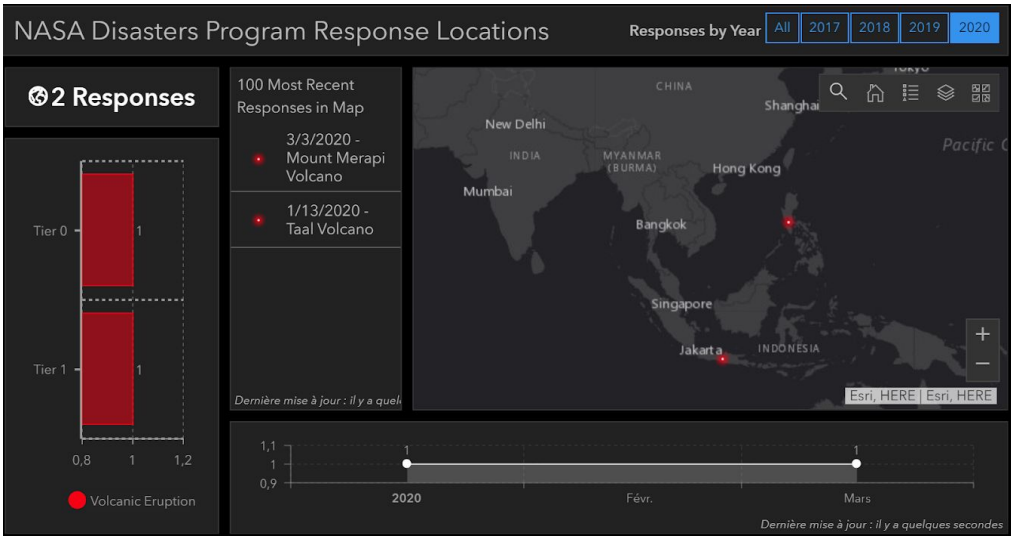
Data Visualization: Process Book

Leo Marco Serena - Loïc Karl Droz - Justinas Sukaitis

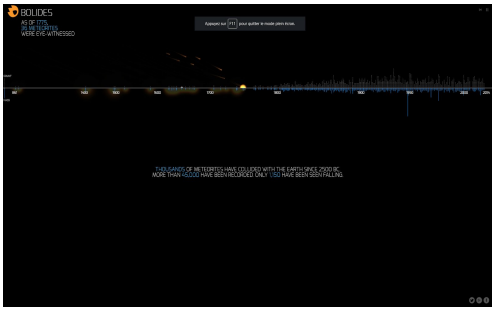
Main motivation

Our goal was to design an intuitive tool to analyse data sets relative to natural disasters with respect to their time period. We were influenced by many different ressources. First the NASA portal for volcanoes:



where we are able to get insight on a map about volcanoes with some statistics and time period information.

Then the two following videos, first a timelapse of nuclear detonations, and second another time lapse of meteor landings:



It was only natural to combine the two ideas and propose a way for the user to study a set of events *displayed on a map* but also select a *time period*.

Data Collection and Parsing

First milestone coming, we had to choose datasets of natural disaster events that held three main characteristics, namely **position** (longitude, latitude), **date of event** (esp. year) and **various data**.

We selected thus one dataset each:

- **Volcanoes** (Leo Marco Serena)
- **Earthquakes** (Justinas Sukaitis)
- **Meteors** (Loïc Karl Droz)

And gained some insight on what they contained. We parsed the data and made sure that the format was easy to use from a *.csv* file, and filtered it to keep only relevant informations. We also sorted all datasets in time ascending order.

For volcanoes, all unknown elements were replaced by *NaNs* for *d3* to ignore them. In the case a row presented several countries in the 'country' section in a single string, they have been replaced by a list of countries. 'Primary Volcanoes Types', 'Tectonic Setting', 'Elevation', columns were parsed to remove disturbing strings. To help with the implementation (since some years were below year 0), the years were replaced by their age with respect to 2018. 'Subregions' column was removed since unused.

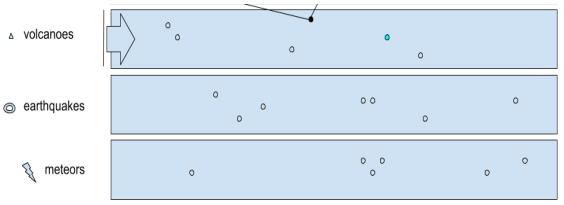
In Earthquakes, we transformed the date to keep only the year in order to unify the information with the other two datasets. Due to high percentage of NaN values in other columns, we only kept the values for the depth of the epicentre and the magnitude as well as the position and time.

For Meteors, we ignored all data points which were missing latitude-longitude coordinates since this was data required for our visualization. We spotted invalid coordinates by checking for legal latitude-longitude in degrees, and also ignoring values equal to "0, 0", as these were not valid according to the dataset explanation on Kaggle. We also ignored some data points which had invalid date information, which was also required by our visualization.

Design

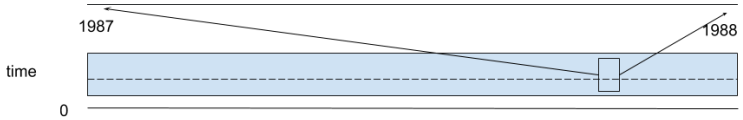
At this point, we needed to determine the practical visualization of our project. The most obvious was that we needed a map to display the position of the events. We decided to use two different projection styles that the user can choose by his will, since the geoNatural earth obfuscates the information of the distribution of points near the edges of the projection while the rectangular, though very common in our daily lives, distorts the curvature and thus the distribution of points near the center of the map.

We then came up with the idea that a time-related representation of statistics could be very convenient, so the rolls were designed. A y-axis for data attributes for each data type and depending on the current window on which the data points are, expressed on the x axis.



Clicking on a point would highlight all the related points on the map, meaning points contributing to the statistic and on the selected year.

As in the nuclear explosions/meteors examples, we wanted to make the window move by itself and display a nice animation. In order to link the rolls for each dataset and the map, we give the user the possibility to choose a point, that will then be highlighted on the map as well.



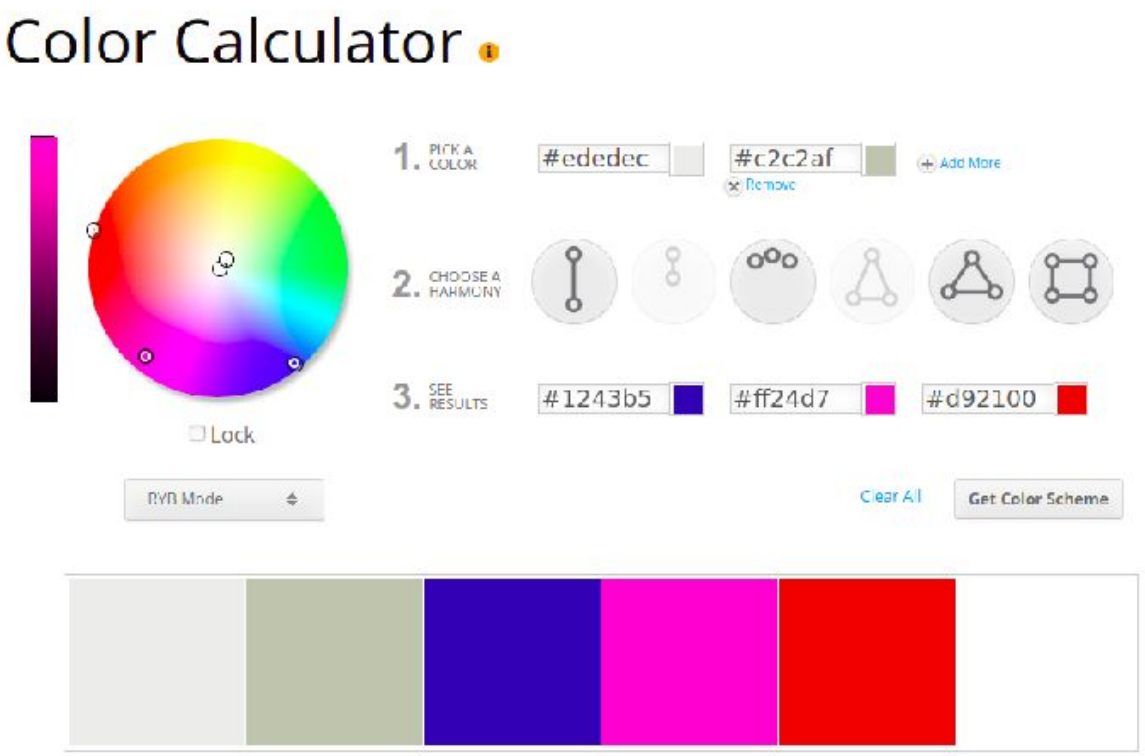
The next step was to design a way for the user to control the time they wanted to focus on.

We then designed a time window selector, draggable and with the ability to be resized. We found it necessary for the user to be able to pause the animation as well, in order to inspect every aspect of the visualization, we added the possibility to pause the animation.

To give the user more control over the data, we added the ability to filter the data and focus only on a chosen selection.

We also added a brush functionality on the map to allow the user select an area and gain some insight on the selected points with various informations.

In order to make different types of data easily distinguishable in our visualisation, we needed to select a triad of strong colors that would accentuate their distinctive type. For this we used a color wheel that allowed us to easily maneuver between our choices.



Note, in addition to the data needing three distinct colors, the background color also needed to be decided that would work with all three of the data colors.

Implementation

Before implementing each visualization element, we created the main layout of our webpage. It consists of two columns, each containing elements stacked vertically. The spacing between columns and between elements can be changed in the stylesheet.

A *node.js* server was up to deliver the *.csv* files to the front-end javascript files and facilitate development as we ran into some security issues with browsers trying to load local files from a webpage. The node.js server is of course not required to use our visualization website, only for convenient development.

Rolls

The first elements implemented were the **rolls**. They represented a real technical challenge as they needed to be updated with the passage of time and manage a huge amount of points at the same time.

The first idea was to draw all the points on each roll and give them individually a transition depending on their year, the current oldest year and the window. But this caused two main problems. First, the number of points was way too large in some time areas. This made the animation wiggle as well as made these areas so dense that the information wasn't really readable. Second, when a slight lag had occurred, the x axis weren't synchronized with the points anymore, as they were moving independently.

To balance that, data was reduced by replacing each point by the mean for each year as well as their distribution with respect to their y axis. This ensured that the number of points would not exceed the number of year in the window.

For the animation, a different approach was also preferred. Instead of giving a transition to each point individually, they were all drawn on a *g* SVG element that had itself attached the transition, and just verify at each moment which points should be displayed on (or removed from) it.

Since sometimes the horizontal grid wasn't precise enough, an information panel was added and is displayed when a point is hovered on.

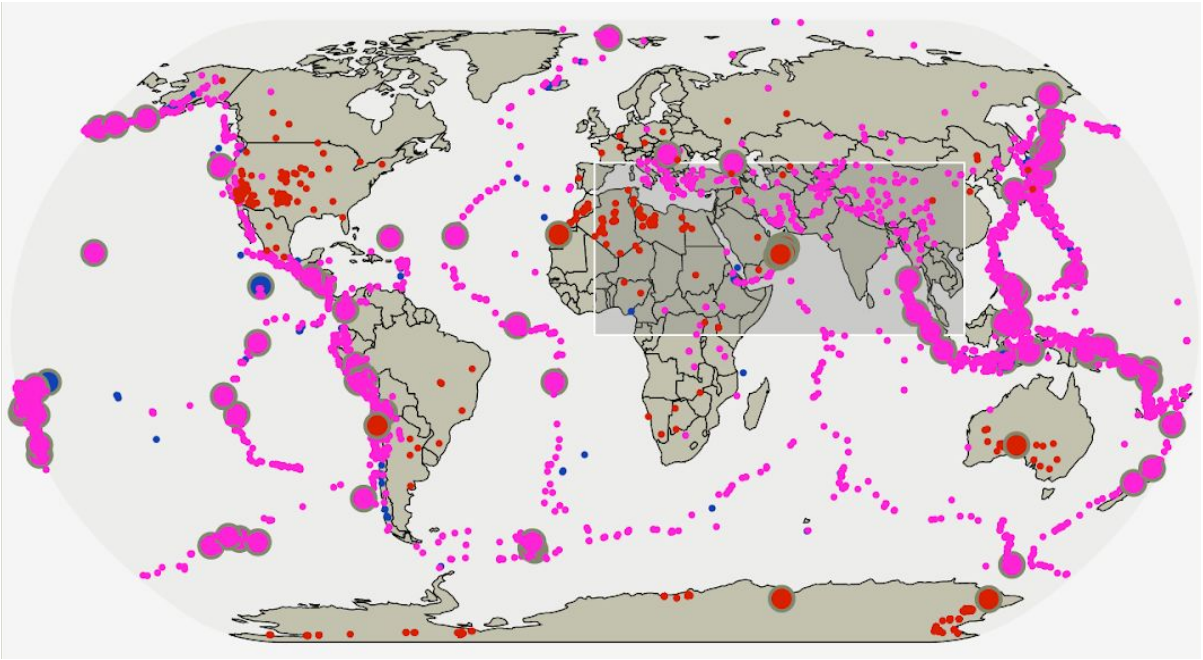
Further in the project, the idea came that since the data couldn't be fully represented on the rolls (only their mean), the y axis could also contain some information on the distribution of the current data on the roll. The histograms where thus implemented and serve to give the user more knowledge on how the data behaves relatively to the y axis.

Yoshi

In the making of the **rolls**, the **map** was being implemented. The need for a common time referential pushed us to create a global class (that we called **yoshi**) that would manage the time for all the other elements, as well as the data, and call their constructors.

As a quick note, the **rolls** and the **map** find the current index of the points that need to be displayed in O(n) complexity as we made so that the datasets were sorted.

Map



Having experimented with drawing maps during the course exercises, the cartography was quite easy to implement using Promises for the map data. We followed the same practices to manage the points, but this proved to be a challenge, since only part of the data needed to be shown, which we managed in the same fashion as in rolls and the points needed to have a fade in and out to soften the transitions in an efficient fashion when the time is moving, thus adding a transition on the point opacity, that removes the point automatically from the current buffer when it reaches zero. Note that, due to having a pause functionality, this transition had to be stopped and continued when needed, that we managed with functions **stop_fade** and **cont_fade** that manipulate their transition speeds accordingly.

When selecting a time period manually with stopped time, the points also needed to appear with full opacity and zero fading, thus we have two seperate ways to draw points: static and normal.

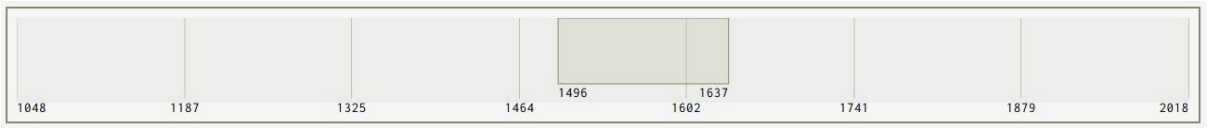
In order to unify our visualization, the selection of the mean values in rolls, calls our map to highlight all the points of the specified type and selected year and increases their radius for the user to see them. In addition, the map showed only the information of position, type and time period about individual points, thus to increase the information, we added information boxes to points that are mouse-overed, which needed to manage the information box window for it to not exceed the svg window.

Lastly, we followed the course practices to implement the brush feature on the map and the points inside. To do so, we used an inverse transform of the selection coordinates into the projection coordinates and selected the point data from the current buffer that are inside the selection square.

Time control slider

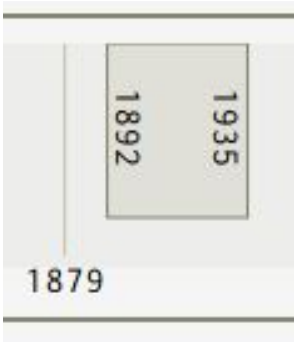
When we started implementing the **time control slider**, we expected to be able to handle both dragging and resizing using *d3js*. However, as resizing proved difficult to implement with *d3js*, we found another library, *interactjs*, which could handle both dragging and resizing.

While we had managed to implement dragging with *d3js*, we found it was difficult to use in conjunction with *interactjs* resizing, because we would have to make sure *d3js*' drag hover zone did not overlap with *interactjs*' resizing hover zone, which would otherwise cause bugs because both actions could be triggered simultaneously. We thus decided to use *interactjs* for both, since it already ensures both actions cannot happen simultaneously. Therefore, *d3js* was only used for rendering the time control slider, but not for handling the user interactions directly.



Time control slider - draggable and resizable

Although *interactjs* provided exactly the functionality we needed, we still had some trouble understanding its API, as very little documentation was provided on their website. Furthermore, implementing the slider required converting an absolute screen position to a year using the screen positions of each bound of the **slider**, which introduced many subtle bugs that would cause the slider to behave in unexpected ways. As a result, it took much longer than we anticipated to make it work. Eventually though, we managed to sort out the problems and find the right algorithm.



Finally, we realized the minimum allowed width for the time window was too large, which would prevent the user from picking small time ranges. It was wide because allowing it to become shallower would have the two bound indicators superimpose each other, and thus become unreadable. We fixed this problem by flipping the indicators vertically once

they start to superimpose horizontally, thus allowing for shallower time windows.

A possible improvement to this could be to draw two vertical lines aligned with the bounds of the time window, and write the text bound labels horizontally beside each line but at a different height, such that they cannot overlap even with a window of size zero. Care should be taken to ensure that one line could not overlap with the other bound's label, or the slider's minimum and maximum bounds.

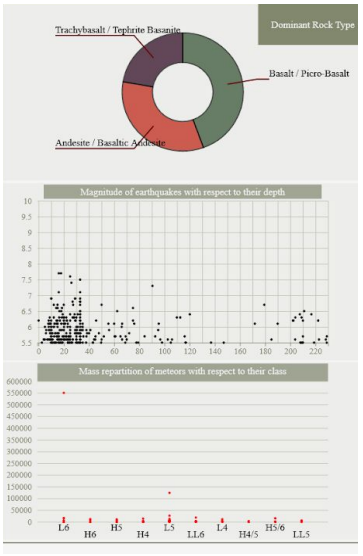
Filters

As for the **filters**, the challenge was due to the fact that to find our current index we required the data to be sorted. So removing data points wouldn't cause any problems since it would keep order, but adding back would. That is why the filters add to take into account all the current checkboxes and filter accordingly.

Then the filtered data needed to replace the current data and update the rolls and the map.

Statistics

The **statistics** were an occasion to perform some interesting data display as well as showing correlations among attributes.



Sadly, our datasets were very categorical and very few columns could give raw floats to perform such things. Instead, the **statistics** were used as a way to display more data on the selected part of the brush in a different way for every data type.

They were made so that they update each time the selection on the map changes. To do so, a general class of statistics calls the three subclasses that draw the plots.

When the *brush* is selected, it parses the data for the **map** and sends it to the statistics main class that redraws the plots.

Peer assessment

rolls, **filters** and **statistics** implementations - Leo Marco Serena
time control implementation & **css** style - Loïc Karl Droz
map implementation & **color scheme** design - Justinas Sukaitis