PRT582
Assignment 2
Software Unit Testing Report

S344917

Introduction

This Software Unit Testing Report is made to show how to use Test-Driven Development and automated unit testing tool to create and effectively code a game of 'Hangman'. Python is the chosen programming language for this software and testing because there is a built-in framework called 'unittest'.

Process

Test-Driven Development (TDD) is the process of creating a series of tests to use before fully coding a program. These tests will check the initial codes and work to improve it, if any issues arise regarding the functionality of the codes. These tests were made with 'unittest', which checks the code with certain conditions and return it as successful or failure. To make these tests run automatically, the conditions and inputs must be hardcoded to check if it works as intended. Once the code works as intended, the code that was tested can be used for the game's code.

Figure 1. First set of tests for the code

```python
class TestHangmanGame(unittest.TestCase):

    #Test for Basic
    @patch('main.random.choice', return_value='potato')  #Word for testing: potato
    def test_basic_level_word(self, mock_choice):
        level = 'basic'
        word = choose_word(level)
        self.assertEqual(word, 'potato')

    #Test for Intermediate
    @patch('main.random.choice', return_value='individual assessment')  #Phrase for testing: individual assessment
    def test_intermediate_level_phrase(self, mock_choice):
        level = 'intermediate'
        phrase = choose_word(level)
        self.assertEqual(phrase, 'individual assessment')

    #Test if underscores are correctly displayed for a word
    @patch('main.random.choice', return_value='potato')  #Word for testing: potato
    def test_underscores_display(self, mock_choice):
        game = HangmanGame('potato')
        display_word = game.get_display_word()
        self.assertEqual(display_word, '______')  #Potato is represented as ______

    #Test that correct guesses reveal the correct letters in the word
    @patch('main.random.choice', return_value='potato')  #Word for testing: potato
    def test_correct_guess_reveal(self, mock_choice):
        game = HangmanGame('potato')
        game.guess('p')
        display_word = game.get_display_word()
        self.assertEqual(display_word, 'p_____')  #The p is revealed, others are still underscores
```

Figure 1. Shows the tests code for a few of the requirements, such as checking the levels of the game (basic/intermediate), underscores displaying for the word, and checking if a correct guess would reveal the correct letters in the word.

Figure 2. Second set of tests for the code

```
#Test that lives are deducted when guessing wrong letters
@patch('main.random.choice', return_value='potato')  #Word for testing: potato
def test_incorrect_guess_deduct_life(self, mock_choice):
    game = HangmanGame('potato')
    game.guess('x')
    self.assertEqual(game.lives, 5)  #Life should go from 6 to 5

#Test that hangman ends when the player runs out of lives
@patch('main.random.choice', return_value='potato')  #Word for testing: potato
def test_game_over_when_lives_zero(self, mock_choice):
    game = HangmanGame('potato', lives=1)  #Lives changed to 1 for testing
    game.guess('x')
    self.assertTrue(game.is_game_over())  #The game should end
    self.assertFalse(game.has_won())

#Test that the game ends when the player guesses the word correctly
@patch('main.random.choice', return_value='potato')  #Word for testing: potato
def test_game_over_when_word_guessed(self, mock_choice):
    game = HangmanGame('potato')
    game.guess('p')
    game.guess('o')
    game.guess('t')
    game.guess('a')
    game.guess('t')
    game.guess('o')
    self.assertTrue(game.is_game_over())  # Game should be over when word is guessed correctly
    self.assertTrue(game.has_won())  # Player won by guessing the word correctly
```

Figure 2. Continues the first set of tests, and shows more test codes for checking that lives go down after guessing the wrong letters, hangman ends if the user does not have anymore lives, and the hangman ends if the user completely gets all the letter of the word.

These tests were used before the game code was fully operational with all the features. These lines of codes were used to test the codes to be used for the game code, and were essential as these tests were for requirements of the hangman.

Conclusion

This software unit testing report has shown me how to use Test-Driven Development and automated unit testing tool to start creating code. Although it requires planning to know what features need to be tested, or what type of code needs to be used for the testing, I think it is a good way to make sure that the code is functioning just as intended before it is implemented into the main code.

Github Link: https://github.com/Jhon-Carlo/Hangman