# Rocket Trajectory Visualisation Tool – Future Development

Report prepared by Heather Leyssenaar

Recommendations are based on work completed by project developers:

Luke Gerschwitz,

Alyssa Yeo,

David Galbory, and

Heather Leyssenar.

**Table of Contents**

[Rocket Trajectory Visualisation Tool – Future Development 1](#__RefHeading___Toc738_528554679)

[Introduction 2](#__RefHeading___Toc740_528554679)

[Current Project Structure 3](#__RefHeading___Toc770_4069257068)

[Diagrams 3](#__RefHeading___Toc772_4069257068)

[Graph Creator 4](#__RefHeading___Toc1455_4069257068)

[GraphMenuSet 5](#__RefHeading___Toc1457_4069257068)

[Canvases 6](#__RefHeading___Toc1459_4069257068)

[GraphConfig and LargeDataDisplay 7](#__RefHeading___Toc1461_4069257068)

[Visualisation 7](#__RefHeading___Toc1463_4069257068)

[BarGraphSet 8](#__RefHeading___Toc1465_4069257068)

[VRMainCamera 9](#__RefHeading___Toc1467_4069257068)

[Adding New Datasets 10](#__RefHeading___Toc1469_4069257068)

[Future Steps 11](#__RefHeading___Toc495_621865907)

[Next Steps – Creating the dataset(s) 11](#__RefHeading___Toc742_528554679)

[Configuring Unity to display the graph based on new data 15](#__RefHeading___Toc744_528554679)

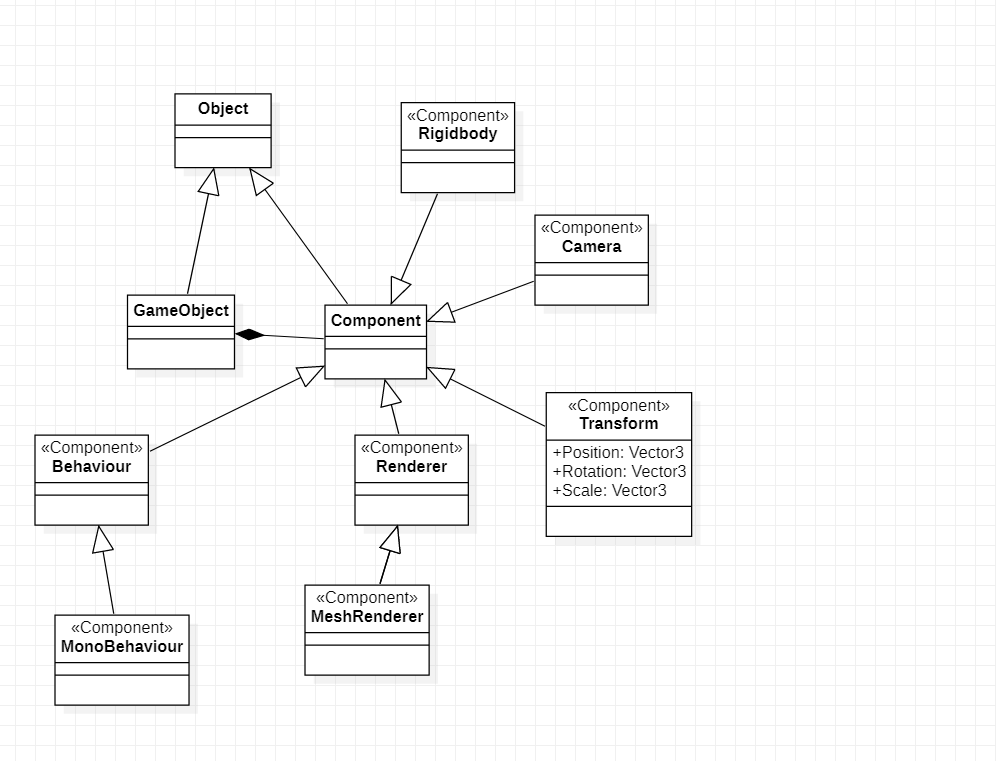
[Additional Work 16](#__RefHeading___Toc746_528554679)

###### Introduction

The Rocket Trajectory Visualisation project, developed for Swordfish Computing, has had the following major features added:

* Additional datasets that change show changes in alternative input variables (as opposed to only mass over time) have been added.
* The user can use the new datasets to generate new graphs at runtime, by accessing a user menu that is linked to the left controller
* Graphs can be deleted from the scene at runtime
* Multiple graphs can be spawned around the user, and displays are updated if a graph is deleted
* Graphs can be regenerated when axes have changed.
* New output variables are automatically read in to the project and can be displayed to a data panel
* Added support for different controllers (HTC Vive)
* The user can choose the axes of the graph via a menu panel
* The user can choose to generate a 2D or 3D scatter-graph
* The user can choose to generate a bar graph or a scatter-graph
* The user can highlight a trajectory with a slider and view updated information about that trajectory in the data panel and on the graph config menu.
* The information displayed on the data panel can be customised by using the data control panel and checking or un-checking the related variable checkbox. The data panel also has a slider control to view larger amounts of data.
* The user can configure existing graphs by using a configuration panel
* Most menu panels can be minimised to reduce screen clutter, and then brought back to original size
* Some quality of life features such as background colour, sound effects, movable data windows and loading time visual cues.

###### Current Project Structure



A set of UML diagrams that model the structure of the project can be found in the ‘V2 Diagrams’ folder in the project repository. There is a StarUML generated website that allows exploration of the project structure in user-friendly manner. Unity objects are arranged in a hierarchy.

Diagrams

The hierarchy of the scene starts with the parent object “Main”, with all objects contained within as it’s child objects.

The conventions used in the diagrams are:

- Root objects are in grey

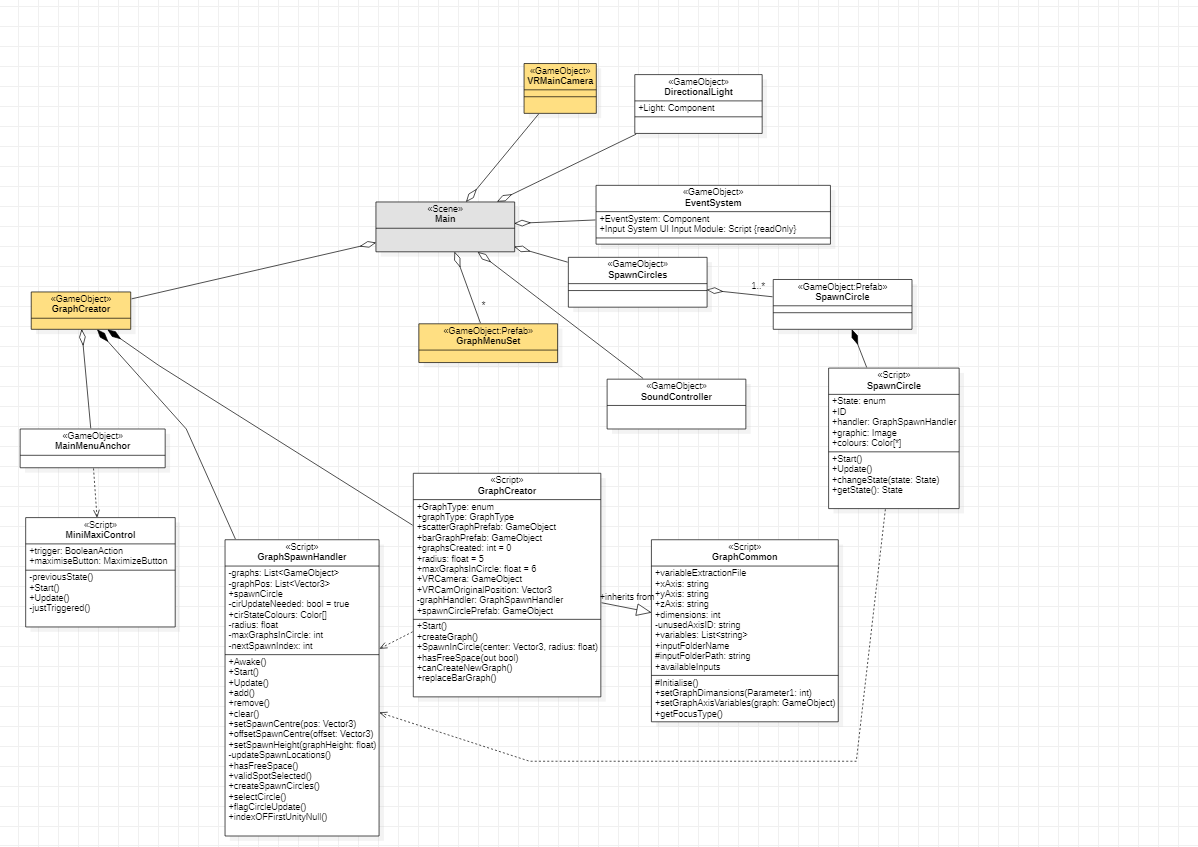
- Child objects that are expanded in another diagram are in yellow

- Parent/child relationships between objects have an aggregation relationship

- Object/component relationships have a composite relationship (this includes scripts that are used as components)

- Script dependencies on other scripts or objects are modelled as dependencies.

***Note that some connections have been omitted to simplify the diagrams.***



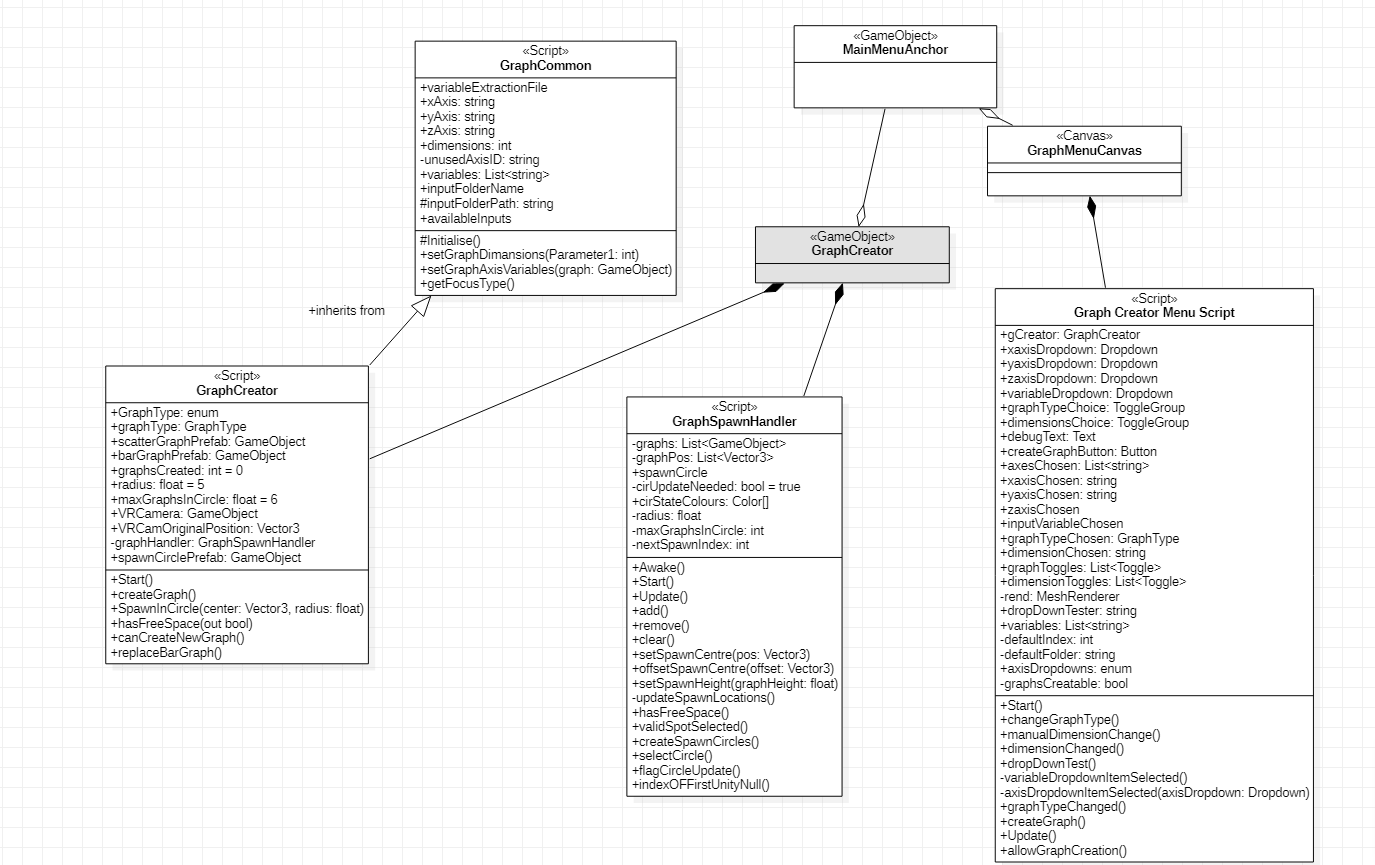
.

Graph Creator

The Graph Creator object is now in the scene upon start. It controls all aspects of the graph that will spawned by the user, such as the type of graph, whether it is 2d or 3d and what is plotted on the axes.

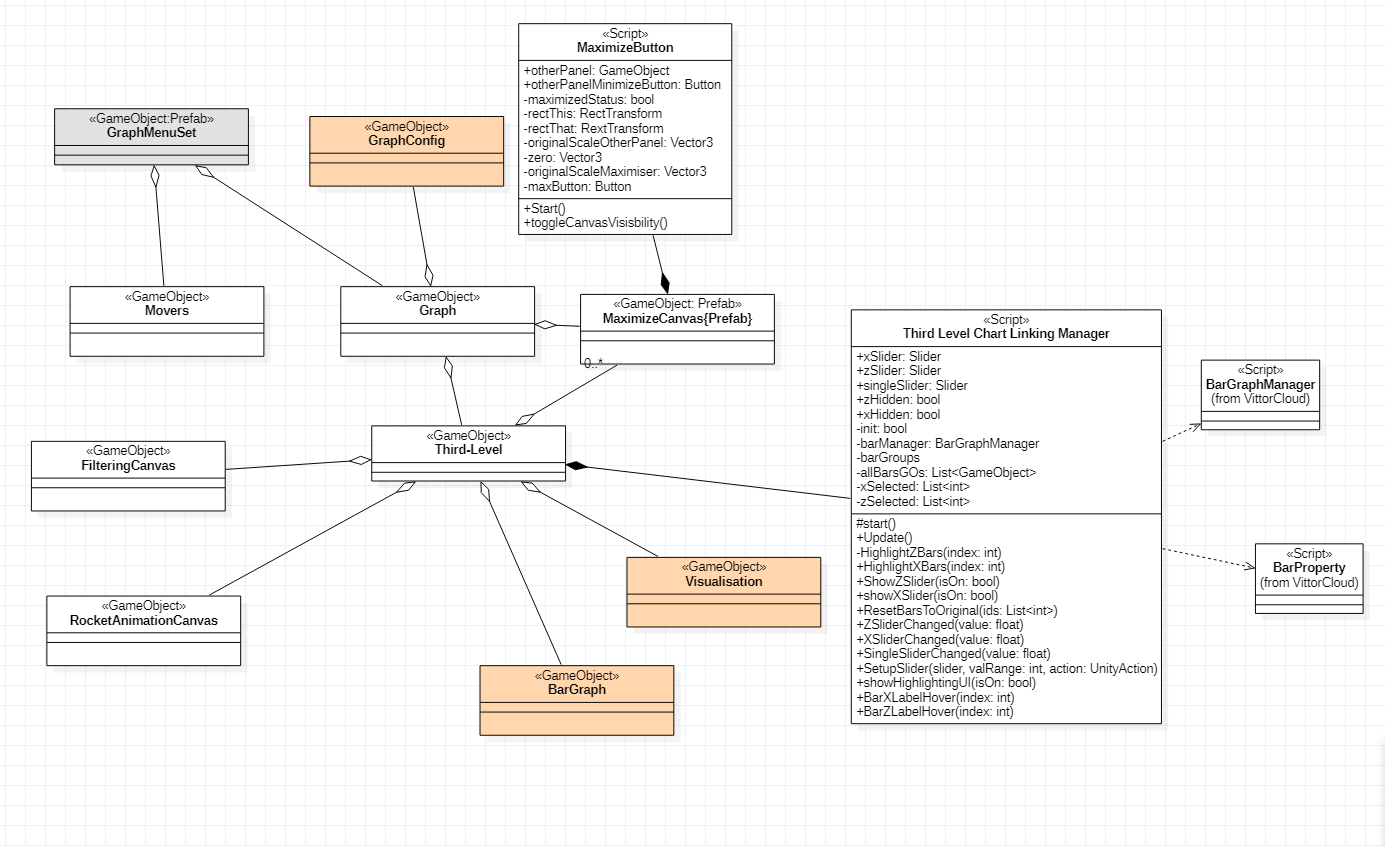
The MainMenuAnchor is used to place the graph creator menu in a certain position in the scene. The MiniMaxiControl script is used to allow the graph to be hidden and expanded, and also connects the left controller trigger to allow minimisation.

The SpawnCircles and their script allow the user to choose and view where the next graph will spawn.

.

GraphMenuSet

The GraphMenuSet is the set of graphs that is created by the user. There is no longer a set of graphs in the scene by default. The standard graph set is a scatter graph with a set of panels. A bar graph can now be spawned instead of, or along with the scatter graph.

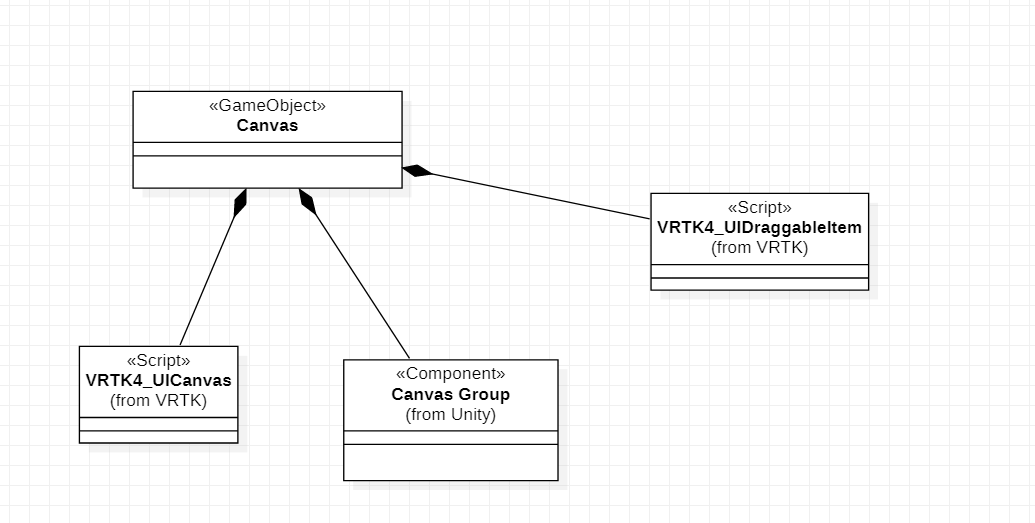


The GraphMenuSet includes the Graph object. There are also several Maximize Canvases that are optional, and can be used to reduce any canvas to a scale of (0, 0, 0), rendering them invisible and non-interactive.

The Third Level Chart Linking Manger is used to control all of the user interface controls in the highlighting manager, and some of the bar graph display properties. It requires the scripts “BarGraphManager” and “BarProperty” from the VittorCloud package to work.

Canvases

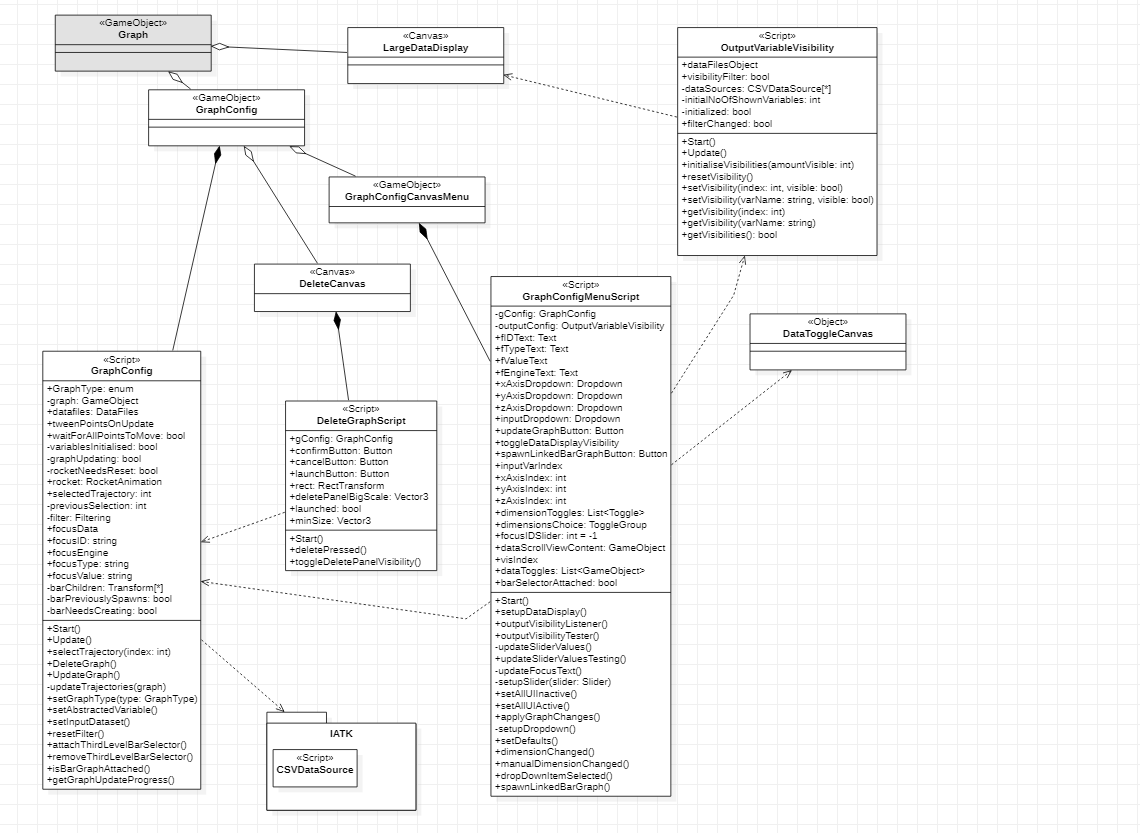
Canvases require the VRTK4\_UICanvas script from the VRTK package to work with VR controllers like the HTC Vive. They require the VRTK4\_UI DraggableItem script to work with the movers to allow grab control. If they contain a scroll view, the also require a CanvasGroup script to scroll properly.



GraphConfig and LargeDataDisplay

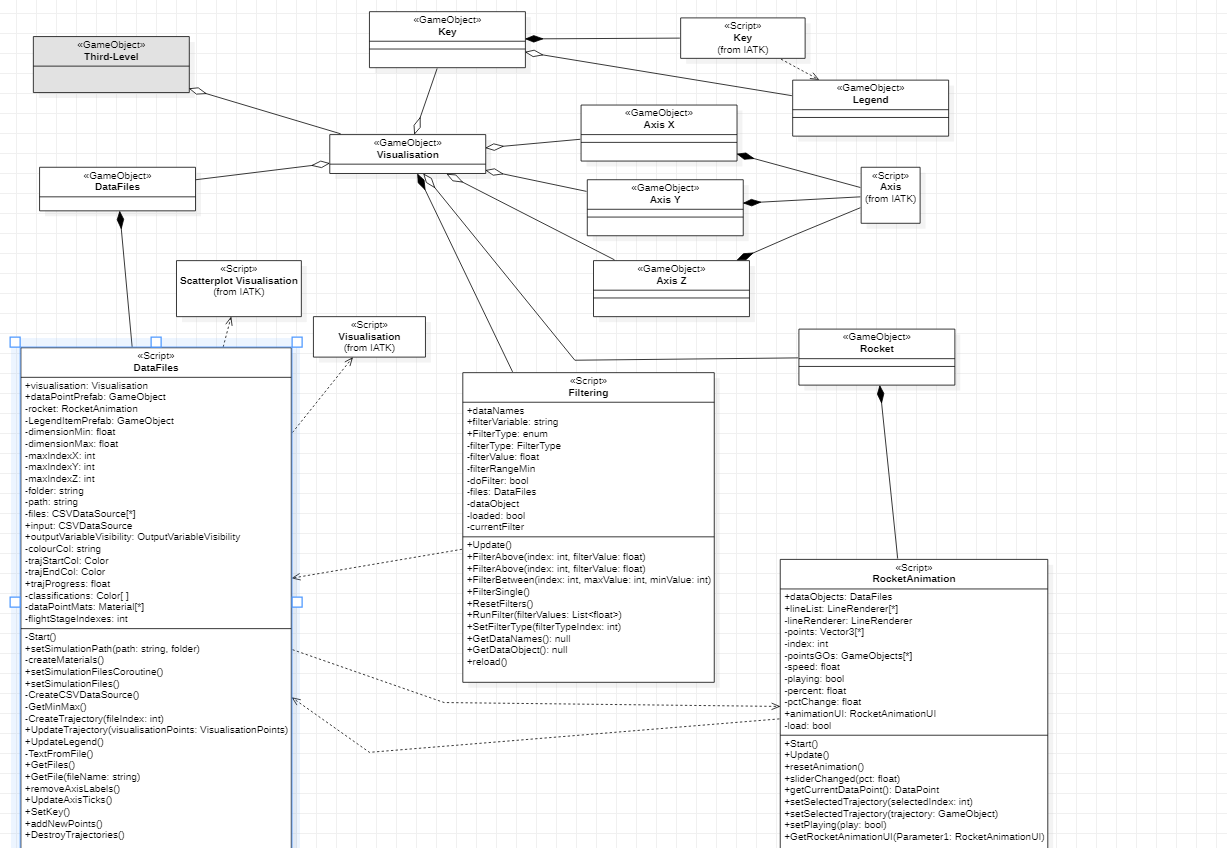
The GraphConfig object is used to configure any existing graph. Like the GraphCreator script, the GraphConfig script inherits many properties from GraphCommon.

The GraphConfigCanvasMenu is used to control the user menu, and calls upon the GraphConfig to update the graph. It also controls the DataToggleCanvas to allow user control over what output variables are displayed in the LargeDataDisplay. The LargeDataDisplay is controlled by the OutputVariableVisibility script.

.

Visualisation

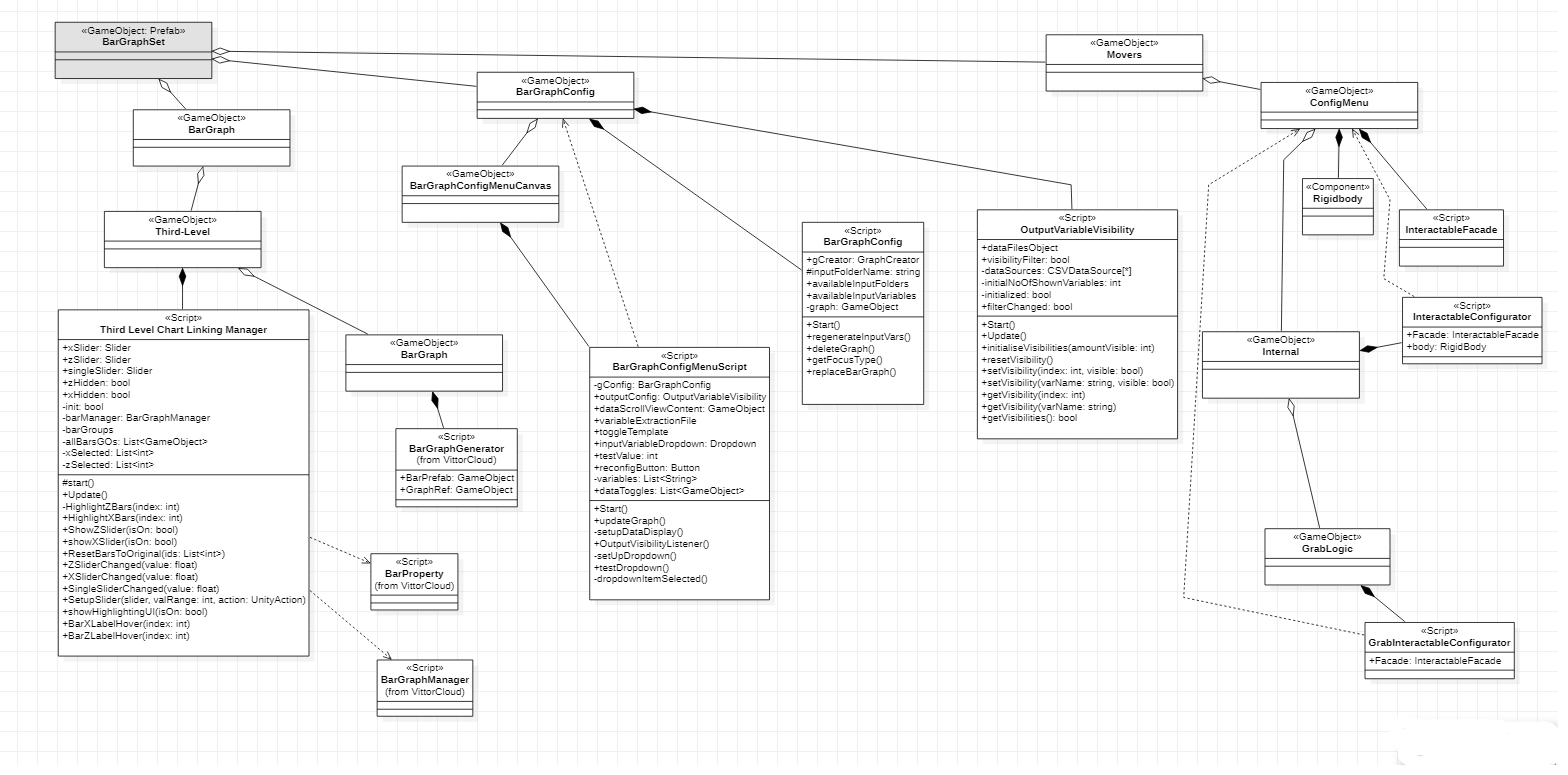
The Visualisation controls the way the graph is displayed. It requires a link to the DataFiles script. There are several scripts required from the IATK package, including Visualisation, ScatterplotVisualisation, Key and Axis. The Rocket Animation depends on the DataFiles script as well. The Key Script is used to render the legend (which will require a MeshRenderer component – not pictured).

.

BarGraphSet

The BarGraphSet has several objects and scripts that are similar to the GraphMenuSet. There is a BarGraphConfig object that requires the BarGraphConfig script and OutputVariableVisibility script. The BarGraph object requires the BarGraphGenerator script from VittorCloud. This script requires references to a Bar prefab object (the graph box prefab found in the VittorCloud prefabs has been used in this case). The Third Level Chart Linking Manager requires BarProperty and BarGraphManager scripts from VittorCloud as well.

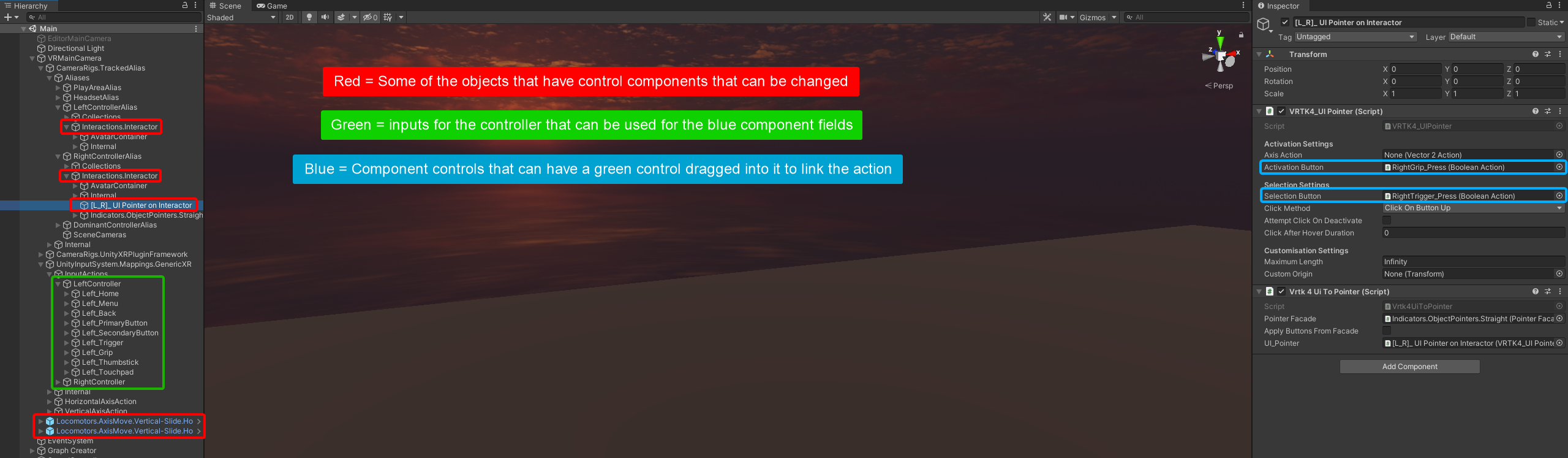
Also pictured here are the movers which are needed to allow the user to grab the menus



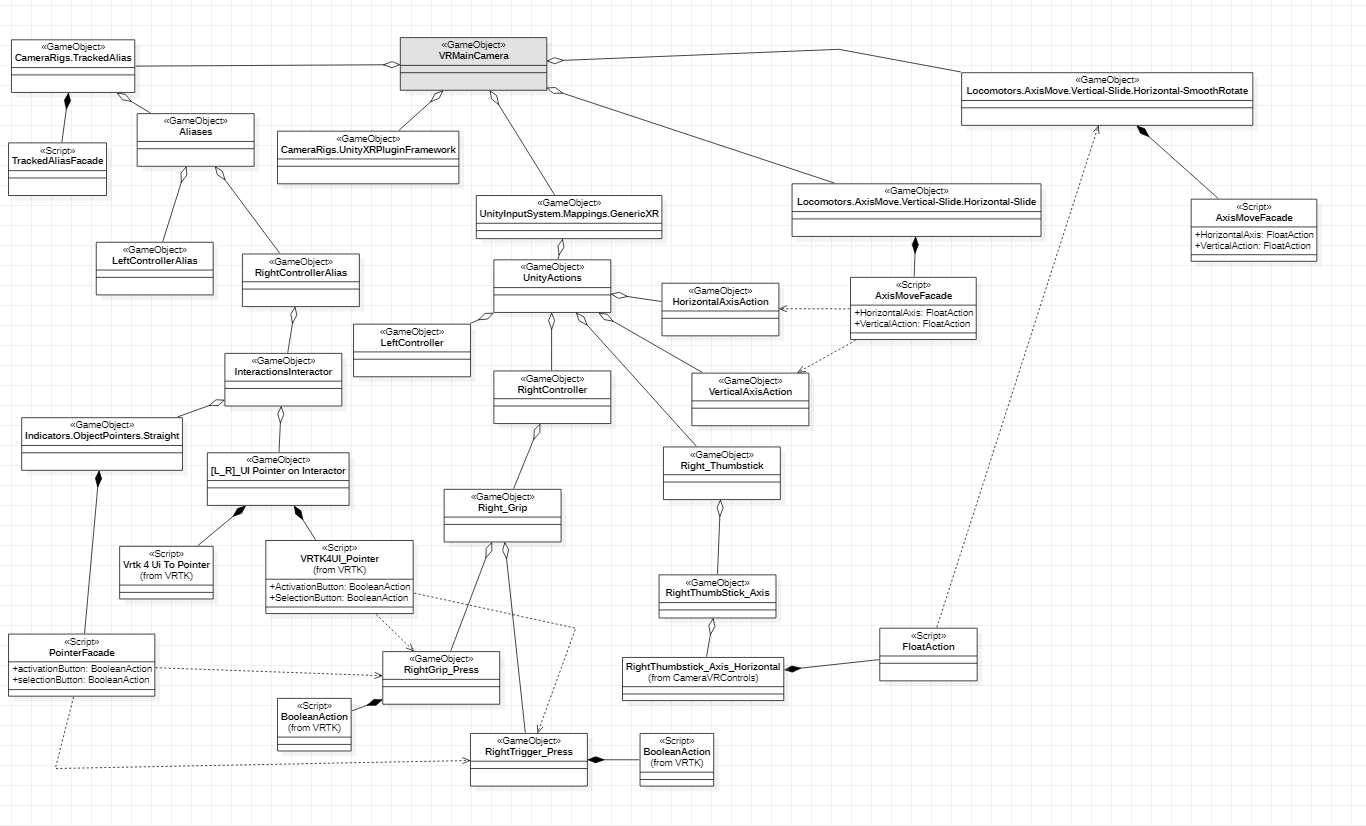
VRMainCamera

The VRMainCamera controls what the user can see while in the scene, and also has objects that control the VR input systems.

For every type of action, (Grip, move, select) a controller action will need to be mapped to it. Several scripts from VRTK and unity are required.

.

There is also a script that makes the pointer straight rather than curved. (PointerFacade).



###### Adding New Datasets

There are already six input types that can be used to generate new datasets (Rocket mass, burnout time, temperature, wind speed, propellant and nozzle distance). If these are sufficient then new datasets can be generated by running the file cell by cell in Google Colab.

1. Login to Google Colab with a google account

2. Click "Upload" on the pop-up window

3. Drop a copy of the python file "ColabRocketPyV2\_1.py", which can be obtained in the project repository under Assets/Resources

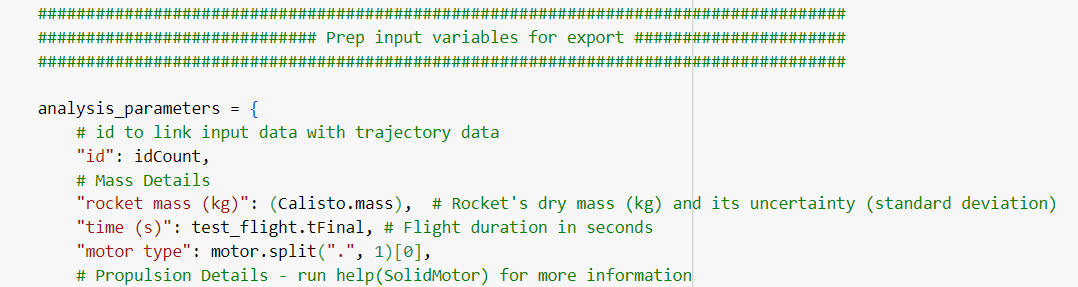
4. Ensure the source code from the RocketPy GitHub is added to the Google Drive folder from <https://github.com/RocketPy-Team/RocketPy/releases>. You will need version 11 to ensure compatibility.

5. Rename the source code folder to Rocket

6. The first cell requires access permission from Google Drive. Grant access.

7. Follow the instructions within the Colab file, running the cells one by one (Some cells can be skipped and are commented accordingly).

1. The output folder can be renamed before it is dropped into the project, but something appropriate to the variable being chosen is recommended, and keep the name relatively short so it can be viewed in the project’s menus.
2. To ensure compatibility with bar graph generation, all Motors need to be included and iterations should be limited to 6 per motor.
3. If you wish to choose a different variable from the given 6, you can skip the menu options cells, but you will need to adjust the file manually in order to get changes over the iterations. You can change the starting value, or leave it as it was. If the value only increases by 1 over every iteration, it is sufficient to add ‘+ iter’ to the end of the desired line. Otherwise you will need to define the step/increase value and add ‘ + (iter \* step)’ to the end of the line. You will also need to ensure that the variable you have chosen is included in the input variables that are exported to the all\_inputs.csv file. If it isn’t, you will need to add it to the analysis\_parameters dictionary in the final cell.



8. After the simulations have run, the folder with the data should exist within Google Drive.

9. Download the folder and extract it.

10. Add the folder to the AdditionalOutputs folder.

11. Map the folder name and the input value being changed in the VariableFocusMapping file found in the AdditionalOutputs folder.

###### Future Steps

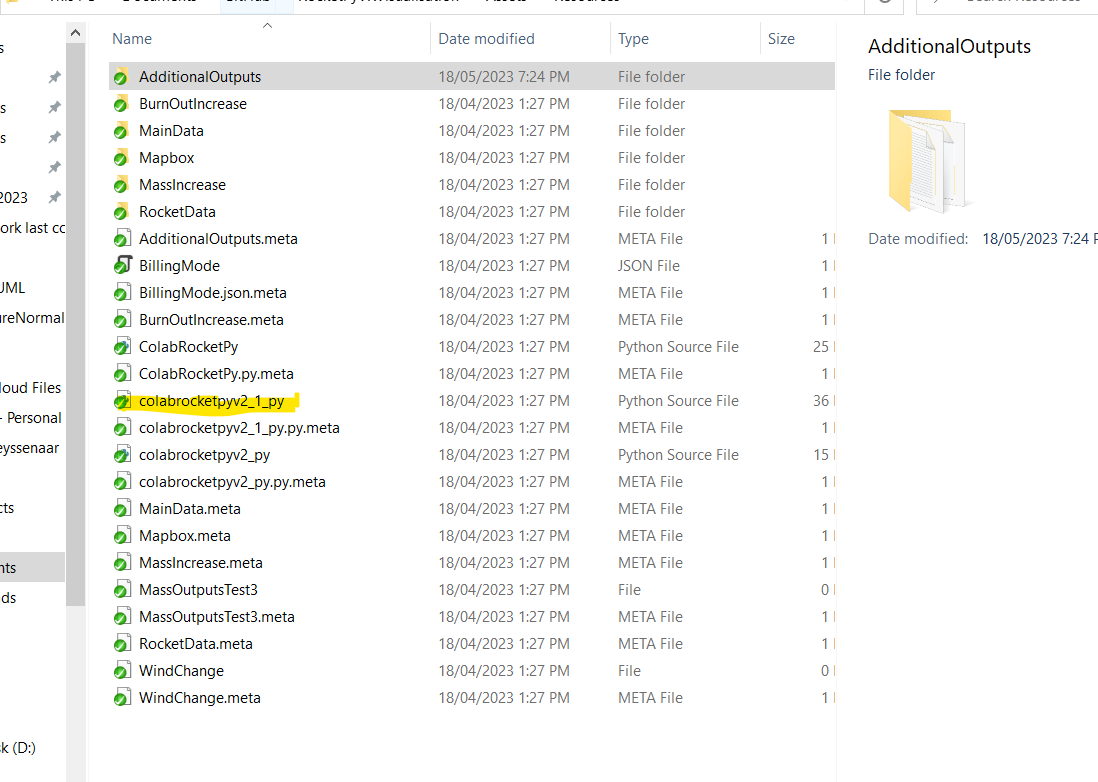
A potential future step was identified, which was adding another layer of abstraction to the graphs.

Applying [Ladder of Abstraction](https://worrydream.com/LadderOfAbstraction/) concepts to the project can be most easily achieved by adding datasets with additional variable changes for the user to select from the graph menu.

Rather than abstracting the change of a single variable over time, the source data can be generated with two (or more) input variables being changed for the rocket trajectory simulations.

###### Next Steps – Creating the dataset(s)

In the Assets/Resources folder of the project, the colabRocketpyv2\_1 file will give the quickest starting point. This file should be run in Google Colab to prevent dependency errors. Mount the file according to instructions that can be found in the manual. (The manual can be found on the GitHub repository for this project).

Figure 1: Location of Google Colab file

If you wish, you can copy or rename the file to preserve the older version.

Follow the steps laid out in the file, correcting any errors that may be raised by missing folders (instructions also found in the manual).

Run any cell that isn’t marked as optional until you get to choosing variable types. This can be done by clicking in the cell, then holding “Ctrl + Enter”.

Figure 2: Colab file first cell

Note on naming the folder: The folder name should indicate what two variables are being chosen, but due to space restrictions on the graph menu, the name should not be too long.

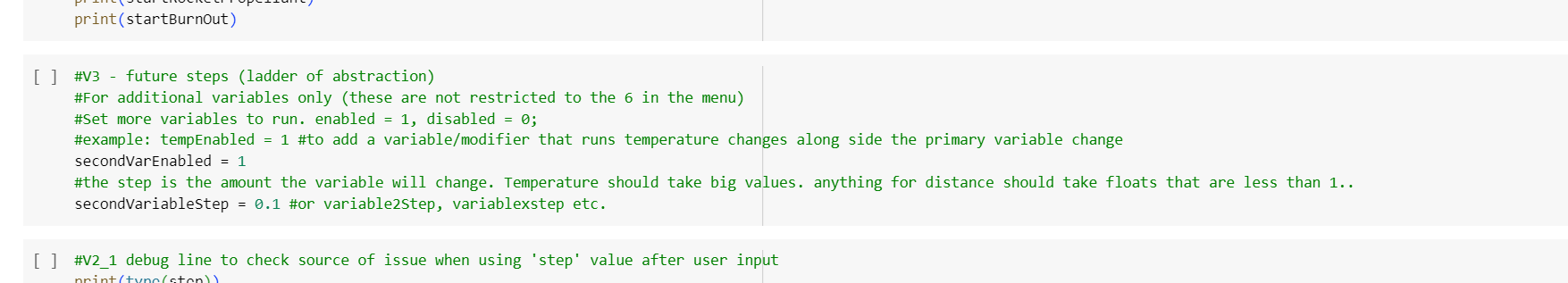
Something like “Mass\_Temp” would be ideal for increasing mass and temperature together, for example.

Run the cells that choose the initial variable, the starting value, and the step value (how much the value increases or decreases). Below these code cells, add a cell to the file for extra variables. An example is shown below. You will be adjusting these values manually, so be sure to choose appropriate step sizes (the amount the variable will change over the simulations).

While not strictly necessary, we are adding secondVarEnabled so that we can easily switch back to single variable changes without reverting all code changes. You may also wish to change the starting variable value. If so, add it here too.

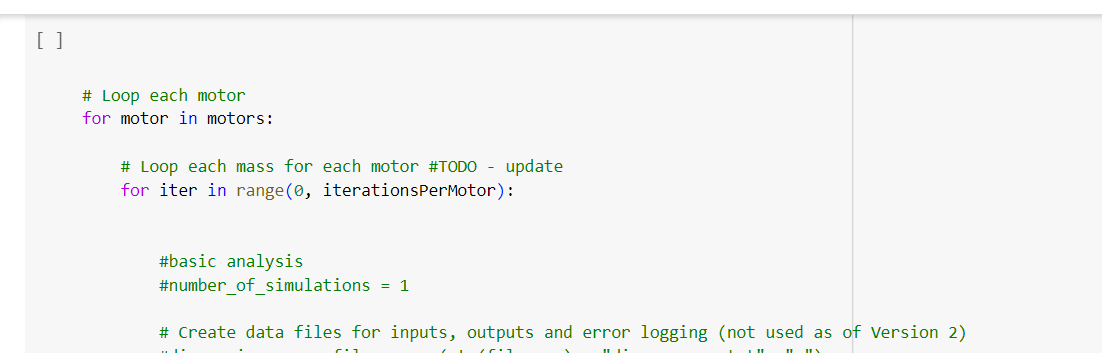
If you will be re-running the file with different variables to use as the second variable, it is a good idea to name secondVarEnabled as something more appropriate to the actual name, and set it to zero, so that you do not have to remove the extra code. This also applies to the step variable and the starting variable (if used).

.

Figure 3: Example code to be added to the file and then run

Then look for this section in the main simulation code (marked by #V1).

.

Figure 4: Main simulation loop code

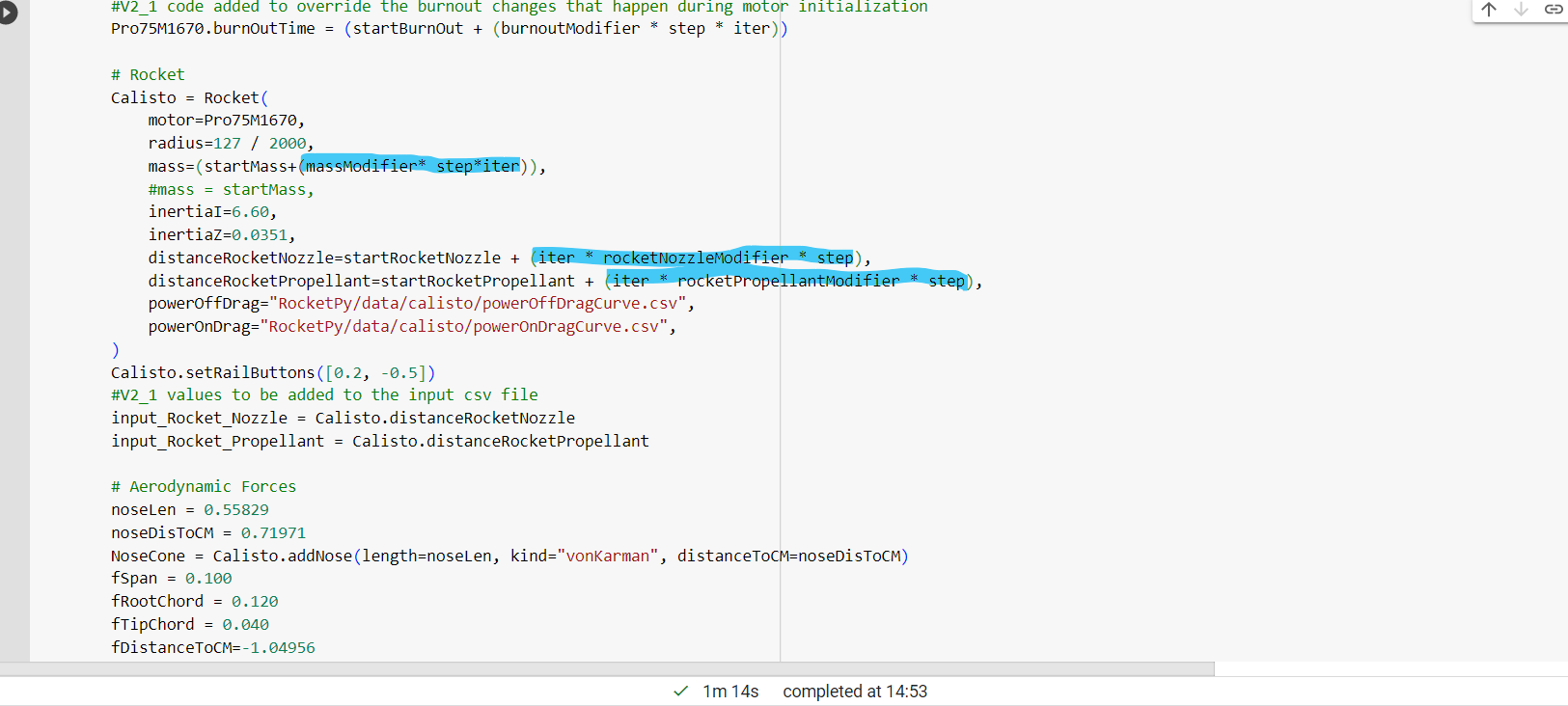
Go down to where the variables are defined.

Choose the variable to change and modify the line so that it will be increased

You can use any variable that can be found in the all\_inputs.csv file (There should already be one generated for Default\_Inputs in Assets/Resources/AdditionalOutputs if you need examples).

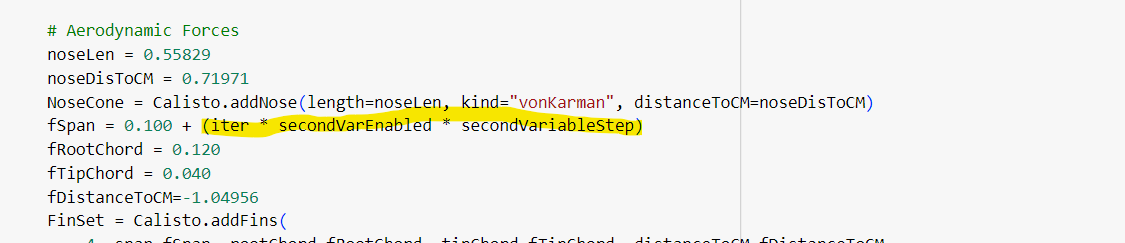
You’re going to add something like the blue highlighted section to your chosen variable.



Figure 5: Examples of existing primary variable that can be changed via the menu

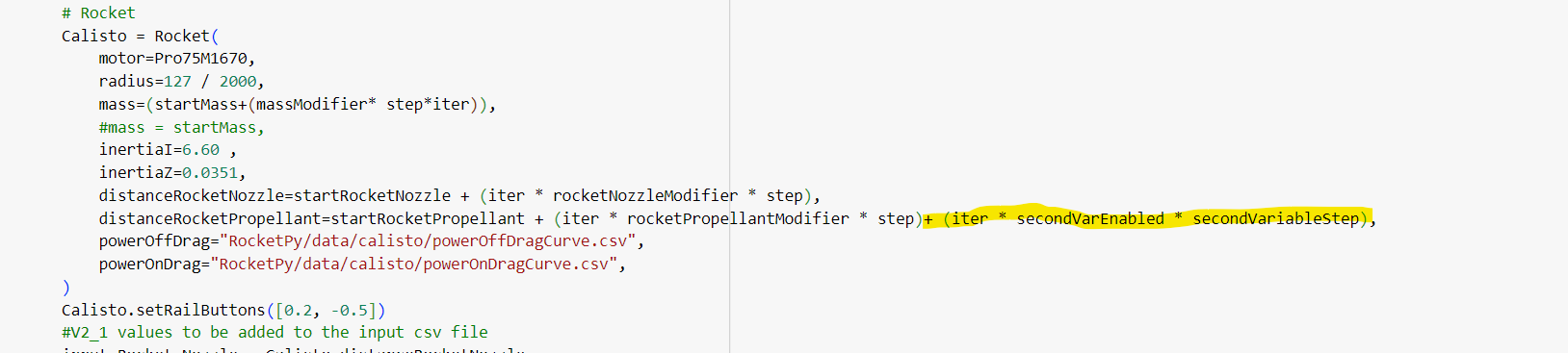
The adjustment will be to add (iter \* secondVarEnabled \* secondVariableStep) to the end to the line. The below is an example of choosing something already shown in all\_inputs.csv, but was not yet used as a changing variable.

.

Figure 6: Second variable (fSpan) that will be changed over the simulations

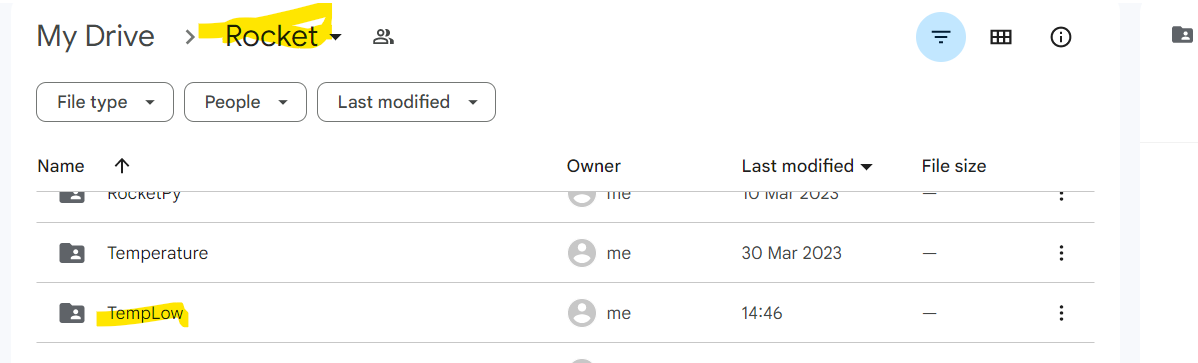
Alternatively, if you wish to use two of the originally changeable variables, run the cell with the menu to set up the primary variable, and add the following to the secondary variable:

.

Figure 7: Using a variable that was already possible to use as primary input as a secondary input

Run the cell. If correctly simulated (note that this will take a few minutes), you will find generated csv files in your Google Drive.

This folder can be downloaded and added directly to the AdditionalOutputs folder. (Be sure that the files are not compressed).

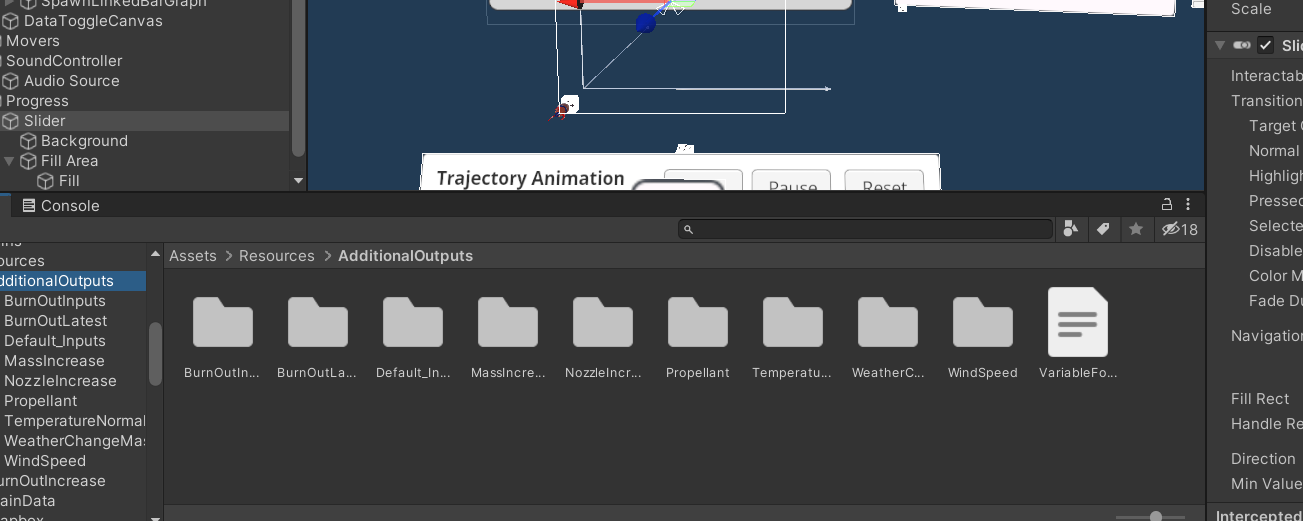
Figure 8: Where to find the generated files in Google Drive

###### Configuring Unity to display the graph based on new data

The Unity project already has code that checks for data in the resources folder. The graph can be generated and displayed by launching the program, then using the folder name that will show up in the Input Conditions drop-down menu.

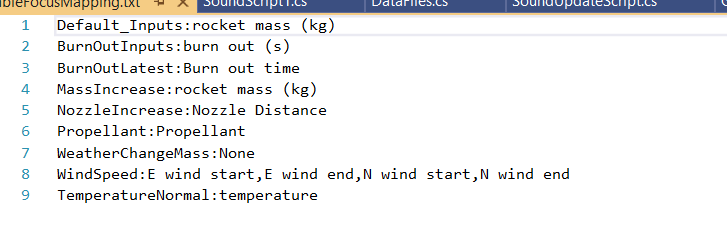
But to allow the graph config menu to track the variable changes, you need to go to the variable focus mapping text file found in the AdditionalOutputs folder.

.

Figure 9: Location of VariableFocusMapping file

Copy your folder name exactly and add it to the file.

.

Figure 10: Existing input folders mapped to their primary input variable

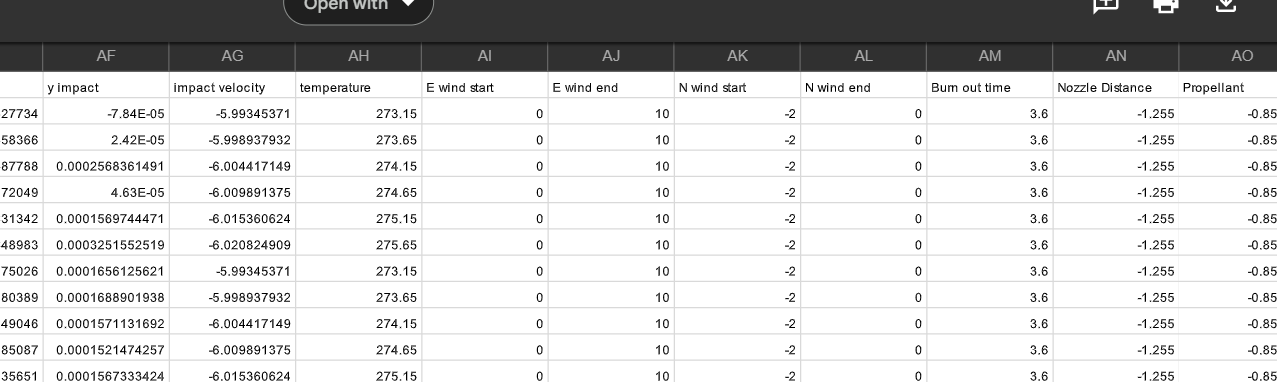
Then you need to match it to the input or inputs that are being changed.

Choose one of the two variables to focus on

(Future development may include having a secondary file to check for additional variables, but for now pick one).

You will find the exact column name in the all\_inputs.csv file.

.

Figure 11: One of the all\_inputs.csv files that has been generated

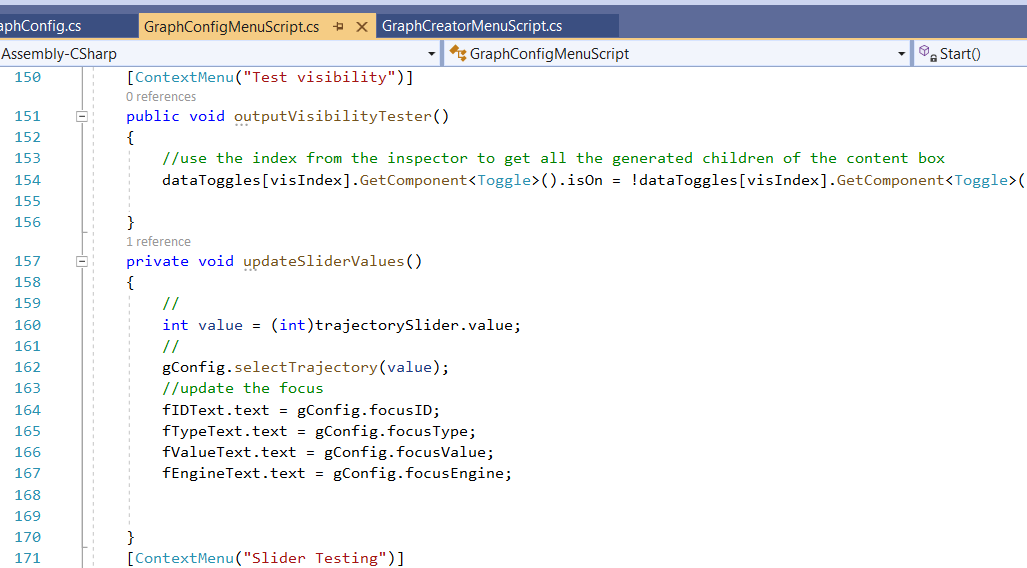
After a ‘:’, paste in the column name next to it’s folder name.

Now the variable details will be updated on the graph config menu panel when the trajectory changes.

###### Additional Work

The GraphConfig file (attached to every generated graph) accesses the trajectory data and extracts the focus data information from it.

This information is then accessed by the script that controls the graph config menu, GraphConfigMenuScript.cs. Shown below is the function “updateSliderValue”

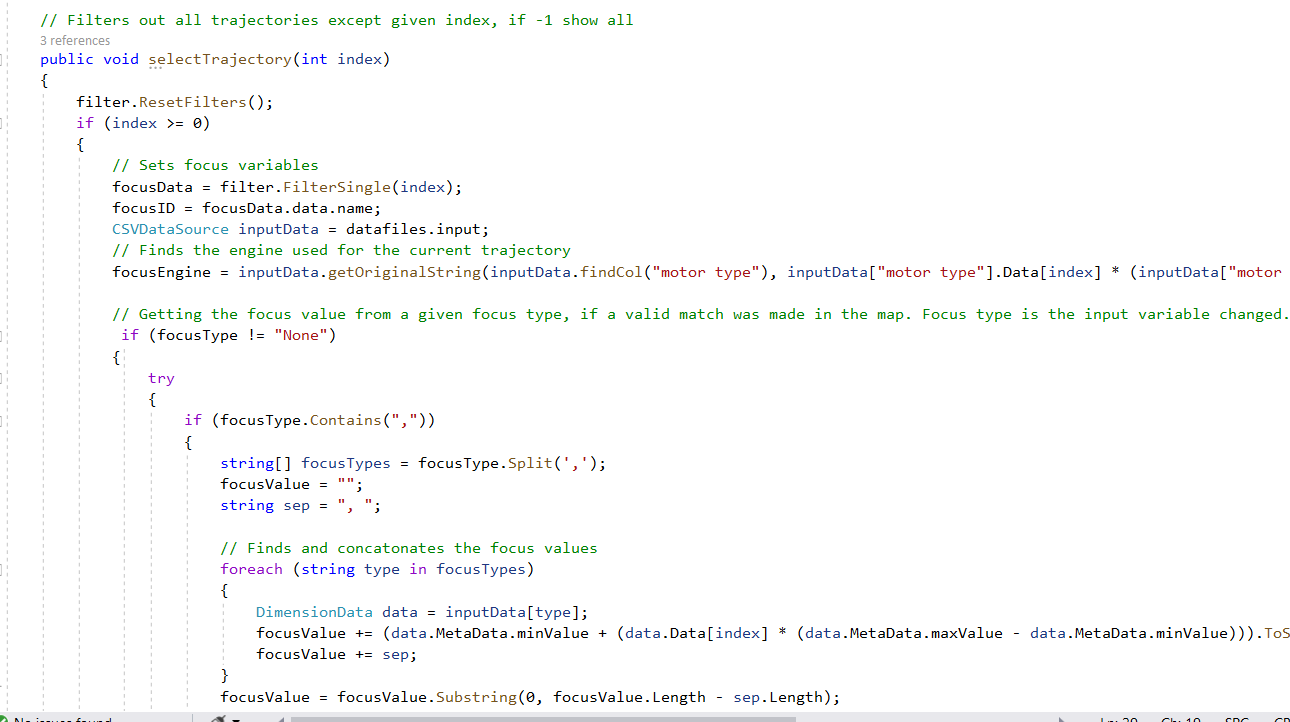
Figure 12: Where the menu gets the focus values from

At present, when the slider is moved, only the single variable listed in the VariableFocusMapping file is tracked and used to update the screen.

The function used to do that is selectTrajectory in GraphConfig. As seen below, the CSV data is parsed to get the values based on the trajectory index.

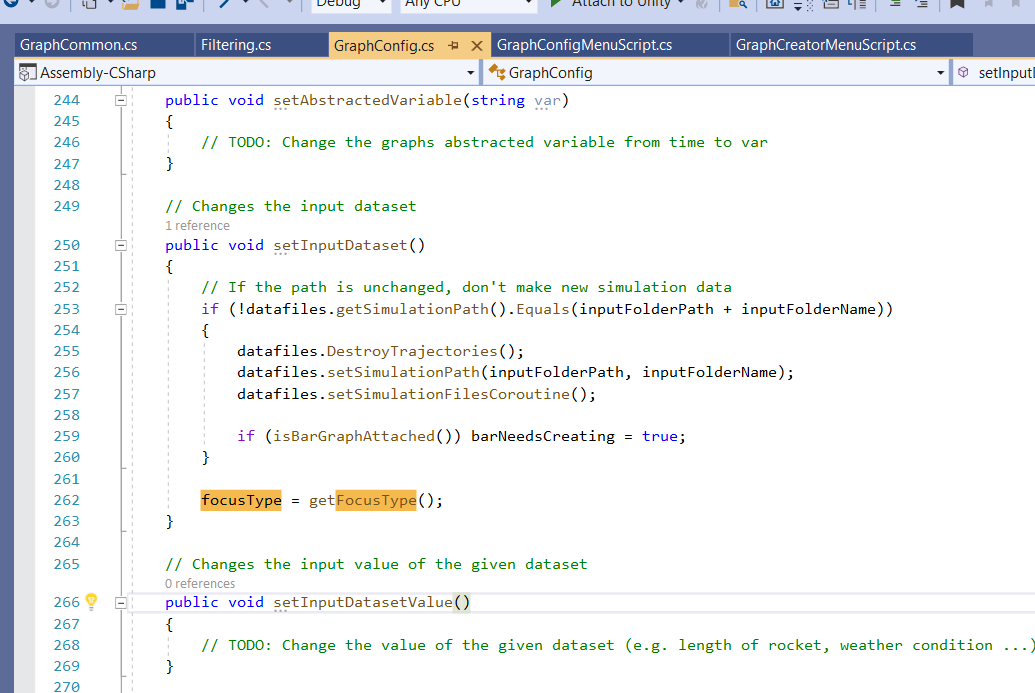
A possible next step would be to determine how best to display the secondary variable and it’s values to the screen when the slider is moved. This might mean checking a secondary file for the input folder name and the secondary variable.

.

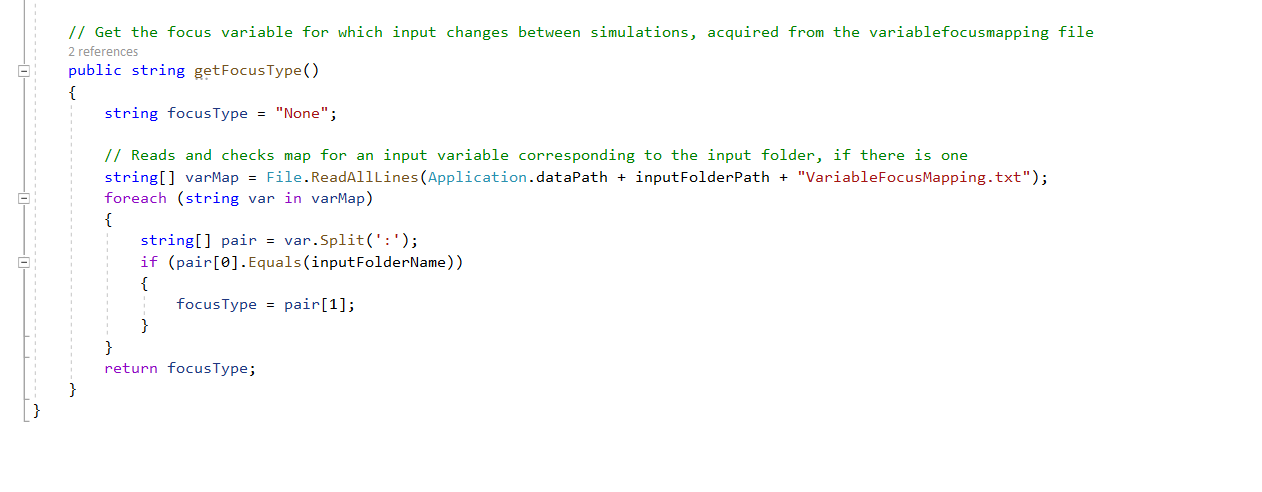
Figure 13: selectTrajectory method in the GraphConfig script

The getFocusType code is inherited from GraphCommon

.

Figure 14: Where the focus Type is called in from GraphCommon within setInputData

The code may, in the future, be adapted to retrieve a secondary focus type from either the variableFocusMappingFile or a new file created to hold secondary variable types.

Figure 15: getFocusType from GraphCommon

A more complicated task might be to connect multiple single variable datasets to the same axes and layer them over each other. All sets except the currently selected set would be hidden from sight. A click of a button could cycle through the set, allowing a better view of how changing the inputs will affect the outputs, as even small changed would be easier to detect. This would require more work with the IATK, and also possibly some custom work. This is a task that is not really necessary, since multiple datasets can already be displayed in the same scene for comparison.