Programming Project Report (Group 38)

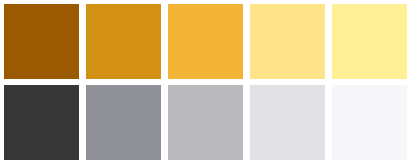
Michal Bronicki, Mark Doyle, Udit Jain & Hazel Hilbert

Introduction

As a group we used the 50k item data set containing information on space objects to create a graphical user interface, allowing an individual to interact and interpret the data.

Design:

We decided to keep the design simple with a grayscale color palette with orange accents.



The user interface consists of one screen split into two parts. On the left is a scrollable area with text displaying space object details along with buttons and search bars to filter the data. To the right are the visual representations of the data including additional object details, summary statistics, and numerous graphs.

The primary aim of our GUI is to represent the data and available queries in the most intuitive, user friendly way possible. In this way, users will be able to easily navigate and discover all the features of our application.

Group organization:

Work on the project was split between individual contributions and pair programming. We meet in person on average three times a week to work, in addition working occasionally at home. Breaking down the project most broadly between group members, Michal worked heavily on the backend (among many other things), Hazel focused on the user interface, and Mark and Udit coded many of the graphical representations (right portion of the screen).

The User Interface

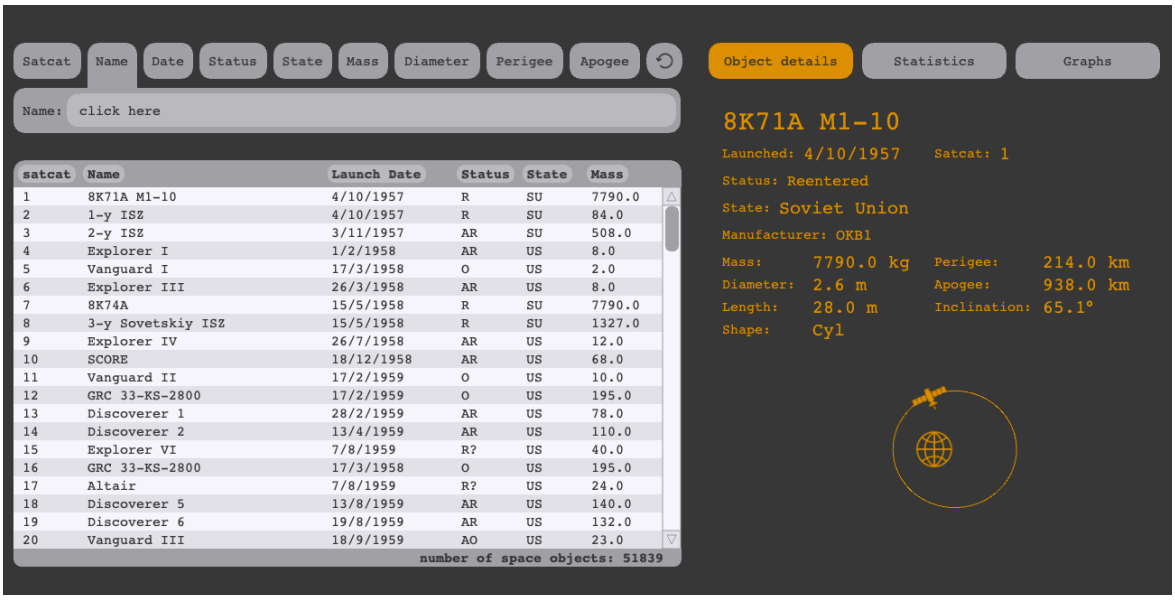
Text Area:

The text area displays information on 20 space objects at a time. To view more data, the list is scrollable, either with the scroll bar and mouse wheel, or by using the up and down arrows and keys.

Elements of the text area change color when hovered over, and when clicked, additional “Object details” are displayed. The Toal number of space objects (which vary when the data is filtered) is displayed at the bottom.

Another way in which the list is made interactive is by allowing the user to sort the data by clicking on the field headings. The data is sorted in both ascending and descending order (either alphabetically, numerically, or chronologically) depending on what field is selected.

Additionally, while the text area always displays each objects satcat, name, launch date, status, and state, an additional field showing mass, diameter, perigee, or apogee is displayed depending on what field is selected on the filter menu.



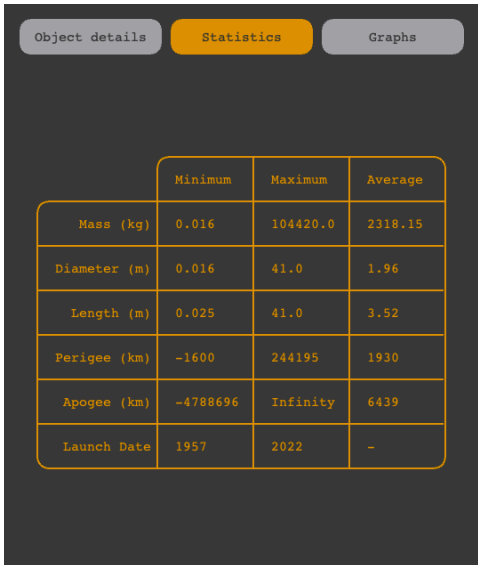
Filter Menu:

The filter buttons and the search bar work together to allow user to get the desired data they wants. For fields represented by Strings, any object that contains a substring equal to the input is displayed. For satcat, only the element with the exact satcat is returned. For the numerical fields, a range of values can be inputed (max and min) or simply an upper or lower bound.

The most important feature of the filter menu is that multiple filters can be applied to the data at once. To remove a filter, the user can click and enter nothing into the search bar of a single field, or press the reset button on the far right to remove all filters.

Object Details Screen:

Additional information on a particular space object can be viewed by chicken on dissed element on the list. Half of the text is animated giving a typewriter effect. An animation (for objects not in deep space or some other statuses) shows a satellite orbiting a central body. White the orbit is not to scale, effort was made on maintaining the ratio between perigee and apogee.

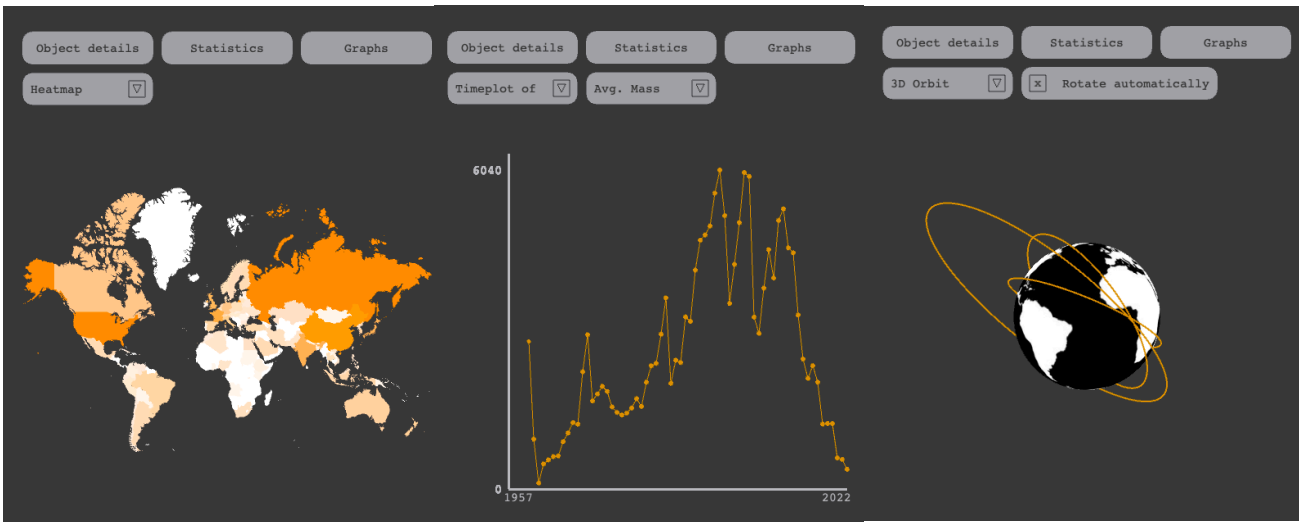


Statistics Screen:

The statistics screen displays summary statistics (min, max, mean) of various

fields- mass, diameter, length, perigee, apogee and launch date. Importantly, it changes according to the currently filtered data.

Chart Screen:



The user is able to choose various chart types from the drop-down menus. All graphs are based on the currently filtered data.

Frequency graphs showing the number of space objects per status and state are represented both as bar and pie charts. Both charts are interactive, displaying additional information when hovered over. (Segments of the pie chart shows the label and its percentage, bars show the label and number of objects).

Pie charts displaying the range frequency of several fields are also implemented. A time plot shows the average of different fields over the filtered period.

Graphical representation of 3D Orbits are displayed by clicking on space objects from the list. Multiple orbits can be displayed at once, and the user can chose to pan, or zoom in or out.

A heat map shows the number of space objects per country with darker oranges corresponding to more objects. Again, the user can filter the data to view an altered graph.

Implementation details overview

Widget classes:

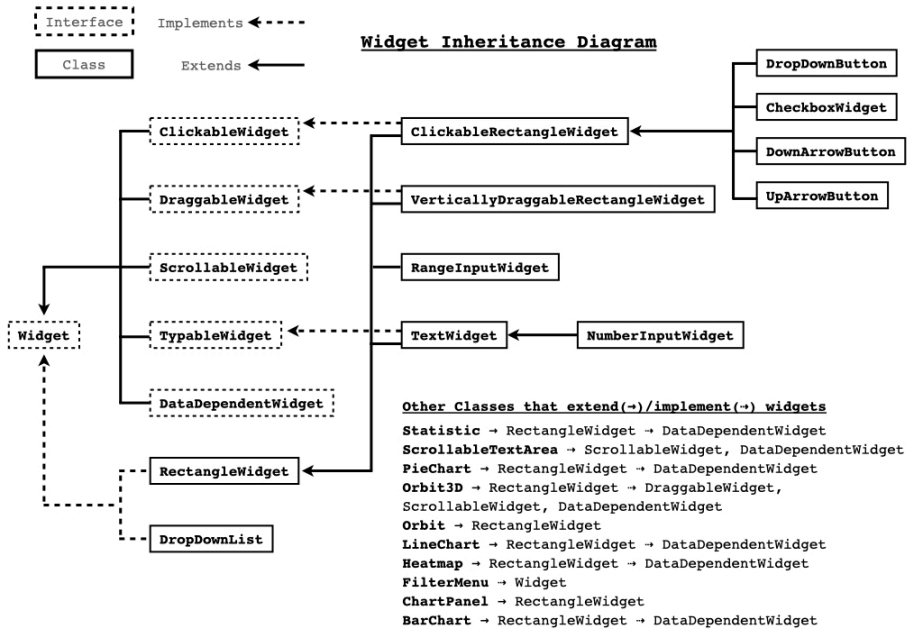
The implementation of a chain of linked classes (all implementing the Widget interface) is probably the most pivotal part of the codebase. These widgets enabled elements of the GUI to be easily implemented in a proper and interactable manner. A chart showing the written widgets and inheritance hierarchy is shown below.

The Widget interface requires every implementation to define the draw() method, responsible for drawing the widget to the screen. Widgets can have their own state and can interact with the user through the event system, further described below. This approach

allows the widgets to be self-contained in terms of their appearance and behaviour in response to events.

New widgets can be created either by means of inheritance (to extend the functionality of existing widgets), or from scratch by providing the implementation for methods declared in the interfaces. Fundamental to the extensibility of the taken approach is the ability to build up more complicated widgets by combining several smaller widgets together. This allowed us to create more generic, smaller reusable widgets, as well as decreased the code complexity and improved its overall readability.

Due to the fact that we aimed to make the basic widgets generic and reusable, they usually require a fairly large number of parameters. As using traditional approach with multiple overloaded constructors is not viable in that situation, several “Builder” classes were created in line with the builder design pattern.



Event Manager:

In order to maintain the code extensibility, we created an EventManager class responsible for handling various events in response to the user interaction.

Widgets that interact with the user have to implement specified interfaces (e.g. ClickableWidget interface). When such widget is put on the screen, a reference to it is added to a list stored in the EventManager object. Conversely, when such widget is being hidden, a reference to it is removed from the list.

The EventManager object is responsible for calling the appropriate method of each registered widget to “notify” that the event has occurred. The widget then handles the event. The behaviour of the widget in response to the event does not have to be fully specified

in the class definition — widgets can store non-final (reassignable) Callback objects which can be called in response to the event. This is the approach that is widely used in our design, as it further improves the code extensibility.

Data Provider:

Data manipulation is implemented mostly in the DataProvider class, responsible for reading the data from the file, filtering and sorting.

From each entry in the data file a DataPoint object is created. All the DataPoint objects are then stored in an ArrayList in the data provider.

The DataProvider class provides the option to easily filter the data. A setFilter method can be called with a parameter of Predicate<DataPoint> in order to filter the data points by any desirable criteria.

A setSortingComparator method (taking a parameter of type Comparator<DataPoint>) is also provided (along with predefined, final comparators for needed use cases) allowing to easily sort the data points by chosen criteria.

Data provider class also automatically calculates some statistics of the filtered data (such as averages, minimum and maximum values).

While some of the visualisation widgets access the DataProvider directly to fetch the necessary data, various classes implementing the ChartDataGenerator interface are used to automatically construct the data needed by some of the graph classes (e.g. labels and values for the bar graph).

Problems encountered:

Early on we ran into issues with the scroll bar once data began to be filtered. Bugs we had to fix include, prohibiting scrolling past the last element, jumping back to the top of the list when a new filter is applied, and removing the scroll bar when there are <= 20 items.

Once the data started to be filtered (alphabetically and numerical) we ran into issues when fields where left empty. We were able to fix this by converting numbers to Double.NaN and then representing them with a dash in the list.

An issue encountered with the 3D orbits was that the provided data set only included apogee, perigee, and inclination, which limited the accuracy of our model.

Additionally, we tried implementing a 3D heat map that could be used with the 3D orbits, however, the geographical projection of the SVG we used did not project well onto a sphere.