Dashboard / My courses / COSC368-20S2 / Sections / Labs / Lab 3: Python TkInter Canvas Widget (and Fitts' Law Experimental UI)

Started on	Tuesday, 28 July 2020, 7:51 AM
State	Finished
Completed on	Tuesday, 28 July 2020, 10:19 AM
Time taken	2 hours 28 mins
Marks	1.00/1.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Aims -- the Canvas Widget

This is the last of three labs aiming to familiarise you with GUI programming using Python/TkInter. This lab focuses on the powerful Canvas widget, which is particularly useful for drawing interactive elements within a window and for rapidly prototyping new types of interactive widget. You will learn the canvas widget by building an experimental interface for conducting Fitts' Law experiments.

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

- the TkInter Canvas widget; and
- the interface used for examining one-dimensional Fitts' Law experiments.

The Canvas Widget

There is a good overview of the Canvas widget at: http://effbot.org/tkinterbook/canvas.htm

The Canvas widget is important because of its versatility: it can be used to draw graphs and plots, implement custom widgets, and prototype/simulate interactive systems.

Try the following self-explanatory program:

```
from tkinter import *
from tkinter.ttk import *
master = Tk()
c = Canvas(master, width=200, height=200)
c.pack()
c.create_line(0, 0, 200, 100)
c.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))
c.create_rectangle(50, 25, 150, 75, fill="blue")
master.mainloop()
```

Types of canvas items

The Canvas widget supports several types of items. The program above demonstrates *line* and *rectangle* items; the others are *arc*, *bitmap*, *image*, *line*, *oval*, *polygon, rectangle, text,* and *window*. Window items are used to place other TkInter widgets within a Canvas widget. They are all created using Canvas methods equivalent to the line and rectangle creation statements above (e.g., w.create_arc, etc.)

Keeping track of canvas items or groups of items

How do you keep track of items in order to manipulate or modify them? There are two ways. First, each item created on the canvas has a unique integer identifier that is the return value of its create_ method. The first item created receives the identifier 1, the second 2, etc.

Second, you can explicitly assign one or more string tags to any canvas item. Groups of items can be created by assigning the same tag to a series of items.

Modifying canvas object items

You can configure any of the properties of items by using the itemconfigure method (and stipulating their identifier or tag).

Modify the program above by stipulating a tag 'cool' for both of the line items (pass in a parameter tag='cool' on lines 6 and 7) and change line 8 so that the result of the rectangle creation is assigned to a variable, as follows:

```
rect = c.create_rectangle(50, 25, 150, 75, fill="blue")
```

Then add the following two item configuration statements before the main loop:

```
c.itemconfigure('cool', fill='blue')
c.itemconfigure(rect, fill='red')
```

Note that both of the lines tagged 'cool' change blue, and the uniquely identified rectangle changes red.

Some items provide specific methods for common manipulations. For instance, the coordinates of a rectangle item tagged 'rect' might be stipulated using the following statement:

```
c.coords('rect', 10, 10, 50, 100)
```

Binding to items

Function calls can be bound to specific items (or groups) using similar methods to item configuration.

For example, the following statements bind function calls to a Canvas item named "w". In both cases the functions receive one parameter describing the event:

```
w.tag_bind(rect, "<ButtonPress-1>", foo)
w.tag_bind('cool', "<ButtonPress-1>", bar)
```

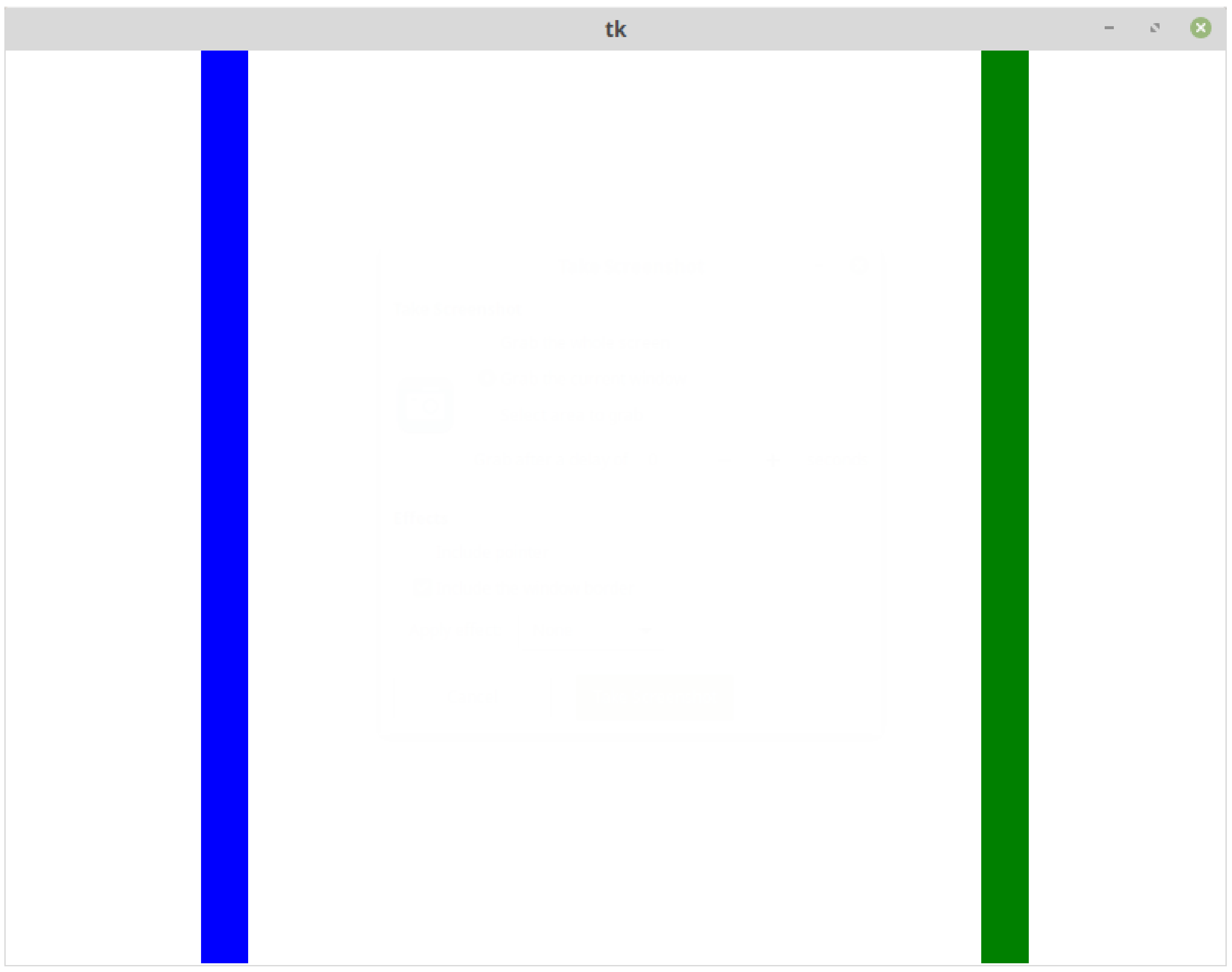
Question **1**

Complete

Mark 1.00 out of 1.00

Fitts' Law UI

In upcoming labs you will investigate human pointing performance, and these results will be used in to help you measure human performance in visual search and decision tasks. The Canvas widget is very useful for creating simple experimental interfaces. To prepare for the lab in week 5, you need to write a program that looks and behaves as described below.



Your experimental system will be used to control a series of one-dimensional pointing tasks (horizontal pointing only), which is similar to Fitts' original 1954 experiment (although he used electrified plates). Users move the cursor as quickly as possible to the green vertical bar and click on it. Clicking on the green bar causes the bars to switch colours (green turning blue, and vice versa), so users move "as quickly and accurately as possible" between clicks on the two alternating targets. Both targets always have the same width.

The width of the targets and the distance between the centers of the targets is controlled by the software. Your program should use two lists to control these settings; something like the following:

```
distances = [64, 128, 256, 512]
widths = [4, 8, 16, 32]
```

Your program should ensure that all combinations of distance and width are administered in a random order. Furthermore, for each combination of distance and width, the user should execute an even number of task repetitions at that setting: e.g., if the number of repetitions is four, the participants tap on the right, then left, then right, then left targets, after which the software moves on to the next distance and width combination.

Finally, your program must log the time taken between showing the next green target and it being successfully clicked on. Each line of the log file should have the following format:

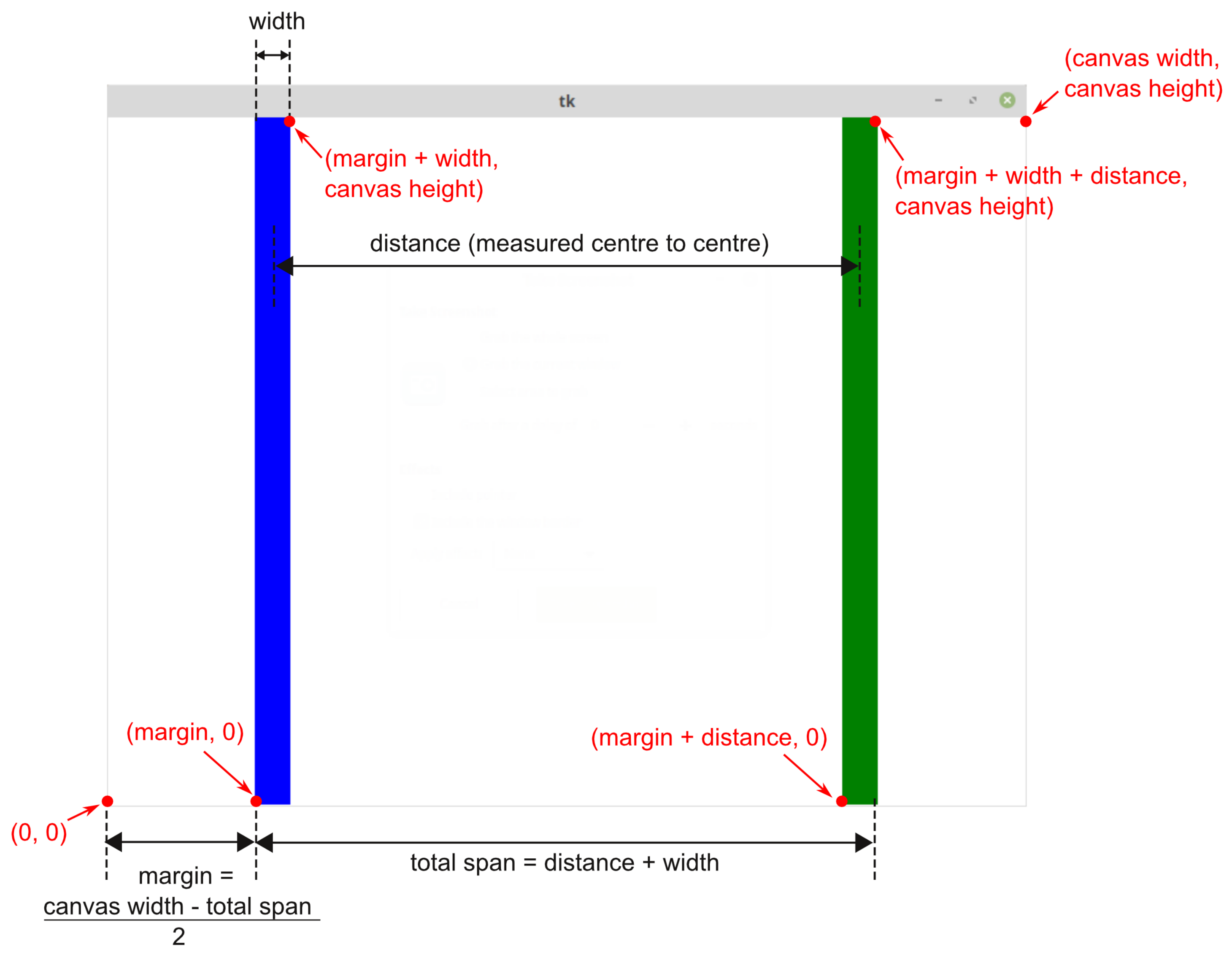
[Name] [distance] [width] [selection number] [time]

where "selection number" ranges from 0 to the number of repetitions at each combination of distance and width. For example,

Andy 256 32 1 0.6

The python csv module can be useful here (https://docs.python.org/3/library/csv.html#csv.writer)

The following diagram may help you with laying out elements on your Canvas.


width
(canvas width, canvas height)
tk
(margin + width, canvas height)
(margin + width + distance, canvas height)
distance (measured centre to centre)
(margin, 0)
(margin + distance, 0)
(0, 0)
margin = canvas width - total span / 2
total span = distance + width

Submit the program here.

```
# main.py
# Author: Bach Vu
# Window Class
from tkinter import *
from tkinter.ttk import *
from random import randint
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 CanvasWindow(TestWindow):
    def __init__(self, title):
        super().__init__(title)
        self.distances = [64, 128, 256, 512]
        self.distance = IntVar()
        self.widths = [4, 8, 16, 32]
        self.width = IntVar()
        self.score = {}
        self.REPEAT = 4
        self.start = time()

        self.C_WIDTH, self.C_HEIGHT = 600, 600
        self.canvas = Canvas(self.window, width=self.C_HEIGHT, height=self.C_HEIGHT, bg="red")
        self.canvas.pack()

        self.left_rect, self.right_rect = None, None
        self.updateTarget()

        self.canvas.tag_bind(self.left_rect, "<ButtonPress-1>", self.leftClickObj)
        self.canvas.tag_bind(self.right_rect, "<ButtonPress-1>", self.rightClickObj)

    def updateTarget(self, reverse=True):
        """ Change config of pillars """
        distance, width = self.randomizeConfig()
        total_span = distance + width
        x1_left = (self.C_WIDTH - total_span)/2
        x2_left = x1_left + width
        x2_right = x1_left + total_span
        x1_right = x2_right - width

        if self.right_rect == None:
            self.left_rect = self.canvas.create_rectangle(x1_left, 0, x2_left, self.C_HEIGHT, fill="blue",
state="disabled")
            self.right_rect = self.canvas.create_rectangle(x1_right, 0, x2_right, self.C_HEIGHT,
fill="green")
        else:
            if reverse:
                self.canvas.itemconfigure(self.left_rect, fill="green")
                self.canvas.itemconfigure(self.left_rect, state="normal")
                self.canvas.itemconfigure(self.right_rect, fill="blue")
                self.canvas.itemconfigure(self.right_rect, state="disabled")
            else:
                self.canvas.itemconfigure(self.left_rect, fill="blue")
                self.canvas.itemconfigure(self.left_rect, state="disabled")
                self.canvas.itemconfigure(self.right_rect, fill="green")
                self.canvas.itemconfigure(self.right_rect, state="normal")
            self.canvas.coords(self.left_rect, x1_left, 0, x2_left, self.C_HEIGHT)
            self.canvas.coords(self.right_rect, x1_right, 0, x2_right, self.C_HEIGHT)

    def randomizeConfig(self):
        i_dist = self.distance.get()
        i_width = self.width.get()
        while self.score.get((i_dist, i_width), 0) >= self.REPEAT:
            # set even number of task repetitions at that setting
            self.distance.set(randint(0, len(self.distances)-1))
            self.width.set(randint(0, len(self.widths)-1))
            i_dist = self.distance.get()
            i_width = self.width.get()

        self.score[(i_dist, i_width)] = self.score.get((i_dist, i_width), 0) + 1
        return self.distances[i_dist], self.widths[i_width]


    def leftClickObj(self, event):
        self.createLogRecord()
        self.updateTarget(False)
        self.start = time()

    def rightClickObj(self, event):
        self.createLogRecord()
        self.updateTarget(True)
        self.start = time()

    def createLogRecord(self):
        total_time = (time() - self.start) * 1000
        i_dist = self.distance.get()
        i_width = self.width.get()
        self.write_CSV("Bach", self.distances[i_dist], self.widths[i_width],
                       self.score[(i_dist, i_width)], total_time)

    def write_CSV(self, mode, *param):
        filename = 'sample_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)

if __name__ == "__main__":
    window = CanvasWindow("Canvas")
    window.mainloop()
```

Comment:

◄ Lab 2: GUI Programming with Python/TkInter (II)

Jump to...

Lab 4: Conducting a Fitts' Law Experiment ►