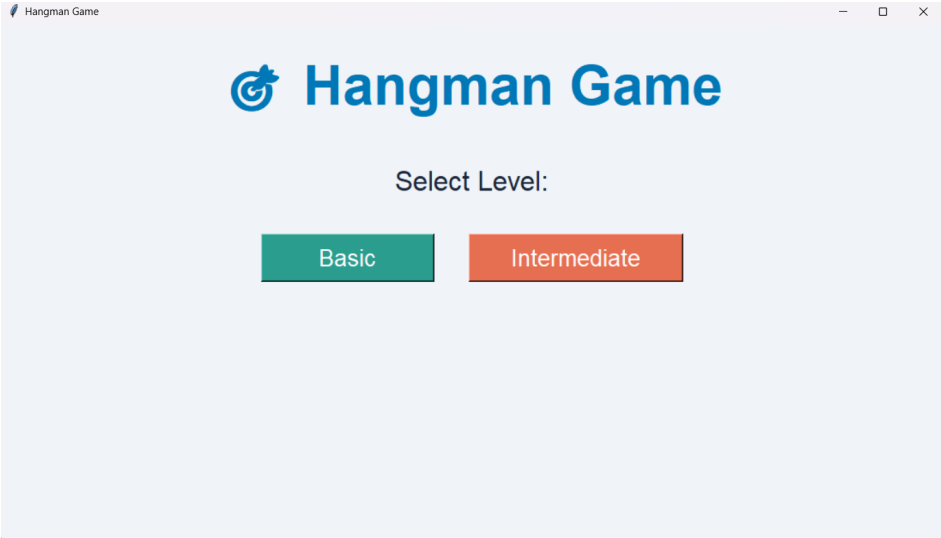

Fig 18: Start screen with the level selection

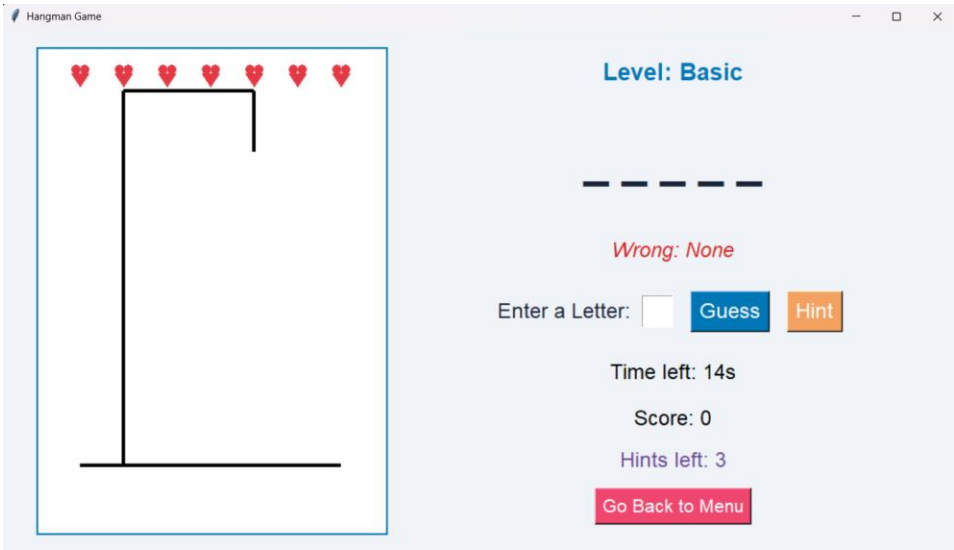

Fig 19: UI for the game with 7 lives in there

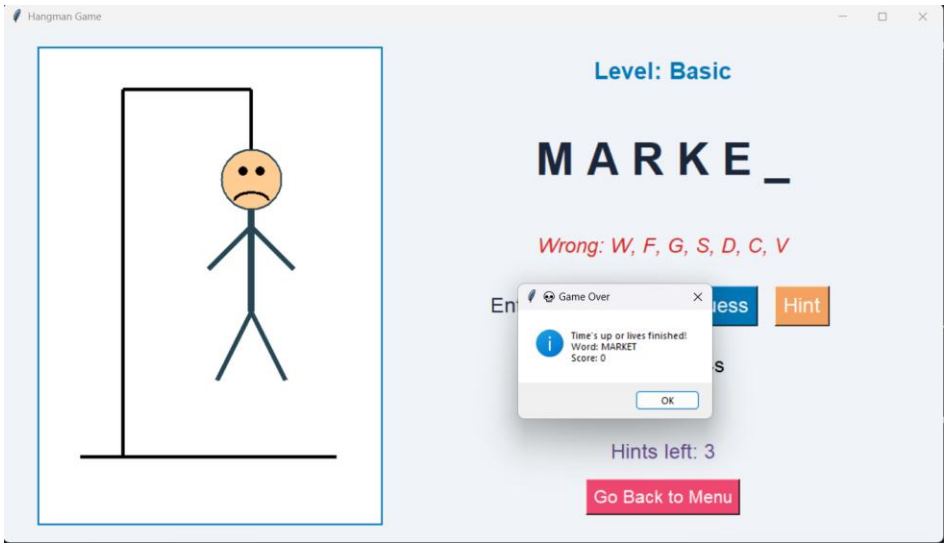

Fig 20: sticky man hanging when user guessed all the wrong

2.4 Unit Testing Evidence

To validate the Hangman game performed as expected, automated unit tests were developed using both unittest and pytest test frameworks. The tests validated the following:

- Correct letter guesses properly reveal the letters of the masked word.
- Incorrect letter guesses lost lives and updated wrong letter guesses.
- Repeated letter guesses did not penalize the user.
- Invalid inputs (anything except a letter a-z or A-Z) were ignored and handled correctly.
- Winning and lose conditions were correctly detected.

The following process is to capture evidence for unit testing:

1. First, open a terminal/command prompt in the directory containing the project.

2. Run the test file in Python:

- (py test_hangman.py)

3. Inspect the output or the test results in the terminal.

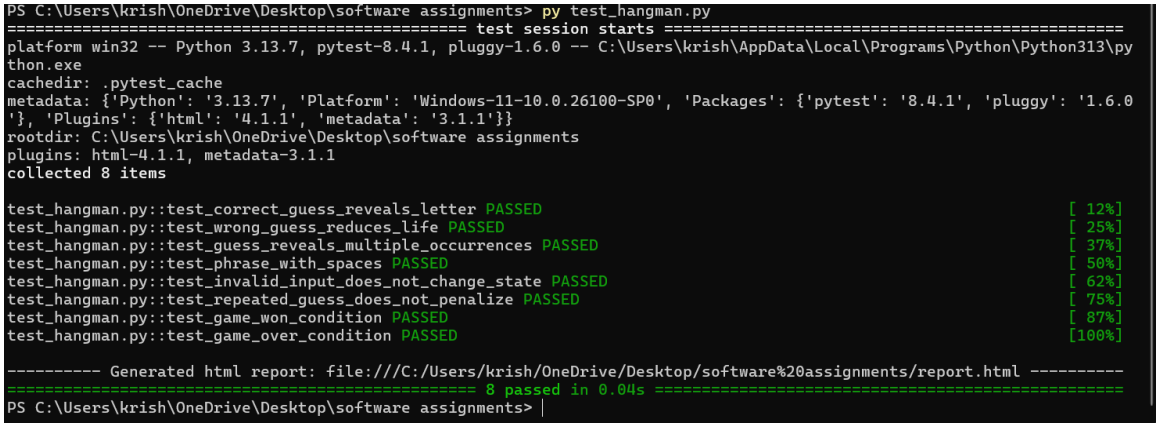

Figure 21: Terminal output showing intermediate test execution results

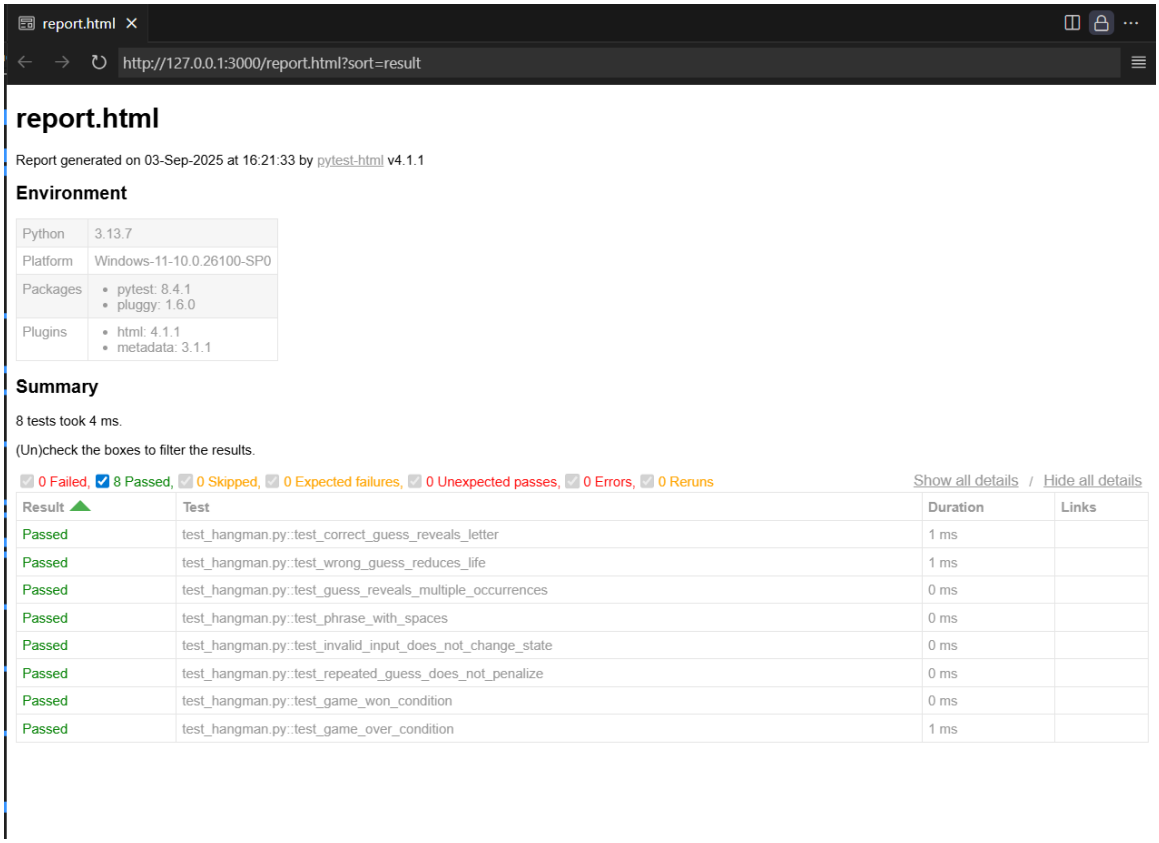

Figure 22: HTML report summary showing all unit tests passed

3. Conclusion

The Hangman game development in Python was a wonderful way to practice and demonstrate principles of Test-Driven Development (TDD), automated unit testing, and modular software design.

In the course of the project, all major requirements have been developed:

- Two difficulty levels (Basic and Intermediate).
- Life-based guessing with a graphical user interface (GUI).
- Timer capabilities for decisions made in a timely fashion.
- Hint capabilities to support players, moderated with life deduction.
- All game features were supported with unit testing.

The use of pytest and pytest-html was very effective for automated testing. The pytest and pytest-html were not only good for establishing correctness, but also provided documentation on results in a professional manner, for verification and for future maintenance.

Lesson learnt

- Writing test cases first before implementing (Test Driven Development (TDD)) made us think clearly about the requirements, and assigned little bugs during process of implementation.
- Modularizing my code into different files. (hangman.py, hangman_engine.py, hangman_gui.pyw, and test_hangman.py) helped my code in terms of readability and maintainability.
- Using automated testing saved me time and together with a few new test cases for regression tests, I was able to check the game easily after new features were introduced.
- Integrating the GUI with the backend logic focused me on the importance of synchronizing game states and user interactions.
- The timer implementation showed me the importance of edge case handling (e.g., what happens when the timer reaches zero).

Areas for consideration:

- Making the game multiplayer or have an online play feature would definitely make the game better.
- Increasing the size of the dictionary / phrases bank would provide the game with more replay ability.
- Adding a few animations, or more detailed drawings of the hangman to the GUI, would also improve some of the user experience.

In summary, I think the build of my project was successful since I was able to develop an entirely working Computer Engineering based Hangman game and use software engineering practices in TDD and automated testing.

References

Van Rossum, G., & Drake, F. L. (2009). *The Python language reference manual*. Network Theory Ltd. https://docs.python.org/3/reference

Beck, K. (2002). *Test-driven development: By example*. Addison-Wesley Professional. https://www.informit.com/store/test-driven-development-by-example-9780321146533

Python Software Foundation. (2024). *unittest — Unit testing framework*. In *Python documentation*. https://docs.python.org/3/library/unittest/

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., & Bruhin, F. (2024). *pytest: Helps you write better programs*. *Pytest Documentation*. https://docs.pytest.org/en/stable/