Data-Visualisation project

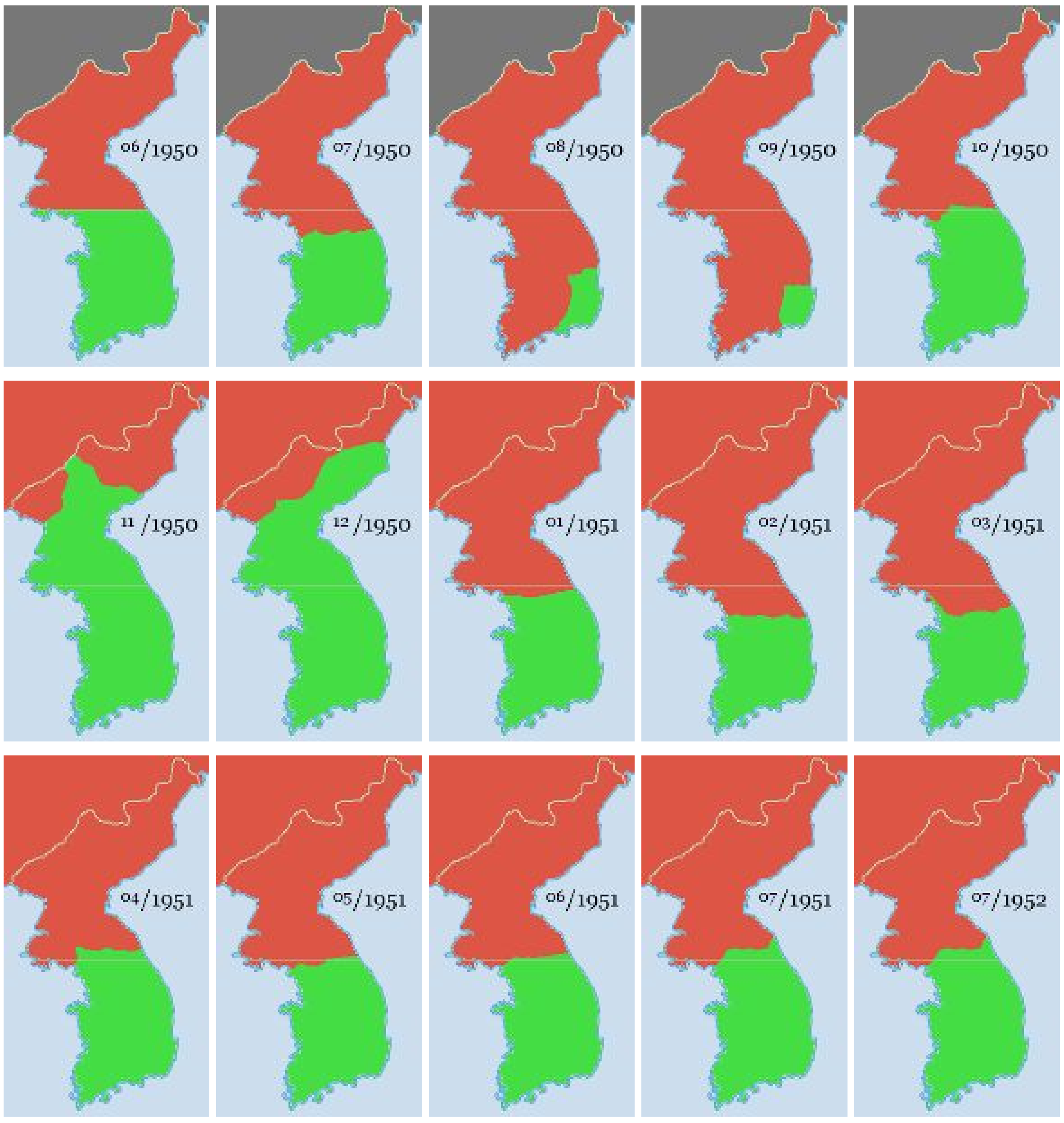
Process Book

Quentin Bacuet, Leandro Kieliger, Isaac Leimgruber

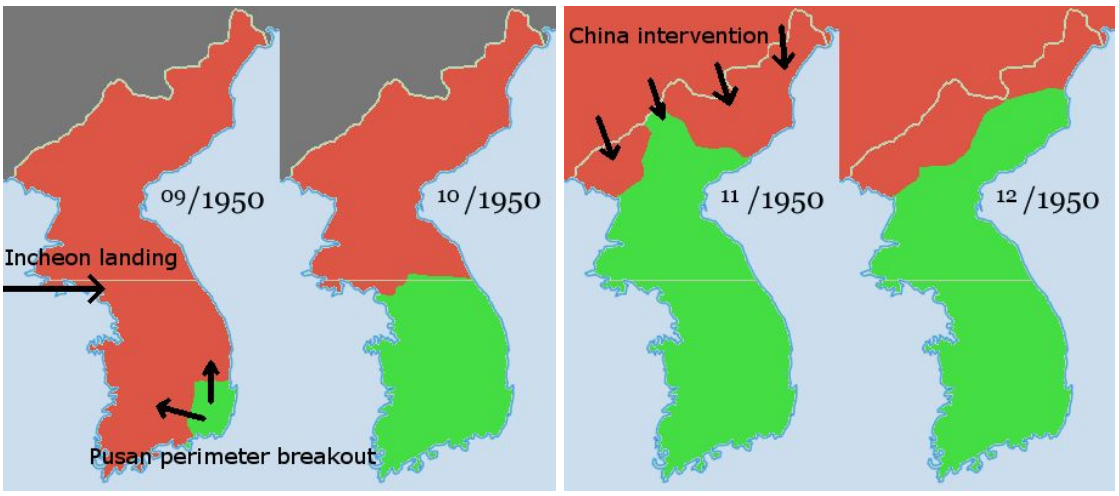
Initial Ideas

The Korean War

Our initial motivation was to create a visualisation which would provide insights on the Korean War, which, because of its geographical distance, is poorly known to Europeans. One interesting aspect about the Korean war is the way the battlefront, and hence the border between the two Koreas changed during the three years that the conflict lasted. Below is illustrated how the demarcation line evolved. The illustration is taken from the wikipedia page on the Korean conflict, red represents North Korea, backed by the Soviet Union and China whereas green represents South Korea, backed by the United Nations.



Along with the change in the demarcation line we planned to show specific attacks and counter-attacks that led to a significant change in the demarcation line along with the number of victims at each point in time. The goal was to tell a dynamic story about the battles that shaped the conflict. Moving arrows and short descriptive texts would have explained the purpose of each operation as well as who was involved. An illustration for the former part is shown below:



Issues faced

Finding the data

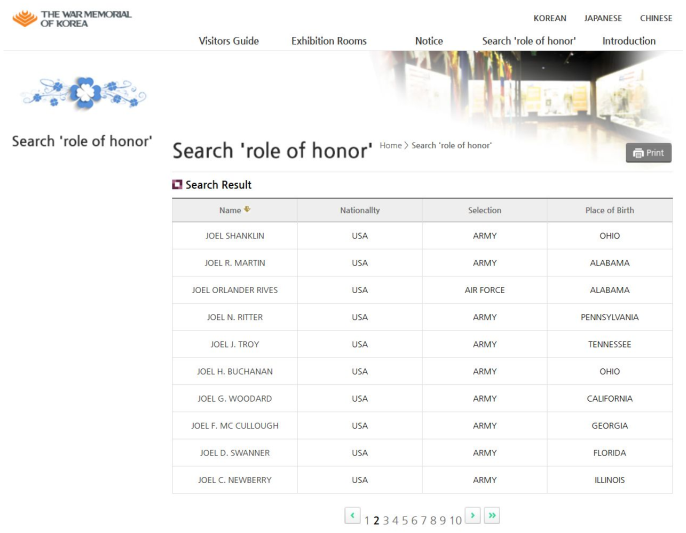
The main issue with our initial project idea was the lack of publicly available data. First, there is no readily available geographical data about the exact state of the demarcation line during the conflict, we would have needed to create it "by hand" either by following maps or reports.

The list of attacks and counter attacks can easily be aggregated from various online sources. However, their effect on the demarcation line would have required more in-depth studies of the conflict.

The final hurdle that we encountered and that eventually led to the abandon of this initial project idea was the lack of available data on the casualties and people involved in the conflict. This was a major turnoff as we wanted to be able to narrate the unfolding of the conflict as a serie of timed events with the number of people involved in each event. The number we were looking for were the number of engaged soldiers, the name of the generals, the casualties and the wounded.

We found the following resources:

- The War Memorial of Korea, has a database for "role of honor" but no indication on the precise role of the individual, whether the he was killed and if so, when.



Search 'role of honor'

Search Result

Name	Nationallty	Selection	Place of Birth
JOEL SHANKLIN	USA	ARMY	OHIO
JOEL R. MARTIN	USA	ARMY	ALABAMA
JOEL ORLANDER RIVES	USA	AIR FORCE	ALABAMA
JOEL N. RITTER	USA	ARMY	PENNSYLVANIA
JOEL J. TROY	USA	ARMY	TENNESSEE
JOEL H. BUCHANAN	USA	ARMY	OHIO
JOEL G. WOODARD	USA	ARMY	CALIFORNIA
JOEL F. MC CULLOUGH	USA	ARMY	GEORGIA
JOEL D. SWANNER	USA	ARMY	FLORIDA
JOEL C. NEWBERRY	USA	ARMY	ILLINOIS

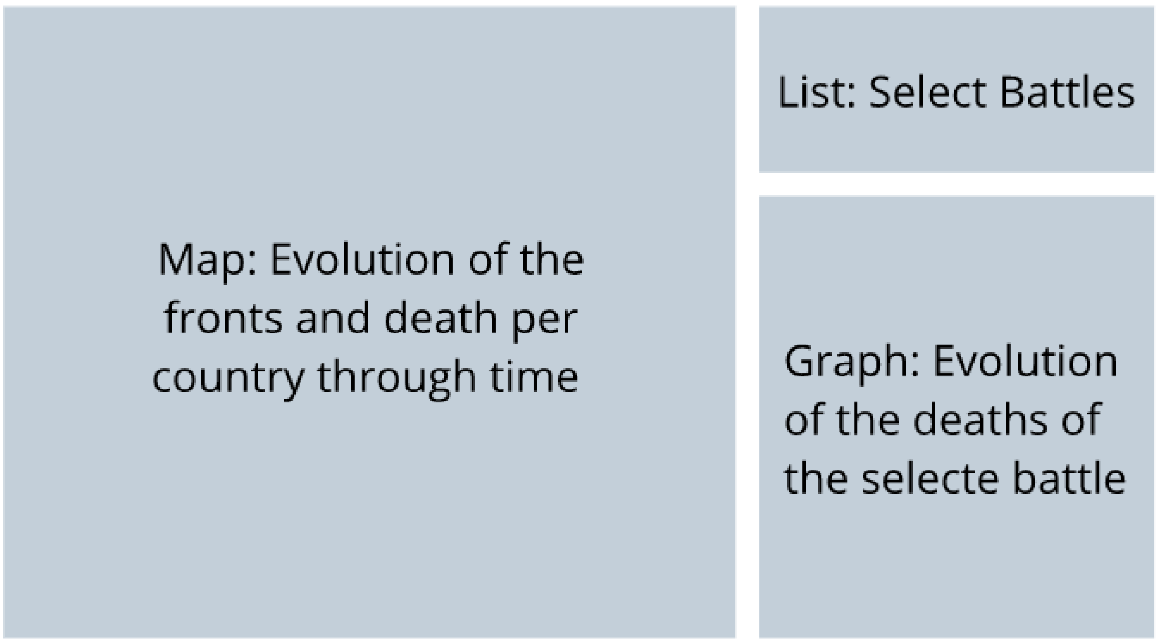
- The National Archives (USA), has a database for all american personnel involved in the Korean conflict but only allows the download 1000 entries at a time. Scraping the whole dataset would have taken a long time.

Basically, each country keeps a database of its engaged personel but there is not centralized resources, all the formats are different and there are several limitations to actually retrieving the data.

Because this subject would have incurred a significant amount of preliminary work on the data, the decision was made to give up the visualisation on the Korean conflict in favor of a topic whose data is more easily accessible.

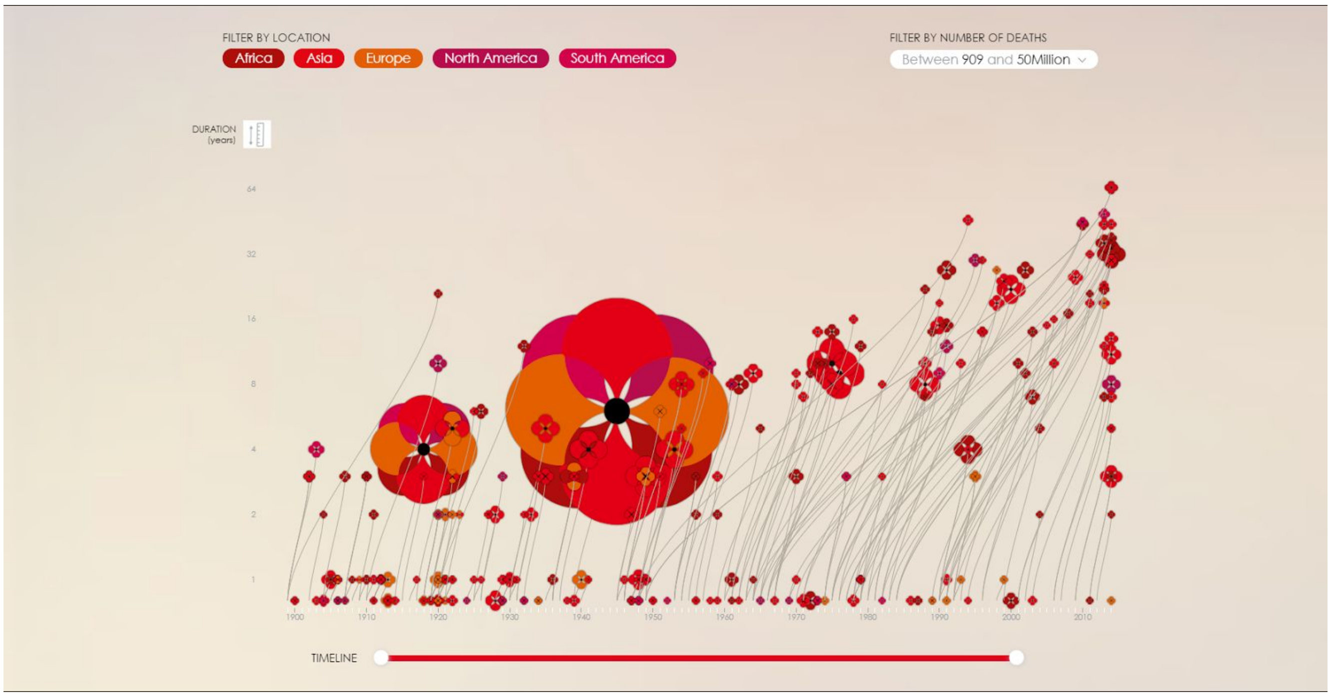
World War I

We oriented our search towards World War I. One of our ideas was to show the evolution of casualties. We had the idea to show deaths by battle and country in a map. At the beginning of the war, the country would be full and the country would slowly get empty through time:



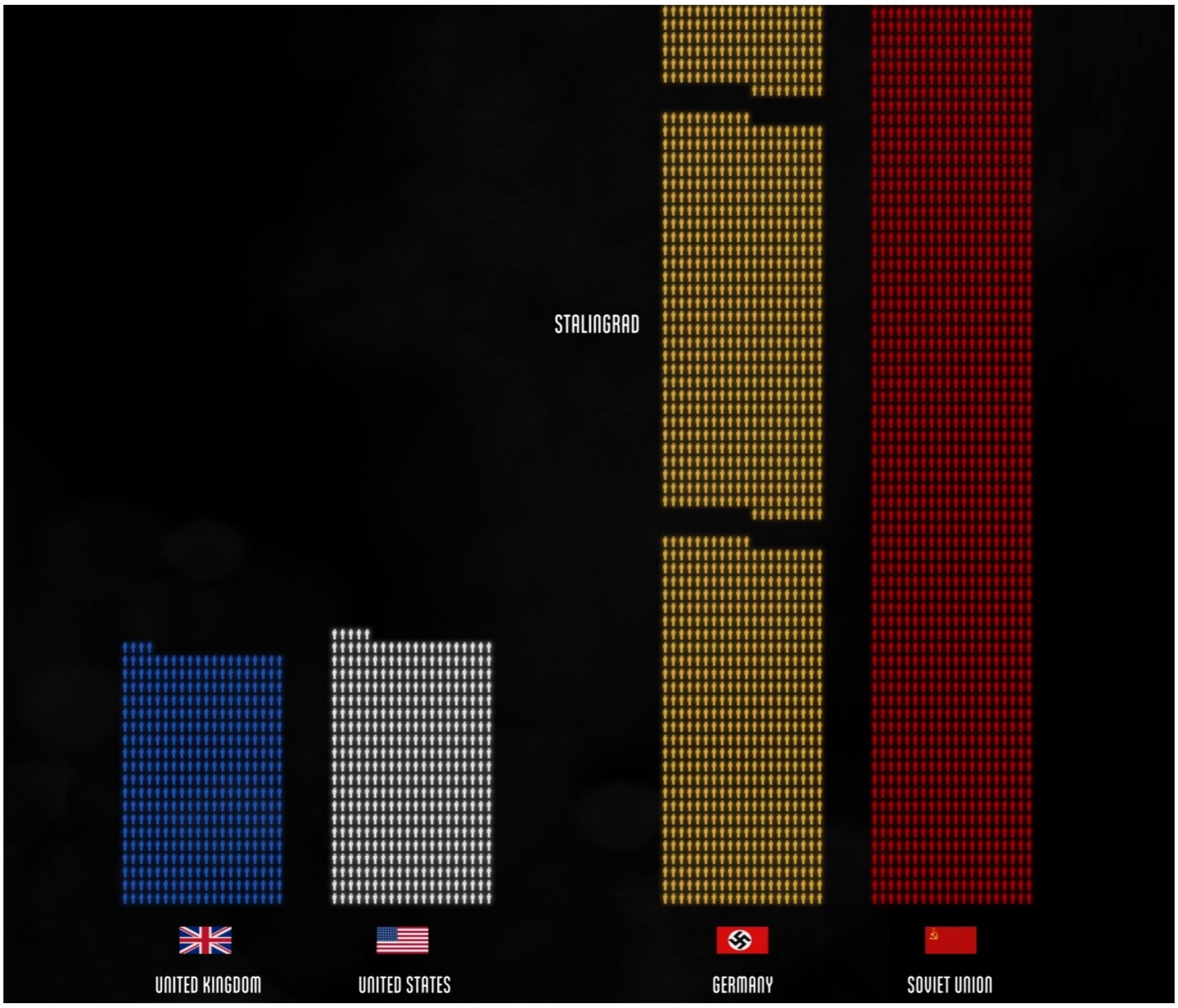
Map: Evolution of the fronts and death per country through time
List: Select Battles
Graph: Evolution of the deaths of the selecte battle

The motivation for the visualisation came from work such as the one available on http://www.poppyfield.org/.



World War II

In parallel, we researched data and visualisations about World War II. Unlike the Korean war, it is much better documented, especially since it directly affected Europe. Again, the idea was to illustrate the casualties count evolving over time. The motivation for the visualisation came from work such as the one available on www.fallen.io/ww2/.



Fallen.io/ww2/

Because of the quantities of different sources available and the extensive topic possibilities we decided to change our approach for the visualisation: instead of choosing a topic and looking for data about it, we chose to browse freely available datasets and let us inspire by their contents.

The main resources we found for free public datasets were:

- https://www.kaggle.com/
- https://opendata.swiss/en/
- https://github.com/caesar0301/awesome-public-datasets
- https://www.kdnuggets.com/datasets/index.html

The main dataset that captured our attention was the refugee time serie from the UNHCR available on kaggle. The reason for this is that as explained in the following chapter, it already had all the structure needed for creating a representation which would evolve over time. From this dataset we developed our final idea.

Final Idea

Create a tool to provide insights on refugee data using animated points to represent flows of populations across the world. With our visualisation, the user should be able to navigate through the data by isolating flows by year and by country.

Overview

The refugee crisis is a hot topic for almost everyone. With our visualisation, we wanted to present the data as objectively as possible without preaching any viewpoint. This visualisation provides a simple yet complete tool to explore and understand refugee flows from 1985 to the present date.

Target Audience

Target user group	**Journalist**
Knowledge about the refugees	Intermediate
Key questions	● How many refugees have fled from a particular area? ● How many refugees have fled towards a specific country ? ● Is the current crisis a recurrent or new phenomenon?

Target user group	**Citizen**
Knowledge about the refugees	Small to none
Key questions	● Where do refugees mainly come from ? ● Where do they go in general ? ● What are those refugees fleeing from ?

Related work

http://www.therefugeeproject.org/
Build on the same data but emphasises the history behind each conflict.

https://explorables.cmucreatelab.org/explorables/annual-refugees/examples/webgl-timemachine/
WebGL animation of refugee flows also based on the UNHCR data

Datasets

UNHCR Refugee Data

Available at https://www.kaggle.com/unitednations/refugee-data/data

- Asylum_seeker.csv
  Yearly progress of asylum-seekers through the refugee status determination process, with data on UNHCR assistance.

- Asylum_seekers_monthly.csv
  (1999-2016) Monthly totals about asylum applications opened in 38 European and 6 non-European countries, by month and origin. Repeat/reopened/appealed applications are largely excluded.

- Demographics.csv
  (2001-2016) Information on refugees according to residence territory, broken down by regional level, age, and gender demographics. Warning-may be incomplete or otherwise conflict with persons_of_concern.csv and time_series.csv.
- Persons_of_concern.csv
  (1951-2016) Yearly time series information on UNHCR's populations of concern for a given year and country of residence with origin.

- Resettlement.csv
  (1959-2016) Yearly data on resettlement arrivals, with or without UNHCR assistance, and should exclude humanitarian admissions.

- Time_series.csv
  (1951-2016) Yearly population statistics on refugee movement changes from an origin to a destination.

From this dataset we use the time_series.csv file

The CSV file contain the following columns:

1. Year
2. Country of asylum
3. Origin
4. Population type
5. Value

The population type indicates the type of movement. For example, people which are forced to leave their home but can stay within the country are labelled as “Internally displaced persons”. In our visualisation we only consider flow of people between countries and not displaced persons within their country.

For each year, there are multiple rows where pairs of asylum and origin countries have an associated number of people which travelled from the origin country all the way to the country of asylum.

Example:

2014	Pakistan	Afghanistan	Refugees	1’504’912

Longitude & Latitude coordinates for countries

Available at https://developers.google.com/public-data/docs/canonical/countries_csv

This dataset provides us with the geographical coordinates needed to animate points moving on the map.

Example:

AD	42.546245	1.601554	Andorra

World population in 2016

Available at https://data.worldbank.org/indicator/SP.POP.TOTL

Contains world population as a time series for each country

Belarus	BLR	1978	9’525’000

Processing Steps

In this section we present the processing steps to bring the data in a format usable for our visualisation. The implementation was made using two different IPython Notebooks with the help of the Pandas library. Pandas is a data analysis library which provides a plethora of useful functions to filter, reorganise, join and perform computation on large sets of data.

data_preparation_map.ipynb

This notebook extract from the dataset the features necessary to display refugee flows. That is, a year, a country of origin, a country of asylum and a population count. The notebook then create a csv file names flows.csv which contains the following information:

1. Origin country of the refugees using the ISO 3166-1 alpha-2 format (two letters)
2. Latitude of origin country
3. Longitude of origin country
4. Asylum country (destination) using the ISO 3166-1 alpha-2 format.
5. Latitude of the asylum country
6. Longitude of the asylum country
7. Number of refugees

Therefore, each pair of country can potentially have a refugee flow. Flows.csv contains roughly 107k entries.

data_preparation_in_out_fraction.ipynb:

This notebook creates 3 files:
1. data_immigration_fraction.csv
2. data_immigration_entry.csv
3. data_immigration_exit.csv

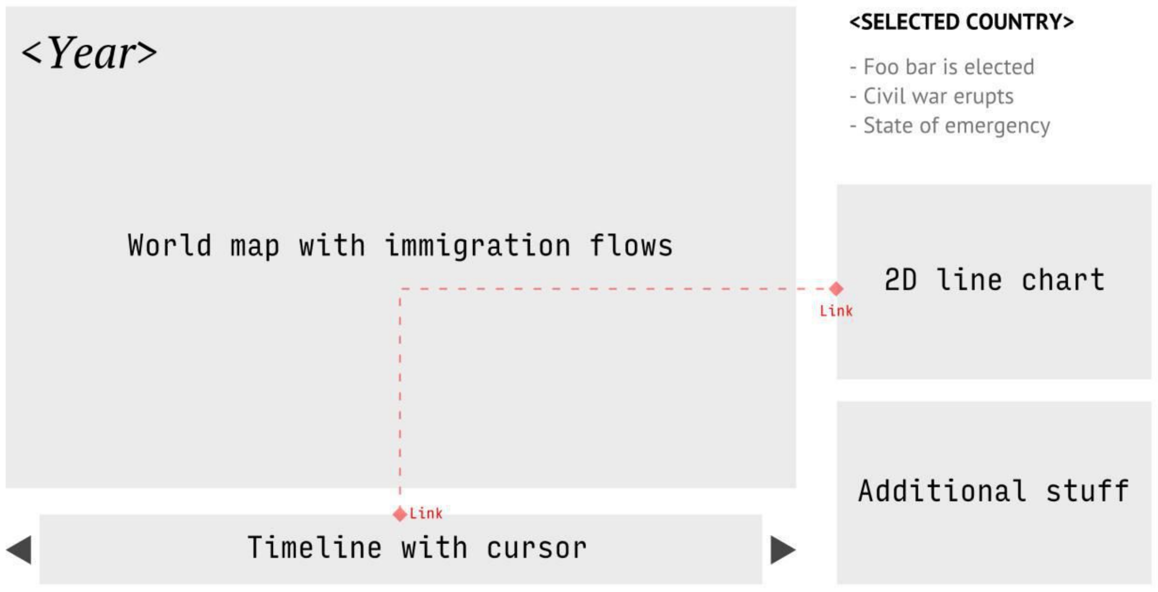
The _fraction file stores the net difference between the total outflow and the total inflow for a country per million inhabitants. In addition to outputting the ratios, the notebook also computes their quantiles. The file and the quantiles are used to render the choropleth map.
The _entry file store the total number of refugee entries for a country and for a given year. It is used for the refugee count graph.
The _exit file is analogous, it is also used for the refugee count graph.

Each row in these files has the following information:

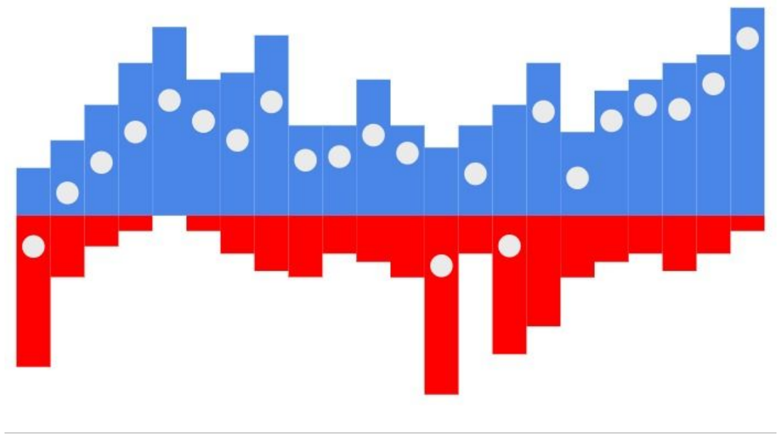
1. Country
2. Year
3. Quantity (one of the three described above)

Design

Initial draft



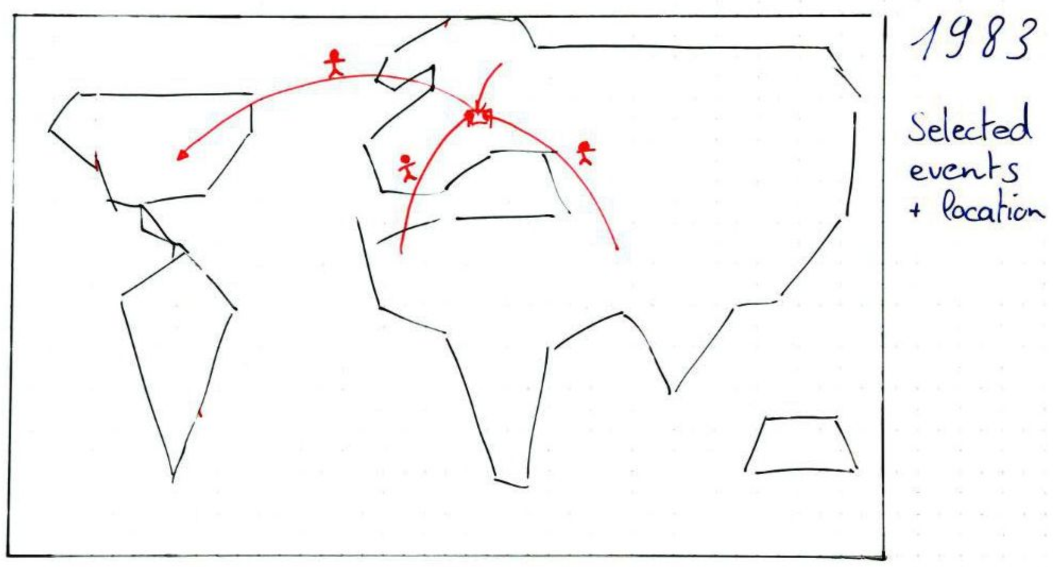
The initial design idea is represented on the above schema. The central part of the visualisation is a map of the world where immigration flows for a given year are represented as points moving around the map. The visualisation is tied to a 2D line chart on the right which for a given country show the cumulative amount of people that left the country against the cumulative amount of people that entered the country. We had the idea to put on the graph 3 types of information: the entries as the blue rectangles, the exits as the red rectangles and the difference between the entries and the exits as white dots:



Navigating along the years is made possible thanks to a timeline box with a cursor. Since the data is a time series, changing the currently displayed year updates the visualisation on the map and on the 2D line chart.

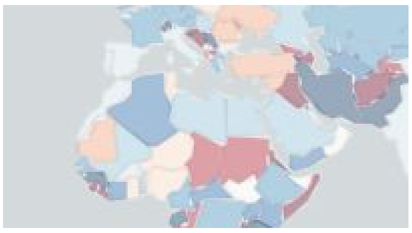
The flow visualisations are made between a selection of origin countries towards a selection of asylum countries. When the selection is done, points starts moving across the world map.

We assumed that certain historical events directly influenced the refugee flows on the map. To point the user towards a possible cause of the population outflows, we wanted to add on the side of the visualisation a list of major historical events that happened in the selected period of time.

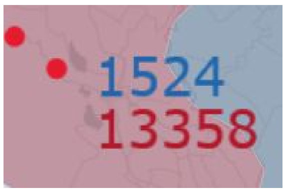


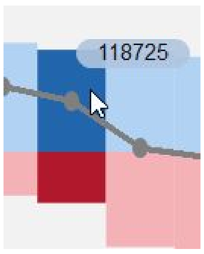
Evolution of design

We quickly felt the need the add a choropleth layer on top of the map. Indeed, when all the countries are the same color, there no indication regarding which country should be selected for the flow visualisation. Changing the color of each country depending on the amount of refugees that entered or left allowed to instantly spot on the map the regions with interesting data.



The graph showing refugee counts was a good starting point to obtain exact numbers about a particular country but since it displayed total amounts, it was not possible to differentiate from how many refugees went to a specific country A or how many came from a specific country B. This is why we added on top of the choropleth





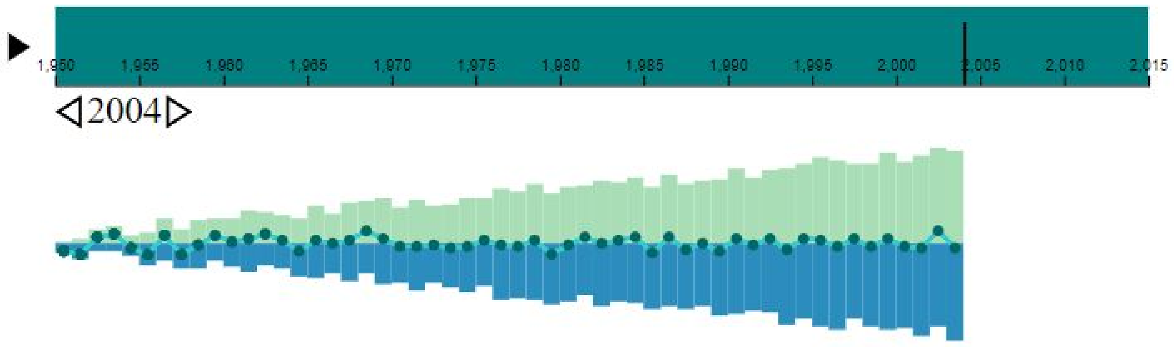
layer the exact number of refugees coming from and going to specific countries

We discarded the idea of allowing the user to select the countries for which we wanted to display the refugee flows. Indeed, displaying the inflow and outflow for only one country is already visually heavy as refugees often scatter over tens of countries. Adding even more origin countries to be displayed at the same time would have made the visualisation essentially uninterpretable.

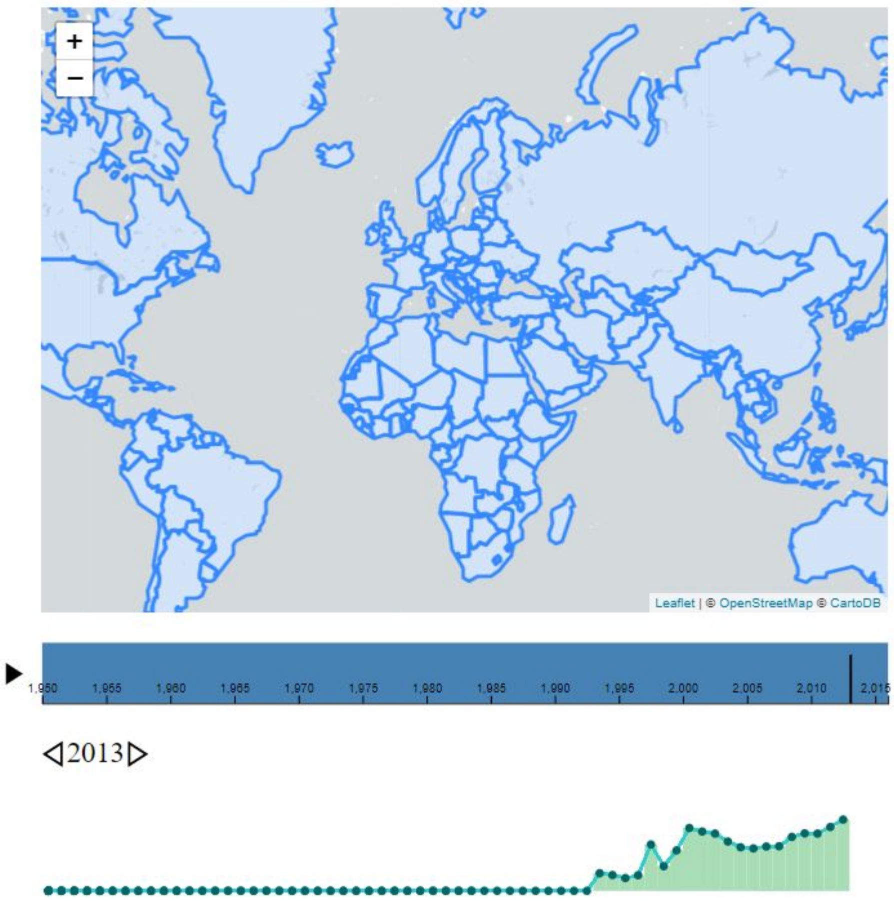
We chose not to display a descriptive text along with the historical event as we were starting to get short of space. Instead, we associated to each event a Wikipedia page that the user can read, should he be interested in knowing more. However, to add a bit of context to the visualisation we also decided to associate to the historical events a series of images taken from their respective wikipedia page.

Evolution of the Visualization

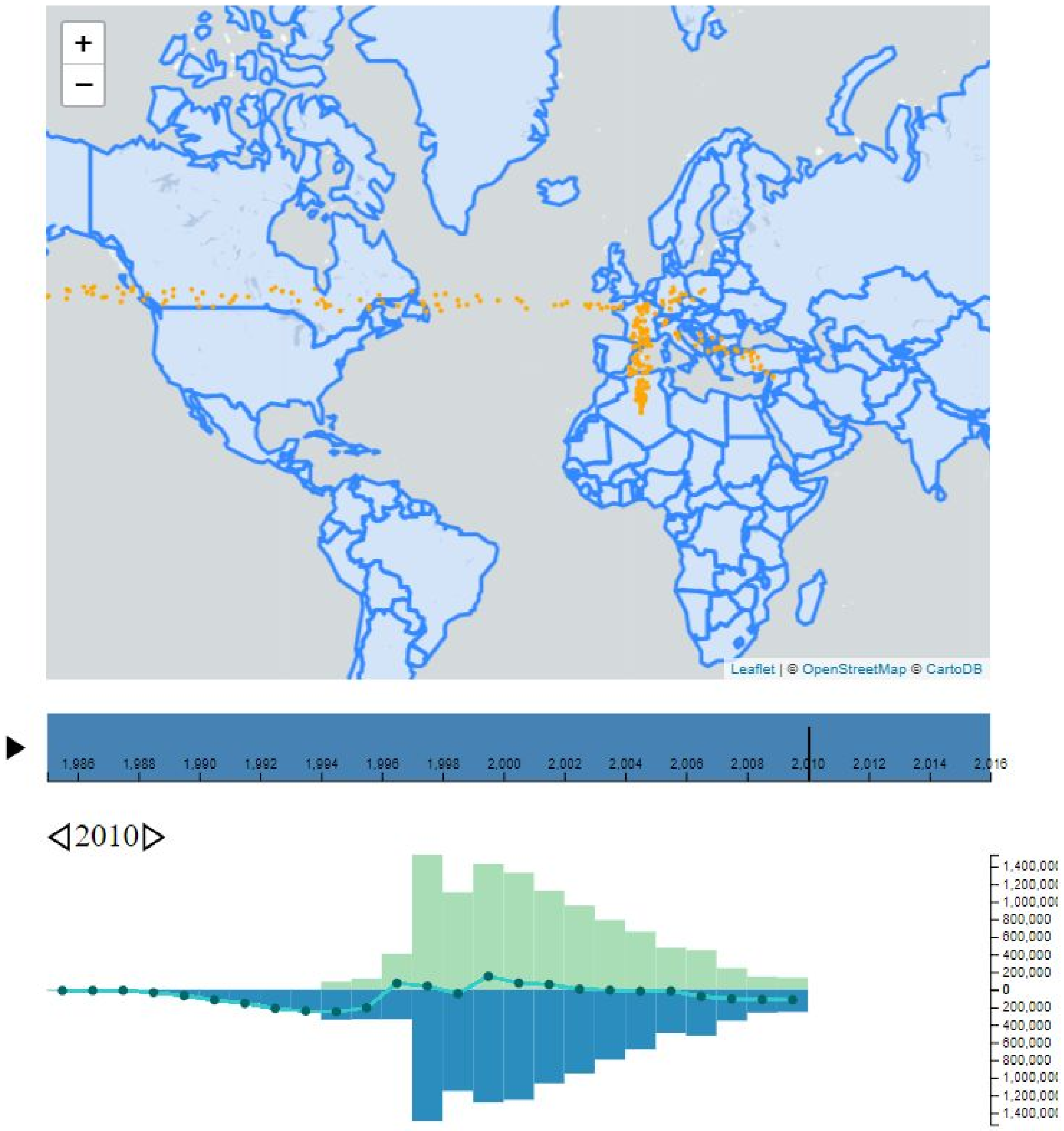
We first started our visualisation with a timeline and a simple graph. At this time, the graph would only display dummy data from 1950 to the year selected in the timeline.



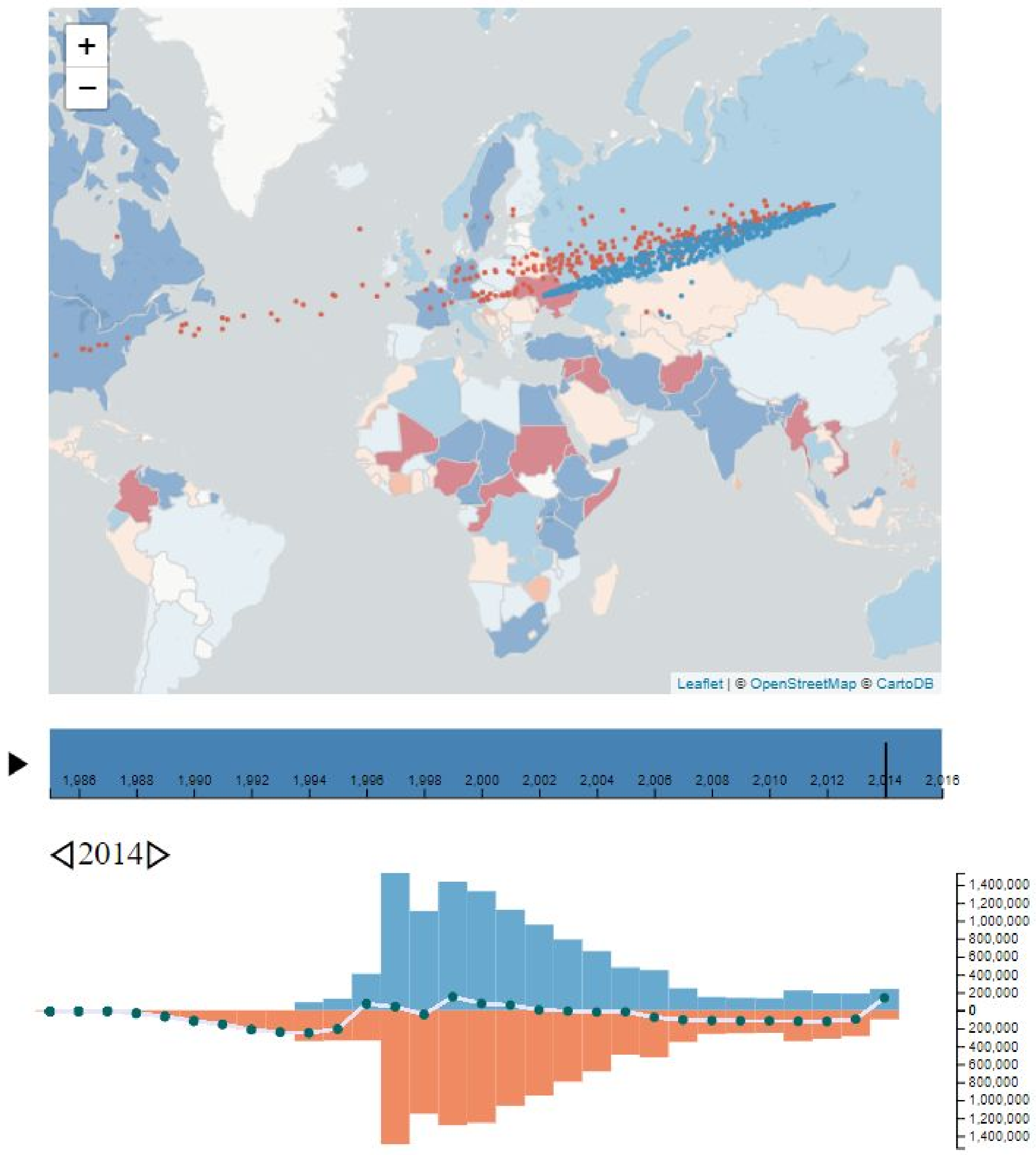
We then added a simple map Leaflet map with a GeoJSON overlay with each country. The graph was improved to actually make use or real data.



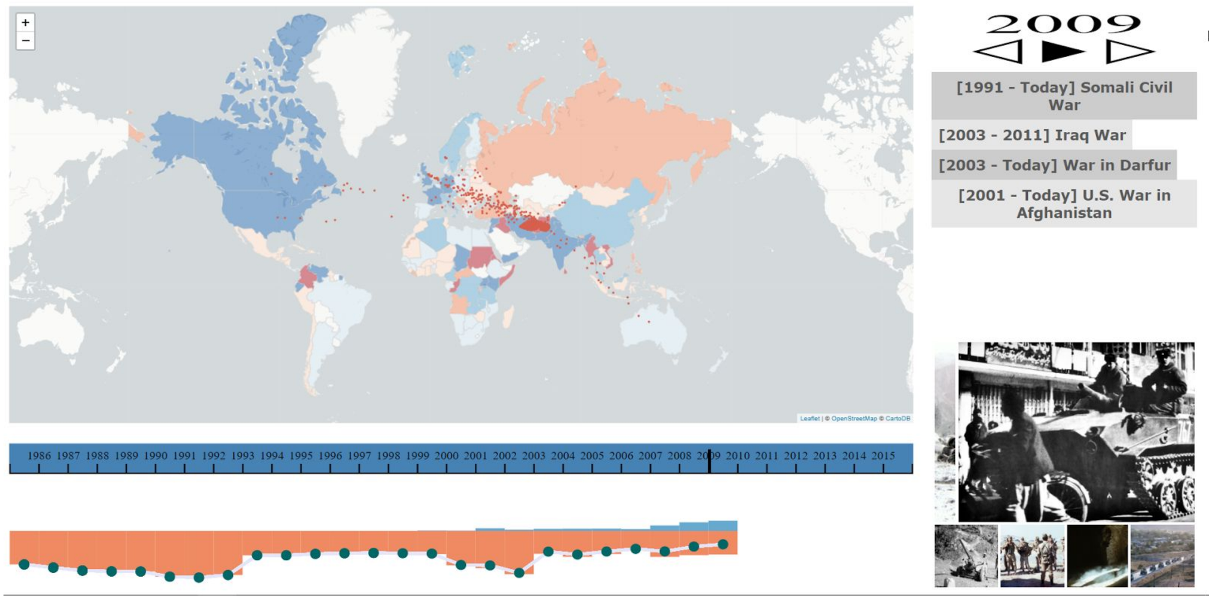
We implemented a canvas layer on the of the Leaflet map and started prototyping refugees flows as moving points on the map. Dummy quantities were used for the density of the flows.



We implemented a choropleth map showing the differences in refugee inflow and outflow for each country. The flow visualisation was improved to use the real data.



We completed the visualisation with some images and key historical events.



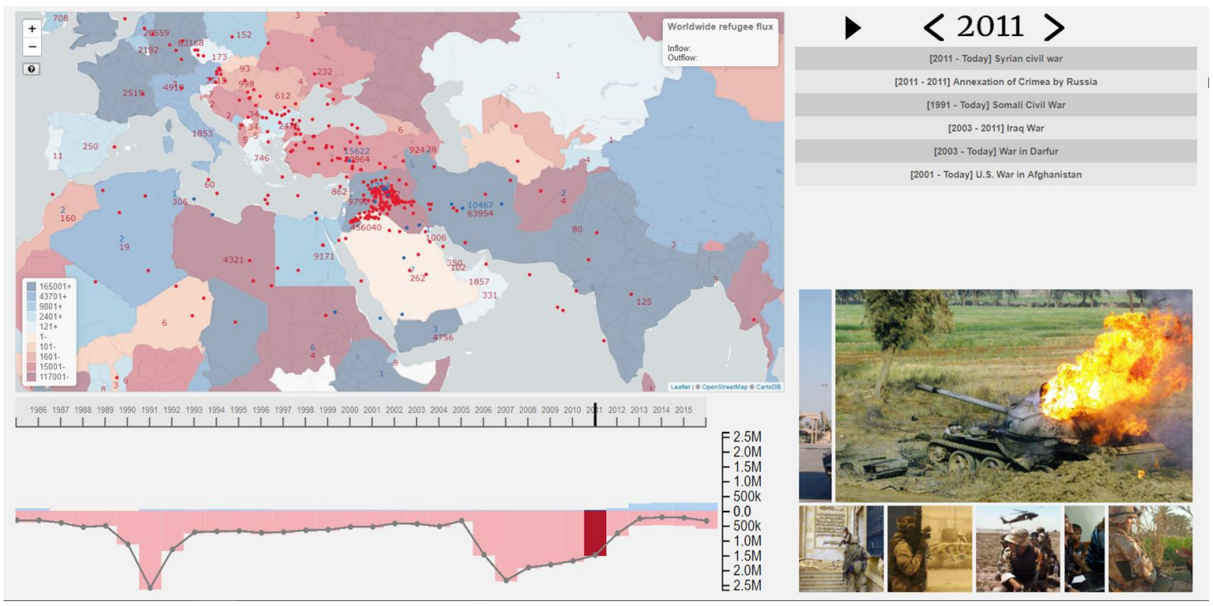
[1991 - Today] Somali Civil War

[2003 - 2011] Iraq War

[2003 - Today] War in Darfur

[2001 - Today] U.S. War in Afghanistan

Finally, we rearranged all the elements in a more compact way. We changed the graph so that it always showed all the data and highlighted the current year. We added the exact number of refugees for each flow on the map.



[2011 - Today] Syrian civil war

[2011 - 2011] Annexation of Crimea by Russia

[1991 - Today] Somali Civil War

[2003 - 2011] Iraq War

[2003 - Today] War in Darfur

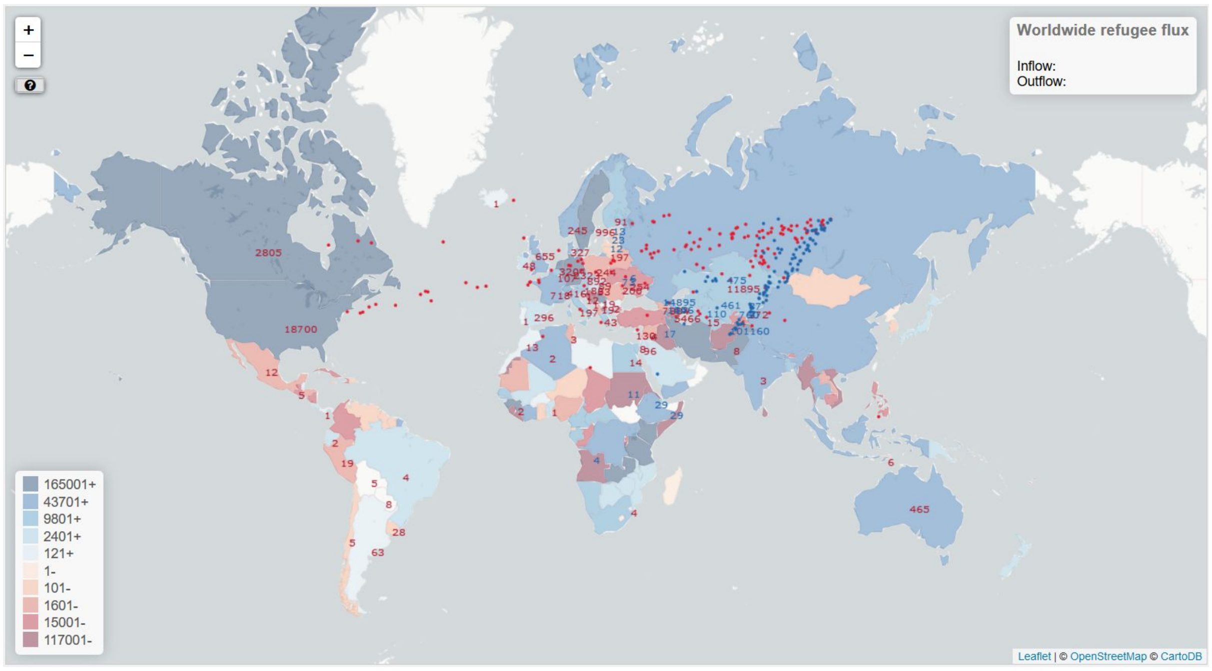
[2001 - Today] U.S. War in Afghanistan

Implementation

General structure

The first steps of the implementation were to define a DOM hierarchy of containers for each element of the visualisation. We created several container divs which were appended to a main div identified as the root. The root would then scale with the explorer window and resize its children accordingly. The timeline and the graph are made with SVGs.

Map



The core technology behind our map visualisation is the Leaflet library for JavaScript.This visualisation is made of several components which are in bottom-up oder:

- The base map tiles
- The GeoJSON overlay
- The canvas layer
- The control layer

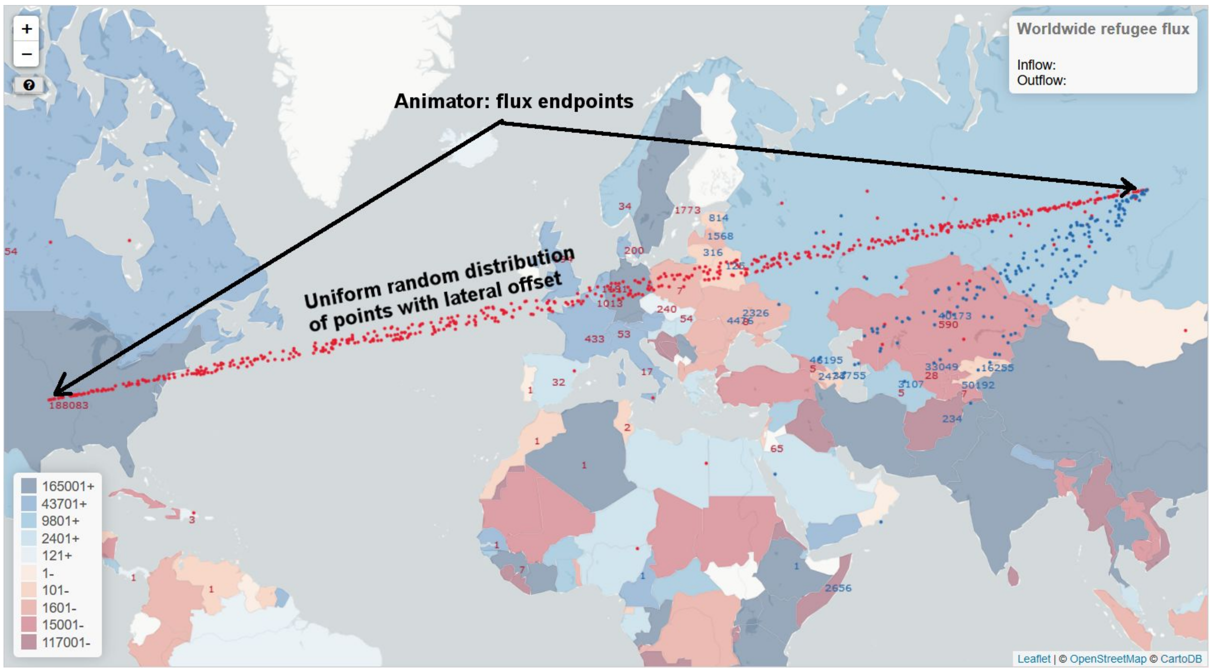
The base map tiles are the positron tiles by CartoDB. Leaflet provides out-of-the box support for loading such tiles.

Leaflet also provides support for joining GeoJSON data on top of the map tiles. We used the geographical data available here.
The canvas layer however, deserves more explanation.

Canvas Layer

When it comes to implementing visual cues moving on a leaflet map, the common approach is to use leaflet markers. However, due to the very large number of marker we would have needed to use, we were concerned this approach would not scale appropriately.
The alternative approach is to use HTML's canvas element to repeatedly draw the markers on it to produce an animation.
Leaflet does not provide out-of-the box support for canvas layer. Instead, we use a Leaflet plugin, General Layers for Leaflet or gLayers for short. This plugins offers a child class of a Leaflet Layer, CanvasLayer, which handles basic behavior in order to conform to a Leaflet Layer but at the same time, providing a HTML canvas. Our job was then to extend this class and override the default behavior according to the needs of our project.

Animators



The visualisation we wanted to create required animating hundreds of dynamic points across a world map. Providing a way of adding and removing points easily was an absolute necessity for our project. This is why we created the concept of Animators.

Animators are instances of the Animator class found in the *Map.js* script. An animator stores two geographical coordinates corresponding to the beginning and to the end of its corresponding points flow, an array of AnimatedPoints and a quantity. The quantity represent the number of people that should be visualised with the Animator. It is scaled by a fixed multiplicative constant to avoid drawing literally millions of points. When the scaled value is computed, the Animator creates an equal amount of AnimatedPoint instances and store them in an array.

To produce nice-looking animations, we needed the points to be uniformly distributed between their flow endpoints. When an Animator is created, it computes the screen space distance between its endpoints and uses a fixed constant moving speed to derive the travel time. The travel time is then given to each of the AnimatedPoint instances stored by the Animator.

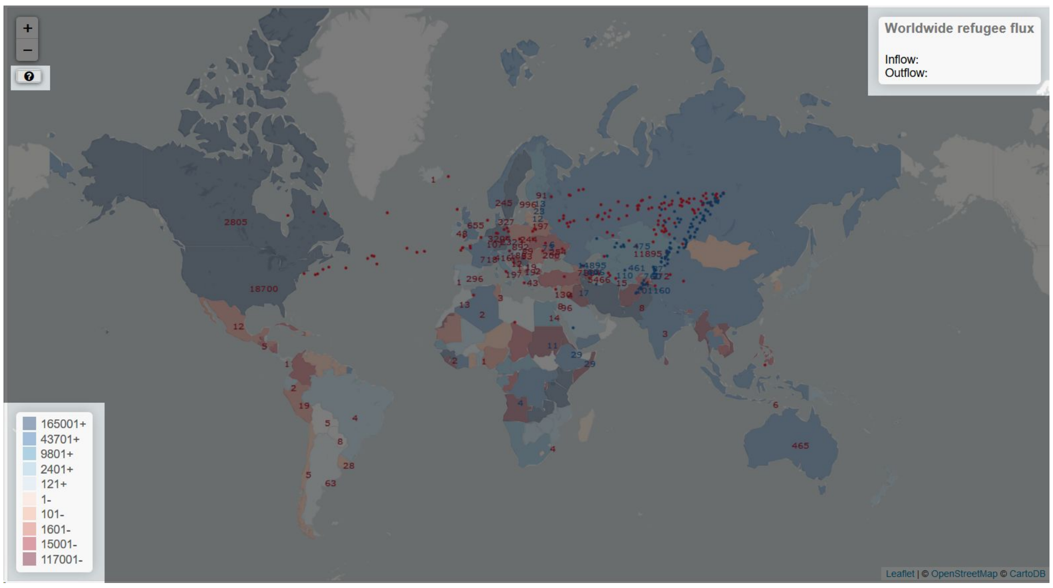
AnimatedPoints

An AnimatedPoint is simply a container for two values which are fixed when the instance is created. The first is a value between zero and the total travel time given by the Animator. This value is computed at random and uniformly distributed across [0, travel time]. The second is a random lateral offset for drawing.

Putting the pieces together

When an animator is used for drawing, each of its AnimatedPoint instances is retrieved sequentially. The position of each AnimatedPoint is determined relative to the Animator endpoints by linear interpolation. This is where the values stored by the AnimatedPoint instances come into play. To produce the animation, we perform the interpolation with respect to time. Before feeding the time as the interpolation parameter, we add to it the random time value stored by the point. This will effectively randomly distribute the point uniformly across the flow endpoints in screen-space. Remember that the AnimatedPoint instances also store a lateral offset value. This offset value is used to distantiate the points in a direction perpendicular to the flow so that they are not drawn systematically on top of each other. Finally, the time parameter for the interpolation is taken modulo the total travel time so that the animation is periodic.

Control Layer



The Control Layer handles the top overlay such as the choropleth legend, the help button and the country flow-indicator box. Both the choropleth legend and the flow-indicator box are Leaflet Control instances whose respective divs are customized to provide the appropriate content.

The flow indicator box has an event handler which is called whenever the mouse hovers a different country in the choropleth map. The handler updates the content of the box to show for the selected year the total number of refugees that entered a particular country and the total number of refugees that left this same country.

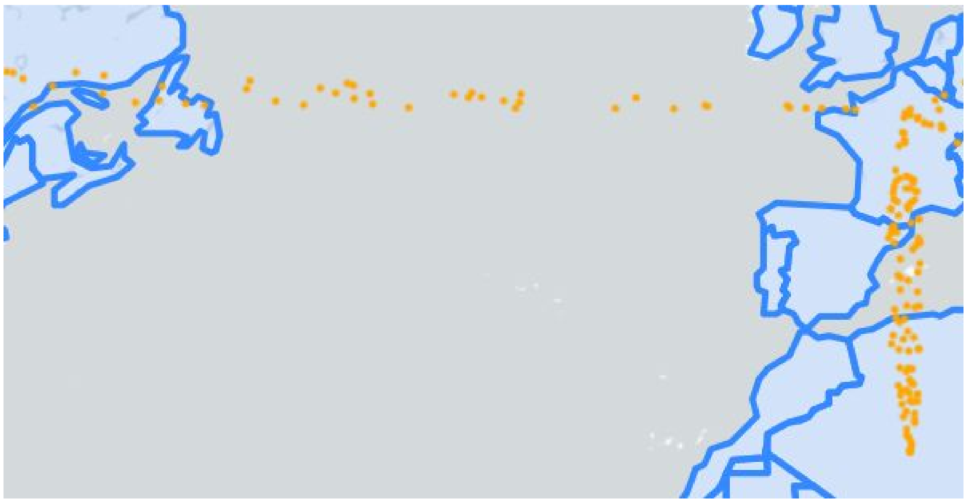
The help button was implemented using the EasyButton Leaflet plugin. Clicking on this button simply attaches or detaches a Leaflet control (just like the choropleth legend) with static content explaining how to use the visualisation and where the data comes from.

Design choices

Flow representation

The principal design choice we had to make for the flow visualisation is how to represent as accurately as we can the number of refugees going from one country to another. The initial approach was to draw between two countries a constant number of points at all times. The number of points would have to be directly proportional to the number of refugee as found in the data. However, we argue that although accurate, this representation does not give the "right feeling" to someone watching the visualisation. Consider the following case:

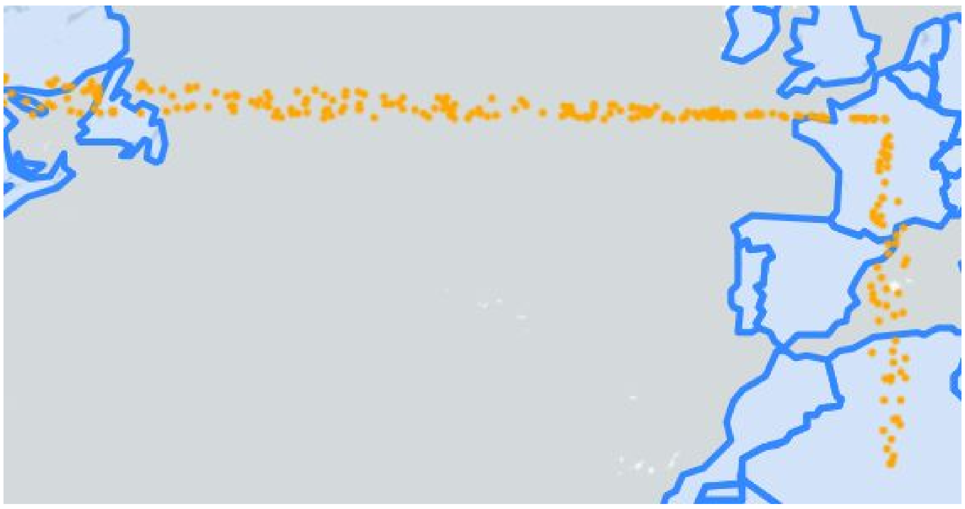
Take a pair of countries AB, very close to one another and a pair CD of countries far apart. Because the speed of the points is constant and the visualisation periodic, the number of points produced and consumed at each endpoint of the flows differ greatly. In addition, because we chose to uniformly distribute the points across the travel distance, the farther apart the countries are, the farther apart the points are from each other. This gives the impression than the flow is more sparse while in reality they represent the same quantity. This situation is illustrated below.



The flow coming from the left represent the same quantity as the one coming from the bottom but the distance makes it sparser.

Another approach is to keep the travel time constant. That means that countries further apart produces flows of points moving faster from one another. This does not change the fact that points are still very spaced. However the number of points being produced and consumed by the endpoints of the animator is equal no matter the distance. The differences in speeds still resulted in an unsatisfactory feeling.

The last approach is to distort the data to give the "right feeling". We reverted back to the constant point speed solution but this time scaled the number of points represented by the flow linearly with the distance. This means that for the same number of refugees, the farther apart the countries are, the more points are drawn. Although this distorts reality, it gives the right impression to the user as the perceived density of flows is equal.



The quantities have been scaled linearly with the distance

We chose this approach for the final visualisation. We further argue that this distortion of data is acceptable as long as the user has a mean of verifying the raw data. Which is the case since we provide the country flow box to read the exact number of refugees coming in and out, the number of refugees going to each countries as text on the map and the country graph to give a visual and accurate representation of the data.

475
11895
461
110
15
5466
17
8

Data distortion is mitigated by showing raw numbers

Choropleth colors

Initially we used the choropleth map to show the net difference between the number of refugees that entered a country against the number of refugees that left this same country. However, fixing the choropleth thresholds with this kind of data is unfair because larger countries will tend to attract or produce larger flows of refugees and thus appear almost the same color throughout the years. This is why we used an additional dataset from the World Bank which enumerate world population as a time series to instead show the percentage of the population that those refugees represent.

Timeline

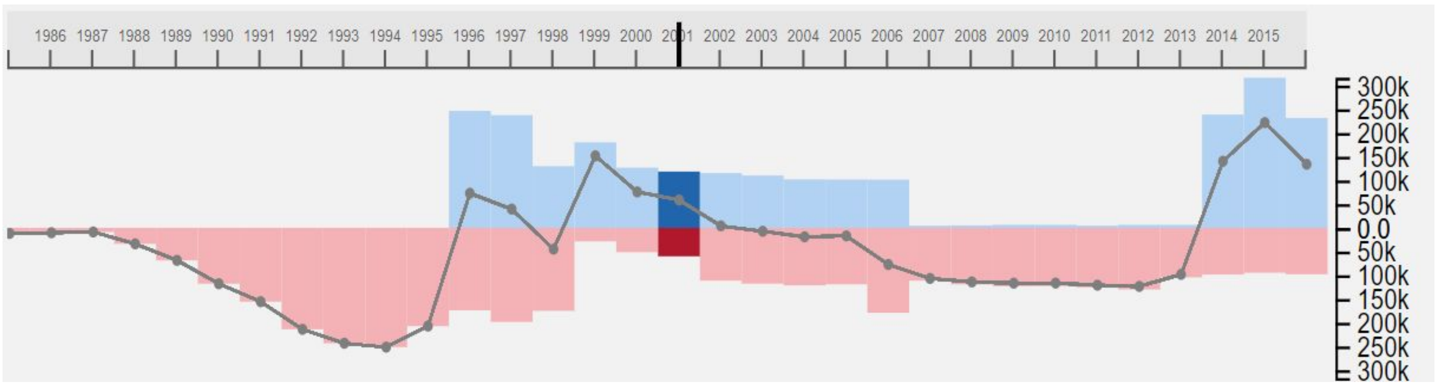
The timeline is a svg rect with three event listeners for the click, move and release events. On top of that, we added a d3 axis for the years and another rect for the cursor. After some testing, we removed pointer events on the cursor and the axis to relay events directly to the timeline rect. The cursor being originally the main interaction point, the architecture of the code built events around it. The cursor is the object providing information about the current year to other objects.

Bar plot

The implementation of the bar plot can be divided in 2 main parts:

- Load the data: To ease the drawing we needed to have data for each year even if there was no data (we put 0 in those cases). But, for evident memory concerns, we did not add in the files entries with values equal to 0. Hence the first task was to fill the gap of the data to have an array that has a length equal to the number of year covered (in our case 32 years).
- Draw the plot: We then used classic d3 functionalities (enter, exit and transition) to draw the bars. For the scale, we used a predefined height mapped to the max between the biggest value of the entries and the biggest value of the exits. The y-axis and the rectangles are scaled accordingly.

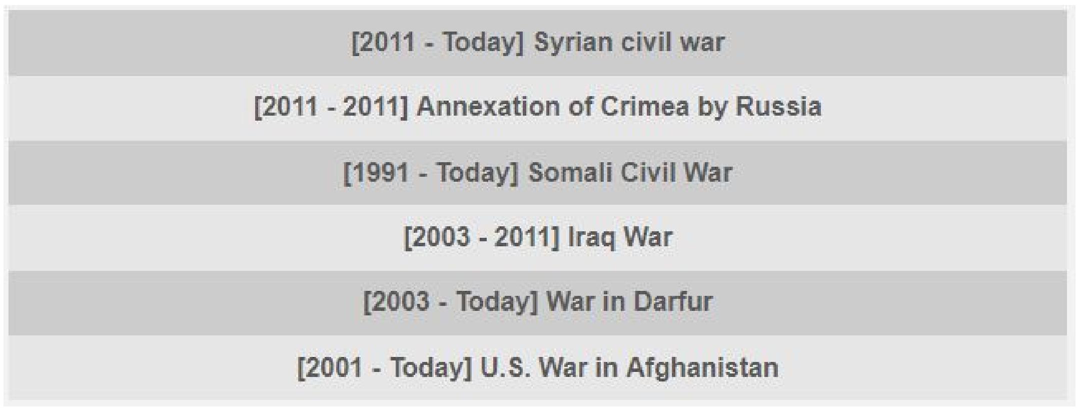
We also added some interaction, the current year is highlighted and if the mouse is on a bar, the exact value is displayed in a small tooltip.



Country graph with highlighted year 2001

Historical events

We added some historical events to give some context. Those events are buttons, clicking on them will perform 2 actions; change the images of the gallery below and zoom on the country where the events occur. Moreover, double clicking on them will open the wikipedia page of the event. The events are displayed if and only if the current time is when they occur.



[2011 - Today] Syrian civil war

[2011 - 2011] Annexation of Crimea by Russia

[1991 - Today] Somali Civil War

[2003 - 2011] Iraq War

[2003 - Today] War in Darfur

[2001 - Today] U.S. War in Afghanistan

Context images

We decided to have a gallery of images to increase the impact of our visualization by giving a glimpse of the reality behind the numbers. The gallery was implemented using a treemap. A treemap is represented as squares, each of which has a weight. The bigger it is, the more space it takes on the visualization. We replaced each of those squares by images.
We then added the following functionality: when a user clicks on an image, it gets bigger and re-clicking on it puts the gallery to its original state (each images have roughly the same weight).
We used image urls directly from wikipedia:

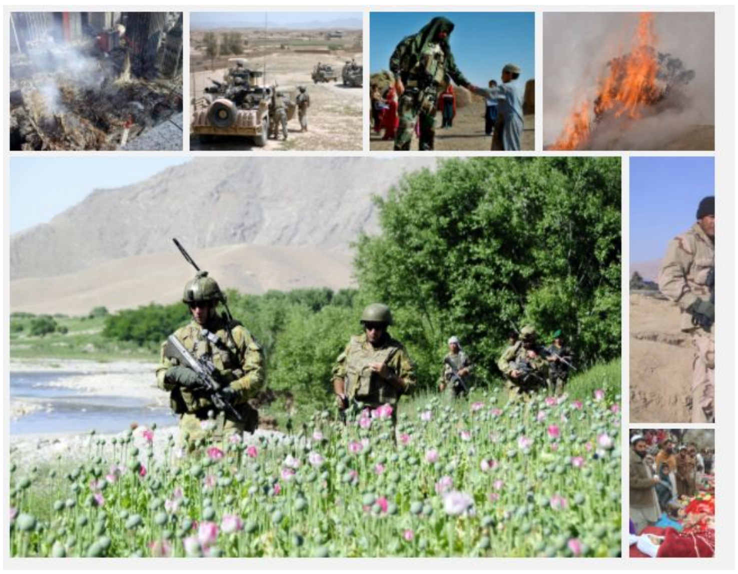


Image gallery with an image selectionned

Evaluation:

The main question was whether there were observable changes in the data that could be linked to historical events like wars. This question was best answered in the late parts of our visualization. The addition of an event list and images highlighted very well the strong correlation between what was happening in our map (in the data) and what historically happened. However, we could hint the user even further to get insight into the interesting events by giving small markers in the timeline on the years corresponding to important events.

Overall, our tool provides a good insight on the data and a fun way to play with the map and the years as a mean to observe changes in the refugee flows. Further improvements would mainly be a stronger emphasis on the user-friendly aspect of the visualization such as a bigger part dedicated to some kind of tutorial and better hints toward parts of the visualization that may be more interesting for a lambda user.

Peer assessment:

We've had the chance to constitute a work group where we knew each other since the Bachelor cycle. Therefore working and communicating together was not an issue at all. More specifically:

Preparation

Preparation ahead of meetings was made through a Telegram group and the backlog of feature to be implemented during the lab meetings was defined ahead of time.

Contribution

Each team member focused on a different part of the project while striving to provide efficient and simple to use interface to others. Specifically:

Isaac Leimgruber: Prepared the general class architecture as well as the layout organisation. Implemented the timeline widget with its cursor, the time lapse animation and the animated year text. Helped on the implementation of the animation of dots on the canvas layer of the map.

Leandro Kieliger: Implemented the Leaflet map visualisation with its additional canvas layer for displaying thousands of dots and other information. Implemented the choropleth visualisation. Gathered data for historical events on wikipedia and implemented the "historical events" buttons.

Quentin Bacuet: Prepared the data for all the needs of the project using python scripts. Implemented the animated graph which shows cumulative amounts of refugee per country.

Searched and implemented the animated pictures which are shown when the user clicks on the historical events buttons

Openness and Flexibility

As mentioned earlier, there was no major conflict that arose during the project. There was however some discrepancies on the technical part of the visualisation for example with the way the flow of refugee should scale relative to the number of individuals. Those discrepancies were resolved by testing different approaches and keeping the best one. Indeed, once the feature has been implemented it is much easier to assess whether it is superior or not. From there, every team member played it fair and accepted the superior solution.