PROJECT 3: PROGRAMMABLE DRUM MACHINE

In this project, we will build a programmable drum machine. The GUI of the drum machine is based on TKinter. You will be able to create an unlimited no. of beat patterns using an unlimited no. of drum samples. You can then store multiple riffs in a project and playback or edit the project later on.

To create your own drum beat patterns, simply load some drum samples using the buttons on the left. You can change the units that constitute a beat pattern, which in turn decides the tempo of the rhythm. You can also decide the no. of beats per unit.

DOWNLOAD PYMEDIA MODULE

We will make one using OOP approach.

PTO

classmate

STEP ONE → PREPARE FOR LIFT-OFF

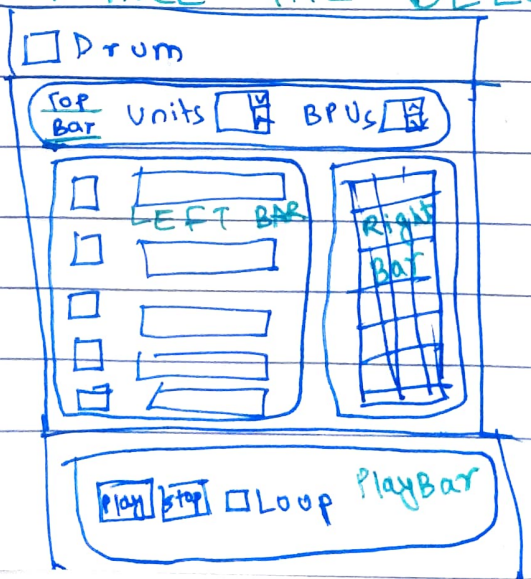
An indicative OOP structure for our drum program could be as follows:

```
from tkinter import *
class DrumMachine():
    def app(self):
        self.root = Tk()
        #all other code are call from here
        self.root.mainloop()
```

The description of the code is -

- We create a class called DrumMachine and define a method app() to initialize the Toplevel window.
- If the program is run as a standalone program, a new object is created and the app method is called to create the Toplevel window.
- This code creates a blank Toplevel window.

STEP TWO - MAKE THE BELOW ONE



History of the Notebook

Imp config is used to access an ~~attribute's~~ object's attributes after its initialisation. For example, here you define l = Label (root, bg='ivory', fg="dark green") but then u want to set its text attribute, so u use config & l.config (text="Correct Answer!") Engage Thrusters

1. First, we will create the top bar. The top bar is one that holds the spinbox widgets, which lets the user change the units and beats ~~as~~ per unit in a rhythm pattern. These two together decide the tempo and the cyclical pattern of a rhythm as follows:

```
def create_top_bar(self):
1/  top_bar_frame = Frame(self.root)
    top_bar_frame.config(height=25)        rows=20
    top_bar_frame.grid(row=0, columnspan=12,   columns=22
    rowspan=10, padx=5, pady=5)
2/  Label(top_bar_frame, text='Units:').grid
    (row=0, column=4)
    self.units = IntVar()
    self.units.set(4)
    self.bpu units_widget= Spinbox(top_bar_frame,
    from_=1, to=8, width=5, textvariable=
    self.units)
    self.units_widget.grid(row=0, column=7)
3/  Label(top_bar_frame, text="BPUs")
    .grid(row=0, column=6)
    self.bpu = IntVar()
    self.bpu.set(4)
    self.bpu_widget = Spinbox(top_bar_frame,
    from_=1, to=10, width=5,
    textvariable=self.bpu)
    self.bpu_widget.grid(row=0, column=7)
```

classmate

Impl config is used to access an ~~attribute's~~ object's attributes after its initialisation. For example, here you define l=Label(root, bg='ivory', fg="dark green") but then u want to set its text attribute, so u use config ⬇ l.config (text="Correct Answer!") Engage Thrusters during runtime. that way you can set the text, and modify it

1. First, we will create the top bar. The top bar is one that holds the Spinbox widgets, which lets the user change the units and beats ~~as~~ per unit in a rhythm pattern. These two together decide the tempo and the cyclical pattern of a rhythm as follows:

```
def create_top_bar(self):
1/  top_bar_frame = Frame(self.root)
    top_bar_frame.config(height=25)
    top_bar_frame.grid(row=0, columnspan=12,
    rowspan=10, padx=5, pady=5)
2/  Label(top_bar_frame, text='Units:').grid
    (row=0, column=4)
    self.units = IntVar()
    self.units.set(4)
    self.bpu units_widget= Spinbox(top_bar_frame,
    from_=1, to=8, width=5, textvariable=
    self.units)
    self.units_widget.grid(row=0, column=7)
3/  Label(top_bar_frame, text="BPUs")
    .grid(row=0, column=6)
    self.bpu = IntVar()
    self.bpu.set(4)
    self.bpu_widget = Spinbox(top_bar_frame,
    from_=1, to=10, width=5,
    textvariable=self.bpu)
    self.bpu_widget.grid(row=0, column=7)
```

rows=20
columns=22

classmate

2. Next we will create the left bar. The left bar is one that will let the user load drum samples

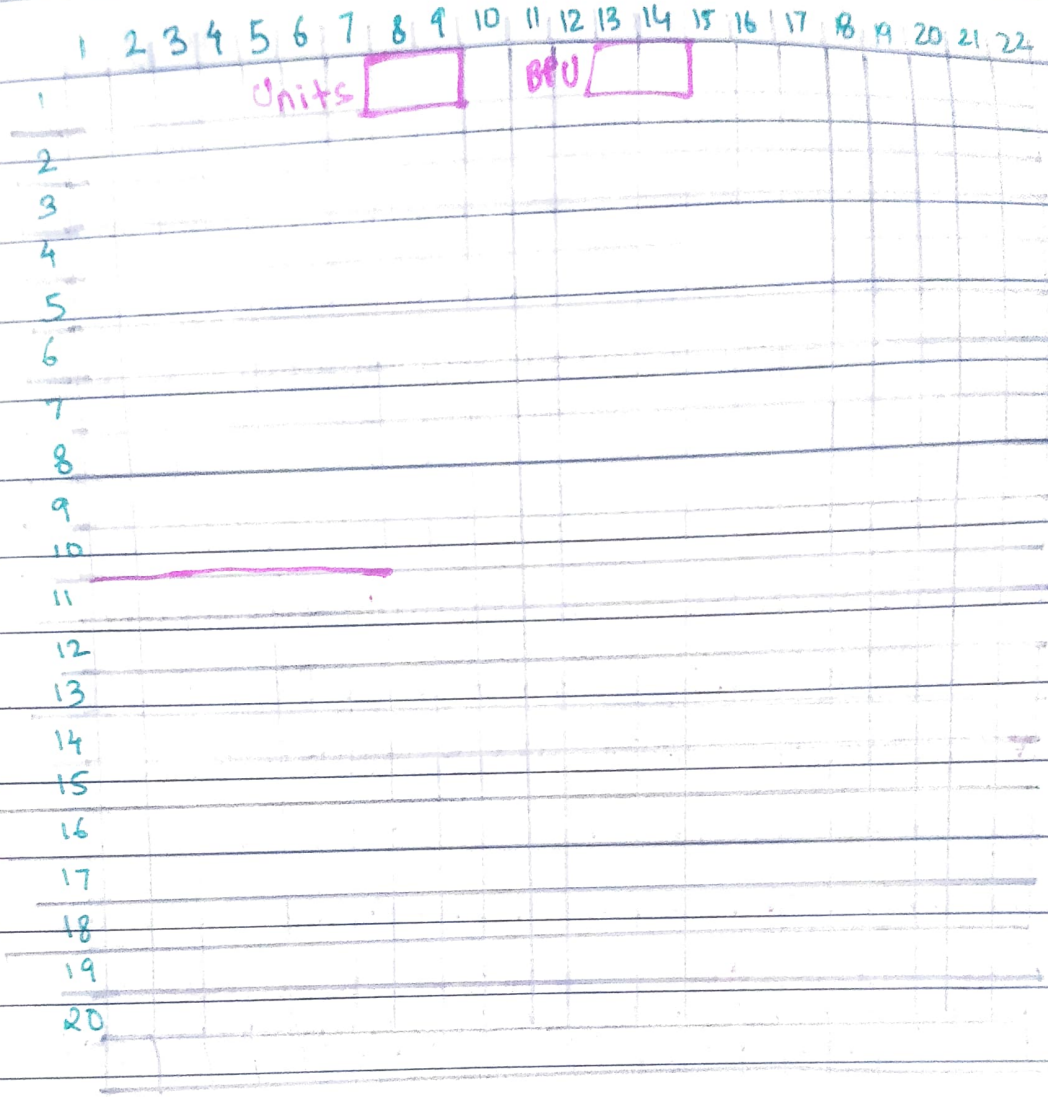
The buttons on the left opens

The description of the code is lifted below as:

- We first create a new method in order to create the top bar. We add a frame top_bar_frame for the top bar and add two spinboxes to keep track of the units and the BPUs. We did not add callbacks now. The callbacks will be added later.
- We define two Tkinter variables self.units and self.bpu to hold the current value of both the Spinbox widgets. This is defined as an object variable (self) because we will need these variables outside the scope of this method.
- The widgets are placed using the grid geometry manager.

2. Next, we will create the left bar. The left bar is one that will let the user load drum samples.

The buttons on the left bar will open an upload file. When the user uploads a drum sample, the



name of the drum sample will automatically populate the Entry widget adjacent to that button.

~~Engage Thrusters~~

```
MAX_DRUM_NUM = 5

def create_left_pad(self):
        left_frame = Frame(self.root)
        left_frame.grid(row=10, column=0,
        columnspan=6, sticky=W+E+N+S)
        tbicon = PhotoImage(file="")
```

```
for i in range(0, MAX_DRUM_NUM):
    button = Button(left_frame, image=tb_icon)
    button.image = tb_icon
    button.grid(row=i, column=0, padx=5, pady=2)
    self.drum_entry = Entry(left_frame)
    self.drum_entry.grid(row=i, column=4, padx=7,
                         pady=2)
```

3/ Next, we will create the right bar. The right bar is the area that lets the user define the beat pattern. This area consists of a series of buttons. The no. of row of buttons is equal to the drum samples that can be loaded. The no. of columns of buttons is equal to the product of the no. of units and beats per unit.

We are not connecting the spinboxes with the buttons right now. For now, let us place buttons in four columns for each individual drum sample that can be loaded as follows:

```
def create_right_pad(self):
    right_frame = Frame(self.root)
    right_frame.grid(row=10, column=6,
        sticky=W+E+N+S, padx=15, pady=2)
```

2.

```
self.button = [[0 for x in range(4)] for x in
                range(MAX_DRUM_NUM)]
for i in range(MAX_DRUM_NUM):
    for j in range(4):
        self.button[i][j] = Button(right_frame,
                                   bg='grey55')
        self.button[i][j].grid(row=i, column=j)
```

Description of the code is given below -

- Using list comprehension, we create an empty list of size 4 * MAX_DRUM_NUM

NOTE There is a reason behind grouping widgets into different methods. For example, we have created the separate left pad and right pad using two ~~different~~ methods. If we had defined these two group of widgets within the same method, the user would have to reload the drum samples every time the left buttons changed due to changes in BPU and units.

As a rule of thumb, it is always advisable to keep related widgets within a single method. However, the deciding class structure is more of an art than science to be learned and refined over a lifetime.

PTO

1 Next, we will create the play bar. The play bar at the bottom includes the Play button, the stop button, and a loop check button.

```
def create_play_bar(self):
    playbar_frame = Frame(self.root, height=15)
1   ln = MAX_DRUM_NUM + 10
    playbar_frame.grid(row = ln, columnspan = 13,
     sticky = W+E, padx = 15, pady = 10)
2   button = Button(playbar_frame, text='Play')
    button.grid(row = ln, column = 1, padx = 1)
3   button = Button(playbar_frame, text='Stop').
     grid(row = ln, column = 3, padx = 1)
4   loop = BooleanVar()
    loopbutton = Checkbutton(playbar_frame,
     text='Loop', variable = loop).grid
     (row = ln, column = 16, padx = 1)
```

5 Now that we have created all the widgets, its now time to actually display them by explicitly calling the methods that created them. We do that within the mainloop of our program as follows:

PTO

```
def app(self):
    self.root = Tk()
    self.root.title('Drum Beast')
    self.create_top_bar()
    self.create_left_pad()
    self.create_right_pad()
    self.create_play-bar()
    self.root.mainloop()
```