## Match Analysis - Logic and Functional Structure Overview

The following is an outline of the program structure and the logic. It may be easier to get a feel for it here than scrolling though the code.

Logic >>

1. **\_\_init\_\_()**
   1. Create root
   2. Create the control variables
      1. IntVar
      2. StrVar
   3. Create and position the frame
   4. At end >> Call the function > **csv\_file\_create()**
2. **csv\_file\_create()**
   1. Get the paths for current file and define csv file name
   2. Define headers for csv file
   3. Check if file already exists
      1. If not exists > write the headers to the csv file
      2. Else >
   4. At end >> Call the function > **create\_widgets()**
3. **create\_widgets()**
   1. create widgets and text for date, time data and place using grid()
      1. Widgets = Label + Combobox
   2. Create widgets for him and away teams and place using grid()
      1. Widgets = Label + Entry
      2. Define stringvar() for home and away teams
   3. Add a separator?
      1. Widgets= separator?
   4. Create widgets for desc text, shots, free kicks and goals and place using grid()
      1. Call functions to update button clicks
         1. Call the function >>> **shots\_on\_target\_click()**
         2. Call the function >>> **shots\_off\_target\_click)**
         3. Call the function >>> **free\_kicks\_for\_click()**
         4. Call the function >>> **free\_kicks\_against\_click()**
         5. Call the function >>> **goals\_for\_click()**
         6. Call the function >>> **goals\_against\_click()**
   5. Add a separator
      1. Widgets = separator?
   6. Create finish button and desc text and place using grid()
      * 1. Call the function >>> **finish\_button\_click()**
        2. Widgets = Label + Button
   7. Create user guide button and place using grid()
      1. Widgets = Button
   8. resizing – columns 0 to 4 and rows 0 to 14
   9. At end >> Call the function > **realtime\_intvar\_values()**
4. **shots\_on\_target\_click() -** functionality when shots\_on\_target\_button is clicked
5. **shots\_off\_target\_click()-** functionality when shots\_off\_target\_button is clicked
6. **free\_kicks\_for\_click() -** functionality when free\_kicks\_for\_button is clicked
7. **free\_kicks\_against\_click() -** functionality when free\_kicks\_against\_button is clicked
8. **goals\_for\_click()** - functionality when goals\_for\_button is clicked
9. **goals\_against\_click()-** functionality when goals\_for\_button is clicked
   * 1. functions 5 to 9 all use intvar() .set() and .get() methods
10. **sports\_replace\_frame()** – functionality to replace initial data collection frame with new results frame which will display final stats
    1. open csv –
       1. read first and last rows and place in List
       2. use zip to make dictionary containing header(key) and last row(value)
    2. get rid of initial frame – use .destroy() method
    3. Create new frame, header and desc text and place using grid()
       1. Widgets = Label
    4. Add separator
       1. Widgets = separator?
    5. Create display strings for display data – build using strings values and dict values earlier created dictionary
    6. Create team date, ko time widgets and place using grid()
       1. Widgets = Label
    7. Add separator
       1. Widgets = separator?
    8. Create shots, free kicks and foals widgets and place using grid()
       1. Widgets = Label
    9. Create close button, that can close and exit the application and place using grid()
       1. Widgets = Button
    10. resizing – columns 0 to 4 and rows 0 to 9
11. **finish\_button\_click()** - functionality when Finish button is clicked
    1. Get final data from widgets using .get() method on different widget types (combo and entry widgets
    2. Write that data final data and also the data from the button clicks to our csv file, ‘match\_analysis.csv’ for persistent storage, future reference and analysis.
    3. At end >> Call the function > **sports\_replace\_frame()**
12. **realtime\_intvar\_values()** – create widgets to display realtime click button updates, so user knows it is working and can ser real time click data
    1. Uses textvariable attribute > calling earlier defined stringvar() and intvar()
13. **user\_guide\_button\_click()** -functionality when click help button
    1. Get file and path name for ‘user\_guide\_match.txt’ and assign to variable
    2. Define user guide window, title and size
    3. Open user\_guide and assign content to variable
    4. Define text widget and insert user\_guide.txt content from variable, read only
    5. Widgets = text
14. **close\_button\_click()** – functionality to close the application
    1. quit the python program and destroy the root gui window
15. **start\_match\_analysis\_app()** – function to call root.mainloop() and start the tkinter event loop

## Planet Travel Time - Logic and Functional Structure Overview

The following is an outline of the program structure and the logic. It may be easier to get a feel for it here than scrolling though the code.

1. **\_\_init\_\_():**
   1. Create root, frame, header and desc text and place using grid()
   2. Add a separator
   3. Widgets = **Labels** and **separator**
   4. At end >> Call the function > **create\_planet\_and\_transport\_widgets()**
2. **create\_planet\_and\_transport\_widgets()**
   1. defined stringvar() variable and created radio buttons for each planet and place using grid()
   2. defined stringvar() variable and created radio buttons for each mode of travel and place using grid()
   3. add separator
   4. widgets = **radiobutton** and **separator**
   5. At end >> Call the function > **create\_calculate\_userguide\_button()**
   6. At end >> Call the function > **create\_datasets()**
3. **create\_calculate\_userguide\_button()**
   1. creates calculate and userguide buttons and places them using grid()
   2. adds seperator
   3. Widgets= **Button**
4. **create\_datasets()**
   1. Uses 2 provided csv files which contain needed data
   2. Reads csv files, creates two lists for each csv file, then zips them, creating two dictionaries one for speed and one for distance
   3. At end >> Call the function > **create\_time\_data()**
5. **create\_time\_data()**
   1. creates ‘nested dictionary’ which contains all calculated values that are needed at run time.
6. **calculate\_button\_click()** – functionality for what happens when calculate button is clicked
   1. At start >> calls the function >> **create\_results\_widgets()**
   2. Gets current value of string var for planet and travel using .get()
   3. Gets ‘time’ value from nested dictionary based on those two get() values
   4. Creates some labels to display travel result in hours/days &years
   5. Widgets = **Label**
7. **User\_guide\_button\_click()**
   1. Functionality for what happens when user guide button is clicked.
   2. Calls ‘user\_guide\_planet.txt’ file which should be alongside the .py file.
   3. Define user guide window, title and size
   4. Open user\_guide and assign content to variable
   5. Define text widget and insert user\_guide.txt content from variable, read only
   6. Widgets = **text**
8. **close\_button\_click()** – functionality to close the application
   1. quit the python program and destroy the root gui window
9. **start\_planet\_travel\_time\_app()** – function to call root.mainloop() and start the tkinter event loop

## Gravity - Logic and Functional Structure Overview

The following is an outline of the program structure and the logic. It may be easier to get a feel for it here than scrolling though the code.

1. **\_\_init\_\_():**
   1. Create root, frame, header and desc text and place using grid()
   2. Add a separator
   3. Widgets = **Labels** and **separator**
   4. At end >> Call the function > **create\_gravity\_widgets()**
2. **create\_gravity\_widgets()**
   1. created Label and Entry widgets and placed them using grid()
   2. defined stringvar() and intvar() here also. Set default values for them aswell.
   3. add separator
   4. re-sizing
   5. widgets = **Label, Entry** and **separator**
   6. At end >> Call the function > **create\_calculate\_userguide\_buttons()**
3. **create\_calculate\_userguide\_buttons()**
   1. creates calculate and userguide buttons and places them using grid()
   2. adds seperator
   3. Widgets= **Button and separator**
4. **Calculate\_button\_click()** – functionality for what happens after calculate button is clicked
   1. Got latest values for name, mass and distance variables using get()
   2. Some error handling to deal with null or ineligible values
   3. Calculated gravitational force using variables
   4. Displayed resultant calculation along with descriptive string using Label
   5. Created a Close button to allow us close the app when finished
5. **User\_guide\_button\_click()**
   1. Functionality for what happens when user guide button is clicked.
   2. Calls ‘user\_guide\_gravity.txt’ file which should be alongside the .py file in the same folder.
   3. Define user guide window, title and size
   4. Open user\_guide and assign content to variable
   5. Define text widget and insert user\_guide.txt content from variable, read only
   6. Widgets = **text**
6. **close\_button\_click()** – functionality to close the application
   1. quit the python program and destroy the root gui window
7. **start\_planet\_travel\_time\_app()** – function to call root.mainloop() and start the tkinter event loop