How Our Code Does its Job

Simply put, our code takes a database approach to the problem. Data from text files is imported in the form of lists, interpolated into functions, and displayed in various ways. That’s the short version.

# Functions

## Stellar Evolution Simulator[solarMass, start , stop]

The main wrapper for our interface, this function takes the arguments of starting mass, start point, and end point, and initializes all of the necessary data and functions we need for the rest of the program (as global variables), as well as running our other main function. It also is fully standalone from the rest of our UI—it can be called on its own with those arguments and will run fine. The parameters were made to be optional, so if you leave out specific parameters, they will be set to defaults.

## Interface[start, stop]

Interface is where all of our code comes together. It uses a Manipulate inside of a DynamicModule. It takes the arguments of start time and stop time, and uses the global variables defined above. It returns the main output of our project.

## Initialization Functions

We have two functions in this category:

* initialize[filename]: Takes a string value (assumed to be a file name) and pulls that file from our database, turning the data into lists and associations and returning those.
* connect[lst,ist]: Takes two lists and returns a riffled, partitioned two-dimensional list of coordinates. Barely a function in its own right. However, it is extremely useful due to the shortening of the code that must be explicitly be written.

## Visualization Functions

These functions are used in our interface function to populate and organize some aspects of the visual:

* exterior[age]: Takes an age and returns the graphic for our external display. Its main purpose is to locate special cases (white dwarf collapse and planetary nebula) and display those when necessary. It does this with a Which statement.
* ColorFunction[min, max, radius]: Takes input radius and returns the color of the exterior sphere. It scales the color between the color values of yellow and red proportional to the range of radius between max and min. This isn’t necessarily a scientifically accurate way of determining color, but Dr. Carini said it was okay. If he shows interest in using this project in the future, we would change the scaling function to temperature, since that’s a more accurate way of predicting color.
* SphereRange[radius]: Takes a value for the radius of the external display graphic and returns a value for plotrange. It’s necessary to show accurate scaling for our external graphic, because stars often reach radii hundreds of times greater than that of their starting radius, and it’s impossible to show both sizes on the same scale. We made it as simplistic and obvious as possible so as to avoid confusing users as to how big the star actually is. It’s basically just an If statement.
* Core[age]: Takes parameter age and returns the set of disks defining the (visible) internal element composition. Again, very simple.

## UI

These functions run the UI around our code. While not necessarily important, we thought it would be good to mention their existence:

* homeScreen[]: Lets you input pre-specified masses for SES. It will pass this mass to the bounds function.
* bounds[num]: Takes as a parameter the starting mass of the star as chosen within homeScreen[]. Lets you input time ranges for the graphic.