HANGMAN GAME

La Hoang Quy

s334197

1. Introduction
    a. System overview

Hangman game can be defined as a game for two players where one player tries to guess letters of a word and other player will record failed attempts by drawing a gallows and someone hanging on it line by line.

b. Purpose

The objective of this document is to provide a detailed description and all testing requirements for hangman game. The purpose will be shown in the document. Furthermore, it will explain what type of test case will be performed manually and automatically, how it is performed, what the results of each test case are, and how testing process will be conducted in hangman game.

2. Test plan

a) Feature to be tested

There are some features need will be tested in the software unit testing document.

- The first feature will be word generation which will generate a random word.
- The second feature is comparing a letter chosen by a player with a generated word.
- The third feature will be comparing a random letter with a word in a generated word.
- The next feature will be checking if a winning text is generated or not.
- The fifth feature will be checking if a losing text is generated or not.
- The sixth feature will be checking if a reset function is working correctly.
- The final feature will be checking if a picture of hangman is generated or not.

b) Feature not to be tested

The feature is not tested in the unit testing program is getting a position of a mouse. The reason for that is this feature is imported from a Pygame library, so the Pygame framework does not support testing of this feature and it is very hard to get exactly the position of clicking the mouse to perform this test automatically.

However, this feature will be tested manually, by clicking randomly a letter, if this letter is chosen correctly, all places of that letter will appear, if a letter is chosen incorrectly, this letter will disappeared to show that a position of a mouse is received correctly. This test will test a position of clicking the mouse is working correctly or not.

Furthermore, the second feature is not tested in this unit testing program is the life of the player. The main reason for that, the life of the player is dependent on the position of clicking the mouse, so the pygame library framework does not support this feature and it is very hard to get exactly the position of clicking the mouse to perform this test automatically.

But this feature will be tested manually by choosing any given words in a game, if a word chosen by a play is correct, all places of that letter will appear, and if a word chosen by a play is incorrect, a picture of hanging a man will be drawn. This picture will represent a number of chances that a player can have to guess a word. If a picture is drawn completely and a player can’t guess a word, the player will lose. This test will test a life a player is deducted.

c) Testing tools and environment

To test hangman game, there are several software and hardware components need to be set up to perform the test.

- Hardware requirement

There are several hardware components included in this test. They are laptop, desktop computer, and WIFI connection.

- Software requirement

There are several software components needed for this test. They include operating systems, such as Windows and Mac. Furthermore, there will be a need to install python 3.7 with 64bits and there are some python libraries involved in this test.

3. Test cases
    a. Case: test_randomWord
        i. Purposes:

The purpose of this test is to make sure that a word is generated correctly.

ii. Inputs:

Inputs for this will be randomWord function in the hangman program.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be a tested word from a word list. The test will pass if the input is matched with the expected output. The test will fail if the input is not matched with the expected output.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be the randomWord function from hangman program will be imported.
- The second step will be importing the tested word from a word list.
- The final step will be the program will run to test this functions in the test file to check if a word from the randomWord function is matched with a tested word.

b. Case: test_hang
    i. Purposes

The purpose of this test is to check if a guess word from a play is lower case or not. It will help to compare with the word from a list which will be generated.

ii. Inputs:

Inputs for this test will be a word from a player. The test will check if the guess word from a player is in lower case.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be a word chosen by a player but is in lower case. The test will pass if input is matched with expected output. The test will fail if input is not match with expected output.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a guess word from a player.
- The second step will be importing one lower case letter from a word list and one upper case letter from a word list.
- The final step will be the test program will run and check if a word chosen by a player is in lower case or not.

c. Case: hiding the letter

i. Purposes

The purpose of this test is to check each letter in a word from a list is hided or not.

ii. Inputs:

Inputs for this test will be a word from a list.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be each letter in a word will be replaced with "_". The test will pass if input is matched with expected output. The test will fail if input is not matched with expected output.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a word from a list word.
- The second step will be importing a number of "_" which is matched with a number of letter in a word.
- The final step will be the program will run and check if a number of letter in a word is matched with a number of "_".

d. Case: test match word

i. Purposes

The purpose of this test is to check if the player's chosen letter exists in the answer, then all places in the answer where that letter appear will be revealed.

ii. Inputs:

Inputs for this test will be a letter in a chosen word.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be all places in the word where that letter appear will be revealed.

The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a word from a word list.
- The second step will be choosing a letter in that word.
- The final step will be the program will check if all places in the answer where that letter appear will be revealed or not.

e. Case: test not match word

i. Purposes

The purpose of this test is to check if letters are not in a chosen word, then all places in the answer where that letter appear will not be revealed.

ii. Inputs:

Inputs for this test will be a random letter which is not in a chosen word.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be all places in the word where that letter appear will not be revealed.

The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a word from a word list.
- The second step will be choosing a random letter which is not in that word.
- The final step will be the program will check if all places in the answer where that letter appear will be revealed or not.

f. Case: test hangman pics

i. Purposes

The purpose of this test is to check if hangman picture is displayed correctly or not.

ii. Inputs:

Inputs for this test will be a hangman pics function

iii. Expected outputs, pass and fail criteria:

Expected outputs will be true if the function is working correctly. The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a function from the main function.
- The second step will be writing the function to check if the function performs correctly.
- The final step will be the program will run and check if the hangman pics function is working correctly or not.

g. Case: test not match word

i. Purposes

The purpose of this test is to check if letters are not in a chosen word, then all places in the answer where that letter appear will not be revealed.

ii. Inputs:

Inputs for this test will be a random letter which is not in a chosen word.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be all places in the word where that letter appear will not be revealed.

The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a word from a word list.
- The second step will be choosing a random letter which is not in that word.
- The final step will be the program will check if all places in the answer where that letter appear will be revealed or not.

h. Case: test the winning and losing text

i. Purposes

The purpose of this test is to check if the winning and losing text is displayed correctly or not.

ii. Inputs:

Inputs for this test will be a winning and losing text variable from the main function.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be true if the winning and losing text variable is displayed correctly. The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

- The first step will be importing a winning and losing text variable.
- The second step will be writing the function to check if a winning and losing text variable performs correctly.
- The final step will be the program will run and check if the winning and losing text variable is working correctly or not.

i. Case: test the reset function

i. Purposes

The purpose of this test is to check if the reset function is working properly or not.

ii. Inputs:

Inputs for this test will be a reset function from the main function.

iii. Expected outputs, pass and fail criteria:

Expected outputs will be true if the reset function is working correctly. The test will pass if the input is matched with the output and all places are displayed correctly. The test will fail if the input is matched with the output and all places are not displayed correctly.

iv. Test procedures:

There are several steps to perform this test.

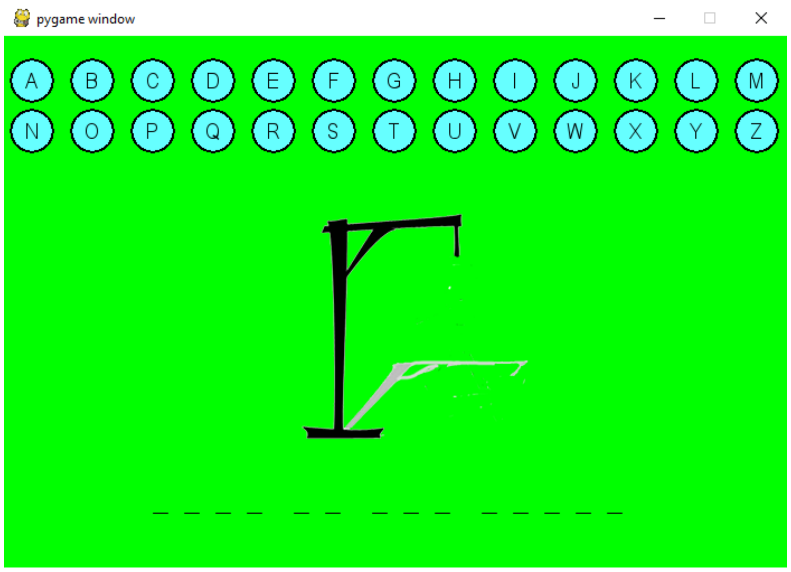
- The first step will be importing a reset function from the main function.
- The second step will be writing the function to check if a reset function is working correctly.
- The final step will be the program will run and check if a reset function is working correctly or not.

4. Hangman program:

a. Overview:

Hangman program is written in the python program with a library Pygame. By importing the Pygame library, all the functions of the Pygame library will be used. Furthermore, the code is written in IDLE python 3.7. It can help to run the code quick and when the program is run, there will be a separate window created by Pygame. Pygame library is the most crucial, popular and powerful library in python to make any 2D games. With the help of the Pygame library, any clicking even button is used which will make the game easier to play and it is easier to code any 2D games with the help of Pygame library. Furthermore, Pygame library can help to design a beautiful layout for the game.

The image below will demonstrate how hangman game looks like:

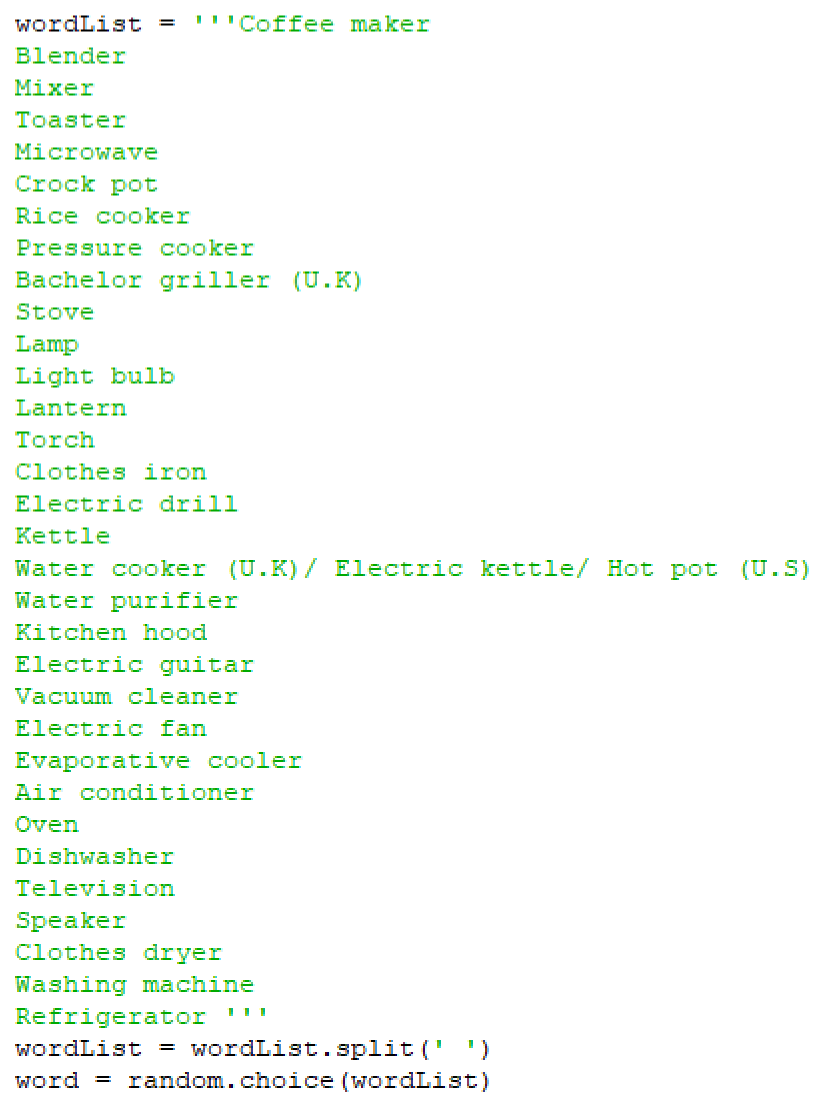


b. How to play

As can be seen from the image, at the start of the game, the actual word is hided and each letter of a word is replaced by “_”. By clicking a button representing a letter, this letter will be imported and it will compared with all the letter in a given word. If the letter is chosen correctly, all places of that letter will be revealed otherwise it will be hidden until a player guess a correct word. Furthermore, there are twenty-six letters are shown at the top of the image for a player to key in to guess the unknown word. Each time a player guesses a correct letter, this letter will be shown instead of “_”. However, if a player guesses incorrect letters, there will be a head appear at the middle of the image and if a picture of hanging a man is display fully and a player did not guess a word correctly, the player will lose. A player will win this game if all letters are guessed correctly, and a player will lose if an image of hanging a man appeared.

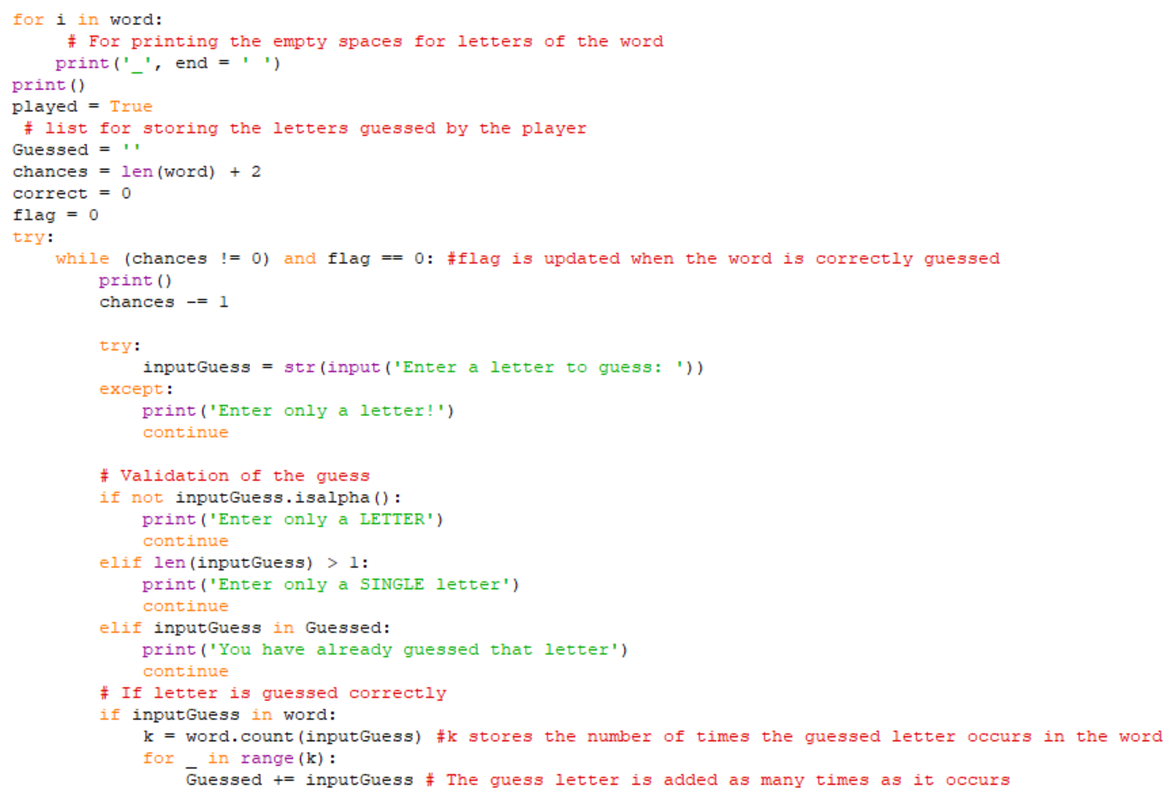
c. Original hangman code:

To build a complete game, there are many functions need to be used. However, there are three main functions to build this program. The first function is to generate a word randomly is shown below:



```
wordList = '''Coffee maker
Blender
Mixer
Toaster
Microwave
Crock pot
Rice cooker
Pressure cooker
Bachelor griller (U.K)
Stove
Lamp
Light bulb
Lantern
Torch
Clothes iron
Electric drill
Kettle
Water cooker (U.K)/ Electric kettle/ Hot pot (U.S)
Water purifier
Kitchen hood
Electric guitar
Vacuum cleaner
Electric fan
Evaporative cooler
Air conditioner
Oven
Dishwasher
Television
Speaker
Clothes dryer
Washing machine
Refrigerator '''
wordList = wordList.split(' ')
word = random.choice(wordList)
```

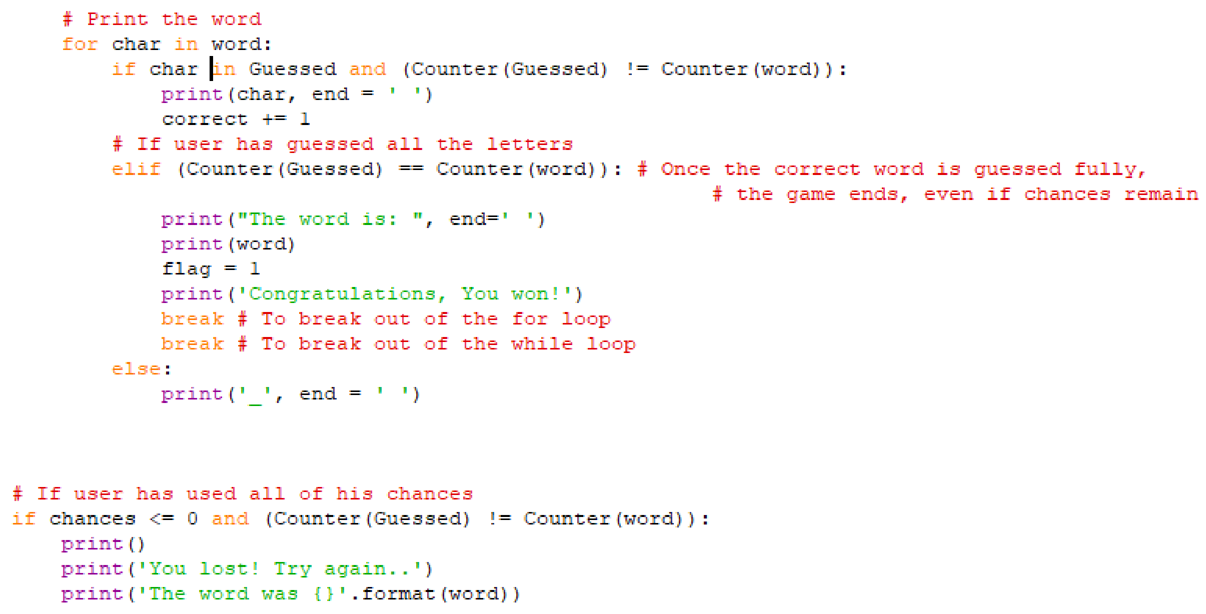
The second function is to print letters and replace them with "_" and compare them with the guessed letter and the third function is to compare each letter imported from a player with letters inside the generated word. The second and third function is shown below:



```
for i in word:
     # For printing the empty spaces for letters of the word
    print('_', end = ' ')
print()
played = True
 # list for storing the letters guessed by the player
Guessed = ''
chances = len(word) + 2
correct = 0
flag = 0
try:
    while (chances != 0) and flag == 0: #flag is updated when the word is correctly guessed
        print()
        chances -= 1

        try:
            inputGuess = str(input('Enter a letter to guess: '))
        except:
            print('Enter only a letter!')
            continue

        # Validation of the guess
        if not inputGuess.isalpha():
            print('Enter only a LETTER')
            continue
        elif len(inputGuess) > 1:
            print('Enter only a SINGLE letter')
            continue
        elif inputGuess in Guessed:
            print('You have already guessed that letter')
            continue
        # If letter is guessed correctly
        if inputGuess in word:
            k = word.count(inputGuess) #k stores the number of times the guessed letter occurs in the word
            for _ in range(k):
                Guessed += inputGuess # The guess letter is added as many times as it occurs
```



```
    # Print the word
    for char in word:
        if char in Guessed and (Counter(Guessed) != Counter(word)):
            print(char, end = ' ')
            correct += 1
        # If user has guessed all the letters
        elif (Counter(Guessed) == Counter(word)): # Once the correct word is guessed fully,
                                                  # the game ends, even if chances remain
            print("The word is: ", end=' ')
            print(word)
            flag = 1
            print('Congratulations, You won!')
            break # To break out of the for loop
            break # To break out of the while loop
        else:
            print('_', end = ' ')


# If user has used all of his chances
if chances <= 0 and (Counter(Guessed) != Counter(word)):
    print()
    print('You lost! Try again..')
    print('The word was {}'.format(word))
```

As can be seen from the first function, the list of word is attached to the program and it will make the program harder to follow and debug if there are errors in the program.

Furthermore, there are many repetition of if, else-if and else statement so it will be hard to read and understand.

In addition there are also so many try-catch functions, the try-catch function is usually used to catch errors so these try-catch functions need to be replaced with other functions which will be easily to read and follow. In addition, the design of the game is not looking nice, so the game needs to be designed much nicer.

Moreover, there are too many times to guess a word, so the times to guess a word needs to be reduced and there is no function to handle a chosen letter which is not a word so when a player choose a letter which is not a word the program will break.

In conclusion, from all of reasons above, the code needs to be refactored to improve the program.

d. Solution

There are many solutions to refactor the code. However, there are four main solutions.

The first solution is to replace all the try catch function with the if- else statements. The reason for that is try catch method is only used to catch error. The if-else statement will help the code much easier to read.

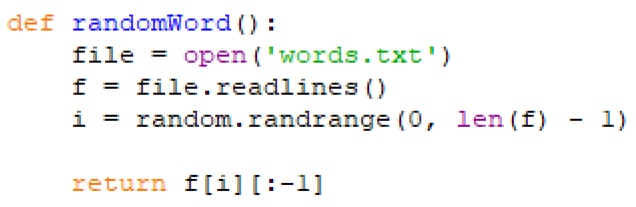
The second solution is to draw a game in a separate screen with buttons representing all words from the keyboard imported inside the game. By clicking the button in the game, players will not import any letters which are not words. Furthermore it will also improve the design of the game.

The next solution will be drawing a picture of hanging a man when a player guess wrong letter will reduce the number of chances to guess into six times.

The final solution is to separate some codes into a different functions instead of combining all codes inside a main function, it will help to easily not only read but also check errors. Furthermore, it will help to reduce the chances of making mistakes.

e. Refactored hangman code:

The image below will show the first function on how to generate a word:

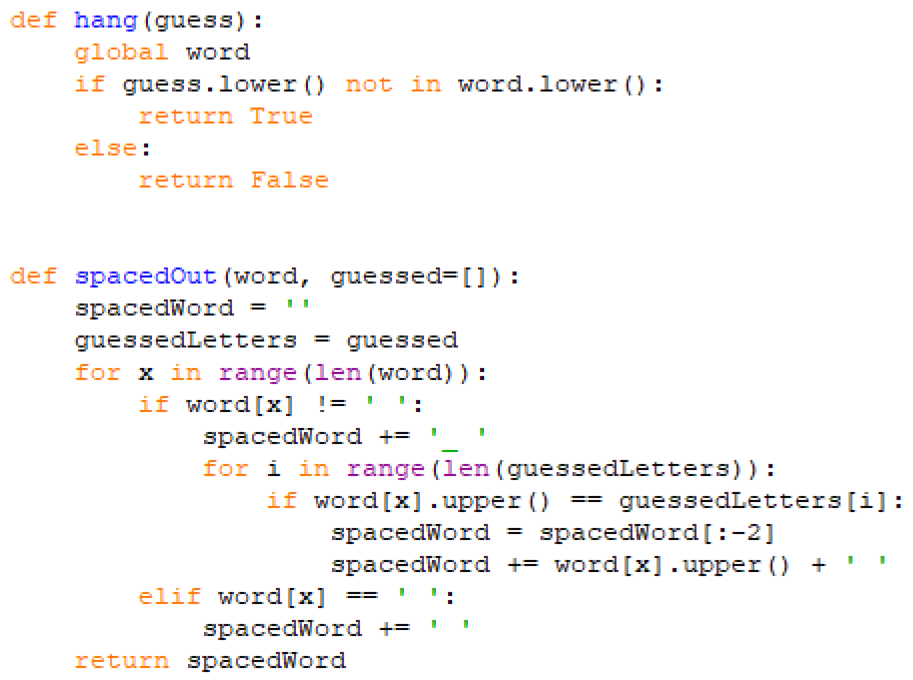


```
def randomWord():
    file = open('words.txt')
    f = file.readlines()
    i = random.randrange(0, len(f) - 1)

    return f[i][:-1]
```

As can be seen from the image above, words are stored inside the words.txt file. Moreover, by have a list of word on a separate file, it will keep the program shorter so it will be easier to follow. Furthermore, by having a separated function to generate word, it will be easier to debug if there are some mistakes related to the program.

The picture below will show the second function on how to import letters from users and the and third function on how to compare imported letters from users with letters in a generated word.



```
def hang(guess):
    global word
    if guess.lower() not in word.lower():
        return True
    else:
        return False


def spacedOut(word, guessed=[]):
    spacedWord = ''
    guessedLetters = guessed
    for x in range(len(word)):
        if word[x] != ' ':
            spacedWord += '_ '
            for i in range(len(guessedLetters)):
                if word[x].upper() == guessedLetters[i]:
                    spacedWord = spacedWord[:-2]
                    spacedWord += word[x].upper() + ' '
        elif word[x] == ' ':
            spacedWord += ' '
    return spacedWord
```

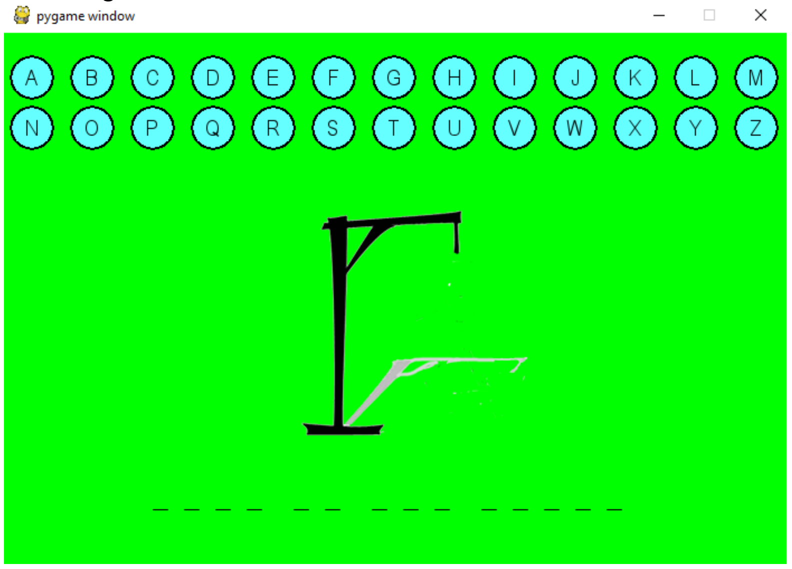
As can be seen from the image above, functions are declared separately so it will be easier to read and understand. The hang function will import the letters from the users and stored in the guess variable. The spaceOut function will compare letters from the generated word with the guessed letter from users stored in the guess variable. Furthermore, in for loop, if letters in a generated word is not blank, it will be replaced with the "_" and if letters in a generated word is blank it will be replaced with blank space. Both the hang function and spaceOut function will be called in the main function to run. As can be seen from the image above, by using separated function, it will not only help to improve the readability of code, but also eliminate many repeated codes.

The picture below will illustrate how to get a click from users and how to replace "_" with a letter if users guess correct word. Furthermore, the image below also illustrates how the life is deducted if users guess all wrong and how words are displayed if users guess all correct.

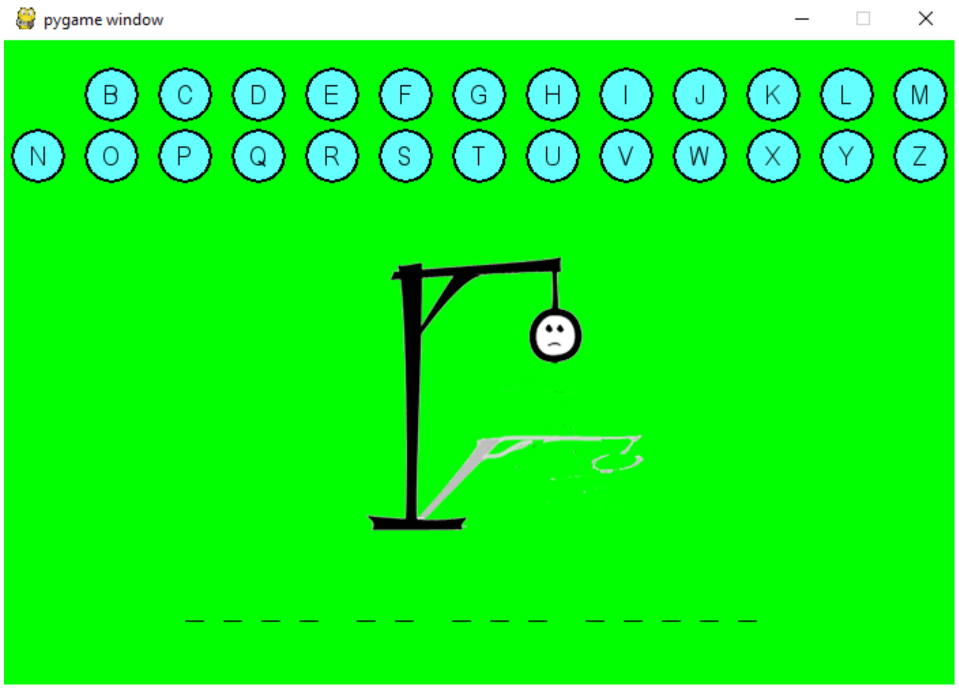
```
if event.type == pygame.MOUSEBUTTONDOWN:
    clickPos = pygame.mouse.get_pos()
    letter = button_pressed(clickPos[0], clickPos[1])
    if letter != None:
        guessed.append(chr(letter))
        buttons[letter - 65][4] = False
        if hang(chr(letter)):
            if limbs != 5:
                limbs += 1
            else:
                end()
        else:
            print(spacedOut(word, guessed))
            if spacedOut(word, guessed).count('_') == 0:
                end(True)
```

f. Output.

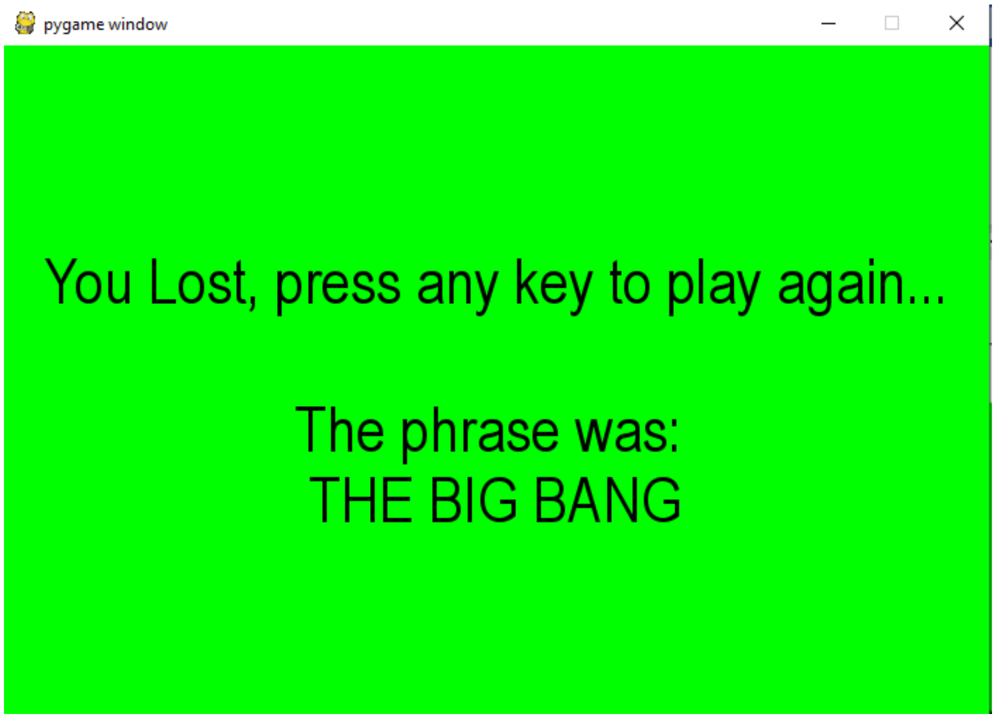
Hangman program is written in the python program with a library pygame and demonstrated in the image below:



As can be seen from the image, letters in a word is shown as a "_" to hide the actual word. To play the game, a player need to press the letter from the keyboard shown inside the image. Once a letter is pressed, it will disappeared. If player guess the first time wrong the head will appear as shown in the image below:



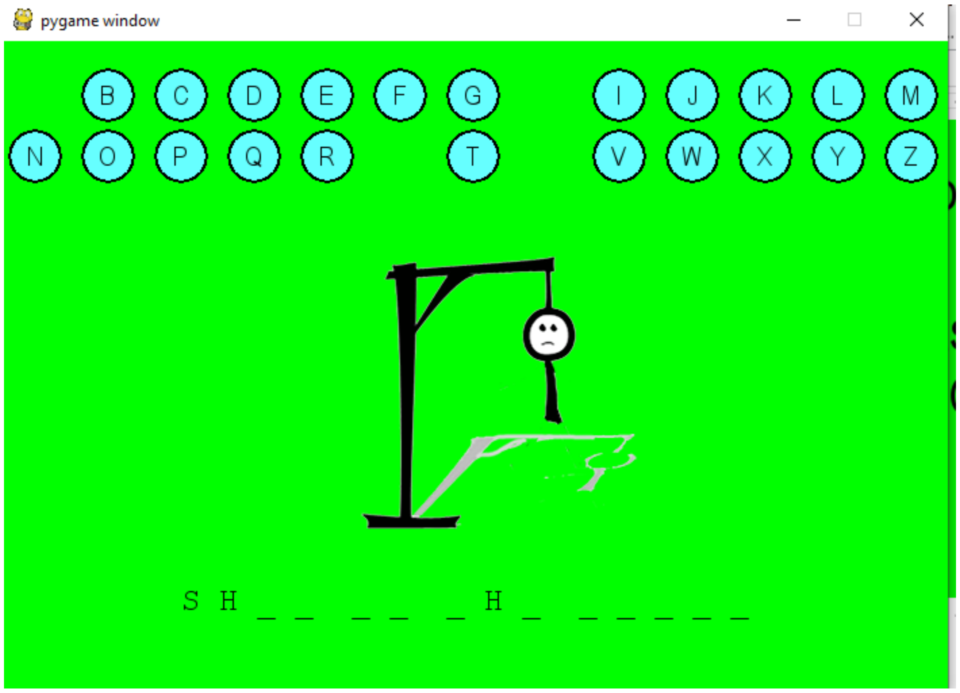
If a player continues to guess wrong, the body of a person will appear until the full image of hanging a person will appear. If the full image of hanging a person appears, the message loser will be sent out and full word will appear. The image below will show that



You Lost, press any key to play again...

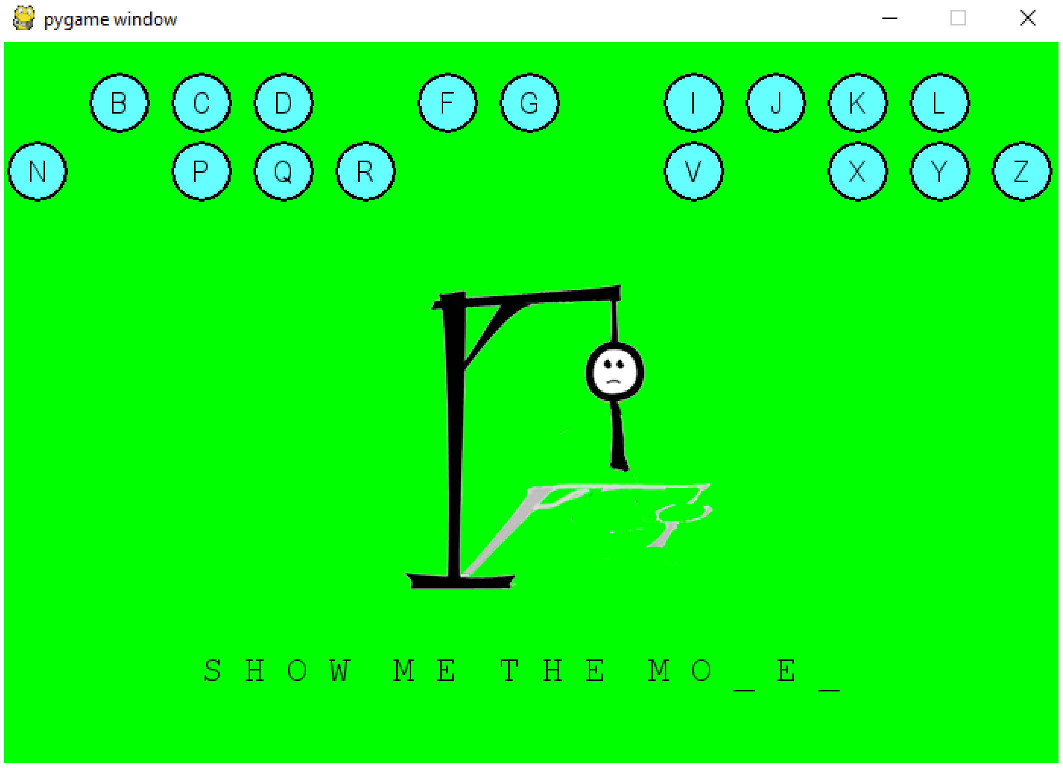
The phrase was:
THE BIG BANG

However, if a player guesses correct letters, these letters will appear instead of underscore. The image below will show that



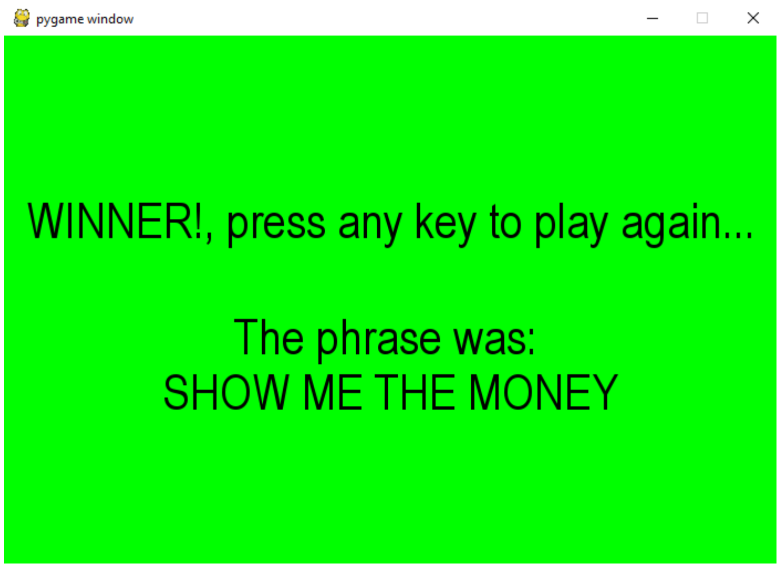
As can be seen from the image the H letter is chosen correctly so there will be two H letters appear and they replace with underscore. A is not a correct letter, so it will not appear.

If a player continues to guess letters correctly, more letters will appear. The image below will show that



Similarly, E letters are chosen correctly, therefore, three E letters appear instead of the underscore. The situation is the same as letters O, M, and H.

And if a player guesses all correct letters the complete word will be displayed and the winning message will appear. The picture below will show that



WINNER!, press any key to play again...

The phrase was:
SHOW ME THE MONEY

The game will not end in here. It does not matter that players win or loses a game, if they want to play more, they can press any key to continue. Each time, this game starts or restarts, a different word or phrase will be generated so a player will not easily remember a word or phrase. It will make the game more interesting.

To quit the game, a user will need to press the "x" button at the corner as shown in the above image.

g. Github link:

The main code is in the main.py which is used to run the program. Furthermore, there will be a test.py where all the unit code test is inside. Moreover, there is a file called words.txt where all the generated words are inside. There will be a software unit testing report.pdf and the code before refactoring is in the file called before.py. Finally all the file is uploaded in the Github link which is shown below:

https://github.com/HoangQuyla/hangman/tree/master

5. Appendix

a. Test logs before coding

Log for unit testing

- Test results

UT-1.1: A word is generated randomly---Fail
UT-1.2: Missing letter is represented in blank space--Fail
UT-1.3: Letters in all blank spaces will appear if player choose correct letters---------------Fail
UT-1.4: Player guess wrong, player's life will be deducted---------------------------------------Fail
UT-1.5: Message will appear if player can't guess correct word---------------------------------Fail

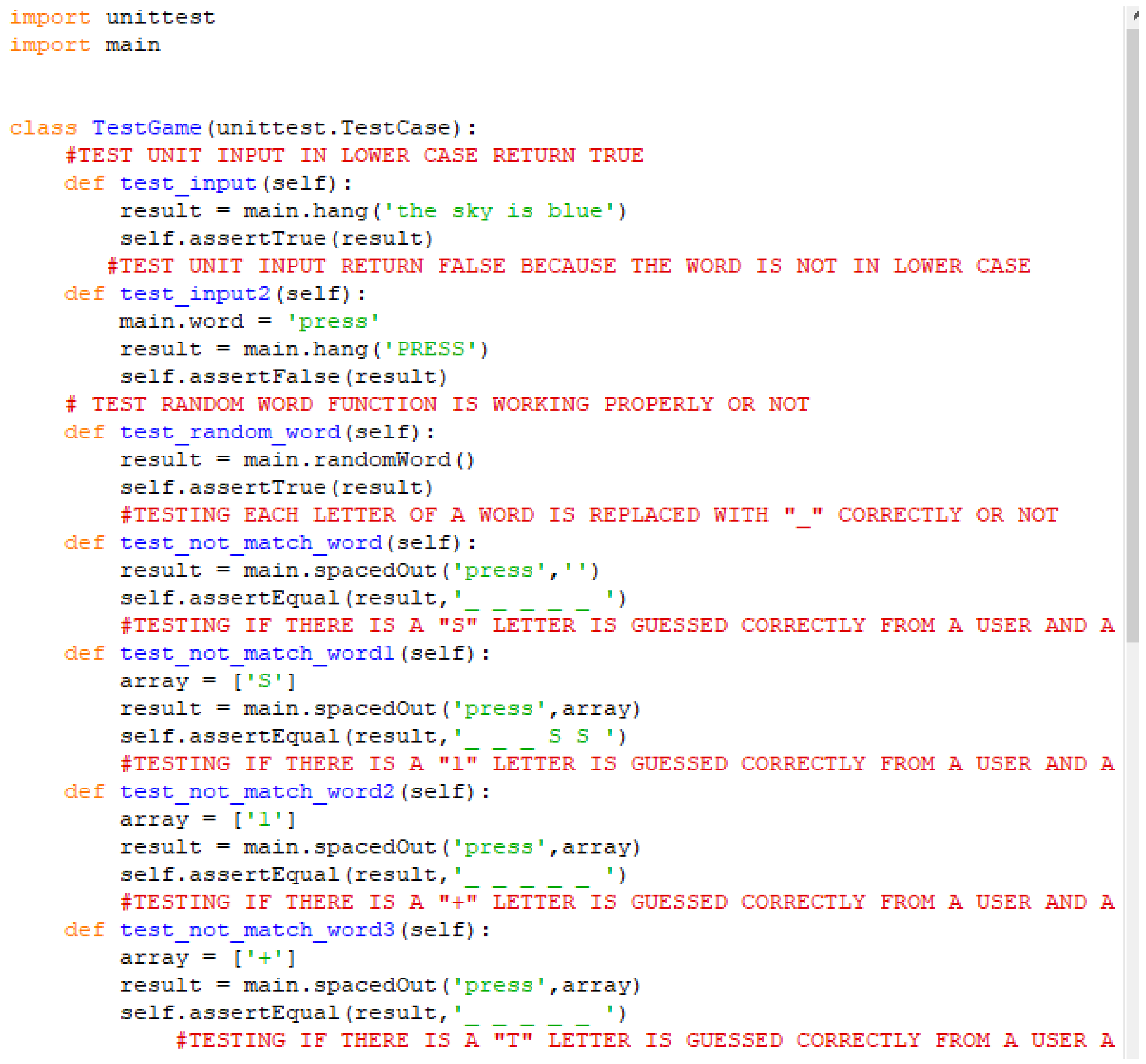
b. Appendix Test logs after coding

Log for unit testing

- Test results

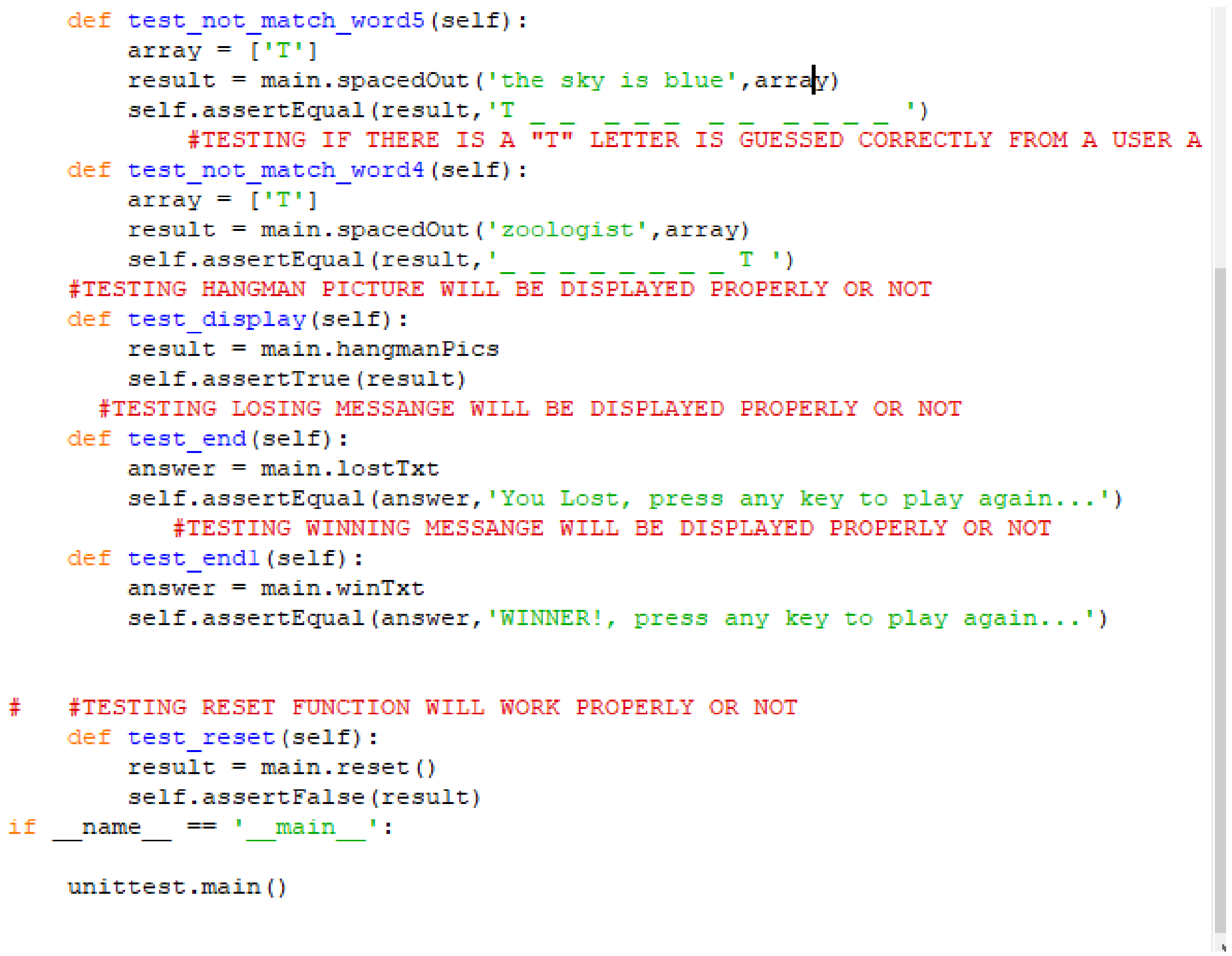
UT-1.1: A word is generated randomly -- Pass
UT-1.2: Missing letter is represented in blank space --- Pass
UT-1.3: Letters in all blank spaces will appear if player choose correct letters---------------- Pass
UT-1.4: Player guess wrong, player’s life will be deducted --------------------------------------- Pass
UT-1.5: Message will appear if player can’t guess correct word on time ----------------------- Pass
UT-1.6: Reset function is working correctly --- Pass
UT-1.7: A picture of hanging a man is displayed correctly -- Pass
UT-1.8: Message will appear if player guess correct word on time ----------------------------- Pass
UT-1.9: If players guess incorrect letters, letters in all blank spaces will not appear -------- Pass
UT-1.10: If players guess a number, letters in all blank spaces will not appear ---------------Pass
UT-1.11: If players guess a “+”, letters in all blank spaces will not appear ---------------------Pass

c. Unit test code:



```python
import unittest
import main


class TestGame(unittest.TestCase):
    #TEST UNIT INPUT IN LOWER CASE RETURN TRUE
    def test_input(self):
        result = main.hang('the sky is blue')
        self.assertTrue(result)
       #TEST UNIT INPUT RETURN FALSE BECAUSE THE WORD IS NOT IN LOWER CASE
    def test_input2(self):
        main.word = 'press'
        result = main.hang('PRESS')
        self.assertFalse(result)
    # TEST RANDOM WORD FUNCTION IS WORKING PROPERLY OR NOT
    def test_random_word(self):
        result = main.randomWord()
        self.assertTrue(result)
        #TESTING EACH LETTER OF A WORD IS REPLACED WITH "_" CORRECTLY OR NOT
    def test_not_match_word(self):
        result = main.spacedOut('press','')
        self.assertEqual(result,'_ _ _ _ _ ')
        #TESTING IF THERE IS A "S" LETTER IS GUESSED CORRECTLY FROM A USER AND A
    def test_not_match_word1(self):
        array = ['S']
        result = main.spacedOut('press',array)
        self.assertEqual(result,'_ _ _ S S ')
        #TESTING IF THERE IS A "1" LETTER IS GUESSED CORRECTLY FROM A USER AND A
    def test_not_match_word2(self):
        array = ['1']
        result = main.spacedOut('press',array)
        self.assertEqual(result,'_ _ _ _ _ ')
        #TESTING IF THERE IS A "+" LETTER IS GUESSED CORRECTLY FROM A USER AND A
    def test_not_match_word3(self):
        array = ['+']
        result = main.spacedOut('press',array)
        self.assertEqual(result,'_ _ _ _ _ ')
            #TESTING IF THERE IS A "T" LETTER IS GUESSED CORRECTLY FROM A USER A
```



```
    def test_not_match_word5(self):
        array = ['T']
        result = main.spacedOut('the sky is blue',array)
        self.assertEqual(result,'T _ _  _ _ _  _ _  _ _ _ _ ')
            #TESTING IF THERE IS A "T" LETTER IS GUESSED CORRECTLY FROM A USER A
    def test_not_match_word4(self):
        array = ['T']
        result = main.spacedOut('zoologist',array)
        self.assertEqual(result,'_ _ _ _ _ _ _ _ T ')
    #TESTING HANGMAN PICTURE WILL BE DISPLAYED PROPERLY OR NOT
    def test_display(self):
        result = main.hangmanPics
        self.assertTrue(result)
      #TESTING LOSING MESSANGE WILL BE DISPLAYED PROPERLY OR NOT
    def test_end(self):
        answer = main.lostTxt
        self.assertEqual(answer,'You Lost, press any key to play again...')
           #TESTING WINNING MESSANGE WILL BE DISPLAYED PROPERLY OR NOT
    def test_end1(self):
        answer = main.winTxt
        self.assertEqual(answer,'WINNER!, press any key to play again...')


#   #TESTING RESET FUNCTION WILL WORK PROPERLY OR NOT
    def test_reset(self):
        result = main.reset()
        self.assertFalse(result)
if __name__ == '__main__':

    unittest.main()
```

d. Unit test Run by code:

```
.
----------------------------------------------------------------------
Ran 13 tests in 0.049s

OK
```

e. Full code before refactoring

import random

from collections import Counter

wordList = '''Coffee maker

Blender

Mixer

Toaster

Microwave

```
Crock pot
Rice cooker
Pressure cooker
Bachelor griller (U.K)
Stove
Lamp
Light bulb
Lantern
Torch
Clothes iron
Electric drill
Kettle
Water cooker (U.K)/ Electric kettle/ Hot pot (U.S)
Water purifier
Kitchen hood
Electric guitar
Vacuum cleaner
Electric fan
Evaporative cooler
Air conditioner
Oven
Dishwasher
Television
Speaker
Clothes dryer
Washing machine
Refrigerator '''
wordList = wordList.split(' ')
word = random.choice(wordList)
```

```
if __name__=='__main__':
    print('Guess the word! Hint word is a name of a household appliance')
    for i in word:
         # For printing the empty spaces for letters of the word
        print('_', end = ' ')
    print()
    played = True
     # list for storing the letters guessed by the player
    Guessed = ''
    chances = len(word) + 2
    correct = 0
    flag = 0
    try:
        while (chances != 0) and flag == 0: #flag is updated when the word is correctly guessed
            print()
            chances -= 1

            try:
                inputGuess = str(input('Enter a letter to guess: '))
            except:
                print('Enter only a letter!')
                continue

            # Validation of the guess
            if not inputGuess.isalpha():
                print('Enter only a LETTER')
                continue
            elif len(inputGuess) > 1:
```

```
            print('Enter only a SINGLE letter')
            continue
        elif inputGuess in Guessed:
            print('You have already guessed that letter')
            continue
        # If letter is guessed correctly
        if inputGuess in word:
            k = word.count(inputGuess) #k stores the number of times the guessed letter occurs in the
word
            for _ in range(k):
                Guessed += inputGuess # The guess letter is added as many times as it occurs


        # Print the word
        for char in word:
            if char in Guessed and (Counter(Guessed) != Counter(word)):
                print(char, end = ' ')
                correct += 1
            # If user has guessed all the letters
            elif (Counter(Guessed) == Counter(word)): # Once the correct word is guessed fully,
                                                        # the game ends, even if chances remain
                print("The word is: ", end=' ')
                print(word)
                flag = 1
                print('Congratulations, You won!')
                break # To break out of the for loop
                break # To break out of the while loop
            else:
                print('_', end = ' ')
    # If user has used all of his chances
    if chances <= 0 and (Counter(Guessed) != Counter(word)):
        print()
        print('You lost! Try again..')
        print('The word was {}'.format(word))

except KeyboardInterrupt:
    print()
    print('Bye! Try again.')
    exit()
```

f. Full code after refactoring

```
import pygame
import random
##Displaying window size and drawing a window of a game
pygame.init()
win_height = 480
winWidth = 700
win=pygame.display.set_mode((winWidth,win_height))

# initialize global variables/constants #
BLACK = (0,0, 0)
WHITE = (255,255,255)
RED = (255,0, 0)
GREEN = (0,255,0)
BLUE = (0,0,255)
LIGHT_BLUE = (102,255,255)
```

```
btn_font = pygame.font.SysFont("arial", 20)

guess_font = pygame.font.SysFont("monospace", 24)

lost_font = pygame.font.SysFont('arial', 45)

word = ''

buttons = []

guessed = []

lostTxt = 'You Lost, press any key to play again...'

winTxt = 'WINNER!, press any key to play again...'

hangmanPics = [pygame.image.load('hangman0.png'), pygame.image.load('hangman1.png'),
pygame.image.load('hangman2.png'), pygame.image.load('hangman3.png'),
pygame.image.load('hangman4.png'), pygame.image.load('hangman5.png'),
pygame.image.load('hangman6.png')]


limbs = 0


## The function is used to draw a backgrond of the game and all the buttons which representing a letter
inside the game

def redraw():

    global guessed

    global hangmanPics

    global limbs

    win.fill(GREEN)

    # Buttons. The for loop will is used to draw all buttons and label them

    for i in range(len(buttons)):

        if buttons[i][4]:

            pygame.draw.circle(win, BLACK, (buttons[i][1], buttons[i][2]), buttons[i][3])

            pygame.draw.circle(win, buttons[i][0], (buttons[i][1], buttons[i][2]), buttons[i][3] - 2

                               )

            label = btn_font.render(chr(buttons[i][5]), 1, BLACK)

            win.blit(label, (buttons[i][1] - (label.get_width() / 2), buttons[i][2] - (label.get_height() / 2)))
```

```
   spaced = spacedOut(word, guessed)## this variable is stored a word after comparing a guess letter
with a word from a list
   label1 = guess_font.render(spaced, 1, BLACK)## label letters
   rect = label1.get_rect()
   long = rect[2]

   win.blit(label1,(winWidth/2 - long/2, 400))

   picture = hangmanPics[limbs]## storing picture of hanging a man
   win.blit(picture, (winWidth/2 - picture.get_width()/2 + 20, 150))
   pygame.display.update()

##This function will generate a word from a file words.txt and it will return that word
def randomWord():
   file = open('words.txt')
   f = file.readlines()
   i = random.randrange(0, len(f) - 1)
   word = f[i][:-1]

   return word

##This function will check if any guess word imported from a player will be in lower case or not
def hang(guess):
   global word
   if guess.lower() not in word.lower():
      return True
   else:
      return False
```

```
## This function will compare a generated word with any guess letter from a player. If a player guess
correctly a letter it will be replaced and if not, it will remain "_"

def spacedOut(word, guessed=[]):
    spacedWord = ''
    guessedLetters = guessed
    for x in range(len(word)):
        if word[x] != ' ':
            spacedWord += '_ '
            for i in range(len(guessedLetters)):
                if word[x].upper() == guessedLetters[i]:
                    spacedWord = spacedWord[:-2]
                    spacedWord += word[x].upper() + ' '
        elif word[x] == ' ':
            spacedWord += ' '
    return spacedWord


##This function will check if buttons are pressed or not. If a button is pressed the function will return
that button, otherwise it will return non.

def button_pressed(x, y):
    for i in range(len(buttons)):
        if x < buttons[i][1] + 20 and x > buttons[i][1] - 20:
            if y < buttons[i][2] + 20 and y > buttons[i][2] - 20:
                return buttons[i][5]
    return None


##This function will end the game.

def end(winner=False):
    global limbs
    lostTxt = 'You Lost, press any key to play again...'
    winTxt = 'WINNER!, press any key to play again...'
    redraw()
    pygame.time.delay(1000)
    win.fill(GREEN)
## If a winner guess all correct word, a winning word word will be displayed, otherwise, the losing text
will be displayed
    if winner == True:
        label = lost_font.render(winTxt, 1, BLACK)
    else:
        label = lost_font.render(lostTxt, 1, BLACK)


    wordTxt = lost_font.render(word.upper(), 1, BLACK)
    wordWas = lost_font.render('The phrase was: ', 1, BLACK)


    win.blit(wordTxt, (winWidth/2 - wordTxt.get_width()/2, 295))
    win.blit(wordWas, (winWidth/2 - wordWas.get_width()/2, 245))
    win.blit(label, (winWidth / 2 - label.get_width() / 2, 140))
    pygame.display.update()
    ## The function below will keep the game playing if a player wants to play more. Either winning or
losing, a player can continue to play
    again = True
    while again:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
            if event.type == pygame.KEYDOWN:
                again = False
    reset()
```

```
## This function will reset the game if the player wants to replay the game
def reset():
    global limbs
    global guessed
    global buttons
    global word
    for i in range(len(buttons)):
        buttons[i][4] = True

    limbs = 0
    guessed = []
    word = randomWord()



# Setup buttons
increase = round(winWidth / 13)
for i in range(26):
    if i < 13:
        y = 40
        x = 25 + (increase * i)
    else:
        x = 25 + (increase * (i - 13))
        y = 85
    buttons.append([LIGHT_BLUE, x, y, 20, True, 65 + i])
    # buttons.append([color, x_pos, y_pos, radius, visible, char])

word = randomWord()
##The main function to play the game
```

```
def play():
    global limbs
    inPlay = True
    while inPlay:
        ##function redraw is call
        redraw()
        ## setting the time delay of the game
        pygame.time.delay(10)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                inPlay = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    inPlay = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                ##Geting the mouse possition and compared it and stored in a letter variable
                clickPos = pygame.mouse.get_pos()
                letter = button_pressed(clickPos[0], clickPos[1])
                ## if the letter is not none, the guess will append the letter
                if letter != None:
                    guessed.append(chr(letter))
                    buttons[letter - 65][4] = False
                    ## if the hang function is called, the limbs variable continues to increase until it reach 5
otherwise the program will end if the limbs variable = 5
                    if hang(chr(letter)):
                        if limbs != 5:
                            limbs += 1
                        else:
```

```
                end()
        else:
            ##print a comparation ofa generated word with a guessed
            print(spacedOut(word, guessed))
            ## if the count of  "_"  is zero, the game will end
            if spacedOut(word, guessed).count('_') == 0:
                end(True)


play()
#quit a game
pygame.quit()
```