

**S225 PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS**

**Assessment 01**

**Full Name:** Dharma Bahadur Budhathoki

**Student ID:** S395172 **Campus Location:** Sydney **Course Code:** PRT575

**Assessment Title:** Assessment 01 **Submission Date:** 10 August 2025 **Lecturer:** Muhammad Rana

**Table of Contents**

[1. Introduction 1](#_Toc207964493)

[2. Tools and Technologies Used 1](#_Toc207964494)

[3. Development and Testing Process (TDD) 1](#_Toc207964495)

[4. Unit Test Cases 2](#_Toc207964496)

[5. Test Results Summary 18](#_Toc207964497)

[Terminal 20](#_Toc207964498)

[6. Conclusion 21](#_Toc207964499)

[References 22](#_Toc207964500)

**Table of Table**

[Table 1: Test Case 1 – Initial Display 2](#_Toc207964501)

[Table 2: Test Case 2 – Correct Guess 3](#_Toc207964502)

[Table 3: Test Case 3 – Incorrect Guess Deducts Life 5](#_Toc207964503)

[Table 4: Table 4. Test Case 4 – Win Condition 6](#_Toc207964504)

[Table 5: Test Case 5 – Lose Condition 8](#_Toc207964505)

[Table 6: Test Case 6 – Timer Deducts Life 9](#_Toc207964506)

[Table 7: Test Case 7 – Random Word Selection (Basic Level) 11](#_Toc207964507)

[Table 8: Test Case 8 – Random Phrase Selection (Intermediate Level) 14](#_Toc207964508)

[Table 9: Test Case 9 – Random Phrase Selection (Hard Level) 16](#_Toc207964509)

[Table 10: Test Case 10 – Phrase Guess 17](#_Toc207964510)

[Table 11: Overall Test results 19](#_Toc207964511)

**Table of Figures**

[Figure 1: Initial display 3](#_Toc207964514)

[Figure 2: Displaying corrected guessed letter 4](#_Toc207964515)

[Figure 3: Displaying deducted life after guessing wrong letter 6](#_Toc207964516)

[Figure 4: Displaying guessing letters 7](#_Toc207964517)

[Figure 5: Displaying ‘You Win!’ 8](#_Toc207964518)

[Figure 6: Displaying 'Game Over!' 9](#_Toc207964519)

[Figure 7: Displaying life deducted after 15 seconds 11](#_Toc207964520)

[Figure 8:Displaying numbers according to the levels 12](#_Toc207964521)

[Figure 9: Displaying Basic level 13](#_Toc207964522)

[Figure 10: Displaying random phrase selection 15](#_Toc207964523)

[Figure 11: Displaying chosen number 2 as intermediate (phrase) 15](#_Toc207964524)

[Figure 12: Random Phrase Selection (Hard Level) 16](#_Toc207964525)

[Figure 13: Displaying chosen number 3 as level hard (Expert) 17](#_Toc207964526)

[Figure 14:Revealing correct letters 18](#_Toc207964527)

[Figure 15: Displaying combination of unit testing and feature/UI 20](#_Toc207964528)

# Introduction

The goal of this report is to detail the design, development, and testing of a Hangman game that was built in Python using a Test-Driven Development (TDD) methodology. The game expands on the classic Hangman premise, in which a player must identify a concealed word or phrase by suggesting letters in a limited number of attempts. This project now has three difficulty levels: a Basic level that creates a single word, an Intermediate level that generates a brief phrase, and a hard level that generates a more sophisticated phrase. To increase the difficulty, the program includes a timer that allows just fifteen seconds for each estimate, if the player fails to act within that time, a life is removed. When a player makes an incorrect guess, they lose a life. The game continues until the player correctly answers the word or phrase or loses all of their lives.

The purpose of this report is to show how the Hangman software was written and tested with Python's built-in unittest framework. Automated unit testing was required to ensure that the game logic functioned properly, including right and wrong guesses, random word and phrase selection, and timer-based life deduction. The paper focuses on the tools utilized, the TDD approach used during development, extensive unit test cases with results, and the lessons learnt from the experience.

# 2. Tools and Technologies Used

The Hangman game was built with Python 3.11, which was selected for its simplicity, readability, and broad support for software engineering tasks. The Pygame library was used to construct a graphical interface that allowed the player to see the concealed word or phrase, the timer countdown, and the number of lives remaining. Automated testing was carried out using the unittest framework, which comes with Python. It allowed the application to be created in stages, with each feature properly evaluated. In addition to these tools, flake8 and pylint were used to confirm that the source code was Python-compliant. These technologies worked together to make the program modular, long-lasting, and easy to maintain.

# 3. Development and Testing Process (TDD)

The Hangman game was developed utilizing the Test-Driven Development (TDD) process. This strategy guaranteed that testing was central to the coding process rather than an afterthought. At the beginning of each feature's development, a unit test case was built to define the new functionality's expected behavior. For example, before developing the code that showed underscores for each concealed letter, a test was built to confirm that the word "apple" was represented by five underscores. Running the test at this stage confirmed that it failed since no implementation existed. After that, the necessary functionality was built, and the test was re-run. The test was successful once the feature had been correctly written, indicating that the functionality worked as intended. The last part of the cycle involved reorganizing the code to improve readability and maintainability while remaining unchanged in behavior.

This process was repeated for all main needs, including word and phrase selection, right and wrong guessing, win-loss circumstances, and timer-based life reductions. To demonstrate functioning, screenshots of test results and the game UI were taken. Following the TDD technique in this manner guaranteed that all requirements were tested and confirmed before proceeding to the next feature.

# 4. Unit Test Cases

A set of unit test cases were created to ensure that the Hangman software met all of the criteria. Each test case was recorded, including its goal, input, expected and actual results. Screenshots were created for each example to offer visual evidence of the program's activity.

**Table 1: Test Case 1 – Initial Display**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Verify initial underscores display for the word |
| Input | none |
| Expected Result | \_ \_ \_ \_ \_ |
| Actual Result | \_ \_ \_ \_ \_ |
|  |  |
| Screenshot | **Figure 1: Initial display** |

**Table 2: Test Case 2 – Correct Guess**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Reveal all occurrences of guessed letter |
| Input | Guess = p , a, Word = programming |
| Expected Result | P \_ \_ \_ \_ a \_ \_ \_ \_ \_ |
| Actual Result | P \_ \_ \_ \_ a \_ \_ \_ \_ \_ |
| Screenshot | **Figure 2: Displaying corrected guessed letter** |

**Table 3: Test Case 3 – Incorrect Guess Deducts Life**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Deduct life when a wrong guess is made |
| Input | Guess = z |
| Expected Result | Lives reduced by 1 |
| Actual Result | Lives reduced by 1 |
| Screenshot  before life 15 sec lives 6 |  |
| Screenshot  after life 15 sec lives 5 | **Figure 3: Displaying deducted life after guessing wrong letter** |

**Table 4: Table 4. Test Case 4 – Win Condition**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Ensure game ends with win when all letters guessed |
| Input | Guess letters: school |
| Expected Result | win = True, game\_over = False |
| Actual Result | win = True, game\_over = False |
| Screenshot of guessing letter | **Figure 4: Displaying guessing letters** |
| Screenshot of You win | **Figure 5: Displaying ‘You Win!’** |

**Table 5: Test Case 5 – Lose Condition**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Ensure game ends with loss when lives reach 0 |
| Input | No any inputs only by 15 sec timer |
| Expected Result | win = False, game\_over = True |
| Actual Result | win = False, game\_over = True |
| Screenshot  Of game over | **Figure 6: Displaying 'Game Over!'** |

**Table 6: Test Case 6 – Timer Deducts Life**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Verify life deduction when timer expires |
| Input | Timer exceeds 15 seconds |
| Expected Result | Lives reduced by 1 |
| Actual Result | Lives reduced by 1 |
| Screenshot before 15 seconds |  |
| Screenshot after 15 seconds | **Figure 7: Displaying life deducted after 15 seconds** |

**Table 7: Test Case 7 – Random Word Selection (Basic Level)**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Verify random word selection |
| Input | Level = Basic |
| Expected Result | Word from [python, hangman, school, programming, testing] |
| Actual Result | Word chosen from list |
| Screenshot choose level by pressing a number for basic its 1 | **Figure 8:Displaying numbers according to the levels** |
| Screen shot of basic level | **Figure 9: Displaying Basic level** |

**Table 8: Test Case 8 – Random Phrase Selection (Intermediate Level)**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Verify random phrase selection |
| Input | Level = Intermediate |
| Expected Result | Phrase from [open source, unit test, software engineering, artificial intelligence] |
| Actual Result | Phrase chosen from list |
| Screenshot choose level by pressing a number for Intermediate its 2 | **Figure 10: Displaying random phrase selection** |
| Screen shot of Intermediate level | **Figure 11: Displaying chosen number 2 as intermediate (phrase)** |

**Table 9: Test Case 9 – Random Phrase Selection (Hard Level)**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Verify random phrase selection for hard level |
| Input | Level = Hard |
| Expected Result | Phrase chosen from hard phrase list |
| Actual Result | Phrase chosen from list |
| Screenshot choose level by pressing a number for hard its 3 | **Figure 12: Random Phrase Selection (Hard Level)** |
|  | **Figure 13: Displaying chosen number 3 as level hard (Expert)** |

**Table 10: Test Case 10 – Phrase Guess**

|  |  |
| --- | --- |
| **Attribute** | **Details** |
| Purpose | Reveal guessed letters in a phrase |
| Input | Phrase = asynchronous event loop  , Guess = a , j , r , u, v , y |
| Expected Result | Game Over  Answer: asynchronous event loop  Press R to restart and q to quit |
| Actual Result | Correct letters revealed |
| Screenshot of Phrase Guess | **Figure 14:Revealing correct letters** |

# 5. Test Results Summary

All automated unit tests for the Hangman game were executed using Python’s unittest framework. A total of 13 test cases were implemented, covering:

* Initial game display
* Correct and incorrect guess logic
* Life deduction on wrong input
* Timer-based life deduction
* Win and loss conditions across Basic, Intermediate, and Hard levels
* Random word and phrase selection for each difficulty level

**Table 11: Overall Test results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Suite** | **Total Tests** | **Passed** | **Failed** | **Status** |
| HangmanGame Unit Tests | 13 | 13 | 0 | All Passed |

## Terminal

A screen shot of a computer

AI-generated content may be incorrect.

**Figure 15: Displaying combination of unit testing and feature/UI**

This confirms that the Hangman game implementation meets all functional requirements specified in the assignment. The combination of unit testing and feature/UI verification via Pygame screenshots provides strong evidence of correctness and reliability.

# 6. Conclusion

The development of the Hangman game using Python, Pygame, and unittest demonstrated the effectiveness of the Test-Driven Development methodology. By writing tests before implementation, each feature was thoroughly validated and integrated into the game in a structured way. The program achieved all specified requirements, including multiple levels of difficulty, timer-based life deduction, and full win/loss detection. Automated testing confirmed the accuracy of each function, while the Pygame interface provided visual confirmation of results.

The project highlighted several lessons. The modular design of the game made both development and testing easier. The use of automated unit testing reduced the need for manual debugging and ensured greater confidence in the code. However, there are certain aspects that might be improved in the future, such as introducing more complicated phrases for the Hard level, improving input validation for unexpected entries, and allowing players to switch difficulty levels dynamically during gaming.

In summary, the project successfully demonstrated how TDD can guide the development of reliable and maintainable software. The final game met all requirements, passed all unit test cases, and provides a strong foundation for future enhancements. The complete project, including code, test cases, and report, is available in the GitHub repository: <https://github.com/Dharma001/hangman>.

# References

*How to play Hangman: 8 simple rules* (no date) *wikiHow*. Available at: https://www.wikihow.com/Play-Hangman (Accessed: 05 September 2025).

*GitHub · build and ship software on a single, Collaborative Platform* (no date) *GitHub*. Available at: https://github.com/ (Accessed: 05 September 2025).

*What is python used for? A beginner’s guide* (no date) *Coursera*. Available at: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (Accessed: 05 September 2025).

Writer, C. BasuMallick    Technical *et al.* (2022) *What is TDD (test driven development)?*, *Spiceworks Inc*. Available at: https://www.spiceworks.com/tech/devops/articles/what-is-tdd/ (Accessed: 05 September 2025).