GameVis: Game Data Visualization for the Web

Victor R. M. Feitosa  José G. R. Maia  Leonardo O. Moreira  George A. M. Gomes

Federal University of Ceará, Virtual University Institute, Brazil

Figure 1: GameVis’ example charts included in the source package. While the first chart provides a delta-fashioned info of score over time, the second provides an end-result chart of collected character items during the match.

Abstract

Gaming and eSports have become increasingly popular and complex. For that reason, it is necessary that tournament hosts and game developers provide a great amount of meaningful information for their audiences. Data visualization tools and frameworks are of utmost importance since these provides means to accomplish this task. In this paper we propose GameVis, a framework designed to aid developers in building game data visualization components for the Web. GameVis allows multiple data formats to be translated into a single data structure that can be extended and customized in many ways, allowing visualizations to suit different screen sizes and platforms while showing a great amount of variation and meaning. Visualizations built within the framework may feature animations, annotations and might even adapt to different visual themes in order to provide a richer, more interactive user experience. This is shown through experimental results.

Keywords: visualization, game analytics, tools.

Authors’ contact:
victor.rm.feitosa@outlook.com
{gilvanmaia, leoomoreira, george}@virtual.ufc.br

1. Introduction

Electronic games were never such a fast-evolving, lucrative and creative market. These games are often looked at as mere form of entertainment. However, modern gaming represents a complex, exciting phenomenon worthy of attention. Most games and genres provide a common ground for players to bring forth real-life experiences such as dedicated organizations, game community meetings and worldwide eSports events. Some Websites promote game streaming, forums and communities. Curator systems allow people to expose their opinion on games and other digital content they own and play. Therefore, gaming allows players to reach a myriad of people from the entire globe, interact with them, gather knowledge about games, and even build successful careers in this field.

Since games play a big part in social media, the need for gathering, storing and analyzing data from games, game matches, players and else is growing rapidly [Jackson 2013, NewZoo 2015]. For that matter, game data visualization is becoming an important asset for game developers, professional gamers and general audience [Hazzard, 2014].

Data visualization components are becoming a standard feature in online competitive games. These components can help developers to maintain and polish their games. Players can use data visualizations to improve their skills by acting based on the displayed data and bring out the best playing practices by making an extensive analysis of data collected from thousands of other players. Even data from the game itself may be used for such purposes. Visualizations can also be of use for spectators: the crowd can make use of real-time gameplay charts in competitive events to better understand how the game is flowing and even to
Understand why a certain team has an advantage over the others.

Visualizations can be built using any kind of graphical API (Application Programming Interface) or framework. Libraries such as the D3 [Bostock et al. 2011], PhiloGL [Belmonte 2015], Google Charts [Google 2015], ProtoVis [Bostock and Heer 2009], OpenGL [Segal and Akeley 2015] and a plethora of others can be used to build data-driven visualizations from ground up. However, most of them are not oriented specifically to customizable data formats and neither are they constructed for game data visualizations purposes. These aspects were the major motivations for the conception of GameVis.

GameVis is a game data visualization API designed to help game designers, players and enthusiasts organize and visualize data from virtually every kind of digital game. GameVis was built to be customizable, cross-platform and easy to adapt to the user’s needs. It works across all devices supporting HTML 5 and JavaScript browsers. Some of its features include:

- Clean, easy-to-learn, highly customizable API with simple coding semantics;
- Industry-standard data exchange format;
- Support for a wide range of graphical chart elements such as lines, curves, bars, tooltips, tokens, graph ticks, etc.;
- Chart and component animations;
- Platform-agnostic; and
- Genre-agnostic.

The remaining of this paper is organized as follows. Section 2 contains a brief discussion about games and visualization. Prior works found in the literature on game data visualization are analyzed in Section 3. Section 4 presents GameVis and its architecture. Implementation details about the GameVis are shown in Section 5. Experimental evaluation and results are scrutinized at Section 6. Finally, conclusions and future research topics are presented in Section 7.

2. Visualization and Games

Data visualization has been an active development subject from the dawning of the digital era. It has also been a popular topic of scientific research from as early as 1786, as illustrated by William Playfair’s “The Commercial and Political Atlas”; in which the author states that tables are not the optimal way to represent certain data formats. This implies that over two centuries ago scholars were already concerned of how to display data in an understandable manner, hence how to visualize data correctly.

Visualization is the process to present data as images in order to communicate information, explore data, observe phenomena, formulate new questions, provide means to achieve answers and to support decisions [Ware 2000]. Visualization explores results about the sophisticated mechanisms behind human perception and memory, thus allowing charts to encode information as colors and abstract shapes efficiently [Kahnemann 2011].

Games have been using data visualization for generations. Most visualization components in older games are related to gameplay, and are often treated as simple annotations designed to track the player’s evolution along the game [Bowman et al. 2012]. Modern electronic games make use of visualization components in a wide range of situations: to display match status; to provide gameplay feedback; and even to make annotations in real time, for example.

Modern games already offer visualization and data analysis features that facilitates the access of the game’s community to gameplay data, such as match overviews, competition information and player profiles. Dota2™ and Starcraft™ 2, for example, already present these features. However, users cannot customize the game’s visualization components and analyzes, neither can they feed data to them. Virtually any of these aspects present in modern games is limited to present only the game’s specific genre and event models.

Data visualization has become a fundamental tool for games, and not only for playing, but in every step of the development process [Joslin et al. 2007; Walner and Kriglstein 2014]. Visualizations can help game developers to balance gameplay mechanics, test for bugs, and even improve game features according to beta testing.

Gaming has become irrevocably social. Since the social sharing aspects of games have become popular, developers started including visualizations that relates to players’ performance and progress. Developers also started bundling features and tools to build comparisons between players and teams, as seen in Figure 2. Simulation games (Figure 3) were the first games to insert data visualization as a performance-tracking feature.
Moreover, with the advent of broadband Internet connections, game-related Web portals and communities now hold means to provide videos along with other multimedia content. Actually, most gaming platforms also provide support for browsing the web.

On the other hand, players can now publish their gameplay experiences by either uploading or streaming videos. The PlayStation® 4 controller features a share button that enables players to stream their gaming experiences by broadcasting their gameplay in real-time, sharing information with friends and viewing friend’s gameplay with ease. Portals such as Twitch TV and Steam promote game streaming and social media altogether. Valve’s Steam platform also promotes game reviews and discussion forums that integrate players in a wider social environment. Hence, visualization plays key role in the enrichment of such social experiences.

Independent gaming communities are building tools of their own to deliver gameplay data, even when publishers and developers do not offer direct access to these. Some examples are listed below:

- Dota2 replay parser by Bruno Carlucci [Carlucci 2013];
- Full Meter Apps’ Ultra Street Fighter IV Frame and Notes (2014) [http://fullmeter.com];
- ArcadePerfect’s KOF XIII Pocket Guide (2014) mobile application [reference];
- Challenge Exploration Quadrigram for World of Warcraft© [http://blog.quadrigram.com]; and
- Sc2Gears - A StarCraft 2 utility software for data analysis and replay [https://sites.google.com/site/sc2gears].

3. Related Work

Data mining and electronic games have been popular research topics for the past few years. Joslin et al. [2007] proposed a framework for logging and visualization of online gameplay data, intended to be used by both game developers and game testers. The tool displays log information about the game flow, objectives and the difficulty aspect of the challenges in the game. The author presents a case study under the light of a Massive Multiplayer Online Role Playing Game (MMORPG) and analyzes the issues related to this particular scenario as he iterates the need for visualization within the development workflow. The use of data analysis and visualization can maximize the effectiveness of the game’s design and improve the quality of the final product.

Thawonmas and Iizuka [2008] adopt a classical multidimensional scaling and a KeyGraph for analyzing players’ action behavior. These techniques are capable of discovering clusters of players who behave similarly
and interpreting action behaviors of players in a cluster of interest, respectively. Players are then classified as achievers, explorers, and socializers according to Bartle’s taxonomy [Bartle 1996].

Evidence of visualization being able to enlighten gameplay data and promote data gathering, as well as data promote playfulness, has been discussed by Medler and Magerko [2011], and Macklin et al. [2009]. These authors claim that visualization plays an important part in the task of analyzing data in order to provide insightful information related to players and gameplay. Visualization may even reveal social aspects such as sharing and competition.

Medler et al. [2011] proposed Data Cracker, a web tool designed to analyze online gameplay visually, as illustrated by Figure 6. Data Cracker shares a similar purpose with Joslin et al.’s work [2007], but also aims to aid game data gathering and storage.

Figure 6: Data Cracker Client [Medler et al., 2011]. A line chart shows the player’s gameplay history. However, it is only able to handle/present data from Dead Space 2® by Electronic Arts™.

Bowman et al. [2012] defined a design space and proposed a set of design patterns addressing the challenge of applying visualization technology to games. The authors asserts how this kind of visualization differs from general Human Computer Interaction (HCI), reaffirms the importance of telemetry, and explore visualization components used by several game genres to present gameplay information. The proposed design space covers topics such as: the primary purpose of the visualization; target audience; temporal usage; visual complexity; and a junction between the immersion and the integration aspect that the visualization proposes. The authors analyze the benefits of a standardized manner of constructing gameplay visualizations and discuss social motivators that are specific to the gaming community.

Wallner and Kriglstein [2014] proposed a visual analytics system designed to handle large amounts of time-dependent and multidimensional gameplay data collected by means of code instrumentation. The system is limited to 2D gameplay data represented as a graph, and, hence, integrates a number of graph processing and visualization algorithms. The authors present results obtained with data extracted from a MMORPG, a platformer, a FPS and a puzzle. However, users often need to refer to adjacency matrices in order to better comprehend the visualizations.

Figure 7: PLATO [Wallner and Kriglstein, 2014] can provide visualization for the results of complex analysis algorithms running over multidimensional gameplay data represented as graphs.

After a careful observation of the work developed in gameplay data visualization and game data analysis, there is still a gap between representing game data graphically, and allowing the gaming community to build their own visualizations. We propose GameVis to fill that gap by providing an easy, cross-platform solution aimed towards helping enthusiasts to build charts and visualization components that would conform to any game and data format. In the next section we present GameVis and how its designed to accomplish such goals.

4. GameVis

GameVis is an API written entirely in JavaScript that adopts HTML5 [W3C 2015] to render all of its graphical elements. The API is based on D3 and it adopts D3 code semantics in order to maintain a comfortable development experience for most developers. It is worthy to observe that D3 and JQuery share similar semantics. In the following subsections, we discuss how GameVis’ architecture and how each of its component works in order to provide users the ability to construct functional and meaningful data charts.

4.1 GameVis Architecture

It is of paramount importance to state that GameVis’ architecture is fashioned to facilitate clean code writing, module customization and the separation between input, data and rendering. This derives from the well-known Model View Controller (MVC) architecture. GameVis
separates its core features from customized code provided by users, while it also abstracts the origin of game data that is fed to visualizations – as seen in Figure 7.

In fact, GameVis does not concern with data analysis directly and it assumes that users are responsible to perform these tasks. Data analysis can take place in the data source as a pre- or post-processing step, or can be embedded to the visualization itself.

Figure 7: GameVis architecture at high level. It is based on the MVC model, thus providing a clear separation between data, charts and animations. D3 is used for rendering onto Web Browsers.

4.2 Data Source Component

GameVis decouples the sources of game data from the visualization. The responsibility of defining how data will be retrieved from a game should not be assigned to GameVis itself. Ideally, this task should be entrusted to specific, official APIs provided along with the game that instruments the code, hacks the game’s memory or scrutinizes its files in order to extract gameplay data. Hence, raw data originating a chart can be a local file, a database in a remote server or even come from data stream fed directly to the API.

Figure 8 depicts two components: the game data itself and the Data API – which is an API provided by the game publisher or by the game community that mediates the access to the game data.

The examples included in GameVis present the game’s data as local files of public. The data is stored in several JSON files and is referenced in the driver code (see driver definition in the next subsection). In the provided examples there is no middle-man between the data and the game driver. The examples include only a hard-coded approach to the data source implementation and access. The user can extend these concepts further to promote any kind of data fetching method. A popular example of a data API would be a public web API [VALVE CORPORATION 2013] – a public web service that can be used to retrieve data from various Valve’s games and its players.

4.3 Core Components

GameVis’ core components are designed in a MVC fashion. There are three modules that compose the core code of GameVis: drivers, data structures and graphics. The drivers and the data structures compose the ‘model’ part of the MVC architecture. Both these modules work together in order to provide a comprehensive data model to GameVis and also to provide support for concise data exchange with the JSON data format.

The driver module allows users to build pieces of code that are responsible for translating data fetched from the data source into the GameVis data structure. Each data source developer should implement a driver for GameVis in order to promote independence between GameVis’ internal data format and publishers’ data format. In the examples included in GameVis there is a Dota2 local data driver implemented that translates Steam's JSON data into GameVis data, but drivers implemented by the users could take advantage of web services, remote data streaming or any other solution of data fetching, and not relay only in local data parsing.

The data module offers a variety of data structures aimed to represent generic game data and events. Although the data module is a complete API module the user should feel free to customize the data structures to his need and even create new one as he sees fit. Most of the data designed in this module is modeled to fit most
gameplay mechanics of competitive games, but this piece of code should not be treated as a definite solution for all game genres. The data module can also act on its own without relying in a driver implementation, which allows users to create mock data before they have a working driver implementation.

The third module of GameVis’ core is the graphics module – it represents the ‘view’ aspect of GameVis’ MVC. This piece of code has a number of data structures that are responsible for rendering from graphical components to whole data charts. The graphics module handles every animation and rendering behavior the graphical elements have and it allows the styling of the graphical components to be controlled either by code or by Cascaded Style Sheets (CSS). Similarly, to the previous modules, GameVis’ users should feel encouraged to modify and add functionality to this module as they see fit.

Note that there is no ‘controller’ aspect in GameVis. That is due to the input being responsibility of the charts developer and it must be implemented and adapted for each case separately.

### 4.4 Customizable Components

GameVis allows users to customize the existing code base by adding new pieces of code to the API with the purpose of extending its functionalities. There are three ways the user can add custom code to GameVis: drivers, game data structures and graphics components.

The drivers are the only mandatory piece of custom code that publishers and communities should add to GameVis. Each game data format demands a driver implementation, for example, if a gaming community wants to display static charts of a FPS (first person shooter) game competition they should provide a driver that gathers and converts the game’s data to GameVis. Otherwise, if the community wants to display real-time game data of a match, they should implement a driver that streams data from the game and translates it promptly into GameVis. The later method differs from the former due to how the game’s data is retrieved and stored. While the former method can simply pull data from a static database, the later must implement a data streaming mechanism that feeds gameplay data into GameVis in real-time.

Users can also build new data structures that may fit their needs better. The custom data structures can be either independent of GameVis base code or can be a specialization of the existing classes. An example of a custom data type would be a new game data format suited to represent sports games. It is recommended that users stay concise with GameVis’ semantics so the code stays clean and compatible across GameVis’ code-base.

At last it is strongly advised that users create new graphic structures and data chart models for GameVis. The API comes bundled with only a few data structures and charts, hence it is far from having all of the components needed to represent every game data model precisely. Structures such as pies and areas, and charts such as directed graphs, tree charts and heat maps are perfect examples of new graphic elements that could be implemented into GameVis. The users can even improve existing graphic elements in order to extend their functionalities, for example, adding textures to graph bars or inserting video annotations into tooltips.

### 5. Implementation

This section describes the implementation details of GameVis and how it can be used in real web pages. The project is available online. The API code itself should also be used as a learning tool. Users can get inspiration to implement new graphics components, data fetching methods and extend the API at their own needs.

#### 5.1 Dependencies

As previously stated, GameVis is written entirely in the JavaScript language, thus enabling developers to insert dynamic content into webpages, to animate components and to modularize pages. In GameVis, the rendering aspect is achieved through the HTML 5 canvas, while the styling, through CSS 3. These are compatible with a number of browsers found in modern devices such as smartphones and tablets. GameVis optionally uses CSS 3 to stylize graphics elements and animate components with the help of JavaScript events.

GameVis uses D3 as the framework that handles most HTML and CSS tasks. D3 acts as the middleman between the end user and the lower level web browser features. Otherwise, it would be necessary to implement several graphical components and lower-level functionalities required to render data charts.

Modern browsers load JavaScript files asynchronously. However, scripts loaded in an arbitrary order might break dependencies between modules and lead to severe execution failures. GameVis adopts RequireJS [Burke 2011] to overcome these limitations.

#### 5.2 Initialization

```javascript
// Module configuration through requirejs
requirejs.config({
  baseUrl: './',
  paths: {
    d3: './d3',
    gamevis: './gamevis',
    drivers: './gamevis/drivers',
    matches: './gamevis/matches'
  }
});
```

**Figure 9:** An example of a RequireJS configuration code for a GameVis project.

The webpage’s directory structure must be set up correctly in order to have an environment suitable to execute GameVis applications. After uploading the main application code and dependencies to your web
After setting up configuration directives, developers can load up their code to the webpage. For convenience, a data fetching script is defined separately in order to modularize data loading in the examples included in GameVis’ package. The data loading script is defined as its own module. It is responsible for instantiating a driver class and loading up the data to be visualized. This process is depicted in Figure 10.

The resulting elements are then appended to the page, as demonstrated by Figure 11. The web page must contain an HTML code snippet that loads the RequireJS module correctly, routing its “data-main” directive to our main JavaScript file. The examples included in GameVis also define a ‘div’ element with a singular id to contain the charts, as shown by Figure 12.

5.3 GameVis code and assets

There are two game drivers included in GameVis’ source-code: a bare bones driver and a Dota 2 match results driver. The bare bones driver is a base interface created for users to build new driver modules from. It includes all methods that should be provided by a working driver implementation. On the other hand, the Dota 2 driver is a fully functional driver that gives a glimpse of what users can do to feed game data into GameVis, while limited to only represent non-temporal gameplay data loaded from static files. Users can implement more sophisticated drivers by taking advantage of all features of the data module, such as: building charts based on temporal gameplay events; use data streaming; and even retrieve real-time gameplay information from web servers.

GameVis also offers a handful of examples that do not use drivers at all. These examples use hard-coded JavaScript files that register and react to time-sensitive gameplay events, thus providing a much richer visualization experience. Above all that, the examples included with GameVis can demonstrate how users can set up projects, import modules and use the API extensively. Finally, novice users are recommended to keep the project structure presented in GameVis’ examples until they feel comfortable with its coding standards.

6. Experimental Evaluation

In this section we perform practical evaluations over GameVis. This is the experimental setup in which we take the API to simulated situations in order to observe both, its maturity and performance. These are the experiments we conducted: a Simulated Data Match – with simulated data events; and a Real Game Data Match – that uses a real game driver implementation.

6.1 Experiment I – Simulated Data Match

This experiment (Figure 13) was constructed with the purpose of proving that GameVis has the ability to represent data-over-time events, one of the most important aspects of visualizations in competitive games.

The example describes a simulated MOBA match. The code simulates several game events and updates the time counter of the match periodically. The players are assigned kill-scores and gold in every time stamp. The data is fed unto the match’s data structure and used by the charts as the source of gameplay data. These charts
may also add tooltips to time stamps to indicate events and attach sensible information related to that particular game event. For example, an outstanding move performed by a team could be added as an annotation. The charts provided in this experiment display comparisons between teams and their performance throughout the match. As can be seen in Figure 13, a line and a bar graph are used for comparing values over time, while a status token chart displays players’ statuses over time.

These example charts may be used as a base code for developers that want to implement drivers sensible to game events based on temporal data.

6.2 Experiment II – Real Game Data Match

This example (Figure 14) uses the Dota 2 Driver implementation, a driver that translates Steam’s Web API Dota 2 data into GameVis’ native data format. Since Steam’s Web API does not support time-based telemetry, the driver only provides statistics about the match’s result. The charts related to this example also display only results information such as ending match points and items.

Game features such as items are treated as GameVis’ resource structures. Each resource can have a parent resource, e.g., an item can belong to a player’s character. Note that resources can represent any kind of key-value attribute a player may have. Users must also note that knowledge of GameVis’ semantics is essential if they want to take full advantage of the game driver’s features.

These examples can be used as a base for new driver implementations. Moreover, they exemplify how users can separate data fetching tasks from chart rendering and input handling in their projects.

6.3 Experiment III – Custom Graphics Element

In this experiment (Figure 15) we take advantage of GameVis’ modularization ability by creating a custom graphic element. The custom component is a status polygon, a chart that is found mostly in competition-intensive games and is generally used to compare avatars’ statuses and attributes. This widget is composed of a radar-like shape with several radial axes that representing particular attributes or statuses. A polygon shape inside of the radar displays how accented each attribute is from zero to a maximum value defined by the game’s specific metrics.

The status polygon implementation follows GameVis’ semantics, thus providing several methods that enable the code’s consistency with the core modules. This implementation also provides the inclusion of tooltips, attribute names and animations. The Status Polygon component is located in a dedicated script file and can be included in any other script, similarly to GameVis’ core modules.

Furthermore, this experiment demonstrates how new graphics can be created and integrated within GameVis. This also shows how users can keep the components’ compatibility with the core modules. We suggest that
users take some time analyzing this example in order to learn how to build new graphic elements for GameVis.

**Figure 15:** The creation of custom graphic elements for GameVis.

### 6.4 Result Discussion

Each experiment demonstrates an aspect of GameVis. While the first experiment ensures that gameplay data over time can be represented with little effort, the second demonstrates how straightforward it is to setup a game data driver using the API. Finally, the third example drifts away from the data aspect of GameVis: it features a custom chart code with annotation support that can be plugged into any visualization project. Developers should feel encouraged to tinker with the examples and modify them to better fit their needs.

### 7. Conclusion and Future Work

Game analytics and visualization is a trending topic for further development of gaming and also eSports as people get more involved through competitions and social experiences. GameVis addresses this challenge by means of a clean, simple and powerful HTML5 API for accessing game metadata. It was shown through experiments that GameVis can handle different kinds of charts, game genres and data formats. Moreover, our framework also allows users to explore visualizations by resorting to rich annotations for interaction analytics.

Further investigation should consider the following subjects: what-you-see-is-what-you get visualization editor; real-time game visualization with streamed data; visualization synchronization management; adaptive chart parameters for easier configuration; genre-specific tools and classes; data-driven design; integration of algorithms for analytics and knowledge discovery. Some of these topics are already work in progress. Such development corroborates that GameVis is, by itself, a relevant contribution.

### References


