Practical Work Report

Sven Ligensa Camille Alazard Lige Zhao

Winter Term 2024/25

This visual analytics app was developed by Sven Ligensa, Camille Alazard and Zhao Lige in the course *Data Visualization* at the UPM in the academic year 2024-2025. The source code is made available on GitHub and deployed on shinyapps.io. We used Shiny for Python as the framework, while the visualizations utilize different libraries, as discussed in Section 5.

Game of Thrones, adapted from the novel series *A Song of Fire and Ice* written by George R.R Martin, is one of the most famous fantasy series that ever existed. Produced by HBO the series has a total of 73 episodes broadcasted to the public in 8 seasons from 2011 to 2019. Between murder, treason, secrect alliances and family stories, fans were captivated by the complex web of political conflicts among the noble families of Westeros seaking the power to sit on the *Iron Throne* and reign over the *Seven Kingdom*.

However, the vast and complex world of *Game of Thrones*, which makes the show popular, is also its fatal defect. A lot of people dropped the show after some seasons because they were lost or they forgot the story from a season to another. This work aims at giving tools for the lost audience to help them following the story and avoid dropping it. It can also be considered a recreational tool for the experts of the series.

Credits

The data used in this project was obtained from this repository by Jeffrey Lancaster. We appreciate the effort he put into creating this dataset and thank him for releasing it as open-source. The map of Westeros and Essos was obtained from HBO's website.

Contents

1 Analysts Questions

The fans can have multiple questions about the series, which are tackled by this visual analytics application:

1. Where do the characters travel in the *Known World* (Westeros and Essos) and how much time do they spend at various places?
2. What are the relationships between the characters, including family relationships, alliances and animosities?
3. How does the screentime of a character evolve throughout the series, and how does it compare to other characters' screentime?

2 Data Abstraction

The dataset consisted of the following *Dataset Types* and *Abstract Data Types*:

2.1 Datasets

- Characters (Table)
- Episodes (Table)
- Locations (Geometry)
- Relationships (Network → Multilayer Graph)

2.2 Abstract Data Types

- Character (Item)
  - Name (Text)
  - House (Categorical)
  - Image
- Episode (Item)
  - Identifier (Text)
  - Title (Text)
- Location (Item)
  - Coordinates on Map (Position)
  - Name (Text)
- Screentime (Qualitative; In minutes; Relates Character, Episode, Location)
- Relationship (Categorical: ParentOf, Siblings, Kills, Married, Serves, GuardianOf, Allies; Link between Characters)

3 Task Abstraction

Note: All of our visualizations fall into the *Enjoyment* category, and they can be used by fans of the series to gain a deeper understanding and connect more deeply with the series.

3.1 Character Travel

Actions

- Use: *Enjoy* (Understand the physical space of the Known World better.)
- Search: *Locate* (The target you want to look for is known: the selected characters. The location in the visualization is not known, as this is the insight the user wants to get: Where did a character stay for how long?)
- Query: *Identify* and *Compare* (Trace the path of a single or multiple characters.)

Targets

- All data → *Trends* (Is there a trend where certain characters spend their time?)
- All data → *Features* (What patterns emerge for a character or when looking at multiple characters?)
- Single Attribute → *Distribution of all values* (How is the location of the characters distributed?)

3.2 Relationships

Actions

- Use: *Enjoy* (Understand the relationships of the characters better.)
- Search: *Browse* and *Explore* (You don't necessarily know what you are looking for, e.g. To whom was Tyrion Lannister married?, and maybe even not what the target is: Who has killed the most people?)
- Query: *Identify* and *Compare* (See the relationships of a single or multiple characters.)

Target

- *Network Topology* (as we deal with a multilayer graph of character relationships.)

3.3 Screentime

Actions

- Use: *Enjoy* (Understand when which characters were active in which seasons or episodes.)
- Search: *Explore* (The user freely searches for patterns in the screentimes of characters.)
- Query: *Identify*, *Compare*, and *Summarize* (View the screentimes of as many characters as desired.)

Targets

- All data → *Trends* (Is there a trend regarding the screentime of characters?)
- Single Attribute → *Distribution of all values* (How is the screentime of the characters distributed?)
- Multiple Attributes → *Similarity* (Which characters appear together?)

4 Idioms (Visual Encoding + Interaction)

4.1 Character Travel

We encode the locations by using their positions (*Encode → Arrange → Use*), and show a **map** of "The Known World" as the background.

Marks

- *Circle* for time spend at a location
  - More natural to display locations on a map and can be nested more easily → Favored over Bars
- *Line* (Arrow for travel direction) between location for travel path

Channels

- Location → *Spatial position (2D) on unaligned scale*
- Screentime → *Area* (of Circle)
- Number of travels → *Width* (of Line)
- Character → *Hue* (of Circle and Line)

Interactivity

- *Select* up to three characters to be displayed
- *Select* start and end episode of interval to be considered
- Show or hide the circles/travel paths/map background
- *Embed*: Click on location to show the exact amount of time spent there by the characters and the incoming/outgoing travel paths

4.2 Relationships

We visualize the multilayer graph with the means of a **Force Directed Graph**, as we focus on the network topology and the algorithm can effectively place the nodes.

Marks

- *Point/Picture* for Character
- *Line* for relationships between Characters

Channels

- Amount of interaction between characters → *Spatial position (2D) on unaligned scale* (determined by Force Directed Graph algorithm)
- Relationship → *Hue*

Interactivity

- *Select* characters to be displayed (either in sidebar or by clicking on the character in the graph)
- *Filter* types of relationships to be displayed
- Show or hide character pictures
- Hover over character/relationship to see details

4.3 Screentime

We visualize the characters' screentimes in a **Cluster Heatmap** with the keys "character name" and "episode".

Marks

- *2D area* for screentime

Channels

- Character + Episode → *Spatial position (2D) on aligned scale*
- Screentime → *Luminance / Saturation*

Interactivity

- *Filter* characters based on their total screentime (to only focus on more/less prominent characters)
- Hover over cell to see details

5 Algorithmic Implementation

Before the data could be used for visualization, it was preprocessed to include only the information we need and in the correct format (done in the `data` folder in the source code; afterwards, the files from `data/processed` are copied to `visual-analytics-shiny/data`).

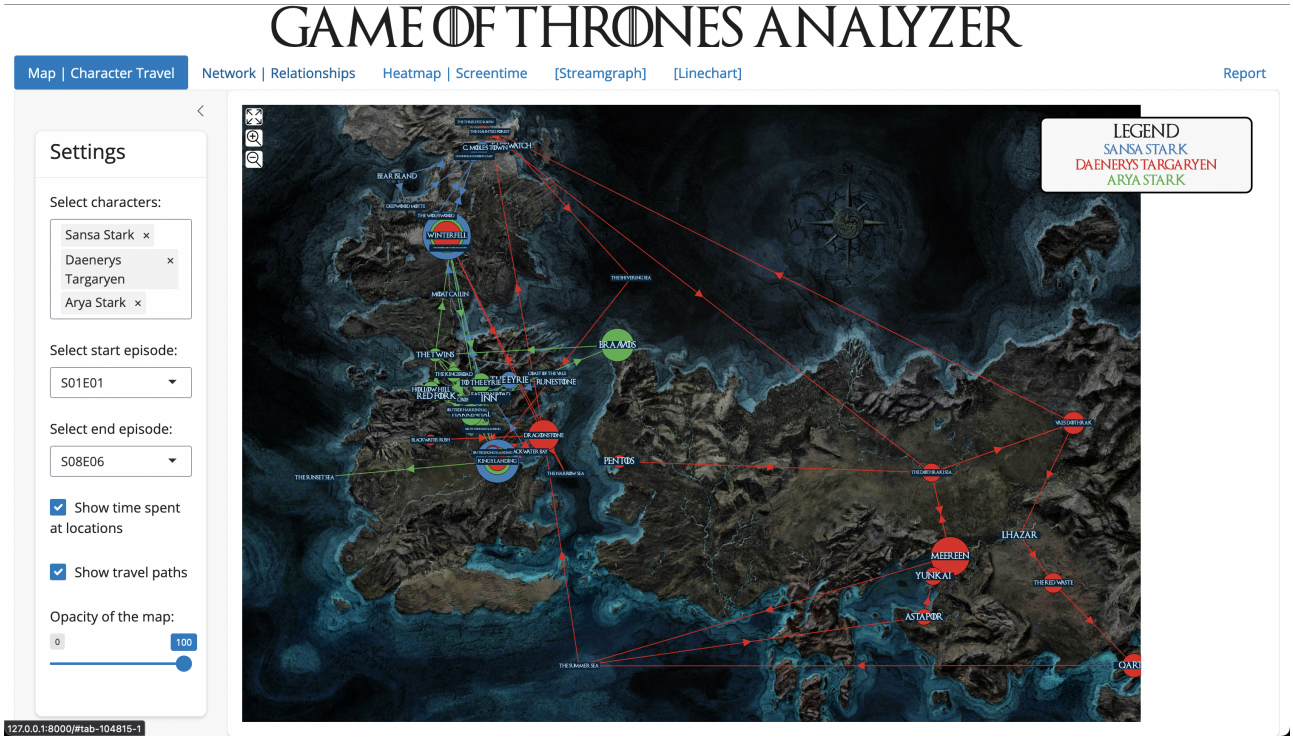
The main file of a Shiny Python application is `app.py`. Here, we load the data (and CSS and JS code located in other files), define the UI layout (`ui.page_fluid`), and the reactive elements (`server` function).

The **UI** has four parts: the title (always shown on the top), the navigation panel (letting the user switch between the visualizations) right below the title, the minimizable settings sidebar (letting the user set the inputs for the visualizations) on the left, and the visualization area (taking up the remaining space). The input types are checkboxes, sliders, and most importantly multi-select text input fields with search functionality to quickly find characters or seasons.

The **server** function handles changing user inputs, filters the data accordingly, and displays the visualization. Two variants exist: (1) When using a Python library (`plotly` or `matplotlib`), the visualization is created directly inside `app.py`, (2) when using a JavaScript library (`d3` or "handmade" SVG), an asynchronous message is sent, which is listened to by JS code.

Generally, we use ColorBrewer2's qualitative color pallete with seven colors (Network, Map (subset of 3 colors), Linechart, Streamgraph) for qualitative variables, and the "Blues" colormap for the quantitative variable screentime (Heatmap).

5.1 Character Travel



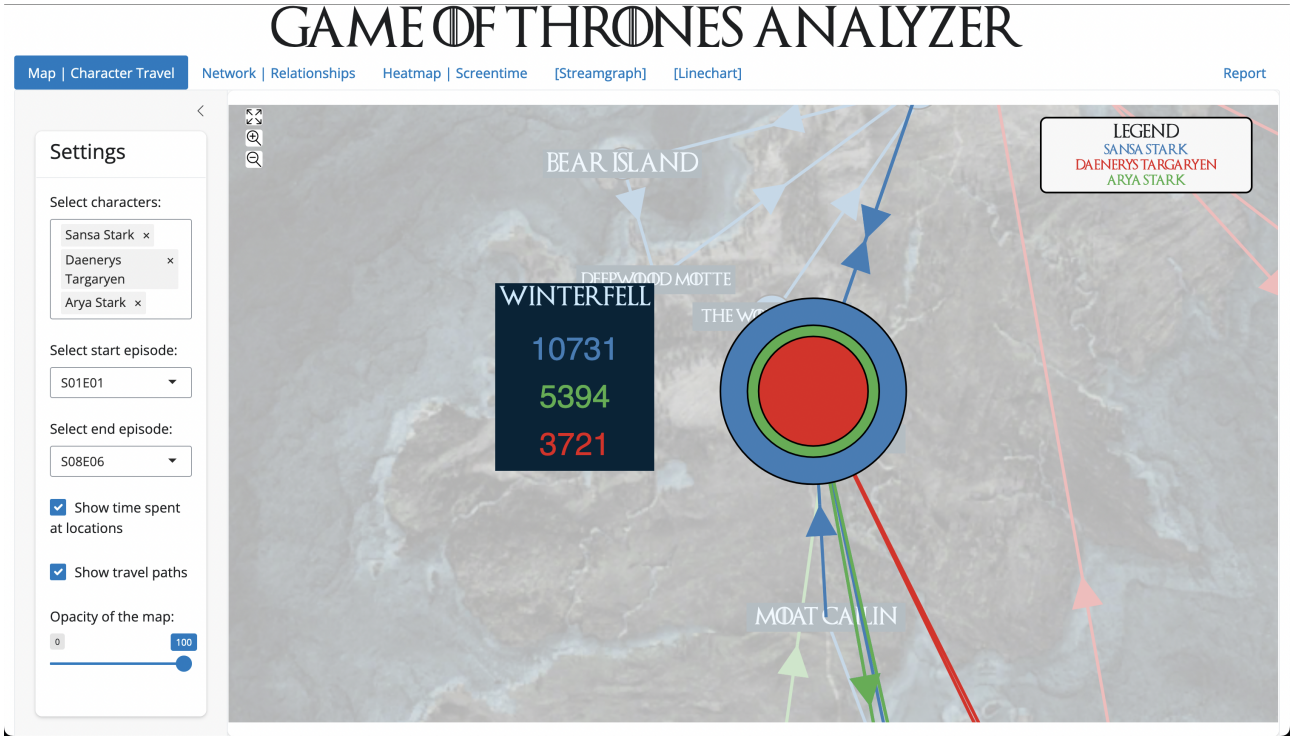
Obtaining an extensice and visually appealing map of Westeros and Essos turned out to be more difficult than expected. To obtain the PNG file we used, we followed the following steps:

1. Visit https://www.hbo.com/house-of-the-dragon/map-of-westeros
2. Inspect → Network tab → https://hotd-interactive-map.micro.hbo.com/models/gltf/Map_etc1s.glb
3. View in: https://gltf-viewer.donmccurdy.com/
   - Adjust angle and lighting settings
4. Take screenshot

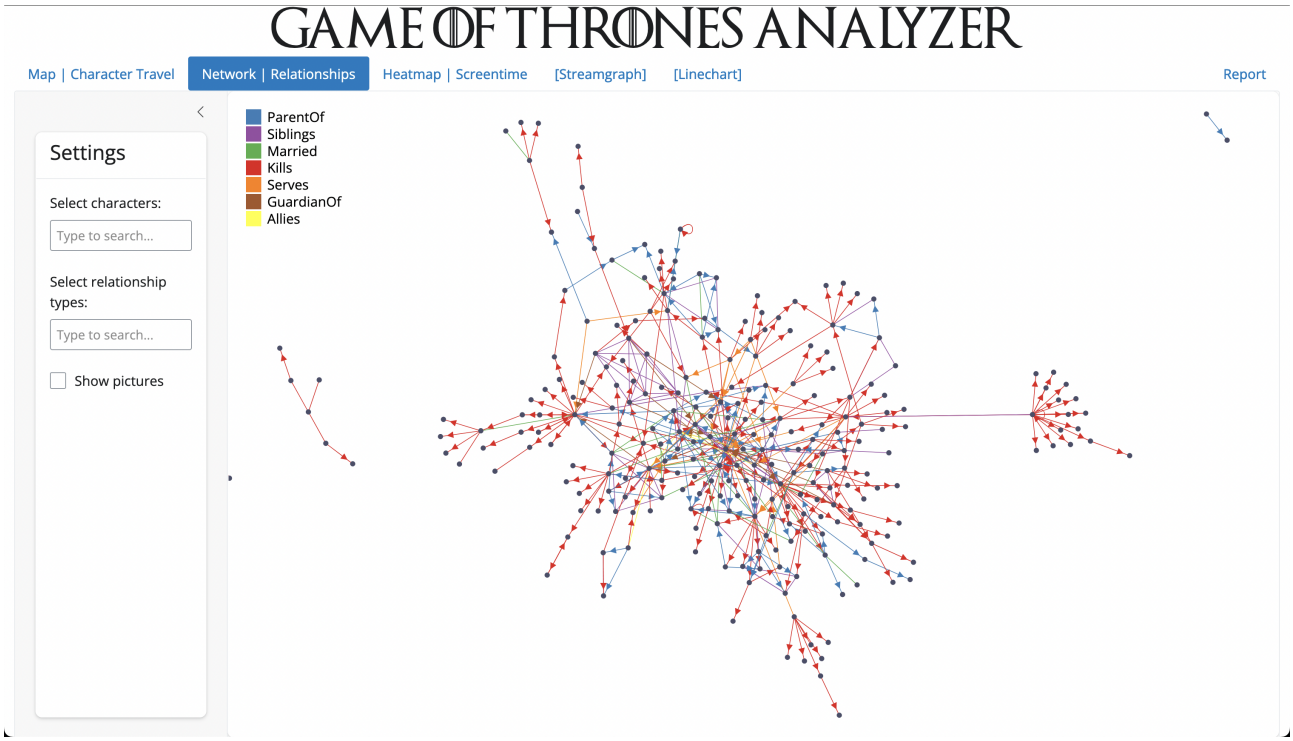
The pixel positions of the locations were then determined by hand and saved in `locations.csv`. The circles and arrows are then drawn on a top layer SVG aligned with the image of the map. The user can fit the image to the screen width/height, and zoom in and out with the controls on the top left. In general, the arrows are drawn first, then the circles and lastly the location labels. To avoid occlusions (a bigger circle hiding a smaller one, for example), the ordering of the circles is determined per location.

When clicking on a location label, the relevant information is highlighted and the rest fades into the background. This is achieved by overlaying a white square with 70% opacity over the map and redrawing the SVG elements with certain classes on top.

To focus more on the character movements and less on the topology of the map, the user can reduce the opacity of the map.



5.2 Relationships

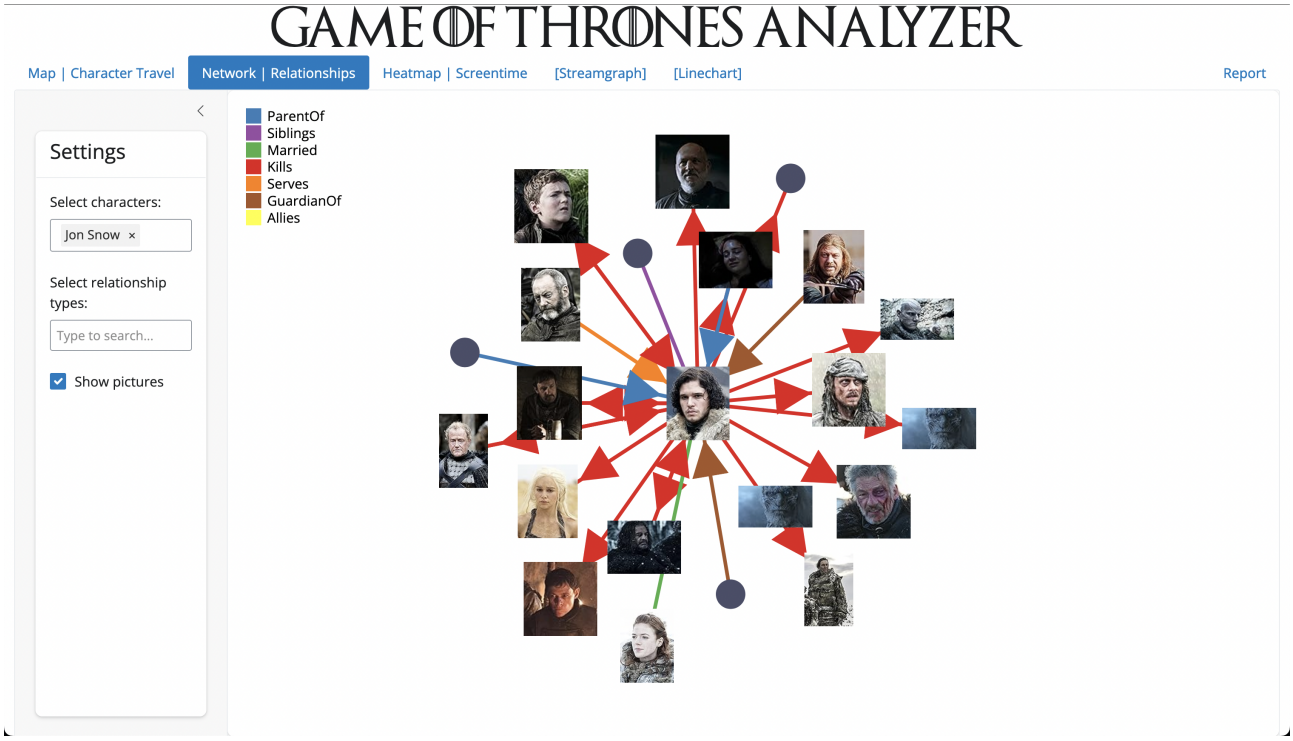


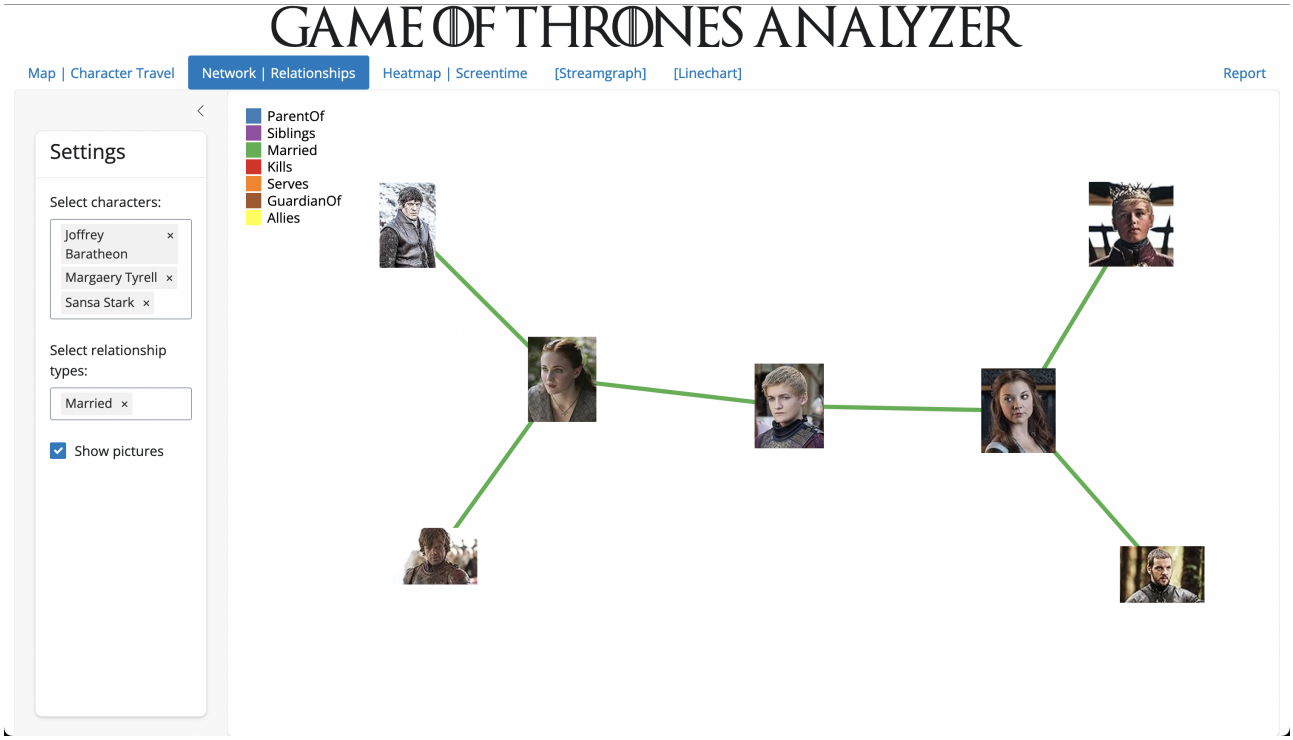
We visualize the relationships between the characters—which can be seen as a Multilayer Graph—where the characters (a set of nodes) is connected via multiple relationship types (types of links) by means of a force-directed graph. The user can focus on specific layers of the graph by selecting the relationship type(s) and/or on a subset of the nodes by selecting characters (either by clicking on them in the graph or via the settings panel). Zooming in and out, as well as displaying information on hover, is also supported. The characters' images can be disabled for fewer distractions. As the visualization library, we use d3, which lets us describe the properties of the force-directed graph on a high-level and creates an SVG as a result. We

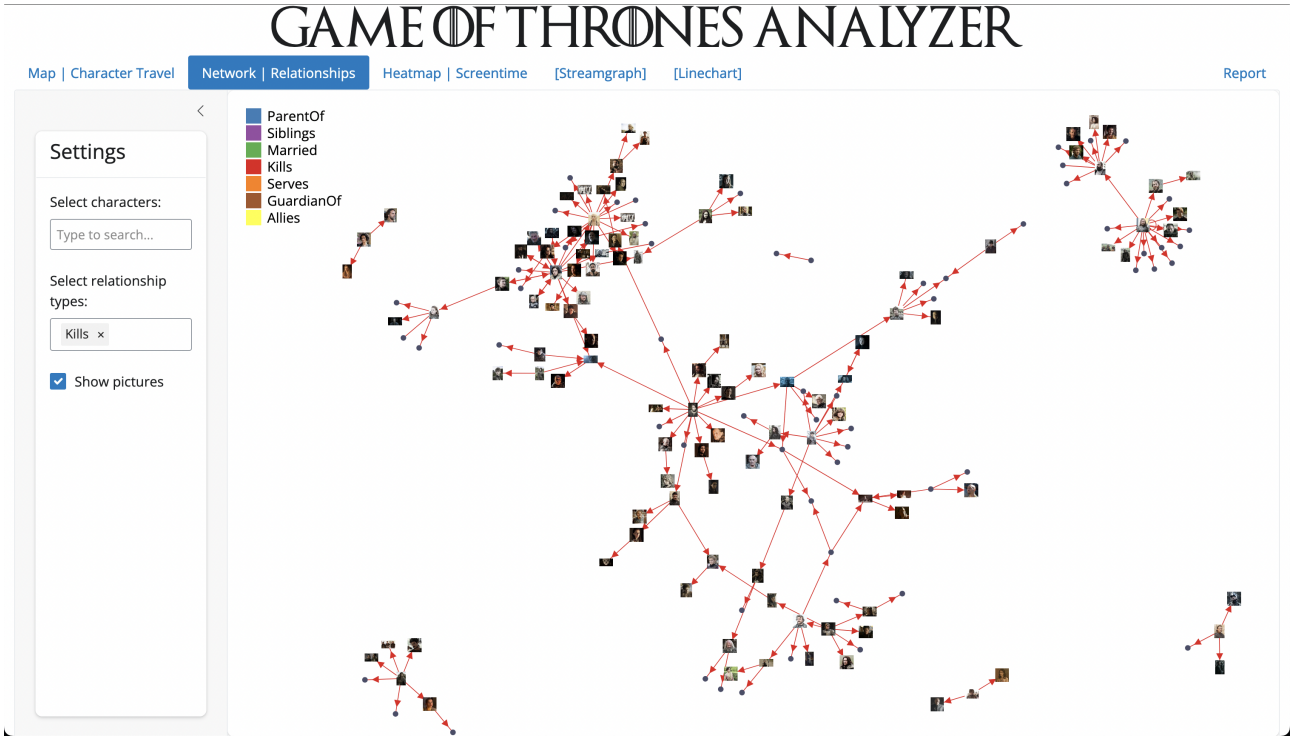
tweaked the variables of the algorithms (forces of links, between nodes, towards center, ...). Nevertheless, we are not able to avoid cluttering due to the vast number of nodes and edges. The user needs to zoom in on what they are interested in.

Bidirectional relationships are displayed as lines, while unidirectional ones are displayed as arrows. Loops (edges where source and target are the same) are also handled correctly (mainly suicides).

The following screenshots show (1) filtering by character, e.g., Jon Snow, (2) filtering by characters and relationships, and (3) filtering by relationship, e.g., Kills.

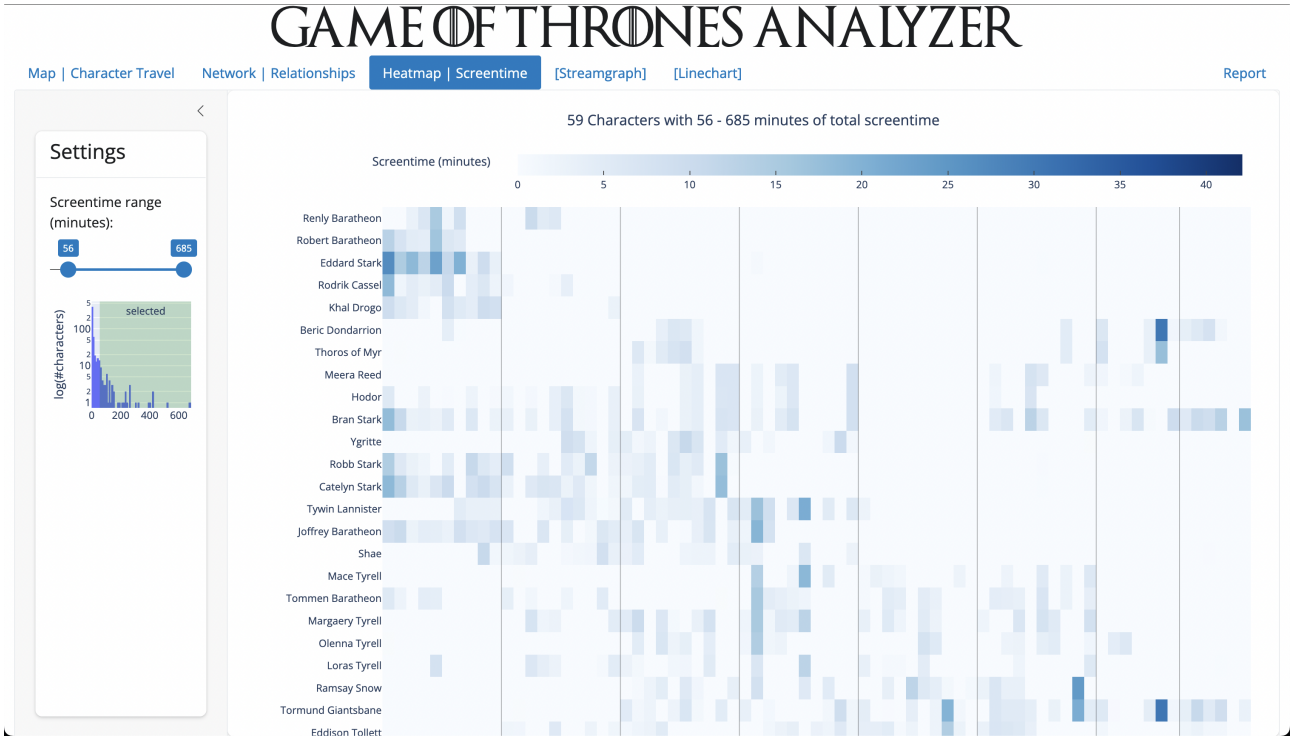


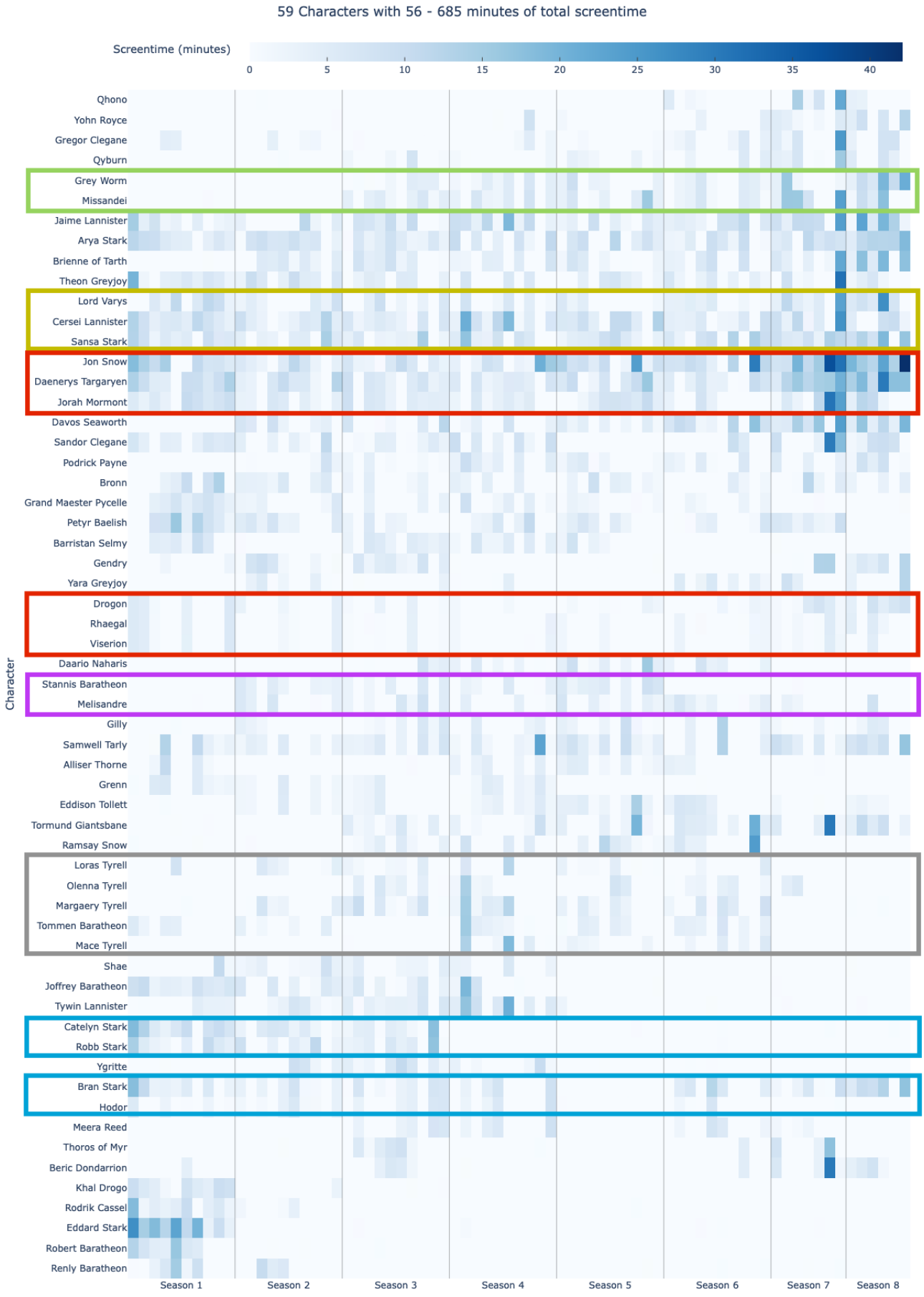




5.3 Screentime

The heatmap showing comparisons between screentimes of the characters is implemented with the help of `plotly`. The x-axis is the ordered attribute "episode". Along the y-axis, we have the characters. As the characters have no inherent ordering–which is critical to effectively compare screentimes between adjacent characters–we need to establish one ourselves. We derive a meaningful ordering by performing hierarchical clustering according to Ward's method on the normalized screentime matrix. The resulting dendrogram determines the ordering of our characters on the y-axis. The following image shows that the clustering yields an ordering which at least partly corresponds to our gut feeling of which characters appear similarly.




59 Characters with 56 - 685 minutes of total screentime
Screentime (minutes)
0
5
10
15
20
25
30
35
40
Character
Qhono
Yohn Royce
Gregor Clegane
Qyburn
Grey Worm
Missandei
Jaime Lannister
Arya Stark
Brienne of Tarth
Theon Greyjoy
Lord Varys
Cersei Lannister
Sansa Stark
Jon Snow
Daenerys Targaryen
Jorah Mormont
Davos Seaworth
Sandor Clegane
Podrick Payne
Bronn
Grand Maester Pycelle
Petyr Baelish
Barristan Selmy
Gendry
Yara Greyjoy
Drogon
Rhaegal
Viserion
Daario Naharis
Stannis Baratheon
Melisandre
Gilly
Samwell Tarly
Alliser Thorne
Grenn
Eddison Tollett
Tormund Giantsbane
Ramsay Snow
Loras Tyrell
Olenna Tyrell
Margaery Tyrell
Tommen Baratheon
Mace Tyrell
Shae
Joffrey Baratheon
Tywin Lannister
Catelyn Stark
Robb Stark
Ygritte
Bran Stark
Hodor
Meera Reed
Thoros of Myr
Beric Dondarrion
Khal Drogo
Rodrik Cassel
Eddard Stark
Robert Baratheon
Renly Baratheon
Season 1
Season 2
Season 3
Season 4
Season 5
Season 6
Season 7
Season 8

The number of characters considered can be reduced by setting a range of minimum and maximum total screentime. This way the user can focus on more or less prominent characters. The histogram below the range slider shows how many characters fall into that range (we use a log-scale for the y-axis). Hovering over a cell displays details. Small vertical black lines make the distinction between different seasons easier.

6 Discussion

We also experimented with a Linechart (created with `plotly`) and a Streamgraph (created with `matplotlib`) to visualize the characters' screentime, but they are not to be viewed as part of the final work.

There are some aspects of the application which could be improved:

- More consistency regarding the look and feel across the visualizations
  - The problem arised due to the experimentation with different frameworks ("hand-made" SVG, `d3`, `plotly`, and `matplotlib`)
- In the Map, occlusions of locations are possible due to a large amount of locations in "The North"
  - The problem was reduced (but not completely fixed) by manually moving the locations a bit further apart
  - Implementing jittering manually was too time consuming
- In the Network, lots of distractors (in form of crossing lines) may occur
  - The user needs to filter the displayed nodes and edges accordingly or play with the force-directed layout
- In the Map (and Linechart and Streamgraph), the color coding of a character changes, when characters (which have been added before) are removed
- Allowing for more sophisticated filtering in the Heatmap and reordering of the rows