(Established under Karnataka Act No. 16 of 2013)

100 Feet Ring Road, BSK III Stage, Bengaluru-560 085

Department of Computer Science & Engineering

PROJECT HANGMAN

Vishal Nagaraj: PES1202100683

Ujwal Dinesh Jain: PES1202100647

Siddarth Rao: PES2202100787

ABSTRACT

For our first mini-project, we had chosen the topic, ‘hangman’. It is a fun, word-guessing game which involves multiple players, where one enters a word to be guessed by other players, letter by letter. The code we have created takes a string as an input. The code then takes a series of inputs in the form of singular characters, and then performs multiple comparisons with the word so as to determine whether or not the character is a part of the word. The player is to guess the word successfully, before their eventual mistakes result in the man being hung.

Table of Contents

Introduction

The game hangman is fun and interactive game which enhances the thinking skills and risk assessment skills of a player. While trying to guess the word\phrase which has been provided by the other player, one must be careful while listening to the hints and must be able to correlate to various objects, people, and other points of interest. The game also enables a fair playing ground for all players.

This project was created by us to try and understand the idea of how we can process strings, containing words, characters, and in some cases symbols as well, in order to deconstruct the workings behind something as simple as a game of hangman converted into a computer format. The scope of string comprehension is extremely important in this day and age, primarily in search engines, where we are provided with the desired search results based on what we type, and more recently, personalized recommendations to each user based on an aggregated search and browsing history. With the onset of AI, and virtual assistants, which rely on the comprehension and understandings of user requests, in the form of statements, questions, and commands, while also being able to generate text on its own to provide back to the user, with the help of text-to-speech engines, it has become more and more important to develop and refine the machines ability to comprehend and understand data, in order for these machines to become more and more viable alternatives in the workspace, thus exponentially increasing work efficiency.

Design And Implementation

The code is as follows:

```
#Importing all pertinent modules.
import tkinter
from PIL import ImageTk,Image
import tkinter.messagebox
import os
number = 0


def change_image(number):
    #Updating the image
    global lab
    lab.grid_forget()
    lab = tkinter.Label(image=im_list[number+1])
    lab.place(relx=0.5, rely=0.39, anchor="center")

    if number == 5:     #WHEN THE NUMBER OF TURNS ARE EXHAUSTED
        tkinter.messagebox.showinfo('Game Over.', message="The man was
hung. Thank you for playing!")
        root.destroy()
    number += 1
    return number


def check(button, no):
    global e2
    global number
    global lab

    text = button["text"]

    no = list(uppercase).count(text)
    print(no)

    if uppercase=='':
        tkinter.messagebox.showinfo(message="Make sure Player 1 enters a
word first.")

    elif (no == 0): #Changes the image label when a mistake has been made
```

```
        number = change_image(number)

    else: #Updating list hangman and giving it to label (if the letter is
in the input)
        for i in range(len(uppercase)):
            if uppercase[i] == text:
                hangman[i] = text

    #Deleting previous hangman label, printing the updated one
    e2.grid_forget()
    e2 = tkinter.Label(root,text=' '.join(hangman).upper(), width=40,
height=2, relief='sunken',borderwidth=5, font=("Rosewood Std
Fill",15,"bold"))
    e2.place(relx=.09,rely=.09)

    button.destroy()

    if '_' not in hangman:
        tkinter.messagebox.showinfo(message="Awesome! You guessed the
word!")
        root.destroy()

def obtain_destroy():
    global hangman
    global uppercase
    global e2
    #Obtaining the input from Player 1
    lo = e1.get()
    if len(lo) == 0:
        tkinter.messagebox.showinfo(message="You must enter something!")
        e1.delete(0,tkinter.END)
        return 0

    if lo.isalpha()==True or ' ' in lo:

            uppercase+=lo.upper()
            e2.destroy()
            e1.destroy()
            b1.destroy()
            hangman = ['_' if i!=' ' else ' ' for i in uppercase]
            e2 = tkinter.Label(root,text=' '.join(hangman).upper(),
width=40, height=2, relief='sunken',borderwidth=5, font=("Rosewood Std
Fill",15,"bold"))
            e2.place(relx=.09,rely=.09)
    else:
```

```
        tkinter.messagebox.showinfo(message="Please only enter letters of
the alphabet, please.")
        e1.delete(0,tkinter.END)


    print(lo)

#Creating the window for player 2
root = tkinter.Tk()
root.title("Hangman")
root.geometry("600x590")
root.configure(bg='#CBB3E4')
#Defining the label that displays completion of the word.
e2 = tkinter.Label(root)

#Creating path such that any user may access said file
userpath = os.environ['USERPROFILE']

#Creating variables that point to the displayed images
im=ImageTk.PhotoImage(Image.open(f"{userpath}\Downloads\HangmanProject\What
sApp Image 2022-01-12 at 7.24.31 PM.jpeg"))
im2=ImageTk.PhotoImage(Image.open(f'{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (1).jpeg'))
im3=ImageTk.PhotoImage(Image.open(f"{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (2).jpeg"))
im4=ImageTk.PhotoImage(Image.open(f'{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (3).jpeg'))
im5=ImageTk.PhotoImage(Image.open(f'{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (4).jpeg'))
im6=ImageTk.PhotoImage(Image.open(f'{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (5).jpeg'))
im7=ImageTk.PhotoImage(Image.open(f'{userpath}\Downloads\HangmanProject\Wha
tsApp Image 2022-01-12 at 7.24.31 PM (6).jpeg'))
#Creating a list of all the images to cycle through.
im_list = [im, im2, im3, im4, im5, im6, im7]
#Creating the frame of the images
lab = tkinter.Label(image=im_list[0], anchor=tkinter.CENTER)
lab.place(relx=.5, rely=.39, anchor="center")

#Defining an empty string to which the input string is appended.
uppercase = ''
#x allows Player 1 to input the word to be guessed.
x = tkinter.StringVar()
#Defining and placing the entry field as well as the submit button for
Player 1.
```

```
e1 = tkinter.Entry(root,borderwidth=4,show='*',relief='sunken')
e1.place(relx=.36,rely=.09)
b1 = tkinter.Button(root,text="Submit", command=obtain_destroy)
b1.place(relx=.455, rely=.14)

#DEFINING OF VIRTUAL KEYBOARD OBJECTS. Place and beautify these!

Q = tkinter.Button(root, width=2, height= 1, borderwidth=3,
relief='raised', font=("Helvetica", "20"), text='Q', bg="#FCE8EA",
command=lambda: check(Q, 0))
Q.place(relx=0.05, rely=0.6)

W = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='W', bg="#FCE8EA", borderwidth=3,
command=lambda: check(W, 0))
W.place(relx=0.14, rely=0.6)

E = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='E', bg="#FCE8EA", borderwidth=3, command=lambda: check(E, 0))
E.place(relx=0.23, rely=0.6)

R = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='R', bg="#FCE8EA", borderwidth=3, command=lambda: check(R, 0))
R.place(relx=0.32, rely=0.6)

T = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='T', bg="#FCE8EA", borderwidth=3, command=lambda: check(T, 0))
T.place(relx=0.41, rely=0.6)

Y = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='Y', bg="#FCE8EA", borderwidth=3, command=lambda: check(Y, 0))
Y.place(relx=0.5, rely=0.6)

U = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='U', bg="#FCE8EA", borderwidth=3, command=lambda: check(U, 0))
U.place(relx=0.59, rely=0.6)

I = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='I', bg="#FCE8EA", borderwidth=3,
command=lambda: check(I, 0))
I.place(relx=0.68, rely=0.6)

O = tkinter.Button(root, relief='raised',width=2, height= 1,
font=("Helvetica", "20"), text='O', bg="#FCE8EA", borderwidth=3,
command=lambda: check(O, 0))
```

```
O.place(relx=0.77, rely=0.6)

P = tkinter.Button(root, relief='raised', width=2, height= 1,
font=("Helvetica", "20"), text='P', bg="#FCE8EA", borderwidth=3,
command=lambda: check(P, 0))
P.place(relx=0.86, rely=0.6)

A = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='A', bg="#CFFED4", borderwidth=3,
command=lambda: check(A, 0))
A.place(relx=0.09, rely=0.71)

S = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='S', bg="#CFFED4", borderwidth=3,
command=lambda: check(S, 0))
S.place(relx=0.18, rely=0.71)

D = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='D', bg="#CFFED4", borderwidth=3,
command=lambda: check(D, 0))
D.place(relx=0.27, rely=0.71)

F = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='F', bg="#CFFED4", borderwidth=3,
command=lambda: check(F, 0))
F.place(relx=0.36, rely=0.71)

G = tkinter.Button(root, relief='raised', font=("Helvetica", "20"),
text='G', bg="#CFFED4", borderwidth=3, command=lambda: check(G, 0))
G.place(relx=0.45, rely=0.71)

H = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='H', bg="#CFFED4", borderwidth=3,
command=lambda: check(H, 0))
H.place(relx=0.54, rely=0.71)

J = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='J', bg="#CFFED4", borderwidth=3,
command=lambda: check(J, 0))
J.place(relx=0.63, rely=0.71)

K = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='K', bg="#CFFED4", borderwidth=3,
command=lambda: check(K, 0))
K.place(relx=0.72, rely=0.71)
```

```
L = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='L', bg="#CFFED4", borderwidth=3,
command=lambda: check(L, 0))
L.place(relx=0.81, rely=0.71)

Z = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='Z', bg="#CAF3F9", borderwidth=3,
command=lambda: check(Z, 0))
Z.place(relx=0.18, rely=0.82)

X = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='X', bg="#CAF3F9", borderwidth=3,
command=lambda: check(X, 0))
X.place(relx=0.27, rely=0.82)

C = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='C', bg="#CAF3F9", borderwidth=3,
command=lambda: check(C, 0))
C.place(relx=0.36, rely=0.82)

V = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='V', bg="#CAF3F9", borderwidth=3,
command=lambda: check(V, 0))
V.place(relx=0.45, rely=0.82)

B = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='B', bg="#CAF3F9", borderwidth=3,
command=lambda: check(B, 0))
B.place(relx=0.54, rely=0.82)

N = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='N', bg="#CAF3F9", borderwidth=3,
command=lambda: check(N, 0))
N.place(relx=0.63, rely=0.82)

M = tkinter.Button(root, width=2, height= 1, relief='raised',
font=("Helvetica", "20"), text='M', bg="#CAF3F9", borderwidth=3,
command=lambda: check(M, 0))
M.place(relx=0.72, rely=0.82)

root.mainloop()
```

The following are the pertinent modules used in the program.

1) tkinter:

"The tkinter package ("Tk interface") is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems."

'Tkinter' is essentially the default GUI option for Python. 'Tkinter' comprises 'Tk', written by John Ousterhout, 'Tkinter' by Steen Lumholt and Guido van Rossum.

'Tkinter' uses a class known as 'Widgets'. These are divided into subclasses such as 'Label', 'Button', 'Entry', etc. 'Tkinter' is object oriented in nature. Programming using 'tkinter 'is generally a two-step process: defining the various widgets being used in the program and introducing them to the output window. Additionally, widgets can be beautified by altering the parameters of the same.

The program makes liberal use of labels, entries and buttons. Buttons have the unique property of being able to point to functions, which proves useful for performing specific tasks. Additionally, messageboxes have been included which are indispensable for the basic flow of the game as well as user experience.

2) Pillow

"This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool."

Pillow is the fork of the Python Image Library module. The glaring issue with the latter is the sheer outdatedness of it. Not only is image processing less feasible, but the module can only work smoothly with .gif images.

The program makes use of this module in opening .jpeg images. In order to cycle through the various stages of hangman, a list 'im_list' is created, containing variables to which images have been assigned. When the variable 'number', starting from 0, goes to 5 (i.e. 6 mistakes are made), the label in which the image is contained is destroyed and replaced by another one with the updated image using list indexing.

3) os

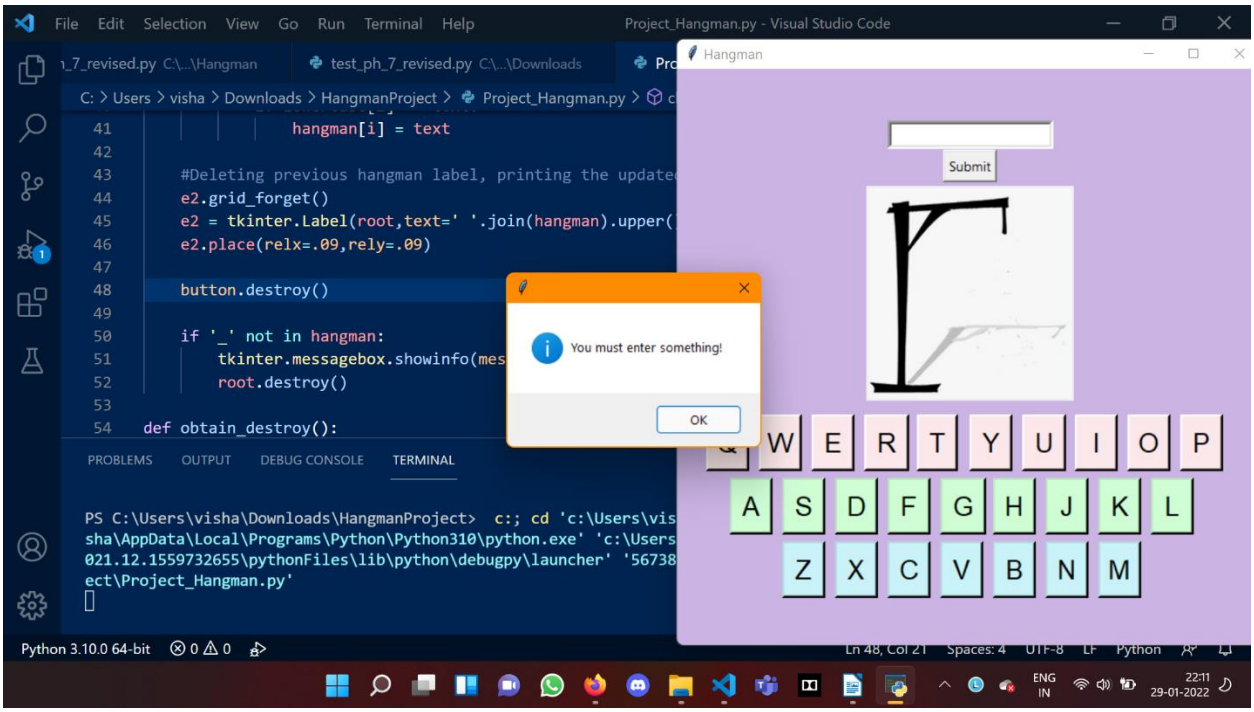
"This module provides a portable way of using operating system dependent functionality."

The module predominantly deals with two categories: system paths and internal IDs, for which it contains a myriad of functions and methods, as well as low level file handling. The program uses this to simplify the process of changing the source of the image files, which will vary from system to system. It enhances the user experience significantly. `{userpath}` returns a string of the directory in which the program is being executed. The fixed image path is then appended to this.

Testing

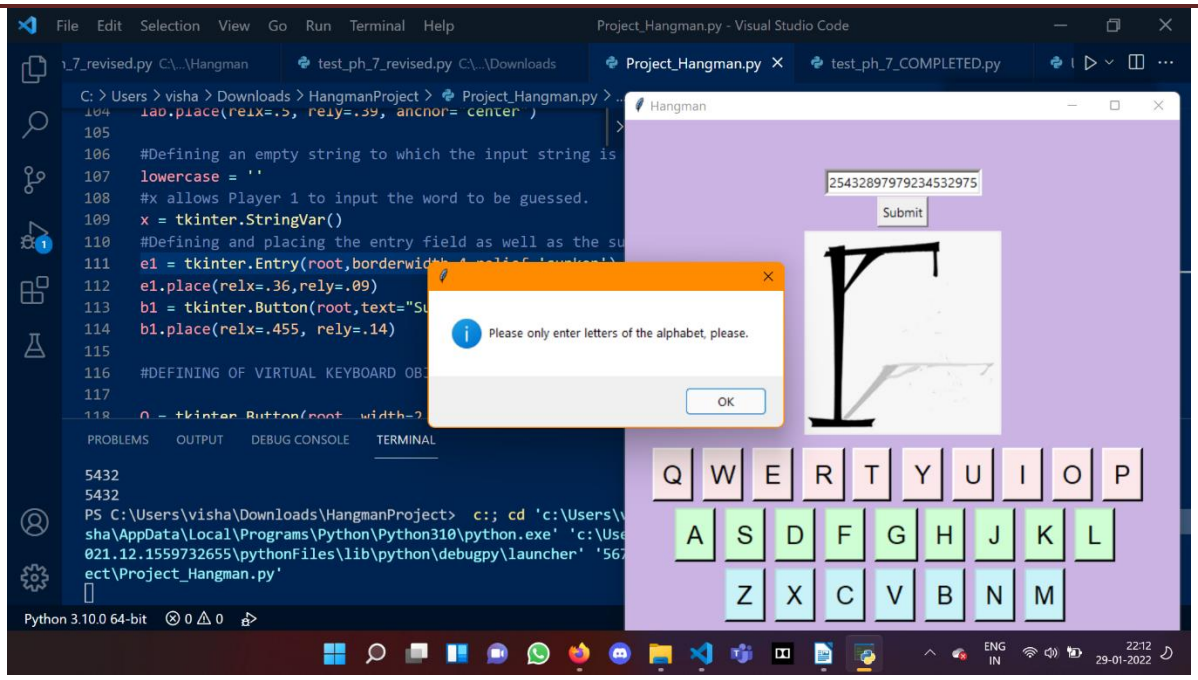
Here we run through the various test cases of the program. It has been designed to withstand erroneous mistakes such as Player 1 not entering a word and the inclusion of numeric and or special characters in the program.

1) Entering a null character



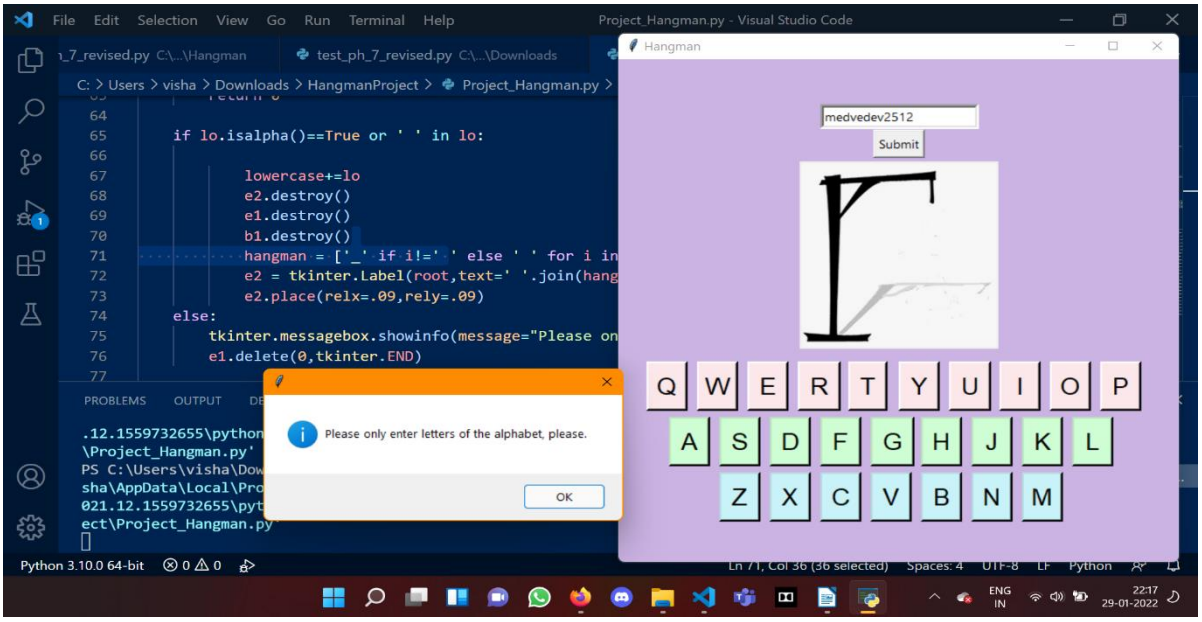
In the above screenshot, the submit button was entered immediately after the window pops up. We immediately get this error.

2) Entering only integers



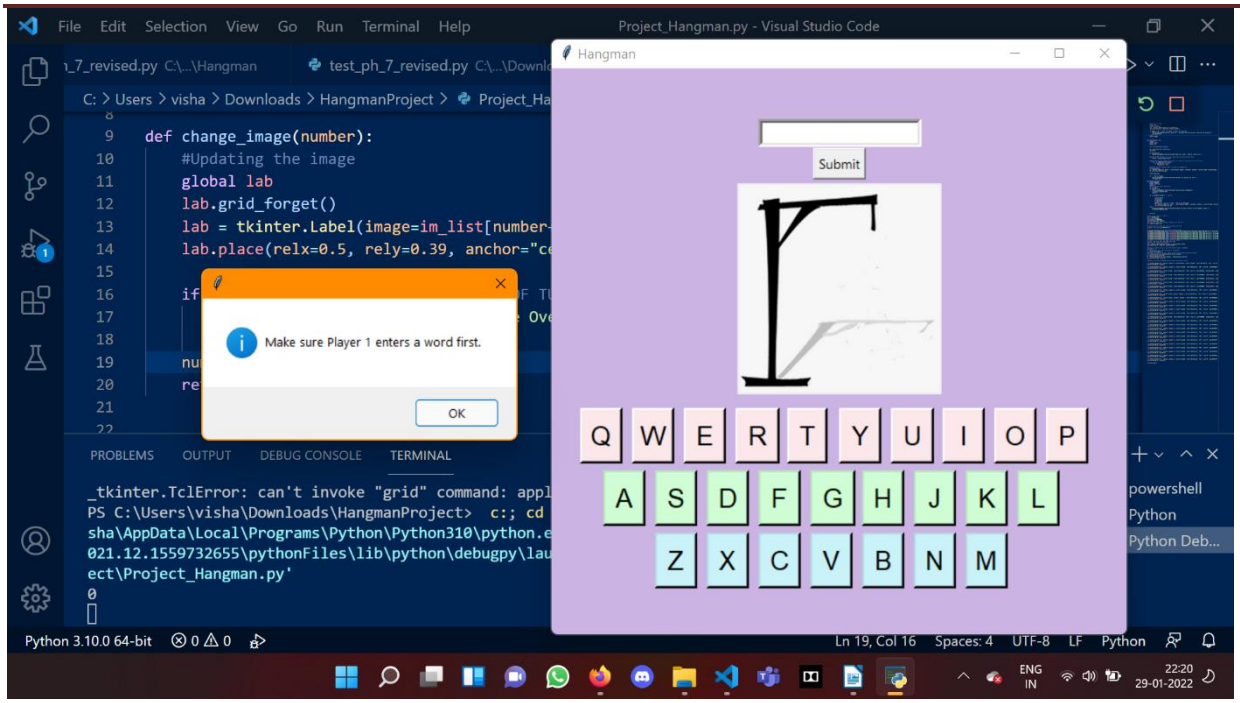
Only numbers are entered in the above image. It produces a unique messagebox giving an error.

3) Both a character and integer are entered



When ‘medvedev2512’ is entered. It throws Player 1 a unique error.

4) When Player 2 enters before Player 1



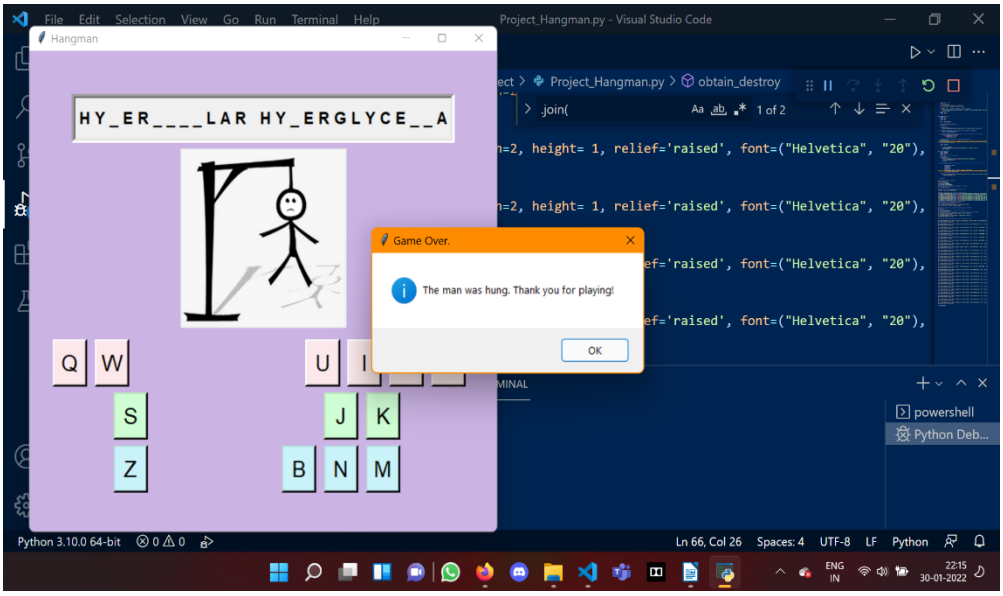
Player 2 clicks on the virtual keyboard, then the above messagebox is displayed.

All of these unique outputs can be displayed repeatedly in any permutation. The program will only terminate when Player 2 finishes the game.

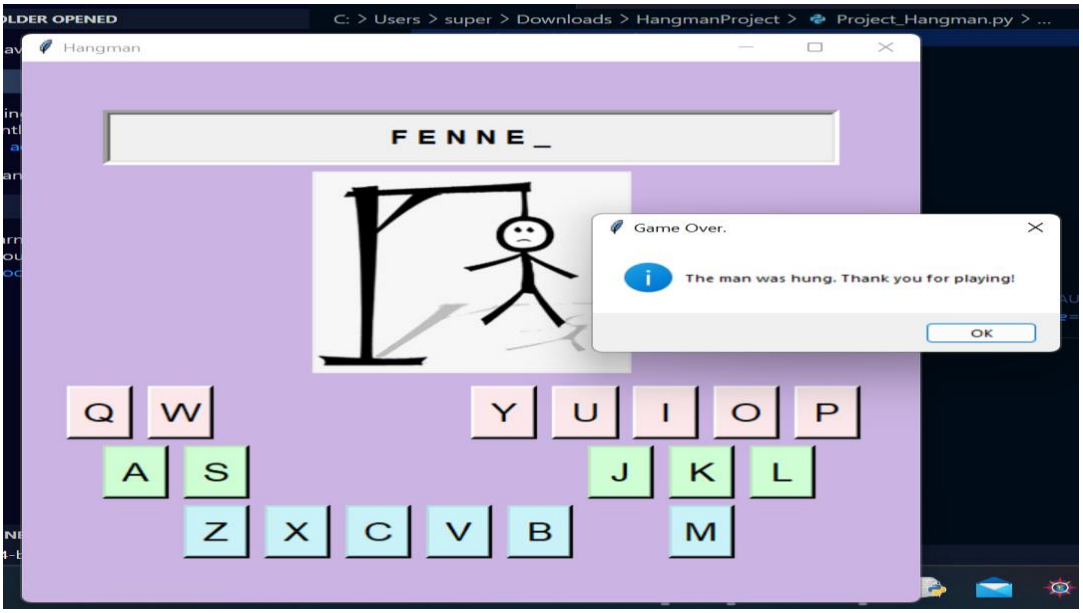
Result and Analysis

FAILED CASES:

Word 1: Hyperosmolar Hyperglycemia

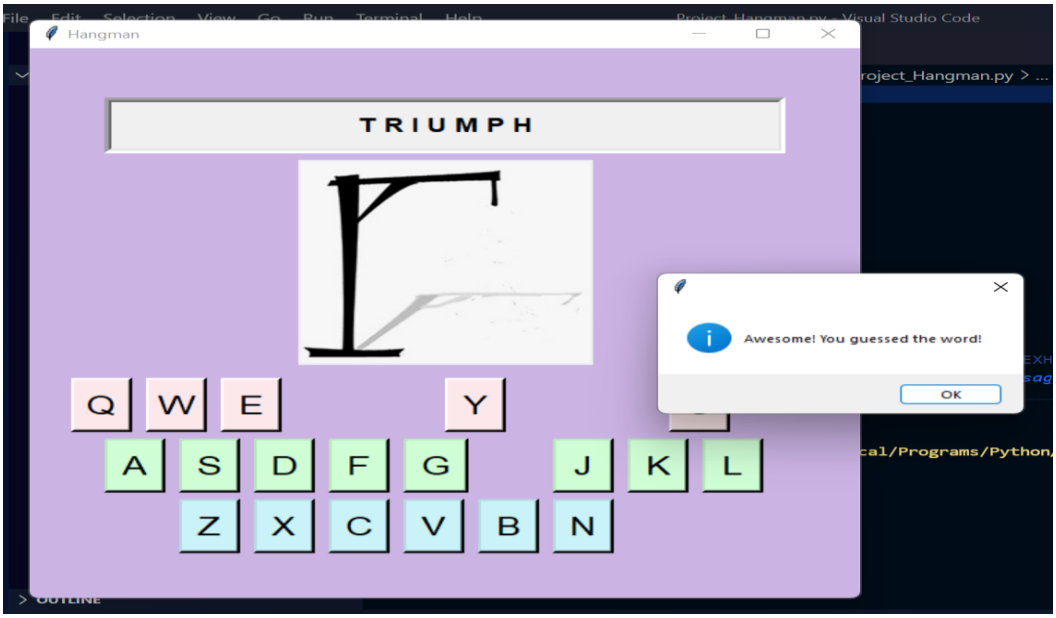


Word 2: Fennel

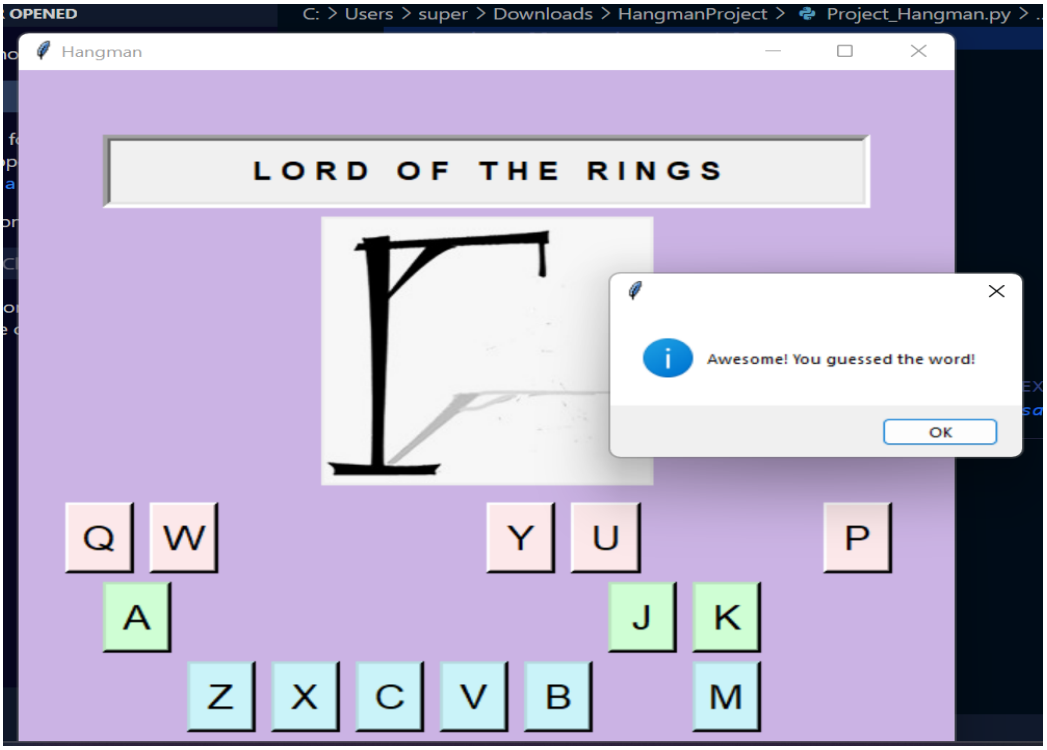


SUCCESS CASES:

Word 1: Triumph



Word 2: Lord of the Rings



Conclusions and Future Enhancements

1) Conclusions:

A) The label successfully displays inputs of practical sizes.

B) The program is able to handle and process extremely large inputs, however cannot display these outputs completely.

C) The compiler doesn't have any issues starting up the GUI, in spite of the code being nearly 200 lines long.

2) Future Enhancements:

A) Implementing a character length sensitive label such that it displays words of extremely large lengths.

B) Adding variable window functionality: the main window could automatically adapt to the player's desired window size.

C) Including the possibility for conducting multiple rounds of the game in a loop, without having to re-run the program.

References

1) The Python Standard Foundation, “Tkinter - Python interface to TCL/TK¶,” *tkinter - Python interface to Tcl/Tk - Python 3.8.12 documentation*. https://docs.python.org/3.8/library/tkinter.html.

2) A. Clark, *Pillow*. https://pillow.readthedocs.io/en/stable/.

3) The Python Software Foundation, “OS - miscellaneous operating system interfaces¶,” *os - Miscellaneous operating system interfaces - Python 3.8.12 documentation*. https://docs.python.org/3.8/library/os.html/.