PRT582
SOFTWARE ENGINEERING: PROCESS AND TOOLS

Software Unit Testing Report

Name: Simin Li

ID: S390115

Darwin Danala Campus

Introduction

This report focuses on the development of a Hangman game using Python. The project adopts a Test Driven Development (TDD) approach, where tests are written before implementing the actual features. This method ensures that each function is developed according to its expected behavior, reducing errors and improving code reliability. To support this approach, unittest, Python's built-in unit testing framework, will be used for automated testing. By combining TDD with unittest, the development will proceed step by step, gradually fulfilling all the functional requirements specified in this assessment.

Process

Step 1: Create test file and minimal Hangman Class

Create a file named test_hangman.py (hereafter referred to as the test file), import the unittest module, and write a minimal Hangman class interface along with a failing test.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):
        game = Hangman(answer="apple")
        self.assertEqual(game.display_word(), "_ _ _ _ _")

if __name__ == "__main__":
    unittest.main()
```

Run this code, and the output is as follows:

```
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", line
 2, in <module>
    from hangman import Hangman
ModuleNotFoundError: No module named 'hangman'
(base) iammin@iammindeMacBook-Air Software Unit Testing Report %
```

The error occurs because hangman.py has not been created yet and the Hangman class is undefined. Next, create hangman.py and write the minimal code that passes the test.

```python
class Hangman:
    def __init__(self, answer):
        self.answer = answer.lower()
        self.guessed = set()

    def display_word(self):
        return ' '.join([c if c in self.guessed else '_' for c in self.answer])
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets this requirement.

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(base) iammin@iammindeMacBook-Air Software Unit Testing Report %
```

Step 2: Implement guess functionality and display correct letters

Add code in test_hangman.py to check whether correctly guessed letters are displayed.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):
        game = Hangman(answer="apple")
        self.assertEqual(game.display_word(), "_ _ _ _ _")

    def test_correct_guess(self):
        game = Hangman(answer="apple")
        game.guess('a')
        self.assertEqual(game.display_word(), "a _ _ _ _")

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
ERROR: test_correct_guess (__main__.TestHangman.test_correct_guess)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py",
 11, in test_correct_guess
    game.guess('a')
    ^^^^^^^^^^
AttributeError: 'Hangman' object has no attribute 'guess'. Did you mean: 'guessed'?

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (errors=1)
```

Update hangman.py to implement the guess( ) method.

```python
class Hangman:
    def __init__(self, answer):
        self.answer = answer.lower()
        self.guessed = set()

    def display_word(self):
        return ' '.join([c if c in self.guessed else '_' for c in self.answer])

    def guess(self, letter):
        self.guessed.add(letter.lower())
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets this requirement.

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Step 3: Deduct life on wrong guess

Add a test in the test file for “Every time the player guesses a letter wrong, the player’s life will be deducted.”

```
class TestHangman(unittest.TestCase):
        self.assertEqual(game.display_word(), "_ _ _ _ _")

    def test_correct_guess(self):
        game = Hangman(answer="apple")
        game.guess('a')
        self.assertEqual(game.display_word(), "a _ _ _ _")

    def test_wrong_guess_deducts_life(self):
        game = Hangman(answer="apple", lives= 5)
        game.guess('x')
        self.assertEqual(game.lives, 4)   # life should be deducted
        self.assertEqual(game.display_word(), "_ _ _ _ _")  # still all underscores

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
TypeError: Hangman.__init__() got an unexpected keyword argument 'lives'

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (errors=1)
```

Update the __init__( ) and guess( ) methods in Hangman.py to define the player's lives and implement life deduction for each wrong guess.

```
class Hangman:
    def __init__(self, answer, lives = 6):
        self.answer = answer.lower()
        self.guessed = set()
        self.lives = lives

    def display_word(self):
        return ' '.join([c if c in self.guessed else '_' for c in self.answer])

    def guess(self, letter):
        letter = letter.lower()
        if letter in self.guessed:
            return
        self.guessed.add(letter)
        if letter not in self.answer:
            self.lives -= 1
```

Execute hangman.py. It runs successfully without errors. Then run the

test file, and the output shows the test passes, indicating that the function meets this requirement.

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Step 4: Check win and lose Conditions

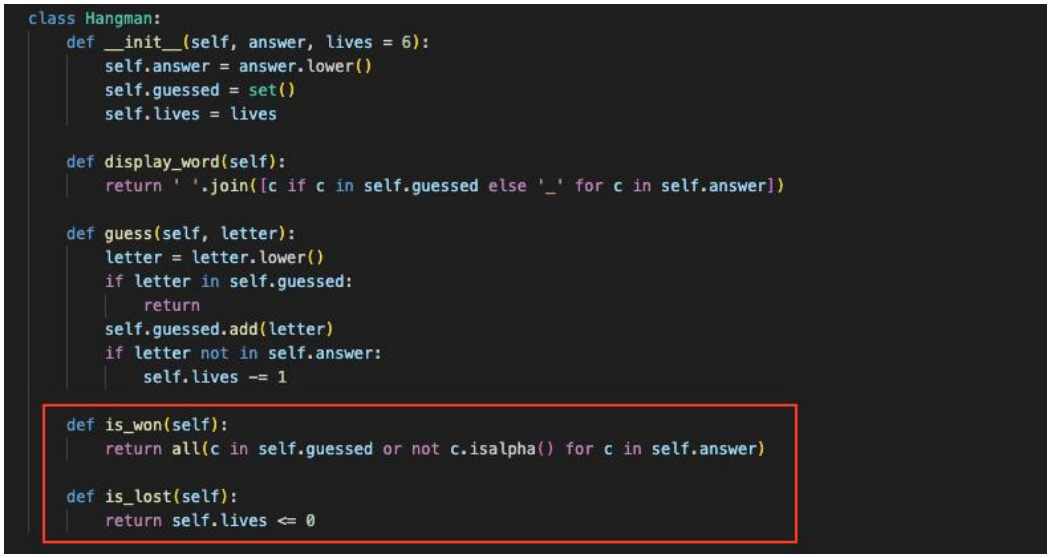
Add tests code in the test file to detect win (all letters guessed) and lose (lives reach 0) conditions.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):
        game = Hangman(answer="apple")
        self.assertEqual(game.display_word(), "_ _ _ _ _")

    def test_correct_guess(self):
        game = Hangman(answer="apple")
        game.guess('a')
        self.assertEqual(game.display_word(), "a _ _ _ _")

    def test_wrong_guess_deducts_life(self):
        game = Hangman(answer="apple", lives= 5)
        game.guess('x')
        self.assertEqual(game.lives, 4)   # life should be deducted
        self.assertEqual(game.display_word(), "_ _ _ _ _")  # still all underscores

    def test_win_condition(self):
        game = Hangman(answer="apple")
        for letter in set("apple"):
            game.guess(letter)
        self.assertTrue(game.is_won())
        self.assertFalse(game.is_lost())

    def test_lose_condition(self):
        game = Hangman(answer="apple", lives=2)
        game.guess('b')
        game.guess('c')
        self.assertFalse(game.is_won())
        self.assertTrue(game.is_lost())

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", 
 24, in test_win_condition
    self.assertTrue(game.is_won())
                    ^^^^^^^^^^^
AttributeError: 'Hangman' object has no attribute 'is_won'

----------------------------------------------------------------------
Ran 5 tests in 0.002s
```

Update hangman.py, add is_won( ) method to verify winning conditions and is_lost( ) method to verify losing conditions.



```python
class Hangman:
    def __init__(self, answer, lives = 6):
        self.answer = answer.lower()
        self.guessed = set()
        self.lives = lives

    def display_word(self):
        return ' '.join([c if c in self.guessed else '_' for c in self.answer])

    def guess(self, letter):
        letter = letter.lower()
        if letter in self.guessed:
            return
        self.guessed.add(letter)
        if letter not in self.answer:
            self.lives -= 1

    def is_won(self):
        return all(c in self.guessed or not c.isalpha() for c in self.answer)

    def is_lost(self):
        return self.lives <= 0
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Step 5: Support the input of phrases with spaces (e.g.hello world)

Add code to test phrases like "hello world" in test_hangman.py.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):
        game = Hangman(answer="hello world")
        self.assertEqual(game.display_word(), "_ _ _ _ _  _ _ _ _ _")
        game.guess('h')
        game.guess('e')
        game.guess('l')
        game.guess('o')
        game.guess('w')
        game.guess('r')
        game.guess('d')
        self.assertEqual(game.display_word(), "h e l l o  w o r l d")
        self.assertTrue(game.is_won())

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
    self.assertEqual(game.display_word(), "_ _ _ _ _  _ _ _ _ _")
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: '_ _ _ _ _ _ _ _ _ _ _' != '_ _ _ _ _  _ _ _ _ _'
- _ _ _ _ _ _ _ _ _ _ _
?         --
+ _ _ _ _ _  _ _ _ _ _
?       ++


----------------------------------------------------------------------
Ran 6 tests in 0.001s

FAILED (failures=1)
```

Update display_word() in hangman.py to handle spaces.

```python
class Hangman:
    def __init__(self, answer, lives = 6):
        self.answer = answer.lower()
        self.guessed = set()
        self.lives = lives

    def display_word(self):
        return ' '.join([c if (c in self.guessed or not c.isalpha()) else '_' for c in self.answer])

    def guess(self, letter):
        letter = letter.lower()
        if letter in self.guessed:
            return
        self.guessed.add(letter)
        if letter not in self.answer:
            self.lives -= 1

    def is_won(self):
        return all(c in self.guessed or not c.isalpha() for c in self.answer)

    def is_lost(self):
        return self.lives <= 0
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
Ran 6 tests in 0.000s

OK
```

Step 6: Store guessed letters in a list

Add code in the test file, to test whether guessed letters can be stored in a list, which provides game information and helps players track their guesses.

```
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):
        game = Hangman(answer="apple")
        game.guess('a')
        game.guess('e')
        game.guess('x')
        guessed = game.get_guessed_letters()
        self.assertEqual(set(guessed), {'a', 'e', 'x'})
if __name__ == '__main__':
    unittest.main()
```

Run the test file, and the output is as follows:

```
    guessed = game.get_guessed_letters()
              ^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Hangman' object has no attribute 'get_guessed_letters'

----------------------------------------------------------------------
Ran 7 tests in 0.001s

FAILED (errors=1)
```

Add get_guessed_letters( ) method in the Hangman class in hangman.py. It aims to store the guessed letter in a list.

```python
class Hangman:
    def __init__(self, answer, lives = 6):
        self.answer = answer.lower()
        self.guessed = set()
        self.lives = lives

    def display_word(self):
        return ' '.join([c if (c in self.guessed or not c.isalpha()) else '_' for c in self.answer])

    def guess(self, letter):
        letter = letter.lower()
        if letter in self.guessed:
            return
        self.guessed.add(letter)
        if letter not in self.answer:
            self.lives -= 1

    def is_won(self):
        return all(c in self.guessed or not c.isalpha() for c in self.answer)

    def is_lost(self):
        return self.lives <= 0

    def get_guessed_letters(self):
        return list(self.guessed)
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
(base) iammin@iammindeMacBook-Air Software Unit Testing Report % /opt/homebrew/bin/python3 "/Users/iammin/Documer
/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py"
.......
----------------------------------------------------------------------
Ran 7 tests in 0.000s

OK
```

Step 7: Display “Game Over” message

Update the test_lose_condition( ) in the test file for testing “Game Over” messages.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):
        game = Hangman(answer="apple", lives=2)
        game.guess('b')
        game.guess('c')
        self.assertFalse(game.is_won())
        self.assertTrue(game.is_lost())
        self.assertEqual(game.game_over_message(), "Game Over! You lost.")

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):...


if __name__ == "__main__":
    unittest.main()
```

Run the test file, and it shows that game_over_message( ) is missing.

```
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", l
 33, in test_lose_condition
    self.assertEqual(game.game_over_message(), "Game Over! You lost.")
                     ^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Hangman' object has no attribute 'game_over_message'

----------------------------------------------------------------------
Ran 7 tests in 0.001s

FAILED (errors=1)
```

Add the game_over_message() method in hangman.py.

```python
class Hangman:
    def __init__(self, answer, lives = 6):
        self.answer = answer.lower()
        self.guessed = set()
        self.lives = lives

    def display_word(self):...

    def guess(self, letter):...

    def is_won(self):...

    def is_lost(self):
        return self.lives <= 0

    def game_over_message(self):
        if self.is_lost():
            return "Game Over! You lost."
        return ""

    def get_guessed_letters(self):...
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
----------------------------------------------------------------------
Ran 7 tests in 0.000s

OK
```

Step 8: Implement Level Selection

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):...

    def test_basic_level(self):
        game = Hangman(level='basic', word_list=['cat', 'dog', 'banana', 'grape'])
        self.assertIn(game.answer, ['cat', 'dog', 'banana', 'grape'])

    def test_intermediate_level(self):
        game = Hangman(level='intermediate', phrase_list=['good morning', 'thank you', 'how are you'])
        self.assertIn(game.answer, ['good morning','thank you', 'how are you'])

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output shows that Hangman needs to define two levels.

```
E..E.....
======================================================================
ERROR: test_basic_level (__main__.TestHangman.test_basic_level)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", li
 57, in test_basic_level
    game = Hangman(level='basic', word_list=['cat', 'dog', 'banana', 'grape'])
TypeError: Hangman.__init__() got an unexpected keyword argument 'level'. Did you mean 'lives'?

======================================================================
ERROR: test_intermediate_level (__main__.TestHangman.test_intermediate_level)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", li
 61, in test_intermediate_level
    game = Hangman(level='intermediate', phrase_list=['good morning', 'thank you', 'how are you'])
TypeError: Hangman.__init__() got an unexpected keyword argument 'level'. Did you mean 'lives'?

----------------------------------------------------------------------
Ran 9 tests in 0.002s

FAILED (errors=2)
```

Update hangman.py to include random, modify __init__() to accept levels, and add generate_answer() in the Hangman class.

```python
import random

class Hangman:
    def __init__(self, answer = None, level='basic', lives=6, word_list=None, phrase_list=None):
        self.level = level
        self.lives = lives
        self.word_list = word_list if word_list else ['apple', 'banana', 'orange']
        self.phrase_list = phrase_list if phrase_list else ['ice cream', 'hot dog', 'green tea']
        self.answer = answer if answer is not None else self.generate_answer()
        self.guessed = set()

    def generate_answer(self):
        if self.level == 'basic':
            return random.choice(self.word_list)
        elif self.level == 'intermediate':
            return random.choice(self.phrase_list)
        else:
            raise ValueError("Invalid level")


    def display_word(self):...

    def guess(self, letter):...

    def is_won(self):...

    def is_lost(self):...

    def game_over_message(self):...

    def get_guessed_letters(self):...
```

Execute hangman.py. It runs successfully without errors. Then run the

test file, and the output shows the test passes, indicating that the function meets the requirements.

```
.........
----------------------------------------------------------------------
Ran 9 tests in 0.000s

OK
```

Step 9: Validate Words/Phrases from Dictionary

Add tests to check that the generated word/phrase is valid according to a dictionary.

```python
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):...

    def test_basic_level(self):...

    def test_intermediate_level(self):...

    def test_answer_in_dictionary(self):
        dictionary = {'apple', 'banana', 'orange', 'ice cream', 'hot dog', 'green tea'}
        game = Hangman(answer='banana', dictionary=dictionary)
        self.assertTrue(game.is_valid_answer(game.answer))
        with self.assertRaises(ValueError):
            Hangman(answer='notaword', dictionary=dictionary)

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
======================================================================
ERROR: test_answer_in_dictionary (__main__.TestHangman.test_answer_in_dictionary)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", line
 66, in test_answer_in_dictionary
    game = Hangman(answer='banana', dictionary=dictionary)
TypeError: Hangman.__init__() got an unexpected keyword argument 'dictionary'

----------------------------------------------------------------------
Ran 10 tests in 0.001s

FAILED (errors=1)
```

Update Hangman class to include a dictionary parameter, add is_valid_answer() and update generate_answer( ).

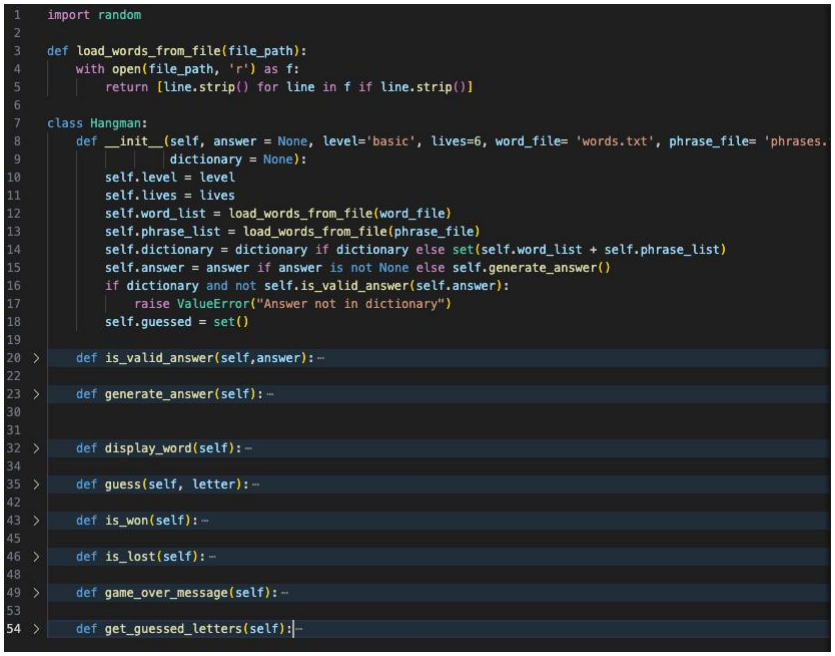
```python
import random

class Hangman:
    def __init__(self, answer = None, level='basic', lives=6, word_list=None, phrase_list=None,
                 dictionary = None):
        self.level = level
        self.lives = lives
        self.word_list = word_list if word_list else ['apple', 'banana', 'orange']
        self.phrase_list = phrase_list if phrase_list else ['ice cream', 'hot dog', 'green tea']
        self.dictionary = dictionary if dictionary else set(self.word_list + self.phrase_list)
        self.answer = answer if answer is not None else self.generate_answer()
        if dictionary and not self.is_valid_answer(self.answer):
            raise ValueError("Answer not in dictionary")
        self.guessed = set()

    def is_valid_answer(self,answer):
        return answer in self.dictionary

    def generate_answer(self):
        if self.level == 'basic':
            return random.choice(self.word_list)
        elif self.level == 'intermediate':
            return random.choice(self.phrase_list)
        else:
            raise ValueError("Invalid level")


    def display_word(self):...

    def guess(self, letter):...

    def is_won(self):...

    def is_lost(self):...

    def game_over_message(self):...

    def get_guessed_letters(self):...
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
..........
----------------------------------------------------------------------
Ran 10 tests in 0.000s

OK
```

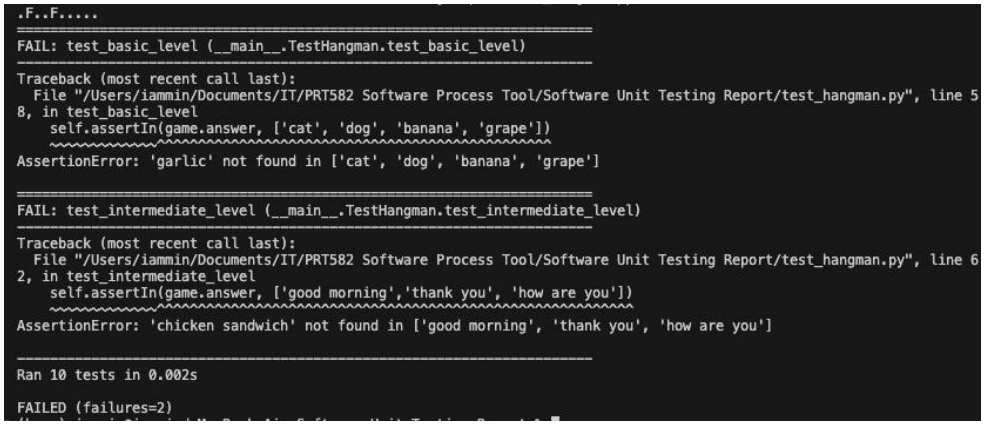
However, the default dictionary is too small, so in hangman.py, two files—words.txt and phrases.txt—each containing 100 randomly

generated words or phrases, are loaded. The modifications are as follows:



```python
import random

def load_words_from_file(file_path):
    with open(file_path, 'r') as f:
        return [line.strip() for line in f if line.strip()]

class Hangman:
    def __init__(self, answer = None, level='basic', lives=6, word_file= 'words.txt', phrase_file= 'phrases.
                 dictionary = None):
        self.level = level
        self.lives = lives
        self.word_list = load_words_from_file(word_file)
        self.phrase_list = load_words_from_file(phrase_file)
        self.dictionary = dictionary if dictionary else set(self.word_list + self.phrase_list)
        self.answer = answer if answer is not None else self.generate_answer()
        if dictionary and not self.is_valid_answer(self.answer):
            raise ValueError("Answer not in dictionary")
        self.guessed = set()

    def is_valid_answer(self,answer):...

    def generate_answer(self):...


    def display_word(self):...

    def guess(self, letter):...

    def is_won(self):...

    def is_lost(self):...

    def game_over_message(self):...

    def get_guessed_letters(self):...
```

Run the test file again, the results show that the tests fail.



```
.F..F.....
======================================================================
FAIL: test_basic_level (__main__.TestHangman.test_basic_level)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", line 5
8, in test_basic_level
    self.assertIn(game.answer, ['cat', 'dog', 'banana', 'grape'])
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 'garlic' not found in ['cat', 'dog', 'banana', 'grape']

======================================================================
FAIL: test_intermediate_level (__main__.TestHangman.test_intermediate_level)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", line 6
2, in test_intermediate_level
    self.assertIn(game.answer, ['good morning','thank you', 'how are you'])
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 'chicken sandwich' not found in ['good morning', 'thank you', 'how are you']

----------------------------------------------------------------------
Ran 10 tests in 0.002s

FAILED (failures=2)
```

This is because the assertions in test_basic_level() and test_intermediate_level() in the test file use a fixed small list, while the game has updated the answer list. The answers generated randomly from the txt files. As a result, some words or phrases fall outside the fixed

list, so the test file needs to be modified as follows:

```
import unittest
from hangman import Hangman

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):...

    def test_basic_level(self):
        game = Hangman(level='basic')
        self.assertIn(game.answer, game.word_list)

    def test_intermediate_level(self):
        game = Hangman(level='intermediate')
        self.assertIn(game.answer, game.phrase_list)

    def test_answer_in_dictionary(self):...

if __name__ == "__main__":
    unittest.main()
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
----------------------------------------------------------------------
Ran 10 tests in 0.002s

OK
```

Step 10: Timer for 15 Seconds per Guess

Import time in the test file, ddd test_guess_timeout_deducts_life( ) :

```python
import unittest
from hangman import Hangman
import time

class TestHangman(unittest.TestCase):
    def test_initial_underscores(self):...

    def test_correct_guess(self):...

    def test_wrong_guess_deducts_life(self):...

    def test_win_condition(self):...

    def test_lose_condition(self):...

    def test_phrase_support(self):...

    def test_show_guessed_letters(self):...

    def test_basic_level(self):...

    def test_intermediate_level(self):...

    def test_answer_in_dictionary(self):...

    def test_guess_timeout_deducts_life(self):
        game = Hangman(answer="apple", lives=2)

        start_time = time.time()
        time.sleep(0.1)
        # Set timeout to 0.05 seconds to ensure life is deducted
        game.check_timeout(start_time, timeout=0.05)
        self.assertEqual(game.lives, 1)

        # Trigger timeout again
        game.check_timeout(start_time, timeout=0.05)
        self.assertEqual(game.lives, 0)
        self.assertTrue(game.is_lost())

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and the output is as follows:

```
======================================================================
ERROR: test_guess_timeout_deducts_life (__main__.TestHangman.test_guess_timeout_deducts_life)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", line 78, in test_guess_timeout_deducts_lif
e
    game.check_timeout(start_time, timeout=0.05)
    ^^^^^^^^^^^^^^^^^^
AttributeError: 'Hangman' object has no attribute 'check_timeout'

----------------------------------------------------------------------
Ran 11 tests in 0.106s

FAILED (errors=1)
```

Update Hangman class with self.last_guess_time = None and check_timeout( ) method:

```python
import random
import time

def load_words_from_file(file_path):
    with open(file_path, 'r') as f:
        return [line.strip() for line in f if line.strip()]

class Hangman:
    def __init__(self, answer = None, level='basic', lives=6, word_file= 'words.txt', phrase_file= 'phrases.
                 dictionary = None):
        self.level = level
        self.lives = lives
        self.word_list = load_words_from_file(word_file)
        self.phrase_list = load_words_from_file(phrase_file)
        self.dictionary = dictionary if dictionary else set(self.word_list + self.phrase_list)
        self.answer = answer if answer is not None else self.generate_answer()
        if dictionary and not self.is_valid_answer(self.answer):
            raise ValueError("Answer not in dictionary")
        self.guessed = set()
        self.last_guess_time = None

    def is_valid_answer(self,answer):...

    def generate_answer(self):...


    def display_word(self):...

    def guess(self, letter):...

    def check_timeout(self, start_time, timeout=15):
        if time.time() - start_time > timeout:
            self.lives -= 1
            self.last_guess_time = time.time()

    def is_won(self):...

    def is_lost(self):...

    def game_over_message(self):...

    def get_guessed_letters(self):...
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes, indicating that the function meets the requirements.

```
Ran 11 tests in 0.106s

OK
```

At this point, the basic requirement for the Hangman game has been mostly implemented. However, to make the game run smoothly and meet the requirement that “the timer must be shown,” a GUI can be implemented.

Step 11: Minimal GUI Test

Add a simple GUI test in the test file, importing tkinter and HangmanGUI.

```python
import unittest
from hangman import Hangman
from hangman import HangmanGUI
import tkinter as tk
import time

class TestHangman(unittest.TestCase):…

class TestHangmanGUI(unittest.TestCase):
    def test_gui_launch(self):
        try:
            from hangman import HangmanGUI
        except ImportError:
            self.fail("HangmanGUI class does not exist in hangman.py")

        root = tk.Tk()
        try:
            app = HangmanGUI(root)
            self.assertIsNotNone(app)
        finally:
            root.destroy()

if __name__ == "__main__":
    unittest.main()
```

Run the test file, and an error occurs.

```
(base) iammin@iammindeMacBook-Air Software Unit Testing Report % /usr/local/bin/python3 "/Users/iammin/Documents
/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py"
Traceback (most recent call last):
  File "/Users/iammin/Documents/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py", l
ine 3, in <module>
    from hangman import HangmanGUI
ImportError: cannot import name 'HangmanGUI' from 'hangman' (/Users/iammin/Documents/IT/PRT582 Software Process
Tool/Software Unit Testing Report/hangman.py). Did you mean: 'Hangman'?
(base) iammin@iammindeMacBook-Air Software Unit Testing Report %
```

Add a minimal HangmanGUI class in hangman.py to pass the

test.

```python
import random
import time
import tkinter as tk

def load_words_from_file(file_path):…

class Hangman:…


class HangmanGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Hangman Game")

        self.welcome_label = tk.Label(root, text="Welcome to Hangman!", font=("Arial", 16))
        self.welcome_label.pack(pady=20)

        self.start_button = tk.Button(root, text="Start Game", command=self.start_game)
        self.start_button.pack(pady=10)

    def start_game(self):
        print("Game started!")

if __name__ == "__main__":
    root = tk.Tk()
    app = HangmanGUI(root)
    root.mainloop()
```

Execute hangman.py. It runs successfully without errors. Then run the test file, and the output shows the test passes.

```
(base) iammin@iammindeMacBook-Air Software Unit Testing Report % /usr/local/bin/python3 "/Users/iammin/Documents
/IT/PRT582 Software Process Tool/Software Unit Testing Report/test_hangman.py"
............
----------------------------------------------------------------------
Ran 12 tests in 0.413s
```

Step 12: Full GUI implementation with game loop and timer

Since the Hangman class already handles core logic, reuse existing tests.

Next, implement the full HangmanGUI in hangman.py including: Level selection; Timer countdown, Information prompts, Game loop.

```python
import random
import time
import tkinter as tk
from tkinter import messagebox

def load_words_from_file(file_path):…

class Hangman:…


class HangmanGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Hangman Game")
        self.root.geometry("400x300")
        self.time_left = 15
        self.timer_id = None
        self.timer_running = None
        self.hangman = None
        self.level = None

        self.welcome_frame = tk.Frame(root)
        self.game_frame = tk.Frame(root)
        self.show_welcome()

    def show_welcome(self):
        self.welcome_frame.pack(fill="both", expand=True)

        welcome_label = tk.Label(self.welcome_frame, text="Welcome to Hangman!", font=("Arial", 18))
        welcome_label.pack(pady=20)

        basic_button = tk.Button(self.welcome_frame, text="Basic", font=("Arial", 14),
                                 command=lambda: self.start_game(level='basic'))
        basic_button.pack(pady=5)
```

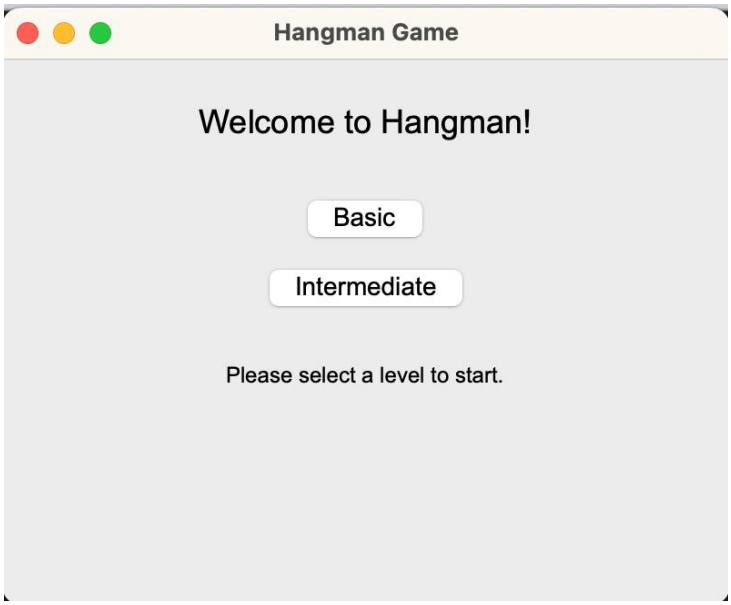
```python
class HangmanGUI:
    def __init__(self, root):…

    def show_welcome(self):
        self.welcome_frame.pack(fill="both", expand=True)

        welcome_label = tk.Label(self.welcome_frame, text="Welcome to Hangman!", font=("Arial", 18))
        welcome_label.pack(pady=20)

        basic_button = tk.Button(self.welcome_frame, text="Basic", font=("Arial", 14),
                                 command=lambda: self.start_game(level='basic'))
        basic_button.pack(pady=5)

        intermediate_button = tk.Button(self.welcome_frame, text="Intermediate", font=("Arial", 14),
                                        command=lambda: self.start_game(level='intermediate'))
        intermediate_button.pack(pady=5)

        instruction_label = tk.Label(self.welcome_frame, text="Please select a level to start.", font
        instruction_label.pack(pady=20)

    def start_game(self, level):
        self.level = level
        self.welcome_frame.pack_forget()
        self.hangman = Hangman(level=self.level)
        self.build_game_ui()
        self.game_frame.update_idletasks()
        self.update_display()
        self.reset_timer()

    def build_game_ui(self):
        self.game_frame.pack(fill="both", expand=True)

        self.word_label = tk.Label(self.game_frame, text="", font=("Arial", 20))
        self.word_label.pack(pady=10)
```

```
70  class HangmanGUI:
111     def build_game_ui(self):
112         self.game_frame.pack(fill="both", expand=True)
113
114         self.word_label = tk.Label(self.game_frame, text="", font=("Arial", 20))
115         self.word_label.pack(pady=10)
116
117         self.lives_label = tk.Label(self.game_frame, text="", font=("Arial", 14))
118         self.lives_label.pack()
119
120         self.timer_label = tk.Label(self.game_frame, text="Time left: 15", font=("Arial", 14))
121         self.timer_label.pack()
122
123         self.entry = tk.Entry(self.game_frame, font=("Arial", 14))
124         self.entry.pack(pady=10)
125         self.entry.bind("<Return>", lambda event: self.make_guess())
126
127         guess_button = tk.Button(self.game_frame, text="Guess", font=("Arial", 14), command=self.make_g
128         guess_button.pack()
129
130     def make_guess(self):
131         letter = self.entry.get().strip().lower()
132         self.entry.delete(0, tk.END)
133
134         if not letter or len(letter) != 1 or not letter.isalpha():
135             messagebox.showwarning("Invalid", "Please enter a single letter.")
136             return
137
138         result = self.hangman.guess(letter)
139         if result == "wrong":
140             messagebox.showinfo("Wrong", f"'{letter}' is not in the word! Life lost.")
141         elif result == "already_guessed":
142             messagebox.showinfo("Oops", f"You already guessed '{letter}'.")
143
```

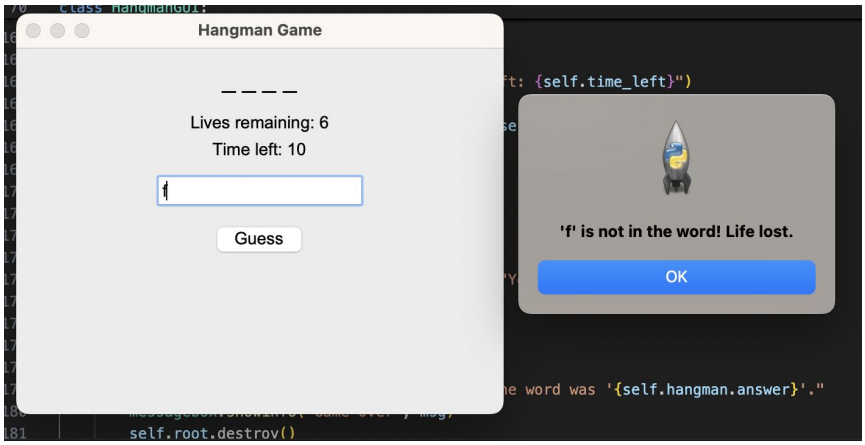
```
70  class HangmanGUI:
130     def make_guess(self):
131         letter = self.entry.get().strip().lower()
132         self.entry.delete(0, tk.END)
133
134         if not letter or len(letter) != 1 or not letter.isalpha():
135             messagebox.showwarning("Invalid", "Please enter a single letter.")
136             return
137
138         result = self.hangman.guess(letter)
139         if result == "wrong":
140             messagebox.showinfo("Wrong", f"'{letter}' is not in the word! Life lost.")
141         elif result == "already_guessed":
142             messagebox.showinfo("Oops", f"You already guessed '{letter}'.")
143
144         self.update_display()
145
146         if self.hangman.is_won():
147             self.end_game(won=True)
148         elif self.hangman.is_lost():
149             self.end_game(won=False)
150
151         self.reset_timer()
152
153     def update_display(self):
154         self.word_label.config(text=self.hangman.display_word())
155         self.lives_label.config(text=f"Lives remaining: {self.hangman.lives}")
156
157     def reset_timer(self):
158         self.time_left = 15
159         if self.timer_id is not None:
160             self.root.after_cancel(self.timer_id)
161         self.update_timer()
162
```

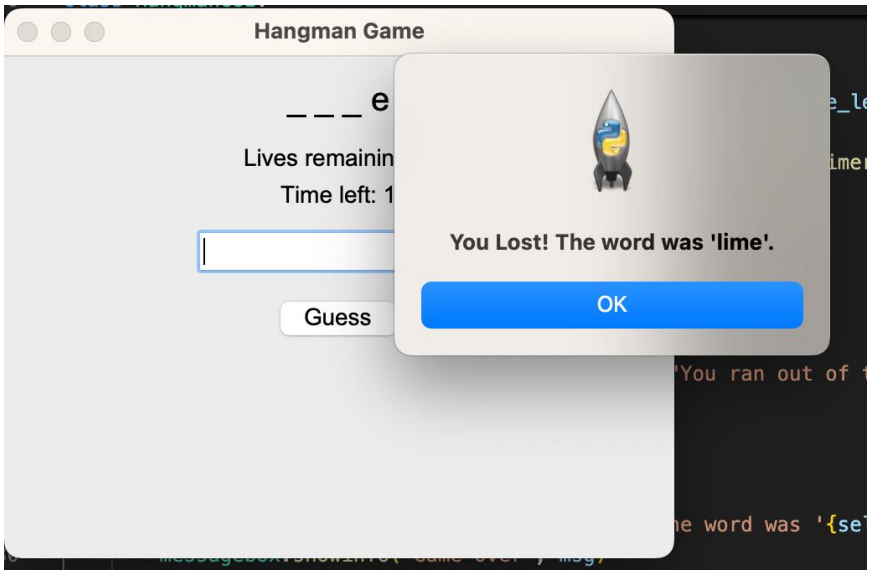
```
70  class HangmanGUI:
130     def make_guess(self):
131         letter = self.entry.get().strip().lower()
132         self.entry.delete(0, tk.END)
133
134         if not letter or len(letter) != 1 or not letter.isalpha():
135             messagebox.showwarning("Invalid", "Please enter a single letter.")
136             return
137
138         result = self.hangman.guess(letter)
139         if result == "wrong":
140             messagebox.showinfo("Wrong", f"'{letter}' is not in the word! Life lost.")
141         elif result == "already_guessed":
142             messagebox.showinfo("Oops", f"You already guessed '{letter}'.")
143
144         self.update_display()
145
146         if self.hangman.is_won():
147             self.end_game(won=True)
148         elif self.hangman.is_lost():
149             self.end_game(won=False)
150
151         self.reset_timer()
152
153     def update_display(self):
154         self.word_label.config(text=self.hangman.display_word())
155         self.lives_label.config(text=f"Lives remaining: {self.hangman.lives}")
156
157     def reset_timer(self):
158         self.time_left = 15
159         if self.timer_id is not None:
160             self.root.after_cancel(self.timer_id)
161         self.update_timer()
162
```

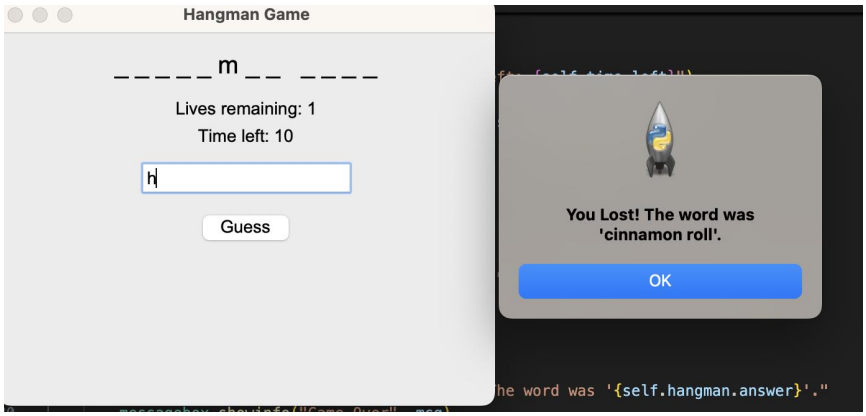
Running the code will display the following GUI window:

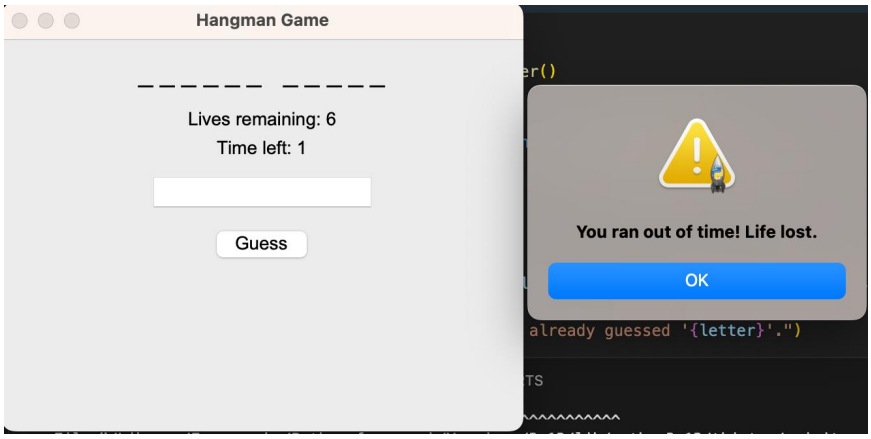


Run the program and test “Basic” and “Intermediate” levels. Wrong guesses, life lost ,correct guesses, timeout, and game over messages, timer and remaining life are all displayed correctly.


Hangman Game
_ _ _ _
Lives remaining: 6
Time left: 10
f
Guess
'f' is not in the word! Life lost.
OK
self.root.destroy()


Hangman Game
_ _ _ e
Lives remainin
Time left: 1
Guess
You Lost! The word was 'lime'.
OK


Hangman Game
_ _ _ _ _ m _ _ _ _ _ _
Lives remaining: 1
Time left: 10
h
Guess
You Lost! The word was
'cinnamon roll'.
OK



Since the interface looks somewhat empty and plain, Hangman ASCII art can be added to enhance the visual appeal. First, add the following code in the build_game method of HangmanGUI:

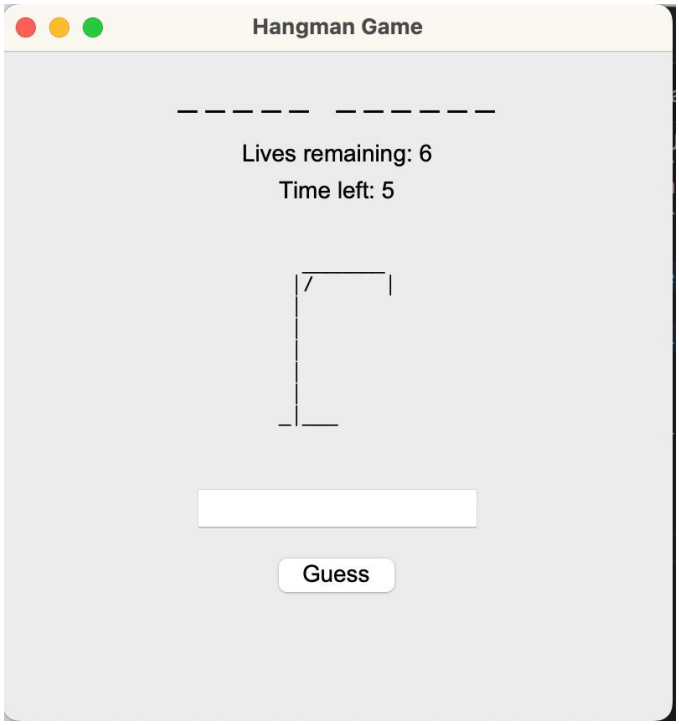
```
class HangmanGUI:
    def __init__(self, root):…

    def show_welcome(self):…

    def start_game(self, level):…

    def build_game_ui(self):
        self.game_frame.pack(fill="both", expand=True)

        self.word_label = tk.Label(self.game_frame, text="", font=("Arial", 20))
        self.word_label.pack(pady=10)

        self.lives_label = tk.Label(self.game_frame, text="", font=("Arial", 14))
        self.lives_label.pack()

        self.timer_label = tk.Label(self.game_frame, text="Time left: 15", font=("Arial", 14))
        self.timer_label.pack()

        self.ascii_label = tk.Label(self.game_frame, text="", font=("Courier", 12), justify="left")
        self.ascii_label.pack(pady=10)

        self.entry = tk.Entry(self.game_frame, font=("Arial", 14))
        self.entry.pack(pady=10)
        self.entry.bind("<Return>", lambda event: self.make_guess())

        guess_button = tk.Button(self.game_frame, text="Guess", font=("Arial", 14), command=self.make_g
        guess_button.pack()
```

Next, modify the update_display method. Since the game starts with 6 lives, set 7 stages to gradually complete the Hangman figure as lives are lost:

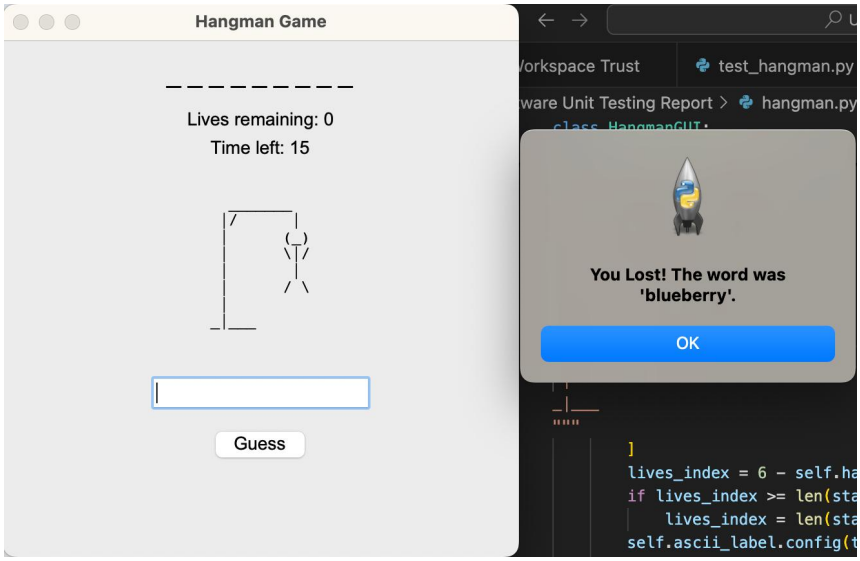
```
70  class HangmanGUI:
156     def update_display(self):
157         self.word_label.config(text=self.hangman.display_word())
158         self.lives_label.config(text=f"Lives remaining: {self.hangman.lives}")
159
160         stages = [
161 """
162    _______
163   |/      |
164   |
165   |
166   |
167   |
168   |
169 _|___
170 """,
171 """
172    _______
173   |/      |
174   |      (_)
175   |
176   |
177   |
178   |
179 _|___
180 """,
181 """
182    _______
183   |/      |
184   |      (_)
185   |      \|
186   |
187   |
188   |
```

```
70  class HangmanGUI:
156     def update_display(self):
206   |       |
207   |
208   |
209 _|___
210 """,
211 """
212    _______
213   |/      |
214   |      (_)
215   |      \|/
216   |       |
217   |      /
218   |
219 _|___
220 """,
221 """
222    _______
223   |/      |
224   |      (_)
225   |      \|/
226   |       |
227   |      / \\
228   |
229 _|___
230 """
231         ]
232         lives_index = 6 - self.hangman.lives
233         if lives_index >= len(stages):
234             lives_index = len(stages) - 1
235         self.ascii_label.config(text=stages[lives_index])
236
237         self.root.update_idletasks()
```

Finally, call self.update_display() in the end_game() method, and adjust the GUI window size in __init__. After making these changes, run the program; the game initially stage appears as follows:



When all lives are lost, it displays as follows:



The above completes the development of a simple Hangman game implemented using Test-Driven Development (TDD).

Conclusion

Through developing this Hangman game, I learned how to apply Test-Driven Development (TDD) to incrementally build a program, which is highly valuable for real-world software development. Firstly, TDD helps improve code quality by promoting modular, maintainable, and well-structured code, which is easier to debug and extend. Secondly, it provides developers with continuous feedback through automated tests, boosting confidence that the software behaves as expected. Fixing bugs early reduces later debugging effort, saves development time, and enhances overall efficiency. TDD also encourages thinking from the end-user perspective, helping to create a product that better meets user needs.

Although TDD may initially slow down development due to the time spent writing tests and learning the process, it can accelerate progress in the long term and improve test coverage, code cohesion, and maintainability. The experience of using TDD in this project demonstrated that it also acts as living documentation, supports future refactoring, and ensures that design decisions are verified through tests. These findings are consistent with previous research, which suggests that TDD improves internal and

external software quality, enhances code cohesion, reduces complexity, and helps developers maintain a clear focus during incremental development [1].

In summary, the Hangman project showed that adopting TDD not only improves software reliability and maintainability but also strengthens a developer's ability to anticipate and address potential problems, providing both technical and practical benefits in software development. The code for this project is available on GitHub at: https://github.com/iamminapril/A-simple-Hangman-game-with-TDD/tree/main

Reference

[1] Z. Khanam and M. N. Ahsan, "Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls," *International Journal of Applied Engineering Research*, vol. 12, no. 18, pp. 7705–7716, 2017. [Online]. Available: http://www.ripublication.com