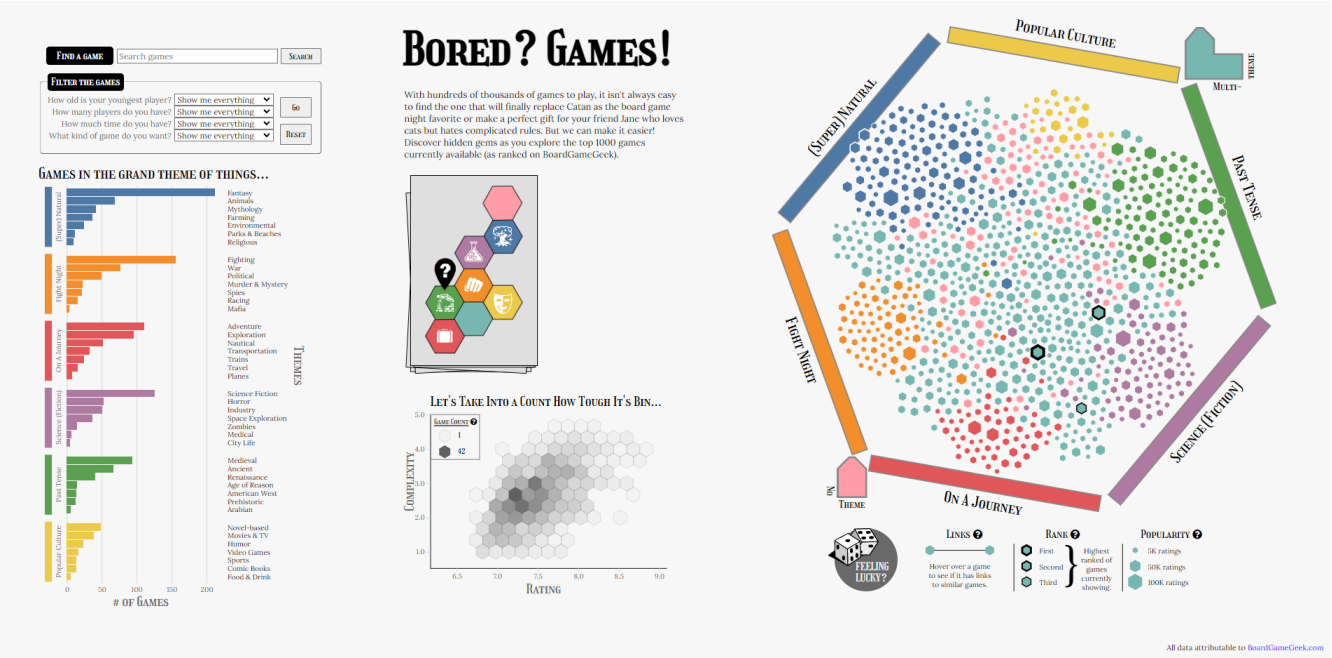
# Bored? Games!

Milestone 3 Final Report | Group 22 | Nicole Gaboury, Michelle Langlois, Andrew Tsai

## 1. Overview

With over 130,000 board games listed on the popular website BoardGameGeek, it isn't always easy to find the one that will finally replace Catan as the board game night favourite or make a perfect gift for your friend Jane who loves cats but hates complicated rules. To help board game lovers and newbies alike explore and map the board game landscape, we have created an app visualizing the top 1000 ranked games on BoardGameGeek. Our visualizations focus on the stories games tell, on finding the sweet spot between how highly rated a game is and how difficult it is to play, and on the similarities between games. Through widgets and linked interactions between views, users can iteratively narrow down their map of games to ones that are a good fit for them, then explore those games and their common links. Our hope is that users will discover hidden gems – and have fun doing so.




## 2. Data

Our original data is sourced from Kaggle. Specifically, we are using the *games_detailed_info.csv* file, which includes detailed information about board games that was scraped from the BoardGameGeek website in January 2022.

### Data Preprocessing Pipeline

The original dataset was a table with 56 columns/attributes and over 21.6K items. Since we wanted to use a force-directed graph, which does not scale well past 1000 nodes, we chose to use a subset of

1000 games. We did our preprocessing in python using the numpy, pandas, and sci-kit learn libraries and saved the output as a json file to make it easier to deal with the attributes that are lists. Our data preprocessing pipeline includes the following steps:

1. Drop 31 columns because: (a) they were redundant with attributes we are using; (b) they were missing for too many games; (c) we decided they were not important to our visualization goals.
2. Rename some attributes for clarity (e.g. *primary -> name*).
3. Derive type attributes by converting 8 existing ordinal rank columns (the game's rank among games of a specific type or blank if not of that type) to boolean attributes (1 if has type, 0 if not)
4. Filter the games to those with the top 1000 rankings.
5. Process and extract theme data by:
   a. extracting information about themes from the *boardgamecategory* column (which includes theme, skills, component, activity, and type information) based on BGG guidance;
   b. filling in theme information for some games without a theme by using information extracted from the existing *boardgamefamily* column (which also lists themes);
   c. combining some themes together (all wargame themes into "war");
   d. deriving 6 new attributes representing whether or not each game belongs to one of 6 thematic clusters/groups that we created to link common themes together, with each attribute being a list of the themes extracted in steps *a, b*, and *c* belonging to that group.
6. Create links between games by calculating the pairwise cosine similarity between each pair of games based on common elements in the original *boardgamecategory, boardgamefamily, boardgamemechanics, boardgamedesigner*, and *type* attributes. We retained all links with a cosine similarity of > 0.4 to create an attribute, *links,* that is a list of these links in the row for each board game, with each link represented as an object {source: *gameID*, target: *gameID*, weight: *weight*}. The threshold was chosen because it gave us an average of 6 similar games per game, which we felt was reasonable for our visualization; making the threshold higher lost information and lower gave us too much noise.
7. Derive a *url* attribute from the *id* attribute, which is the BoardGameGeek url for the game, in the form "https://boardgamegeek.com/boardgame/{id}".

Our final preprocessed dataset contains 1000 items and 28 attributes.

In addition to the preprocessing above, we also process data in the browser on the fly when creating and updating the barchart and hexbin visualizations, which both display aggregated counts of games. Aggregation for the hexbin is done by the D3 library we used. Aggregation for the barchart was done with custom code since D3's rollup function could not handle one game belonging to multiple themes.

# Description of data

| **Attribute** | **Type** | | | **Cardinality / range** | **Notes** (see details above when attribute is derived) |
|---|---|---|---|---|---|
| | **Categorical** | **Ordinal** | **Quantitative** | | |
| Basic game information: 5 attributes | | | | | |
| id, name, thumbnail | x | | | 1000 levels each | |
| year | | x | | -2200 to 2021 | publication year (or creation: e.g. Go in 2200 BC) |

| | | | | | |
|---|---|---|---|---|---|
| url | x | | | 1000 levels | derived |
| Game play parameters: 4 attributes | | | | | |
| min_players | | | x | 1 to 6 | publisher's suggestion |
| max_players | | | x | 1 to 100 | publisher's suggestion |
| min_age | | | x | 0 to 18 | publisher's suggestion |
| playing_time | | | x | 0 to 1200 minutes | publisher's suggestion |
| Detailed game information: 15 attributes | | | | | |
| type_strategy, type_family, type_party, type_abstract, type_thematic, type_war, type_customizable, type_children | x | | | 2 levels (0 false, 1 true) | derived<br><br>each game has 1-3 types |
| theme_cluster_history | x | | | 14 levels, including [ ] | derived<br><br>each attribute is a list of 0-7 themes<br><br>each game has non-empty lists in 0-5 clusters |
| theme_cluster_nature | x | | | 19 levels, including [ ] | |
| theme_cluster_journey | x | | | 22 levels, including [ ] | |
| theme_cluster_culture | x | | | 12 levels, including [ ] | |
| theme_cluster_science | x | | | 14 levels, including [ ] | |
| theme_cluster_war | x | | | 20 levels, including [ ] | |
| complexity | | | x | 1.0251 to 4.734 | a measure of game difficulty |
| Rating and rank information: 3 attributes | | | | | |
| users_rated | | | x | 1285 to 109,006 | our proxy for popularity |
| rating | | | x | 6.67 to 8.84 | average of user ratings |
| rank | | x | | 1 to 1001 | each game has a unique rank; rank does not directly match rating but takes into account other factors and is determined by BGG (i.e. the game with rank 1 does not have the highest average rating) |
| Similarity measures: 1 attribute | | | | | |
| links | x | | | 900 levels, including [ ] | derived<br><br>a list of links with 0-80 other games<br><br>each link can itself be considered an item with three attributes:<br>- source: categorical, 899 levels<br>- target: categorical, 899 levels<br>- weight: quantitative, 0.4 - 1.0<br><br>there are 8662 link items (4331 unique links since source -> target != target -> source) |

# 3. Goals and Tasks

Our project supports several tasks, described below in domain-specific and abstract language. The overall goal of the application is to help users find new board games they might like, encouraging users to **{enjoy}** and **{explore}** rather than quickly find information.

| # | Domain-specific | Abstract |
|---|---|---|
| 1 | Bob is looking for a new game to play with friends who just got back from a trip. He wants to know what themes are featured in games currently available, which ones are most/least common, and how many games have each theme. | {discover distribution} |
| 2 | Bob likes games that will not take too long to learn to play but that are highly rated. However, he is willing to spend a bit more time learning to get a *really* good game. He wants to understand the tradeoffs between rating and complexity to find games that meet his needs. | {discover distribution} {identify correlation} |
| 3 | Bob has filtered games to see only ones with interesting themes and with the right complexity/rating tradeoff. He now wants to browse through these games, find ones that are popular and see how games are connected to each other. | {explore network} {discover similarities} |
| 4 | Bob remembers his friends really loved playing Catan. He wants to know what games might be similar and how it compares with other games in terms of complexity, rating, popularity, and themes. | {query nodes} |

# 4. Visualization

## Overview

Our website contains three views: a force-directed graph showing individual games, a grouped bar chart showing aggregated data, and a hexbin chart showing aggregated data. It also contains a set of global filters, a search bar, a button for random game exploration, and various tooltips to provide additional information to the user. Our visualization looks best on a screen size of 1920 x 1080.

## Description and rationale of views and interactions

### General interactions

All our views are bidirectionally linked to provide users with many avenues for exploring the data. **Filtering** generally flows from left to right: using the global filters limits data shown in all the views, using the barchart as a filter limits data shown in the hexbin and graph, and using the hexbin as a filter limits data shown in the graph. Conversely, **linked highlighting** generally flows back from right to left:

selecting a game in the graph highlights relevant parts of the hexbin and barchart, and selecting a hexbin highlights relevant parts of the barchart. We found this flow to be intuitive, helping users to expect how selections made in various views or widgets might change other views.

Applying a global filter first resets the data in the whole visualization back to the full dataset. We decided to do this because filters should start with everything and then remove items based on the filters selected. Also, we did not want to cause confusion for the user about how the global filters interact with the filter interactions the other grouped bar chart and hexbin chart provide.

**Searching** or **clicking the random game button** works within the currently filtered dataset. This way when the user has applied the filters they wish to have, the search doesn't undo the filters, it just searches within the subset. If a game is found but isn't in the subset of data, then the user is notified that the game exists but just not within the subset. The search bar has an autocomplete function that shows users matching games as they type; the search bar can therefore also be used as a way to see the names of games currently on screen.

## View 1: Hexy Game Network (Innovative View)

This view primarily supports tasks 3 and 4, described above. The view is a **force directed graph with two extensions**: (1) the initial location of the nodes is not random but is **constrained by a grouping force** that tries to place nodes with the same theme group together; (2) rather than show all links, we use **dynamic visual layering** to only show one-hop neighbors for selected/hovered games.

### Rationale for view

We decided to use the force directed graph since tasks 3 and 4 require a way to explore games and then find similar games to a chosen game. A graph is easy to explore and allows the user to see nodes that are one hop away from (similar to) the target node. Since graphs are commonly used for representing networks, the average user will understand how to read the graph, which is important because our visualization is geared towards a general audience. We also thought a graph visual was an enjoyable way to explore the data. The fizzy movement as the layout settles is fun to watch and is quick enough to not be frustrating to the user.

### Analysis of marks and channels

Each **point mark** represents a game. The hexagonal shape of the mark is a stylistic choice to tie in with our overall design and does not encode anything. There are three attributes encoded:

- Theme group (via **hue**): We chose to encode theme group membership with hue since this is an effective identity channel that was scalable to the cardinality of the attribute (there are 6 groups). One issue with this design choice was that some games belong to multiple groups or no groups, so we had to use two more colors to represent these possibilities at the expense of losing some detail for the multiple group games (users cannot immediately see which theme groups they belong to). We used a categorical color scale with high saturation and good discriminability since we have small regions of color mixing together.

- Popularity (via **size**): We chose to encode popularity (number of users who rated the game) with size (2D area of hexagon) since this is an effective magnitude channel. We originally thought we might use only three sizes of nodes (not very, medium, and very popular), corresponding to cutting the extent into three pieces via D3's threshold scale, but decided in the end to use a square root scale to directly encode the attribute. Although we recognize that it is hard to distinguish small differences in size with 1000 marks that are not aligned along a common scale, we felt that this final encoding was more useful to users when comparing games. They are probably not interested in the exact number of users who rated the game, and the square root scale gives the users a nice overview of which games had much higher ratings versus those that had the lowest ratings.

- Rank (via **line width of border**): We wanted to give users a way to identify the top 3 ranked games in the graph and chose to do this by placing a dark border around the marks representing the first, second, and third top ranked games currently visible. A thicker border indicates a higher rank. We chose to show only three ranks to limit visual clutter and emphasize popout for the top games.

**Spatial region** does not directly encode one attribute since the location of the node is affected by both the theme group membership *and* the game's similarity links to other games. However, the combination of these two forces does indirectly help convey a notion of clustering. That is, games generally tend to cluster with others with the same theme group and to get pulled towards the central edges of the clusters when they are similar to games outside their theme group. Clusters of similar games also form within theme groups.

Each **connection line mark** represents a weighted similarity link between two games. We chose not to specifically encode the weight of the link. Although we considered doing so via line width, this did not work because thick line marks occluded small point marks. Similarly, encoding weight via saturation was not visible enough to be useful. We also chose to show the connections on demand to limit the occlusion that would otherwise occur when visualizing a network with an average link density of 6 links per node in a small space.

### Analysis of interactions

There are three interactions for this view:

- Users can **hover over a game** to see info about that game, displayed in a fixed location on the screen instead of in a tooltip to avoid hiding the graph. Hovering also reveals the game's links to one-hop neighbors. This interaction allows users to see whether games are similar to many games or to no games, and to get initial information about a game, helping with exploration.

- Users can **select a game** by clicking on it. This also reveals links to one-hop neighbors and info about the game, displayed in the same fixed location on the screen. Visually, the opacity of games other than the selected game and its one-hop neighbors is lowered. Selecting a game also visually highlights the theme group(s) the game belongs to in the graph border – this fills in the lost information from using a different hue to encode multiple group membership. Clicking on the same game or the background removes the selection. Clicking on a different game changes the selection. Selecting a game does not filter Views 2 or 3, but does place marks on them to show what themes, complexity, and rating the game has (almost like **superimposing** a layer on those charts).

- Users can **hover over other games when a game is selected**. This behaves similarly to the hover interaction described above, giving the user a way to see games that are up to two hops away from the target node, or to explore a different part of the graph entirely without losing their current selection. The information about the hovered game is displayed next to the information about the selected game to make it easy to compare and support the task of finding similarities.

In addition, this view is controlled by the global filters and by Views 2 and 3, all of which can be used to filter the data in the graph. It is also controlled by the search bar, which allows the user to query nodes, supporting task 4. If a search returns a valid game in our dataset, then the corresponding node in the graph behaves as if it was selected (as described in interaction 2 above). Users can also use the random game selection button to select a random game, which does not support a specific task but is most definitely fun and supports our overall goal of making a visualization users can enjoy.

## View 2: Games in the Grand Theme of Things

This view primarily supports task 1, described above. By allowing users to choose a single theme or several themes to filter by, this view also supports task 3 above. This view is a fairly classic **bar chart** that shows aggregated data. Each **line mark** represents a categorical theme and there are two attributes encoded: categorical theme group (via **hue**, using the same hues as in View 1, and **vertical spatial region**) and count of games with that theme (via **length**). A bar chart with separated and aligned bars is a logical choice of idiom to support task 1, which requires a way to see an overview of the themes, to accurately compare the number of games across themes, and to locate specific theme bars easily. We chose to redundantly encode the thematic grouping of bars with spatial region and hue. Even though this makes it harder to compare magnitude across theme clusters, this choice connects the view with View 1, emphasizes similarities within groups, and makes it easier to find themes of interest since they are organized by group. On page load, this view shows the distribution for all games, sorted in descending order (by group and then internally to the group by theme).

There are two interactions for this view:

- Users can **hover over a theme bar** to see a tooltip. The bar gets a border to indicate it is hovered and a line also connects the bar and its label to make finding the right label easier.

- Users can **select one or more themes** by clicking on an individual bar or by clicking on the vertical bar next to a whole theme group. Visually, this decreases the opacity of other bars as a cue that the theme is selected. Clicking on the same theme or group removes the selection. Clicking on additional themes or groups adds to the selection (this is an OR operation). Clicking on the background removes all selections. Selecting a theme filters data in Views 1 and 3, allowing users to iteratively narrow down the games they are exploring to ones that match their needs. Filtering clears any existing selection from View 1.

In addition, this view is controlled by the global filters and by Views 1 and 3. The user can filter the data using the global filters. The order of the bars is not changed during filtering, which is a deliberate choice to allow users to compare (at least at an ordering level) the distribution of themes in different slices of the dataset (e.g. for party games vs all games). When a user selects a game in View 1, a small mark is placed next to the theme bars corresponding to that game's theme(s). When a user selects one or more hexbins in View 3, the opacity for the selected bars changes so that the fully opaque portion of the bar corresponds to the number of games with that theme **and** selected

rating/complexity. In this way, fully opaque portions of the bar always indicate games currently shown in the graph. (See the user flow for a visual of what this looks like.)

## View 3: Let's Take Into a Count How Tough It's Bin

This view primarily supports task 2, described above. By allowing users to choose a single bin or several bins to filter by, this view also supports task 3 above. This view is a **hexbin chart** that shows aggregated data. The hexbin bins data across two quantitative variables: complexity and rating. Each **interlocking area mark** represents a bin. There are three attributes encoded: complexity of games (via bin **vertical position**), rating of games (via bin **horizontal position**), and count of games (via bin **saturation**). We used saturation and a gray hue to avoid confusion with the hues used to encode thematic groups in Views 1 and 2. A hexbin is a good choice of idiom to support task 2, which requires a way to see the distributions for complexity and rating, but also to understand the relationship between these two. We used a hexbin instead of a scatterplot because a scatterplot would not scale well to 1000 marks. On page load, this view shows the distribution for all available games as well as a dynamic legend showing maximally and minimally saturated bins and their respective game counts.

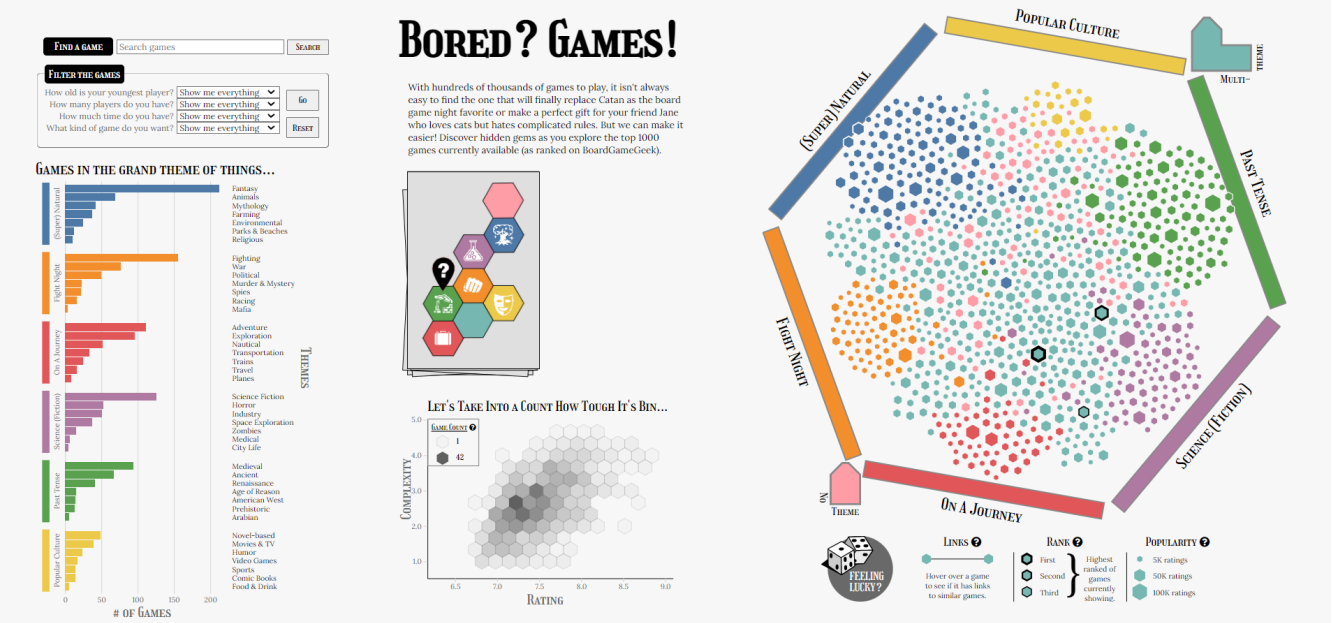
There are two interactions for this view:

- Users can **hover over a bin** to see a tooltip containing the range of rating and complexity for the games in a bin, as well as the most popular game in that bin. The hovered bin's border becomes moderately thicker.
- Users can **select one or more bins** by clicking on them. The selected bin's border becomes much thicker. Selection semantics are the same as in View 2, though selection in the hexbin filters only View 1 and affects View 2 as described above.

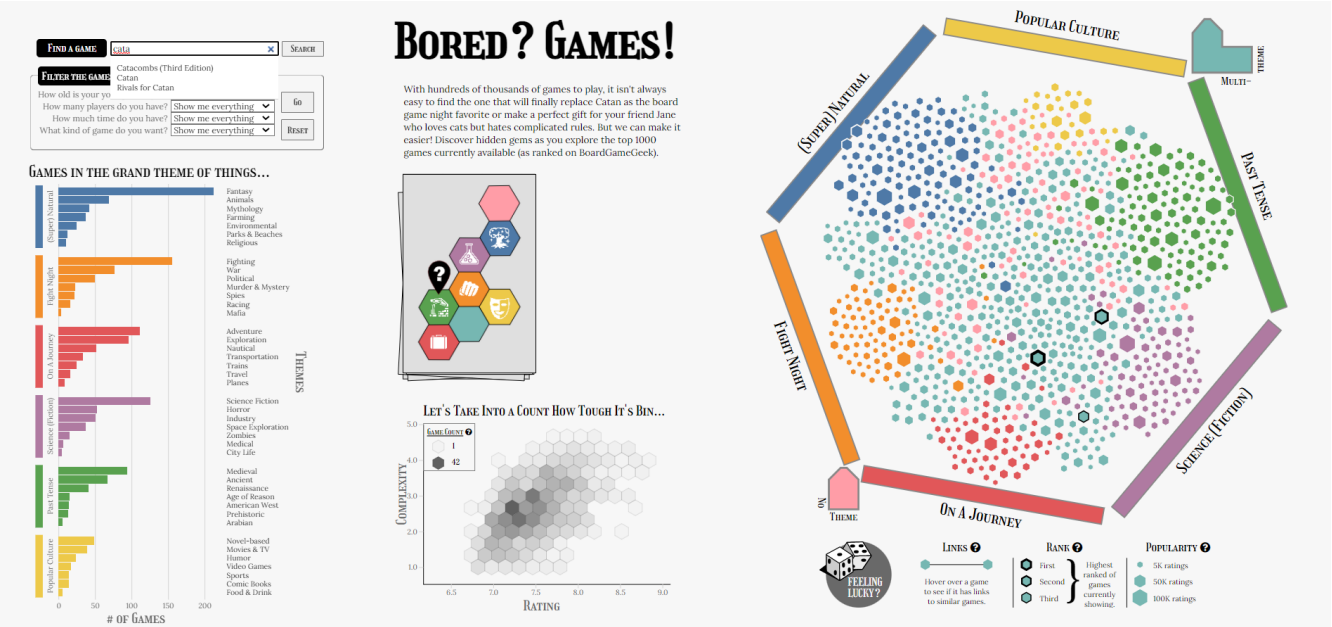
In addition, this view is controlled by the global filters and by Views 1 and 2. The user can filter the data using the global filters and/or View 2. When a user selects a game in View 1, a small mark is placed on the bin corresponding to that game's complexity and rating.

# Usage scenario

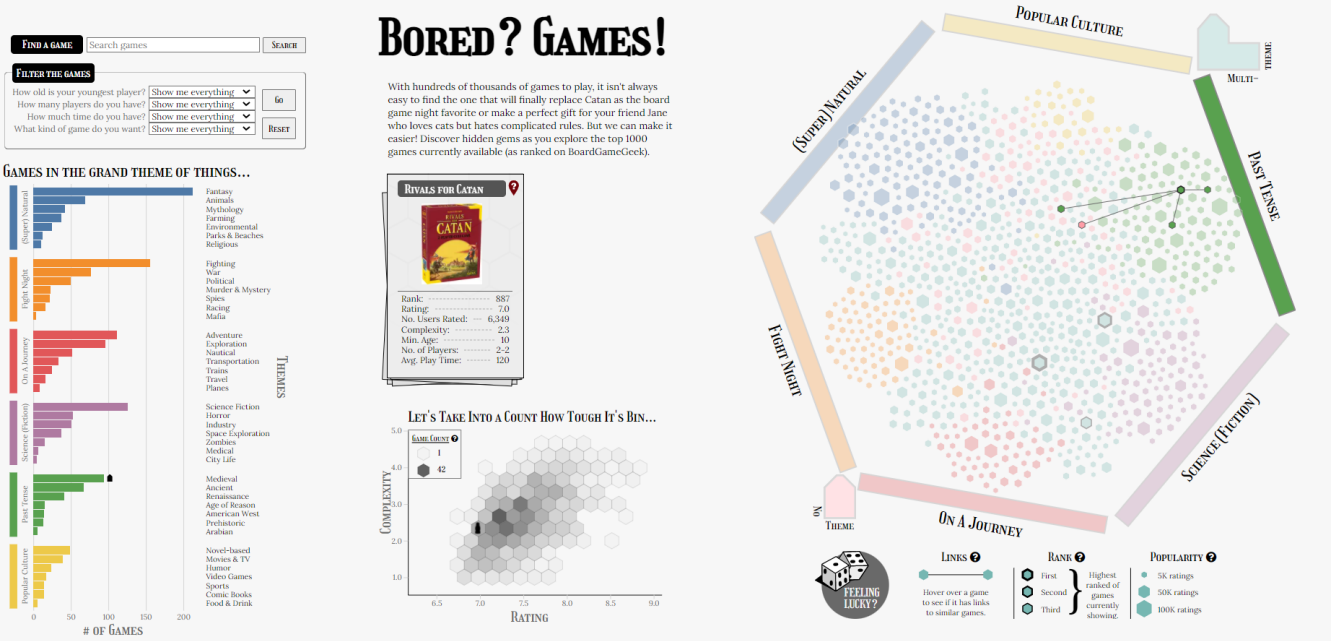
Bob is a board game enthusiast and loves spending his free time learning about and playing games. He wants an application that will allow him to **explore** and **discover** new board games. When he opens the application he is able to see an overview of the games that are currently loaded.



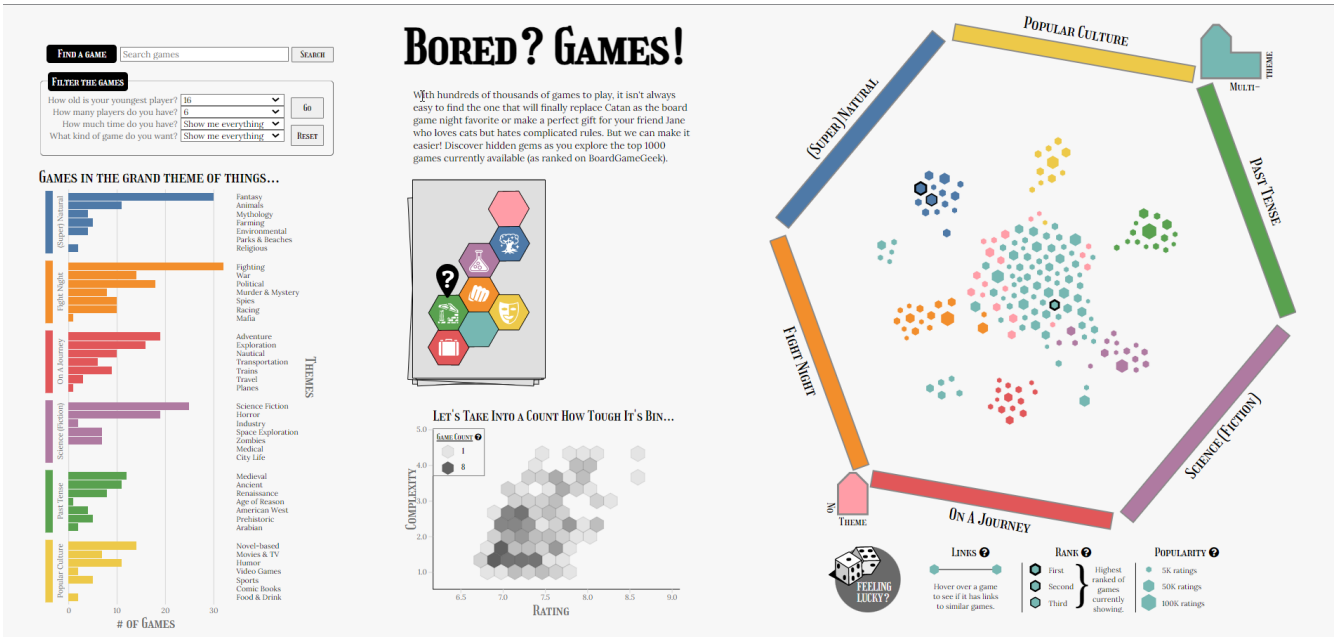

Since information about all the games are displayed at once (distribution of games and their themes via the bar chart and graph, and complexity vs rating in the hexbin), he decides he wants to filter the games so that he can get a sense of how the visualization works. He decides to use the search bar to **locate** “Catan”. While he’s typing, he is able to see games that contain his search word in it.



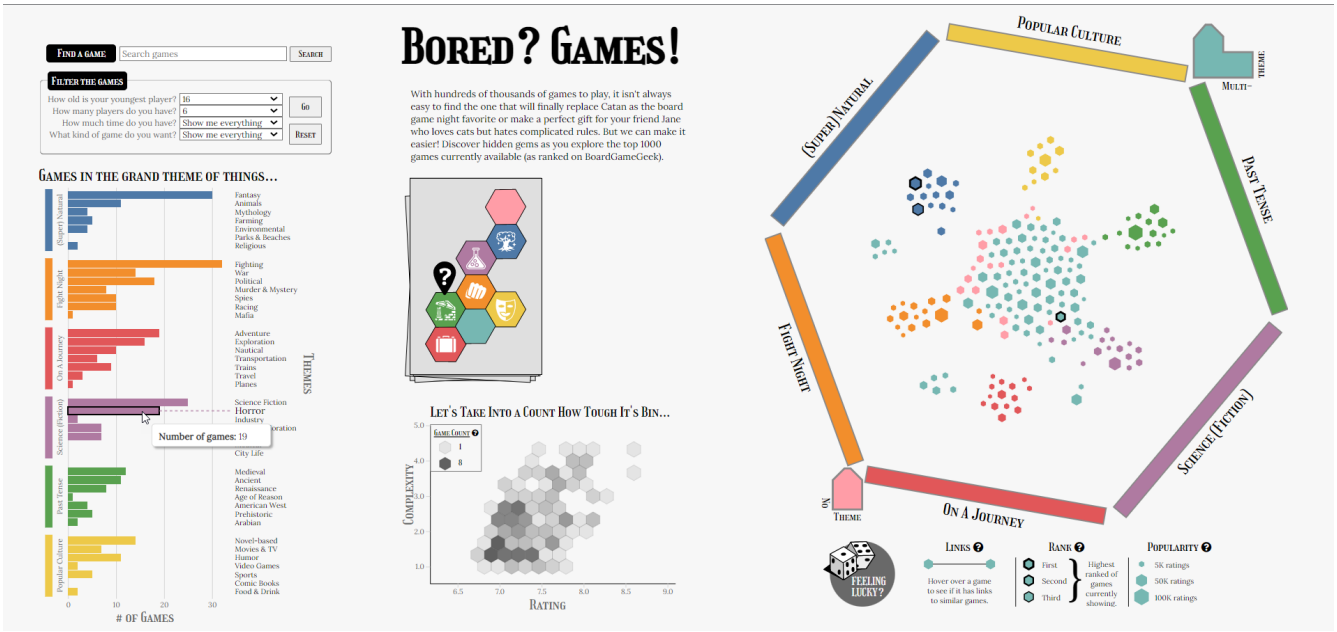
He didn't know that there was a game titled "Rivals of Catan", so he decides to complete his search with that game. The game is highlighted on the graph, along with its neighboring nodes that share some similarity with it. The border around the graph also indicates which theme the game belongs to. The barchart and hexbin also have new icons that appear that indicate which theme and bin the game belongs to. The deck of cards has a card flipped over that shows more information on the game.

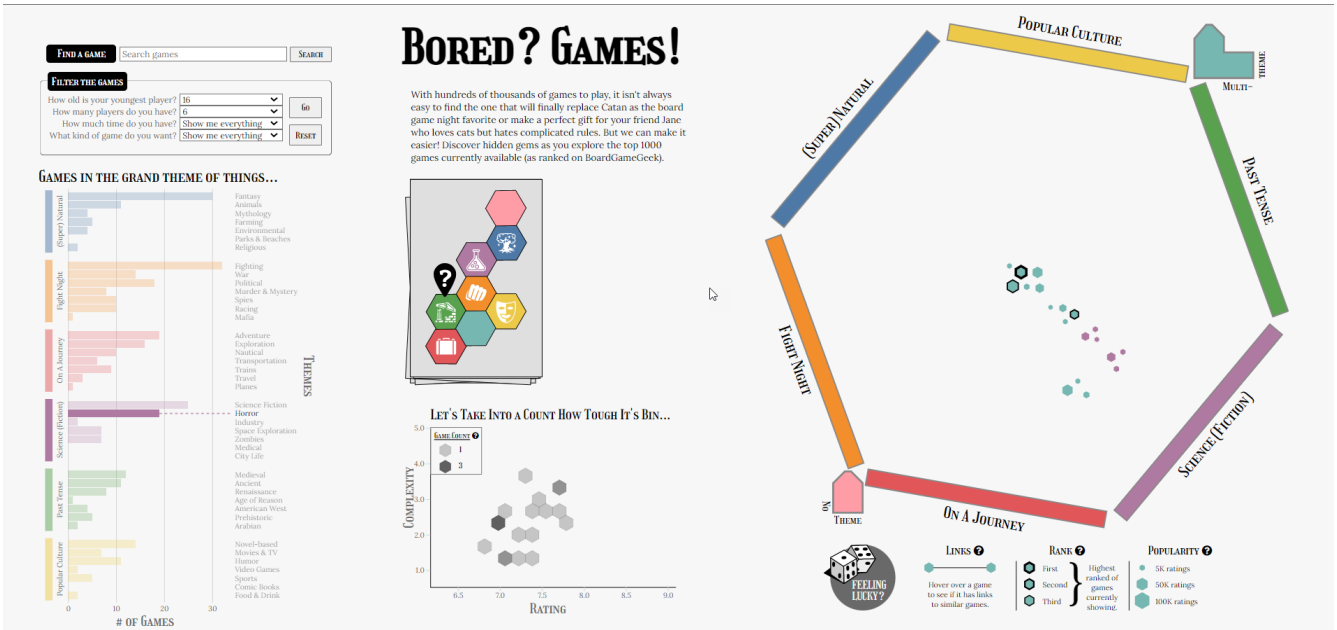



He is now ready to **explore** the visualization further. He decides to start by looking for a game that will suit the Halloween party he's attending next weekend. He updates the filters to only show games that are where the minimum age for the game is less than or equal to 16 years old, and can accommodate at least 6 players. This filters the games shown in the visualization.

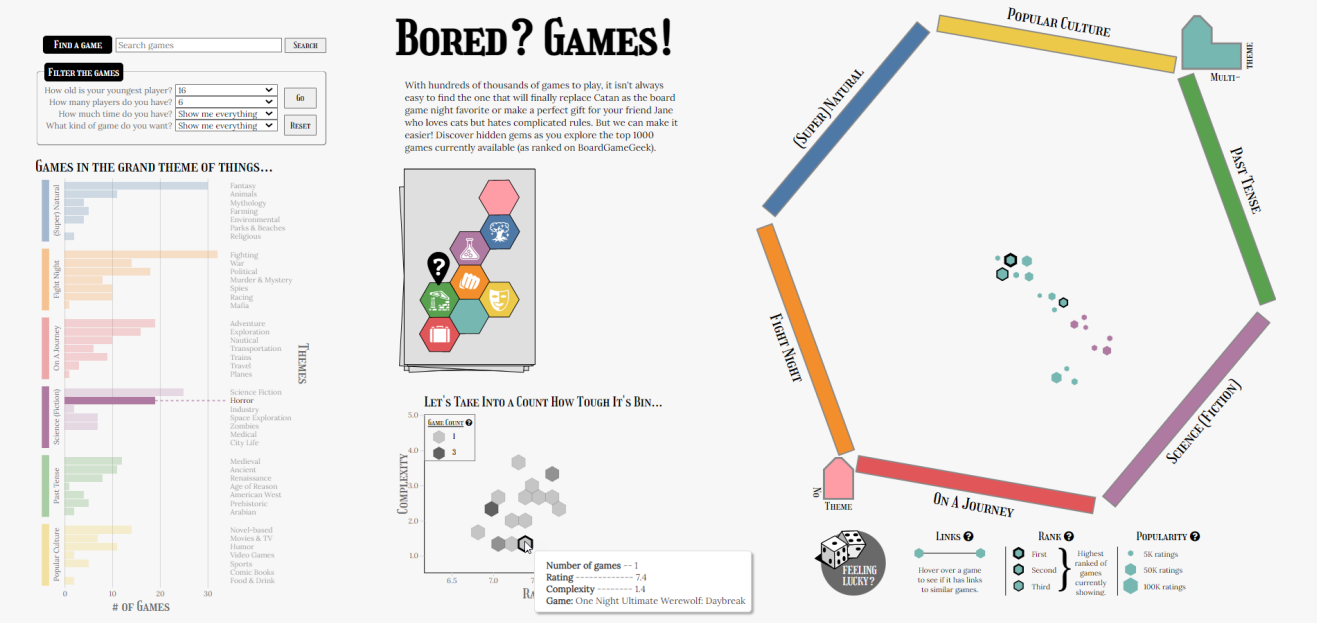


Since he is attending a Halloween party, he decides to further filter the games by clicking on the label “Horror” on the barchart. He notices that 19 games will appear in the graph, thanks to the tooltip that is shown when he’s hovering over the bar.

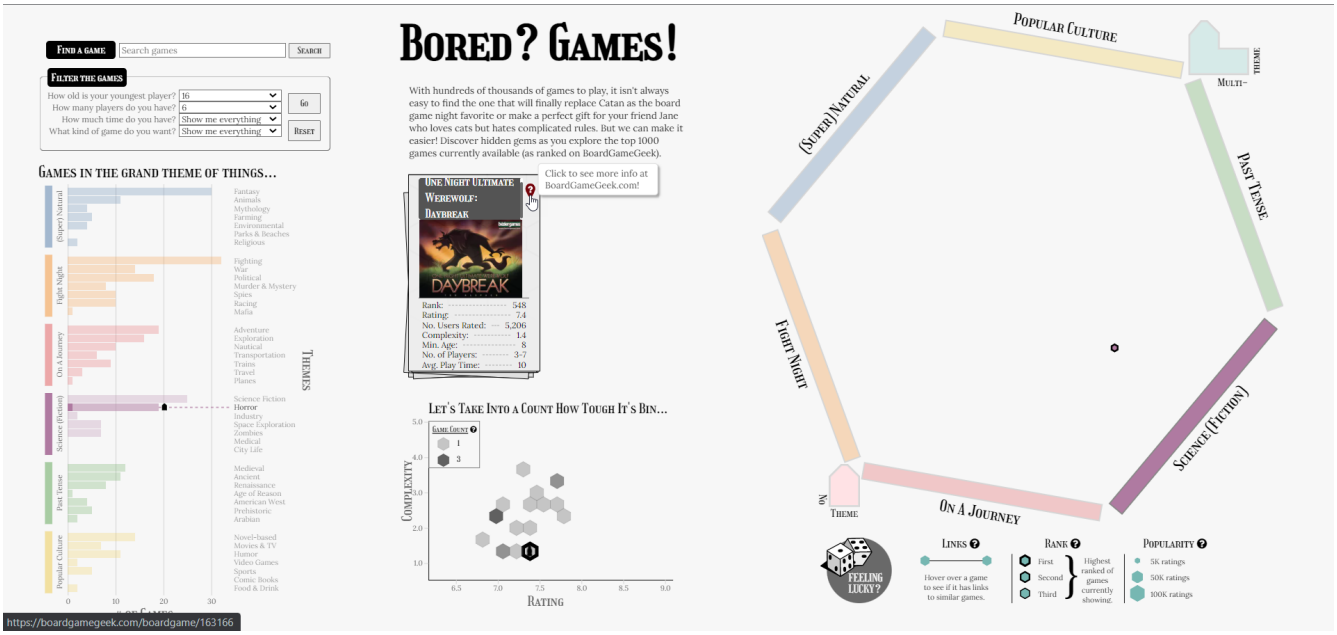


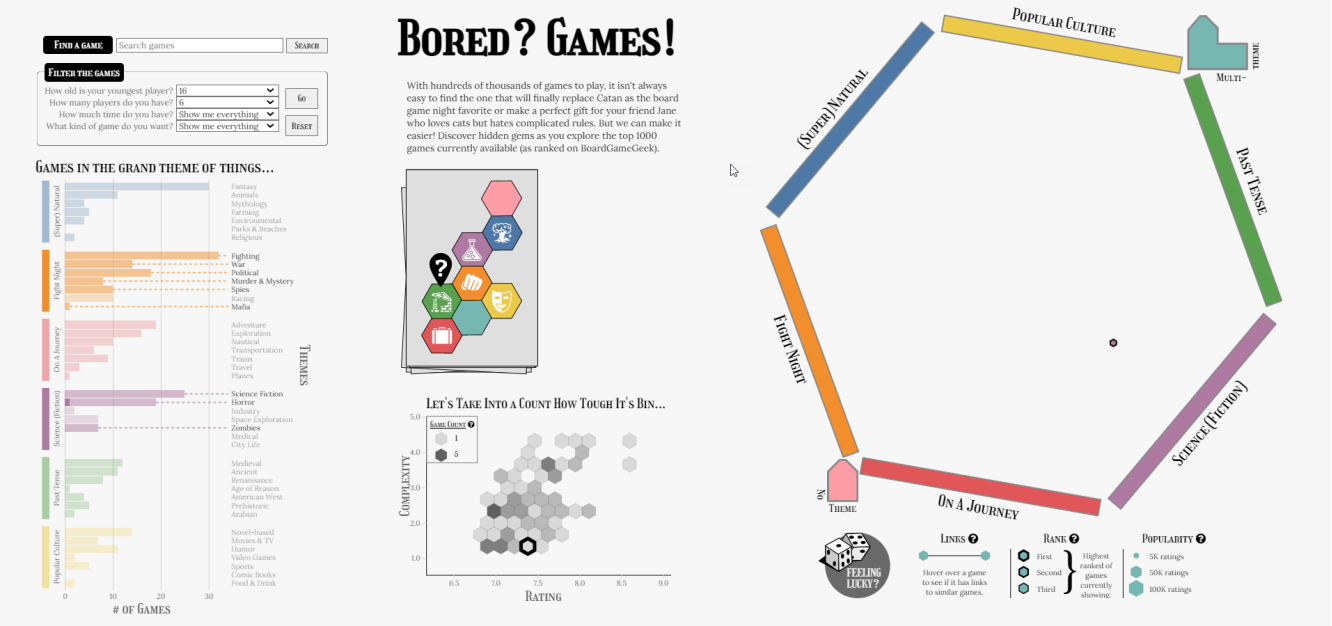

Since the event is a party, he wants a highly reviewed game that is not too complex: he uses the hexbin plot to **discover the distribution** of games for these attributes and clicks on a bin to further filter the games in the graph. He knows that the bin he’s selecting will only have one game, thanks to the tooltip.



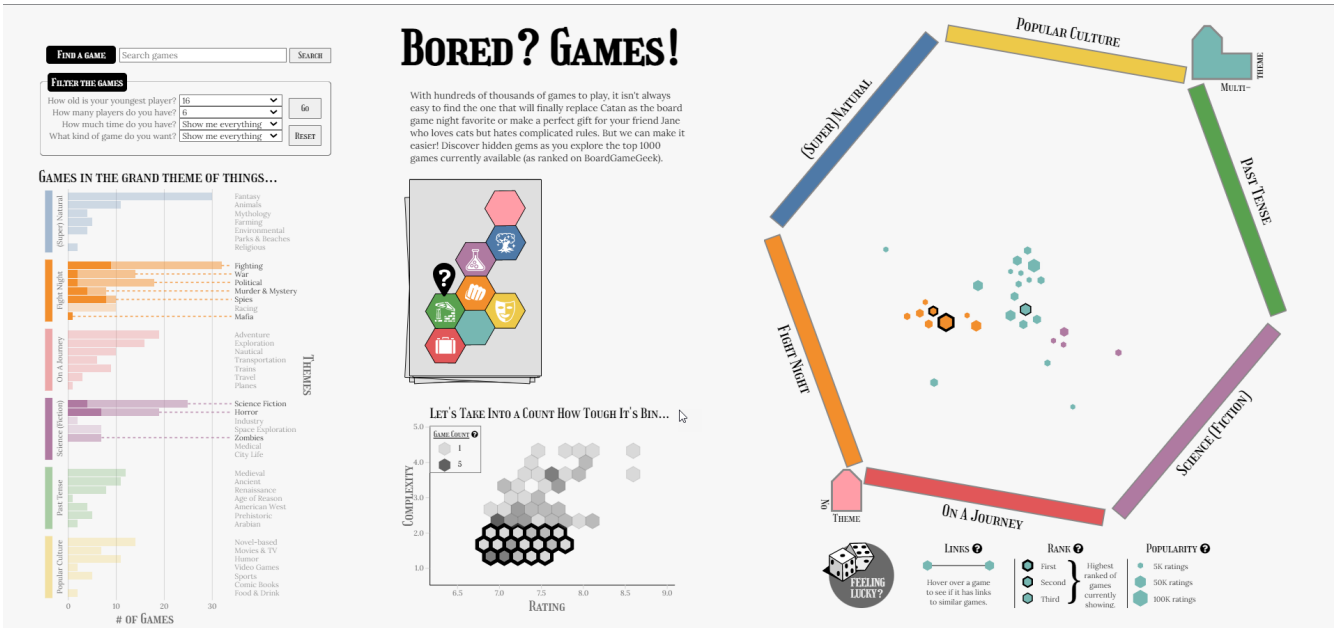

There is now only one game in the graph “One Night Ultimate Werewolf: Daybreak”. After selecting the game in the graph, he clicks on the information icon on the game’s card which takes him to the BoardGameGeek website. After reading the reviews he decides that maybe this is not the game for him.



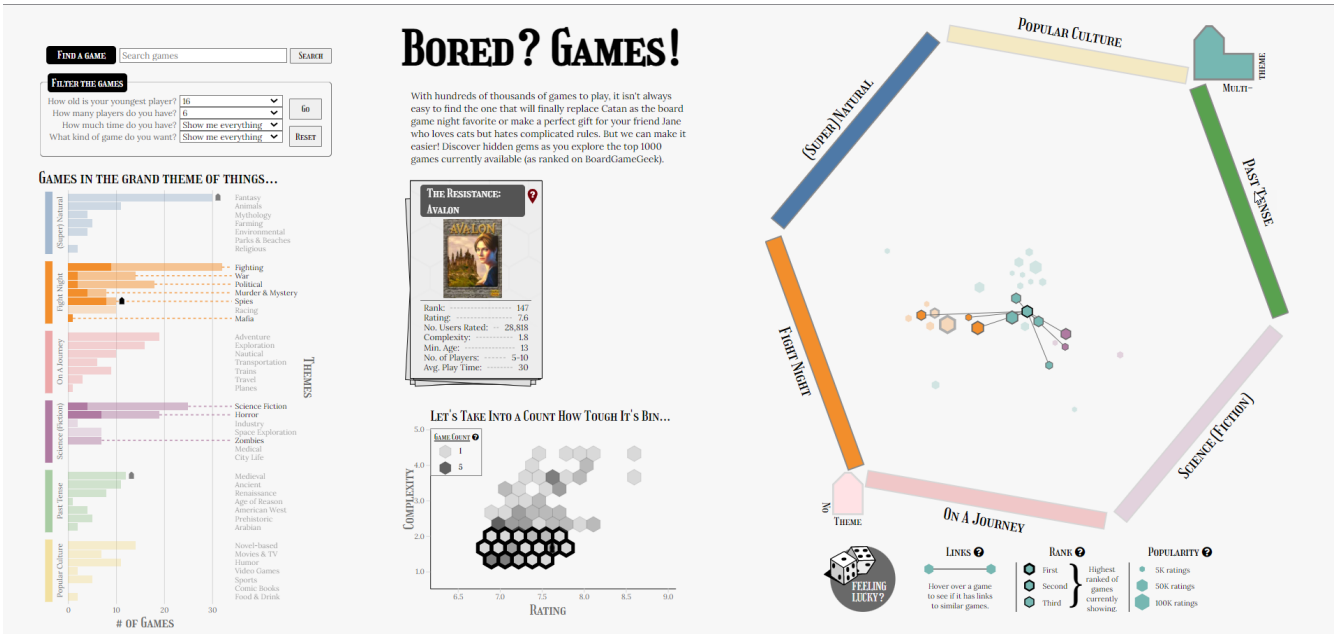
So, he uses the barchart to quickly **identify** which themes have a lot of games in them and decides to select the theme “Fight Night” group, then remove the “Racing” theme from that group. He also decides to add “Science Fiction” and “Zombies” to the selected “Science (Fiction)” themes. This added more bins to the hex chart, and he did not lose his initial selection.



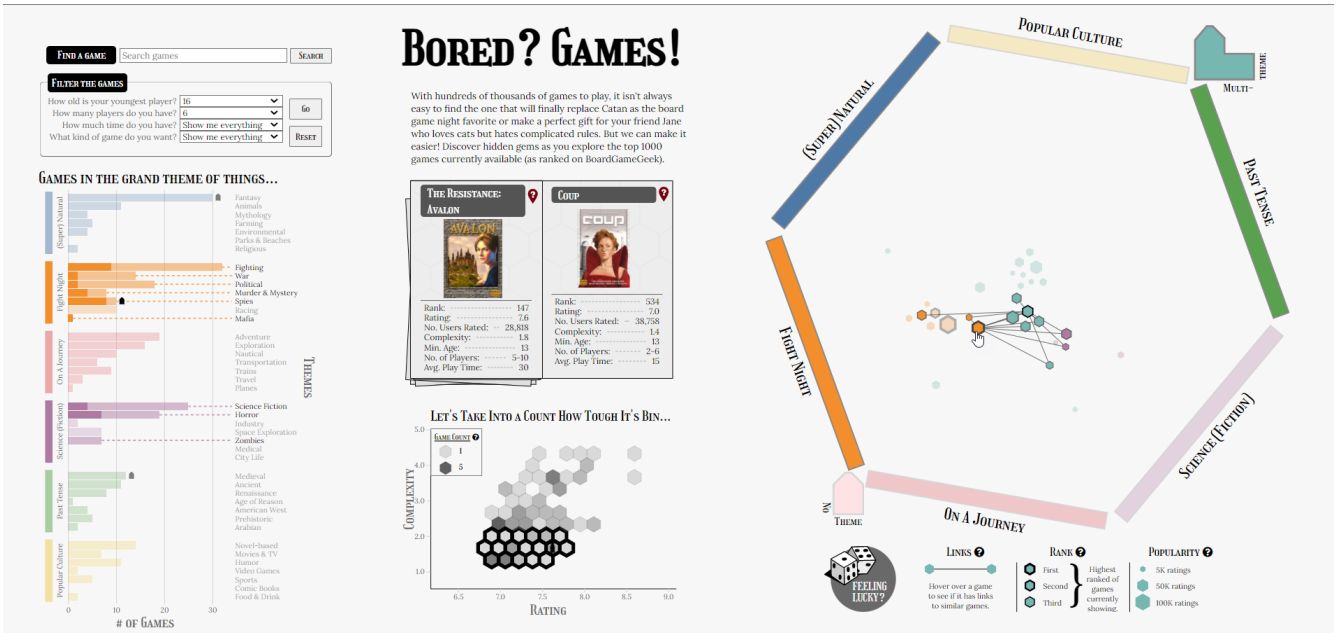

He selects some more bins which adds more games to the graph and he’s able to explore them by hovering over a game and seeing which games are similar to it. He is able to quickly see which 3 games are ranked the highest in his selected group of games because of the thickness of the border outlining the games in the graph.



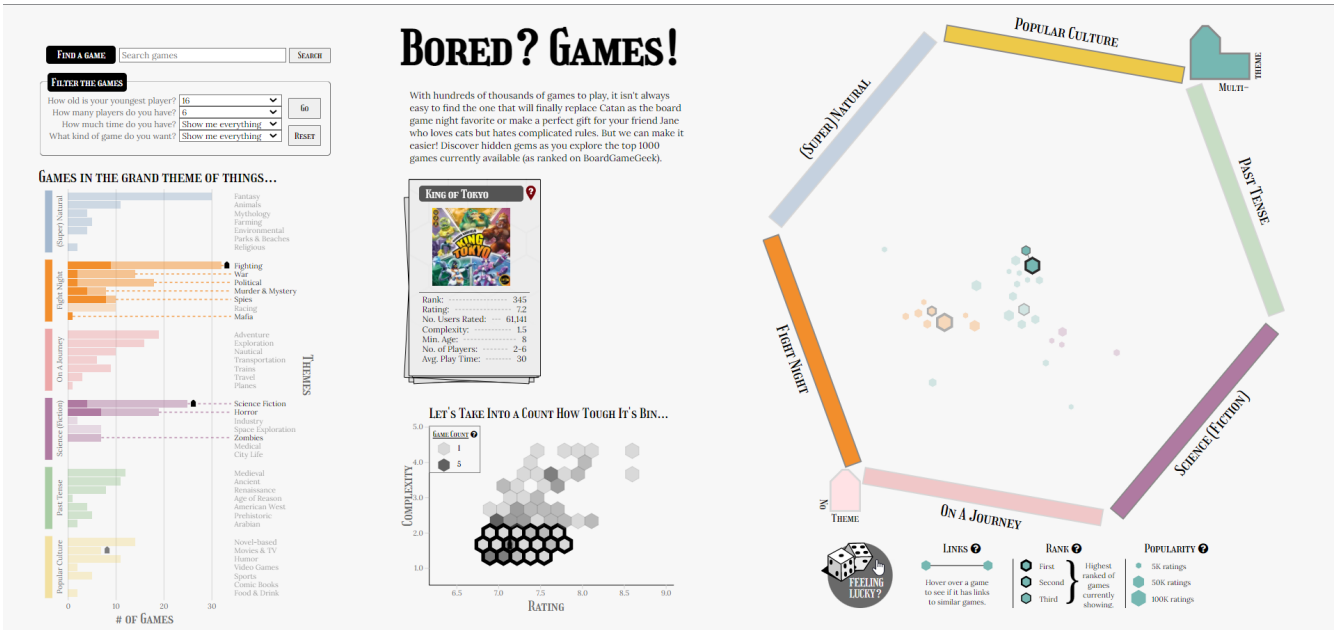
As he's exploring the nodes, he comes across the game "The Resistance: Avalon". He knows this game so he clicks on the game so that it can remain selected.




He is now able to explore the games similar to "The Resistance: Avalon" by hovering over its neighbours. He can **compare** attributes of the two games by comparing the information displayed in the games' information cards.



He doesn't know which game he wants to select because they all look interesting, so he decides to click on the 'Feeling Lucky' button. This randomly highlights a game from the subset of games he selected. He decides to choose that random game as the game he will buy and bring to the party.




He enjoys using this tool and decides he'll return when he's in search of a new game to play.

# 5. Credits

We would be remiss to not first acknowledge that the general structure of our classes and some elements (e.g. tooltips) are based on previous programming assignments and tutorials.

## Force Directed Graph

### Hexagon shapes

- We copied this hexagon path definition for nodes, used this guide on calculating geometries of a hexagon to position nodes and used this guide to scaling shapes with d3.

### Positioning of nodes within hexagon

- We copied the code that calculates if a point has intersected with a line from this example: d3 Force Layout Bound to N Sided Random Polygon and used the example to help understand where to place the code so that the nodes in our graph don't escape the hexagon border we created.
- We copied the formulas for calculating the points for a hexagon from Polygon vertex calculator so that we could input the center of the container and find out where the vertex of the hexagon would lie when drawing the border around the graph.
- We referenced changing link strength for the syntax to specify custom link strengths in the graph.
- We are using the 'Force in a box' Library to have nodes group into clusters: using forceInABox

### Interactions

- We looked at the answer from this stackoverflow question - highlighting an element when hovering over another and then got the idea to use an id to each node to help with highlighting nodes on hover and select.

## Barchart

- We adapted this grouped barchart example to figure out how to size the y-cluster axis scale within the larger y-axis scale.
- We used these two code blocks (scatterplot small multiples and line chart small multiples) to figure out how to make different axes for small multiples and how to share an axis.
- We used this block to figure out how to rotate the axis labels.

## Hexbin

- We used this d3 hexbin library to set up and display the hexbins.
- We adapted this hexbin example to help us format input data for the hexbins and set up for the rendering step.

## Deck

- We used this site to learn how to set up and use css to make a card flip, and this site to learn how to make cards slide in and out as a transition.

## Filters, search bar, dice

- We used this documentation and these examples as the basis for the code that animates the circle in the background of the dice. We adapted the code to allow us to flip through the 8 theme colors (rather than moving from one color through a spectrum to another).
- We used this code on the jQuery documentation page almost unchanged to populate the autocomplete for the search bar. The last demo was especially helpful.
- We used the css styling for the search bar dropdown essentially unchanged from the Zero Trust: Data Breaches past CPSC436V student project.
- We used the code in this StackOverflow post to detect when a user cleared the search bar via the 'X' (so we could clear the error text).
- We learned how to hide an errant node inserted by jQuery here.

## Layout, styling, colors, images

- We used the Tableau10 color scale available here.
- We used this svg path maker to create the paths for settlement and city shapes.
- We used this MDN documentation to learn how to use grid layout for the filters section.
- We used sketch.io to create the deck image, with icons from game-icons.net.
- Other images used: Catan robber, dice.

## Preprocessing

- This StackOverflow post gave us the idea to use the MultiLabelBinarizer to create vectors out of games so that we could calculate cosine similarity.
- Code from this post on the pandas Github page was used as-is to re-assign columns in a loop.

## General inspiration

- Horrified was our original inspiration as we started to explore the dataset.
- EdgeMaps: Philosophers inspired the idea of dynamic link highlighting.
- Actor Adaptability inspired the idea to place games by theme cluster.
- DelicaSee inspired the idea to derive a link attribute between games based on similarities.

# 6. Reflection

Overall, we were able to complete what we set out to do in our initial proposal. The three views we proposed are present in essentially their initial form, as are our proposed interactions and widgets. As we implemented views and tested out interactions between them, we did make three larger changes, all of which we believe only strengthen the final project:

- We chose to display all the links from a node, instead of just the top 5 links as originally planned. We thought that it would help users to see all the games that are connected to a node, especially when querying nodes for a specific game. When testing this change, we did not think there was an issue of occlusion for most games, although the graph can get busy in some cases.

- We changed the nature of the bidirectional linking between the barchart and hexbin following our M2 submission. Immediately prior to M2, we discovered a bug in the filtering interaction between these two views. When we spent more time using the bidirectional filters, we decided that allowing the hexbin to also filter to barchart was confusing. It made it difficult to understand what set of filters was in play at any one time or to move back and forth iteratively between the two views. Instead, we turned the link from hexbin to barchart into a highlighting one.

- We chose to make it possible for users to continue to hover over all nodes in the graph when a node is selected. Originally, we planned to allow hovering only over one-hop neighbors of the selected node but we found that allowing all nodes to remain hoverable gave users more flexibility in how to explore the graph. For example, users can now see two-hop neighbors. This required us to be able to show two tooltips for the graph (one “selected” card and one “hovered” card), which does mean we have unused pixels when a user is not interacting with the graph in this way. However, overall we feel the benefits outweigh the costs.

Our visualization and technical goals stayed consistent throughout the project. Our initial usage scenario and tasks guided our project throughout. We wanted to push ourselves to explore different aspects of D3, which we did: each of our components either uses a new-to-us D3 library (specifically the forceInABox and d3-hexbin libraries) or pushes beyond the standard plots covered in tutorials and programming assignments (the barchart is grouped in small multiples).

We were able to stay consistent because our original goals were realistic in terms of what is technically possible in D3. Before brainstorming for our project, we had spent time looking at examples on Observable and we felt confident that each view was feasible. With that said, although our goals were realistic, they were also loftier than we initially thought they would be! Using the libraries for the force directed graph and hexbin ended up being more complicated than we expected. For the graph, there was a lot of experimenting to figure out what forces to apply in what combination. There was also a lot of math involved to calculate the layout of the hexagonal border, constrain the placement of the nodes within that border, and set the centroid coordinates for each theme cluster group. Certainly it would have been easier to just use a standard rectangular graph shape, but we enjoyed the challenge and thought the hexagons fit well with the theme. For the hexbin, the main obstacle was that the library did not automatically tell us what the boundaries of a bin were, so we had to figure out workarounds to enable some interactions.

There was nothing we wanted to implement that we could not figure out how to do. We decided against pursuing some stretch goals because we were running out of time and screen pixels. Ultimately, we ran out of time to make the visualization responsive. This would have required figuring out the calculations needed to have the graph be able to scale with the size of the container it was in. In the interest of time, we decided to fix the size of the container that the graph was in. This way we could define the specific coordinates of the border and city/settlement icons around the force graph.

If we were to make the project again, we would try to enhance the graph by visually encoding the similarity weight into the links, which could make it more clear why certain nodes are found in certain positions on the graph. We also wished we could give users more information about what kinds of similarities there were between the games (not just the presence or absence of a similarity). Ultimately, however, we are proud of what we accomplished.

# 7. Project Management & Team Assessment

Our team worked well together to complete what we set out to do. For code development and integration, our process was to individually develop features on branches, then create pull requests that had to be approved by one other team member before merging to master. For project management, we met weekly to discuss progress and needed changes. Additionally, we communicated via a group chat as needed to ask questions and/or discuss issues. We essentially followed our initial plan, which was minimally revised in our M2 report to include some new tasks (notably testing and integration). No new tasks were added since the M2 report.

| **Team member** | **Nicole** | **Michelle** | **Andrew** |
|---|---|---|---|
| **Main tasks** | Graph, search bar, write-ups, testing | Data processing, barchart, global filters, search bar, dice, legend, write-ups, testing | Hexbin, deck, tooltips, layout & styling, write-ups, testing |
| **Total hours** | 60 | 61 | 59 |

| | Task | Task Owner(s) | Completion Date | | # Hours | | Note |
|---|---|---|---|---|---|---|---|
| | | | Target | Actual | Target | Actual | |
| Write-Ups | | | | | 30 | 23.5 | |
| M2: WIP | Complete write-up | A, M, N | Mar 24 | Mar 29 | 6 | 6 | |
| M3: Final Write-Up | Discussions | A, M, N | Apr 1 | Apr 5/11 | 6 | 6 | |
| | Draft | A, M, N | Apr 6 | Apr 10 | 12 | 5.5 | able to use M2 rpt |
| | Finalizing | A, M, N | Apr 8 | Apr 12 | 6 | 6 | |
| Data processing | | | | | 5 | 5 | |
| Processing & clean-up | Processing, clean-up | M | Mar 14 | Mar 14 | 2 | 2 | |
| | Similarity calculations | M | Mar 18 | Mar 15 | 3 | 3 | |
| Coding | | | | | 112 | 144 | |
| Static | Overall layout, styling | A | Mar 16 | Mar 15 | 6 | 15 | changes post Mar 15 |
| | Theme border (graph) | N | Mar 16 | Mar 20 | 2 | 5 | |
| | Hexbin chart | A | Mar 18 | Mar 28 | 4 | 12 | |
| | Barchart | M | Mar 18 | Mar 21 | 4 | 8 | |
| | Legend | M | Mar 18 | Mar 25 | 4 | 4 | |
| | Force directed graph | N | Mar 20 | April 8 | 20 | 35 | |
| Interaction / Hover | Graph to border | N | Mar 24 | Mar 25 | 4 | 6 | |
| | Graph to barchart | N | Mar 24 | Mar 28 | 4 | 1 | |
| | Graph to hexbin chart | N | Mar 24 | Mar 28 | 4 | 1 | |
| | Barchart external | M | Mar 24 | Apr 6 | 12 | 8 | changes in M3 |
| | Barchart internal | M | - | Mar 24 | 0 | 6 | |
| | Hexbin to graph | A | Mar 24 | Mar 28 | 8 | 1 | |
| | Hexbin to barchart | A | Mar 24 | Mar 28 | 4 | 1 | |
| | Deck & tooltips | A | Apr 1 | Mar 27 | 8 | 14 | |
| UI Widgets | Filters | M | Mar 20 | Mar 23 | 8 | 8 | |
| | Search bar | N / M | Apr 1 | Apr 6 | 8 | 3 | |
| Testing | Manual testing | A, M, N | Apr 3 | Apr 10 | 8 | 6 | |
| | Bug fixes | A, M, N | Apr 3 | Apr 11 | 4 | 10 | |
| | | | | **Total hours:** | **147** | **172.5** | |
| Stretch Goals | | | | | 30++ | 7.5 | |
| Functionality | Random discovery | M | Apr 6 | Apr 6 | 2 | 2 | roll dice |
| | Targeted discovery | any | Apr 6 | - | 8 | - | chose to not do |
| | Additional filters | any | Apr 6 | - | 4 | - | chose to not do |
| | Scented widget filters | any | Apr 6 | - | 8 | - | chose to not do |
| | Annotations | any | Apr 6 | - | 8 | - | chose to not do |
| | Responsive layout | A | Apr 6 | - | 4 | 5 | dropped due to time |
| Aesthetics | Additional styling | any | Apr 6 | Apr 12 | rabbit hole | 0.5 | |