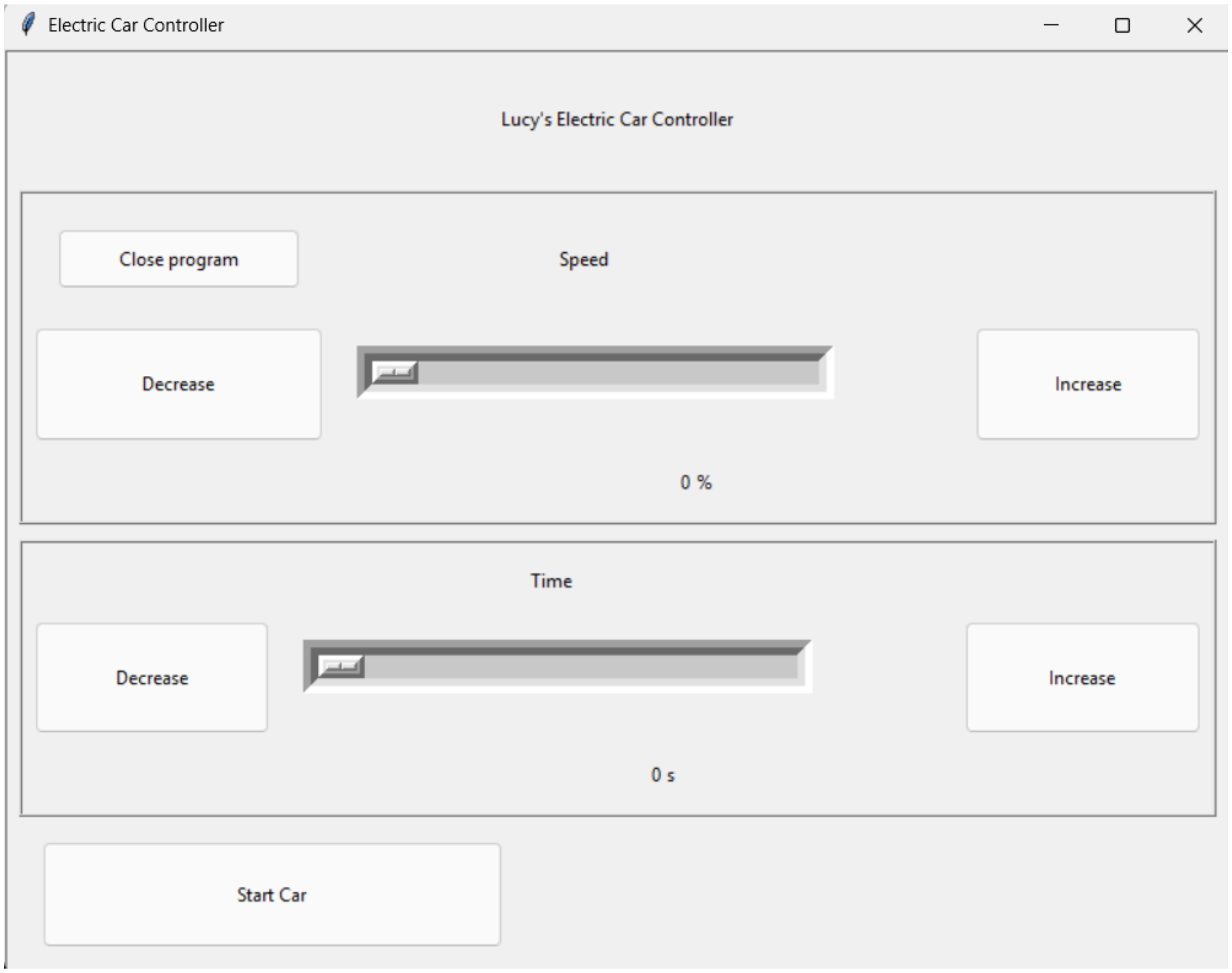
The Electric Car User Interface - frontend.py

A Breakdown

What does the user interface look like?



The user interface is divided into two main sections. The first section allows the user to set the car's speed as a percentage, ranging from 0% to 100%. A slider is provided for making large adjustments, while buttons on either side allow for finer, incremental changes. The second section controls the duration the car will travel at the selected speed. This time value can be set anywhere between 0 and 15 seconds, using the same combination of a slider for broad changes and buttons for small adjustments.

How was this made?

The entire UI was coded with the help of the Tkinter library. At the suggestion of Sheel I explored the Tkinter docs and taught myself how to use it to code this simple interface. The website I used to help with learning Tkinter was https://tkdocs.com/ so a big thanks to the lovely people who wrote the docs.

A quick explanation of Tkinter

Tkinter is a library that allows users to build UIs. Tkinter can be broken down into a few key commands.

- Control Commands
    - These commands allow coders to control large scale things such as overall positioning of objects and general stylistic choices such as background color.

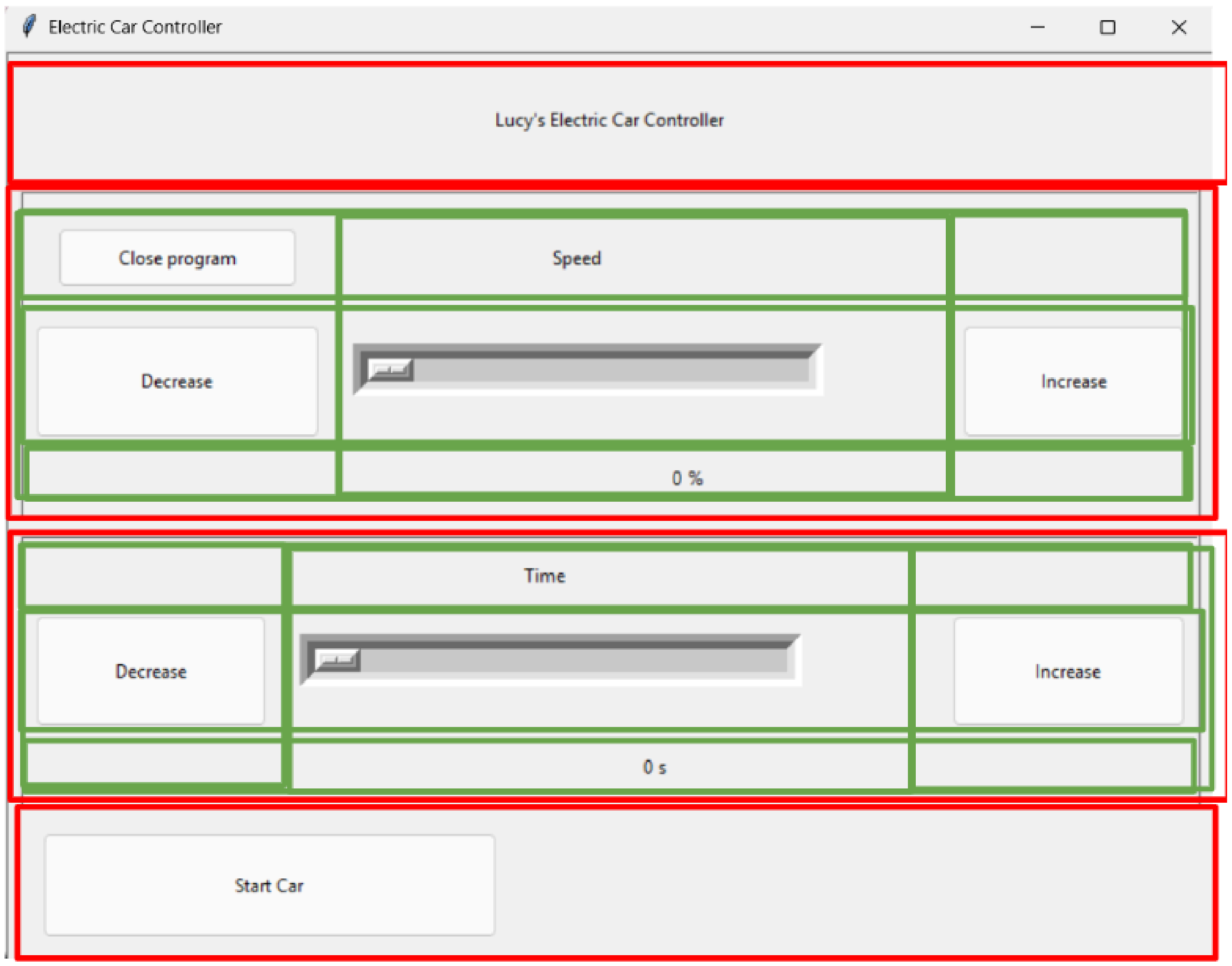
- Create commands
    - These commands create specific things on the screen such as an image, table, button, text, or slider.

- Place commands
    - These commands almost always follow a create command. These commands tell the computer where to place the objects it has created. Without this command, objects created do not appear on the screen because the computer doesn't know where to put them.

Pack vs. Grid

In Tkinter there exist two main commands for placing objects--Pack and Grid. Pack uses relative locations around the screen to decide where to place objects such as left, right, up, down, and center. Meanwhile, Grid uses a series of columns and rows and then places each object inside a specific column and row. Both can be used, but Grid is newer than Pack. Personally, I prefer Grid since it allows the user better control over precise object placement and in this project will be using it.

A breakdown of the layout



What do these boxes mean?

When using the Grid system in Tkinter, everything is positioned using rows and columns. Because of this, it's usually easier to divide large areas into smaller sections that nest neatly within each other. In the diagram above, I could have placed all the widgets in a single frame—similar to a table layout—but doing so would have required managing eight rows and four columns in one place, which gets confusing fast. Instead, I broke the interface into four main sections, each enclosed in its own frame (highlighted in red). Each of these red-framed sections contains its own internal frame (highlighted in green), with its own simpler grid layout. For example, the "Speed" and "Time" sections each have their own grid of rows and columns for the buttons, sliders, and labels. This approach makes the layout much more manageable and helps keep the code organized and easier to debug.

Problems

Overall, this project went quite smoothly, but I did encounter a few hiccups along the way. One of the biggest issues was setting up the columns and rows in Tkinter. According to the documentation, the first column and row are labeled as one. However, in practice, they actually start at zero. The documentation didn't make it clear that starting at one was just a stylistic choice. As a result, I initially added an extra column and row, which led to a second problem: getting the application to resize dynamically. I wanted dynamic resizing because I didn't know the exact screen dimensions where the app would be displayed, and I wanted it to fit the screen perfectly. However, the extra column and row threw off the resizing behavior. Eventually, I asked an AI assistant why the layout wasn't resizing as expected, and it helped me identify the issue. After that, I simply adjusted the numbers, but it was a bit frustrating that such a small detail caused so much trouble.

The code itself with explanations for what things do

```
#The first thing I do is import tkinter
from tkinter import *
from tkinter import ttk
import time

#These imports are for the backend so don't worry about them
#import board
#import busio
#from adafruit_mcp4725 import MCP4725
#import RPi.GPIO as GPIO
#import backend

#These are a few initial configurations for the project, such as the size
of the window and the title of the application.
root = Tk()
root.title('Electric Car Controller')
root.geometry("800x600")
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
```

```
#The application keeps track of a couple of numbers that the user inputs,
so that's these.
spe_total = StringVar(value='0')
time_total = StringVar(value='0')
start_op = StringVar(value='')

#i2c, dac = backend.initialize()

#This is the function the application uses to decrease the numbers on
screen. It’s a simple logic function that subtracts the number on screen
by one and then updates the screen.
def sub(total, min_val, change):
    try:
        value = int(total.get())
        if value > min_val:
            total.set(value - change)
    except ValueError:
        pass

#This function does the exact same thing as the function above except it
adds numbers.
def add(total, max_val, change):
    try:
        value = int(total.get())
        if value < max_val:
            total.set(value + change)
    except ValueError:
        pass
#This is a temporary function for the start button when the backend is not
connected. All it does is prevent the user from clicking the button more
than once and activating the start function many times over.
def start():
    start_button.config(state='disabled')
    start_button.config(state='normal')

#This function resets the values incase they become broken
def reset():
   spe_total.set('0')
   time_total.set('0')
```

```
#This function looks complicated, but all it really does is generate a
number with two buttons on either side. I did it like this so that I could
easily create one version for speed and one for time without repeating
myself. Basically it makes all the green boxes except for the slider.
def layout(parent, title, total, unit, min_val, max_val, change):
    ttk.Label(parent, text=title, anchor='center').grid(column=2, row=0,
sticky='nsew', padx=5, pady=5)
    ttk.Button(parent, text='Decrease', command=lambda: sub(total,
min_val, change)).grid(column=0, row=1, sticky='nsew', padx=5, pady=5)
    ttk.Button(parent, text='Increase', command=lambda: add(total,
max_val, change)).grid(column=5, row=1, sticky='nsew', padx=5, pady=5)
    ttk.Label(parent, textvariable=total, anchor='e').grid(column=2,
row=2, sticky='nsew', padx=0, pady=5)
    ttk.Label(parent, text=unit, anchor='w').grid(column=3, row=2,
sticky='nsew', padx=0, pady=5)

#This function defines the number of rows and columns for a frame. In
Tkinter in order to grid something it must be placed inside a frame with a
specified number of columns and rows. I made this function to make it
quicker for me to set the columns and rows for frames.
def config(parent, col_num, row_num, col_weight, row_weight):
    for i in range(col_num):
        parent.columnconfigure(i, weight=col_weight)
    for i in range(row_num):
        parent.rowconfigure(i, weight=row_weight)
#This creates the initial window
content = ttk.Frame(root)
content.grid(column=0, row=0, sticky='nsew')
content.columnconfigure(0, weight=1)
content.rowconfigure(0, weight=1)

#This creates a frame inside that window to place the red boxes defined in
the diagram.
mainframe = ttk.Frame(content, borderwidth=5, relief='ridge')
mainframe.grid(column=0, row=0, sticky='nsew')
mainframe.grid_propagate(False)

#This tells that frame how many columns and rows to create.
config(mainframe, 1, 4, 1, 1)
```

```
#This creates the four red boxes.
title_frame = ttk.Frame(mainframe, borderwidth=5)
topframe = ttk.Frame(mainframe, borderwidth=5, relief='ridge')
bottomframe = ttk.Frame(mainframe, borderwidth=5, relief='ridge')
begin = ttk.Frame(mainframe, borderwidth=5)

#This places the four red boxes in the main frame.
title_frame.grid(column=0, row=0, sticky='nsew', padx=5, pady=5)
topframe.grid(column=0, row=1, sticky='nsew', padx=5, pady=5)
bottomframe.grid(column=0, row=2, sticky='nsew', padx=5, pady=5)
begin.grid(column=0, row=3, sticky='nsew', padx=5, pady=5)

#This tells each of those boxes how many columns and rows to have for all
the text and buttons.
config(topframe, 6, 3, 1, 1)
config(bottomframe, 6, 3, 1, 1)
config(title_frame, 5, 1, 1, 1)
config(begin, 3, 1, 1, 1)
#This is testing function from when I was developing this
def test():
    start_button.config(state='disabled')
    print('Button disabled')
    time.sleep(5)

#This links the front to the back end. Don’t worry about it.
def wrapper(dac, i2c, spe_total, time_total):
    pass
#    start_button.config(state='disabled')
#
#    output_voltage = ((3 - 2.5) * (int(spe_total.get()) / 100)) + 2.5

#    backend.volt(dac=dac, i2c=i2c, output_voltage=output_voltage,
duration=int(time_total.get()))

#    time.sleep(int(time_total.get()))

#    dac.raw_value = 0

#    start_button.config(state='normal')
```

```
#This creates the title and the sliders for time and speed.
title = ttk.Label(title_frame, text="Lucy's Electric Car Controller",
anchor='center').grid(column=2, row=0, sticky='nsew')
res = ttk.Button(topframe, text='Close program', command=lambda:
backend.reset(dac, i2c))
res.grid(column=0, row=0, sticky='nsew', padx=20, pady=20)
s_spe = Scale(topframe, borderwidth=10, resolution=5, variable=spe_total,
showvalue=0, from_ = 0, to = 100, orient = HORIZONTAL)
s_spe.grid(column=1, row=1, columnspan=3, sticky='nsew', padx=5, pady=5)
s_time = Scale(bottomframe, borderwidth=10, variable=time_total,
resolution=1, showvalue=0, from_ = 0, to = 15, orient = HORIZONTAL)
s_time.grid(column=1, row=1, columnspan=3, sticky='nsew', padx=5, pady=5)

#This creates the start button and the radio buttons at the bottom.
# start_button = ttk.Button(begin, text='Start Car', command=lambda:
backend.volt(dac=dac, i2c=i2c, output_voltage=3, duration=3))
start_button = ttk.Button(begin, text='Start Car', command=lambda:
wrapper(dac=dac, i2c=i2c, spe_total=spe_total, time_total=time_total))
start_button.grid(column=0, row=0, sticky='nsew', padx=10, pady=1)
speed_button = ttk.Radiobutton(begin, text="Speed", variable=start_op,
value='speed')
distance_button = ttk.Radiobutton(begin, text="Time", variable=start_op,
value='time')
#speed_button.grid(column=1, row=0, sticky='nsew', padx=5, pady=1)
#distance_button.grid(column=2, row=0, sticky='nsew', padx=5, pady=1)

#This calls the layout function defined at the top which creates the
values, increase and decrease buttons, and displays the values of each.
layout(topframe, 'Speed', spe_total, '%', 0, 100, 1)
layout(bottomframe, 'Time', time_total, 's', 0, 15, 1)

#This just tells tkinter to start. It's like calling main().
root.mainloop()
```

Access this all at https://github.com/Sheel2007/EVCar