Masaryk University
Faculty of Informatics

Visualization of a run of a security game

Bachelor's Thesis

Juraj Uhlár

Brno, Spring 2017

Masaryk University
Faculty of Informatics

Visualization of a run of a security game

Bachelor's Thesis

Juraj Uhlár

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Juraj Uhlár

Advisor: RNDr. Jan Vykopal Ph.D.

Acknowledgement

I sincerely thank my supervisor RNDr. Jan Vykopal, Ph.D. for allowing me to work on this thesis a providing any and all possible support. His advice, frequent consultations and effective communication style is what allowed me to finish the work on time and enjoy the process along the way.

I would also like to thank the members of the KYPO project who took part in the user testing of the visualization tool for giving invaluable feedback and ideas for future development.

Special thanks goes to Mgr. Valdemar Švábenský for his advice and for validating the usefulness of my thesis by using it in his own.

Lastly, I would like to thank my family and friends for their support.

Abstract

Cybersecurity games are a popular and effective method for teaching, exercising and testing cybersecurity skills. Over the course of the game, various types of events are logged and can be used to assess the progress of the game, to alert instructors to players in need of help and to improve the design of the game.

To help accomplish these tasks, a web application visualizing security games in the KYPO: Cyber Exercise and Research Platform at Masaryk University was developed according to given requirements. The application was user-tested in a group session attended by 7 cybersecurity game instructors. The experiment validated the usefulness of the visualization and also revealed a variety of potential improvements - some of which were immediately incorporated into the application, while the rest serves as a roadmap for future development.

Keywords

KYPO, cybersecurity, cybersecurity games, visualization, interface design, JavaScript, D3, Angular

Contents

1 Introduction **1**

2 Background **3**
2.1 *Games and simulations in cybersecurity education* 3
2.2 *KYPO platform* 3
2.3 *KYPO security games* 4
2.4 *Visualization* 5
2.5 *Existing solutions* 8
2.6 *State of Javascript development in 2017* 9

3 Design **13**
3.1 *Motivation* 13
3.2 *Visualization requirements* 13
3.3 *Integration requirements* 14

4 Implementation **17**
4.1 *Building the visualization chart* 17
4.1.1 Creating a mockup 17
4.1.2 Selecting the tools for implementation of the mockup 18
4.1.3 The internal workings of D3 19
4.1.4 Visualizing KYPO game events 21
4.2 *Building the user interface* 23
4.2.1 Porting the user interface of the mockup to Angular 2 24
4.3 *Backend server* 26
4.3.1 Node.js 27
4.3.2 RESTful API 28
4.3.3 Deployment 30

5 Evaluation **31**
5.1 *Experiment design* 31
5.2 *Tested hypotheses* 33
5.3 *User testing results* 33
5.3.1 Tasks evaluation 34
5.3.2 Tool usefulness ranking 36

5.3.3 Session recordings 36
5.3.4 Revealed issues 39
5.3.5 Future development suggestions 40

6 Conclusion **43**

Bibliography **45**

A Complete wording of the user testing questionnaire **49**

B Content of the thesis archive **51**

C Instructions for running the application locally **52**

List of Figures

2.1 Simple bar chart representing 10 floating point numbers. *6*
2.2 Example of a scatter plot with multiple mappings. Source: NVD3.org [16] *7*
2.3 A visualization of KYPO game level times by Miloš Barták. Each line corresponds to a game run of a single player. The time spent on each level is displayed on the X axis. Each level has a corresponding color. *9*

3.1 A diagram illustrating the modularity of the solution. *15*

4.1 Visualization mock-up *17*
4.2 Basic div-based bar chart generated using D3 *20*
4.3 Visualization chart displaying the entire dataset of a KYPO game *23*
4.4 Structure of component relationships in the Angular application *25*
4.5 Visualization chart embedded in an Angular application *26*
4.6 An implementation architecture diagram showing how different parts of the solution communicate with each other and the KYPO platform trough REST API. *29*

5.1 Task 1 - answers *34*
5.2 Task 2 - answers *34*
5.3 Task 3 - answers *35*
5.4 Task 4 - answers *36*
5.5 Visualization usefulness rating *37*
5.6 Heatmap of the user interface showing mouse movement *38*

1 Introduction

Each year, a larger proportion of our society's infrastructure is delegated to interconnected computer systems. Our personal and professional lives grow ever more reliant on online services that store our personal data on their servers. As a result, cybersecurity is a computer science field of ever growing significance and the demand for cybersecurity experts is on the rise [1]. The need for an effective and engaging cybersecurity education is evident [2].

The interest in cybersecurity games as a tool for practical and active learning of cybersecurity skills is growing as well [3]. Games can be run in a safe and highly controlled environment, while at the same time, mirroring real-world problems and scenarios. Creating a game that is welcoming to newcomers but also challenging for more experienced players requires testing and iteration. For game designers, it is helpful to analyze previous runs of the game to gain insight into player behavior and modify the game accordingly [4]. For game instructors, it is useful to have a real-time overview of the ongoing game to see if any of the players need their help. But the record of game events is usually stored in the form of a textual log, making extraction of useful information difficult. This problem can be solved by visualizing the event logs and providing designers and instructors with a tool that makes exploring the game data easier.

The main objective of this thesis is to explore the options of visualizing the run of a security game and develop a prototype web application capable of visualizing the events of games designed for the KYPO platform.

Chapter 2 provides an overview of theoretical background in areas of study relevant to the thesis. Chapter 3 describes a set of requirements for the implementation and the high-level design decisions that were made based on them. Chapter 4 describes the process and development tools used to implement the visualization, its user interface and the back-end server for the application. Chapter 5 contains the design of the user testing experiment and the discussion of its results, including ideas for future development. Chapter 6 concludes by summarizing the topic, my contribution, and the key results.

2 Background

The thesis connects 3 major areas of study: Cybersecurity games, visualization, and modern JavaScript development. This chapter serves as a quick introduction into each topic, providing a description of the context in which the implementation itself was developed.

2.1 Games and simulations in cybersecurity education

The adversarial nature of cybersecurity lends itself well to teaching its concepts through various forms of games, simulations and interactive competitions. Several studies [5, 3, 6] show that games are a beneficial educational tool, as they require active participation of the students rather than just the passive consumption of abstract information.

The objectives, tools, and knowledge requirements of cybersecurity games can be very closely mapped to the actual requirements of being cybersecurity professional. Games are also inherently engaging and motivating, simply by the virtue of being a social activity with elements of team communication, collaboration and competition.

However, teaching cybersecurity through gamification also presents numerous challenges. Games have to be engaging enough to hold students' interest and provide educational value at the same time. Matching game difficulty to players' skills and preexisting knowledge is a tough balancing act [7]. A game that is too easy provides little educational value. A game that is too difficult might cause frustration and discourage players from exploring the given topic altogether. Designing an educational cybersecurity game is, therefore, a long and iterative process. Ideally, the game is continually user-tested by real players. Their feedback and the data collected about their behavior during the game is then used to improve the game and calibrate its difficulty [4].

2.2 KYPO platform

The KYPO - Cyber Exercise and Research Platform is an environment for research and development of new cybersecurity tools and methods

[8]. It uses a virtualized environment running in the CERIT - Scientific Cloud infrastructure to simulate complex cybernetic attacks against critical infrastructure and analyze their impact. It is capable of "creating various scenarios involving extensive computer networks running services and applications and thus facilitates the detailed study of the emergence, spread, and impacts of current cybernetic threats" [9]. The project is the result of cooperation between the Institute of Computer Science at Masaryk University (ICS-MU), Faculty of Informatics and the Ministry of Interior of the Czech republic.

The platform is also used for educating members of security teams and students. It routinely houses cybersecurity games and exercises under the guidance of Computer Security Incident Response Team at Masaryk University (CSIRT-MU).

2.3 KYPO security games

To support the KYPO educational use case, a specialized framework was developed for creating and running various security games on the KYPO platform, along with a prototype game focused on penetration testing [4].

The games are structured into several levels that lead to a single goal, for example, data theft from a remote server. Each level is completed by submitting a **flag** - usually,f a short string that serves as proof the the student discovered the information he needs for the next level. Completion of each level is rewarded with a predetermined number of points that make up the player's overall score. If the learners struggle with the level, they can use optional hints at the cost of lowering their score. Players are able to skip each level or quit the game entirely at any point. The game ends when the player correctly enters the flag of the last level. Players can use external resources such as the internet to find the correct solution. Each level has a time limit for its completion, giving the player a rough idea about its difficulty, but the level can still be finished after the time limit expires [10]. To monitor the activity of the player, the platform logs timestamps of several types of events that occur over the course of the game:

- start and end of the game

- start and end of each level
- submission of incorrect flags and their content
- hints being used
- skipping a level
- displaying a level's solution (entering a help level)

These events capture only the interaction between the learner and the game module of the KYPO platform. They are purposefully generic and independent from the specific configuration of the game. The advantage is that they can be observed during any game of this type. The disadvantage is the limited amount of information they provide about players actions inside the game.

ID	Timestamp	Logical time	Level	Event
17280	2016-04-07 06:41:09	00:00:00	1	Game started
44095	2016-04-07 07:05:34	00:05:23	2	Wrong flag submited
43815	2016-04-07 07:30:23	00:15:28	3	Hint 3 taken
25780	2016-04-07 08:11:53	00:53:45	4	Correct flag submited

Table 2.1: Example of input data. The 'Logical time' attribute represents time since the beginning of the level.

2.4 Visualization

Visualization is a way of displaying information using a graphical representation. It is a tool whose purpose is to provide **insight into data** for its users. While visualization as a discipline spans a wide range of methods and can be dated back to the beginning of human history, for the purposes of this text, we will define it as "the use of computer-generated, interactive, visual representations of data to amplify cognition" [11].

The three main functions of visualization are storing, analyzing and communicating information. The power of visualization comes from the fact that vision is the most developed of human senses. Researchers estimate that 80 - 90 percent of the information transmitted to the brain is visual [12].

Consider the difference in the amount of cognitive effort required for two tasks:

- Selecting the largest number from a set of 10 floating point numbers.

  `179.3858, 172.6002, 133.9771, 292.571, 312.6785, 191.9028, 108.7445, 236.6949, 297.7262, 67.2187`

- Selecting the largest bar from a bar chart representing the same 10 floating point number values, as shown in Figure 2.1

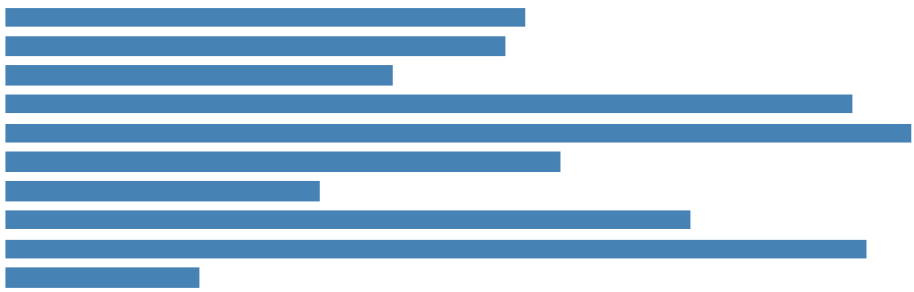

Figure 2.1: Simple bar chart representing 10 floating point numbers.

The nature of the tasks seems to be the same, yet the first one requires a noticeable amount of concentration, while the second one is almost automatic. Comparing the size and position of objects, spotting patterns, recognizing visual features of objects is a natural instinct of not only humans, but of almost all mammals. Processing abstract information, on the other hand, has to be learned, requires active concentration and depletes mental energy. This is why people find it tiring to compare prices of products in the supermarket but recognize

an acquaintance at the check-out line automatically. In fact, people cannot stop themselves from recognizing faces and their emotions, even if they tried. [13] [14]. Converting abstract information into its visual representation can lead to better and easier understanding of the information, discovery of new interconnections within the data and improved decision-making [15].

The process of visualization is often divided into four steps:

1. Analysis of input data
2. Analysis of output requirements
3. Mapping of data onto visual elements and their attributes
4. Enabling interactive manipulation of the data

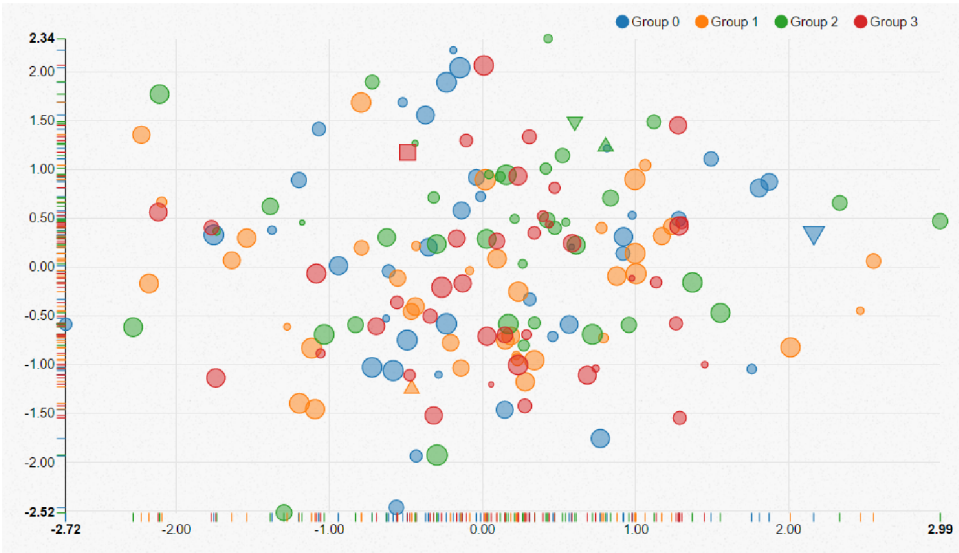

Figure 2.2: Example of a scatter plot with multiple mappings. Source: NVD3.org [16]

The core of the process is the creation of mappings between the underlying data and the visual elements that represent it. A mapping is a relationship between an attribute of the data point and the attribute

of its respective visual element. For a example, in a simple bar chart, the value of each data point is mapped to the height of the corresponding bar. More complex data require the mapping of multiple attributes. In the scatter plot shown in Figure 2.2, for example, different data attributes are mapped the X position, Y position, color, shape, and size of the visual elements.

The choice and design of these mappings determine the usefulness and readability of the visualization. The goal is to choose mappings that come naturally to the user and provide him with a clear and accurate representation of the data.

Visualization of security games and related work

As far as I was able to find, there is little to no literature available focused specifically on the visualization of security games. Security visualization in general, however, is a field with growing body of literature [17, 18] and an active community [19].

Most security visualizations try to display network topology, data flow, packet logs and other data directly tied to network security. The goal of this thesis is slightly different, as the input data consists of meta information about the players progress through the game and their interaction with the KYPO platform, rather than specific information about their actions in the game.

2.5 Existing solutions

The events of each security game are currently being recorded in a CSV file (the format of which is described in Table 2.1). If needed, one could extract a lot of information from the CSV log directly, but it would be an inefficient and tedious process.

A visualization of the logs was created by Miloš Barták [10] as a part of his diploma thesis (figure 2.3). A simple diagram generated using Google Charts shows how much time each player spent solving each level. While this is useful for comparing level times, it does not provide the complete picture of the game. Without displaying data about hints, incorrectly submitted flags or the use of help levels, it

cannot be used to investigate *why* certain levels take more time than others.

The original design, however, is solid and was used as a basis for the final visualization, which builds on it significantly.

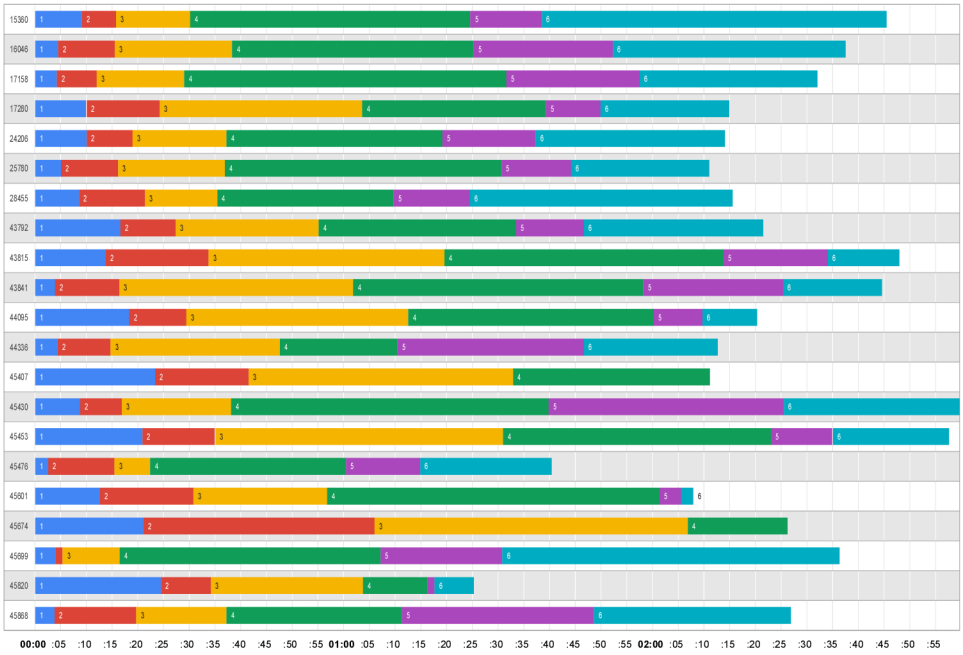

Figure 2.3: A visualization of KYPO game level times by Miloš Barták. Each line corresponds to a game run of a single player. The time spent on each level is displayed on the X axis. Each level has a corresponding color.

2.6 State of Javascript development in 2017

The development of any non-trivial web application necessitates extensive use of the JavaScript language. JavaScript is notorious for its dynamic and ever-growing development ecosystem consisting of dozens of competing tools and libraries [20]. Therefore, it is useful to describe a simplified view of the JavaScript landscape at the time of developing the application.

The web is no longer a place for documents and static websites. The most visited sites on the web are, in fact, complex web applications with dozens of interdependent user interface elements that are expected to run smoothly and efficiently. Abandonment of native programs in favor of web applications that store data in the cloud and are available on any system is a growing trend that places ever more demands on the JavaScript language - demands that it was not designed to meet at the time of its creation.

But JavaScript is continually evolving to meet the requirements of its new role. In 2015 a new version of the ECMAScript standard was released, with new language features such as classes, arrow functions, multi-line strings, string-templates, block-scoped variables and more [21].

ES6 (shorthand for ECMAScript2015) is a significant step forward, but it is not yet supported in all browsers. Therefore, developers use transpilers such as Babel or TypeScript that allow them to write code in ES6 and then transpile it into ES5 to retain backward compatibility with older browsers. The transpilation is often combined with other optimization measures, such as

- Minification: removing all unnecessary characters from source code without changing its functionality to decrease file size.
- Concetation: combining multiple files into one to decrease the number of HTTP requests needed to load the application.

The build process is often automated using tools such Webpack, Grunt or Gulp [22].

React vs Angular2

There are a multitude of frameworks and libraries trying to make developing user interfaces on the web easier. They are all trying to solve two basic problems:

- Combining the power of HTML and JavaScript to render pages dynamically as a collection of interlocking, reusable components.
- Detecting changes in the state of the application and effectively making the required changes on the page.

Two most mature and widely used front-end development tools are React, created by Facebook, and Angular2, created by Google. React solves the first problem by implementing JSX - a syntax extension of JavaScript, that allows including HTML markup in the component-generating React code. Angular solves the same problem by enhancing HTML templates with its own programmatic syntax for data-binding, for-loops, conditions, event handling etc. To put it extremely simply: React puts HTML into JavaScript, Angular puts JavaScript into HTML [23].

There is also a significant difference in scope. React is JavaScript library focused just on building user interfaces from components. Angular, on the other hand, is a fully fledged framework with a command-line tool, support for cross-platform development, modules for building forms, animations, integrated testing and more.

The advantage of using Angular is that all the tools one needs for developing a web application are in one place and ready. Developers do not need to stitch their development stack together from a number of libraries and make time-consuming decisions about which ones to use. The decisions have already been made by the Angular team.

Another thing that sets Angular apart from other frameworks is that it is completely written in Typescript. Typescript is a superset of JavaScript that adds static types and ES6 features to the language. It can be compiled into plain JavaScript. The use of types is optional, developers can decide where it makes sense to use them. Static typing increases robustness of the code and makes it more readable, and easily testable. Static type checking allows developers to discover bugs at design time and makes them much more productive thanks to intelligent code completion.

In the end, Angular was chosen over React for the reasons mentioned above and also as a stakeholders' preference. There already are pre-existing KYPO related visualizations implemented in Angular and the secondary goal of the thesis is to further explore its capabilities and the possibility of integrating different types of Angular visualizations into a single dashboard-like application.

3 Design

3.1 Motivation

The motivation behind visualizing game events is two-fold:

- Provide game instructors with real-time overview of players' progress during the game that they can use to identify players who might need their assistance.

- Provide game designers with a tool that aids them in post-mortem analysis of the game. It should it make as easy as possible to answer questions like these at a glance:

  - Which levels take too much or too little time to complete?
  - Where are players most likely to get stuck and waste time trying an incorrect solution to the problem?
  - Where are players most likely to get frustrated, lose motivation and consequently skip the level or end the game prematurely?
  - Do player make good use of the hints and help levels available in the game? Do the hints work as intended?

3.2 Visualization requirements

To achieve the desired level of usefulness, a set of requirements for the visualization itself was defined:

- **Clarity**: The mappings of event attributes to their corresponding visual elements are obvious and intuitive enough that a user is able understand them without explanation after a couple of minutes of experimentation with the tool.

- **Compactness**: Visualization can display the entire dataset (with the expected size of hundreds of events) in a single diagram with reasonable readability and without being overwhelming.

- **Interactivity**: Users can display only the data-points they are interested in and hide the ones that are irrelevant to the question they are trying to answer at that moment. They can filter events based on their type or show only events from a specific level. Auxiliary elements such as axis can be toggled on or off.

- **Completeness**: If needed, user is able to easily display the complete information available about the particular event, by clicking on or hovering over the visual element that represents it.

- **Responsiveness**: Visualization scales itself according to the available viewport and is readable on devices ranging from tablets to large computer monitors.

- **Development flexibility and extensibility**: The visualization is easy to maintain, modify and extend. For example, it should be able to display events of a single level in logical time (the beginning of the level normalized to 0 for each player).

3.3 Integration requirements

The visualization chart itself is only one part of the application. The implementation has to be modular enough to easily integrate itself into the existing KYPO platform.

For that reason, we have made the decision to split the application into two distinct parts: a front-end client and a back-end server that communicate through a Web API. Again, we have defined a set of requirements for each part of the application:

- Front end client is implemented as a **web application**, as it needs to work across devices and operating systems and must be available instantly without the need to install anything. It provides the user interface for working with the visualization chart.

- Back-end server is implemented as a standalone **API-based application that serves as a storage facility for the events**. It exposes API endpoints for entering events into the database (either one by one or in bulk) and retrieving them from the database

based on a specified query (for example, all events from a single dataset).

- The client never accesses the data directly, it can only retrieve it from the server using the API. The two concerns (data storage and data visualization) are logically separated, independent applications. Each adheres to a pre-defined contract but does not require knowledge of the implementation details of the other modules in order to function.
- API is documented and the documentation is publicly available online.

The idea is that KYPO platform can use the API to send events directly to the back-end server as they happen in real-time. Front-end client then retrieves the data and visualizes them, providing an overview of the game almost in real time.

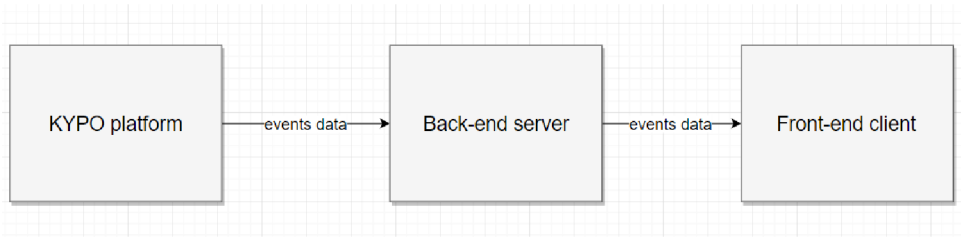

Figure 3.1: A diagram illustrating the modularity of the solution.

The advantage of using a separate data storage server is that it can run on a separate machine, be available on a different domain, and most importantly, be re-implemented in a different programming language that the front-end client. That makes it easy to integrate it directly into the KYPO platform later on if such a need arises. Implementation details of the architecture are further described in the following chapter.

4 Implementation

4.1 Building the visualization chart

4.1.1 Creating a mockup

The first step was to create a simple wireframe mockup of the design and basic features the visualization needed to have. There are numerous online tools available for exactly this purpose. In this case, however, a pen and paper had proven to be the most efficient method for creating the mockup shown in Figure 4.1 .

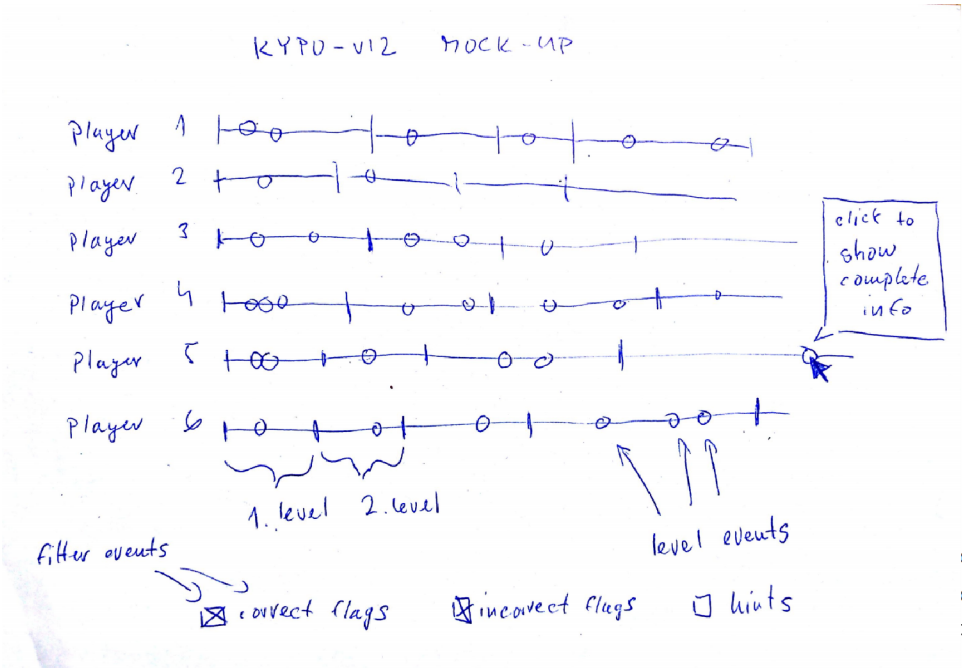

Figure 4.1: Visualization mock-up

The mockup serves mostly as a decision-making tool when choosing a development framework or library for the implementation. It clarifies what the visualization must be able to do, serving as a feature checklist against which potential development tools are compared.

The basic layout is identical to the pre-existing visualization created by Miloš Barták [10]. Each line of the chart is occupied by data from one player. The timespan of each level is signified by a line or a rectangle. Additionally, each event is represented by circle or some other visual element, its X coordinate determined by its timestamp. The type of event is signified by its color. The user can display all the information about the event by clicking on it or hovering above it with the mouse cursor. Users can use check-boxes below to chart to show and hide groups of events based on their type.

4.1.2 Selecting the tools for implementation of the mockup

There are many charting libraries and visualization tools available that could have been used for implementing the mockup. They usually differ in their capabilities, ease of use, quality of documentation, complexity, and level of abstraction.

First, an attempt was made to find a library that supports some kind of timeline visualization out of the box and modify it to suit the needs defined in the requirements.

Google charts is a charting library by Google. It has a wide range of use-cases from simple graphs to dynamic and interactive visualizations. It offers a large gallery of available charts - one of which was used to create the original time level chart shown in Figure 2.3. The possibility of extending the original chart and continuing development using Google's library was examined and rejected. While the Timeline chart is suitable for displaying periods of time with variable length, it was not designed to display individual events inside those periods. No straightforward way of extending the chart was found during the evaluation.

Vis.js is an open-source JavaScript visualization library with one of its modules dedicated to timeline visualizations. Its timeline chart supports a wide array of advanced features such as animations, set selection, zoom, editing, drag and drop, date localization and more. Its documentation appears robust but also unwelcoming to new users. Reading through it, no clear path toward implementing the prototype was apparent. The high number of overlapping features makes it difficult to orient oneself in its API, which appears complex and limiting at the same time. Its timeline module had a problem similar to Google

charts, in that it could display either longer periods of time, or single moment events, but not both at the same time, which is what was needed. For that reason, Vis.js was also rejected.

NVD3 is a charting library built on top of D3.js, trying to provide reusable components for D3, without taking away its expressive power. However, none of the components available out of the box resemble the type of visualization that needed to be built. The use of NVD3 was also rejected.

D3.js is a JavaScript library for manipulating documents based on data. The D3 shorthand stands for Data Driven Documents - a descriptive name that highlights the library's unique approach. Unlike all the libraries described above, it purposefully limits its scope to the core problem of visualization - efficient manipulation of documents based on data [24]. D3 operates on a much lower level of abstraction. It does not try to provide pre-made charts ready to use out of the box. Instead, it affords its users extraordinary flexibility and freedom to implement exactly what they need by exposing the full capabilities of common web standards such as HTML, SVG[1]and CSS. This trade-off makes it unsuitable for creating the most common types of visualizations, as even the creation of a simple pie chart requires decent knowledge of geometry, JavaScript, and the SVG standard. But if one needs to create a custom web visualization with little resemblance to common charts and graphs, D3 is the best and only option, even at the price of a steeper learning curve and longer development time. The use of common web standards as visualization output also brings other advantages, such as low file size and scalability. For those reasons, D3.js was chosen as the visualization library for the project.

4.1.3 The internal workings of D3

To understand how the visualization is implemented, it is useful to understand the basic principles of how the creation of visualizations using D3.js works.

1. SVG (Scalable Vector Graphics) is a vector image format based on XML. In SVG, visuals are defined using a markup language and can be stored as plain text. Since the shapes are drawn based on their mathematical properties, SVG images can be scaled indefinitely without becoming pixelated - a major advantage over bitmap-based images.

D3 provides a set of helper methods for *selecting and modifying elements of a web page* (such HTML and SVG elements and their attributes). For example, one can select all paragraphs on a page and make them red:

```
d3.selectAll("p").style("color", "red");
```

You also have the ability to *define element attributes as functions*. For example, one can apply different colors to odd and even paragraphs:

```
d3.selectAll("p").style("color", function(d, i) {
  return i % 2 ? "#fff" : "#eee";
});
```

The real power of D3 comes form *binding page elements to data*. The attribute-defining functions can then accept the data as parameters. For example, one can make a simple bar chart by making the width of divs on the page a function of numbers stored in an array:

```
var dataset = [ 50, 100, 150, 200, 250 ];
d3.select("body").selectAll("div")
.data(dataset)
.style("width", function(d) {
return d + "px";
});
```

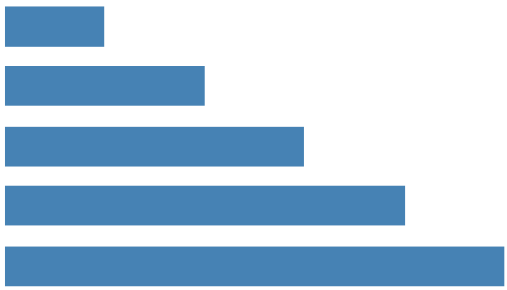

Figure 4.2: Basic div-based bar chart generated using D3

Figure 4.2 shows the resulting output of the code, assuming the divs' height, margin and color is set using CSS. D3 automatically binds the first div to the first number in a array, the second div to the second number at so on [25]. The number is then passed as *d* to the anonymous function, which is executed for every div element.

All D3 visualizations work using these simple principles [26]. With the general overview out of the way, the following section describes how the capabilities of D3 are leveraged in the visualization of KYPO games.

4.1.4 Visualizing KYPO game events

The first step is to transform the incoming data into suitable format and prepare helper functions and data structures needed during the visualization. We assume that different players can start their game at different times, but for illustration purposes, we want to visualize the game as if their starting times are exactly the same. Therefore, we need to enhance every event with an attribute that represents the *time of the event since the beginning of the game*:

- An array of unique player IDs is extracted from the dataset, assigning each player an index number later mapped to Y coordinates of event objects.
- The array of starting times is extracted from the dataset. Each player ID is associated with their starting time by the indexes in their respective arrays.
- For every event, a new `game_seconds` attribute is calculated as a difference in seconds between the timestamp of the event and the starting time of the player.

Step number two is the definition of scales. Scales are functions that map values from an input domain to an output range. They are used to transform the values in our dataset to attributes of objects representing the dataset.

- The input domain of our X scale is all the possible times of events in the dataset. The domain is represented as a range of values between zero and the time of the latest event.

- The output range of our X scale is all possible X coordinates of the event objects. The range is represented as a range of values between zero and the detected width of the browser window, minus some padding for aesthetic purposes.

- The input domain of our Y axis are the indexes of players in an array.

- The output range of our Y axis are all possible Y coordinates, ranging between zero and the detected height of the browser window, minus the padding.

With the dataset and scales prepared, generating and positioning circles representing the events is relatively straightforward:

```
enteringCircles
    .attr("cx", (d) => {
        return this.xScale(d.game_seconds);
    })
   .attr("cy", (d) => {
        return this.yScale(this.players.indexOf(d.ID));
    });
```

Each circle is also assigned a class according to its event type. The color of the circle for each class is defined in a separate CSS file.

Drawing a line for each level works on a similar principle: For every level-ending event (such as a submission of the correct flag or a level skip) a corresponding level-starting event is found and a line is drawn between them.

The scales are also utilized when generating axis: X axis is generated using the D3 axisBottom() function, while the Y axis is generated manually. The tooltip showing complete information about the event is added by simply adding a <title> element as a child of each circle.

D3 automatically keeps track of the binding relationships between data and visual elements. If the dataset changes, running the rendering function again will add or remove circles accordingly. The implementation uses `enter()` and `exit()` D3 functions to select the entering and exiting circles and apply transition animations to them.

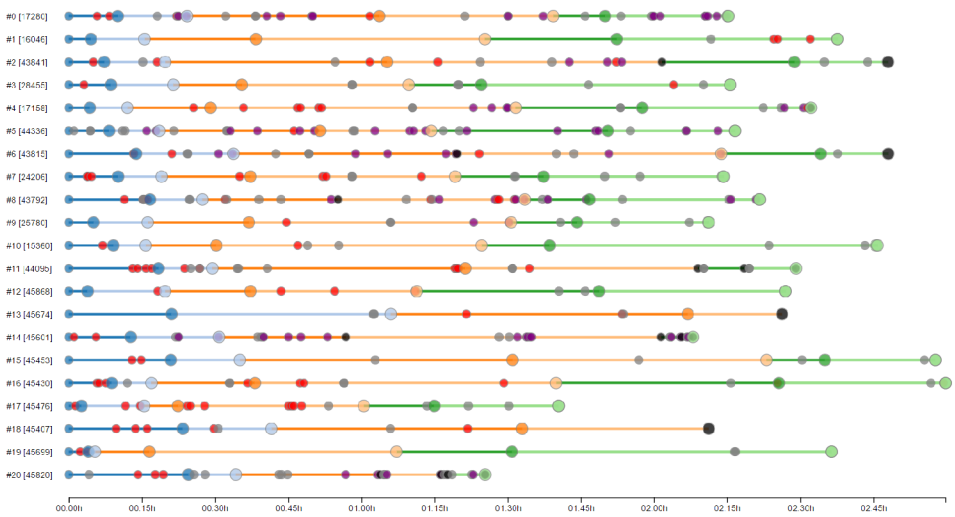

Figure 4.3: Visualization chart displaying the entire dataset of a KYPO game

4.2 Building the user interface

The user interface consists of the following elements:

- A checkbox for each event type allowing the user to hide or show the events
- Two buttons allowing the user to hide or show all events at once
- Checkboxes allowing the user to hide or show the X and Y axis.
- A drop-down menu allowing the user to display a single level from the dataset in isolation. Users can also choose to display a single level in logical time - meaning that the X coordinate of each event is determined by its time since the beginning of the level, instead of time since the beginning of the game (start of each level is normalized to zero).

In the first prototype, the interface was implemented in plain JavaScript and the visualization was being manipulated by manually selecting SVG elements and changing their properties. While this

was useful for quickly creating a functional prototype that could be evaluated by stakeholders, it also created a substantial amount of technical debt. The code was verbose and difficult to understand. Each new feature was exponentially more difficult to implement that the previous one and made the code less maintainable. Two major steps were taken to address these issues:

First, the system of filtering and manipulating the visualization was completely rewritten. Instead of rendering all SVG elements once and then manually hiding and moving them around, the algorithm would work with 2 datasets - an original dataset and a filtered dataset. Any change in the UI would trigger the creation of a new filtered dataset (based on the current state of the UI) and instruct D3 to run all rendering functions again on the new dataset. The rendering function then computes the difference between the old and new filtered datasets, rendering only the new elements and hiding the ones that are no longer needed. This approach is both faster and more maintainable, by grouping related functionality in one place and making sure each piece of code only has one job to do.

Secondly, the entire user interface was reimplemented using the Angular 2 front-end framework. Angular allowed us to break the user interface into semi-independent reusable components and easily bind our UI elements to the underlying data. The motivation behind using Angular, its advantages, disadvantages and implications are described in Chapter 2.

4.2.1 Porting the user interface of the mockup to Angular 2

The user interface was split into several components that communicate with each other.

- **The chart component** holds the entire visualization logic and nothing else. Its job is to receive the filtered dataset as input and render it.

- **The visualization view component** is a parent of the chart component. Its job is to receive the original dataset, allow the user to filter it using the web interface and pass it on to the chart for visualization. Any change in the filtered dataset in this compo-

nent automatically propagates itself to the chart component and triggers a re-rendering of the visualization.

- **The menu component** is the only component responsible for communicating with the backend server. It requests the names of all datasets and offers them to the user for selection. Once the user selects a specific dataset, all its data is fetched from the server and passed on to the visualization view. Communication with the server is abstracted in a separate service class.
- **The sidebar component** is third-party reusable Angular 2 component [27] that was incorporated into the application. It allows the menu to be hidden from sight when not need and summoned into view on demand.
- **The app component** is a root component of the application that holds all the other components.

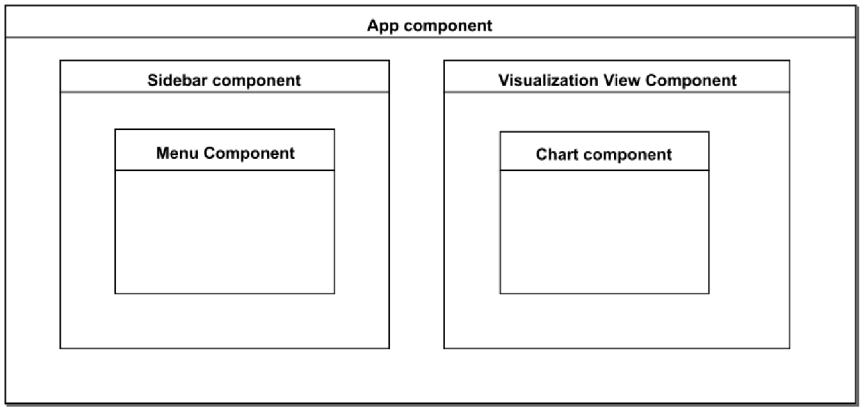

Figure 4.4: Structure of component relationships in the Angular application

In most of the application information flows from parent components to their children. This is partially by design, but we also wanted

to implement a proof-of-concept feature that required the backward flow of communication: The user can click on a player ID on the Y axis of the chart and the menu sidebar opens, revealing a list of all the games that player has taken part in. This is useful when the user wants to compare the performance a player across different games. It required the Player ID to be passed from the chart component, to the visualization view component, to the menu component.

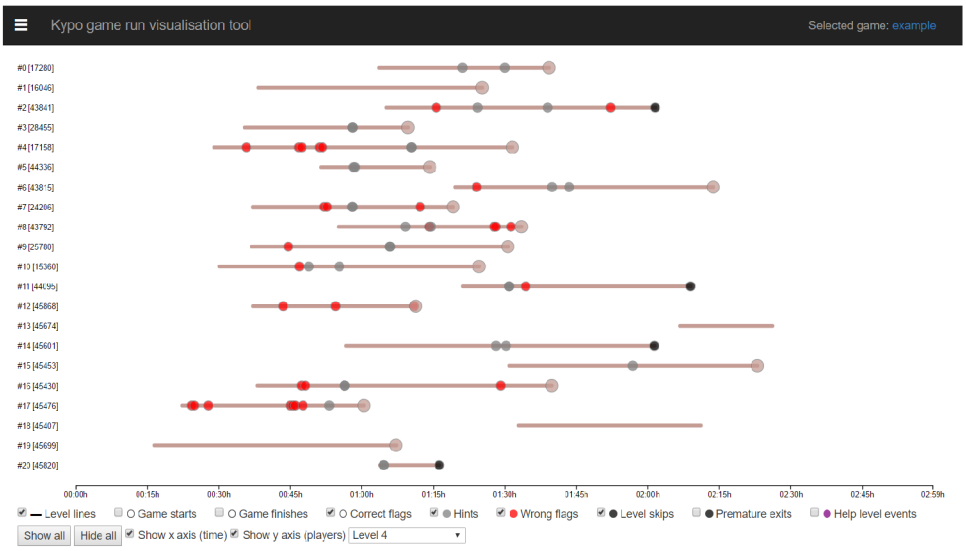

Figure 4.5: Visualization chart embedded in an Angular application

The source code of the Angular application was compiled using the **Angular command line interface**, and deployed using the **Surge** command line web publishing tool.

4.3 Backend server

For the purposes simplicity and elimination of context-switching costs, the back-end server was developed using Node.js, allowing the entire project to be written in JavaScript.

4.3.1 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It enables the development of asynchronous JavaScript applications that run on the server and can be deployed to a wide range of hosting solutions. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world [28].

The asynchronous nature of server applications necessitates the use of callbacks for most of the core functionality of the server. Callbacks are functions passed as parameters to asynchronous methods, to be performed when the asynchronous method is finished. They are typically used in IO operations that can take an unpredictable amount of time. As result, the program does not have to halt and wait for a database query to finish - it an continue executing the rest of its code with the knowledge that when the database query is done, it will execute the callback function that was passed to it [29].

```
// synchronous program
var data = fetchData ();
processData(data);

// eqivalent asynchronyous program using callbacks
fetchData(function (err, data) {
if (err) {
      console.log("An error has occured.");
    }
    processData(data);
});
```

Express.js is a minimal and flexible Node.js web application framework. Its collection of HTTP utility methods and middleware provides a thin layer of abstraction that greatly speeds up the development of APIs.

```
// example of an API endpoint definition using Express.js
app.get('/events', function(req, res) {
```

```
    console.log("Request body: " + JSON.stringify(req.body));
    database.getEvents(req,res);
});
```

The `database.getEvents()` method in the example above translates the HTTP request into a SQL query, runs it against the database and sends the result in an HTTP response.

A common problem when developing a Web API service is that serving requests outside the current domain is not natively supported in all web browsers. This is solved by enabling Cross-Origin Resource sharing, which allows the server to treat all request as same-domain requests. With Express.js, this can be achieved with a single line of code, instructing the server to use the appropriate **CORS** Express middleware package.

```
var cors = require('cors');
...
app.use(cors());
```

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process but reads and writes data directly to ordinary disk files. A complete SQL database with multiple tables is contained in a single disk file. The database file format is cross-platform - it can be freely copied between 32-bit and 64-bit systems or between big-endian and little-endian architectures [30].

Because of its lightweight size and simplicity of use, SQLite was chosen as the database for the back-end server. The SQL queries are abstracted from the main router code in a separate Node module.

4.3.2 RESTful API

REST stands for Representational State Transfer. It is an architecture style for designing networked applications, which relies on a stateless, client-server communications protocol (usually HTTP)[2]. Separation of concerns is the principle behind the client-server constraint. Its goal is to improve portability and scalability by separating the user interface

concerns from the data storage concerns [31]. Each request from any client contains all the information necessary to respond to the request (the stateless constraint).

In a REST Web service, requests made to a resource's URI trigger a response - most commonly in a JSON format. Using HTTP, request types are based on HTTP verbs: GET, POST, PUT, DELETE and so on. They can contain URL and query parameters to specify details of the request.

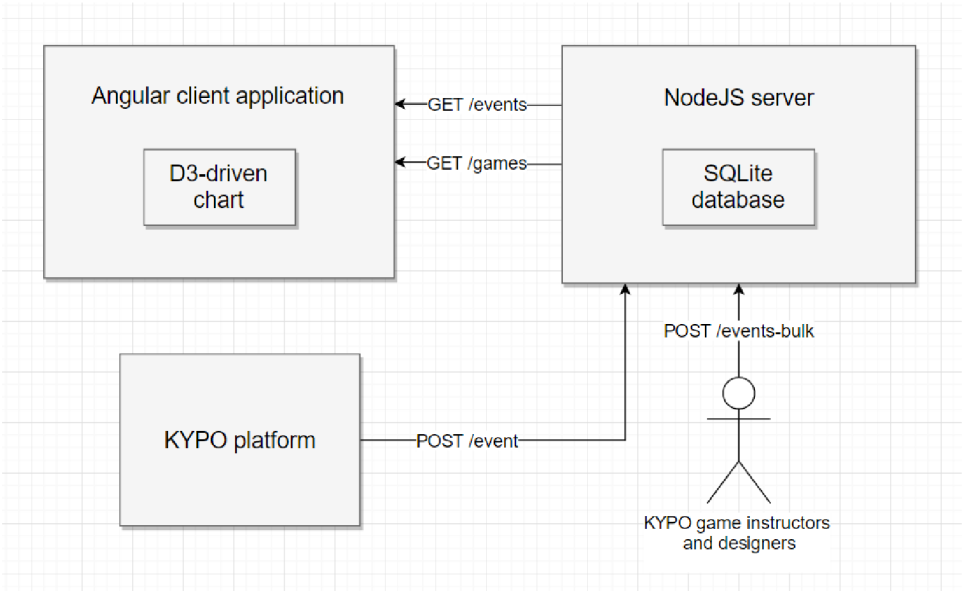

Figure 4.6: An implementation architecture diagram showing how different parts of the solution communicate with each other and the KYPO platform trough REST API.

The KYPO visualization API exposes 2 POST endpoints for saving events and 2 GET endpoints for retrieving events and retrieving game IDs. The communication use case of each endpoint is depicted in Figure 4.6. The complete reference for the API is documented using Swagger.

2. The exact definition of the term 'REST' slowly eroded and the label is now commonly commonly applied even to APIs that do not completely adhere to all the REST principles originally defined by Fielding in 2000.[32]

The Swagger specification is a definition format used to describe RESTful APIs. The files describing the API using the Swagger specification are represented as JSON objects, describing the API's endpoints, request parameters, response objects etc. YAML (a more human readable superset of JSON) can also be used. These files can then be utilized by other tools to generate API documentation, clients or testing tools in various programming languages [33].

The API documentation of the backend server in Swagger is available at `https://app.swaggerhub.com/api/JuroUhlar/kypo-backend/1.0.0` and included in the thesis archive.

4.3.3 Deployment

For the purposes of development and testing, the Node.js application was deployed to Redhat Openshift and is available at `kypo2-uhlar.rhcloud.com`

The application can also be deployed in a virtual environment using **Vagrant**. Vagrant is a tool for building and managing virtual machine environments, that can be used to isolate development dependencies and their configurations. This is especially useful when working in a larger team - Vagrant ensures everyones development environment is exactly the same and its creation is automated [34].

The environment is defined using a single declarative Vagrantfile and can be created using a single `vagrant up` command. Vagrant instantiates the virtual machine on top of Virtualbox and runs a provisioning script that installs all the dependencies. The virtual environment shares selected folders and forwards selected ports to the host machine. This allows developers to edit files on the host machine, run them in the virtual environment and display the output back on the host machine.

5 Evaluation

To evaluate the visualization prototype, a user testing experiment was designed. The goal was to test the usability of the tool and collect feedback on how it should be improved and developed further. We wanted the experiment to reflect real world usage of the tool as closely as possible, while collecting maximum possible amount of useful and actionable data.

5.1 Experiment design

The experiment was designed as a group session lasting approximately 60 minutes. Participation was limited to people who have direct experience with KYPO security games - either designers of the game or facilitators running the exercises themselves.

At first, the participants are introduced to the motivation behind the tool and shown the raw data that it visualizes.

Once provided with the basic context, they are introduced to the tool showing an example dataset and instructed to familiarize themselves with it without help or further instructions. The goal is to see how intuitive the visualization and its user interface is, and what the testers find confusing when guided only by the interface design itself and their intuition. Testers are then given the opportunity to ask questions and clarify their understanding of how the tool works.

After making sure the testers know how to use the tool, they are presented with a series of tasks, each one accompanied by its own dataset. Testers complete the tasks independently but can ask questions any time they need.

- Task 1: Users are shown a dataset corresponding to events data available 35 minutes into the run of the game. They are asked to identify up to three players who most need their assistance.

- Task 2: Identical to task 1, except users are shown a dataset corresponding to data available 60 minutes into the game. Again, they must identify the players who most need their help.

- Task 3: Users are shown a complete dataset of the entire game. They are asked to identify at least 2 players who match 2 types of behavioral archetypes:
  - Players who didn't have enough knowledge and experience to complete the game.
  - Players who tried to learn as much as possible during the game.

The first two tasks simulate the first use case of the tool - the work of the game instructor whose job is to help struggling players get through the game. The third task simulates the second use case for the tool - the work of the game designer, whose job is to analyze previous games to identify and explain patterns in player behavior.

After completing the tasks, testers are presented with a short questionnaire asking for their individual feedback. They rank the usefulness of the tool for each use case on a numerical scale. They are also asked to point out any flaws, usability problems or bugs they have discovered during the experiment and describe what features they were missing, that would make their work easier in the future.

The final part of the experiment is a collective discussion about visualizing KYPO games in more general terms. Users are asked to discuss the biggest problems of the current state of affairs and brainstorm suggestions for future development.

Some overlap between the points raised during the discussion and the feedback provided in individual questionnaires is expected. We have nevertheless decided to use both forms of collecting feedback, as each one has its own set of advantages and disadvantages. Questionnaires are filled out by each participant individually, at their own pace and immediately after completing the tasks to best capture the users' raw and immediate perception of the tool uninfluenced by the opinions of other participants. On the other hand, discussion allows the participants to react to each other, confirm or oppose each others' opinions and combine their ideas together into more refined concepts and suggestions.

The entire session is videotaped to avoid the need for note-taking and allow us to focus on the discussion itself. The application is also equipped with a session-recording script, that tracks the users' mouse

movement and clicks. This allowed us to observe the testers' behavior as if looking over their shoulder.

5.2 Tested hypotheses

The experiment was designed to either confirm or disprove the following hypotheses:

- The visualization is intuitive enough, that a person familiar with the structure of the underlying data can use it effectively even if they see it for the first time. All the features of the tool can be discovered and understood by simply clicking through the user interface. If one or more testers are unable to complete the tasks of the experiment without detailed instructions, this hypothesis is proven false.

- When completing tasks 1 and 2, there would be a significant agreement between testers as to which players most need their help - a small subset of players would the receive the vast majority of the testers' votes. If the votes are distributed evenly in a random fashion, this hypothesis is proven false.

- When completing task 3, there would be a significant agreement between testers as to which players lacked the required knowledge and which ones tried to maximize their learning. If the votes are distributed evenly in a random fashion, this hypothesis is proven false.

5.3 User testing results

The user testing session was attended by 7 participants - all former facilitators of KYPO security games and employees of Institute of Computer Science at Masaryk University. One of the participants also had experience conducting postmortem analyses of KYPO games and has used the visualization tool before.

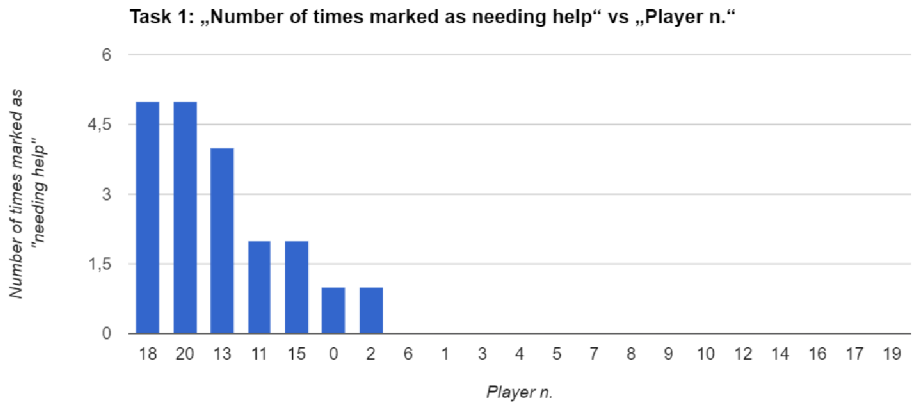

Figure 5.1: Task 1 - answers

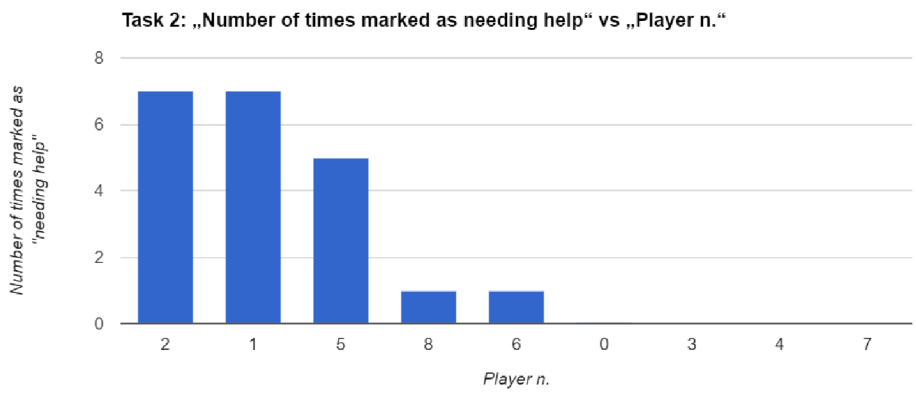

Figure 5.2: Task 2 - answers

5.3.1 Tasks evaluation

Figure 5.1 shows the number of times each player was identified as needing help by our testers. Two of the 21 players were selected by 5 of the 7 testers, one selected by 4 out of 7 testers. Four players received one or two votes, with the remaining 14 players receiving none. This confirms the hypothesis that the testers would use the same heuristics

to identify players in need of help, interpret the data similarly and submit similar results.

Figure 5.2 shows the results for the second task, where testers had only 9 players to choose from. We observe that the agreement between testers in this case is stronger. Two of the 9 players were identified as needing help by all the testers. One player was identified by 5 of the 7 testers, with two other players receiving one vote each. The stronger overlap between answers can be explained by the reduced number of choices and also by more data being available to testers, or some combination of the two. From the data, it is unclear which factor was more influential than the other.

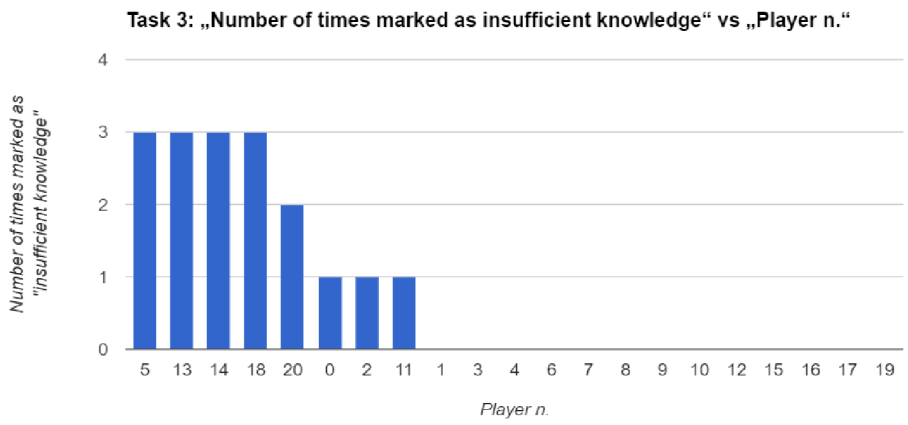

Figure 5.3: Task 3 - answers

Figure 5.3 shows the number of times each player was identified as having insufficient knowledge for the game. Here, the answers of the testers are again more spread out, possibly because the task is more subjective and open to interpretation.

The same pattern can be seen in Figure 5.4 showing the number of time each player was identified as trying to maximize his learning.

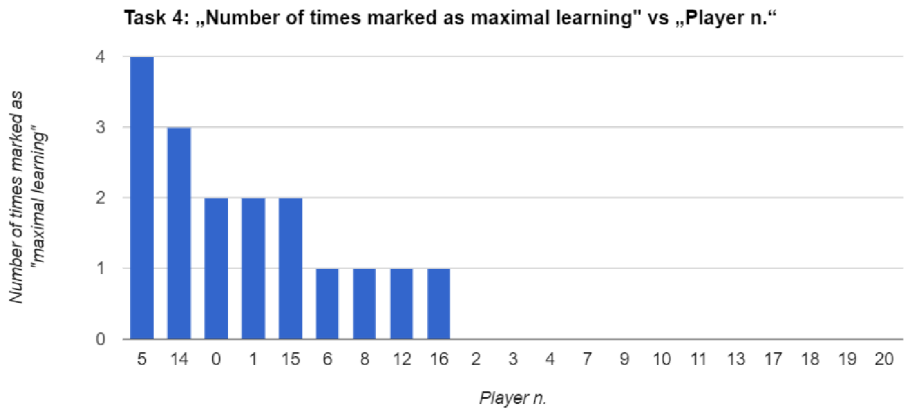

Figure 5.4: Task 4 - answers

5.3.2 Tool usefulness ranking

When asked to rank the tool according to its usefulness at providing situational overview during the game on a scale from 1 to 5 (1 being the worst, 5 being the best), the testers gave the visualization an average score of 4.4 out of 5.

When asked to rank the tool according to its usefulness at aiding post-game data analysis on the same scale, the testers gave the visualization an average score of 4.1 out of 5. Complete breakdown of the answers can be seen in Figure 5.5.

5.3.3 Session recordings

The session recordings were obtained using **Mouseflow** - a visitor recording tool capable of tracking clicks, mouse movement, scrolls, forms, and more. Users were asked to type their name into a field on the welcome screen of the application so that their recordings could be identified by name and cross-referenced with the rest of collected data. Quantitative analysis of the recordings is difficult, but the data can be examined qualitatively. Especially looking at the testers' first interactions with the tool can provide interesting insights and has

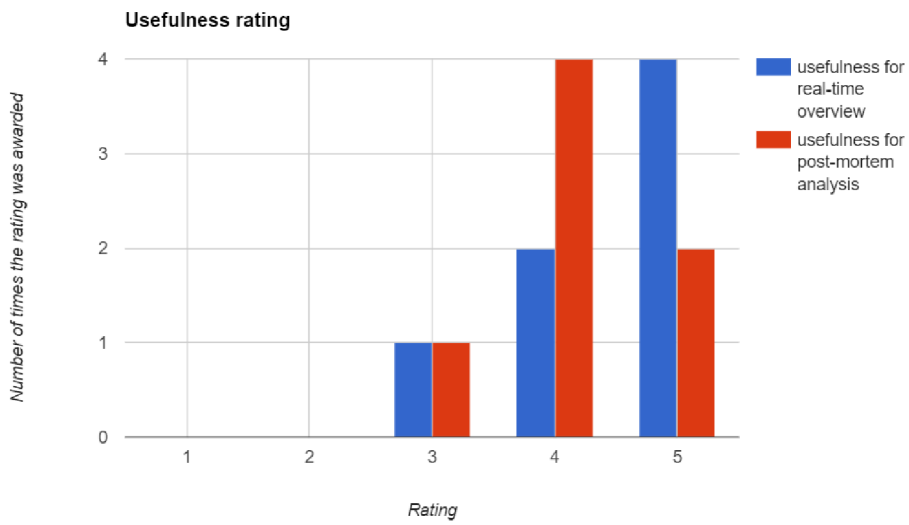

Figure 5.5: Visualization usefulness rating

confirmed the assumption that users can, for the most part, teach themselves how to use the interface without detailed instructions [1].

After opening the example dataset, all recorded testers proceeded to first toggle all the event filters one by one, effectively building a mental list of all the various types of events. After that, all testers experimented with the level selection drop-down, displaying various levels in real-time as well as logical time. All testers intuitively understood that displaying a level in logical time shows them the same data points, only shifted to zero as if the level was started simultaneously by all players.

On one of the recordings, we can observe the user experience a brief moment of confusion when seeing the *task1* dataset. Then he immediately resolves it by examining the X axis of the visualization

1. Only four session recordings were obtained, instead of all seven. This was caused by three of the participants using anti-tracking extensions in their browsers, such as Ghostery or uBlock Origin. These tools can disable a variety of visitor tracking and recording tools, including Mouseflow. This oversight on the author's part serves as a lesson for future experiments: Always ask participants to disable their privacy tools at the beginning of the exercise.

and realizing that the displayed data only spans the first 30 minutes of the game. Moments like this confirm the assumption that our design works and the user interface is, for the most part, self-explanatory enough for users to understand it by themselves.

The recordings allow for a real-time observation of many of the problems mentioned by users in their feedback. Two users were observed spending minutes of their time trying to click on the event dots to no effect, while the other two recorded testers discovered the correct way to display tool-tips immediately. Users were also recorded unsuccessfully trying to zoom in with the native browser scaling function.

In addition, Mouseflow is capable of turning the recordings into heatmaps of users' clicks and mouse movement, which can then be analyzed. The click heatmap is particularly useful for estimating how frequently users utilize each element of the user interface. From the heatmap generated during the user testing, for example, we can see the toggle check-boxes for X and Y axis are almost never used. Therefore, on the next user interface redesign, we will consider giving the toggle buttons a less prominent location or hiding them completely.

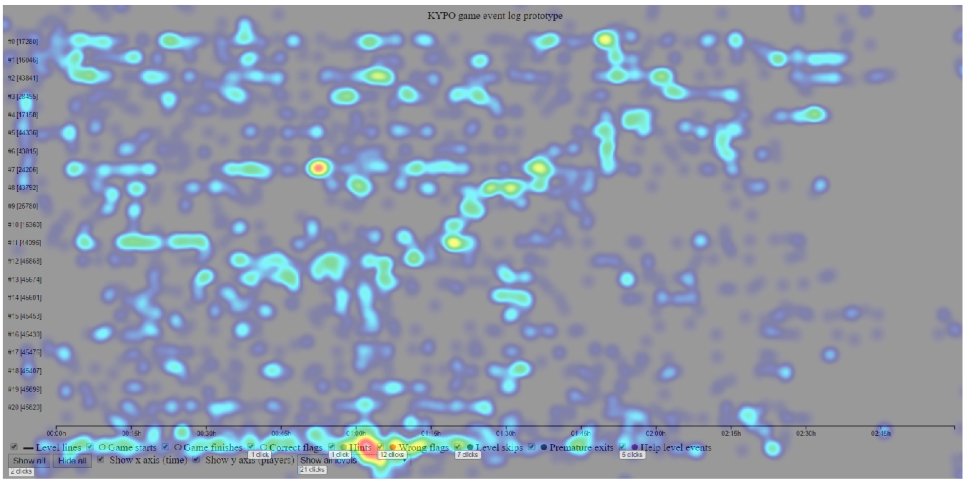

Figure 5.6: Heatmap of the user interface showing mouse movement

5.3.4 Revealed issues

The experiment helped identify a number of issues with the tool. Combining the questionnaire answers, output from the discussion and our own observations made during the experiment itself, we have compiled the following list of usability problems, ordered by severity:

1. Colors representing individual levels were too similar and got mixed up easily.

   At the time, the colors were assigned automatically from a D3 color scale. The problem was fixed by explicitly defining the range of level colors and deliberately choosing more distinct ones.

2. There was no line representing the duration of unfinished levels.

   This initially made sense from an implementation point of view: Level lines were drawn between the beginning and end of each level, so if the end of the level is not present, no line is drawn. This solution, however, turned out to be a source of massive confusion for the users. They were not sure how to interpret events hovering in the air with no line beneath them and asked for clarification multiple times. To amend this, the rendering algorithm was updated to draw a line from the beginning of the level to the last event of the level present in the dataset. It is drawn in a dashed pattern to signify that the level is not yet finished.

3. When a user clicked on a circle, the tool-tip with information stayed hidden.

   This is an example of usability problem the author has never discovered while testing the tool himself. The complete information about an event was stored in a native **title** attribute of the circle that represents it. The tool-tip was typically shown when a user hovered with the mouse cursor above the circle. The visualization simply utilized native browser behavior. The problem was that if a user clicked on the circle, the tool-tip was never shown. Some of our testers automatically started clicking

on the circles and did not realize they could access the information by hovering their mouse above them. The solution was to implement a custom tool-tip system that displays tool-tips without delay and does not hide them when the circle is clicked.

4. Placement of the individual filter check-boxes was confusing.

   At the time, the individual check-boxes for each type of events and their labels were located too closely to each other. The users complained about being unsure which check-box does what. This was straightforwardly solved by adding a right margin to each check-box label.

5. The X axis was missing the last tick.

   The tick showing the time of the end of the game was not shown on the X axis. This has been fixed by explicitly passing the time of the last event as an argument to the function that generates the axis.

6. The drop down menu does not work as expected on MacOS.

   The drop down menu for selecting levels is located at the bottom of the interface, so it should automatically open upward. And it does, except on MacOS, where the drop-down menu opens downward and goes off-screen. This could only be replicated using the Chrome web browser on MacOS, which is a rare edge case. Nevertheless, the issue will have to be investigated further.

5.3.5 Future development suggestions

Combining the responses from the questionnaire with the output from the brainstorming discussion we have collected a list of ideas, feature requests and improvement suggestions which will serve as a roadmap for future development of the visualization tool. Here is the list, loosely ordered by the expected impact of the requested feature on the usefulness of the tool:

1. Visualize the expected duration of each level to allow facilitators to more easily identify players who are lagging behind.

2. Allow users to add annotations or notes into the dataset.

   During the game, there are many factors affecting the players performance that are not reflected in the collected data. For example, technical problems with the platform, misunderstandings, connectivity issues etc. If these events are not recorded on the spot, they can get lost, which leads to an imprecise and possibly misleading record of the game. Allowing users to quickly and easily add notes to specific players at specific points during the run of the game would significantly improve the precision of collected data.

3. Show a table with basic aggregated statistics of the game: Number of each type of events for each level, per player.

4. Automatically assign labels to players based on pre-defined conditions.

   An instructor of the game should be able to look at the data a identify a player who needs help immediately without having to spend minutes of his time analyzing the data. For example, the tool could automatically highlight players who submitted 3 incorrect flags in a row or a level takes them much longer than expected.

5. Order players from top to bottom based on their score.

6. Implement a zoom function

   One of the problems of the implementation is that the circles representing the events sometimes overlap, forcing the user to hunt for pixels when trying to display the event detail. This is especially problematic when there are two events of the same type right next to each other. Allowing users to zoom in and explore the data more comfortably would noticeably improve the usability of the tool.

7. Allow users to hide data from selected players.

   Implement an additional filtering option users can use to toggle shown data for each individual player.

8. Show information about the maximum number of hints available during each level.
9. Show a summary of the last 10 events.
10. Show a legend for colors associated with each level.
11. Show a grid extending the X axis for easier reading of time.
12. Allow users to export the visualization as a SVG file.
13. Allow users to save dataset view with its filter configuration and share it with colleagues via a link.

6 Conclusion

Creating this thesis pushed me to familiarize myself with the field of cybersecurity games and learn how to use a variety of modern development tools, such as D3, Node.js, and Angular.

First, we analyzed the use cases and requirements of the visualization, then used those insights to make the appropriate architectural design decisions and choose the right development tools. The most important contribution is the creation of an application that is able to visualize the run of KYPO security game in an intuitive and interactive way. Thanks to its modular nature, it can be seamlessly incorporated into the existing KYPO platform and immediately used to provide insights into existing as well as new game events data. The implementation was tested by instructors of KYPO security games, who provided valuable feedback as well as a list of potential improvements and feature requests which serve as a roadmap for future development (described in Chapter 5).

In general, the testers regarded the work as useful and a step in the right direction. The tool has been used by Mgr. Valdemar Švábenský in his diploma thesis [7] and by my advisor RNDr. Jan Vykopal, Ph.D.

Hopefully, my work can serve as a foundation for future visualization needs of the KYPO project. I plan to continue developing the tool further in cooperation with the KYPO team as a subject of my diploma thesis.

Bibliography

1. *Mitigating the Cybersecurity Skills Shortage* [online]. 2015 [visited on 2017-05-28]. Available from: `http://www.cisco.com/c/dam/en/us/products/collateral/security/cybersecurity-talent.pdf`.
2. DISHMAN, Lydia. *The Most Critical Skills Gap: Cybersecurity* [online]. 2016 [visited on 2017-05-28]. Available from: `https://www.fastcompany.com/3062210/the-most-critical-skills-gap-cybersecurity`.
3. TOBEY, David; PUSEY, Portia; BURLEY, Diana. Engaging Learners in Cybersecurity Careers: Lessons from the Launch of the National Cyber League. *ACM Inroads*. 2014, vol. 5, no. 1, pp. 53–56. ISSN 2153-2184. Available from DOI: `10.1145/2568195.2568213`.
4. VYKOPAL, Jan; BARTÁK, Miloš. On the design of security games: from frustrating to engaging learning. In: *2016 USENIX Workshop on Advances in Security Education (ASE 16)*. 2016.
5. PUSEY, Portia; TOBEY, David; SOULE, Ralph. An Argument for Game Balance: Improving Student Engagement by Matching Difficulty Level with Learner Readiness. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, 2014. Available also from: `https://www.usenix.org/conference/3gse14/summit-program/presentation/pusey`.
6. WOUTERS, Pieter; VAN NIMWEGEN, Christof; VAN OOSTENDORP, Herre; VAN DER SPEK, Erik. A meta-analysis of the cognitive and motivational effects of serious games. *Journal of Educational Psychology*. 2013, vol. 105, no. 2, pp. 249.
7. ŠVÁBENSKÝ, Valdemar. *Prerequisite testing of cybersecurity skills [online]*. [cit. 2017-05-28]. Available also from: `http://is.muni.cz/th/395868/fi_m/`. Master's thesis. Masaryk University, Faculty of Informatics, Brno. Supervised by Jan VYKOPAL.
8. ČELEDA, Pavel; ČEGAN, Jakub; VYKOPAL, Jan; TOVARŇÁK, Daniel. KYPO – A Platform for Cyber Defence Exercises. In: *STO-MP-MSG-133: M&S Support to Operational Tasks Including War Gaming, Logistics, Cyber Defence* [[online]]. Munich (Germany): NATO Science and Technology Organization, 2015, p. 12. ISBN 978-92-837-2020-1.

Available also from: https://www.cso.nato.int/Pubs/rdp.asp?RDP=STO-MP-MSG-133.

9. *The KYPO - Cyber Exercise & Research Platform* [online] [visited on 2017-05-28]. Available from: https://www.kypo.cz/.
10. BARTÁK, Miloš. *Bezpečnostní hry pro rozdílné úrovně dovedností [online]*. 2016 [cit. 2017-05-29]. Available also from: http://is.muni.cz/th/396568/fi_m/. Master's thesis. Masaryk University, Faculty of Informatics, Brno. Supervised by Jan VYKOPAL.
11. CARD, Stuart; MACKINLAY, Jock; SHNEIDERMAN, Ben. *Information Visualisation-using vision to think*. Morgan Kaufmann San Francisco, 1999.
12. HYERLE, David. *A Field Guide to Using Visual Tools*. Association for Supervision and Curriculum Development, 2000. ISBN 9780871203670. Available also from: https://books.google.cz/books?id=LRhKAAAAYAAJ.
13. KAHNEMAN, Daniel. Thinking, fast and slow. In: Farrar Straus Giroux, 2011, pp. 19–22.
14. BRODSKI, Alla; PAASCH, Georg-Friedrich; HELBLING, Saskia; WIBRAL, Michael. The Faces of Predictive Coding. *Journal of Neuroscience*. 2015, vol. 35, no. 24, pp. 8997–9006. ISSN 0270-6474. Available from DOI: 10.1523/JNEUROSCI.1529-14.2015.
15. KEIM, Daniel; MANSMANN, Florian; SCHNEIDEWIND, Jorn; ZIEGLER, Hans. Challenges in Visual Data Analysis. In: *Tenth International Conference on Information Visualisation (IV'06)*. 2006, pp. 9–16. ISSN 1550-6037. Available from DOI: 10.1109/IV.2006.31.
16. *Scatter / Bubble Chart* [online] [visited on 2017-05-28]. Available from: http://nvd3.org/examples/scatter.html.
17. CONTI, Greg. *Security data visualization: graphical techniques for network analysis*. No Starch Press, 2007.
18. MARTY, Raffael. *Applied security visualization*. Addison-Wesley Upper Saddle River, 2009.
19. *SecViz - Security Visualization* [online] [visited on 2017-05-28]. Available from: http://secviz.org/.

20. AGUINAGA, Jose. *How it feels to learn JavaScript in 2016* [online]. 2016 [visited on 2017-05-27]. Available from: https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f.

21. *ECMAScript® 2015 Language Specification* [online] [visited on 2017-05-28]. Available from: https://www.ecma-international.org/ecma-262/6.0/.

22. *NPM vs. Bower vs. Browserify vs. Gulp vs. Grunt vs. Webpack* [online] [visited on 2017-05-28]. Available from: https://stackoverflow.com/questions/35062852/npm-vs-bower-vs-browserify-vs-gulp-vs-grunt-vs-webpack.

23. ELLIOT, Eric. *Angular 2 vs React: The Ultimate Dance Off* [online] [visited on 2017-05-28]. Available from: https://medium.com/javascript-scene/angular-2-vs-react-the-ultimate-dance-off-60e7dfbc379c.

24. *Data Driven Documents* [online] [visited on 2017-05-28]. Available from: https://d3js.org/.

25. MURRAY, Scott. *Binding data* [online]. 2015 [visited on 2017-05-28]. Available from: http://alignedleft.com/tutorials/d3/binding-data.

26. BOSTOCK, Mike. *Let's Make a Bar Chart* [online]. 2013 [visited on 2017-05-28]. Available from: https://bost.ocks.org/mike/bar/.

27. CHEUNG, Eugene. *Angular sidebar* [online] [visited on 2017-05-28]. Available from: https://github.com/arkon/ng-sidebar.

28. *Nodejs.org* [online] [visited on 2017-05-28]. Available from: https://nodejs.org/en/about/.

29. OGDEN, Max. *Callback hell* [online] [visited on 2017-05-28]. Available from: http://callbackhell.com/.

30. *About SQLite* [online] [visited on 2017-05-28]. Available from: https://www.sqlite.org/about.html.

31. FIELDING, Roy T; TAYLOR, Richard N. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

32. HOUSE, Cory. *How RESTful is Your API?* [online]. 2012 [visited on 2017-05-27]. Available from: http://www.bitnative.com/2012/08/26/how-restful-is-your-api/.

33. *OpenAPI Specification* [online] [visited on 2017-05-27]. Available from: `http://swagger.io/specification/`.

34. STURGEON, Philip. *Vagrant: What, Why, and How* [online]. 2012 [visited on 2017-05-28]. Available from: `https://code.tutsplus.com/tutorials/vagrant-what-why-and-how--net-26500`.

A Complete wording of the user testing questionnaire

Task 1

Dataset task1 displays data about player activity 35 minutes into the game. (The game has 6 levels and is expected to last two and a half hours).

- **Which players most the need help from the instructor at this moment?**

Write at most 3 serial numbers of players, separated by a comma. Order players based on priority (the first one needs you the most).

Task 2

Dataset task1 displays data about player activity 60 minutes into the game. (The game has 6 levels and is expected to last two and a half hours).

- **Which players most the need help from the instructor at this moment?**

Write at most 3 serial numbers of players, separated by a comma. Order players based on priority (the first one needs you the most).

Task 3

Dataset task3 contains complete data about the whole run of the game.

- **Which players lacked the knowledge necessary to play the game?**

Identify at least two.

- **Which players tried to learn as much as possible?**

Identify at least two.

Evaluation

Would the tool be useful for providing overview of player activity during the game? (Rate on a scale from 1 to 5)

- 1 (useless)
- 2
- 3
- 4
- 5 (very useful)

Space for additional comments:

Would the tool be useful for postmortem analysis of the game? (Rate on a scale from 1 to 5)

- 1 (useless)
- 2
- 3
- 4
- 5 (very useful)

Space for additional comments:

How do you like the user interface? Was there something confusing or counter-intuitive?

Space for comments:

What features would you add to the visualization tool?

Space for comments:

B Content of the thesis archive

The thesis archive available at `https://is.muni.cz/th/422160/fi_m/` includes experiment data and source code organized in the following folder structure:

- `data`

  Game events datasets used during development and for the purposes of user testing in the JSON format.

- `doc`

  The text of the thesis in PDF and TEX format.

- `src`

  All of the source code, available under an MIT license.

  - `KypoViz-client`: Source code of the client web application
    - `src`: Source code in Typescript.
    - `dist`: The code of the application transpiled into vanilla JavaScript, bundled, minified and ready for distribution. Open `index.html` to launch the application.
  - `KypoViz-server`: Source code of the backend server.
  - `Vagrantfile`: Configuration of the vagrant virtual development environment.
  - `vagrant-scripts`: Scripts used in the vagrant virtual environment.

- `readme.txt`: A file explaining the folder structure above

C Instructions for running the application locally

To run the application locally, there are 2 options:

- Run the application directly on your machine (requires Node).
- Run the application inside a virtual development environment (requires Vagrant and VirtualBox).

Running the app directly

1. If you do not have it yet, download and install the latest version of NodeJS (including npm) at `https://nodejs.org/en/`.
2. Navigate to the `src/KypoViz-server` folder using the Command Prompt / Shell.
3. Run `npm install` to install dependencies.
4. Run `node index.js` to start the server.
5. Navigate to the `src/KypoViz-client/dist` folder and open `index.html` in your browser.

Running the app using Vagrant

1. Download and install the latest version of VirtualBox: `https://www.virtualbox.org/`
2. Download and install the latest version of Vagrant: `https://www.vagrantup.com/`
3. Navigate to the `src` folder using a **ssh-enabled console** (Cmder, Git bash...) **with administrator rights** and run `vagrant up` to create the virtual environment (it will take a long time).
4. Run `vagrant ssh` to log into the virtual machine
5. Run `runall` to start the application.

6. Open `localhost:4201` in a browser on the host machine to open the application.

Note: This has only been tested on Windows 10, with Vagrant 1.9.5. If you encounter any problems, please contact the author or run the application using the first method.

The (identical) application is also available online at `http://kypo-viz.surge.sh/`. The URL of the API server is `http://kypo2-uhlar.rhcloud.com/`.