**Introduction:** I have built the hangman game.The game is written in Python. This game is built using TDD. Code files are available on “<https://github.com/daljeet0611/Hangman>”. Words are predefined and stored in a string array. The program selects one word and produces blank spaces and lives, based on the letters in word. The user prompt to enter one letter from keyboard. The program takes guessed letter from user and start searching that letter in the selected word. If the letter exists in the word then program filled the blank space. To win the game user has to guess the all letters in word before he/she out of lives.

**Implementation of TDD:** As the TDD is referer to very short development cycle. The whole idea of TDD is to break down the program into specific test cases. Then improved the code to pass the test. I break down the whole game program into parts. Below are the steps that I have followed to develop the game.

1. Gather the requirements and logics need to implement in coding.
2. Decided the parts and placed them into sequence.
3. Gave them names to functions for implementation.
4. Declare the Hangman class and UnitTest class.
5. First I started with random words. So declare the string array and store few words into an array.
6. Then wrote the functions such as \_\_init\_\_, get random word, user lives, search the letters in word guessed by user, Is User Guessed all letters. Test the functions manually (using print()) and also wrote a test case for unit testing. Improve the code until the expected result.
7. After successfully test the functions. Wrote the final function RunGame. This function handle the whole process of game. Means call the other functions as required. User input is also implemented into this function. Test the function and finalize it.
8. Wrote the all comments.
9. Create the class object and call the RunGame function to start the game.
10. Test the game in different scenarios.
11. Checked the whole program again for improvements. Made changes as required for improvements and easy readability.
12. Checked for refactoring if required.

**Functions build:**

1. GetRandomWord(self)
2. GetUserLives(self, randomly\_selected\_word)
3. FillBlankSpaces(self, LettersGuessedByUser, LetterGuessByUser, RandomlySelectedWord, Result)
4. IsAllSpacesFilled(self, Result)
5. RunGame(self)

**Refactoring and code smell:** Below are the points followed for the recfactoring and code smell

1. Removed the unwanted or similar fields and functions to reduce the redundancy.
2. Break down the large program into small functions.
3. Made sure all the unit tests are running.
4. Test all the parts of programs are working as planned.
5. Simplify the code as much as possible.
6. Reduced the complexity of the program made the program simple as possible.
7. Fixed the program structure for better understanding and code-flow.

**Issues and solutions:** Followings are issues and solutions faced during the implementations.

1. Selection of random words. Create a predefined string array to generate the random word.
2. Calculate User's lives. Initially planned 5 lives but later on decide lives should be equal to letters in word. So used count of letters in a word as lives.
3. Blank spaces filling issue. It was initially hard to make a logic to fill the black spaces as it requires to keep the old guessed letter as well. So create new string to store the only guessed letters. Then used this string to fill the blank spaces as user guessed new letter.
4. How to keep game running until user wins the game or loss the game. After spent time on this. Decided to go with while loop with if condition to break it.

**Extra features added:** If user guessed wrong letter. He/She can see the previously guessed wrong letters as well.

**Final and tested Code:**

**import random** # Library to generate a random number

**class Hangman:**

# Array to store the predefined words

**Random\_words = ["SEARCH", "VOLVO", "QUERY", "MATCH", "UPLOAD", "CURSOR", "FOLLOW",**

**"UPDATE", "SCREEN", "REMOVE", "MEMBER", "PLACE", "NAVIGATE"]**

# Fields

**WrongLettersGuessedByUser = ""**

**randomly\_selected\_word = ""**

**LettersGuessedByUser = ""**

**Result = ""**

**Lives = 0**

# \*\*\*\*\*\* Select Random Word \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# This function selects one word randomly \*

# and return \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def GetRandomWord(self):**

# Get random number to choose on word from words collection

**random\_number = random.randint(0, 12)**

# Select the one word from collection of words

**return self.Random\_words[random\_number]**

# \*\*\*\*\*\* Calculate lives \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# This function return lives \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def GetUserLives(self, randomly\_selected\_word):**

#Lives, given to user. equal to letters in words

**return len(randomly\_selected\_word)**

# \*\*\*\*\*\* FillBlankSpaces \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# This function search the location of \*

# letters and place them on correct position \*

# Fill the \_ on the place of not guessed letters. \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def FillBlankSpaces(self, LettersGuessedByUser, LetterGuessByUser, RandomlySelectedWord, Result):**

# Making collection of guessed letters from user, to filled the guessed letters

**LettersGuessedByUser += LetterGuessByUser**

**self.LettersGuessedByUser = LettersGuessedByUser**

# Clear previous result

**Result = ""**

# Selecting one letter from randomly selected word.

**for c in RandomlySelectedWord:**

# Finding selected letter in guessed letters string and filling the correct letters in blank spaces.

**if c in self.LettersGuessedByUser:**

# Filling guessed letter

**Result += c+" "**

**else: # Not guessed letter/s postion**

**Result += "\_ "**

**return Result**

# \*\*\*\*\*\* IsAllSpacesFilled \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# Checking the empty spaces \*

# return true if no empty spaces left \*

# otherwise false \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def IsAllSpacesFilled(self, Result):**

# Checking, are the all letters guessed by user?

**if "\_" not in Result:** # True = All the letters guessed by user and won the game.

**print("\*\*\*\*\*\*\* You Won \*\*\*\*\*\*\*\n\n")**

**return True**

**else:**

**return False** # returned false if

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RunGame \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# Calling the functions in sequence to run \*

# the game in stable state \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def RunGame(self):**

**# While loop, to keep program runing until the end of gamel**

**while True:**

**if self.Lives == 0:** # if user loose all the lives means end of game

**print("\*\*\*\* You Lose \*\*\*\*\n\n")**

**break** # Terminate the while loop, end of game

# Taking input from user and store into string for further use

**LetterGuessByUser = input()**

# Here upper() function is use to make letter, upper case. Reason, all the random words are in

upper case.

**LetterGuessByUser = LetterGuessByUser.upper()**

# Finding letter in randomly selected word. If exist fill the blank space

**if LetterGuessByUser in self.randomly\_selected\_word:**

# Calling FillBlankSpaces and LetterGuessByUser is argument.

**self.Result = self.FillBlankSpaces(self.LettersGuessedByUser, LetterGuessByUser,**

**self.randomly\_selected\_word, self.Result)**

# Print the result

**print("\n"+self.Result+"\n") # Printing result after letter entered by user.**

# Searching, is there any empty space.

**if self.IsAllSpacesFilled(self.Result) == True:**

**break** # Terminate the while loop, end of game. Beacuse there is no empty space.

**else:** # If letter gueesed by user is not exist in randomly selected word.

**self.Lives -= 1 # User loose the one live.**

**print("Live: "+str(self.Lives)+"\n")**

# Storing wrong letter guessed by user and print all the wrong letters in next line.

# User can see wrong words.

**self.WrongLettersGuessedByUser += LetterGuessByUser+" "**

**print("Wrong letter/s guessed by you : "+self.WrongLettersGuessedByUser+"\n")**

# Printing result after letter guessed by user.

**print("\n"+self.Result+"\n")**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \_\_init\_\_ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*

# This function initialize the class object and fields \*

# \*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**def \_\_init\_\_(self):**

**self.WrongLettersGuessedByUser = ""**

**self.randomly\_selected\_word = self.GetRandomWord()**

**print(self.randomly\_selected\_word)**

**self.Lives = self.GetUserLives(self.randomly\_selected\_word)**

**print("\n\n\*\*\*\*\*\*\* Hangman by Diljeet Bains \*\*\*\*\*\*\*\n\n")**

# Print the count of lives given to user

**print("You have "+str(self.Lives)+" lives\n")**

#Preparing black spaces, equal to letters in word

**for \_ in range(0, self.Lives):**

**self.Result += "\_ "**

#print Blank Spaces

**print(self.Result)**

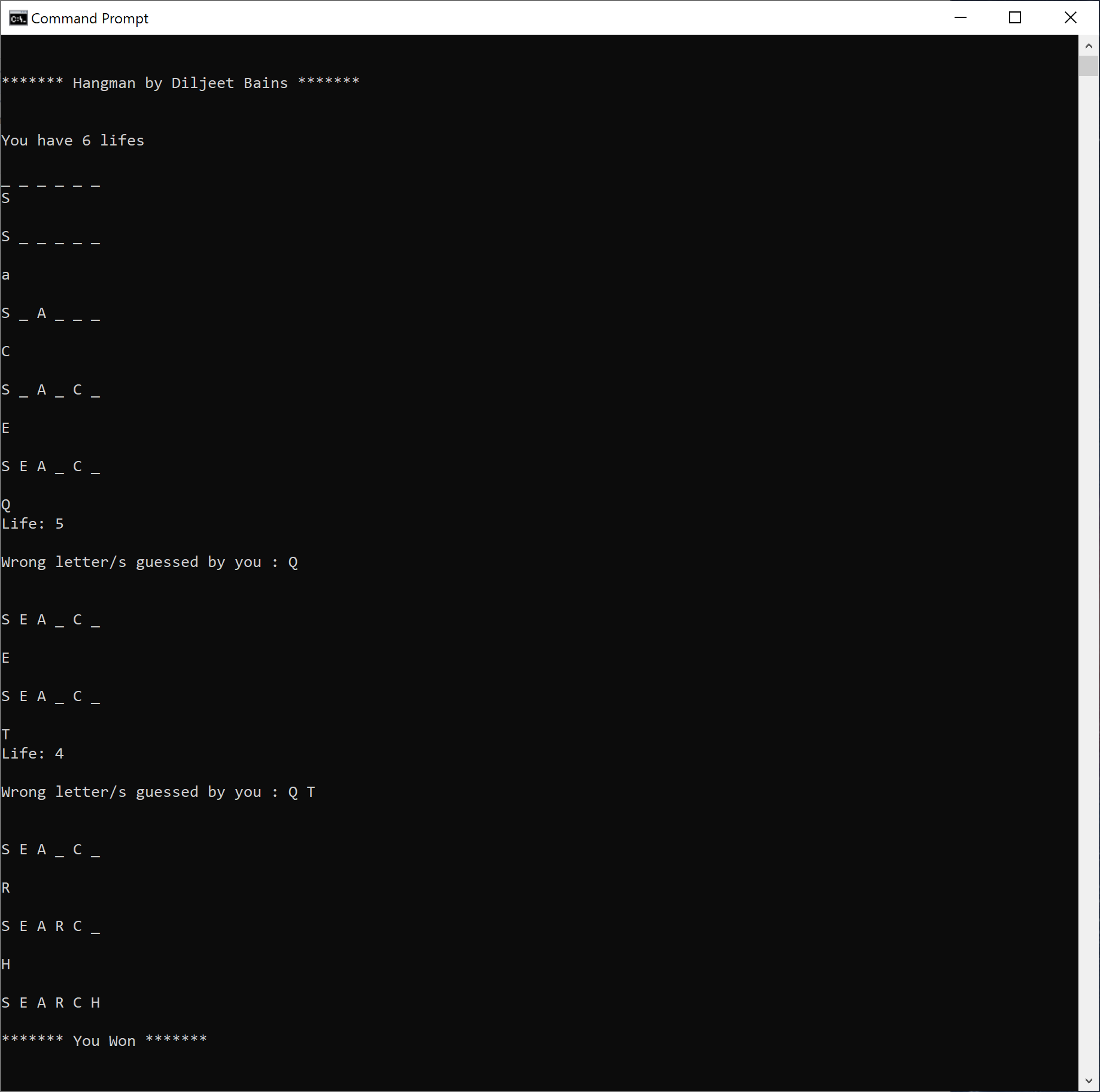
# >>>>>>>>>>> End of class <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

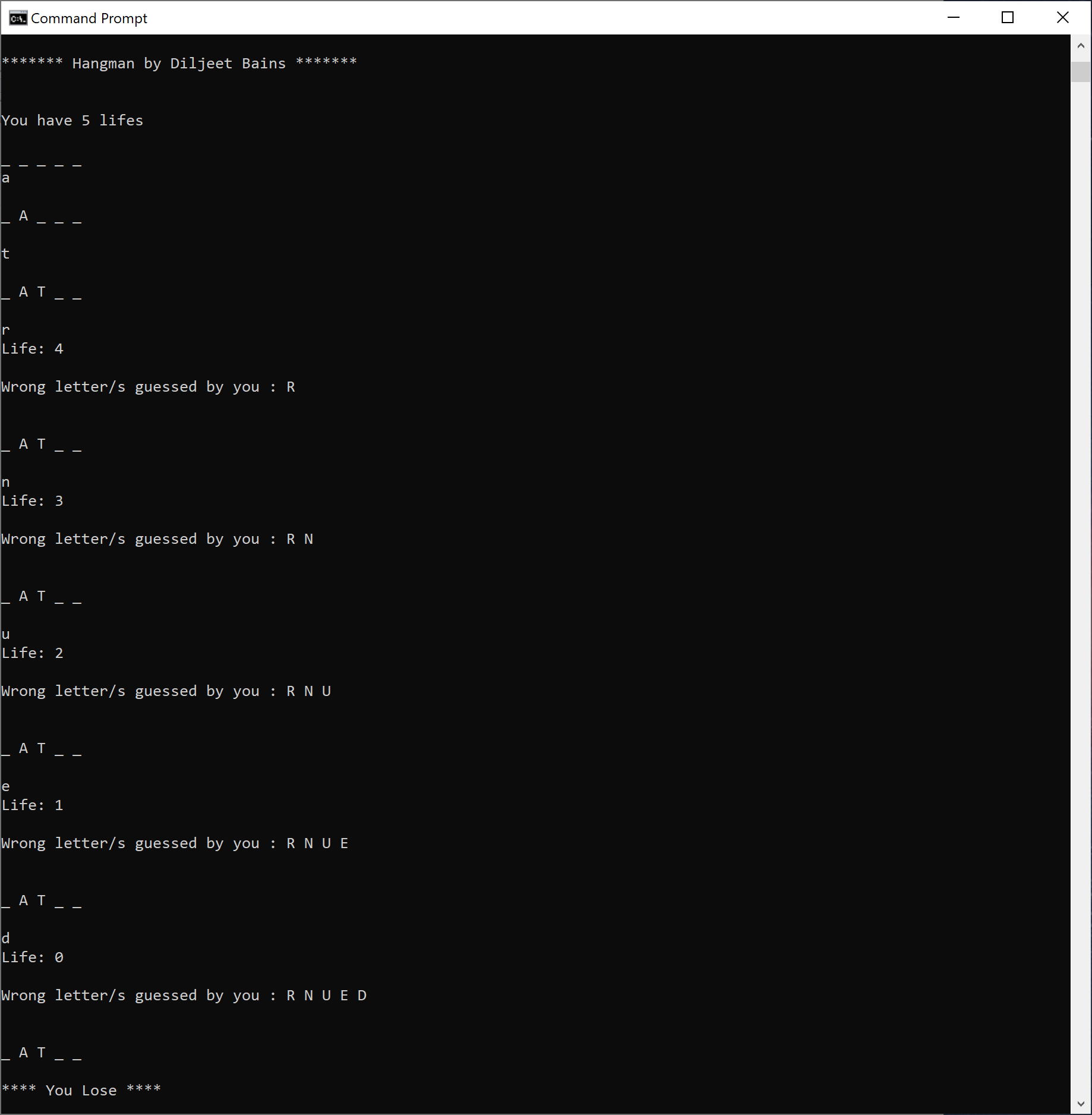
# Creating class object and calling RunGame function

**han = Hangman()**

**han.RunGame()**

**Outputs:**





**Unit Test Code:**

# >>>>>>>>>>>>>>>>>>> Unit Test <<<<<<<<<<<<<<<<<<<<<<

**import unittest**

**import Hangman**

**class TestHangman(unittest.TestCase):**

**def test\_GetRandomWord(self):**

**random\_string = Hangman.Hangman().GetRandomWord()**

**self.assertNotIn(random\_string, "")**

**def test\_GetLive(self):**

**lives = Hangman.Hangman().GetUserLives("Hangman")**

**self.assertGreater(lives, 0)**

**def test\_IsAllSpacesFilled(self):**

**result = Hangman.Hangman().IsAllSpacesFilled("Hangman")**

**self.assertTrue(result, True)**

**def test\_IsNotAllSpacesFilled(self):**

**result = Hangman.Hangman().IsAllSpacesFilled("Ha\_gman")**

**self.assertFalse(result, True)**

**def test\_FillBlankSpaces(self):**

**result = Hangman.Hangman().FillBlankSpaces("ATCH", "M", "MATCH", "\_ATCH") #MATCH**

**self.assertEquals(result, "M A T C H ")**

**if \_\_name\_\_ == '\_\_main\_\_':**

**unittest.main()**