BAIS: 3020

Project Code Descriptions

By Alex Hingtgen and Zak Zahner

GUI Frame Formatting

The code below shows the beginning of creating our GUI frame and menu with different buttons and a list and entry box. The frame for the GUI was created through the code and we provided a red coloring for the frame background. We also provided a main title for the code and then a second title that further described what our program analysis intended to do. This code uses tkinter in its formatting.

```
class AllTkinterWidgets:
      def __init__(self, master):
        frame = Frame(master, width=800, height=800, bg= "red")
        frame.pack(expand = 0)
        frame.pack(side = TOP, anchor = N)
        self.lb1 = Label(frame, text= '2020 Chicago Roadway Collision Data: Car Crash Prevention Simulator')
        self.lb1.pack(fill = X, side = TOP)
        self.lb2 = Label(frame, text='Analyze Past Car Crash Impacts from Chicago')
        self.lb2.pack(fill = X, side = TOP)
        self.label1 = tkinter.Label()
```

Menu Creation

We were able to create the menu for our graphical user interface by integrating the frame that we previously created. Next, we formatted the menu to be at the top of the page. This code followed the steps we learned in class in order to create a menu.

```
        # ----------------------Menu Creation -------------------------
        self.mbar = Frame(frame, relief = 'raised', width=2, bd = 2)
        self.mbar.pack(expand = 0, fill = X, side = TOP)
```

Radio Button Bar Creation

This chunk of code creates and formats the radio buttons in the GUI. It also creates the string variable that is called on every time the radio button is clicked by the user. A name was provided for each radio button in this code and a particular value was given so that a future function would know which line graph to display. The end of this code was done to format the radio button bar.

```
# ----------------- RadioButtonBar Creation -----------------
self.rbbar = Frame(frame, relief = 'sunken', width=2, bd = 2)
self.rbbar.pack(expand = 1, fill = BOTH, side = BOTTOM, pady = 5)

# Radio buttons specified as string variables with values to run a certain action
self.v = StringVar()
Label(self.rbbar, text='Roadway Surface Conditions:').pack(side=LEFT, padx=5)
self.rDry  = Radiobutton(self.rbbar, text='Dry', variable = self.v, value='Dry', command=self.OnDry)
self.rWet    = Radiobutton(self.rbbar, text='Wet', variable = self.v, value='Wet', command=self.OnWet)
self.rSnowSlush = Radiobutton(self.rbbar, text = 'Snow or Slush', variable = self.v, value='Snow or Slush', com
self.rIce = Radiobutton(self.rbbar, text = 'Ice', variable = self.v, value='Ice', command=self.OnIce)
self.rAll = Radiobutton(self.rbbar, text = 'All', variable = self.v, value='All', command=self.OnAll)


self.rDry.pack(side=LEFT)
self.rWet.pack(side=LEFT)
self.rSnowSlush.pack(side=LEFT)
self.rIce.pack(side=LEFT)
self.rAll.pack(side=LEFT)
```

Button Bar Creation

In this code, we create the area where our button bar is later displayed. This also used the frame that we previously created and then we were able to format the button bar to be in the bottom part of the frame. Moreover, this code also followed the steps that we learned in lecture to create a button bar.

```
# -------------------- ButtonBar Creation --------------------
self.bbar = Frame(frame, relief = 'sunken', width=2, bd = 2)
self.bbar.pack(expand = 1, fill = BOTH, side = BOTTOM, pady = 5, before = self.rbbar)
```

Entry Box Frame

This section of the code creates the entry box frame where users are able to input a specific file and then click the "Load Crash Data…" button in order to display the file desired to be seen. If there are no errors and the file is typed in properly, then a user will be able to see a graphical display solely by using this part of the GUI. Most of this code was meant to format the entry box frame as well, such as at the top.

```
# ---------------------- entry box frame -----------------------
self.t = StringVar()
self.ef = Frame(frame, bd=2, relief='groove')
self.lb2 = Label(self.ef, text='File:')
self.lb2.pack(side= LEFT)
self.entry = Entry(self.ef, textvariable = self.t, bg='white')
self.bt = Button(self.ef, text = 'Load Crash Data...', command = self.load)
self.entry.pack(side = LEFT, padx = 5)
self.bt.pack(side = LEFT, padx= 5)
self.ef.pack(expand=0, fill=X, pady=5, before = self.bbar, side = TOP)
```

Listbox Frame

This chunk of code creates the buttons on the bottom half of the GUI. These buttons are how the user picks what graph/chart that they want to see presented on the screen. All of the formatting and creation of these buttons is done through this chunk of code.

```
#---------------------- listbox frame -------------------------
self.lf = Frame(frame, bd=2, relief='groove')
self.lb = Label(self.lf, text='Crash Data Analysis Options:')
self.bt1 = Button(self.lf, text = 'Clear', command = self.clear)
self.bt2 = Button(self.lf, text = 'Average Crash Cost by Speed Zone', command = self.getAverageCostHistogram)
self.bt3 = Button(self.lf, text = 'Calculate 2020 Monthly Crashes', command = self.getMonthlyCrashesGraph)
self.bt4 = Button(self.lf, text = 'Compute Crash Injury Potential', command = self.getInjuryPieCharts)
self.bt5 = Button(self.lf, text = 'Calculate Crashes based on Road Conditions', command = self.getConditionsLineGraphs)
self.listbox = Listbox(self.lf, height=8)
self.sbl = Scrollbar(self.listbox, orient=VERTICAL, command=self.listbox.yview)
self.listbox.configure(yscrollcommand=self.sbl.set)
self.lb.pack(side=LEFT, padx=5)
self.bt1.pack(side = BOTTOM)
self.bt2.pack(side = BOTTOM)
self.bt3.pack(side = BOTTOM)
self.bt4.pack(side = BOTTOM)
self.bt5.pack(side = RIGHT)
self.sbl.pack(side=RIGHT, fill=Y)
self.listbox.pack(padx=5, fill = X)
self.lf.pack(expand=0, fill=X, pady=5, before = self.ef, side = BOTTOM)

self.image1 = None
```

Clear Button Function

The clear button is created by this code. The clear button allows the user to clear any graph that is on the screen at that current time. It clears what was previously loaded and enables the system to properly load a new file in further code chunks.

```
def clear(self):
    self.listbox.delete(0, END)
    self.label1.destroy()
    self.label1 = Label()
    self.lbl1.destroy()
```

Monthly Crashes Graph Function

This is the button that prints the image of the Crashes Per Month graph. The function has already run that creates and saves the picture so this just calls upon that saved image and prints the picture in the centered spot on the screen. Additionally, the image is also placed on the screen through this code.

```
def getMonthlyCrashesGraph(self):
    self.listbox.delete(0, END)
    self.listbox.delete(0, END)
    self.graphImage = Image.open('CrashesPerMonth.png')
    test = ImageTk.PhotoImage(self.graphImage)
    self.lbl1 = tkinter.Label(image=test)
    self.lbl1.image = test
    self.lbl1.place(x=500, y=325)
```

Injury Pie Charts Function

This is the button that prints the image of the Injury Pie Charts. The function has already run that creates and saves the picture so this just calls upon that saved image and prints the picture in the centered spot on the screen. Additionally, the image is also placed on the screen through this code.

```
def getInjuryPieCharts(self):
    self.listbox.delete(0, END)
    self.graphImage = Image.open('InjuryPotential.png')
    test = ImageTk.PhotoImage(self.graphImage)
    self.lbl1 = tkinter.Label(image=test)
    self.lbl1.image = test
    self.lbl1.place(x=500, y=325)
```

Average Cost Histogram Function

This is the button that prints the image of the Average Cost Histogram. The function has already run that creates and saves the picture so this just calls upon that saved image and prints the picture in the centered spot on the screen. Additionally, the image is also placed on the screen through this code.

```
def getAverageCostHistogram(self):
    self.graphImage = Image.open('AverageCostHist.png')
    test = ImageTk.PhotoImage(self.graphImage)
    self.lbl1 = tkinter.Label(image=test)
    self.lbl1.image = test
    self.lbl1.place(x=500, y=325)
```

Conditions Line Graph Function

This function is called on when the Calculate Road Conditions button is clicked. Whichever radio button was last clicked is what the r_b_v variable is set to. That variable value is then passed through the if elif functions until one is met. Whatever condition is met is then called upon in order to print the correct image of the correct line chart of the road condition.

```
def getConditionsLineGraphs(self):
    self.listbox.delete(0, END)
    r_b_v = self.v
    if r_b_v == 'all':
      self.graphImage = Image.open('AllConditionsPlot.png')
      test = ImageTk.PhotoImage(self.graphImage)
      self.lbl1 = tkinter.Label(image=test)
      self.lbl1.image = test
      self.lbl1.place(x=500, y=325)
    elif r_b_v == 'dry':
      self.graphImage = Image.open('DryPlot.png')
      test = ImageTk.PhotoImage(self.graphImage)
      self.lbl1 = tkinter.Label(image=test)
      self.lbl1.image = test
      self.lbl1.place(x=500, y=325)
    elif r_b_v == 'wet':
      self.graphImage = Image.open('WetPlot.png')
      test = ImageTk.PhotoImage(self.graphImage)
      self.lbl1 = tkinter.Label(image=test)
      self.lbl1.image = test
      self.lbl1.place(x=500, y=325)
    elif r_b_v == 'Snow or Slush':
      self.graphImage = Image.open('SnoworSlushPlot.png')
      test = ImageTk.PhotoImage(self.graphImage)
      self.lbl1 = tkinter.Label(image=test)
      self.lbl1.image = test
      self.lbl1.place(x=500, y=325)
    elif r_b_v == 'ice':
      self.graphImage = Image.open('IcePlot.png')
      test = ImageTk.PhotoImage(self.graphImage)
      self.lbl1 = tkinter.Label(image=test)
      self.lbl1.image = test
      self.lbl1.place(x=500, y=325)
```

OnDry, OnWet, OnSnowSlush, OnIce, & OnAll Functions

These are the functions that are called upon when the corresponding radio button is clicked. This makes sure that the r_b_v value is always correct as to which one is selected. When that specific value that is shown in the quotes is chosen through the button, then these functions help link the display of the specific line graph.

```
def OnDry(self):
    self.v = 'dry'

def OnWet(self):
    self.v = 'wet'

def OnSnowSlush(self):
    self.v = 'Snow or Slush'

def OnIce(self):
    self.v = 'ice'

def OnAll(self):
    self.v = 'all'
```

Load and Loading Functions

The code below was used to load the images on to the graphical user interface screen. As shown, the image is opened and configured through the code below to prepare it for display. There is also error handling done in case there is an issue loading the specific file, as seen in both functions with an inserted error message. The load function first opens the image and configures it properly. Subsequently, the loading function opens the image properly with Tkinter using methods we learned in class. This also configures the picture and places it as desired on the screen.

```
def load(self):
    try:
       print(self.t.get())
       self.image1 = Image.open(self.t.get())
       test = ImageTk.PhotoImage(self.image1)
       self.label1.config(image='')
       self.label1.config(image=test)
       self.label1.image = test
       self.label1.place(x=425, y=0)
    except Exception as e:
        print(e)
        self.listbox.insert(END, 'Error loading file' + self.t.get())

    self.listbox.insert(END, 'Loaded File' + self.t.get())

def loading(self,file):
    try:
        img = Image.open(file)
        test = ImageTk.PhotoImage(img)
        print(file)
        self.label1.config(image='')
        self.label1.config(image=test)
        self.label1.image = test
        self.label1.place(x=425, y = 0)
        print(file)
    except Exception as e:
        print(e)
        self.listbox.insert(END, 'Error loading file' + file)

    self.listbox.insert(END, 'Loaded file' + file)
```

Chart Preparation and Main Display Creation

These functions are what actually creates and saves the images of the charts. Each of the functions runs through mathematical processes and data visualizations creation and then saves the images. This allows the button to actually call upon an image that has been previously created.

```
if __name__ == '__main__':
    # Preparing Graphs
    getMonthlyCrashesGraph()
    getInjuryPieCharts()
    getAverageCostHistogram()
    onDry()
    onAll()
    onWet()
    onSnoworSlush()
    onIce()
    # main--display the menu
    root = Tk()

    root.configure(background='Black')
    root.geometry('1500x1200')

    all = AllTkinterWidgets(root)
    root.title('Computational Thinking ~ Final Project')

root.mainloop()
```

Four Main Questions we Answered with Data Visualizations

Bar Chart Creation for Total Monthly Crash Amounts

This code reads through our csv file line by line and sorts crashes in the certain month that they occured. We do this by utilizing a dictionary. Then we can go straight to plotting and formatting this bar chart through the matplotlib library. We wanted to make the graph easy to read with a specific x-axis, y-axis, and title.

```
#function creating the total crashes per month bar chart
def getMonthlyCrashesGraph():
    monthCounts = {}
    with open('Traffic_Crashes_-_Crashes.csv', 'r') as csv_file:
        csv_reader = csv.DictReader(csv_file)

        for line in csv_reader:
            month = line["CRASH_MONTH"]
            monthCounts[month] = monthCounts.get(month, 0) + 1

    names = list(monthCounts.keys())
    values = list(monthCounts.values())

    plt.bar(range(len(monthCounts)), values, tick_label=names)
    plt.xlabel('Month')
    plt.ylabel('Number of Crashes')
    plt.title('Car Crashes per Month')
    plt.savefig('CrashesPerMonth.png')
    plt.clf()
```

Pie Chart Creation for Injury and Fatalities Proportions

This code runs line by line through our csv file and sorts whether there was an injury, fatality, or injury leading to a fatality or not. The value is then placed into the correct bin where we can go straight to creating and graphing and saving the pie chart image. There are basically three sections to these code chunks as we developed a subplot with three different pie charts. Each pie chart provides its own meaningful data analysis and follows a similar coding structure as the one prior.

```
#function creating injury proportion per total crashes, fatalities per total crashes,
#and fatalities per total crashes with injuries pie charts
def getInjuryPieCharts():
    injuryCounts = {}
    with open('Traffic_Crashes_-_Crashes.csv', 'r') as csv_file:
        csv_reader = csv.DictReader(csv_file)

        for line in csv_reader:
            injuryTotal = line["INJURIES_TOTAL"]
            if injuryTotal == '':
                continue
            injuryTotal = int(injuryTotal)
            if injuryTotal > 0:
                injuryTotal = '1+ Injuries'
            else:
                injuryTotal = 'No Injuries Reported'
            injuryCounts[injuryTotal] = injuryCounts.get(injuryTotal, 0) + 1

        y = []
        mylabels = []
        for key in injuryCounts:

            y.append(injuryCounts[key])
            mylabels.append(key)
            plt.subplot(3, 1, 1)
        plt.pie(y, labels = mylabels)
    plt.title('Proportion of Crashes Resulting in Injury')
```

```
fatalCounts = {}
with open('Traffic_Crashes_-_Crashes.csv', 'r') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    for line in csv_reader:
        fatalTotal = line["INJURIES_FATAL"]
        if fatalTotal == '':
            continue
        fatalTotal = int(fatalTotal)
        if fatalTotal > 0:
            fatalTotal = '1+ Fatalities'
        else :
            fatalTotal = 'No Fatalities Reported'
        fatalCounts[fatalTotal] = fatalCounts.get(fatalTotal, 0) + 1

    y2 = []
    mylabels2 = []
    for key in fatalCounts:
        y2.append(fatalCounts[key])
        mylabels2.append(key)
        plt.subplot(3, 1, 2)
    plt.pie(y2, labels = mylabels2)
plt.title('Proportion of Crashes Causing Fatalities')
```

```
injuryTotal =  0
fatalTotal = 0
with open('Traffic_Crashes_-_Crashes.csv', 'r') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    for line in csv_reader:
        fatal = line['INJURIES_FATAL']
        injury = line['INJURIES_TOTAL']
        if injury == '':
            continue
        else:
            fatal = int(fatal)
            injury = int(injury)
            injuryTotal = injuryTotal + injury
            fatalTotal = fatalTotal + fatal
    fatalTotalList = [fatalTotal, injuryTotal]
    labels1 = ["Injuries Resulting in Fatalities", "Injuries Resulting in No Fatalities"]
    plt.subplot(3,1,3)
    plt.pie(fatalTotalList, labels = labels1)
    plt.title("Total Injuries vs Total Fatalities")

    plt.savefig('InjuryPotential.png')
plt.clf()
```

Histogram Creation for Average Costs in each Speed Limit

This code reads through our csv file and creates 3 bins for the amount a car crash could possibly cost someone. Then matplotlib can create a histogram and graph each of these values. Then we save the image, which is then called upon in the main functions. There is a legend created in the code below and each variable in the legend receives its own color to help identify it in the graph. The pandas library help us easily formulate this data.

```
#function creating number of crashes in each speed limit and the cost of damages histogram
def getAverageCostHistogram():
    df = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    x1 = df.loc[df.DAMAGE=='OVER $1,500', 'POSTED_SPEED_LIMIT']
    x2 = df.loc[df.DAMAGE=='$501 - $1,500', 'POSTED_SPEED_LIMIT']
    x3 = df.loc[df.DAMAGE=='$500 OR LESS', 'POSTED_SPEED_LIMIT']
    kwargs = dict(alpha=1, bins=10)
    plt.figure()
    plt.hist(x1, **kwargs, color='g', label='OVER $1,500')
    plt.hist(x2, **kwargs, color='b', label='$501 - $1,500')
    plt.hist(x3, **kwargs, color='r', label='$500 OR LESS')
    plt.legend()
    plt.xlabel('Speed Limit in MPH')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,75,5))
    plt.title('Crash Costs per Speed Limits')
    plt.savefig('AverageCostHist.png')
    plt.clf()
```

Line Chart Creation for Each Condition

These chunks of code all create a unique line graph for each of the stated road conditions. We sort through the amount of crashed hour by hour. Then one of the code chunks combines all of these plots and overlays them on top of each other in order to get the All() chart with a provided legend. The code is very similar for creating the unique line graphs for dry, wet, snowy or slushy, and icy conditions. Finally, the last code chunk shows our efforts to show all of the conditions in one single data visualization. We were able to plot by only inputting the specific line for each roadway condition and then combining these all to create a single line chart display.

```
#function creating line chart for total crashes at each hour in dry conditions
def onDry() :
    data = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    dry = data[(data.ROADWAY_SURFACE_COND == "DRY")]
    dryplot = dry.groupby(['CRASH_HOUR']).count()
    dryplot.rename(columns={'CRASH_RECORD_ID':'DRY'}, inplace=True)
    dryplot = dryplot["DRY"]
    dryplot.plot.line()
    plt.xlabel('Crash Hour in Military Time')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,24,3))
    plt.title('Total Crashes at Each Hour on Dry Roads')
    plt.savefig('DryPlot.png')
    plt.clf()

#function creating line chart for total crashes at each hour in wet conditions
def onWet() :
    data = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    wet = data[(data.ROADWAY_SURFACE_COND == "WET")]
    wetplot = wet.groupby(['CRASH_HOUR']).count()
    wetplot.rename(columns={'CRASH_RECORD_ID':'WET'}, inplace=True)
    wetplot = wetplot["WET"]
    wetplot.plot.line()
    plt.xlabel('Crash Hour in Military Time')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,24,3))
    plt.title('Total Crashes at Each Hour on Wet Roads')
    plt.savefig('WetPlot.png')
    plt.clf()
```

```
#function creating line chart for total crashes at each hour in snow or slush conditions
def onSnoworSlush() :
    data = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    snoworslush = data[(data.ROADWAY_SURFACE_COND == "SNOW OR SLUSH")]
    snoworslushplot = snoworslush.groupby(['CRASH_HOUR']).count()
    snoworslushplot.rename(columns={'CRASH_RECORD_ID':'Snow or Slush'}, inplace=True)
    snoworslushplot = snoworslushplot["Snow or Slush"]
    snoworslushplot.plot.line()
    plt.xlabel('Crash Hour in Military Time')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,24,3))
    plt.title('Total Crashes at Each Hour on Snowy or Slushy Roads')
    plt.savefig('SnoworSlushPlot.png')
    plt.clf()

#function creating line chart for total crashes at each hour in icy conditions
def onIce() :
    data = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    ice = data[(data.ROADWAY_SURFACE_COND == "ICE")]
    iceplot = ice.groupby(['CRASH_HOUR']).count()
    iceplot.rename(columns={'CRASH_RECORD_ID':'Ice'}, inplace=True)
    iceplot = iceplot["Ice"]
    iceplot.plot.line()
    plt.xlabel('Crash Hour in Military Time')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,24,3))
    plt.title('Total Crashes at Each Hour on Icy Roads')
    plt.savefig('IcePlot.png')
    plt.clf()
```

```
#function creating line chart for total crashes at each hour in all conditions
def onAll() :
    data = pd.read_csv("Traffic_Crashes_-_Crashes.csv")
    dry = data[(data.ROADWAY_SURFACE_COND == "DRY")]
    dryplot = dry.groupby(['CRASH_HOUR']).count()
    dryplot.rename(columns={'CRASH_RECORD_ID':'DRY'}, inplace=True)
    dryplot = dryplot["DRY"]

    wet = data[(data.ROADWAY_SURFACE_COND == "WET")]
    wetplot = wet.groupby(['CRASH_HOUR']).count()
    wetplot.rename(columns={'CRASH_RECORD_ID':'WET'}, inplace=True)
    wetplot = wetplot["WET"]

    ice = data[(data.ROADWAY_SURFACE_COND == "ICE")]
    iceplot = ice.groupby(['CRASH_HOUR']).count()
    iceplot.rename(columns={'CRASH_RECORD_ID':'ICE'}, inplace=True)
    iceplot = iceplot["ICE"]

    snowslush = data[(data.ROADWAY_SURFACE_COND == "SNOW OR SLUSH")]
    snowplot = snowslush.groupby(['CRASH_HOUR']).count()
    snowplot.rename(columns={'CRASH_RECORD_ID':'SNOW OR SLUSH'}, inplace=True)
    snowplot = snowplot["SNOW OR SLUSH"]

    allConditionsPlot = pd.concat([dryplot, wetplot, iceplot, snowplot], axis=1)
    allConditionsPlot.plot.line()
    plt.xlabel('Crash Hour in Military Time')
    plt.ylabel('Number of Crashes')
    plt.xticks(np.arange(0,24,3))
    plt.title('Total Crashes at Each Hour in All Conditions')
    plt.savefig('AllConditionsPlot.png')
    plt.clf()
```