PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS

SOFTWARE UNIT TESTING REPORT

Project: Hangman

ABSTRACT
This document is to introduce how to develop a classical game Hangman by implementing TDD approach and how to refactor code to improve program manner.

Yifan Song Student ID:s330809

Contents

Project Overview

The program, named "Hangman", is a classical word guessing game where the objective is to find the missing word based on the information given, and this program is developed using Python Unittest, Python and tkinter (The tkinter is a tool that provides Python interface to the TK GUI Toolkit).

Program Functionality

The requirements of Hangman developed in this program are as followed with relevant screenshots:

1. The Hangman has a user-friendly interface.

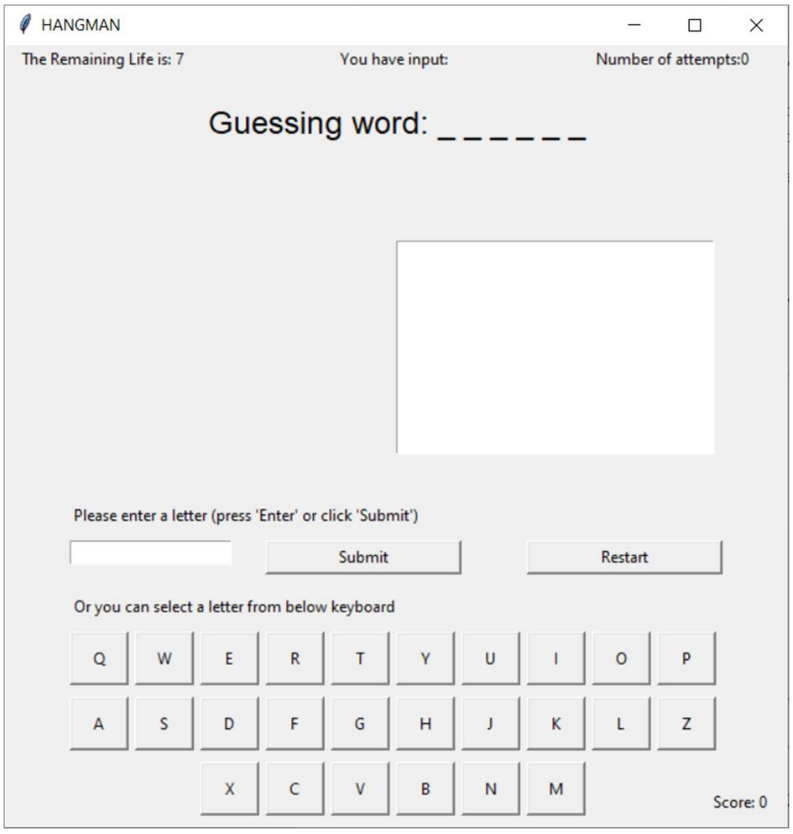


Figure 1: Hangman User Interface

- The remaining life of the player is shown on the top left corner of the window.
- The number of attempts is shown on the top right corner of the window.
- The score is shown on the bottom right corner of the window.
- The letter(s) the player inputs and the missing word are shown on the top of the window.
- The message box is to prompt players with appropriate feedback.

- There are two input methods. One is submitting what player types in the input bar by pressing the “Enter” button on the real keyboard or clicking the “Submit” button on the screen, and another method is clicking on the keyboard provided on screen.
- Also, there is a restart button allowing the player to restart the game, and the score of the player will not be affected if restarted.

2. The missing word will be selected randomly from the word list program provided. Also, it will be represented in format ‘_ _ _’ (the number of ‘_’ depends on the length of the missing word).
3. If the player chooses the incorrect letter,
    a. The part of the hangman will be generated.
    b. The remaining life will reduce 1.
    c. The number of attempts will increase 1.
    d. The letter player selected will be shown on the top.

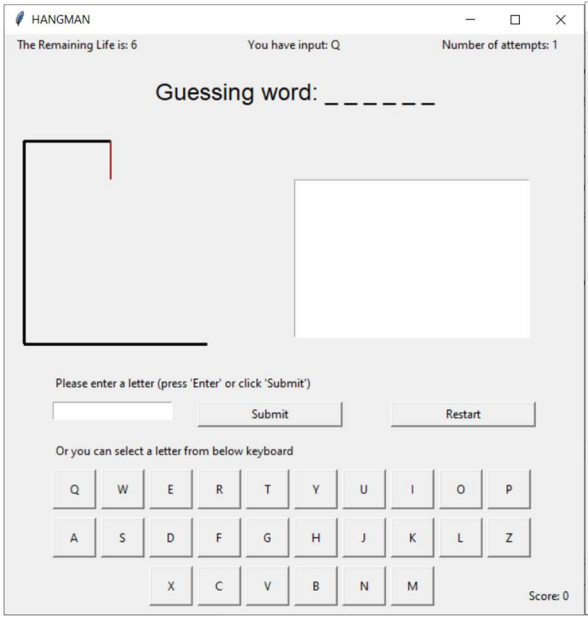


Figure 3.1: Output -1

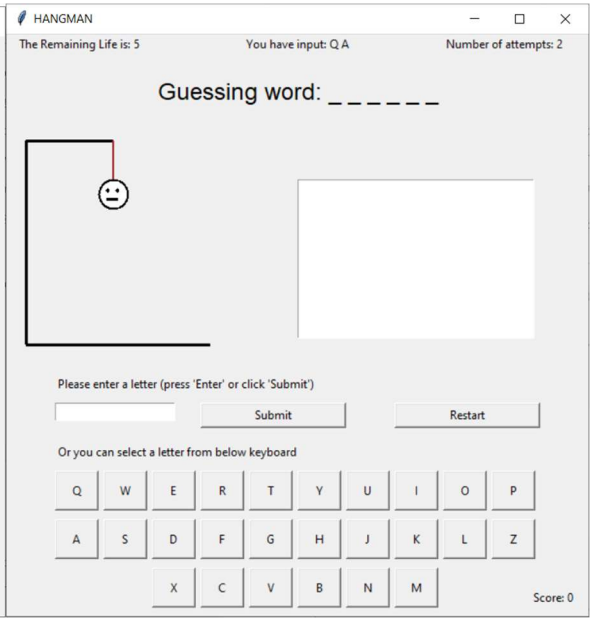


Figure 3.2: Output -2

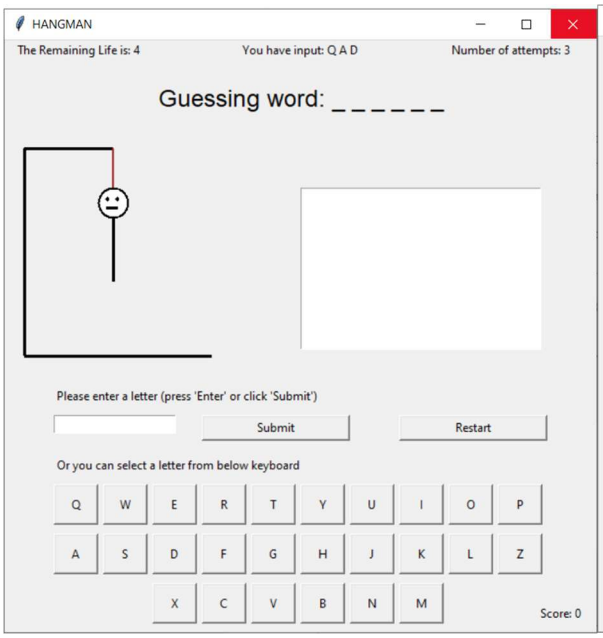


Figure 3.3: Output -3

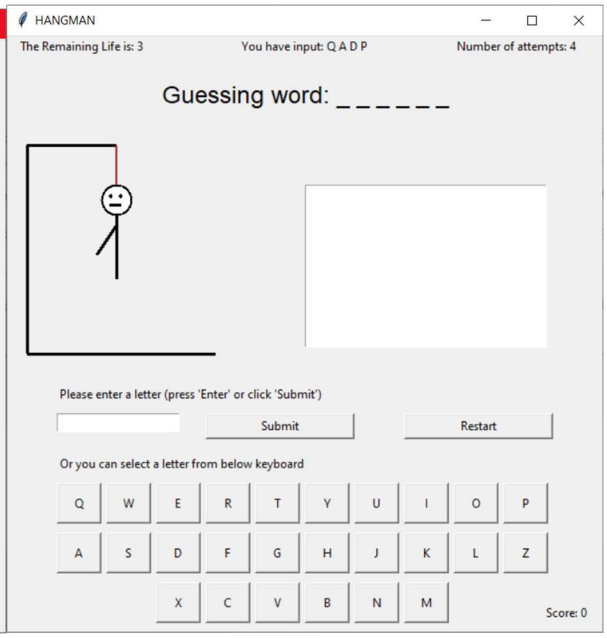


Figure 3.4: Output -4

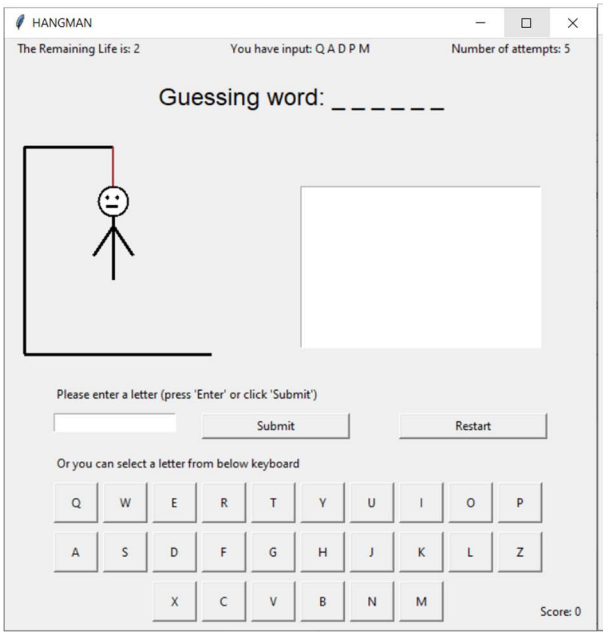


Figure 3.5: Output -5

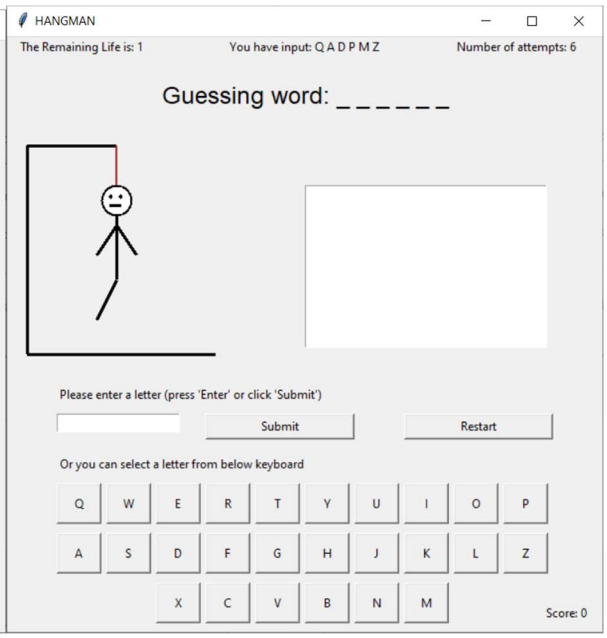


Figure 3.6: Output -6

4. The player will lose the game if the player loses all lives, and the hangman is drawn completely. The appropriate feedback (asking the player to restart) is provided in the message text box. The score will not change.

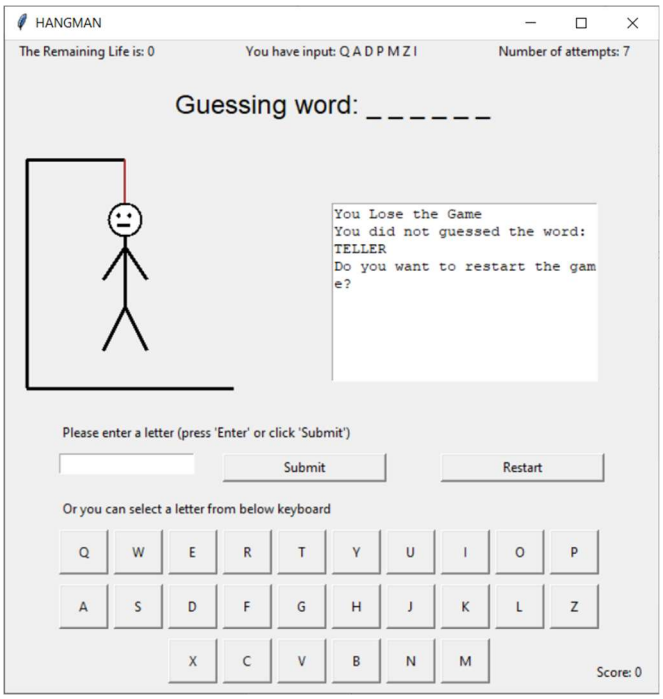
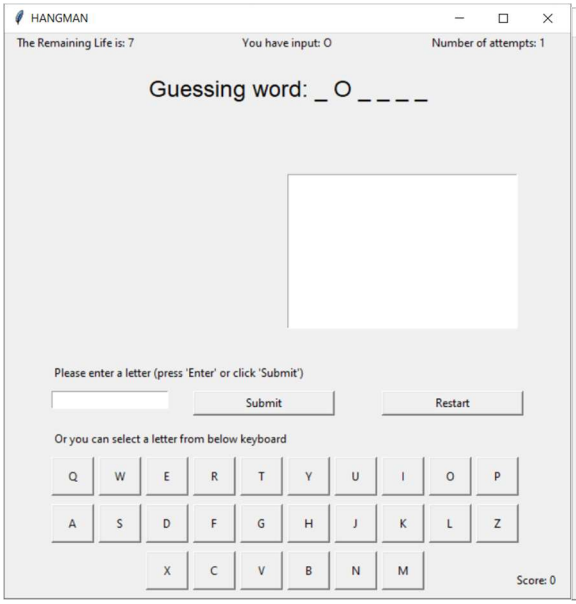


Figure 4: Lose

5. If the player chooses the correct letter,
    a. The hangman will not be generated.
    b. The remaining life will not reduce.
    c. The number of attempts will increase 1.
    d. The letter player selected will be shown on the top.
    e. The correct letter will be shown in the corresponding position of the missing word.
    f. The correct letter will replace all places in the answer where that letter appears.



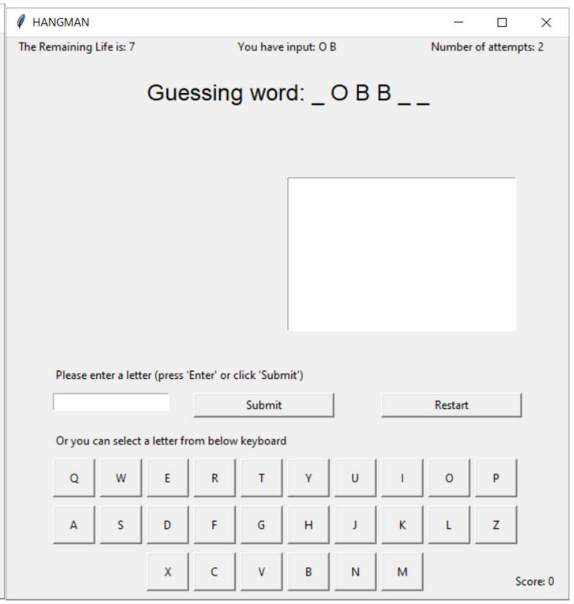


Figure 5.1: Output -1 Figure 5.2: Output -2

6. The player will win the game if the player guesses all letters and life > 0. The appropriate feedback (asking the player to restart) is provided in the message text box. Meanwhile, the score will increase by 1.

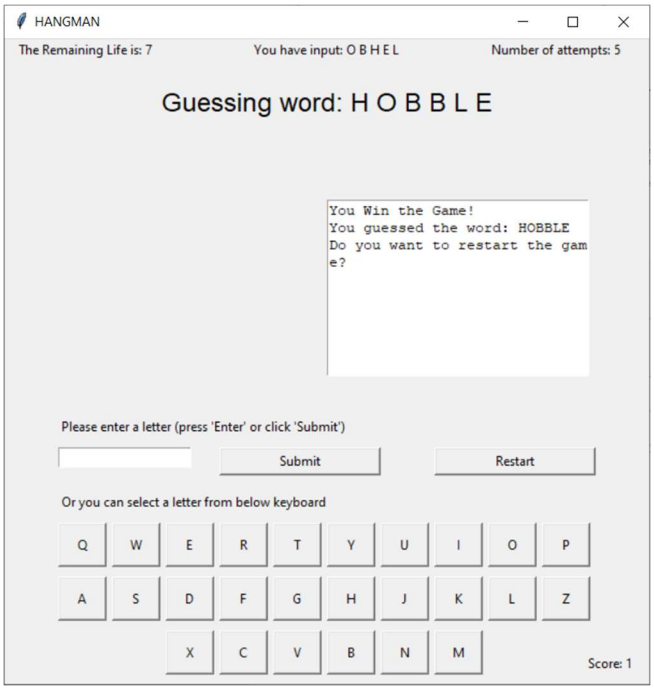


Figure 6: Win

7. The program will show the appropriate feedback based on what player inputs,
    a. If player inputs duplicate letters in input bar/select the same letter on-screen keyboard

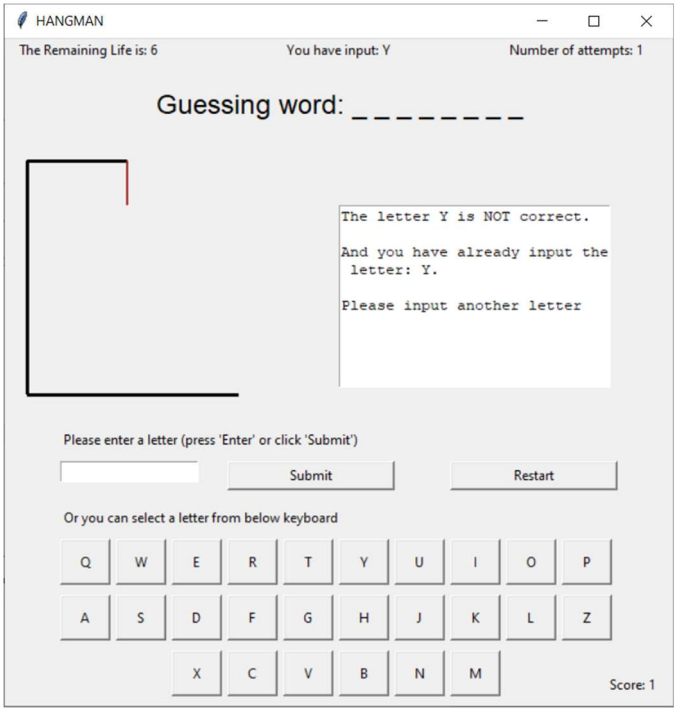


Figure 7.1: Feedback -1

b. If player inputs symbols in input bar

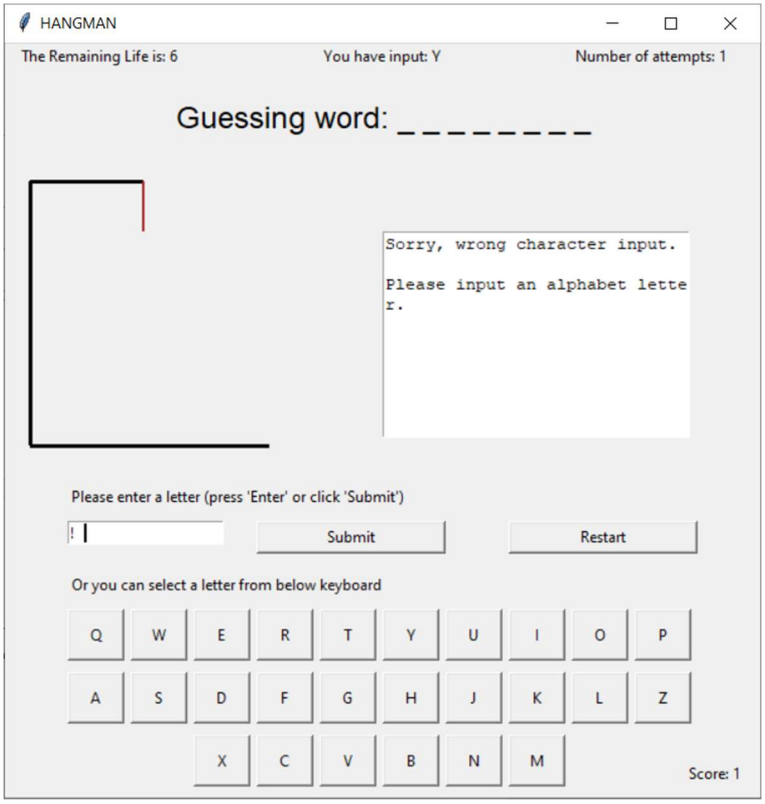


Figure 7.2: Feedback -2

a. If the player inputs more than one letter once

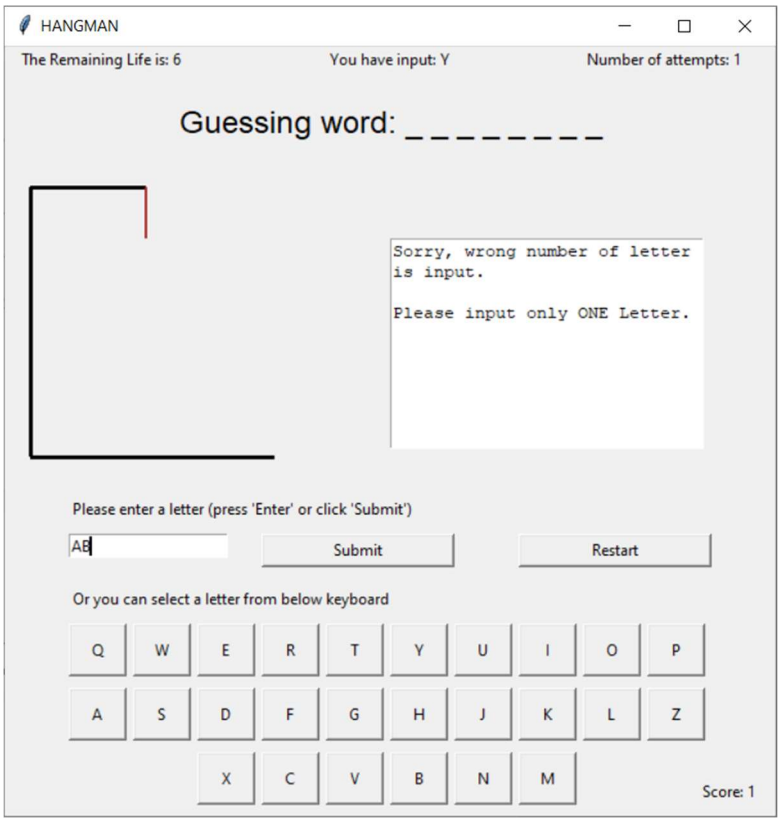


Figure 7.3: Feedback -3

8. The screen keyboard and input bar are not allowed to select any letter when the player wins/loses. The program will pop-up a widget to ask the player to restart. Or the player can restart the game clicking on “Restart” button. If the player chooses “No” for restart widget, the program will close. If the player chooses “Yes” for restart widget, the attributes of the player will be reset, except score.

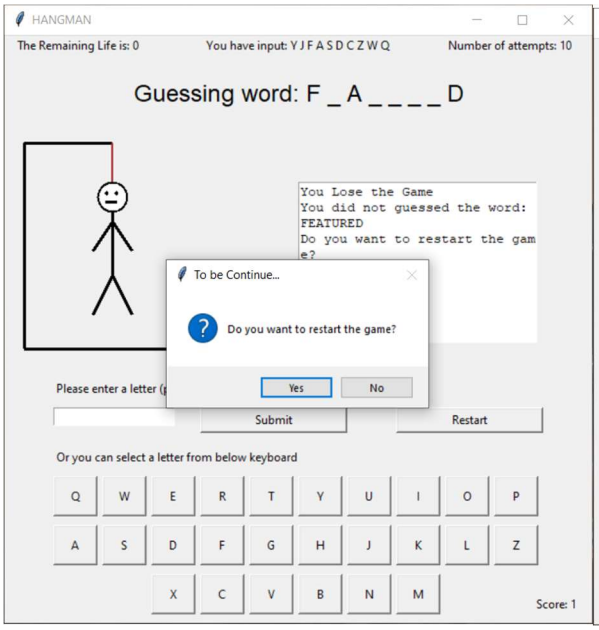


Figure 8.1: Restart Widget

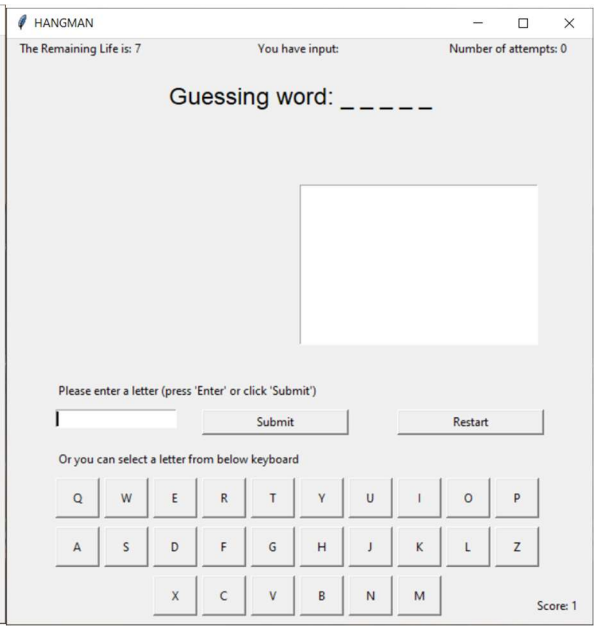


Figure 8.2: Restart

TDD implementation

The Hangman object is developed based on Test-driven Development (TDD), which means the Hangman object and unit testing related to each attribute of objects and methods are developing simultaneously. And the new method, function or attributes will not be programmed until the last unit test for existing individual part passes. TDD provides a clear logic algorithm to code a program because the test cases are written before coding the program.

Before I start developing functionalities of Hangman, I write a test for it and implement it as an automated test. After the functionality is programmed, and the relative unit test runs successfully, I move on to the next step. The Pytest is to create an instance of the Hangman object and to test individual components in isolation. These are the functionality that I use Pytest to develop:

Object Attributes

The Hangman should be built with attributes, including life, attempts, score, missing word, input word, and word. Before the player starts playing the game, all attributes should be set to default values, such as life default (7), score default(0), missing word(empty), input word(empty), and real word(empty). The test case for initializing player attributes is to make sure every attribute equal to its default value.

Functions/Methods

The unit tests have been divided into five different classes to test different functions or methods.

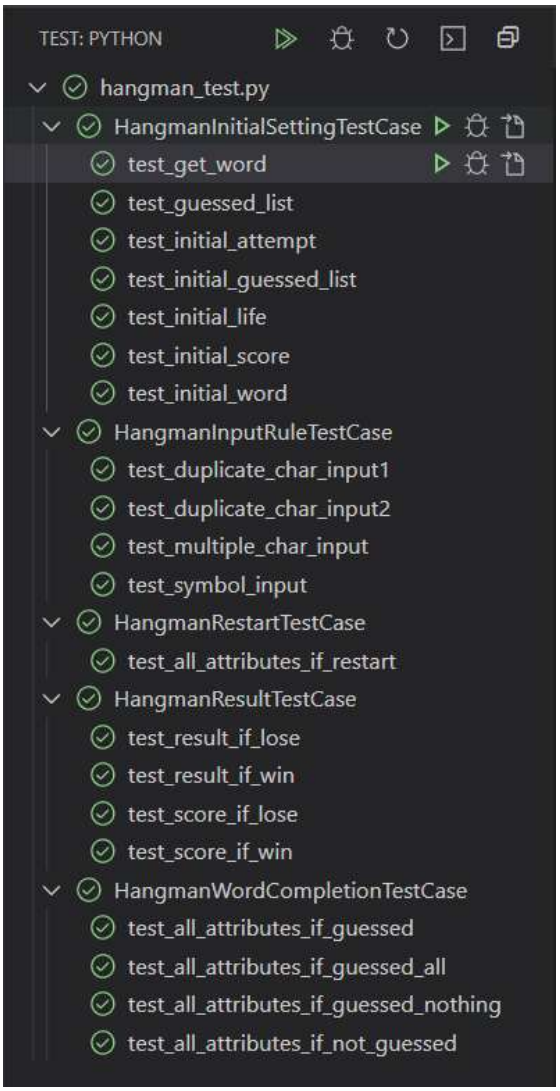


Figure 9: Unit Test

HangmanInitialSettingTestCase

This test case is to test all initial attributes are set to default values before the player starts playing game, and all attributes when the game starts.

HangmanInputRuleTestCase

This test case is to test that the system only allows players to input valid data, and make sure the system will prompt the player with the appropriate hints when plays input invalid characters, such as symbol, number, etc.

HangmanRestartTestCase

This test case is to test that the system will reset all attributes, except score, when the player wins/loses/restarts the game.

HangmanResultTestCase

This test case is to test that the system generates the correct result when the game ends.

HangmanWordCompletionTestCase

This test case is to test the system will process the valid letter in the correct manner when the player starts the game.

The above test cases test all functionality by creating an instance of the Hangman object and passing all possible inputs to each test case to get results. This approach reduces the chance of the bunch of errors occurring at the same time and provides a logical overview of how to build this program.

Refactoring

Issue #1: Duplicated Code

```
20      def __init__(self, life = 7, word =[], attempt = 0, guessed =[], letter_input=[], score=0, result=False):
21          #initialize the player's setup: Default life: 7, Attempt #: start from 0, Won't randomly pick up a word before player is ready to play,
            been guessed, player hasn't input anything before game start.
22          self.life = life
23          self.word = word
24          self.attempt = attempt
25          self.guessed =guessed
26          self.letter_input = letter_input
27          self.score = score
28          self.result = result
29
30          #initialize window
31          self.root=tkinter.Tk()
32
33          #initialize text widget
34          self.text = tkinter.Text(self.root, height = 10, width = 30)
35          self.text.place(x=300,y=150)
36
37          #create canvas object to draw hangman
38          self.canvas= tkinter.Canvas(self.root)
39
40  ........#showing the initial player's status on screen
41  ........self.label_life=tkinter.Label(text='The Remaining Life is: %d'%self.life)
42  ........self.label_life.place(x=10, y=0)
43  ........self.label_attempt=tkinter.Label(text='Number of attempts: %d'%self.attempt)
44  ........self.label_attempt.place(relx=0.75, y=0)
45  ........#self.label_word=tkinter.Label(text='The Word is: %s'%self.word)
46  ........#self.label_word.pack()
47  ........self.label_guessed=tkinter.Label(text='Guessing word: %s'%' '.join([str(v) for v in self.guessed]),font=("Helvetica", 18))
48  ........self.label_guessed.place(relx = 0.5, rely = 0.1, anchor = CENTER)
49  ........self.label_letter_input=tkinter.Label(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
50  ........self.label_letter_input.pack()
51  ........self.label_score=tkinter.Label(text='Score: %d'%self.score)
52  ........self.label_score.place(relx = 0.9, rely = 0.95)
```

Figure 10.1: __init__ function

```
260 >   def restart_widget(self):···
264
265     def restart_game(self):
266         #this function is to reset all attributes of player, except score
267         self.text.delete("1.0",tkinter.END)
268         self.canvas.delete('all')
269         self.life = 7
270         self.word = []
271         self.attempt = 0
272         self.guessed =[]
273         self.letter_input = []
274         self.result = False
275         self.get_word()
276         self.label_life.config(text='The Remaining Life is: %d' %self.life)
277         self.label_attempt.config(text='Number of attempts: %d' %self.attempt)
278         #self.label_word.config(text='The Word is: %s' %self.word)
279         self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
280         self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
281         self.label_score.config(text="Score: %d"%self.score)
```

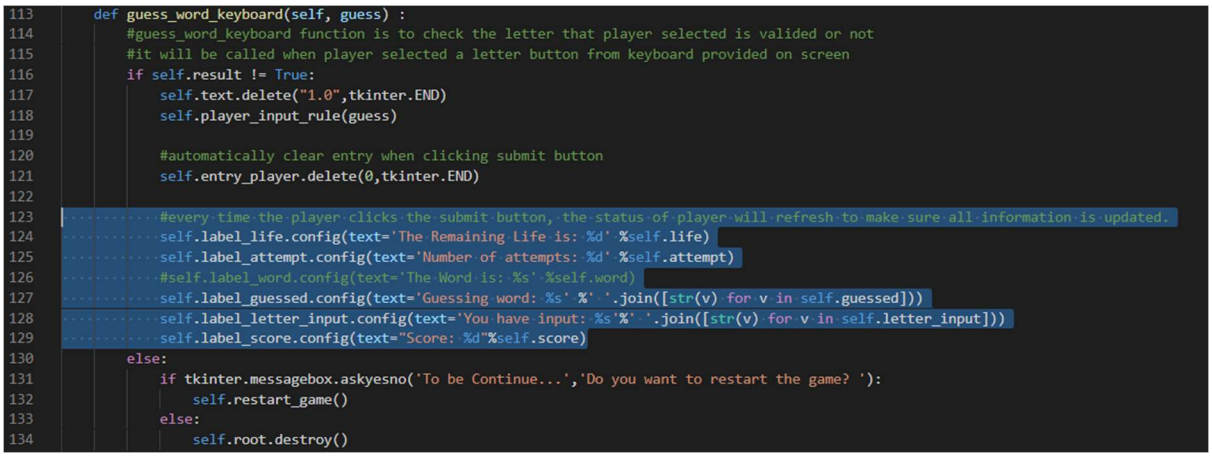
Figure 10.2: restart_game function

```
283     def main(self):
284         #function create_widget is to create a window (or GUI) showing all information, including life, attemps, question word, word
285
286         self.get_word()
287
288         #create a widget/window
289         self.root.title("HANGMAN")
290         self.root.geometry("600x600+0+0")
291         self.root.maxsize(600,600)
292         self.root.minsize(600,600)
293
294         #player attributes update
295         self.label_life.config(text='The Remaining Life is: %d' %self.life)
296         self.label_attempt.config(text='Number of attempts:%d' %self.attempt)
297         #self.label_word.config(text='The Word is: %s' %self.word)
298         self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
299         self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
300         self.label_score.config(text="Score: %d"%self.score)
301
302
303         #restart button
304         self.button_restart = tkinter.Button(self.root, text="Restart",command = self.restart_game)
305         self.button_restart.config(width=20, height=1)
306         self.button_restart.place(x=400, y=380)
307
308         #hint widget/windows will prompt users with the appropriate feedback
```

Figure 10.3: main function

```
85      def guess_word_submit_btn(self,event=None) :
86          #guess_word_submit_btn will be called when player clicks the submit button
87          if self.result != True:
88          #if player hasn't completed the word...
89              print('Submit button is clicked.')
90              self.text.delete("1.0",tkinter.END)
91              #read user's input from input bar
92              guess=self.entry_player.get().upper()
93
94              self.player_input_rule(guess)
95
96              #automatically clear entry when clicking submit button
97              self.entry_player.delete(0,tkinter.END)
98
99              #every time the player clicks the submit button, the status of player will refresh to make sure all information is updated.
100             self.label_life.config(text='The Remaining Life is: %d' %self.life)
101             self.label_attempt.config(text='Number of attempts: %d' %self.attempt)
102             #self.label_word.config(text='The Word is: %s' %self.word)
103             self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
104             self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
105             self.label_score.config(text="Score: %d"%self.score)
106         else:
107         #if player has completed the word...
108             if tkinter.messagebox.askyesno('To be Continue...','Do you want to restart the game? '):
109                 self.restart_game()
110             else:
```

Figure 10.4: guess_word_submit_btn function

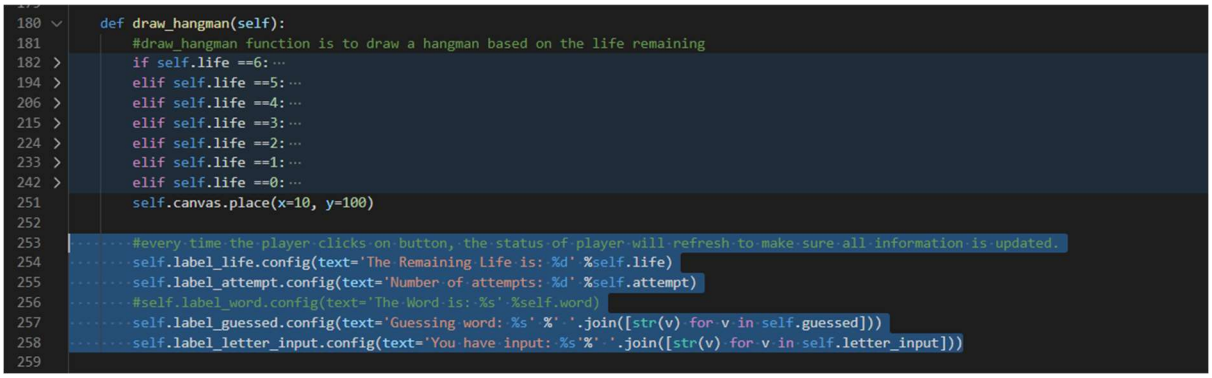


```
    def guess_word_keyboard(self, guess) :
        #guess_word_keyboard function is to check the letter that player selected is valided or not
        #it will be called when player selected a letter button from keyboard provided on screen
        if self.result != True:
            self.text.delete("1.0",tkinter.END)
            self.player_input_rule(guess)

            #automatically clear entry when clicking submit button
            self.entry_player.delete(0,tkinter.END)

            #every time the player clicks the submit button, the status of player will refresh to make sure all information is updated.
            self.label_life.config(text='The Remaining Life is: %d' %self.life)
            self.label_attempt.config(text='Number of attempts: %d' %self.attempt)
            #self.label_word.config(text='The Word is: %s' %self.word)
            self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
            self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
            self.label_score.config(text="Score: %d"%self.score)
        else:
            if tkinter.messagebox.askyesno('To be Continue...','Do you want to restart the game? '):
                self.restart_game()
            else:
                self.root.destroy()
```

Figure 10.5: guess_word_keyboard function



```
    def draw_hangman(self):
        #draw_hangman function is to draw a hangman based on the life remaining
        if self.life ==6:···
        elif self.life ==5:···
        elif self.life ==4:···
        elif self.life ==3:···
        elif self.life ==2:···
        elif self.life ==1:···
        elif self.life ==0:···
        self.canvas.place(x=10, y=100)

        #every time the player clicks on button, the status of player will refresh to make sure all information is updated.
        self.label_life.config(text='The Remaining Life is: %d' %self.life)
        self.label_attempt.config(text='Number of attempts: %d' %self.attempt)
        #self.label_word.config(text='The Word is: %s' %self.word)
        self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
        self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
```

Figure 10.6: draw_hangman function

In function "__init__", the labels representing life remaining, number of attempts, score, missing word, and letter input, are created and initialize the position on the parent widget.

In functions, "restart_game", "main", "guess_word_submit_btn", "guess_word_keyboard", and "draw_hangman", modify the texts of labels to update the current player's attributes using four or five lines of duplicated code.

Solution: Extract Method

To refactor the duplicated code, a new function, called "update_player_attribute", is created. Then, extracting the common code to this function, and removing the code from the above functions. In above functions (except __init__), replacing blue area (above images) with code:

```
self.update_player_attribute()
```

```
262     #refactoring: Duplicated code
263     def update_player_attribute(self):
264         #player attributes update
265         self.label_life.config(text='The Remaining Life is: %d' %self.life)
266         self.label_attempt.config(text='Number of attempts:%d' %self.attempt)
267         #self.label_word.config(text='The Word is: %s' %self.word)
268         self.label_guessed.config(text='Guessing word: %s' %' '.join([str(v) for v in self.guessed]))
269         self.label_letter_input.config(text='You have input: %s'%' '.join([str(v) for v in self.letter_input]))
270         self.label_score.config(text="Score: %d"%self.score)
```

Figure 10.7: solution – update_player_attribute function

Issue #2: Duplicated Code and Nested Loop

```
82      def guess_word_submit_btn(self,event=None) :
83          #guess_word_submit_btn will be called when player clicks the submit button
84          if self.result != True:
85          #if player hasn't completed the word...
86              print('Submit button is clicked.')
87              self.text.delete("1.0",tkinter.END)
88
89              #read user's input from input bar
90              guess=self.entry_player.get().upper()
91
92              #check input is valid or not
93              self.player_input_rule(guess)
94
95              #automatically clear entry when clicking submit button
96              self.entry_player.delete(0,tkinter.END)
97
98              #refactoring implementation: Duplicated code
99              self.update_player_attribute()
100         else:
101         #if player has completed the word...
102             if tkinter.messagebox.askyesno('To be Continue...','Do you want to restart the game? '):
103                 self.restart_game()
104             else:
105                 self.root.destroy()
```

Figure 11.1: guess_word_submit_btn function

```
107     def guess_word_keyboard(self, guess) :
108         #guess_word_keyboard function is to check the letter that player selected is valided or not
109         #it will be called when player selected a letter button from keyboard provided on screen
110         if self.result != True:
111             self.text.delete("1.0",tkinter.END)
112             self.player_input_rule(guess)
113
114             #automatically clear entry when clicking submit button
115             self.entry_player.delete(0,tkinter.END)
116             #refactoring implementation: Duplicated code
117             self.update_player_attribute()
118         else:
119             if tkinter.messagebox.askyesno('To be Continue...','Do you want to restart the game? '):
120                 self.restart_game()
121             else:
122                 self.root.destroy()
```

Figure 11.2: guess_word_keyboard function

Both “guess_word_submit_btn” and “guess_word_keyboard” should be able to restart or close the program. There is a nested if conditional statement which also causes a duplicated code problem.

Solution: Extract Method

Extracting the duplicated code and creating a new function called “restart_widget”.

```
243   #refactoring: Duplicated code and nested loop
244    def restart_widget(self):
245        #this function is to create a restart window and get user repsonses
246        if tkinter.messagebox.askyesno('To be Continue...','Do you want to restart the game? '):
247            self.restart_game()
248        else:
249            self.root.destroy()
250
```

Figure 11.3: solution – restart_widget function

Program files

Directory

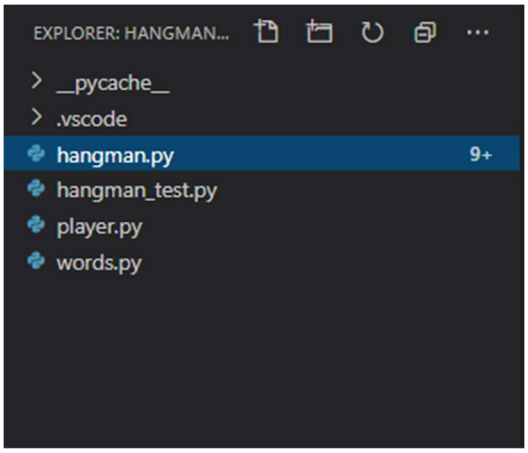


Figure 12.1: Directory

- hangman.py – generating Hangman Object
- hangman.py – python unit test by creating an instance of Hangman Object
- player.py – creating an instance of Hangman Object to run this program
- words.py – providing a word list to allow Hangman selecting a wordSSS from there randomly

Python Unit Test Result

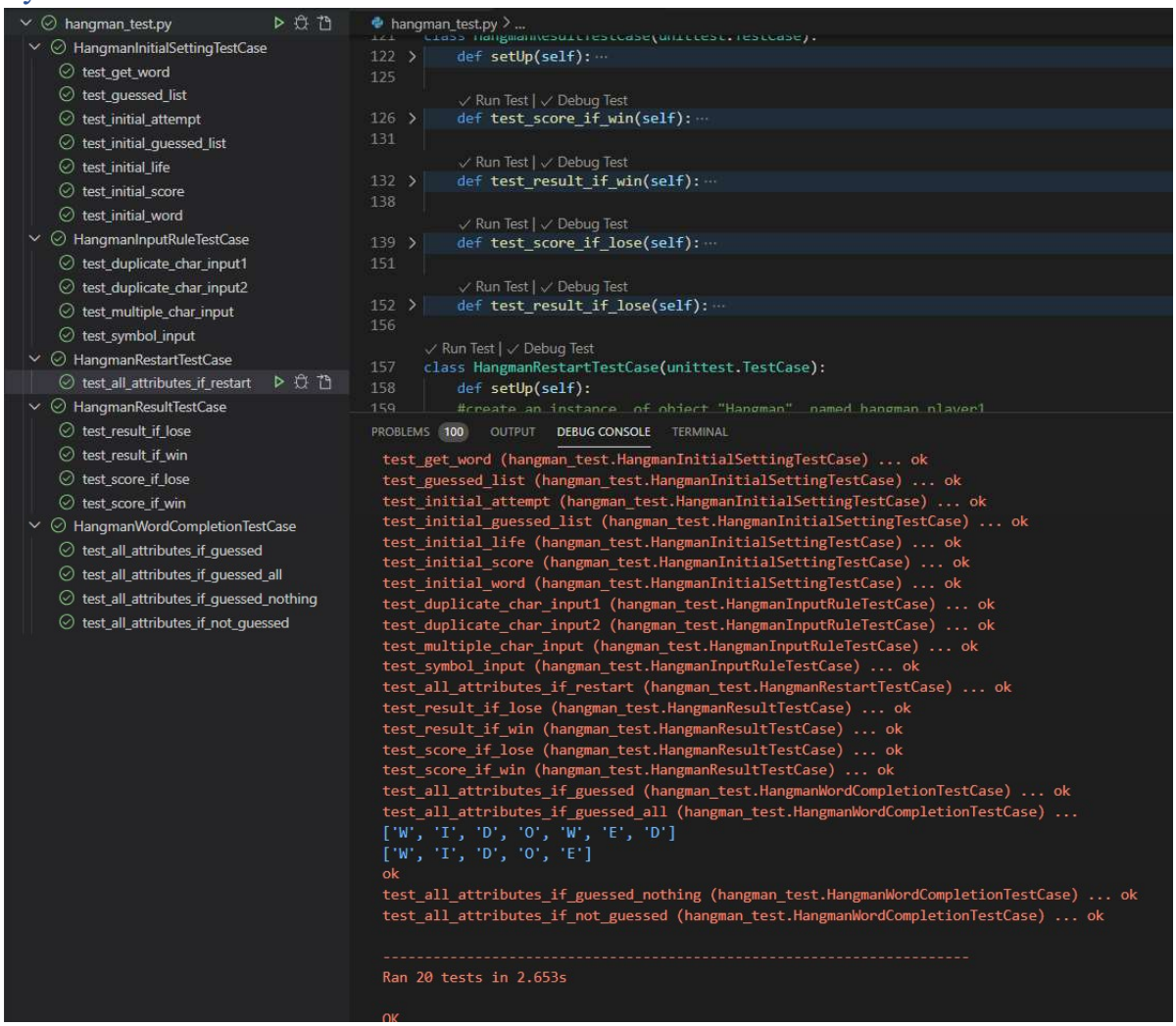


Figure 12.2: Unit Test Result

Code (Github Link)

https://github.com/Yifansong1120/Assignmen01_Hangman.git

Note: For running Hangman Game, please download all files and make sure the python version is 3.7.6 or the newest version. Please run the 'Player.py' file to start the game.