Jasmine Parmar, Kevin Osadiaye
CSE 681 Software Modeling
Project #3

See the README.md file for instructions on running the code for Project #3.

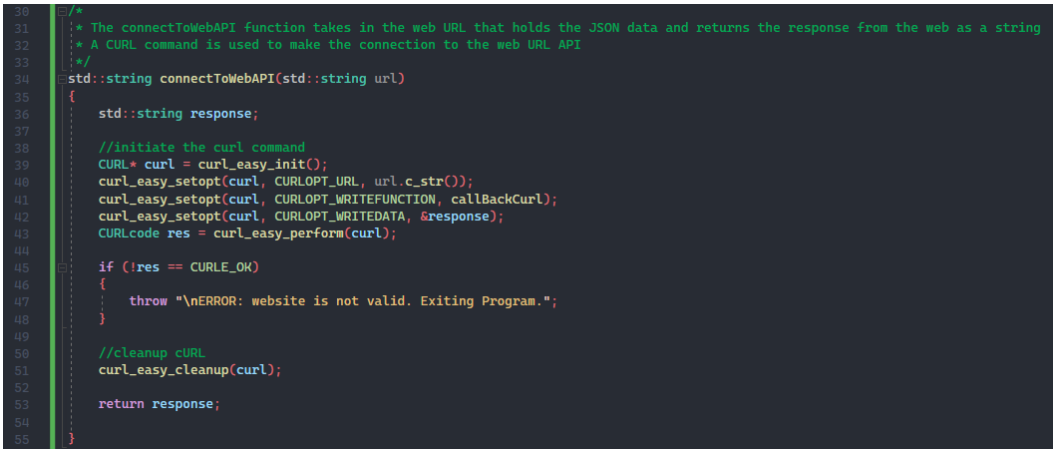
1. Requirement: You need to handle pressing a button and then returning a subset or derived data from a JSON file used in the last assignment. The app does not need to handle selecting data to request from the API server.

Project #3 primarily consists of a Python script and a DLL (written in C++). The Python script creates a Tkinter front-end GUI with button event handlers, and also loads the DLL file that performs the back-end processing. The DLL file, named json.dll, contains the code from Project #2.

Originally, in Project #2, we created a console application that prompts the user to iterate all 32 websites using the API server and then writes the team records to the console. In this project #3, however, we compiled the code into a DLL instead of the executable/binary file that interacts with the command prompt. The function getTeamInformation() below is declared externally as a C function (shown on lines 171 – 175) in order for the Python script to be able to read in the function since it uses the ctypes library to load and interpret the DLL.

```
146  void getTeamInformation()
147  {
148      JSONParser jsonParser;
149      FileHandler fileHandler;
150      std::queue<std::string> messageQueue;
151      std::vector<std::string> responses = std::vector<std::string>();
152
153      for (int i = 1; i <= 32; i++)
154      {
155          std::string webLink = "https://sports.snoozle.net/search/nfl/searchHandler?fileType=inline&statType=teamStats&season=2020&teamName=" + std::to_string(i);
156
157          messageQueue.push(webLink);
158          ProcessMessages(messageQueue, responses);
159      }
160
161      for (int i = 0; i <= 31; i++)
162      {
163          jsonParser.parse(responses[i].c_str());
164
165      }
166
167      teamData = getTeamRecords(jsonParser);
168
169  }
170
171  extern "C" {
172      const char* getTeamInfo() {
173          getTeamInformation();
174          return teamData.c_str();
175      }
176  }
```

The connectToWebAPI function was also reused from the previous Project #1 to make the Web API connection. This function utilizes lib curl to make the connection.



```
30  /*
31   * The connectToWebAPI function takes in the web URL that holds the JSON data and returns the response from the web as a string
32   * A CURL command is used to make the connection to the web URL API
33   */
34  std::string connectToWebAPI(std::string url)
35  {
36      std::string response;
37
38      //initiate the curl command
39      CURL* curl = curl_easy_init();
40      curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
41      curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, callBackCurl);
42      curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);
43      CURLcode res = curl_easy_perform(curl);
44
45      if (!res == CURLE_OK)
46      {
47          throw "\nERROR: website is not valid. Exiting Program.";
48      }
49
50      //cleanup cURL
51      curl_easy_cleanup(curl);
52
53      return response;
54
55  }
```

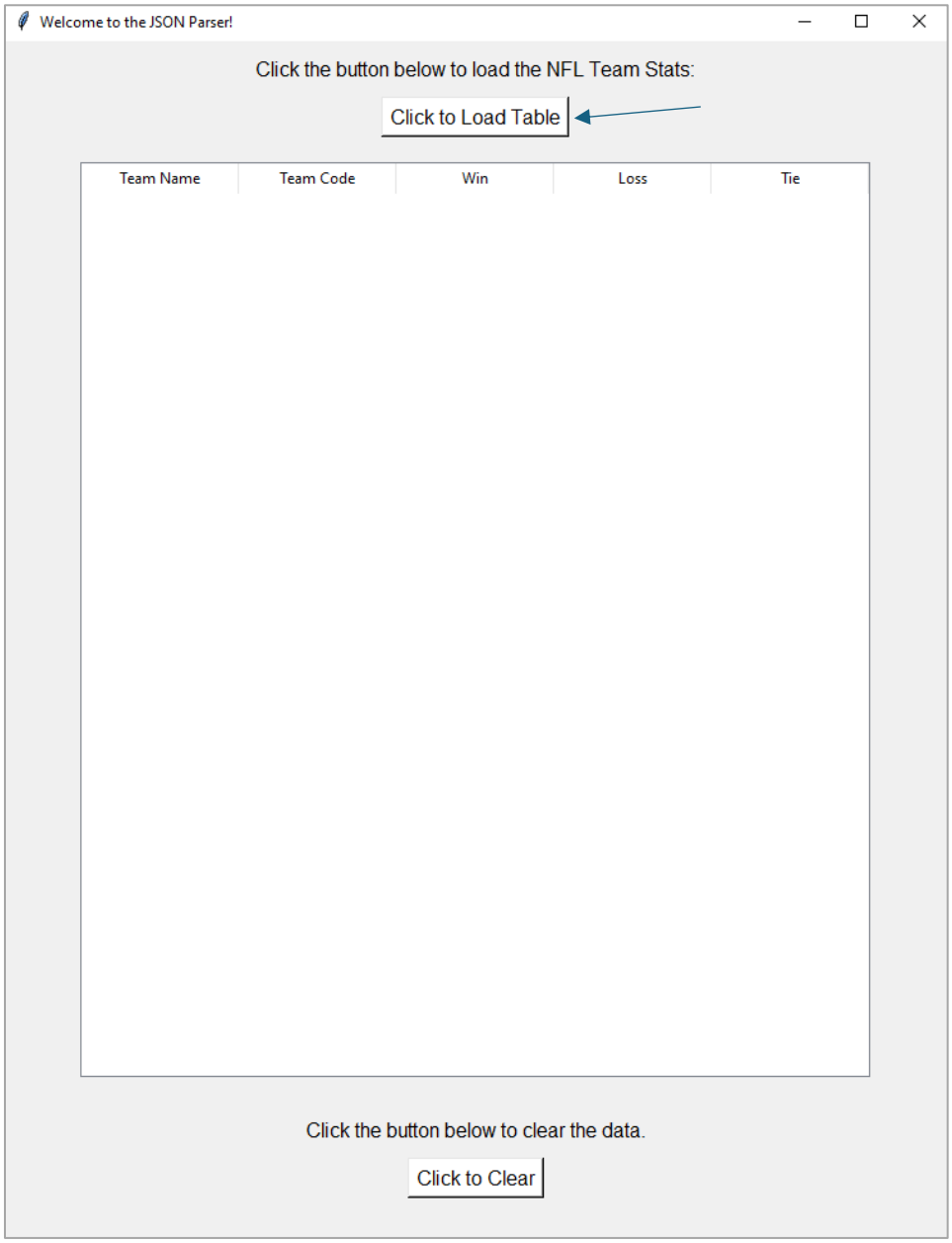
The code for the Python GUI is below (from jsonGUI.py file):

```python
import tkinter as tk
from ctypes import *
from tkinter import ttk

# load the json DLL
lib = cdll.LoadLibrary('./json.dll')
lib.getTeamInfo.argtypes = None
lib.getTeamInfo.restype = c_char_p


def clear_click_handler():
    tree.delete(*tree.get_children())

def load_click_handler():
    # get all information from the json DLL and decode
    data = lib.getTeamInfo()
    all_stats = data.decode("latin1")

    #split all information by the comma delimiter
    formatted_stats = all_stats.split(",")
    team_stats = [formatted_stats[i:i+5] for i in range(0, len(formatted_stats), 5)]

    for item in team_stats:
        tree.insert("", tk.END, values=item)

if __name__ == "__main__":

    # create the main window
    window = tk.Tk()
    window.title("Welcome to the JSON Parser!")
    window.geometry("750x950")

    # create a label widget
    label = tk.Label(window, text="Click the button below to load the NFL Team Stats:", font="Arial")
    label.size=25
    label.pack(pady = 10)

    # create a button widget with a button click event handler
    button = tk.Button(window, bg = 'white', text="Click to Load Table", command=load_click_handler, font = "Arial")
    button.pack()

    # create a tree (table) widget
    tree = ttk.Treeview(window, columns=("Team Name", "Team Code", "Win", "Loss", "Tie"), show="headings", height=35)
    tree.pack(padx=20, pady=20)
    for column in tree["columns"]:
        tree.column(column, width=125, anchor=tk.CENTER)
        tree.heading(column, text=column)


    # create label and button widgets to clear data
    label = tk.Label(window, text="Click the button below to clear the data.", font="Arial")
    label.size=25
    label.pack(pady = 10)

    button = tk.Button(window, bg = 'white', text="Click to Clear", command=clear_click_handler, font = "Arial")
    button.pack()

    # run the Tkinter event loop
    window.mainloop()
```

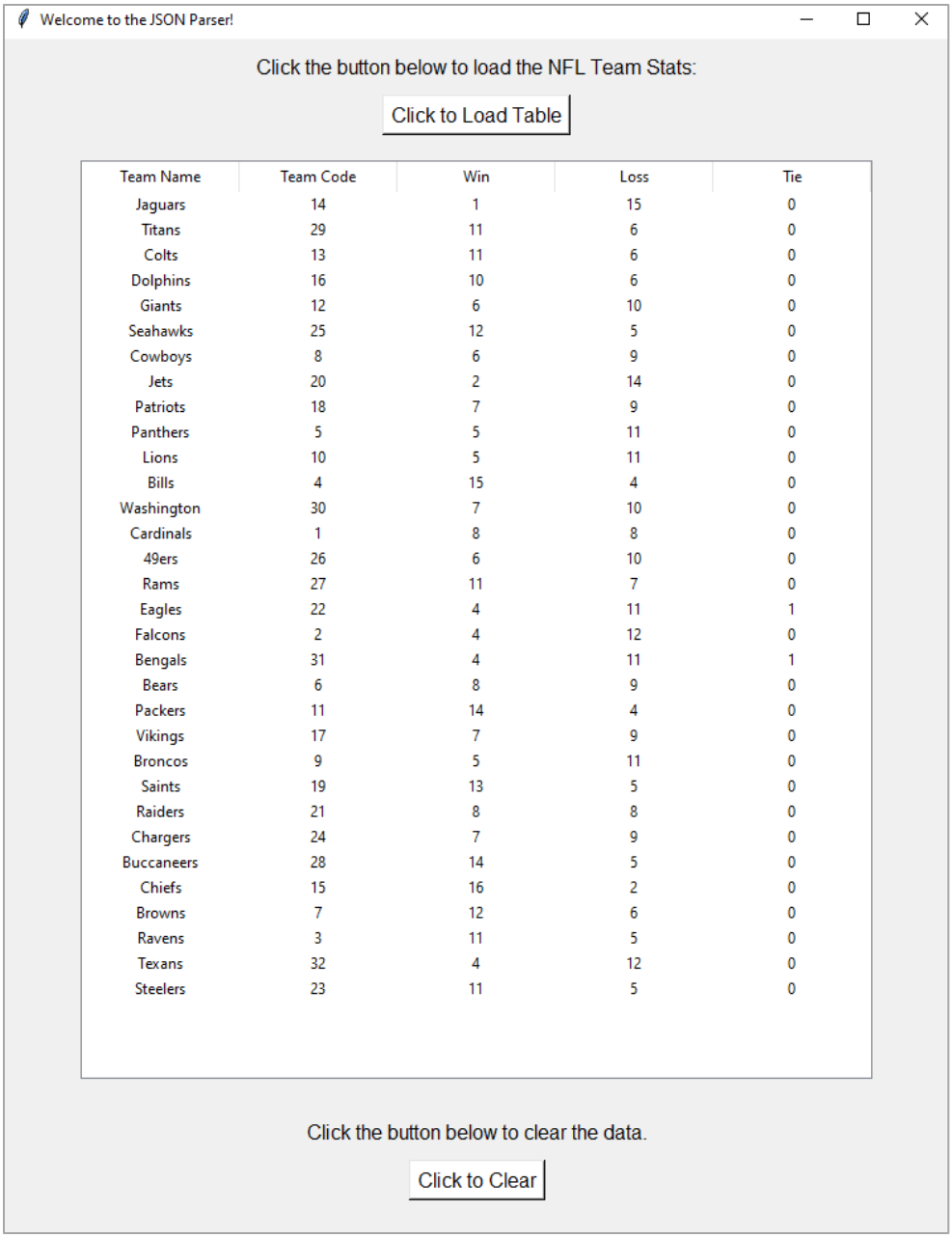

The json.dll file was compiled through the terminal using g++ and the following two commands:

```
C:\Users\jasmine\Desktop\JSONParser_Project3>g++ -c Main.cpp JSONParser.cpp FileHandler.cpp

C:\Users\jasmine\Desktop\JSONParser_Project3>g++ -static -shared -o json.dll Main.o JSONParser.o FileHandler.o /Users/jasmine/Desktop/JSONParser_Project3/libcurl.lib
```

When loaded successfully, the script loads the Tkinter GUI as shown below. The user is given the option to press two buttons. The first button ‘Click to Load Table’ can be pressed to load the table.



When the user presses the ‘Click to Load Table’ button, the getTeamInformation() function is called from the json.DLL library. This function iterates the teamName parameter in the Rest API and obtains all the team record information (same as Project #2). All data is then loaded into the table shown in the GUI. The output is shown below.



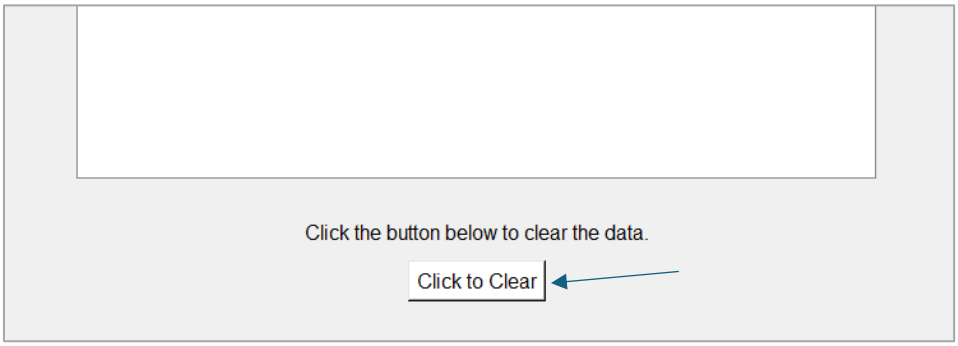
Team Name	Team Code	Win	Loss	Tie
Jaguars	14	1	15	0
Titans	29	11	6	0
Colts	13	11	6	0
Dolphins	16	10	6	0
Giants	12	6	10	0
Seahawks	25	12	5	0
Cowboys	8	6	9	0
Jets	20	2	14	0
Patriots	18	7	9	0
Panthers	5	5	11	0
Lions	10	5	11	0
Bills	4	15	4	0
Washington	30	7	10	0
Cardinals	1	8	8	0
49ers	26	6	10	0
Rams	27	11	7	0
Eagles	22	4	11	1
Falcons	2	4	12	0
Bengals	31	4	11	1
Bears	6	8	9	0
Packers	11	14	4	0
Vikings	17	7	9	0
Broncos	9	5	11	0
Saints	19	13	5	0
Raiders	21	8	8	0
Chargers	24	7	9	0
Buccaneers	28	14	5	0
Chiefs	15	16	2	0
Browns	7	12	6	0
Ravens	3	11	5	0
Texans	32	4	12	0
Steelers	23	11	5	0

The calculations for the season records are performed in the *getTeamRecords* function within the Main.cpp file (unchanged from the Project #2).

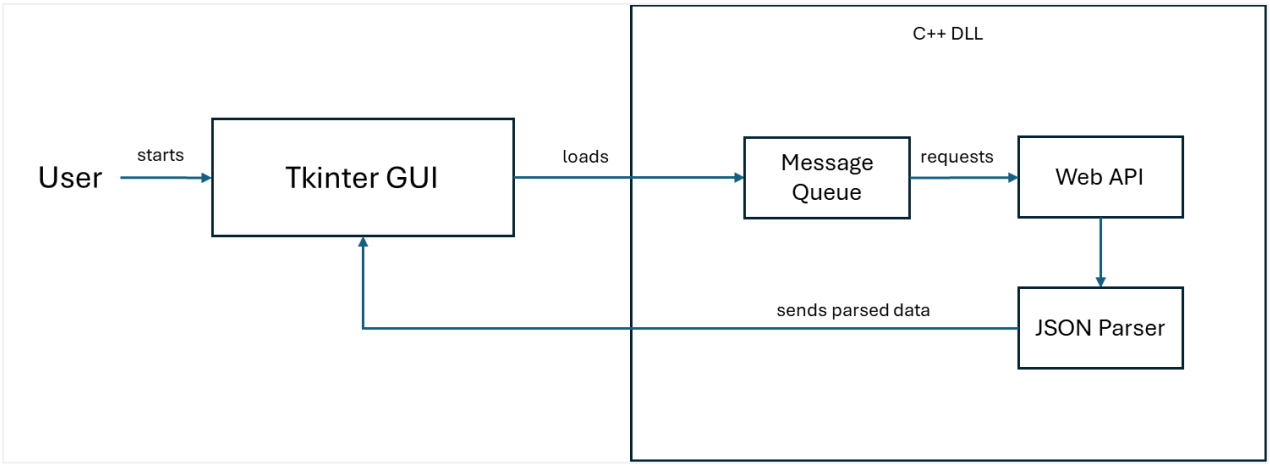
```cpp
std::unordered_set<std::string> games;
std::unordered_map<int, records> teamRecords;

for (auto game : data)
{
    if (games.find(game.visStats["gameCode"]) == games.end()) // if the game has not already been processed
    {
        int visitorScore = std::stoi(game.visStats["score"]);
        int visTeamCode = std::stoi(game.visStats["teamCode"]);

        int homeScore = std::stoi(game.homeStats["score"]);
        int homeTeamCode = std::stoi(game.homeStats["teamCode"]);

        //if the team has not already been stored, create a new records struct for storing data
        records visitorRecord = teamRecords.find(visTeamCode) != teamRecords.end() ? teamRecords[visTeamCode] : records();
        records homeRecord = teamRecords.find(homeTeamCode) != teamRecords.end() ? teamRecords[homeTeamCode] : records();

        visitorRecord.teamName = game.visTeamName;
        homeRecord.teamName = game.homeTeamName;

        if (homeScore > visitorScore)
        {
            homeRecord.win += 1;
            visitorRecord.loss += 1;
        }
        else if (homeScore < visitorScore)
        {
            homeRecord.loss += 1;
            visitorRecord.win += 1;
        }
        else
        {
            homeRecord.tie += 1;
            visitorRecord.tie += 1;
        }

        teamRecords[visTeamCode] = visitorRecord;
        teamRecords[homeTeamCode] = homeRecord;
    }
    games.insert(game.visStats["gameCode"]);
}
```

2. Additional Event Handler: Add an additional event handler – 5 pts

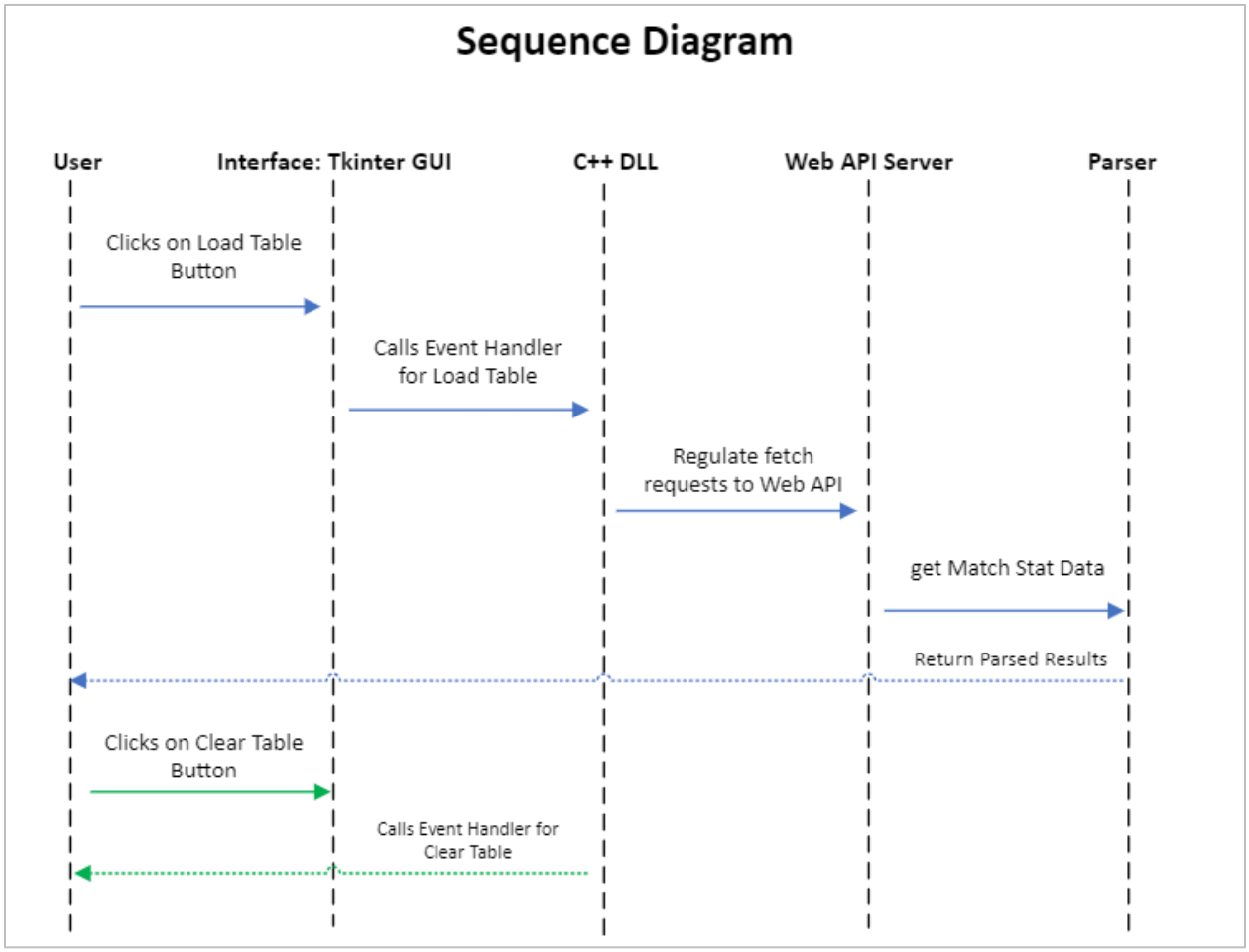
The additional event handler has been included in the jsonGUI.py file. The event handler code is in “button2_click” and is handled when the user clicks the “Click to Clear” button below. Upon click, the contents of the table are cleared.



3. Model Diagram:



4. Sequence Diagram:



5. UML Diagram: