Milestone 4: Final Project Submission

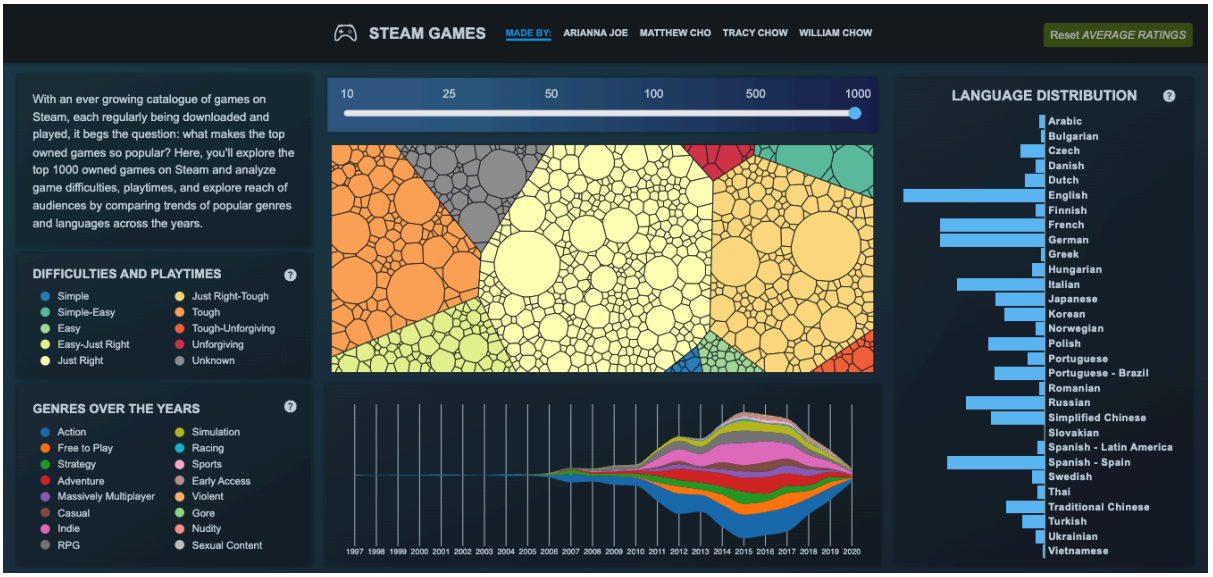
Basic Info

Project title: Analyzing Steam Games
Team members:

- Matthew Cho, matthewymcho@gmail.com
- Tracy Chow, tracyc2780@gmail.com
- Arianna Joe, joearianna828@gmail.com
- William Chow, williamchow604@gmail.com

Video Demo: *https://youtu.be/5HnCil6h2pY*

Overview



With an ever growing catalogue of games on Steam, it can be hard to compare games against each other, discover what kind of features they have in common and figure out why some games are owned by so many people. To help users delve into exploring Steam game statistics, we propose to create a visualization where users will be able to explore top owned games while analyzing game difficulties and playtimes. Users will also be able to explore the reach of audiences by comparing trends of popular genres and languages across the years.

Data

Our project uses this dataset (https://github.com/leinstay/steamdb) of steam game metadata found on https://www.data-is-plural.com/. The dataset includes 53981 distinct items and 46 attributes. Below are our steps for preprocessing of the data, and the narrowed down list of 26 attributes we plan to keep. Attributes are encoded visually, and a subset used for additional filtering in our views.

Preprocessing of the data:

- Filter out the attributes that is not used in our visualizations
- Normalize gfq_rating to be consistent with the other rating systems
- Parse categorical attributes platforms, developers, publishers, languages, voiceovers, categories, genres, tags
    - Break string up by commas, parse into array of strings
    - E.g. tags: "Puzzle,Co-op,First-Person,Sci-fi,Comedy,Singleplayer" into tags: ["Puzzle", "Co-op", "First-Person", "Sci-fi", "Comedy", "Singleplayer"]
- Parse dates into JavaScript Date
- Add attribute 'avg-rating' calculated from 'store_uscore', 'gfq_rating', 'meta_score', 'meta_uscore', 'igdb_score', and 'igdb_uscore'

Preprocess implementation: Initial filtering of unneeded attributes and of top 1000 owned games as defined by the 'stsp_owners' attribute is done using a Python script, addition of new attribute 'avg-rating' is done using a Python script, and the preprocessed dataset uploaded to the GitHub repository. Additional preprocessing is done on the fly in JavaScript.

Final Chosen Attributes Table

Attribute	Type (categorical, ordinal, quantitative)	cardinality/range	Description
sid	categorical	53981	Unique identifier
store_uscore	quantitative	1 - 100	Steam store user score
published_store	ordinal	3764	Date published
name	categorical	53708	Game name
description	categorical	53792	Game desc
platforms	categorical	7	platforms
developers	categorical	33905	developers
publishers	categorical	29355	publishers
languages	categorical	7532	List of languages
voiceovers	categorical	1215	List of voiceovers
categories	categorical	4663	List of categories
genres	categorical	1795	List of genres
tags	categorical	15698	List of tags
gfq_difficulty	ordinal	10	Average Difficulty rating by users

gfq_rating	quantitative	0.5 - 5	Average rating by users
gfq_length	quantitative	0.5 - 80	Average time to beat by users
stsp_owners	quantitative	10 000 - 150 000 000	# of owners according to SteamSpy
stsp_mdntime	quantitative	1 - 137570	Median playtime
hltb_single	quantitative	1 - 5149	Hours to beat Main Story
hltb_complete	quantitative	1 - 4871	Hours to Completionist
meta_score	quantitative	19 - 96	Aggregated critic ratings
meta_uscore	quantitative	5 - 100	Aggregated user ratings
igdb_single	quantitative	1 - 133	Hours to beat “normally”
igdb_complete	quantitative	1 - 400	Hours to beat “completely”
igdb_score	quantitative	10 - 100	Aggregated critic ratings
igdb_uscore	quantitative	11 - 100	Aggregated user ratings
avg_rating	quantitative	30 - 95	Average of ratings metrics

Tasks

Our project supports the tasks below, listed in both domain and abstract languages.

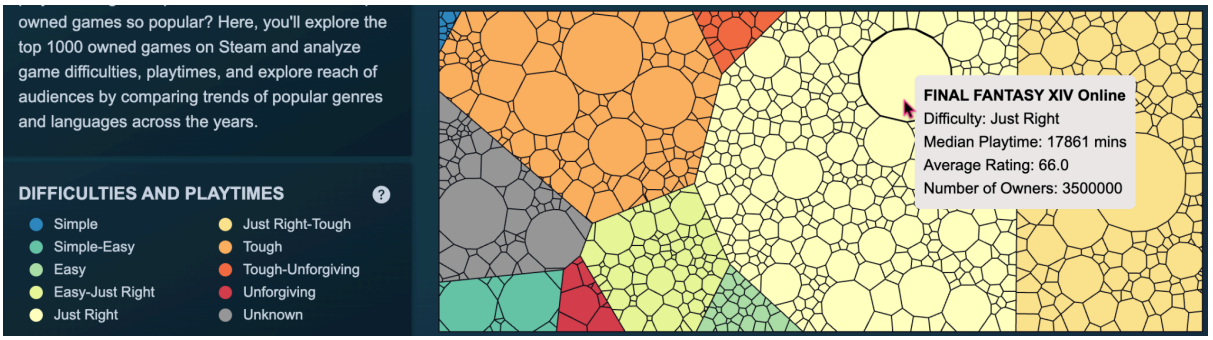
#	Domain	Abstract	Attributes
1	Alice likes playing simpler games as a way to relax. However, she usually doesn’t have a lot of free time and wishes to see how simple, high rated games compare to each other in terms of playtime	- Query items - Compare trends	- stsp_mdntime - gfq_difficulty - Store_uscore - meta_score - meta_uscore - igdb_score - igdb_uscore - gfq_rating
2	Bob is an aspiring game developer who’s interested in Action games. He wants to see how the popularity of Action games have changed over time and contrast it to how other genres have done. That way he can determine if people will likely be interested in a new Action game.	- Identify trends over time - Compare trends	- genre - published_store

3	Jane is an avid gamer stuck at home during a pandemic and can speak 4 languages. She's curious about games that are available in multiple languages, and wants to see what the most popular languages are for games released in the year 2020.	- Query items - Discover distribution	- languages - stsp_owners

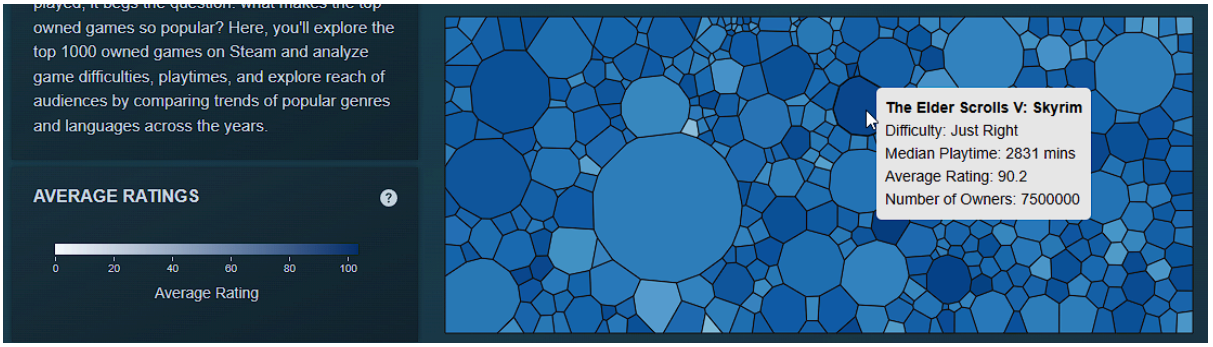
Visualizations

Our data visualization consists of 3 views: a bar chart, a streamgraph and a Voronoi Treemap. We implemented a dashboard style visualization for our project.

Voronoi Treemap (innovative view)

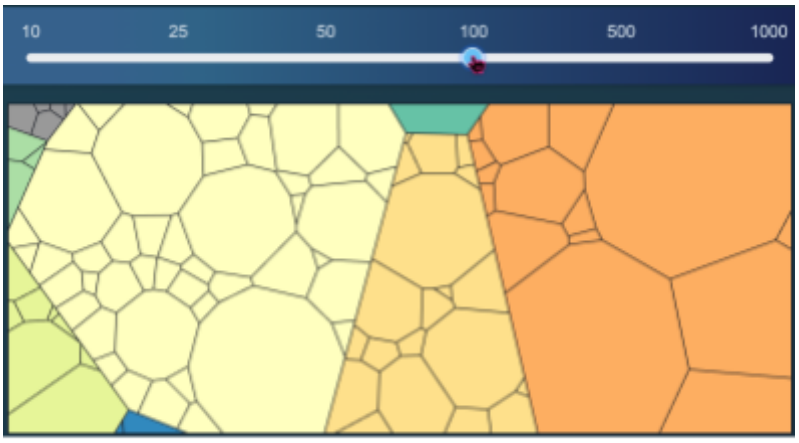


The first view in our visualization is a Voronoi Treemap, consisting of multiple layers to allow for more encoding of information and comparison. The space-filling property of this graph, in particular, allows for a more visually interesting view when the number of marks is low. Layer 1 showcases the different levels of difficulty the games have, with level 1 polygonal marks for an individual game and level 2 polygonal marks for difficulty. As the marks on both levels are space-filling, as mentioned, the graph has neither cartesian nor polar coordinates, all borders are shared, and order does not mean anything. Therefore, position is not encoded for both levels. This is appropriate for our graph, as positioning and order are not important factors that contribute to the comparison between difficulty and median playtime. Rather, the shared borders allow for 2D size encoding, making the comparisons more meaningful. As for other encodings, level 2 is colour encoded by difficulty using a diverging colour scheme to differentiate the different levels of difficulty, and 2D size encoded by total median playtime to contrast this attribute between the different marks in terms of space occupancy. Level 1 does not have any colour encoding (as it is part of level 2's colour encoding) but has its 2D size encoded for median playtime to help with the contrast of the total median playtime in level 2. This allows users to quickly see a broad overview of the median playtime across different difficulties and compare the difficulties against each other by using median playtime as a proxy for player retention.



Users can click on a level 2 mark (depicting difficulty) in the Voronoi Treemap layer 1 to enter the layer 2 view, where each level 1 poly mark represents an individual game. Here, like the level 1 marks of layer 1, the 2D size encodes median playtime to contrast the attribute among the games, and colour saturation now encodes the game's rating to show the quantitative attribute's gradual differences visually. This allows users to discover games with a specific difficulty and analyze how rating and median playtime affect each other. For both layers, the level 1 poly marks all have mouse interaction. Specifically, a tooltip is displayed when hovering over a mark and lets users see the corresponding game name, difficulty, median playtime, average rating, and the number of owners.

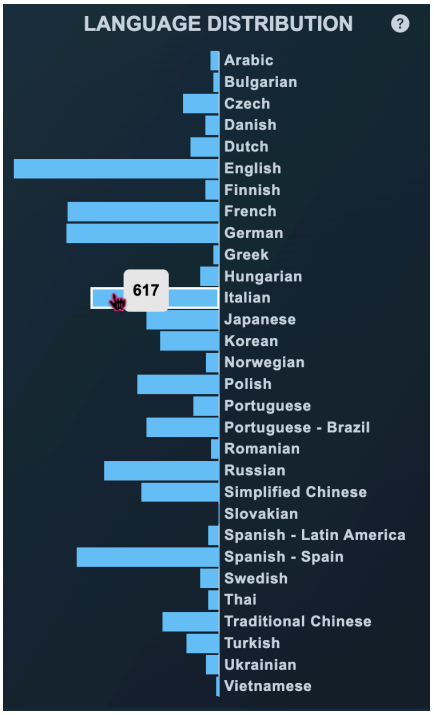
We chose to use a Voronoi Treemap as it matches our tasks well and visualizes our needs better than other types of graphs. For instance, we could have used a circle-pack chart and incorporated layers by allowing users to click into bubbles. However, separated circle symbols for each game mark would likely make it harder to compare against each other for median playtime. The Voronoi Treemap allowed for easier comparisons of game marks and levels of difficulty beside each other, especially when finding the largest and smallest marks, as they share edges. This makes the comparison between space occupancy (and, therefore, the encoded attribute) easier between marks. Voronoi Treemaps also allow us to use the space on the dashboard more efficiently, as we don't want to compress the other charts too much in order to fit everything on the screen. Permitting more room for more insightful details.



Voronoi Treemap Slider Widget

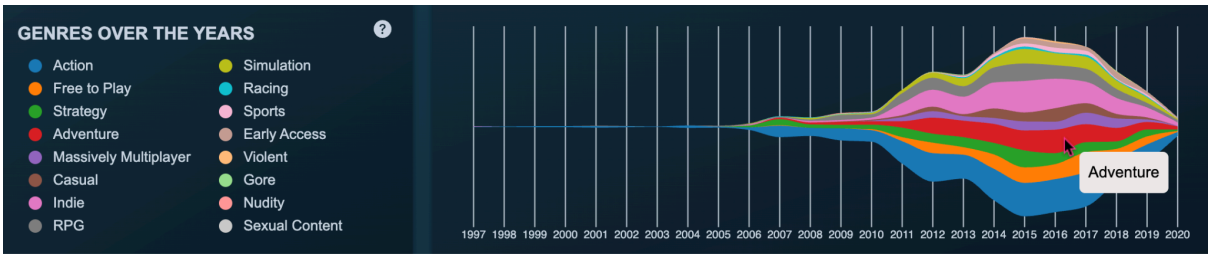
There is also a slider widget for this view that filters the data based on the top **X** owned games (defined by utilising the number of owners attribute). For instance, if a user wants to view the top 100 owned games, they can use the slider to slide to 100. The default is 1000 games. This allows users to compare trends of a specific set of top-owned games.

Bar chart



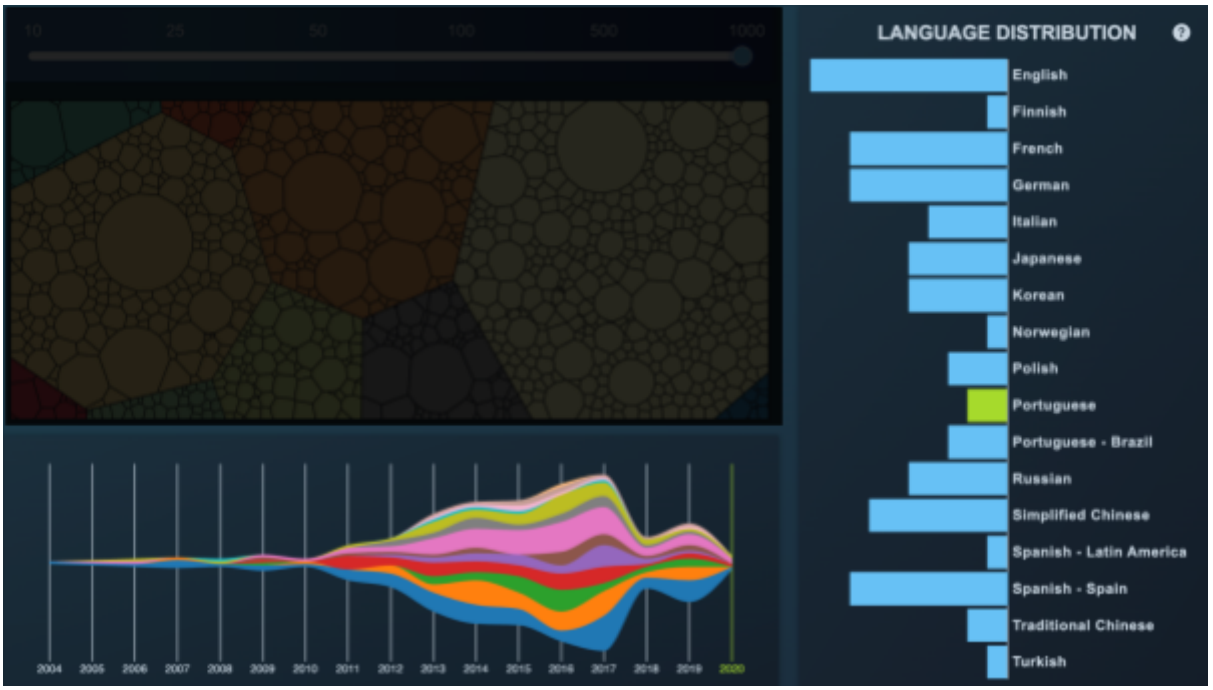
The bar chart displays the categorical attribute languages in the y-axis (vertical position), with each horizontal bar corresponding to a unique language. The x-axis (horizontal position) is the count of games that have the corresponding language available. The bar chart allows the user to easily discover the distribution of languages across the database and compare popular languages against each other. Each bar is easily comparable as they're aligned on a common scale along the y-axis--something not as visually obvious in other charts such as a pie chart or bubble chart with their unalignment. Users curious about the exact count of games for each language can hover over each bar to reveal a tooltip that displays the exact count.

Streamgraph



Our final view is a streamgraph depicting the count of genres per year. The x-axis (horizontal position) contains the year of publication of each game, while the y-axis (vertical position) is the count of games per genre. These position encodings allow the user to see how the proportions of genres change per year throughout the years to identify the most popular genres for each year, which is depicted by the largest area. Each genre is categorically colour-encoded for better visibility and separation and when hovering over each colour (area), a tooltip is then rendered containing the corresponding genre's name. A streamgraph allows us to encode all the genres without overwhelming the user, as popular genres are a lot more visible. It also allows users to more easily discover, view and compare the trends of genres over the years. Other graphs, such as a stacked area chart, are possible but would emphasize specific counts for each genre more, whereas a streamgraph is more fluid and emphasizes the relative differences between genres for each year.

Bidirectional linking between Streamgraph and Bar chart



We have a bidirectional link between the streamgraph and the bar chart. Users are able to click on the bars in the bar chart (which corresponds to a language), causing the streamgraph to update so it only considers games with the selected language. The stream graph also dynamically scales so that the data for the selected language stays visible to the user no matter the range. The default no selection corresponds to all languages being chosen. This allows users to discover and compare genre trends among specific languages.

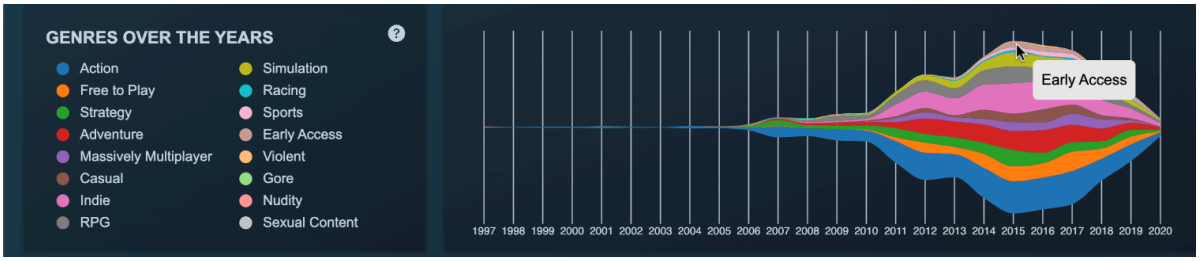
As for the other side of the bidirectional link, users are able to click on a year in the streamgraph, causing the bar chart to update to only consider games released in that chosen year. Filtering the bar chart by release year lets the user compare available and popular languages for specific years instead of only being able to compare the overall, as was mentioned in our third task from the Tasks section. The year selection has been constrained to have either no years selected (default, all years considered) or a single year selected. This constraint was decided due to the feasibility of implementation with limited time.

For both sides - filtering on languages and filtering on a year - the currently selected language/year is highlighted in green to help indicate and remind the user which language/year they are currently filtering the other chart in. Users are also able to clear the filter on each chart by clicking on the background of the respective chart or on the selected element.

Usage Scenarios

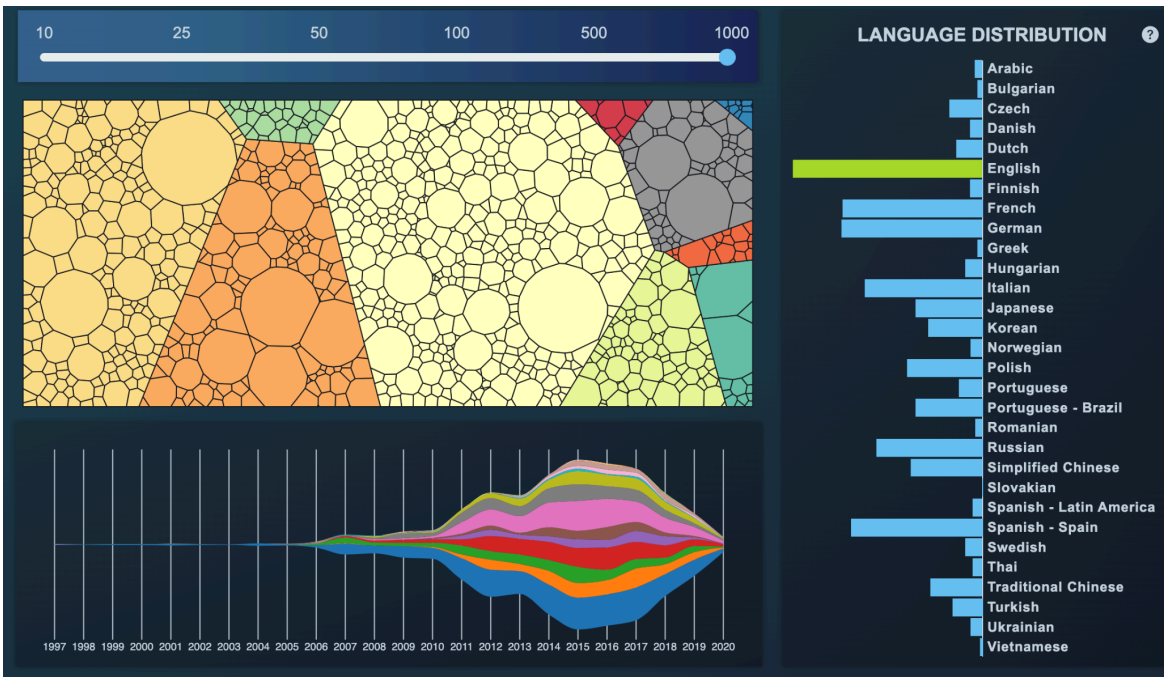
Tom is a video game enthusiast and an aspiring video game developer. He wants to start exploring past games and gain an idea of what would be needed for a game to be popular. He hopes to identify popular features and create a game that will reach and attract a lot of players.

To start off, Tom wants to **compare popular genres from previous years**. He looks at the streamgraph Genres Over the Years, where he can discover and compare trends of different genres over time and learn what genres he should aim to have in his future game. He notices that some genres are very thin, and is curious about what genre it is. As he **hovers over the genre mark**, he's able to see the name of the genre.

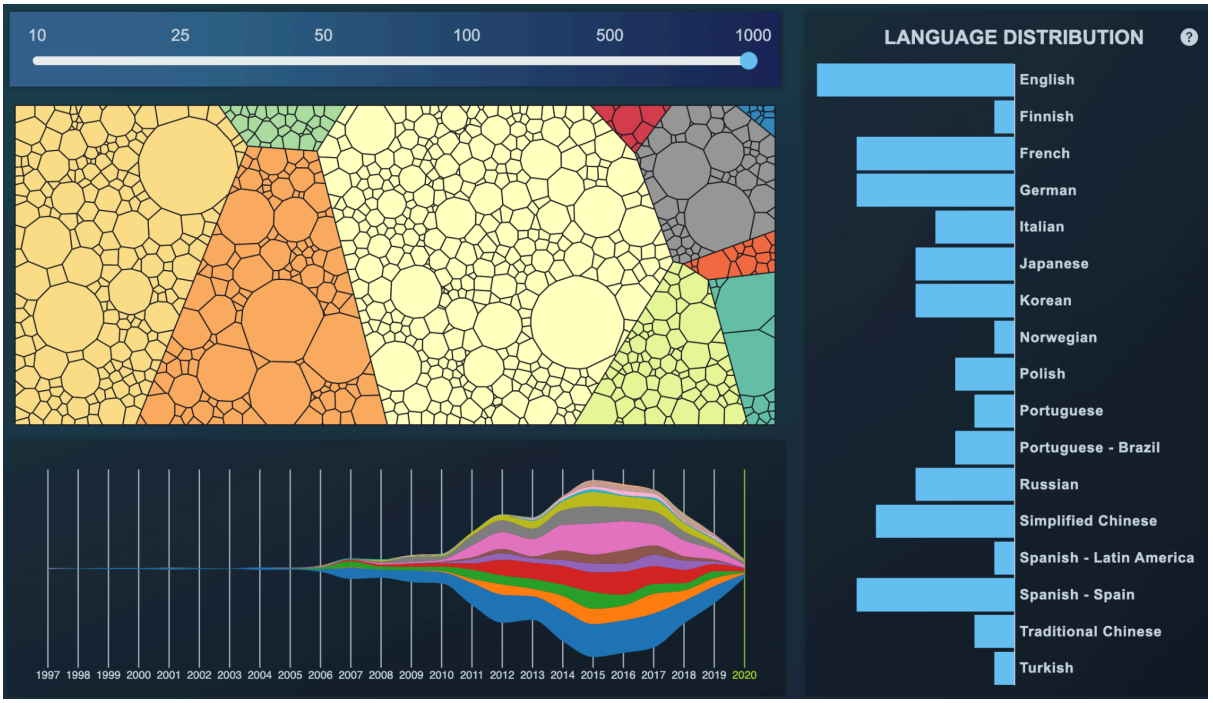


Next, Tom looks at the Languages Distribution bar chart, where he can **discover distribution of games over the number of available languages**. Allowing him to identify the most frequent languages to include in his game should he wish to reach a wider audience. Curious about the exact count of games that have the language 'Japanese', he **hovers over the 'Japanese' bar** and sees a tooltip with the exact count.

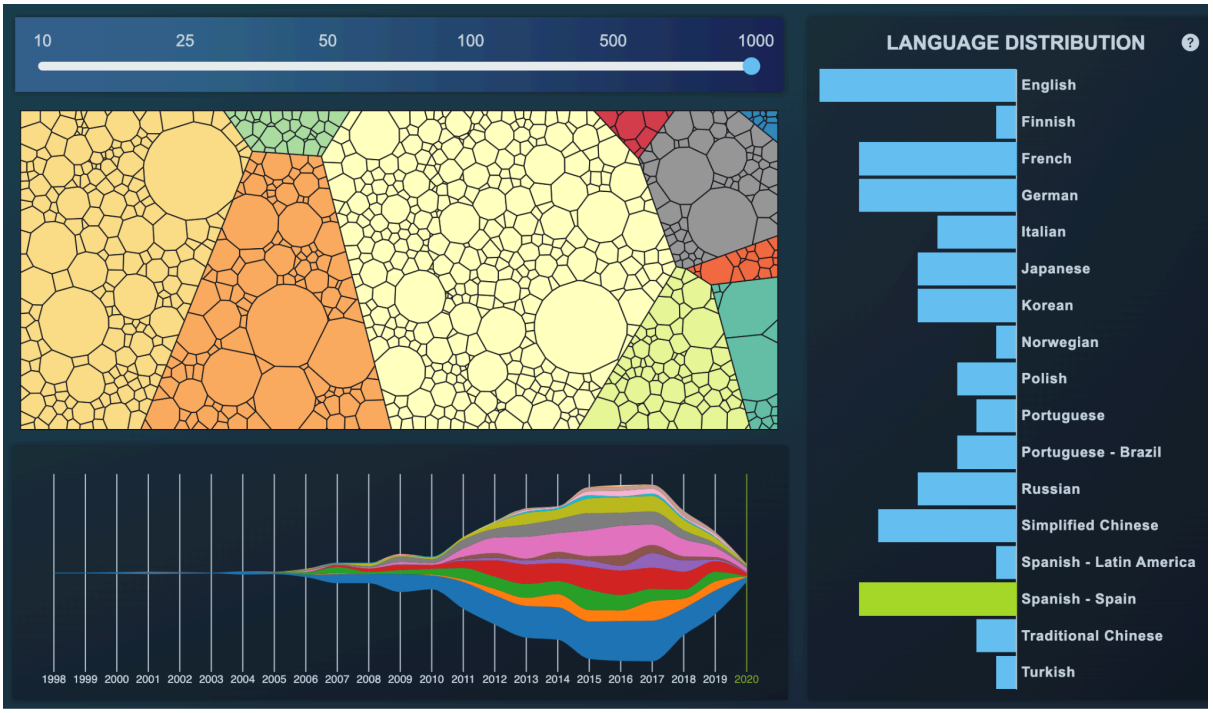
However, after seeing the language distribution, Tom decides that he wants his game to only be available in English, as he doesn't have the funds to translate his upcoming game to other languages. He **clicks the 'English' bar** and watches the Genres Over the Years streamgraph update to only feature games that are available in English, allowing him to **gain insight into popular genres of English games over the years**.



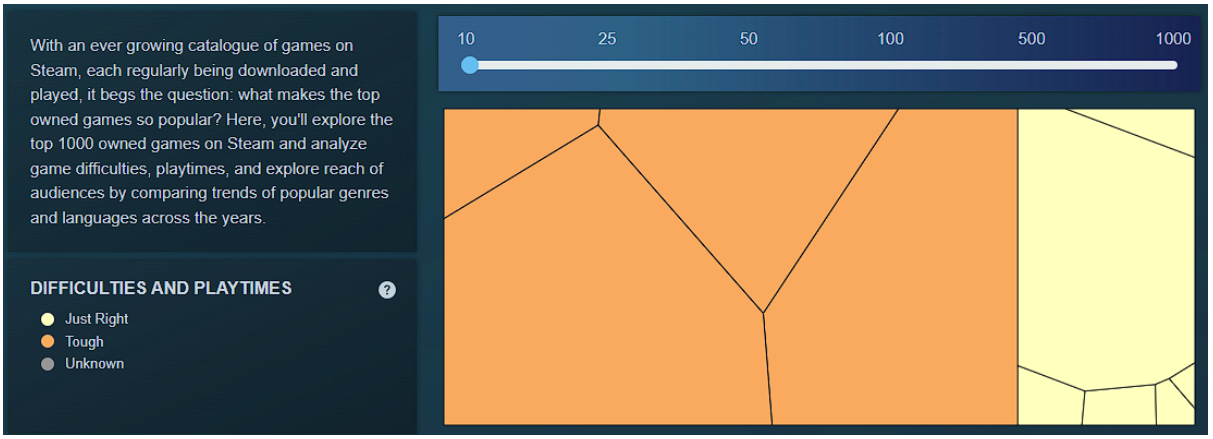
While exploring, Tom's reminded of the year 2020 and is curious how everyone staying home affected what games were released that year. In particular, if there was a more even distribution of games available in different languages in an effort for people to connect. He navigates to the streamgraph and **clicks the year 2020**, causing the language distribution chart to update, allowing him to **discover the distribution of games available in different languages that were released in the year 2020.**



He notices (for example) that 'Spanish - Spain' is almost as popular as 'English', and **clicks on the 'Spanish - Spain' bar**. Allowing him to view the **distribution of languages in the year 2020** on the bar chart as well as **genres of Spanish games over the years** in the streamgraph simultaneously.

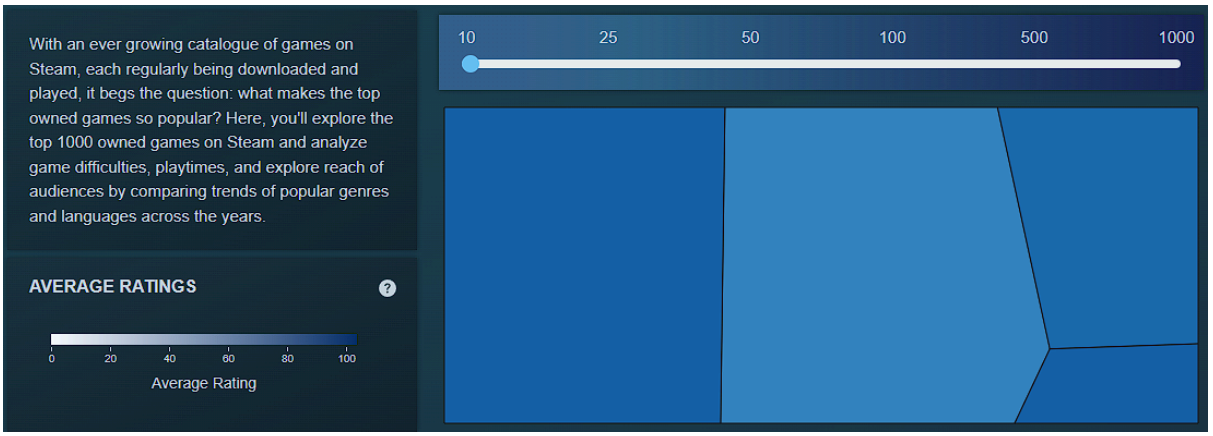


Finally, Tom wants to find an example game with a high median play rate to aim for, so he goes to the Difficulties and Playtimes Voronoi treemap and looks for the largest cell. He only really cares for the most exceptional games so he sets the view from top 1000 games to top 10 games using the slider, allowing him to **identify the top 10 games**.

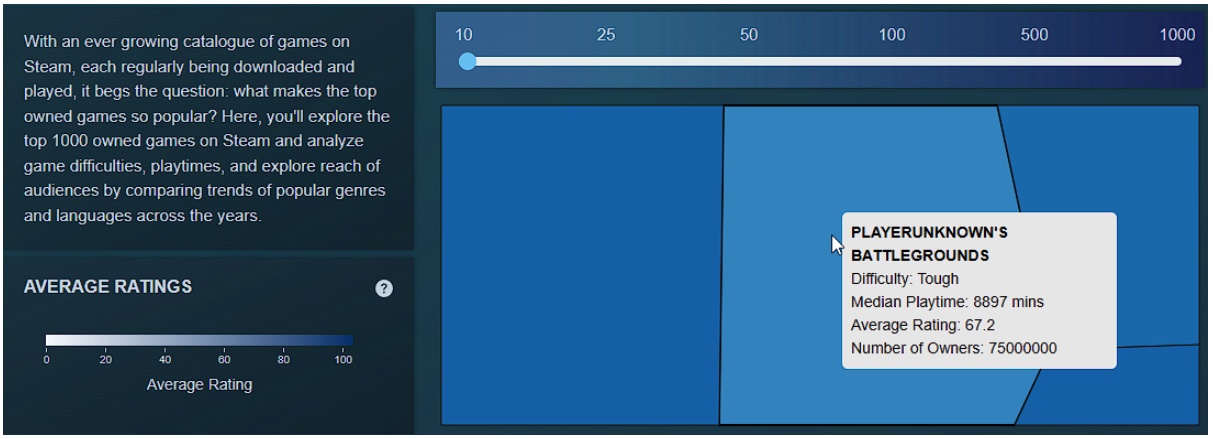


He notices that the largest cells have a difficulty of “Tough”. He **clicks on a “Tough” game cell** so he can view only the games that are listed as “Tough”. Now, these games are colour encoded by their rating, allowing him to **discover the trend of how “Tough” games are rated**. He notes that all but one of the cells have similar shades of blue, indicating that the games have similar average ratings, at around ~80 from looking at the legend. He

notices that the lighter cell is also the largest.



He finds the exact rating of the game with the longest median playtime by **identifying the largest cell** and hovering over it.



Reflections

Most of the major changes we had for our project were in terms of the dashboard layout. In our initial design proposal, we had a fairly simple layout, with our graphs, legends and titles all on a white background. During our M3 check in, our TA made a quick suggestion about emulating the design of Steam through our project. After a quick discussion, we decided to run with it and update our design. With that was mainly changes in the positioning of our views, and a different colour scheme for our project.

Other small changes involved adding spinners when treemap was loading - as it would usually take a while to reset or change after the user interacted with the slider. We also added question mark icons for each view, with instructions for how the user could interact with the view.

Changes in Visualization Goals

A major change we had in our visualization goal was describing our problem domain and what we wanted to achieve with our project. Our initial Milestone 1 proposal was more focused on allowing users to discover new games to play according to user interests and preferences. Although this was something we were interested in implementing, it's also a fairly intricate problem and can involve a lot of user customization in order to find their

preferred game. This resulted in our tasks being fairly complicated, and as we were working with a large dataset, it was harder to think of ways to narrow down games in order to present them individually to the user.

As we kept running into these issues working on our design proposal for Milestone 2, we decided to pivot and change our problem domain. We decided instead to focus on analyzing the top owned Steam games and delve into why these were owned by so many people. This allowed us to more freely explore possible views as we were dealing with fewer items from the dataset, and a clearer idea of how our tasks and design would look for the user, as we wouldn't need to worry as much about user preferences of genres and ratings.

Changes in Technical Goals

Our technical goals didn't experience much change over the course of our project. We kept our design from being too ambitious after looking at previous Hall of Fame winners to gauge what was possible in D3. With the practice from programming assignments, we were able to transfer our learnings appropriately to our project without running into any major or design-changing issues.

A slight change we had in terms of technical work from our original proposal was our preprocessing of the dataset. We had originally planned to store the entire dataset as a JSON file in our GitHub repo and do the preprocessing in JavaScript on the fly. However, our JSON file turned out to be too large to be pushed up. Instead, we wrote a Python script to first filter our dataset, and write that filtered dataset to a JSON file. We store this smaller file on our GitHub repo instead, alongside a zipped version of the original dataset.

Reflecting on our original proposal

Looking back at our original Milestone 2 design proposal, it was realistic in terms of what we were able to implement and is possible in D3. While planning our design, we were careful about choosing views that we knew were possible in D3. For us, this involved diving into the previous Hall of Fame winners for this course - seeing previous implemented designs in order to get an understanding of what was possible. It also involved a lot of research. For each view idea we had, we made sure to search and see if there were implementations of it already in D3. For instance, stumbling upon a Voronoi treemap and looking to see if it was possible to implement in D3.

Things we've had to change or abandon

One thing we did end up changing was how to reset our treemap view from layer 2 to layer 1. We originally planned for it to reset if a user clicked on any space just outside the treemap. However, as we were implementing it became clear that this was not ideal. Since the treemap is space-filling, there is nowhere on the map itself that can be clicked to reset. Having a reset occur when clicking anywhere outside the treemap might cause unintended resets while navigating other views, which could be very frustrating as a user. We instead implemented a reset button that would only be active if the treemap was on the layer 2 view. This also helped make it clearer to the user how they could navigate back to the layer 1 view, as the button has a different colour on its active state vs its disabled state.

Things we'd do differently

Overall, we're fairly happy with our project, especially considering time constraints and effort put into learning D3. One thing we did mention in our original design proposal that we didn't get to implement was allowing users to select a range of years on the Genres over the Years streamgraph. This would then update our Language Distribution barchart and allow users to look at and compare language distribution over a specific range in time instead of just one year.

Another thing we would have thought to accommodate/include in our original plan is the time required for a view to render. Our Voronoi treemap in particular does take a few seconds to fully render, and we put in a bit of extra effort to add in spinners to provide user feedback and indicate that the view is currently loading and not stuck or broken.

Lastly, a nice to have for future iterations or other visualization projects is to implement a responsive web design. For the purposes of this assignment, we focused on implementing the project for a specific screen size, with users having to adjust zoom to appropriately for the entire dashboard on their screen. Having a more responsive design would eliminate this extra step and create a smoother user experience.

Credits

Our project is built upon this database (https://github.com/leinstay/steamdb).

For styling and general colour scheme, we took inspiration from the existing Steam homepage design (https://store.steampowered.com/).

Certain components such as the slider, spinners, controller icon, question mark icons, and reset button were Bootstrap components we then further customized.

The Voronoi Treemap uses this external package (https://github.com/Kcnarf/d3-voronoi-treemap) and the code outlined in their "TL;DR" was used as the basis for our implementation, which was edited to match our two-layer structure.

There were a couple of previous course projects we were inspired by. *Bored? Games!* from https://www.students.cs.ubc.ca/~cs-436v/22Jan/fame/ contributed to our idea of exploring game genres, as this project focused on exploring board game themes, with their network graph colour encoding board games by theme and size encoding by popularity. We also took inspiration from the project *Actor Adaptability* from https://www.students.cs.ubc.ca/~cs-436v/20Jan/fame/, which had a somewhat similar network graph, colour encoding actors by their most acted in movie genre. These were ideas we adapted to our Voronoi treemap, colour encoding games by difficulty and size encoding by median playtime. We also took inspiration from project *Domain Expansion: Expanding Anime Horizons With Data!* from https://www.students.cs.ubc.ca/~cs-447/23Sep/fame/ for our streamgraph. Specifically, their stacked barchart that focused on counts of manga in different genres throughout the years. We did however, want to focus on a more general overview of how genres changed over the years, and instead adapted this idea into a streamgraph to emphasize that instead.