Project	Hangman Game
Student Number	329315
Student Name	Serkan Akbulut
Assessment Title	Assessment 1: Software Unit Testing Report
GitHub Project	https://github.com/openG4NTZ/PRT582
Unit Number and Title	PRT581 Principles of Software Systems
Lecturer/Tutor	Dr. Charles Yeo

TABLE OF CONTENTS

TABLE LIST

FIGURE LIST

1. INTRODUCTION

1.1. PROJECT OVERVIEW

This project is focused on developing Hangman game with using Test Driven Development testing methodology. Hangman is a game where it generates a random word and users tries to guess a letter from this word ones in a time. There are limited number of failed entries in the game and if user cannot guess in the given number of chances the game will be over.

1.1.1. TEST DRIVEN DEVELOPMENT

Test-driven development is a testing method where developer first develops the test cases and then, starts developing the code. For instance, as it is demonstrated below figure developer develops a test and when developer tries the test it fails because there is no code developed yet. So, developer develops the code until it passes the test. After it passes, developer develops new test cases until application is completed ().

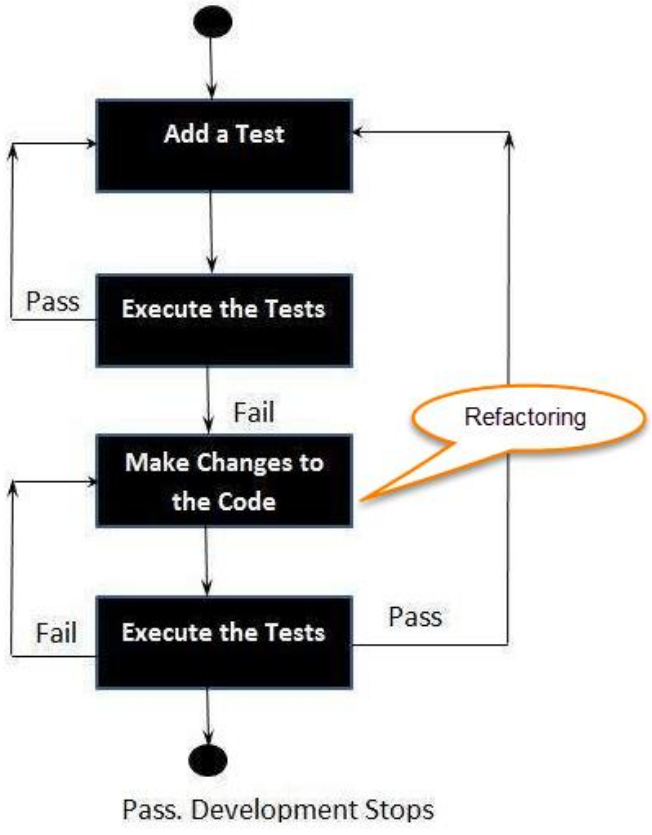


Figure 1 : Test Driven Development (Unadkat J 2020)

2. PROJECT MANAGEMENT PLAN

2.1. HIGH LEVEL REQUIREMENTS

High level project requirements of the game are:

- One word will be generated randomly,
- Player will be presented with a number of blank spaces representing the missing letters the player needs to find.
- If the player's chosen letter exists in the answer, then all places in the answer where that letter appear will be revealed.
- Every time the player guesses a letter wrong, the player's life will be deducted.
- The player must find the missing word before the player's life becomes zero.

2.2. WORK BREAK DOWN STRECTURE

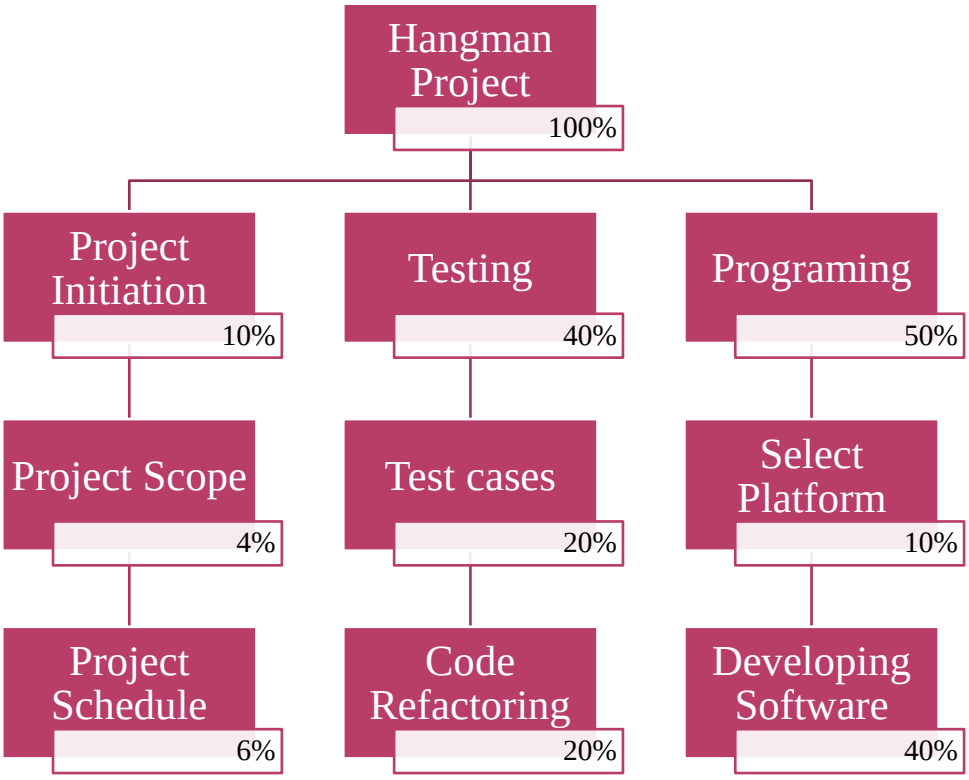


Figure 2 : Work Breakdown Structure

2.3. DELIVERABLES AND MILESTONES

Main deliverable of the project is Hangman Game

Milestones	**Description**	**Estimated Finishing time (D/M/Y)**	**Actual Time (D/M/Y)**
Project Initiation	The project scope, objectives and stakeholders are identified. And the group contract will be sign by all members.	01/08/2020	02/08/2020
Project Scope	Requirements and deliverables are finalized.	05/08/2020	07/08/2020
Selecting Platform and installing to computer	Prototype is developed by using Python in ATOM IDE.	10/08/2020	10/08/2020
Developing Test Cases	Test cases developed.	15/08/2020	17/08/2020
Code Refactoring	Software coding is done using Python programming language in ATOM.	20/08/2020	24/04/2020
Testing functional requirements	Functional testing will be use for the software following black box methodology. Penetration and testing injection testing for security.	25/08/2020	25/08/2020
Product Launch	Completing product and finalizing project report.	28/08/2020	28/08/2020

Table 1 : Deliverables and Milestones

3. SPECIFIC REQUIREMENTS

3.1. FUNCTIONAL REQUIREMENTS

ID: FR1

TITLE: Random word generator

DESC: System should be able to generate random meaningful words

ID: FR2

TITLE: Random Word Should be hidden to user

DESC: Random word will be hidden and only if user guesses correctly, corresponding letter will be shown

ID: FR3

TITLE: Game difficulty

DESC: Game should have 3 difficulty levels easy (8 misses), normal (6 misses) and hard (3 misses).

ID: FR4

TITLE: Wrong Guesses

DESC: If user guesses wrong letter the user life has to be reduced.

ID: FR5

TITLE: Wrong Entries

DESC: If user presses other than single letter such as ‘ab’, ‘!’,’1’ an error code has to be printed.

ID: FR6

TITLE: Game win condition

DESC: If user guesses every letter correctly user wins the game and application returns main screen.

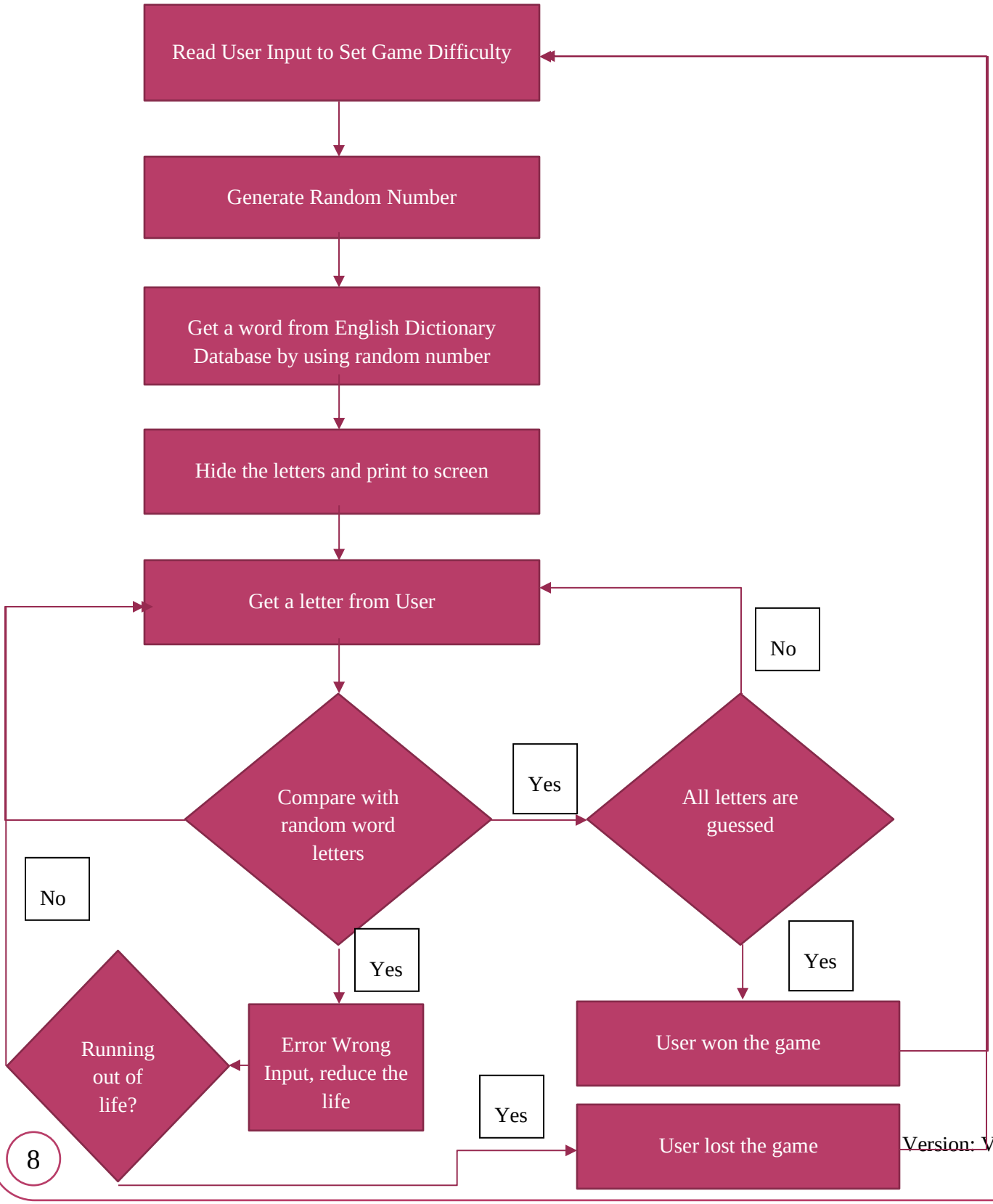
ID: FR7

TITLE: Game over condition

DESC: If user runs out of allowed guesses game will be over. In the end of the game hidden word will be shown and application returns main screen.

4. TEST DRIVEN DEVELOPMENT

4.1. SOFTWARE DIAGRAM BASED ON FUNCTIONAL REQUIREMENTS



Design Inputs

Design Inputs are game level of difficulty and user guesses.

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter:
```

Figure 3 : Example User Inputs

Selected Developing Platform:

Software engineer Serkan Akbulut had Python Programming Language experience therefore this language is used with ATOM interface on Windows 10.

Software requirements:

An operating system that allows command-line or terminal to run Python script

4.2. TEST PLAN

4.2.1. FEATURES TO BE TESTED

The Testing Traceability Matrix below, describes the features to be tested corresponding to the following categories:

A. Random Word Generator
B. Hidden Random Word
C. Game Difficulty
D. Wrong Guesses
E. Wrong Entries
F. Game Over
G. Game Win

Testing Traceability Matrix

Traceability ID	Category	Requirement ID	Requirement Description	Test Case ID	Status
A	Random Word Generator	FR – 001	Program must be able to generate random words	TC001, TC002	Pass/Fail
B	Hidden Random Word	FR – 002	Program must hide the random words from user	TC003	Pass/Fail
C	Game Difficulty	FR – 003	User should be able to choose 3 different game difficulties.	TC004	Pass/Fail
D	Wrong Guesses	FR – 004	If user guesses wrong letter the user life must be reduced.	TC009	Pass/Fail
E	Wrong Entries	FR – 005	If user presses other than single letter such as ‘ab’, ‘!’,’1’ an error code should be printed.	TC005, TC006	Pass/Fail
F	Game Win	FR – 006	If user guesses every letter correctly user wins the game and application must return to main screen.	TC008	Pass/Fail
G	Game Over	FR – 007	If user runs out of allowed guesses game must be over. In the end of the game hidden word should be shown and application must return to main screen.	TC009	Pass/Fail

Table 2 : Testing Traceability Matrix

4.2.2. TEST CASES

Test Case ID	Purpose	Test Function	Inputs	Test Procedure	Expected Outputs
TC001	Data is readable from English Word Database class	test_English_Word_Database()	7825	1.Select 111th element from database 2 Compare with expected output	"english"
TC002	Tests if random word is in English Word Database	test_random_Word_Generator()	Generated Random Number	1.Use random number to select an element from database 2. Compare with expected output	"True"
TC003	Tests if function can split a word	test_split_Word()	"serkan"	1.Send "serkan" to split_Word function 2. Compare with expected output	's','e','r','k','a','n'
TC004	Tests lifecalculator (difficulty) function	test_lifecalculator()	Game mod '1' easy, Game mod '2 normal, Game mod '3' diffucult	1.Send "1","2","3", to split_Word function 2. Compare with expected output	Number of lives "8","6","3 ",
TC005	Tests error message "Wrong_User_Input"	test_print_Error1()	"Wrong_User_Input"	1. Compare with expected output	"Wrong keyboard entry... "
TC006	Tests error message "SameLetter"	test_print_Error2()	"SameLetter"	1. Compare with expected output	"This letter has already been found..."
TC007	Tests User interface	test_user_GUI()	'1', '2', '3', '4'	1. Compare with expected output	'1', '2', '3', '4'
TC008	Test Game Win Scenerio	test_start_Game_Scenerio1 ()	's', 'e', 'r', 'k', 'a', 'n' Number of lives 8	1. Compare with expected output	"serkan"
TC009	Test Game Lost Scenerio	test_start_Game_Scenerio2 ()	'b', 'c', 'd' Number of lives 3	1. Compare with expected output	"serkan"

Table 3 : Test Cases

4.2.3. TEST CASES FOR UNIT TESTING

4.2.3.1. TEST CASE ID: TC001

```python
import unittest
import io
from io import StringIO
from unittest.mock import patch
from hangman import Hangman
from EnglishWords import English_Words
#Boundary conditions or Edge Cases
class HangmanTestCases(unittest.TestCase):
    #Test Case 1: Check English Database
    def test_English_Word_Database(self):
        # Capture the results from the function
        result=English_Words.get_English_Word(7825)
        # Check for expected output
        expected="english"
        self.assertEqual(expected,result,msg="Test Case 1 Failed")

# Run the test
if __name__ == '__main__':
    unittest.main()
```

FAILURE 1

It is failed because get_English_Word is not defined yet. Therefore we have to develop the code in order to pass the test.

```
======================================================================
ERROR: test_English_Word_Database (__main__.HangmanTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\test.py", line 12, in test_English_Word_Database
    result=English_Words.get_English_Word(7825)
AttributeError: type object 'English_Words' has no attribute 'get_English_Word'

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)

Process returned 1 (0x1)        execution time : 0.233 s
Press any key to continue . . .
```

DEVELOPING THE CODE

```
class English_Words:
    set=['empty']
    def get_English_Word(element):
        English_Words.set =[
        'aardvark',
        'Aarhus',
        'Aaron',
```

FAILURE 2

Database returns upper case words but test compares with

```
======================================================================
FAIL: test_English_Word_Database (__main__.HangmanTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\test.py", line 15, in test_English_Word_Database
    self.assertEqual(expected,result,msg="Test Case 1 Failed")
AssertionError: 'english' != 'English'
- english
? ^
+ English
? ^
 : Test Case 1 Failed

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)

Process returned 1 (0x1)        execution time : 0.239 s
Press any key to continue . . .
```

AFTER REFACTORING

Now get_English_Word() always returns lower case words.

```
25483           'zounds',
25484           'zs',
25485           'zucchini',
25486           'Zulu',
25487           'Zurich',
25488           'zygote']
25489           return English_Words.set[element].lower()
```

TC001 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK

Process returned 0 (0x0)        execution time : 0.245 s
Press any key to continue . . .
```

4.2.3.2. TEST CASE ID: TC002

```
        #Test Case 2: Check Random Word Generetor
    def test_random_Word_Generator(self):
        RandomWord=Hangman.random_Word_Generator(self)
        self.assertTrue((RandomWord in list(English_Words.set)),msg="Test Case 2 Failed")
```

DEVELOPING THE CODE

```
import random
import string
import sys
from EnglishWords import English_Words
class Hangman:
    # Generates random number between the numbers 0 and 25480
    #English_Words class has 254800 words and it captures one word
    # Where the index number is equel to random number
    def random_Word_Generator(self):
        random=random.randint(0,25480)
        return English_Words.get_English_Word(random)
```

FAILURE 1

It is failed because python considers random as a local variable and it tries to read before it is assigned. Therefore, random variable has to be defined global.

```
======================================================================
ERROR: test_random_Word_Generator (__main__.HangmanTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\test.py", line 18, in test_random_Word_Generator
    RandomWord=Hangman.random_Word_Generator(self)
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\hangman.py", line 11, in random_Word_Generator
    random=random.randint(0,25480)
UnboundLocalError: local variable 'random' referenced before assignment

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

FAILED (errors=1)

Process returned 1 (0x1)        execution time : 0.190 s
Press any key to continue . . .
```

AFTER REFACTORING

```python
import random
import string
import sys
from EnglishWords import English_Words
class Hangman:
    # Generates random number between the numbers 0 and 25480
    #English_Words class has 254800 words and it captures one word
    # Where the index number is equel to random number
    def random_Word_Generator(self):
        global random
        random=random.randint(0,25480)
        return English_Words.get_English_Word(random)
```

TC001 AND TC002 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK

Process returned 0 (0x0)        execution time : 0.182 s
Press any key to continue . . .
```

4.2.3.3. TEST CASE ID: TC003

```
    #Test Case 3: Check function for splitting words
    def test_split_Word(self):
        result=Hangman.split_Word(self,"serkan")
        expected=['s','e','r','k','a','n']
        self.assertEqual(expected,result,msg="Test Case 3 Failed")
# Run the test
if __name__ == '__main__':
    unittest.main()
```

DEVELOPING THE CODE

```
    #Splits Strings to list
    def split_Word(self,Word):
        return list(Word)
```

TC001, TC002 AND TC003 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK

Process returned 0 (0x0)        execution time : 0.189 s
Press any key to continue . . .
```

4.2.3.4. TEST CASE ID: TC004

```
#Test Case 4: Check game dificulty and coreresponding amount of life
def test_lifecalculator(self):
    # Capture the results from the function
    if Hangman.lifecalculator(self,'1')==8 and Hangman.lifecalculator(self,'2')==6 and Hangman.lifecalculator(self,'3')==3:
        result=True
    else:
        result=False
    # Check for expected output
    expected=True
    self.assertEqual(expected,result,msg="Test Case 4 Failed")
# Run the test
if __name__ == '__main__':
    unittest.main()
```

DEVELOPING THE CODE

```
#Calculates number of lifes depends on lvl of difficulty
def lifecalculator(self,Difficultylevel):
    if Difficultylevel=='1':
        return 8
    elif Difficultylevel=='2':
        return 6
    elif Difficultylevel=='3':
        return 3
    elif Difficultylevel=='4':
        # if user chooses number 4 game closes
        self.end_Game()
    else:
        # if user enters anyother number it go calls back the same function recursively
        return self.lifecalculator(self.user_GUI("Intro",0))
```

TC001, TC002, TC003 AND TC004 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK

Process returned 0 (0x0)        execution time : 0.176 s
Press any key to continue . . .
```

4.2.3.5. TEST CASE ID: TC005

```
    #Test Case 5: Test possible error messages
    def test_print_Error1(self):
        # Capture the results from the function
        result=Hangman.print_Error(self,"Wrong_User_Input")
        # Check for expected output
        expected="Wrong keyboard entry... "
        self.assertEqual(expected,result,msg="Test Case 5 Failed")
# Run the test
```

DEVELOPING THE CODE

```
    def print_Error(self,ErrorCode):
        if ErrorCode = "Wrong_User_Input":
            # Wrong keyboard entry such as double letter, number or special char
            return "Wrong keyboard entry... "
```

FAILURE 1

There is syntax error in the code

```
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\StartGame.py", line 1, in <module>
    from hangman import Hangman
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\hangman.py", line 33
    if ErrorCode = "Wrong_User_Input":
                 ^
SyntaxError: invalid syntax
```

AFTER REFACTORING

```
# It returns error outputs depending on error codes
def print_Error(self,ErrorCode):
    if ErrorCode == "Wrong_User_Input":
        # Wrong keyboard entry such as double letter, number or special char
        return "Wrong keyboard entry... "
```

TC001, TC002, TC003, TC004 AND TC005 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

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

OK

Process returned 0 (0x0)        execution time : 0.181 s
Press any key to continue . . .
```

4.2.3.6. TEST CASE ID: TC006

```
    #Test Case 6: Test possible error messages
    def test_print_Error2(self):
        # Capture the results from the function
        result=Hangman.print_Error(self,"SameLetter")
        # Check for expected output
        expected="This letter has already been found..."
        self.assertEqual(expected,result,msg="Test Case 6 Failed")
        # get_input will return 'yes' during this test
# Run the test
if __name__ == '__main__':
    unittest.main()
```

DEVELOPING THE CODE

```
    # It returns error outputs depending on error codes
    def print_Error(self,ErrorCode):
        if ErrorCode == "Wrong_User_Input":
            # Wrong keyboard entry such as double letter, number or special char
            return "Wrong keyboard entry... "
        elif ErrorCode == "SameLetter":
            # If user enters the visiable letters again this msgg shows
            return "This letter has already ben found..."
    # This function returns user inputs depending on game status
```

FAILURE 1

There is syntax error in the code.

```
...F..
======================================================================
FAIL: test_print_Error2 (__main__.HangmanTestCases)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\test.py", line 48, in test_print_Error2
    self.assertEqual(expected,result,msg="Test Case 6 Failed")
AssertionError: 'This letter has already been found...' != 'This letter has already ben found...'
- This letter has already been found...
?                            -
+ This letter has already ben found...
 : Test Case 6 Failed

----------------------------------------------------------------------
Ran 6 tests in 0.003s

FAILED (failures=1)

Process returned 1 (0x1)        execution time : 0.205 s
Press any key to continue . . .
```

AFTER REFACTORING

```
    # It returns error outputs depending on error codes
    def print_Error(self,ErrorCode):
        if ErrorCode == "Wrong_User_Input":
            # Wrong keyboard entry such as double letter, number or special char
            return "Wrong keyboard entry... "
        elif ErrorCode == "SameLetter":
            # If user enters the visiable letters again this msgg shows
            return "This letter has already been found..."
```

TC001, TC002, TC003, TC004, TC005 AND TC006 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.002s

OK

Process returned 0 (0x0)        execution time : 0.189 s
Press any key to continue . . .
```

4.2.3.7. TEST CASE ID: TC007

```
    #Test Case 7: Test User Interface
    @patch('builtins.input', side_effect=['1', '2', '3', '4'])
    #this function simulates user inputs,
    def test_user_GUI(self, mock_input):
        #function user_GUI called 4 times with '1', '2', '3', '4' inputs
        calling_1 = Hangman.user_GUI(self,"Intro",0)
        calling_2 = Hangman.user_GUI(self,"Intro",0)
        calling_3 = Hangman.user_GUI(self,"Intro",0)
        calling_4 = Hangman.user_GUI(self,"Intro",0)
        # Check for expected output
        self.assertTrue(calling_1 == '1' and calling_2 == '2' and
                        calling_3 == '3' and calling_4 == '4',msg="Test Case 7 Failed")
```

DEVELOPING THE CODE

```
# This function returns user inputs depending on game status
def user_GUI(self,GameStatus,NumberofLife):
    global ScreenGap
    ScreenGap="""                    """
    # beginning of the game
    if GameStatus == "Intro":
       return input("""
                Welcome to Hangman Game

                Please choose a difficulty level to start the game:
                1. Easy (8 Misses)
                2. Medium (6 Misses)
                3. Advanced (3 Misses)
                4. Exit Game
                """)
    elif GameStatus == "Starting":
    # first User entry
        return input("\n" + ScreenGap+"Hangman is starting, number of lives: "+str(NumberofLife)+"\n"
    elif GameStatus == "Started":
    # second and later entries
       if NumberofLife > 0:
           return input (ScreenGap+"Please enter a single letter: ").lower()
       else:
           # out of life
           print(ScreenGap+"GAME OVER...")
           return False
```

TC001, TC002, TC003, TC004, TC005, TC006 AND TC007 PASSES

After code passed the test case, next test case will be created and corresponding code will be developed.

```
.......
----------------------------------------------------------------------
Ran 7 tests in 0.003s

OK

Process returned 0 (0x0)        execution time : 0.196 s
Press any key to continue . . .
```

4.2.3.8. TEST CASE ID: TC008

In this test, ‘s’,’e’,’r’,’k’,’a’,’n’ inputs are used as user inputs to check that game can be win if customer enters correct letters.

```
    #Test Case 8: Test correct guess
    @patch('builtins.input', side_effect=['s', 'e', 'r', 'k', 'a', 'n'])
    def test_start_Game1(self,mock_input):
        # Capture the results from the function
        test_Game =Hangman()
        result=test_Game.start_Game("serkan",8)
        # Check for expected output
        expected=True
        self.assertEqual(expected,result,msg="Test Case 8 Failed")
        # get_input will return 'yes' during this test#Test Case 8: Test actual game
# Run the test
if __name__ == '__main__':
    unittest.main()
```

```
# Starts the game and does necesary calculations
def start_Game(self,random_Word,NumberofLife):
    #Splits random word to a list
    splitted_random_Word=self.split_Word(random_Word)
    Random_Word_Hidden=self.split_Word(random_Word)
    # Random_Word_Hidden holds users correct guesses
    # it is also used for understanding user progress
    for x in range(len(Random_Word_Hidden)):
        Random_Word_Hidden[x]="_"
    user_input=self.user_GUI("Starting",NumberofLife)
    # While loop used for to get multiple entries from user
    # it is also dynamicly handles different type of entries
    while NumberofLife > 0:
        #Checks if user input is not a single letter
        if user_input not in list(string.ascii_letters):
            print(ScreenGap+self.print_Error("Wrong_User_Input"))
        # compares user input with actual word
        elif user_input not in list(splitted_random_Word):
            NumberofLife-=1
            print(ScreenGap+"Wrong Entry")
        # if it is a correct guess there us two possibility
        else:
            for x in range(len(splitted_random_Word)):
                # 1. new cirrect guess
                if splitted_random_Word[x]==user_input and  Random_Word_Hidden[x]!=user_input:
                    Random_Word_Hidden[x]=user_input
                # 2. old correct guess
                elif Random_Word_Hidden[x]==user_input:
                    print(ScreenGap+self.print_Error("SameLetter"))
                    break
        # puts all items together and shows to user his progress
        print(ScreenGap+' '.join(Random_Word_Hidden))
        # shows reamaing lives to user
        print(ScreenGap+"Remaing lives: ",NumberofLife)
        # if there is no empty space in Random_Word_Hidden that means user guess everything
        if "_" not in list(Random_Word_Hidden):
            print(ScreenGap+"Congratulation... YOU WON....")
            print(ScreenGap+"The hidden word :" + random_Word)
            return True
        # End of user input control, new input requested
        user_input=self.user_GUI("Started",NumberofLife)
        if user_input == False:
            print(ScreenGap+"The hidden word :" + random_Word)
            return False
```

TC001, TC002, TC003, TC004, TC005, TC006, TC007 AND TC008 PASSES

After code passed the test case, next test case will be created, and corresponding code will be developed.

```
......                      s _ _ _ _ _
                    Remaing lives:  8
                    s e _ _ _ _
                    Remaing lives:  8
                    s e r _ _ _
                    Remaing lives:  8
                    s e r k _ _
                    Remaing lives:  8
                    s e r k a _
                    Remaing lives:  8
                    s e r k a n
                    Remaing lives:  8
                    Congratulation... YOU WON....
                    The hidden word :serkan
..
----------------------------------------------------------------------
Ran 8 tests in 0.006s

OK

Process returned 0 (0x0)        execution time : 0.189 s
Press any key to continue . . .
```

4.2.3.9. TEST CASE ID: TC009

In this test, ‘b’,’c’,’d’ inputs are used as user inputs to check that game can be lost if customer enters wrong letters.

```
    #Test Case 9: Test wrong guess
    @patch('builtins.input', side_effect=['b', 'c', 'd'])
    def test_start_Game2(self,mock_input):
        # Capture the results from the function
        test_Game =Hangman()
        result=test_Game.start_Game("serkan",3)
        # Check for expected output
        expected=False
        self.assertEqual(expected,result,msg="Test Case 9 Failed")
        # get_input will return 'yes' during this test
# Run the test
if __name__ == '__main__':
    unittest.main()
```

```
# Starts the game and does necesary calculations
def start_Game(self,random_Word,NumberofLife):
    #Splits random word to a list
    splitted_random_Word=self.split_Word(random_Word)
    Random_Word_Hidden=self.split_Word(random_Word)
    # Random_Word_Hidden holds users correct guesses
    # it is also used for understanding user progress
    for x in range(len(Random_Word_Hidden)):
        Random_Word_Hidden[x]="_"
    user_input=self.user_GUI("Starting",NumberofLife)
    # While loop used for to get multiple entries from user
    # it is also dynamicly handles different type of entries
    while NumberofLife > 0:
        #Checks if user input is not a single letter
        if user_input not in list(string.ascii_letters):
            print(ScreenGap+self.print_Error("Wrong_User_Input"))
        # compares user input with actual word
        elif user_input not in list(splitted_random_Word):
            NumberofLife-=1
            print(ScreenGap+"Wrong Entry")
        # if it is a correct guess there us two possibility
        else:
            for x in range(len(splitted_random_Word)):
                # 1. new cirrect guess
                if splitted_random_Word[x]==user_input and  Random_Word_Hidden[x]!=user_input:
                    Random_Word_Hidden[x]=user_input
                # 2. old correct guess
                elif Random_Word_Hidden[x]==user_input:
                    print(ScreenGap+self.print_Error("SameLetter"))
                    break
        # puts all items together and shows to user his progress
        print(ScreenGap+' '.join(Random_Word_Hidden))
        # shows reamaing lives to user
        print(ScreenGap+"Remaing lives: ",NumberofLife)
        # if there is no empty space in Random_Word_Hidden that means user guess everything
        if "_" not in list(Random_Word_Hidden):
            print(ScreenGap+"Congratulation... YOU WON....")
            print(ScreenGap+"The hidden word :" + random_Word)
            return True
        # End of user input control, new input requested
        user_input=self.user_GUI("Started",NumberofLife)
        if user_input == False:
            print(ScreenGap+"The hidden word :" + random_Word)
            return False
```

TC001, TC002, TC003, TC004, TC005, TC006, TC007, TC008 AND TC009 PASSES

All tests are passed.

```
......                    s _ _ _ _ _
                     Remaing lives:  8
                     s e _ _ _ _
                     Remaing lives:  8
                     s e r _ _ _
                     Remaing lives:  8
                     s e r k _ _
                     Remaing lives:  8
                     s e r k a _
                     Remaing lives:  8
                     s e r k a n
                     Remaing lives:  8
                     Congratulation... YOU WON....
                     The hidden word :serkan
.                     Wrong Entry

                     _ _ _ _ _ _
                     Remaing lives:  2
                     Wrong Entry

                     _ _ _ _ _ _
                     Remaing lives:  1
                     Wrong Entry

                     _ _ _ _ _ _
                     Remaing lives:  0
                     GAME OVER...
                     The hidden word :serkan
..
----------------------------------------------------------------------
Ran 9 tests in 0.006s

OK

Process returned 0 (0x0)        execution time : 0.184 s
Press any key to continue . . .
```

4.2.4. OPEN BOX TESTING (WHITE BOX TESTING)

In this part, game will be testing from user perspective. 5 different scenarios are created to test software behavior.

- Game Win
- Game Lost and Wrong Guess
- Wrong Keyboard Entries
- Same Correct Entries
- Random Word Generator

4.2.4.1. GAME WIN SCENERIO

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
3

Hangman is starting, number of lives: 3
Please enter a single letter: e
e _ _ _ _ _ _
Remaing lives:  3
Please enter a single letter: n
e n _ _ _ _ _
Remaing lives:  3
Please enter a single letter: g
e n g _ _ _ _
Remaing lives:  3
Please enter a single letter: l
e n g l _ _ _
Remaing lives:  3
Please enter a single letter: i
e n g l i _ _
Remaing lives:  3
Please enter a single letter: s
e n g l i s _
Remaing lives:  3
Please enter a single letter: h
e n g l i s h
Remaing lives:  3
Congratulation... YOU WON....
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

First Game difficulty is chosen difficult, then it is chosen medium and finally it is chosen easy. After user won the game it goes to game replay. For all cases game did not give any error. Functional requirement FR6 completed.

C	Game Difficulty	FR – 003	User should be able to choose 3 different game difficulties.	TC004	Pass
F	Game Win	FR – 006	If user guesses every letter correctly user wins the game and application must return to main screen.	TC008	Pass

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
2

Hangman is starting, number of lives: 6
Please enter a single letter: e
e _ _ _ _ _
Remaing lives:  6
Please enter a single letter: g
e _ g _ _ _
Remaing lives:  6
Please enter a single letter: l
e _ g l _ _
Remaing lives:  6
Please enter a single letter: n
e n g l _ _
Remaing lives:  6
Please enter a single letter: i
e n g l i _
Remaing lives:  6
Please enter a single letter: s
e n g l i s _
Remaing lives:  6
Please enter a single letter: h
e n g l i s h
Remaing lives:  6
Congratulation... YOU WON....
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: e
e _ _ _ _ _
Remaing lives:  8
Please enter a single letter: n
e n _ _ _ _
Remaing lives:  8
Please enter a single letter: g
e n g _ _ _
Remaing lives:  8
Please enter a single letter: l
e n g l _ _
Remaing lives:  8
Please enter a single letter: i
e n g l i _
Remaing lives:  8
Please enter a single letter: s
e n g l i s _
Remaing lives:  8
Please enter a single letter: h
e n g l i s h
Remaing lives:  8
Congratulation... YOU WON....
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

4.2.4.2. GAME LOSE AND WRONG GUESS SCENERIO

In this test, user loses the game in 3 different difficulty levels. This scenario checks below requirements.

C	Game Difficulty	FR – 003	User should be able to choose 3 different game difficulties.	TC004	Pass
D	Wrong Guesses	FR – 004	If user guesses wrong letter the user life must be reduced.	TC005, TC009	Pass
G	Game Over	FR – 007	If user runs out of allowed guesses game must be over. In the end of the game hidden word should be shown and application must return to main screen.	TC009	Pass

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
3

Hangman is starting, number of lives: 3
Please enter a single letter: z
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  2
Please enter a single letter: x
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  1
Please enter a single letter: m
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  0
GAME OVER...
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
2

Hangman is starting, number of lives: 6
Please enter a single letter: z
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  5
Please enter a single letter: w
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  4
Please enter a single letter: x
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  3
Please enter a single letter: y
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  2
Please enter a single letter: r
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  1
Please enter a single letter: f
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  0
GAME OVER...
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: z
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  7
Please enter a single letter: y
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  6
Please enter a single letter: x
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  5
Please enter a single letter: w
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  4
Please enter a single letter: r
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  3
Please enter a single letter: t
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  2
Please enter a single letter: p
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  1
Please enter a single letter: u
Wrong Entry

_ _ _ _ _ _ _
Remaing lives:  0
GAME OVER...
The hidden word :english

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

4.2.4.3. WRONG KEYBOARD ENTRIES

In this test possible wrong keyboard entries will be tested.

B	Hidden Random Word	FR – 002	Program must hide the random words from user	TC003	Pass
E	Wrong Entries	FR – 005	If user presses other than single letter such as ‘ab’, ‘!’,’1’ an error code should be printed.	TC006	Pass

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1234
Wrong keyboard entry...

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
a
Wrong keyboard entry...

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
$
Wrong keyboard entry...

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
```

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: ab
Wrong keyboard entry...

_ _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: !
Wrong keyboard entry...

_ _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: 1234
Wrong keyboard entry...

_ _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: ff
Wrong keyboard entry...

_ _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter:
```

4.2.4.4. SAME CORRECT ENTRIES

TC006	Tests error message "SameLetter"	Checks same letter entries	"SameLetter"	1. Compare with expected output	Pass

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: e
e _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: e
This letter has already been found...
e _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: e
This letter has already been found...
e _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter: e
This letter has already been found...
e _ _ _ _ _ _
Remaing lives:  8
Please enter a single letter:
```

4.2.4.5. RANDOM WORD GENERATOR

A	Random Word Generator	FR – 001	Program must be able to generate random words	TC001, TC002	Failed
B	Hidden Random Word	FR – 002	Program must hide the random words from user	TC003	Pass

```
                    Welcome to Hangman Game

                    Please choose a difficulty level to start the game:
                    1. Easy (8 Misses)
                    2. Medium (6 Misses)
                    3. Advanced (3 Misses)
                    4. Exit Game
                    3

                    Hangman is starting, number of lives: 3
                    Please enter a single letter: x
                    Wrong Entry
                    _ _ _ _ _ _ _ _
                    Remaing lives:  2
                    Please enter a single letter: x
                    Wrong Entry
                    _ _ _ _ _ _ _ _
                    Remaing lives:  1
                    Please enter a single letter: x
                    Wrong Entry
                    _ _ _ _ _ _ _ _
                    Remaing lives:  0
                    GAME OVER...
                    The hidden word :shepherd

                    Welcome to Hangman Game

                    Please choose a difficulty level to start the game:
                    1. Easy (8 Misses)
                    2. Medium (6 Misses)
                    3. Advanced (3 Misses)
                    4. Exit Game
                    3
Traceback (most recent call last):
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\StartGame.py", line 9, in <module>
    random_Word=newGame.random_Word_Generator()
  File "C:\Users\G4NTZ\Documents\GitHub\PRT582\hangman.py", line 11, in random_Word_Generator
    random=random.randint(0,25480)
AttributeError: 'int' object has no attribute 'randint'

Process returned 1 (0x1)        execution time : 5.620 s
Press any key to continue . . .
```

In white box testing software failed to pass FR01. Python gave **“AttributeError: 'int' object has no attribute 'randint'”** error. It is the part where software generates a random number with **random.randint()** function. It passed the unit testing because this function separately works fine however when it is called from another class second time it gives this error.

CODE REFOCTORING

Before:

Class name and variable name was same and that was causing this problem.

```
class Hangman:
    # Generates random number between the numbers 0 and 25480
    #English_Words class has 254800 words and it captures one word
    # Where the index number is equel to random number
    def random_Word_Generator(self):
        global random
        random=random.randint(0,25480)
        return English_Words.get_English_Word(random)
    #Splits Strings to list
```

After:

```
class Hangman:
    # Generates random number between the numbers 0 and 25480
    #English_Words class has 254800 words and it captures one word
    # Where the index number is equel to random number
    def random_Word_Generator(self):
        global random_Number
        random_Number=random.randint(0,25480)
        return English_Words.get_English_Word(random_Number)
```

TESTIG

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
3

Hangman is starting, number of lives: 3
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _
Remaing lives:  2
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _
Remaing lives:  1
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _
Remaing lives:  0
GAME OVER...
The hidden word :pompon

Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
3

Hangman is starting, number of lives: 3
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _ _
Remaing lives:  2
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _ _
Remaing lives:  1
Please enter a single letter: x
Wrong Entry
_ _ _ _ _ _ _
Remaing lives:  0
GAME OVER...
The hidden word :arcadia
```

4.2.5. BEHAVIOURAL TESTING (BLACK BOX TESTING)

4.2.5.1. RANDOM USER 1 TESTING

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: a
_ _ _ _ a _ _ _ a
Remaing lives:  8
Please enter a single letter: s
_ _ _ s a _ _ _ a
Remaing lives:  8
Please enter a single letter: e
_ e _ s a _ _ _ a
Remaing lives:  8
Please enter a single letter: k
Wrong Entry
_ e _ s a _ _ _ a
Remaing lives:  7
Please enter a single letter: g
Wrong Entry
_ e _ s a _ _ _ a
Remaing lives:  6
Please enter a single letter: t
Wrong Entry
_ e _ s a _ _ _ a
Remaing lives:  5
Please enter a single letter: r
Wrong Entry
_ e _ s a _ _ _ a
Remaing lives:  4
Please enter a single letter: l
_ e _ s a _ _ l a
Remaing lives:  4
Please enter a single letter: m
Wrong Entry
_ e _ s a _ _ l a
Remaing lives:  3
Please enter a single letter: p
p e _ s a _ _ l a
Remaing lives:  3
Please enter a single letter: f
Wrong Entry
p e _ s a _ _ l a
Remaing lives:  2
Please enter a single letter: c
p e _ s a c _ l a
Remaing lives:  2
Please enter a single letter: i
Wrong Entry
p e _ s a c _ l a
Remaing lives:  1
Please enter a single letter: n
p e n s a c _ l a
Remaing lives:  1
Please enter a single letter: o
p e n s a c o l a
Remaing lives:  1
Congratulation... YOU WON....
The hidden word :pensacola
```

4.2.5.2. RANDOM USER 2 TESTING

```
Welcome to Hangman Game

Please choose a difficulty level to start the game:
1. Easy (8 Misses)
2. Medium (6 Misses)
3. Advanced (3 Misses)
4. Exit Game
1

Hangman is starting, number of lives: 8
Please enter a single letter: s
Wrong Entry
_ _ _ _ _ _
Remaing lives:  7
Please enter a single letter: a
Wrong Entry
_ _ _ _ _ _
Remaing lives:  6
Please enter a single letter: e
Wrong Entry
_ _ _ _ _ _
Remaing lives:  5
Please enter a single letter: i
_ _ i _ _ _
Remaing lives:  5
Please enter a single letter: o
Wrong Entry
_ _ i _ _ _
Remaing lives:  4
Please enter a single letter: u
Wrong Entry
_ _ i _ _ _
Remaing lives:  3
Please enter a single letter: f
Wrong Entry
_ _ i _ _ _
Remaing lives:  2
Please enter a single letter: k
Wrong Entry
_ _ i _ _ _
Remaing lives:  1
Please enter a single letter: l
_ l i _ _ _
Remaing lives:  1
Please enter a single letter: k
Wrong Entry
_ l i _ _ _
Remaing lives:  0
GAME OVER...
```

5. CONCLUSION

In this software development, Test Driven Development method used and successfully Hangman Game developed. First, high level functional requirements are created, than test cases are developed to create test functions. After that, unit testing created in python. First, it gave errors because corresponding methods are not defined, so code development and improved until tests are passed. After one test is completed, second test is created, and then corresponding method developed. Until all test are cleared this process repeated. Then white box and black box tests are used the find further problems. One problem detected during the white box testing and final version of the code is released after black box testing is completed.

BIBLIOGRAPHY

1. Unadkat J 2020, *BDD vs TDD vs ATDD : Key Differences*, https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd
2. Technology Conversations 2020, *Test Driven Development (TDD): Example Walkthrough*, https://technologyconversations.com/2013/12/20/test-driven-development-tdd-example-walkthrough/
3. python.org 2020, *random — Generate pseudo-random numbers*, https://docs.python.org/3/library/random.html
4. python.org 2020, *unittest — Unit testing framework*, https://docs.python.org/3/library/unittest.html