Started on	Monday, 20 July 2020, 8:43 PM
State	Finished
Completed on	Monday, 20 July 2020, 11:42 PM
Time taken	2 hours 58 mins
Marks	1.00/1.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Lab Aims

The aim of this lab is to further familiarise you with Graphical User Interface (GUI) programming with Python/TkInter. The lab also introduces you to equipping software for cueing experimental tasks and logging user interaction.

At the end of this lab you should understand the basics of the following:

- cueing and logging experimental tasks for analysing user performance.

Building the Experimental Interface

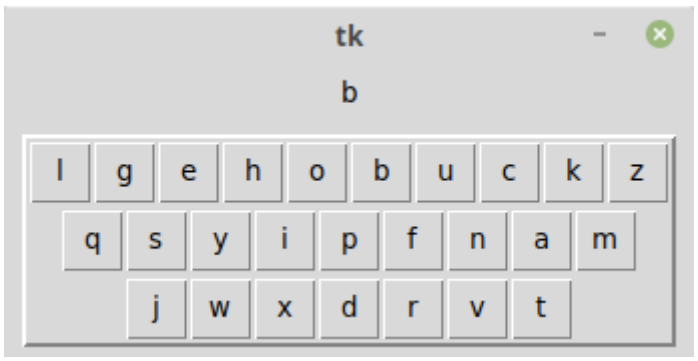
In Lab 1 you developed a simple keyboard UI that allows text to be entered into a Label by pressing buttons. In this lab, you will modify your program from Lab 1 to make an experimental interface.

The experimental interface will operate a bit differently: instead of entering text into the Label, the Label will be used to tell the user which key to press (and will cycle through a series of letters). The keys themselves will be laid out in one of two ways (detailed below).

You are encouraged to read through all the steps below before you begin rewriting your program so that you have a clear understanding of the interface you are making.

Re-arrange the keyboard interface

Make a new version of your keyboard program from Lab 1. Re-arrange the interface so it looks like the screenshot below (ignore the letters on the keys for now):



Resize the Buttons

Make sure that each button is square, with a width and height of 32 pixels. There is example of how to do this at http://effbot.org/tkinterbook/button.htm; search for the text "explicitly set the size". Note that this requires each button to be within its own frame, to which you set the size and propagate options as described in the link.

Cue a target letter and advance target on correct selection

Create a variable storing a string of target letters (e.g., 'abcdef'). Users will complete a set of n "blocks" (repetitions) of selections. (Your program should store a value for "n" and decrement it after each block is completed. Eventually n will be 6, but you may wish to use a lower number while you're developing your program.) Each block consists of one selection of each of the target letters (each letter is randomly selected from the set of targets remaining in the current block).

Once all targets have been selected for one block, the program should begin the next block. Once all blocks are completed, the program should terminate (preferably showing a completion message to the user).

Use the Label widget to show the next target letter. After each correct Button press, the program should advance to the next target.

Log the time taken to correctly select the item

Extend your program so it calculates the time (in milliseconds) from presentation of the cue to successful selection. If you import the time package, the function call time.time() returns a floating point description of the current clock time in seconds, so the following code calculates total time in milliseconds:

```
start = time.time()
doSomething()
total_time = (time.time() - start) * 1000
```

Write the selection time information to a log file, using the following format (the python csv module can be helpful for this: https://docs.python.org/3/library/csv.html#csv.writer):

Andy static a 1 2468.0

The first column shows your name, the second column shows the condition (which you will implement next), the third column shows the target character, the fourth is the block-count (the number of times this character has been shown to the user), and the last column shows the time taken to click key.

Randomise the keyboard layout

Adding the following capabilities may require some re-engineering of your program.

In Lab 6 you will be using your keyboard UI to run some human factors experiments. These studies will involve examining user performance in tasks involving visual search and spatial decisions. To facilitate these analyses your keyboard UI should support two styles of use:

- *Static:* which presents a randomized keyboard layout to users, but the keyboard remains static across selections.
- *Dynamic*: which randomizes the location of every character key after every selection.

Whether the program is running in *static* or *dynamic* mode should be configurable using a variable. When running in *static* mode user events should be written to the log file "experiment_static_log.txt"; and "experiment_dynamic_log.txt" when in *dynamic* mode.

Note that the Python module random includes a useful method shuffle that randomizes list content in place.

```
import random
some_list = ['a', 'b', 'c']
random.shuffle(some_list)
```

Configure the target set

The target set of characters (regardless of mode) used for the cueing mechanism implemented above should comprise six blocks (repetitions) of the six randomly selected letters of the alphabet, using a random order of letters in each block.

For example, when the program runs, the letters 'ahduef' may be randomly selected, and the six blocks may consist of the following:

Block 1: d, u, e, f, a, h

Block 2: u, f, h, a, d, e

...

Block 6: a, u, e, d, f, h

Question **1**

Complete

Mark 1.00 out of 1.00

Submit the Keyboard program

After completing this lab, you should have an experimental program that has the following features:

- Can display a keyboard that is laid out randomly, and either stays *static* for the duration of the experiment, or *dynamically* changes after each successful trial.
- Prompts the user to select six unique letters of the alphabet a total of 36 times.
- Requires the user to select each letter successfully before advancing to the next trial.
- Logs information about the selection to a file.

Submit the program here.

```
# Author: Bach Vu
# Window Class


from tkinter import *
import tkinter.ttk as ttk
from random import *
from time import time
import csv

class TestWindow(Tk):
    def __init__(self, title):
        root = Tk()
        root.title(title)
        self.window = Frame(root)
        self.window.pack(padx=10, pady=10)

    def mainloop(self):
        self.window.mainloop()

class KeyboardWindow(TestWindow):
    """ To test faster, delete some block from return in testData()
        Switch to static/dynamic keyboard in __init__ IS_STATIc
    """
    def __init__(self, title):
        super().__init__(title)

        # Form data
        self.IS_STATIC = False
        self.start = time()
        self.total_time = None

        self.stringBlocks = self.testData()
        self.index = IntVar()
        self.index.set(0)

        self.board = ['qwertyuiop', 'asdfghjkl', 'zxcvbnm']
        self.data = StringVar()

        # GUI Widgets
        self.lblTarget = Label(self.window, textvariable=self.data, anchor=CENTER, justify=CENTER, background='red')
        self.lblTarget.pack(fill=X, pady=5)
```

```
        self.dynamicKeyboard = Frame(self.window, borderwidth=4, relief=RIDGE)
        self.dynamicKeyboard.pack(fill=X)

        self.randomKeyboard()
        self.displayTarget()

    def testData(self):
        block1 = StringVar()
        block1.set("iemnfs")
        block2 = StringVar()
        block2.set("oewfdv")
        block3 = StringVar()
        block3.set("ophfsc")
        block4 = StringVar()
        block4.set("kjldnv")
        block5 = StringVar()
        block5.set("sdfaef")
        block6 = StringVar()
        block6.set("mnasbv")
        return [block1,block2,block3,block4,block5,block6]

    def displayTarget(self):
        if not self.IS_STATIC:
            self.randomKeyboard()
        if self.stringBlocks[self.index.get()].get() == "":
            self.index.set(self.index.get()+1)
            if self.index.get() >= len(self.stringBlocks):
                self.complete()
                return

        block = self.stringBlocks[self.index.get()].get()
        print(block)
        ch = block[randint(0,len(block)-1)]
        self.data.set(ch)

    def keydown(self, ch):
        self.total_time = (time() - self.start) * 1000 # ms

        if ch != self.data.get():
            return
        block = self.stringBlocks[self.index.get()]
        block.set(block.get().replace(ch,'',1))
        self.displayTarget()

        self.start = time()
        if self.IS_STATIC:
            self.write_CSV("Bach","static",ch,self.total_time)
        else:
            self.write_CSV("Bach","dynamic",ch,self.total_time)
```

```
def randomKeyboard(self):
    for i in range(len(self.board)):
        chars = list(self.board[i])
        shuffle(chars)
        self.board[i] = ''.join(chars)

    for child in self.dynamicKeyboard.winfo_children():
        child.destroy()
    for i in range(len(self.board)):
        row = self.board[i]
        for j in range(len(row)):
            f = Frame(self.dynamicKeyboard, height=32, width=32)
            f.pack_propagate(0) # don't shrink
            f.grid(row=i, column=i+j*2, columnspan=2, padx=2, pady=2)

            ch = row[j]
            btn = Button(f, text=ch, width=2,
                         command=lambda x=ch: self.keydown(x))
            btn.pack(fill=BOTH, expand=1)

    def complete(self):
        self.data.set("Congrates! You completed the quiz.")

    def write_CSV(self, *param):
        if self.IS_STATIC:
            filename = 'experiment_static_log.txt'
        else:
            filename = 'experiment_dynamic_log.txt'
        with open(filename, mode='a') as record_file:
            csv_writer = csv.writer(record_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_NONE)
            csv_writer.writerow(param)

def ques1():
    window = KeyboardWindow("Keyboard Random")
    window.mainloop()

if __name__ == "__main__":
    ques1()
```

Comment:

◄ Lab 1: Graphical User Interface Programming with Python/TkInter

Jump to...

Lab 3: Python TkInter Canvas Widget (and Fitts' Law Experimental UI) ►