Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme

Assignment 1

Information Visualization & Visual Analytics (WS 2019/20)

Due: Monday, 28.10.2019, 12:00 **Discussion:** Wednesday, 30.10.2019

Please solve the assignment in **groups of up to three (3) students**. Choose one student, who uploads your solution on the assignments page in ILIAS as PDF (for theoretical submissions) or ZIP (for practical submissions Impl). The submitted files should follow the naming scheme `yourlastname1_yourlastname2_yourlastname3` with respective file-ending, of course. Make sure that you create your team before uploading the solution.

Task 1 Differences in visualization disciplines *[Points: 6]*

Characterize the differences between "Information Visualization" and "Scientific Visualization"

Task 2 Historical Visualization *[Points: 8]*

Charles Minard created the "Napoleon's March to Moscow" visualization in the 19th century.

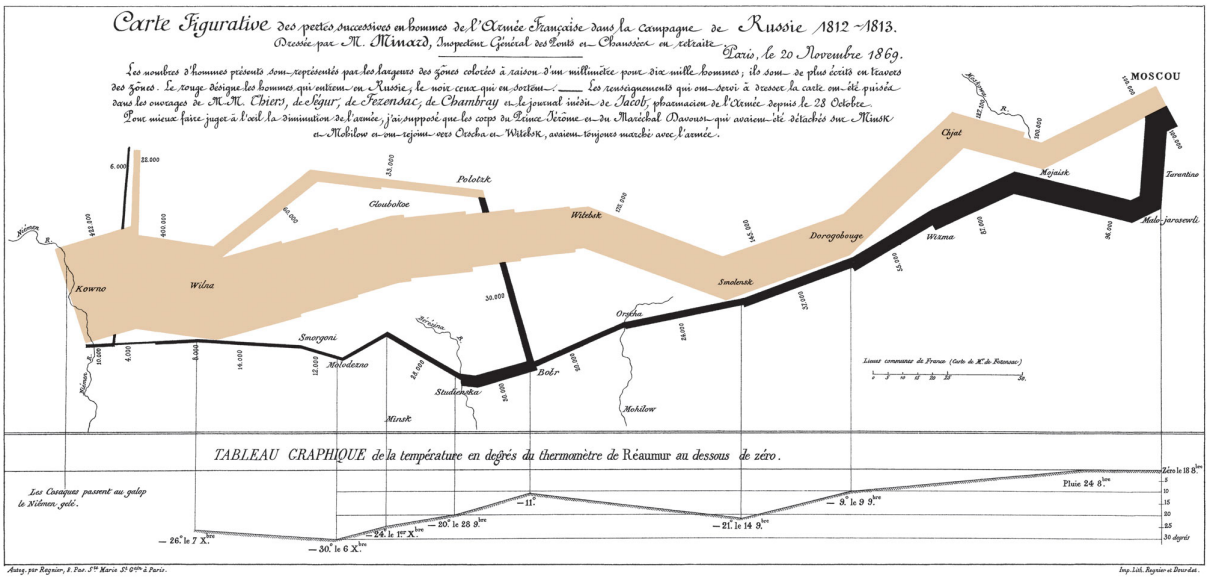


Figure 1: Charles Joseph Minard's visualizaiton of Napoleons campaign.

(a) *(4 points)* List the different "data aspects" on Napoleon's campaign that Charles Minard included in his visualization.

(b) *(4 points)* Is it possible to create an automatic procedure that generates visualizations such as those by Charles Minard automatically from similar data. Discuss if this would be possible and explain your assessment referring to the data aspects identified in the previous question.

Task 3 (Bonus Task): Tag Cloud *[Points: 7]*

A very popular compact visualization to show words together with their frequencies are tag clouds. In this task, you have to implement your own placement algorithm to create a spiral-based tag cloud and visualize it. That means, the tag cloud starts in the middle of the screen and arranges the words along a spiral around the center. You can find some more information about layouting and placement of tag clouds in Feinberg's chapter in *Beautiful Visualization*[1] and in the work of Luboschik et al.[2]

The general idea is to create a spiral and place the given words along the spiral whilst preventing the words from overlapping. The words have a corresponding frequency that has to be mapped to the strings fontsize. Figure 2 shows an example with ten of the given words. For a better clarification, we included the spiral, but it is not necessary to draw the spiral in your submission. Also, we ask you to save as much space as possible (our example uses a lot of space because the spiral grows too fast).

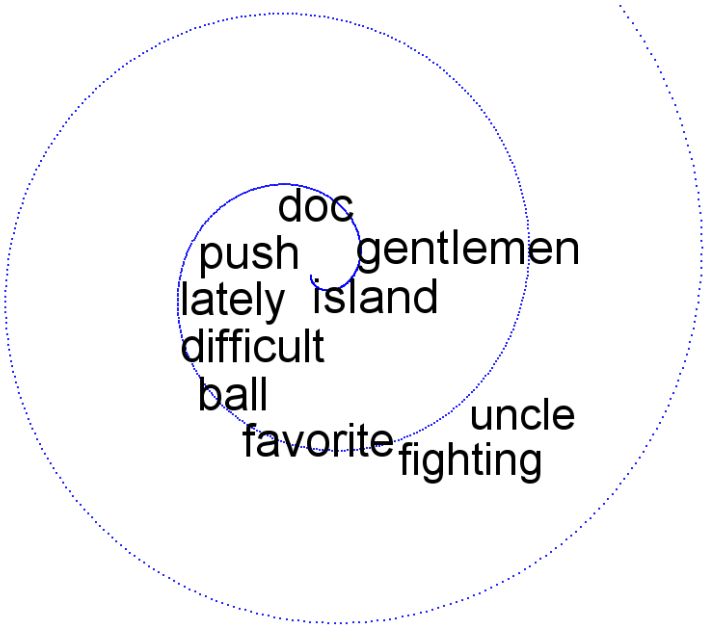


Figure 2: Example tag cloud with a spiral layout.

The assignment folder in ILIAS contains a file `ex1_practical.zip`, which contains a Eclipse project folder. Import the folder into Eclipse. It contains one class, for which all necessary imports have already been added: `Assignment1` in the package `de.uni_stuttgart.vis.submissions.assignment1`. Rename the class to the last names of your group members, in *PascalCase* and alphabetical order; for example, `FrankeKnittelKochMorariu`. To rename the class correctly in Eclipse, right-click the class in the *Package Explorer* and select *Refactor* → *Rename...* Alternatively, press `Shift+Alt+R` on the class. This opens a dialog where you can enter a new name, which renames both the class and the file name.

Data structure for this assignment:

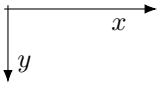
`de.uni_stuttgart.vis.data.DataProvider`
`getWords(): List<WeightedString>` – returns a list of all words with their frequency
`getSentiment(word: String): float` – returns a sentiment score $\in [-1, 1]$ of the word
`de.uni_stuttgart.vis.data.wordCloud.WeightedString`
`getText(): String` – returns the text of the word
`getFrequency(): int` – returns the weight/frequency of the word

[1] Steele, Julie and Iliinsky, Noah: *"Beautiful Visualization: Looking at data through the eyes of experts."*, O'Reilly Media Inc., Sebastopol, 2010. (Chapter 3: Jonathan Feinberg – Wordle)

[2] Luboschik, Martin; Schumann, Heidrun; and Cords, Hilko: *"Particle-based labeling: Fast point-feature labeling without obscuring other visual features."* IEEE Transactions on Visualization and Computer Graphics 14*(6)* (2008): 1237–1244.

Some hints:

- In *display coordinates*, $(0, 0)$ is usually the *top left* corner of the screen. That means that y coordinates get larger *downwards,* which is different to how coordinates are usually handled in mathematics:



- You can get a `java.awt.Rectangle` with a `String`'s bounding box in a specific `java.awt.Font` from our helper class, `de.uni_stuttgart.vis.helper.StringHelper`:

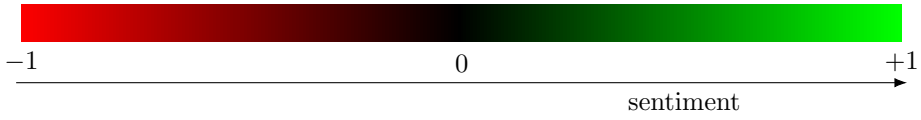
```
1 String s = "Hello World!";
2 java.awt.Font f = ... /* current font & size */;
3 java.awt.Rectangle boundingBox = StringHelper.getStringSize(s, f);
```

Solve the following exercise parts. Submit the final version of your program. There is no need to submit a separate version for each part, as the parts build upon each other.

(a) *(4 points)* Using the provided framework and the algorithm described above, create a tag cloud from the `WeightedString` list. For now, you can ignore the word weighting, and all words can have the same font size. Experiment with the slope of your spiral; there should not be too much whitespace between words (see Figure 2), but at the same time a very small slope will significantly slow down your program. Try to find a good compromise between quality and speed. You do **not** need to draw the spiral explicitely!

(b) *(2 points)* Now, add the weighting of the words into the creation of the tag cloud. Encode the weight into the *font size*. Think about what would be a good maximum font size, and calculate the font sizes accordingly. Sort the `WeightedString` list by weight in descending order, so that the largest word is placed first.

(c) *(1 points)* You have encoded the *score* of a word in its font size. Now suppose that we have analyzed the texts regarding whether words are used in a positive or a negative manner. Each `WeightedString` also has a *sentiment*, which is a number between -1 (negative) and 1 (positive), which can be retrieved using the `getSentiment(str)` method of the `DataProvider` object. A word with a sentiment of -1 should be colored **red**, a sentiment of 0 is mapped to **black**, and a sentiment of 1 is mapped to **green**. Sentiment values inbetween are interpolated linearly between the colors. *Tip: Interpolate each color channel individually. You can call `java.awt.Color`'s constructor with three `int` values, representing red, green and blue value. The values are between* 0 *and* 255. The color gradient should look like this:



Modify your code to color all words in the tag cloud based on their sentiment value.

(d) *(0 points)* Suppose you would create the list of `WeightedString`s yourself, by counting the occurences of terms in a text, or collection of texts. What would you need to consider regarding the number of words you would count? Does the weighting correspond linearly to the number of times the word appears, or would you calculate the weighting differently? Why?
You do not need to submit an answer for these questions now. Instead, we will discuss them together in the next exercise tutorial.

Please submit your result as an archive (`*.zip`) of your Eclipse project and a corresponding `readme.txt`. The `readme.txt` has to include your names, affiliations, and matriculation numbers.