

Criterion C: Development

Table of contents:

SQLite	2
Tkinter	2
Buttons	2
Labels	2
Checkbuttons	3
Entry	3
Grid	4
Main window	4
Randomization	5
Error messages	5
Table output	6
Version division	6
Top bans	7
Civilization description, cursor hovering	8
Adding civilizations	9

SQLite

The database engine for my code will be SQLite because it is an in-process library which is much more convenient, than SQL for my small application, also it is supported nearly by all OS. This will help me to succeed in my 11 success criteria, so I connect to it:¹

```
import sqlite3
from os import startfile
```

Tkinter

Tkinter is a Python library that is used to create the GUI, to use its widgets like buttons or labels I am connecting to it:²

```
from tkinter import *
root = Tk()
```

Buttons

I am using the Tkinter widget “buttons” to create buttons, by width and height I regulate their size, through it I will fulfill 11 success criteria. To access the attributes of an object I use “config” after it is initialized:

```
b1 = Button(text="Draft",
            width=15, height=3)
b1.config(command=lambda:get_data(civpicks,data,Nplayers,Ncivs))
base = Button(text="Base",
              width=15, height=3)
base.config(command=lambda:new_random(1))
RiseFall = Button(text="Rise&Fall",
                  width=15, height=3)
RiseFall.config(command=lambda:new_random(2))
GatheringStorm = Button(text="Gathering Storm",
                        width=15, height=3)
GatheringStorm.config(command=lambda:new_random(3))
DLC = Button(text="DLC",
             width=15, height=3)
DLC.config(command=lambda:new_random(4))
```

Labels

Another widget that I use from the Tkinter is “label” for the displacement of text on the window, using it I to outline main input entries, thus fulfilling 11 success criteria:

¹ <https://www.sqlite.org/index.html>

² <https://docs.python.org/3/library/tkinter.html#module-tkinter>

```

l1 = Label(root,text="Number of players",
            font="Arial 32")

l2 = Label(root,text="Number of civilizations",
            font="Arial 32")

```

Checkbuttons

Users need to ban the civilizations and this can be done through Tkinter widget - “Checkbuttons”, it is simple and intuitive because clients just need to switch civilization, it is either banned or not. Because of its simplicity I am fulfilling the 11 success criteria.

The location of the civilization in the main window is made by simple python instructions like “for i in range”, “if” and “else”, I have chosen only 3 civilization on each row:

```

civpicks=[]
c=-1
r=2
chbtns = []
for i in range (len(data)):
    if i%3==0:
        r+=1
        c=0
    else:
        c+=1
    var1 = BooleanVar()
    var1.set(0)
    c1 = Checkbutton(root, text=data[i],
                    variable=var1,
                    onvalue=1, offvalue=0)

    button1_ttp = CreateToolTip(c1, desc[i])
    civpicks.append(var1)
    c1.grid(row=r, column=c)
    chbtns.append(c1)

```

Entry

Another Tkinter widget that I am using for creating the main window is the “entry” for the client to input data about number of civilizations and number of players, I will also use it for the adding civilization feature, fulfilling 4 and 5 criteria:

```

Nplayers = Entry(width=50)
Ncivs = Entry(width=50)
Nwciv = Entry(root3,width=50)
Nwdesc = Entry(root3,width=50)

```


Grid

To locate the buttons and input fields in the main window I use the “grid” function because it is simply to apply:

```
b1.grid(row=1, column=1)
Nplayers.grid(row=2, column=0)
Ncivs.grid(row=2, column=2)
l1.grid(row=0, column=0)
l2.grid(row=0, column=2)
base.grid(row=1, column=4)
RiseFall.grid(row=2, column=4)
GatheringStorm.grid(row=3, column=4)
DLC.grid(row=4, column=4)
add.grid(row=5, column=4)
top_bans.grid(row=7, column=4)
```

Main window

Finally, this is how my main window will look like:

 Civilization time saver

Reference

Number of players

☐ American - Teddy Roosevelt

☐ Aztec - Montezuma

☐ Byzantine - Basil II

☐ Dutch - Wilhelmina

☐ Ethiopian - Menelik II

☐ Georgian - Tamar

☐ Greek - Gorgo

☐ Incan - Pachacuti

☐ Indonesian - Gitarja

☐ Kongolese - Mvemba a Nzinga

☐ Malian - Mansa Musa

☐ Mongolian - Genghis Khan

☐ Nubian - Amanitore

☐ Phoenician - Dido

☐ Russian - Peter

☐ Spanish - Philip II

☐ Zulu - Shaka

☐ Arabian - Saladin

☐ Babylonian - Hammurabi

☐ Canadian - Wilfrid Laurier

☐ Egyptian - Cleopatra

☐ French - Catherine de Medici

☐ German - Frederick Barbarossa

☐ Greek - Pericles

☐ Indian - Chandragupta

☐ Japanese - Hojo Tokimune

☐ Korean - Seondeok

☐ Mapuche - Lautaro

☐ Māori - Kupe

☐ Ottoman - Suleiman

☐ Polish - Jadwiga

☐ Scottish - Robert the Bruce

☐ Sumerian - Gilgamesh

☐ Australian - John Curtin

☐ Brazilian - Pedro II

☐ Cree - Poundmaker

☐ English - Victoria

☐ Gallic - Ambiorix

☐ Gran Colombian - Simón Bolívar

☐ Hungarian - Matthias Corvinus

☐ Indian - Gandhi

☐ Khmer - Jayavarman VII

☐ Macedonian - Macedonian

☐ Mayan - Lady Six Sky

☐ Norwegian - Harald Hardrada

☐ Persian - Cyrus

☐ Roman - Trajan

☐ Scythian - Tomyris

☐ Swedish - Kristina

Draft

Base

Rise&Fall

Gathering Storm

DLC

Add civilization

Top Bans

Randomization

After I separate set of all my civilizations to bans and nonbans I use simple “random.shuffle” instruction which python has to randomize my civilizations that have lasted:

```
def get_data(civpicks,data,Nplayers,Ncivs):
    bans=[]
    nonbans=[]
    with sqlite3.connect('Mark.db') as connection:
        cursor = connection.cursor()
        for i in range(len(civpicks)):
            if civpicks[i].get()==True:
                bans.append (data[i])
                cursor.execute("insert into bans (name) values(?)", (data[i],))
            else:
                nonbans.append (data[i])
        random.shuffle(nonbans)
```

Error messages

The error messages are inserted after the random shuffle, I use simple “if” and “elif”. I use another tkinter widget “mb.showerror” to display the error text, thus I am fulfilling 6 success criteria:

```
if Nplayers.get() == '':
    mb.showerror(
        "Error",
        "The number should be written. If you can't solve problem individually  
you can write to my email and I will help you: iahogyeu@gmail.com")
elif Ncivs.get() == '':
    mb.showerror(
        "Error",
        "The number should be written. If you can't solve problem individually  
you can write to my email and I will help you: iahogyeu@gmail.com")
elif Ncivs.get().isdecimal():
    mb.showerror(
        "Error",
        "The number should be written. If you can't solve problem individually  
you can write to my email and I will help you: iahogyeu@gmail.com")
elif Nplayers.get().isdecimal():
    mb.showerror(
        "Error",
        "The number should be written. If you can't solve problem individually  
you can write to my email and I will help you: iahogyeu@gmail.com")
else:
    Ncivs2=int(Ncivs.get())
    Nplayers2=int(Nplayers.get())
```

```

if Ncivs2*Nplayers2>50:
    mb.showerror(
        "Error",
        "The number should be written. If you can't solve problem individually
you can write to my email and I will help you: iahogyeu@gmail.com")

```

Table output

If none of the errors occur the table is created. It is connecting to the “draft.txt” and output the randomly distributed table, therefore fulfilling 7 success criteria:

```

else:
    draft=f"N_players: {Nplayers2}\n N_cv: {Ncivs2}\n"
    s = 0
    e = Ncivs2
    for i in range (Nplayers2):
        draft += f'Player_{i+1}: {nonbans[s:e]}\n'
        s = e
        e += Ncivs2
    with open("draft.txt", "w", encoding="utf-8") as f:
        f.write(draft)
    startfile("draft.txt")

```

Version division

I have divided civilizations by the version and gave a unique number to each. This gave me the ability easily to change the civilizations in the main window by “if” and “elif”. I use “cursor.fetchall” which will return the list of tuples, I prefer them more than strings, even though they are nearly the same. Hence, I am fulfilling 1 success criteria:

```

global chbtms, data,civpicks
for c in chbtms:
    c.grid_forget()
with sqlite3.connect('Mark.db') as connection:
    cursor = connection.cursor()
    if version == 1:
        cursor.execute("select * from civ where vers = {} order by
name".format(version))
    elif version == 2:
        cursor.execute("select * from civ where vers = {} or vers={} order by
name".format(1,version))
    elif version == 3:
        cursor.execute("select * from civ where vers = {} or vers={} or vers={}
order by name".format(1,2,version))
    elif version == 4:
        cursor.execute("select * from civ where vers = {} or vers={} or vers={}
or vers={} order by name".format(1,2,3,version))

```

```

data = [row[1] for row in cursor.fetchall()]
if version == 1:
    cursor.execute("select * from civ where vers = {} order by
name".format(version))
elif version == 2:
    cursor.execute("select * from civ where vers = {} or vers={} order by
name".format(1,version))
elif version == 3:
    cursor.execute("select * from civ where vers = {} or vers={} or vers={}
order by name".format(1,2,version))
elif version == 4:
    cursor.execute("select * from civ where vers = {} or vers={} or vers={}
or vers={} order by name".format(1,2,3,version))
desc = [row[2] for row in cursor.fetchall()]

```

Top bans

To create the top bans table I use the function `get_bans`. Firstly, it connects to the base:

```

def get_bans():
    with sqlite3.connect('Mark.db') as connection:
        cursor = connection.cursor()

```

Then, program counts how many bans overall were made for each civilization:

```

cursor.execute("select name, count(*) from bans group by name order by count(*)
desc")

```

Finally, I create table and special window and put the text into it, fulfilling 9 criteria:

```

text = ''
for row in cursor.fetchall():
    text+=f"{row[0]}: {row[1]}\n"
root2 = Tk()
Label(root2, text=text).pack()
root2.mainloop()

```

Civilization description, cursor hovering

Not to break the flow of the user and still show the descriptions of the civilizations I use the tooltip have also created tooltip. Main idea here is that by "bind" when the mouse hovers to the civilization it will be "<Enter>" and in this case "self.enter" is used. When the mouse leaves civilization, name the "bind" "<Leave>" will activate and run the "self.close". Therefore fulfilling 3 success criteria:

```
class CreateToolTip(object):  
    '''  
    create a tooltip for a given widget  
    '''  
    def __init__(self, widget, text='widget info'):  
        self.widget = widget  
        self.text = text  
        self.widget.bind("<Enter>", self.enter)  
        self.widget.bind("<Leave>", self.close)
```

The function "enter" will input the small window in near the civilization name, it will be dark green with white text:

```
def enter(self, event=None):  
    x = y = 0  
    x, y, cx, cy = self.widget.bbox("insert")  
    x += self.widget.winfo_rootx() + 25  
    y += self.widget.winfo_rooty() + 20  
    self.tw = Toplevel(self.widget)  
    self.tw.wm_overrideredirect(True)  
    self.tw.wm_geometry("+%d+%d" % (x, y))  
    label = Text(self.tw, width=50, height=10, bg="darkgreen", fg='white', wrap=WORD)  
    font=('times', 3)  
    label.insert(1.0, self.text)  
    label.pack(ipadx=1)
```

The function "close" will just close, or "destroy" the small green window which has been shown (I have taken this code from the open sources³).

```
def close(self, event=None):  
    if self.tw:  
        self.tw.destroy()
```


Adding civilizations

To fulfill 8 success criteria I add civilizations to the last version. I first of all create a special window for this procedure. I do it through using “entry” “pack” and “label” Tkinter widgets:

```
def add_civ():
    root3 = Tk()
    Nwciv = Entry(root3,width=50)
    Nwdesc = Entry(root3,width=50)
    l4=Label(root3, text="Enter new civilization")
    l4.pack()
    Nwciv.pack()
    l3=Label(root3, text="Enter description")
    l3.pack()
    Nwdesc.pack()
    upload = Button(root3,text="Upload",
                    width=15, height=3)
```

Afterwards, the main function “upload_civ” is made by simple “cursor.execute” and “upload.config”:

```
def upload_civ():
    print(Nwciv.get(), Nwdesc.get())
    with sqlite3.connect('Mark.db') as connection:
        cursor = connection.cursor()
        cursor.execute("insert into civ (id,name,description,vers) values
(?,?,?,?)", (0, Nwciv.get(), Nwdesc.get(), 4))
        root3.destroy()
    upload.config(command=upload_civ)
    upload.pack()
```

For the start of the program the CSIA.py file should be launched.

Bibliography:

1. vegaseat. «A Tooltip Class for Tkinter». DaniWeb,
<https://www.daniweb.com/programming/software-development/code/484591/a-tooltip-class-for-tkinter>. Seen 15 december 2020

word count: 766