PRT582 Software Engineering: Process and Tools

Software Unit Testing Report (Hangman Game)

Campus Location: Danala Campus (ECP)
Semester: S2 2025

Submitted By:
Dipesh Wagle (394745)

Submission Date:
5/09/2025

Lecturer's Name:
Charles Yeo

INTRODUCTION

Objectives and Requirements:

This report provides the documentation of the Test-Driven Development (TDD) approach to implement a Hangman Game, which meets the following requirements:

Two Difficulty Levels:

There are two difficulty levels: Basic and Intermediate. The Basic level contains single words, the Intermediate Level contains multi-word phrases, and the words and phrases are generated randomly.

Validation of Dictionary:

All the words and phrases come from the valid dictionary files (dictionary.txt and phrases.txt)

Presentation of missing letters:

Presentation of underscores for the missing letters to be guessed.

Timed Input:

15-second timer per guess with automatic life deduction on timeout.

Revelation of letters:

A correct guess from the user reveals the correct instance of the letter throughout the word or phrase.

Life Deduction:

Wrong guesses from the user deduct the player's life.

Condition of Win:

The player must find all the missing words before the player's life becomes zero.

Control of Flow of the Game:

The game will continue until a win, a loss, or a manual quit.

Automated Unit Testing Tool

The Unittest was chosen for automated unit testing because of the following reasons:

Built-in Framework: Unittest, the standard library for Python, is readily available, compatible with several Python environments, and doesn't require any further installations.

Simple and familiar syntax: Test cases are defined as classes that inherit from Unittest in the class-based methodology used by the Unittest framework.Most developers are accustomed to using TestCase.

Assertion Techniques: Assertion methods such as assertEqual, assertIn, assertTrue help to compare the expected and actual results.

Test Discovery: The Automated test discovery feature of the Unittest framework makes it possible to find and run every test case in a directory or module.

Extensibility: The Unittest framework is designed to be expanded. Developers can create their own Testcase to add features or change existing methods.

IDE Integration: Unittest works combined with Python IDEs and code editors and also provides visual results for tests and debugging features that improve the developer performance.
Clear Reporting: The Unittest framework generates test reports in detail with clear success and failure messages, making debugging and identifying the issues in an efficient way.

PROCESS

Test Driven Development Methodology

This game strictly followed the Test-Driven Development (TDD) Approach using Red Green Refactor cycle:

Red Phase:

Before the implementation of any code's logical part, I wrote failing tests for every requirement.

Green Phase:

I Implemented the code functionality in order to make the tests pass.

Refactor phase:

The quality of code was enhanced while maintaining all tests passing.

Requirement Implementation with evidence:

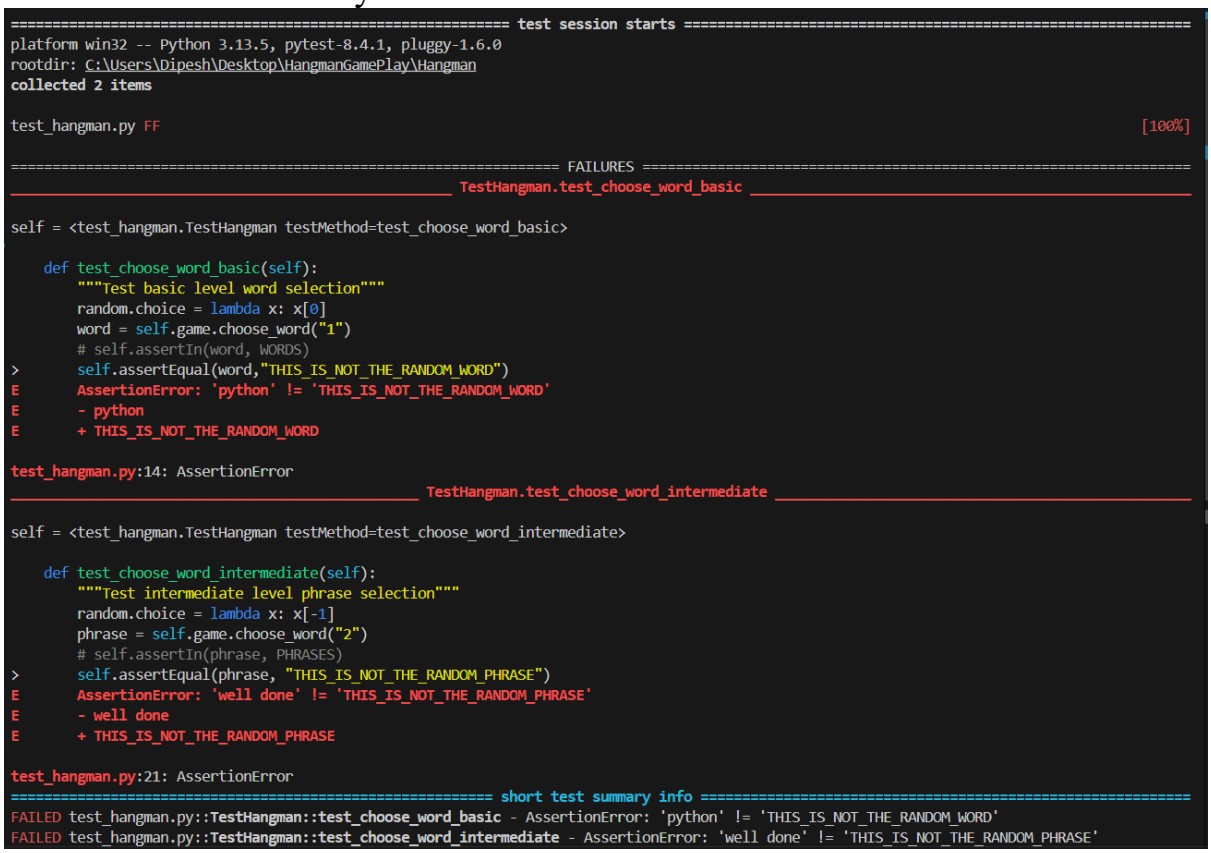
Requirement 1: Two difficulty levels

TDD Approach: Tests were written to verify that selection of level returns appropriate result from the correct dictionaries.

Test_hangman.py Code:

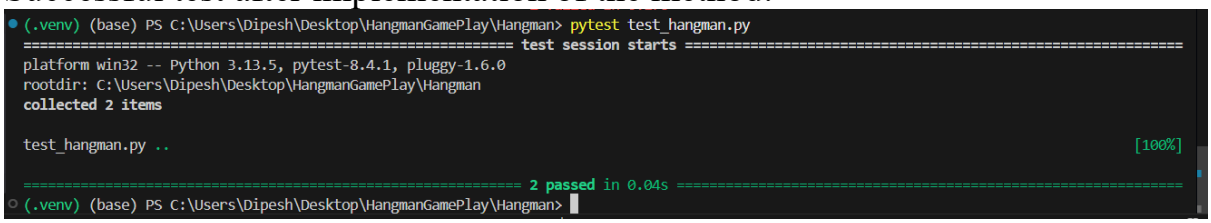
```
    # Requirement 1: Two level tests
    def test_choose_word_basic(self):
        """Test basic level word selection"""
        random.choice = lambda x: x[0]
        word = self.game.choose_word("1")
        self.assertEqual(word,"THIS_IS_NOT_THE_RANDOM_WORD")

    def test_choose_word_intermediate(self):
        """Test intermediate level phrase selection"""
        random.choice = lambda x: x[-1]
        phrase = self.game.choose_word("2")
        self.assertEqual(phrase, "THIS_IS_NOT_THE_RANDOM_PHRASE")
```

Test failure for difficulty level selection:



```
============================================================ test session starts =============================================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 2 items

test_hangman.py FF                                                                                                                    [100%]

================================================================= FAILURES ==================================================================
_____________________________________________________ TestHangman.test_choose_word_basic _____________________________________________________

self = <test_hangman.TestHangman testMethod=test_choose_word_basic>

    def test_choose_word_basic(self):
        """Test basic level word selection"""
        random.choice = lambda x: x[0]
        word = self.game.choose_word("1")
        # self.assertIn(word, WORDS)
>       self.assertEqual(word,"THIS_IS_NOT_THE_RANDOM_WORD")
E       AssertionError: 'python' != 'THIS_IS_NOT_THE_RANDOM_WORD'
E       - python
E       + THIS_IS_NOT_THE_RANDOM_WORD

test_hangman.py:14: AssertionError
_____________________________________________ TestHangman.test_choose_word_intermediate _____________________________________________

self = <test_hangman.TestHangman testMethod=test_choose_word_intermediate>

    def test_choose_word_intermediate(self):
        """Test intermediate level phrase selection"""
        random.choice = lambda x: x[-1]
        phrase = self.game.choose_word("2")
        # self.assertIn(phrase, PHRASES)
>       self.assertEqual(phrase, "THIS_IS_NOT_THE_RANDOM_PHRASE")
E       AssertionError: 'well done' != 'THIS_IS_NOT_THE_RANDOM_PHRASE'
E       - well done
E       + THIS_IS_NOT_THE_RANDOM_PHRASE

test_hangman.py:21: AssertionError
========================================================== short test summary info ===========================================================
FAILED test_hangman.py::TestHangman::test_choose_word_basic - AssertionError: 'python' != 'THIS_IS_NOT_THE_RANDOM_WORD'
FAILED test_hangman.py::TestHangman::test_choose_word_intermediate - AssertionError: 'well done' != 'THIS_IS_NOT_THE_RANDOM_PHRASE'
```

Successful test after implementation of the method:



```
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
============================================================ test session starts =============================================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 2 items

test_hangman.py ..                                                                                                                    [100%]

============================================================= 2 passed in 0.04s ==============================================================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

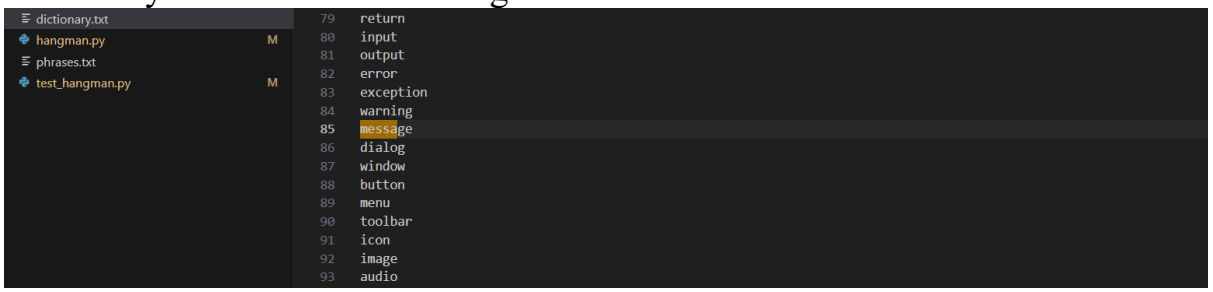
Requirement 2: Valid Dictionary Words
TDD Approach: Test verifies whether the basic or intermediate words come from predefined dictionaries or not.
Test_hangman.py Code:

```python
#Requirement 2: Words come from valid dictionaries
def test_initialize_game_valid_words(self):
    """Words come from valid dictionaries"""
    self.game.initialize_game("1")
    self.assertIn(self.game.word, WORDS)

def test_initialize_game_valid_phrases(self):
    """Phrases come from valid dictionaries"""
    self.game.initialize_game("2")
    self.assertIn(self.game.word, PHRASES)
```

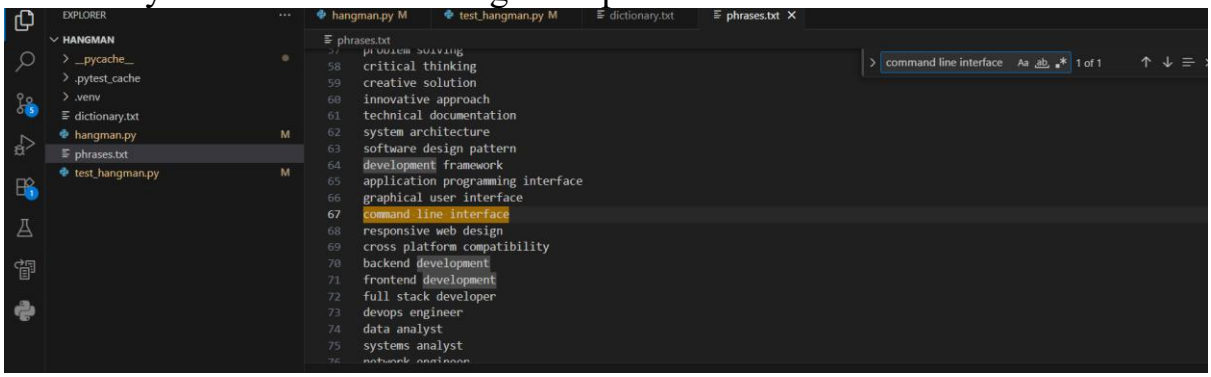
Successful Test execution validating that words or phrases come from predefined dictionaries:

```
Ran 3 tests in 0.001s

OK
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> python test_hangman.py
[DEBUG] Chosen phrase: command line interface
.[DEBUG] Chosen word: message
.
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Dictionary file content containing valid words



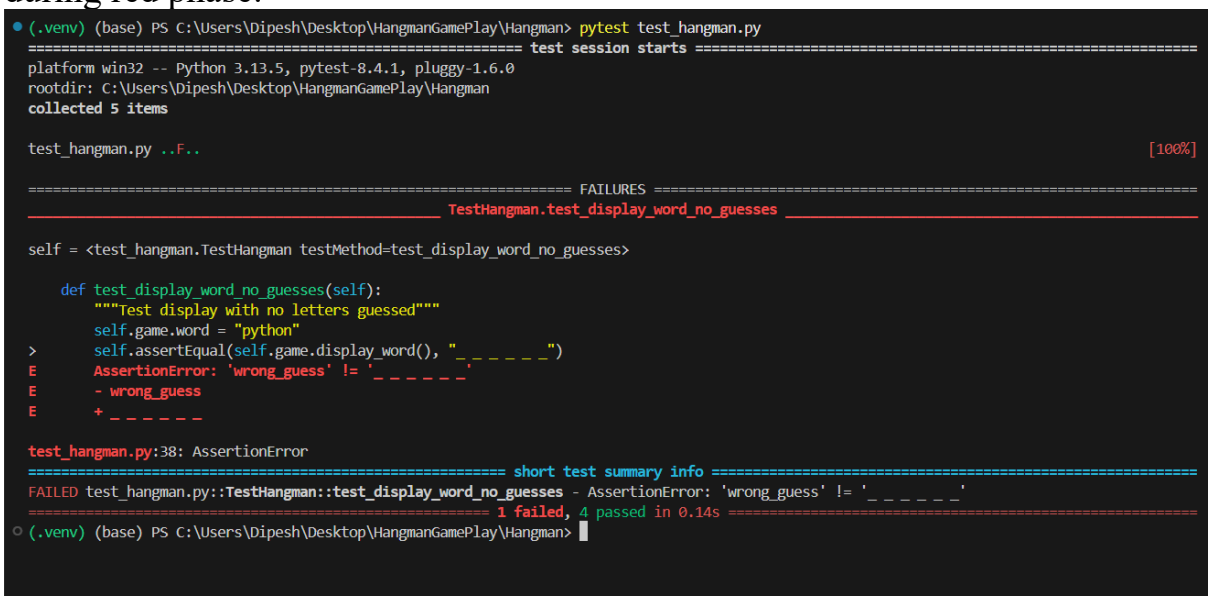
Dictionary file content containing valid phrases



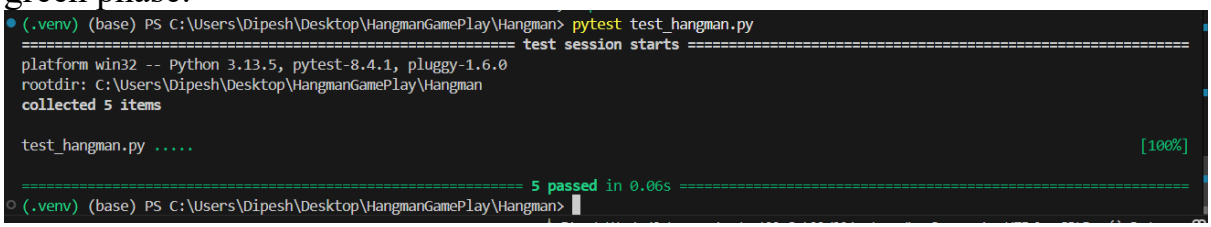
Requirement 3: Displaying underscore for the missing letters
TDD Approach: The test makes sure that unguessed or missing letters are represented as underscores.
Test_hangman.py Code:

```python
# Requirement 3: Underscores for missing letters
def test_display_word_no_guesses(self):
    """Test display with no letters guessed"""
    self.game.word = "python"
    self.assertEqual(self.game.display_word(), "_ _ _ _ _ _")
```

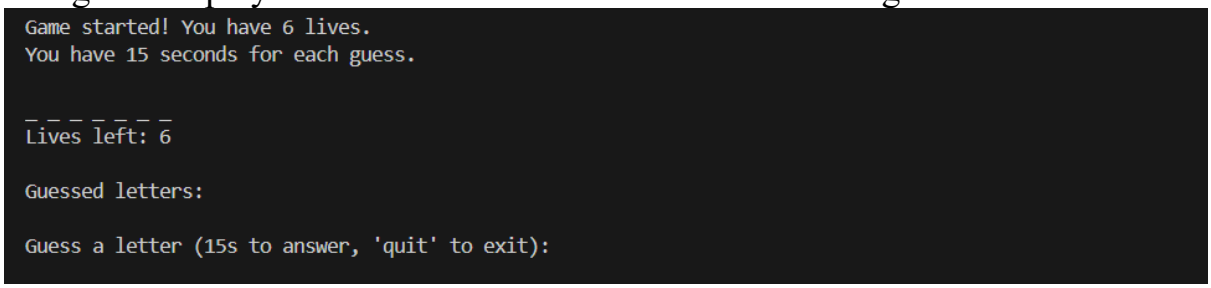
Failure in the execution of Test for displaying underscore for missing letters during red phase:



Successful execution of Test for displaying underscore for missing letters during green phase:



The game displays an underscore for letters that are missing:



Requirement 4: 15-second countdown timer that deducts life
TDD Approach: This Test makes sure that the timeout functionality reduces the players' lives.
Test_hangman.py Code:

```python
# Requirement 4: 15-second timer with life deduction
def test_life_deduction_timeout(self):
    """Timeout reduces lives"""
    initial_lives = self.game.lives
    result = self.game.process_timeout()
    self.assertEqual(result, "timeout")
    self.assertEqual(self.game.lives, initial_lives - 1)
```

Due to the lack of implementation of the process_timeout() method to reduce lives, the test first failed with AssertionError: 6!= 5.

```
OUTPUT  PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
============================= test session starts =============================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 6 items

test_hangman.py .....F                                                   [100%]

================================== FAILURES ===================================
_____________________ TestHangman.test_life_deduction_timeout _____________________

self = <test_hangman.TestHangman testMethod=test_life_deduction_timeout>

    def test_life_deduction_timeout(self):
        """Timeout reduces lives"""
        initial_lives = self.game.lives
        result = self.game.process_timeout()
        self.assertEqual(result, "timeout")
>       self.assertEqual(self.game.lives, initial_lives - 1)
E       AssertionError: 6 != 5

test_hangman.py:46: AssertionError
=========================== short test summary info ===========================
FAILED test_hangman.py::TestHangman::test_life_deduction_timeout - AssertionError: 6 != 5
============================== 1 failed, 5 passed in 0.23s ==============================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

The test was successful after the life deduction logic (self.lives -= 1) was implemented.

```
OUTPUT  PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
============================= test session starts =============================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 6 items

test_hangman.py ......                                                   [100%]

============================== 6 passed in 0.07s ==============================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

Requirement 5: Revelation of the correctly guessed letter.
TDD Approach: The test confirms that if the player's chosen letter exists in the answer, then all places in the answer where that letter appears will be revealed.

Test_hangman.py Code:

```python
# Requirement 5: Correct guess reveals letters
def test_display_word_some_guesses(self):
    """Test display with some letters guessed"""
    self.game.word = "python"
    self.game.guessed_letters = ["p", "o"]
    self.assertEqual(self.game.display_word(), "p _ _ _ o _")
```

Due to insufficient letter revelation logic, the test failed during the execution.

```
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
========================================================= test session starts =========================================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 7 items

test_hangman.py ...F...                                                                                                      [100%]

============================================================== FAILURES ===============================================================
_____________________________________________ TestHangman.test_display_word_some_guesses _____________________________________________

self = <test_hangman.TestHangman testMethod=test_display_word_some_guesses>

    def test_display_word_some_guesses(self):
        """Test display with some letters guessed"""
        self.game.word = "python"
        self.game.guessed_letters = ["p", "o"]
>       self.assertEqual(self.game.display_word(), "p _ _ _ o _")
E       AssertionError: 'p _ _ _ _ _' != 'p _ _ _ o _'
E       - p _ _ _ _ _
E       ?       ^
E       + p _ _ _ o _
E       ?       ^

test_hangman.py:53: AssertionError
======================================================= short test summary info =======================================================
FAILED test_hangman.py::TestHangman::test_display_word_some_guesses - AssertionError: 'p _ _ _ _ _' != 'p _ _ _ o _'
===================================================== 1 failed, 6 passed in 0.21s =====================================================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

The test passed after the display_word() method was fixed.

```
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
========================================================= test session starts =========================================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 7 items

test_hangman.py .......                                                                                                      [100%]

========================================================== 7 passed in 0.06s ==========================================================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

Requirement 6: Player's life reduction for the wrong guess.
TDD Approach: The test ensures that every time the player guesses a letter wrong, the player's life would be deducted.

Test_hangman.py Code:

```python
# Requirement 6: Wrong guess deducts life
def test_life_deduction_wrong_guess(self):
    """Test that wrong guess reduces lives"""
    self.game.initialize_game("1")
    self.game.word = "python"
    initial_lives = self.game.lives
    result = self.game.process_guess("x")  # Wrong guess
    self.assertEqual(result, "wrong")
    self.assertEqual(self.game.lives, initial_lives - 1)
```

Because of incomplete deduction logic, the test failed during the execution.

```
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
============================= test session starts =============================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 8 items

test_hangman.py .......F                                                 [100%]

================================== FAILURES ===================================
_____________________ TestHangman.test_life_deduction_wrong_guess _____________________

self = <test_hangman.TestHangman testMethod=test_life_deduction_wrong_guess>

    def test_life_deduction_wrong_guess(self):
        """Test that wrong guess reduces lives"""
        self.game.initialize_game("1")
        self.game.word = "python"
        initial_lives = self.game.lives

        result = self.game.process_guess("x")  # Wrong guess

        self.assertEqual(result, "wrong")
>       self.assertEqual(self.game.lives, initial_lives - 1)
E       AssertionError: 6 != 5

test_hangman.py:65: AssertionError
=========================== short test summary info ===========================
FAILED test_hangman.py::TestHangman::test_life_deduction_wrong_guess - AssertionError: 6 != 5
========================= 1 failed, 7 passed in 0.19s =========================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

After the implementation of the logic for deduction of lives (self.lives -= 1), the test execution was successful.

```
OUTPUT  PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
============================= test session starts =============================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 8 items

test_hangman.py ........                                                 [100%]

============================== 8 passed in 0.04s ==============================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

Requirement 7: The player must find the missing word before the player's life becomes zero.
TDD Approach: The test makes sure that players are capable of winning before their lives go out.
Test_hangman.py Code:
Condition when the player runs out of his/her life:

```python
# Requirement 7: Win before lives reach zero
def test_win_condition(self):
    """Test that correct final guess wins game"""
    self.game.initialize_game("1")
    self.game.word = "hi"
    self.game.guessed_letters = ["h"]
    self.game.lives = 0 # Player has no lives
    result = self.game.process_guess("i")
    self.assertEqual(result, "win")
    self.assertTrue(self.game.is_word_guessed())
    self.assertTrue(self.game.lives > 0)
```

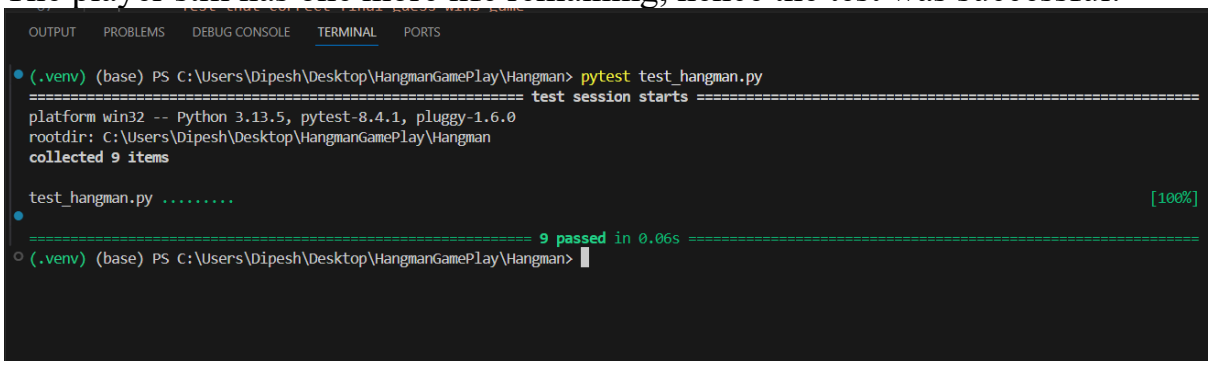
The test failed because the player had run out of life as **"self.game.lives=0".**

```
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman> pytest test_hangman.py
========================================================== test session starts ==========================================================
platform win32 -- Python 3.13.5, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman
collected 9 items

test_hangman.py .......F                                                                                                        [100%]

=============================================================== FAILURES ================================================================
_____________________________________________ TestHangman.test_win_condition _____________________________________________

self = <test_hangman.TestHangman testMethod=test_win_condition>

    def test_win_condition(self):
        """Test that correct final guess wins game"""
        self.game.initialize_game("1")
        self.game.word = "hi"
        self.game.guessed_letters = ["h"]
        self.game.lives = 0 # Player has no lives
        result = self.game.process_guess("i")
        self.assertEqual(result, "win")
        self.assertTrue(self.game.is_word_guessed())
>       self.assertTrue(self.game.lives > 0)
E       AssertionError: False is not true

test_hangman.py:75: AssertionError
======================================================== short test summary info ========================================================
FAILED test_hangman.py::TestHangman::test_win_condition - AssertionError: False is not true
====================================================== 1 failed, 8 passed in 0.16s ======================================================
(.venv) (base) PS C:\Users\Dipesh\Desktop\HangmanGamePlay\Hangman>
```

Condition when there is still life left in the player's life:

```python
# Requirement 7: Win before lives reach zero
def test_win_condition(self):
    """Test that correct final guess wins game"""
    self.game.initialize_game("1")
    self.game.word = "hi"
    self.game.guessed_letters = ["h"]
    self.game.lives = 1 # Player still has a life
    result = self.game.process_guess("i")
    self.assertEqual(result, "win")
    self.assertTrue(self.game.is_word_guessed())
    self.assertTrue(self.game.lives > 0)
```

The player still has one more life remaining, hence the test was successful.

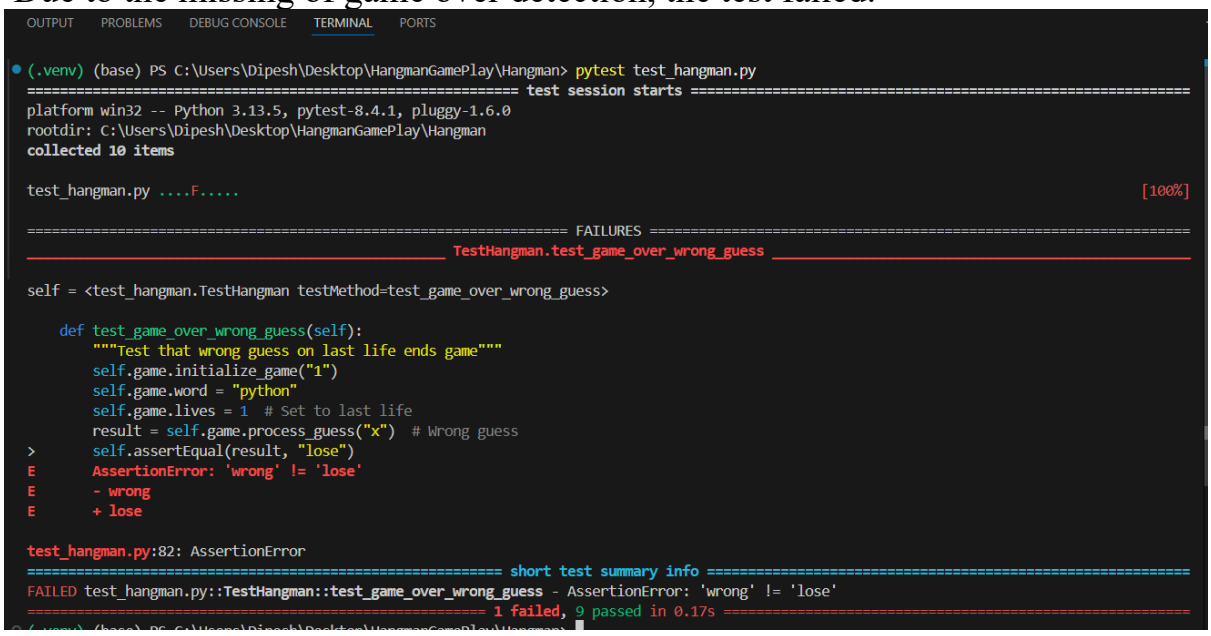


Requirement 8: The game should keep going until the end.
TDD Approach: The Test makes sure that the game will keep going until the player quits the game or life becomes zero.

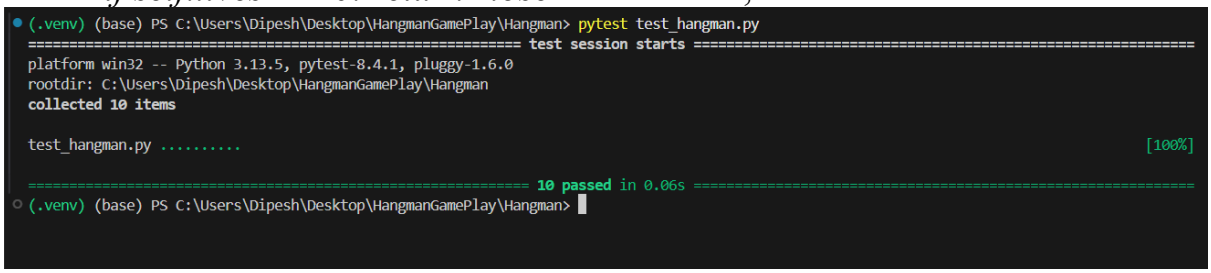
Test_hangman.py Code:

```python
# Requirement 8: Game continues until quit or loss
def test_game_over_wrong_guess(self):
    """Test that wrong guess on last life ends game"""
    self.game.initialize_game("1")
    self.game.word = "python"
    self.game.lives = 1  # Set to last life
    result = self.game.process_guess("x")  # Wrong guess
    self.assertEqual(result, "lose")
    self.assertEqual(self.game.lives, 0)
```

Due to the missing of game over detection, the test failed.



Once '*if self.lives == 0: return "lose"*" was added, the test was successful.



CONCLUSION

What went well:

i. **Structured Implementation:** The TDD approach made sure that all the requirements were met and also whether it offered a clear pathway for development.

ii. **Detection of Error at an earlier Stage:** Writing the test cases at an early stage helped in the detection of edge case scenarios, which saved a lot of time later in the development phase.

iii. **Code Quality:** The code's design became simpler and more testable.

iv. **Documentation:** The test aids all the new developers by showing them how the system works.

v. **Confidence in Changes:** If anything in the system doesn't function properly, the test would reveal it immediately, and with the help of it, it is safer to improve and make changes in the existing system.

vi. **Clear satisfaction of the requirements:** Each and every test was related to a specific requirement. So, verification of all the functional requirements was very easy to verify.

Areas for improvement:

i. **Testing of Timer:** Initially, it was hard to test the countdown timer functionality because we had to wait for 15 seconds every time. Therefore, research about the simulating technique would help make testing much faster.

ii. **Testing of User Interface (UI):** The current tests validate the game logic perfectly, but testing the text-based user interface (e.g., what is printed to the console) is more complex and was not fully implemented.

iii. **Tests rely on External files:**
Some of the tests required actual Word files (dictionary.txt, phrases.txt) to work. If any of the files got missed, the tests would definitely fail. So, instead of relying on local files, the game can retrieve letters from an external API, which would make the game more fun and unpredictable.

iv. **Game running on cross-platforms:**
The game does not run on other OS platforms other than Windows, like Linux and macOS, as the game uses a feature from Windows for its timer. So, to fix this, the code needs to be updated to handle the user's input.

Future enhancement opportunities:

i. **Graphical User Interface (GUI):** The game could be enhanced by upgrading to a graphical interface instead of using a command-line interface.

ii. **Difficulty levels as a Challenge:** By making the game more difficult with the addition of a difficulty level as "Hard" level. This level will have longer words and fewer clues.

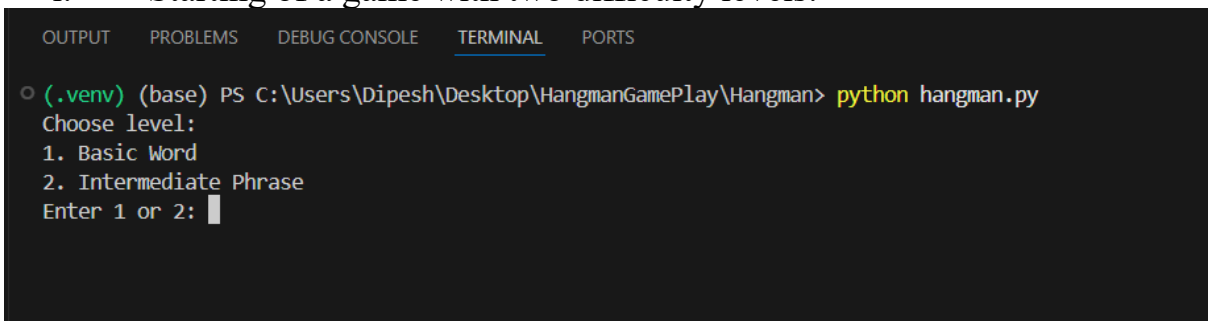
iii. **Continuous Integration/Continuous Deployment (CI/CD):** With the help of a CI/CD pipeline, the test suite would be automatically executed on every code change with quality.

iv. **Player Statistics:** Adding the additional features that track victories, defeats, and fastest speeds, which would increase the long-term player engagement.

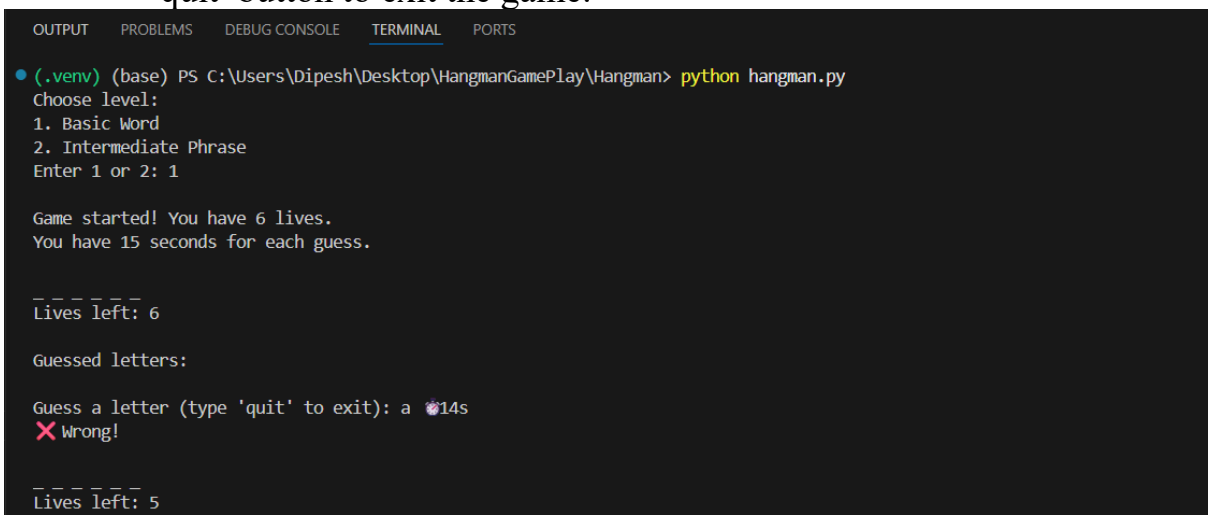
v. **Hint for the game:** By sacrificing the additional life of the player, a hint of a single character could be provided, which could help them when they are stuck.

Final Gameplay Demo:

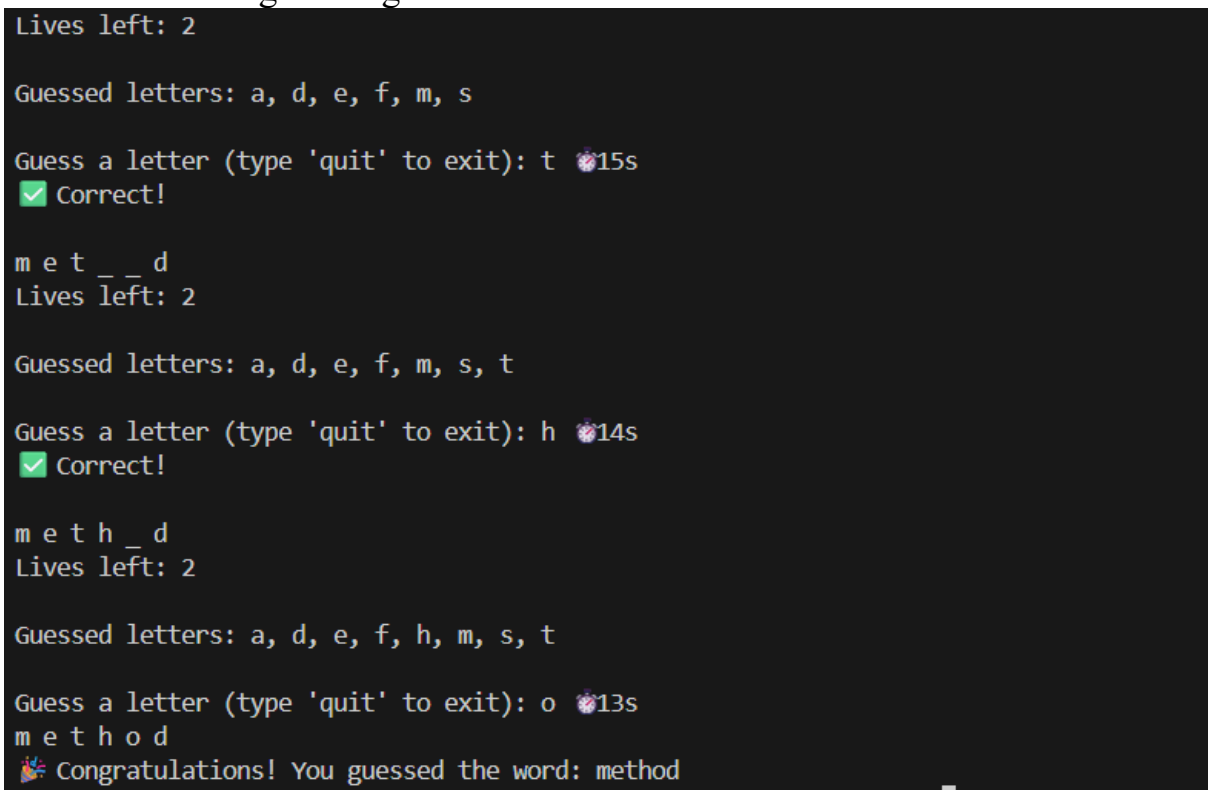
i. Starting of a game with two difficulty levels:



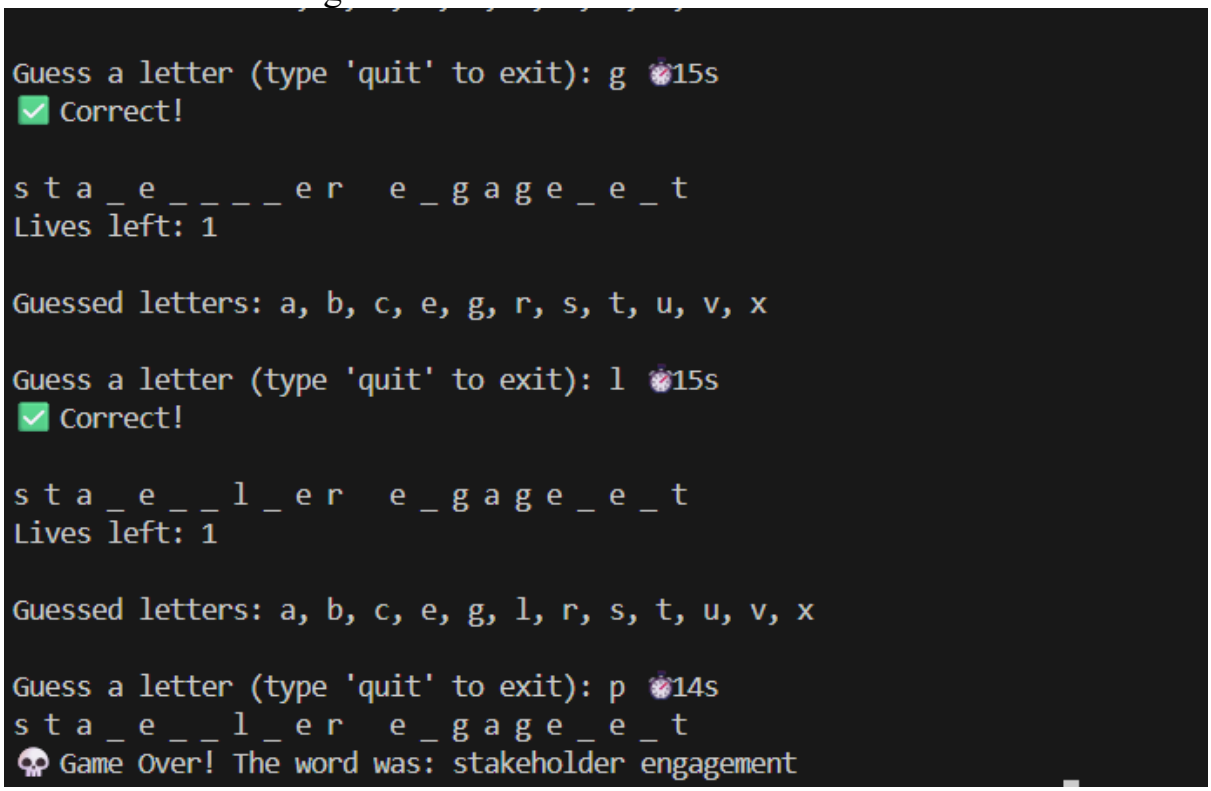
ii. Underscores showing the missing letters the player needs to find, lives left count, wrong letters guessed by players, countdown timer, and a 'quit' button to exit the game:



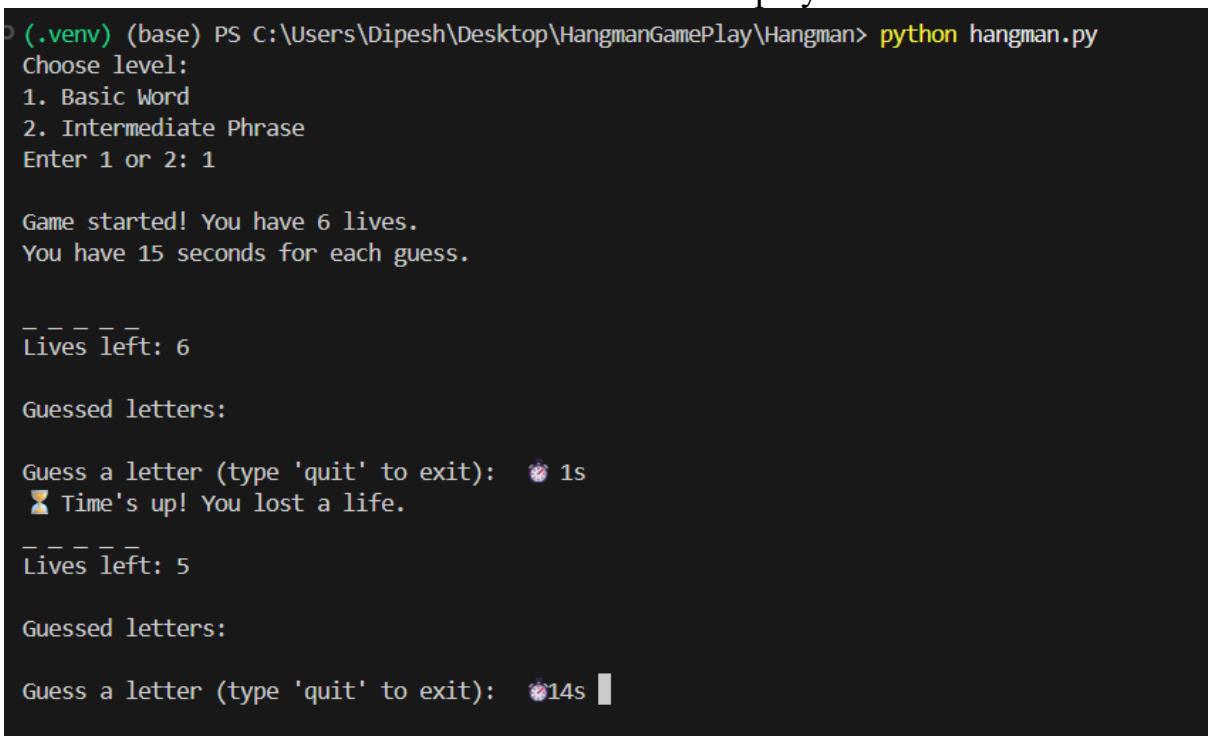
iii. Winning of the game condition:



iv. Loss of the game condition:



v. Countdown timer that reduces lives of a player:

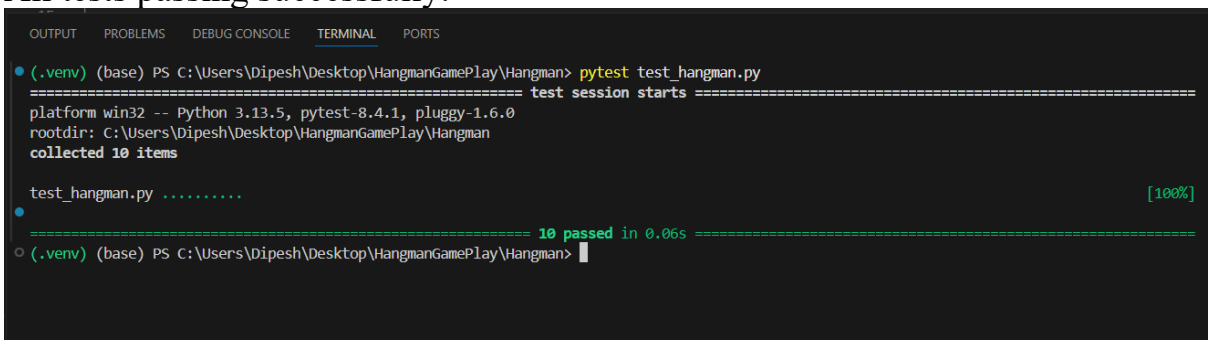


The final source code, unit test file, words/phrases file, and the comprehensive report are available at:
https://github.com/dapesh/HangmanGamePlay.git

The repository includes:
hangman.py: Main game implementation file
test_hangman.py: Comprehensive test suite
dictionary.txt: Words for basic level
phrases.txt: Phrases for intermediate level
SoftwareEngineeringFinalReport.pdf: This is a complete documentation file

Appendix: Test Execution Summary

All tests passing successfully:



Pylint and **Flake8** compliance reports: