Interactive Dashboard for Robo^3 Game

Software Development Report

Prepared by:

Nicolás Leiva Büchi

Politecnico di Milano

August 2019

Table of Contents

1. Project Description

This project's aim is the design, development and implementation of a visualization system as an interactive WebApp dashboard for the game Robo^3, which is an applied game for teaching introductory programming.

All the documentation and code for this project is available in the following repository: https://github.com/nicolasleivab/Interactive-Dashboard

As well as a live demo: https://robo3-dashboard.netlify.com

1.1. Objectives

The goal of the interactive dashboard is to collect and process the data from each student performance from a remote game log and display it to two types of users: students and the tutor in charge. In order to achieve this, several procedures are needed:

- **Requirements Specification** - What specific problems should the system solve? - This depends on each user needs - How does the user interact with the system in order to make a decision?
- **Architectural Design** - Decompose the problem to be solved into several components (elements of the dashboard)
- **Component Design** - Select the best visualizations and interactions for each component according to the user
- **Implementation** - Actual coding
- **Component Testing** - Debugging
- **User Evaluation** - User testing and feedback
- **Documentation and fixes**

1.2. Technologies

JavaScript

- D3.js is a JavaScript library for manipulating documents based on data. It allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. This library is used for the actual data visualizations and the interactive functionalities.

- jQuery is a JavaScript library that simplifies HTML document traversal and manipulation, event handling, animation, and Ajax being widely supported by a multitude of browsers. This library is mainly used for the implementation and integration of date jQuery range sliders to the visualizations and filters of the tutor dashboard as well as some DOM functions.

- Native XMLHttpRequest (XHR), an object whose methods transfer data between a web browser and a web server, is used to request data from different google sheets using Google Sheets API.

- Jest is a JavaScript testing Framework for performing unitary and integral tests. This framework is therefore used to engineer several tests for the different functionalities of the software for debugging purposes.

HTML, CSS and Bootstrap

- HTML and CSS are used for the web structure and the styling of the graphs.

- Bootstrap is used for the implementation of the grid system as well as some styling.

Git

- Git is a free and open source distributed version control system. This tool is used for the source control of the project.

2. Design

How is this built?

Basically, the data of the game is stored in a google sheet in a predefined format. Then, Native XMLHttpRequest (XHR) object is implemented to get JSON from the google sheet using Google Sheets API. Finally, the data is processed and represented in several charts using D3.js in a Web App. The Web App is built in such a way that allows scalability and uses a modular and responsive design.

Figure 1. Conceptual Workflow Map

2.1. User Requirements

Robo^3 game sends a message to a remote server each time a user performs an action of the following: entering a level, exiting a level, completing a level and failing a level. Some additional information related to the number of instructions, cycles and functions used in each solution is also contained. This allow to store data about the user behavior and progress in the game. With this information, we can come up with several ways of showing the data in charts, such as: number of plays, total playtime, distribution of cycles and/or instructions per level, success probability of each level, among others.

However, in order to determine the best way to display and represent this data, it is necessary to understand the needs and motivations of each of the two different users that will have access to the visualization system. What specific user problems does the system solve? How does the user interact with the system in order to make a decision?

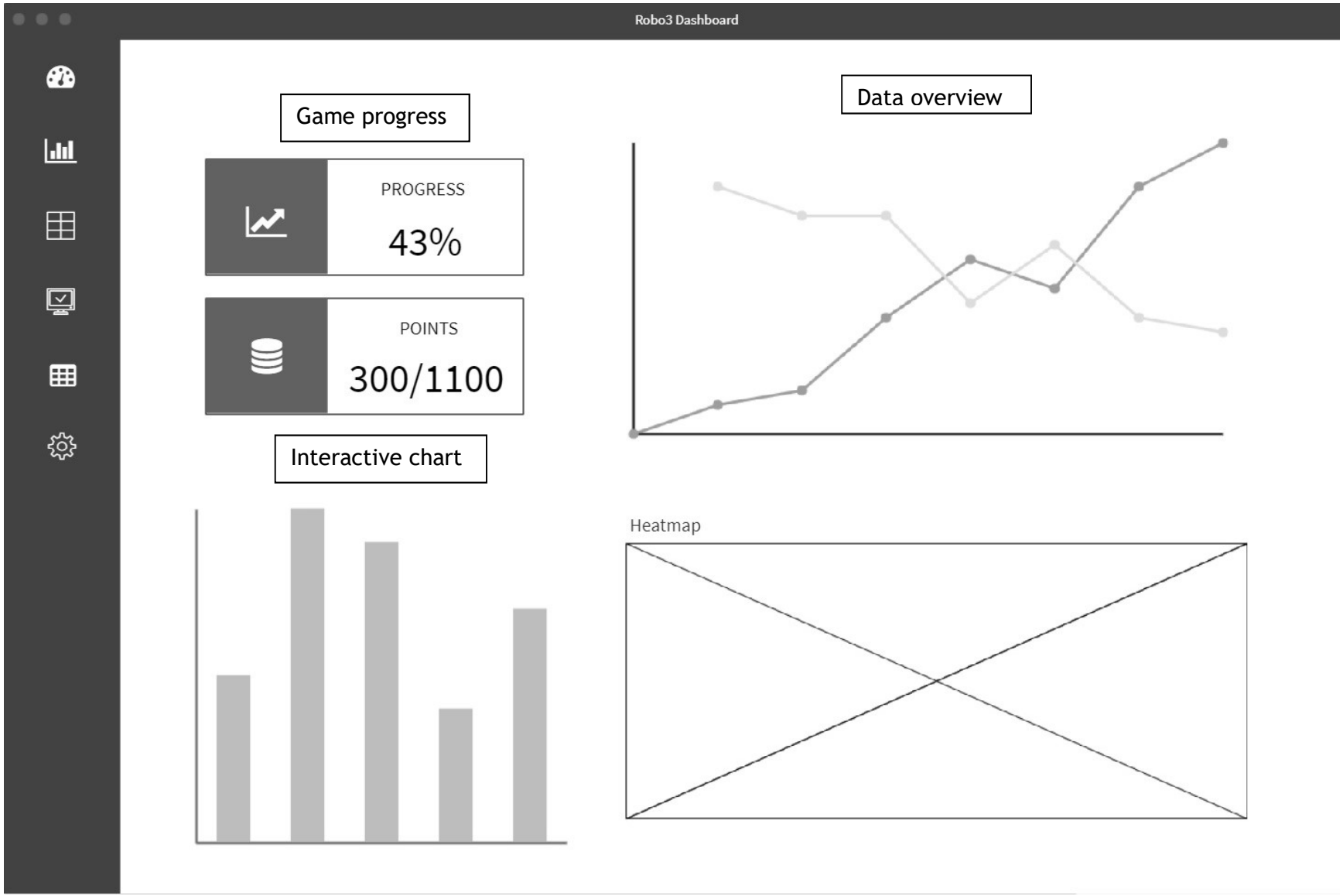
2.1.1. User: Student

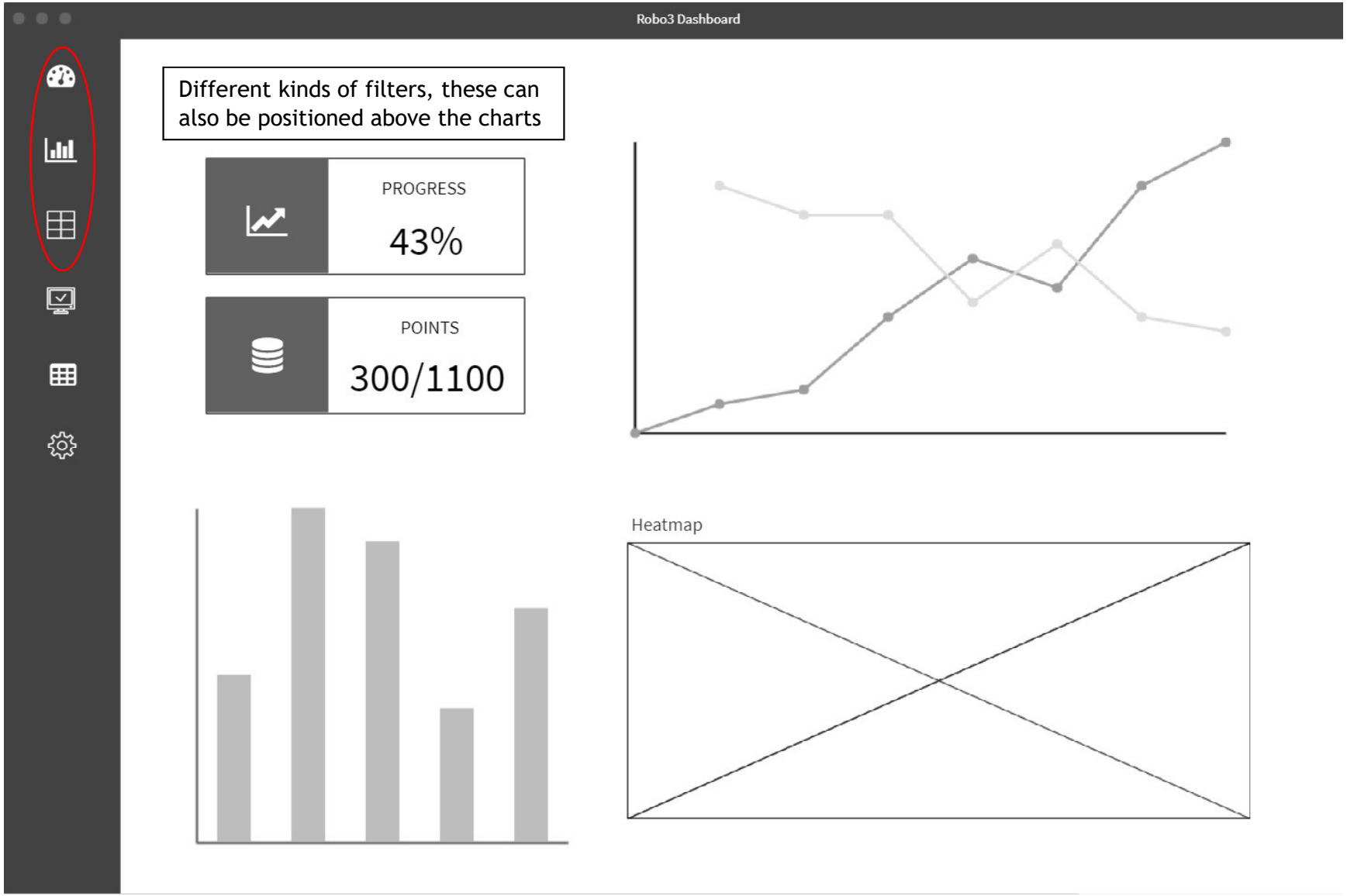
- In general, the student would like to see his personal progress of the game in the dashboard related to the different levels and the total play time.
- Data visualization should be focused on a simple and straight forward way and should aim to engage and motivate the student.
- Some information like success probability of each level should be avoided since it could discourage the student in the levels in which the interaction becomes more like a trial and error.
- A score system can be implemented to motivate the user. This system can assign points depending on the efficiency of the solution for each level. This means that the student can complete the game (100%) but still can replay each level in order to achieve the maximum amount of points by finding an optimal solution. An optimal way to implement this can be through a leaderboard showing the ranking of the top performance players, this can encourage competitiveness between the students which engage users in the game.
- A heat map for each student can be implemented to motivate the user to continue a certain streak and keep track of their daily progress.
- Category filters can be applied to the charts giving the user the possibility to see the data in detail if requested.

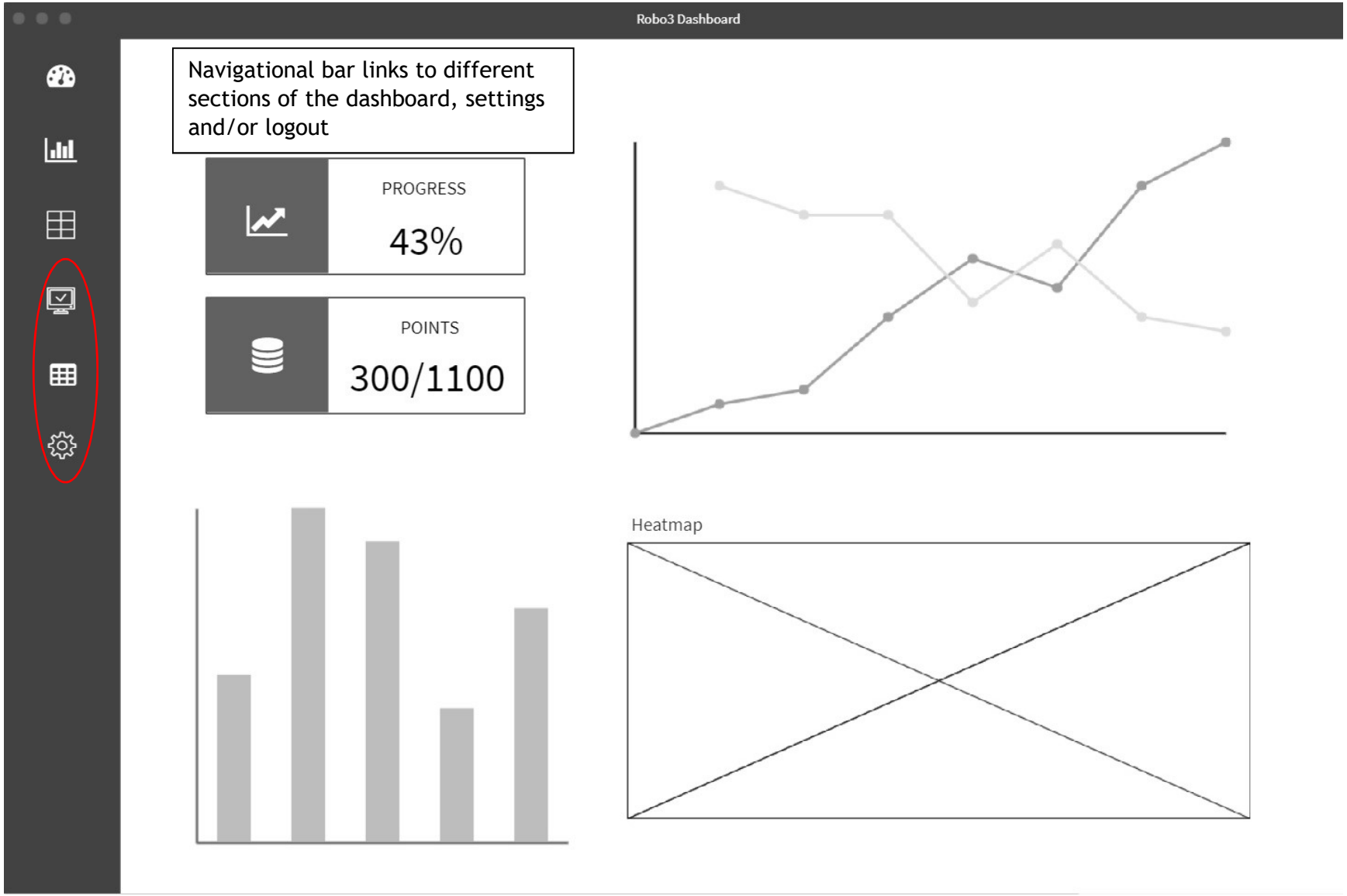
2.1.2. User: Professor/Tutor

- In the case of the Professor/Tutor, the data visualization should include more advanced functionalities in order to take better decisions to evaluate the students.
- First, a clear overview of the whole group progress should be displayed. The distribution of the whole class set of instructions and performance by each level can be shown in a histogram. Filter controls can be applied in order to allow data to be filtered by levels, type of instruction and date. Ideally, a visualization that can show the evolution through time regarding the class performance should be available.
- A bar chart that shows the success of probability can be implemented in order to identify each level difficulty. A date filter can also be integrated if the tutor requires date specific data or an overview of the data through time.
- Finally, a visualization showing the best and average solution for each level can be implemented and integrated with an individual student filter that can help the tutor in the evaluation process.

2.2. Wireframe and Structure




Robo3 Dashboard
Different kinds of filters, these can also be positioned above the charts
PROGRESS
43%
POINTS
300/1100
Heatmap


Robo3 Dashboard
Navigational bar links to different sections of the dashboard, settings and/or logout
PROGRESS
43%
POINTS
300/1100
Heatmap

2.3. Component Design

First, it is necessary to explain the set of variables that are related to the game and are therefore shown in the following visualizations. Each level of the game requires to solve a problem by using a combination of different sets of instructions. The different instructions are Functions, Loops, Movement and Pick or Drop and are meant to act upon different blocks in order to solve each level challenge.

For the dashboard purpose, the different instructions can be selected individually as variables for the different visualizations as well as the whole group under the name of "Instructions". There is another variable called Cycles, which represents the number of iterations the solution does until it successfully completes the challenge in each level. The different solutions are meant to be evaluated in terms of efficiency in the utilization of resources, in other words, the less instructions a solution uses the more efficient it is. The same way works with cycles, a combination of these two optimizations is the focus of performance evaluation for both the student and the tutor.

The different individual visualizations created are shown with their functionalities briefly explained, both from the student and tutor dashboards. These visualizations are showcased from the simplest to the most complex ones that were almost a project by themselves containing their own repositories and documentation.

2.3.1. Student Dashboard

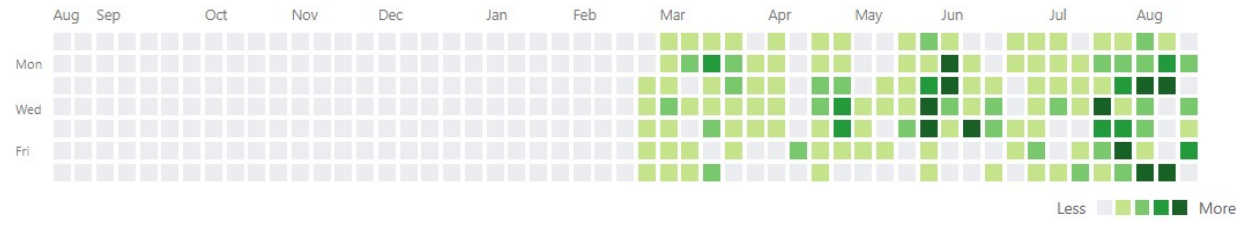
2.3.1.1. Animated Progress Meter

This visualization consists in a donut chart that shows the progress of the student in terms of completed levels, therefore showing a percentage of completion of the game. It contains an initial animation starting from 0 to whatever percentage the student has achieved. The objective of this visualization is to clearly state a level of achievement in terms of succeed levels.

2.3.1.2. Heatmap

A Heatmap consist in a type of visualization of data in the form of a map or cells diagram in which data values are represented as colors. The darker the color the higher the value will be, being the value the number of times a student complete a task each day. As explained before, the objective of this visualization is to engage the student in the game by maintaining streaks and achieving daily goals.

Figure 2. Heatmap



2.3.1.3. Leaderboard

The leaderboard consists in a table that shows the top three students. For this matter, a simple score system was implemented which considers a combination of the best set of instructions and cycles per level for each student. For this visualization, there's a program in charge of calculating the score for each student by making different groups for each level. Then a comparison is made in order to list the students from 1st to 3rd place for each row of the table. The leaderboard aims to motivate the students by inciting competition among students.

Figure 3. Students Leaderboard

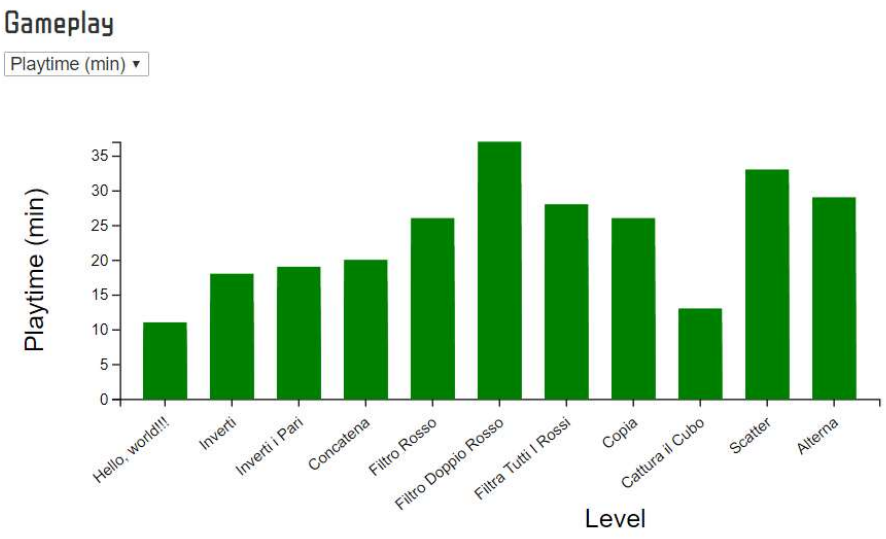
Leaderboard

Position	Player	Score
1	12341234	151
2	10101010	227
3	1111111	270

2.3.1.4. Bar Chart with Dropdown Filter

This visualization aims to show a general overview of the gameplay to each student through a bar chart. The chart shows information about the total amount of time dedicated to each level as well as the total number of times that the student has successfully played each level (Rounds). A dropdown selector is used to allow the user to switch between the different variables.

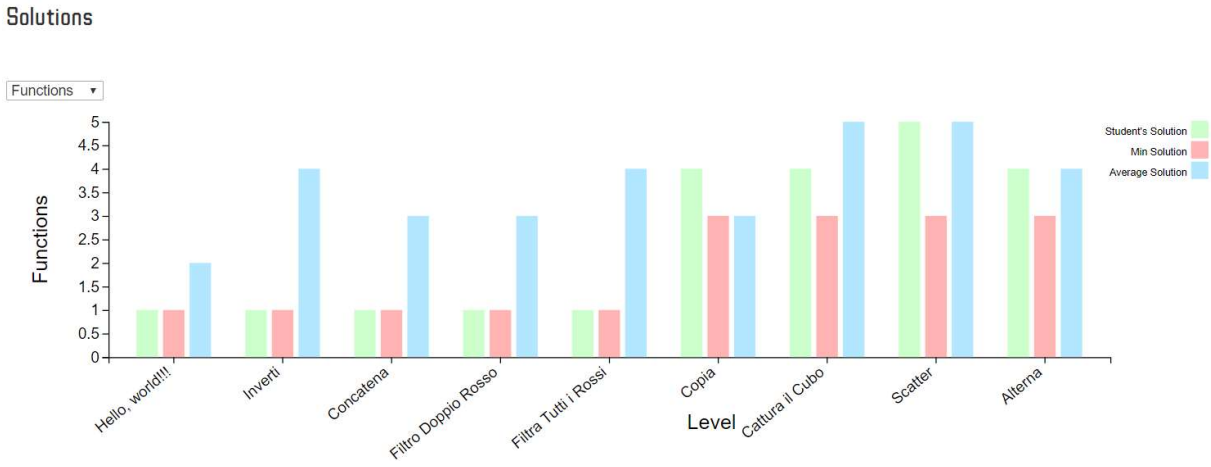
Figure 4. Student Bar Chart



2.3.1.5. Grouped Bar Chart with Dropdown Filter

A grouped bar chart consists in a more complex type of bar chart showing more than one variable in groups. In this case, the best solution for each level that the student has achieved is shown aside with the average and the best solution of the class per level. The objective of this visualization is to motivate the student's competitiveness by showing their performance in contrast with the whole class best and average solution.

Figure 5. Student Grouped Bar Chart



2.3.2. Tutor Dashboard

2.3.2.1. Probability of Success Bar Chart with Date Filter

Regarding the tutor user requirements, this bar chart shows the probability of success of each level for the whole class using the standard formula of failed attempts plus successful attempts divided by the total sample. The purpose of this visualization is to work as an indicator of the difficulty of each level based on their success ratio over time. A date filter is integrated to this graph so the tutor can observe date specific data and see the evolution of the variable over time, which is also an indicator of each class group performance considering that the groups of students change every semester.

2.3.2.2. Grouped Bar Chart with Dropdown and Student ID Filter

The same as the student grouped bar chart, the only difference is that an individual student filter is added. The visualization objective is to show to the tutor each student performance by instruction per level in comparison to the class. Efficiency is evaluated in terms of minimal number of instructions and cycles.

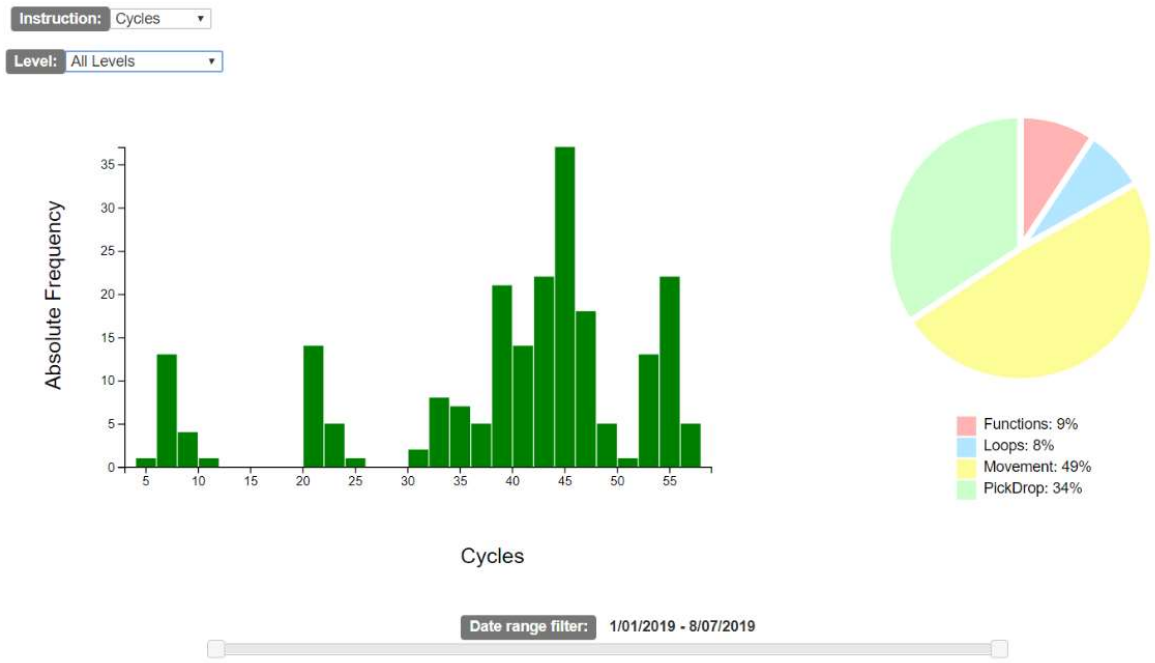
2.3.2.3. Histogram with Double Filter and Date Range Slider

This visualization consists in a histogram that represents the distribution of instructions by category and cycles for each level of the game. The different classes and the bin size of the histogram are selected by default. Since the data will change after the official deployment of the game, a function that let the tutor customize the bin sizes should be implemented and is discussed in the Further Developments section.

The histogram has two dropdown selectors to allow the tutor to filter data by level and type of instruction (rows and columns), an option that shows data distribution for all levels is also added. The purpose of this whole visualization is to give an overview of the class solutions distribution over time, for accomplishing that a date range slider is implemented to allow the tutor to filter data by specific date ranges and see the evolution of the class over time. This also allows to filter the data by different classes since the groups of students change every semester.

A pie chart that shows the percentage of each instruction by level is integrated with the histogram. In order to make the pie chart and labels responsive to the level selection, a correct integration of its updating features in the different layers of data filter must be done. The purpose of this last visualization is to act as a support and show a clearer picture of the distribution of instructions and cycles of the whole class to the tutor.

Figure 6. Instructions Distribution per Level



3. Implementation

This software contains roughly 2300 lines of code for the implementation of the different visualizations, data fetching, data processing and manipulation, data filtering, interactive functionalities, structure and styling. For obvious reasons, only the most important features are shown in this section.

First, it is important to clarify that D3js makes use of SVG in order to display data and visualizations in the client browser. It is necessary to briefly explain how this work in order to understand the implementation code of the visualizations in this project.

SVG stands for Scalable Vector Graphics. It is an XML language and file format, which allows you to code two-dimensional graphics that scale and can be manipulated via CSS or JavaScript. SVG can be directly included in any html file or with JavaScript, being the latter one used in this project. Roughly speaking, all SVG graphics are plotted on a coordinate system of at least an x and y axis in a sort of virtual canvas.

Using D3js it is possible to create and manipulate SVG graphics by different methods and functions. The most relevant functions are shown below

```
<svg id="myCanvas" width="350" height="70"></svg> //SVG dimensions

const svg = d3.select("#myCanvas"); //select element

const rect = svg.append("rect"); //append takes one argument to add to the
element, in this case a rectangle

//we can now pass values with the attr method
rect.attr("x", 30) //x axis position
rect.attr("y", 0) //y axis position
rect.attr("width", 140) //width of the rectangle
rect.attr("height", 70) //height of the rectangle
rect.attr("fill", "green") //color
```

Although we can optimize this code by making use of method chaining in JavaScript, which is the property used for all the code present in this project.

```
const rect = d3.select("#myCanvas")
    append("rect")
        rect.attr("x", 30)
        rect.attr("y", 0)
        rect.attr("width", 140)
        rect.attr("height", 70)
        rect.attr("fill", "green")
```

It is also important to remind that the data for the visualizations is fetched asynchronously using Native XMLHttpRequest (XHR) object (AJAX) and formulated in JavaScript as a promise as shown below.

```
request('GET', 'https://sheets.googleapis.com/v4/spreadsheets/')//restricted key
.then((response) => {
    const data = JSON.parse(response.target.responseText);
//do something with data
}).catch()
function request(method, url) {
    return new Promise(function (resolve, reject){
        const xhr = new XMLHttpRequest();
        xhr.open(method, url);
        xhr.onload = resolve;
        xhr.onerror = reject;
        xhr.send();
    });
}
```

The different individual component implementations of the dashboard are shown and grouped by user as in the Component Design section.

3.1. Student Dashboard

For the visualizations of the student dashboard it was necessary to filter the data coming from the google sheet depending on the student ID. It is obvious that just the data corresponding to the current student is needed. In order to do that, a local storage with the student ID was implemented.

```
var personCode = document.getElementById('personCode').value;
localStorage.setItem( 'objectToPass', personCode );

window.location = 'http://nicolasleivab.github.io/Interactive-
Dashboard/student.html';
//pass person code input value to student dashboard page
```

In the second page the value of the local storage is saved and then its space is freed.

```
var personCode = localStorage.getItem('objectToPass'); //localStorage to var
localStorage.removeItem( 'objectToPass' ); //free localStorage
```

Then, at the beginning of every student visualization the data is filtered using the current student ID

```
// filter user ID
let newData = data.filter(function(d){return d.ID == personCode;});
```

3.1.1. Animated Progress Meter

Like in every visualization, it was necessary to format the data first. By default, the numerical variables are given as string. With the following function it is possible loop through every value and convert it

```
// Clean data
 data.forEach(function(d) {
  d.Rounds = +d.Rounds;
//every variable
});
```

Data was filtered in order to get only the values of Success Probability, which is a boolean counter for indicating failed or completed levels.

```
const donutData = newData.filter(function(d){return d["Success Probability"] >
0;});//Filter completed levels
const completedLevels = d3.map(donutData, function(d){return(d.level)}).keys();
//get each completed level
const startPercent = 0;
const endPercent = ((completedLevels.length)/11); //completed levels/all levels
```

D3.arc() function was used to determine the donut chart properties.

```
const arc = d3.arc()
  .startAngle(0)
  .innerRadius(radius)
  .outerRadius(radius - border);
```

For the animation, a counter was implemented which makes increments of 1% until it reaches the completion percentage according to the student filter.

```
let count = Math.abs((endPercent - startPercent) / 0.01);//mutable count and step
    step = endPercent < startPercent ? -0.01 : 0.01; //step = 1%
    progress = startPercent; //replace progress with last value

(function loops() {   //animation loop
  updateProgress(progress);

  if (count > 0) {
      count--;
      progress += step; //add 1% until count = 0 (reached endPercent)
      setTimeout(loops, 10);
  }
})();
```

You can check the code for this visualization with this link (starting from line 181: Donut Chart):
https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/overview_Student.js

3.1.2. Heatmap

The heatmap uses cal-heatmap.js library to render the visualization. It is however necessary to process the data in order to pass it as a value. A pure JavaScript function was implemented in order to get values for each grouped date in unix timestamp format to pass as data input to the cal-heatmap methods.

```
function sumAll(arr) {
  let sums = {}, value = [], mapDate;
  for (var i = 0; i < arr.length; i++) { //Loop through array
    mapDate = new Date(arr[i].date).valueOf()/1000; //new date in unix timestamp
    if (!(mapDate in sums)) {
      sums[mapDate] = 0;
    }
    sums[mapDate] += arr[i]['Success Probability']; //sum of values per date
    }

  for(mapDate in sums) {
    value.push({ date: mapDate, value: sums[mapDate]}); //push elements to new
array with dates and values
    }
    return value; //return array of objects
}
```

Data format is not ready yet and has to pass through an integrated parse function. Once ready, it is possible to call a new heatmap by setting its attributes. For this visualization, the heatmap was implemented by displaying daily cells organized by each month. This was the optimal organization according to the needs identified in the student user requirements.

```
var calendar = new CalHeatMap();

calendar.init({

  start: new Date(2019, 2), //starting date
  data: heatmapData,
  domain : "month", //several customizable attributes
  subDomain : "x_day",
  range : 12,
  cellsize: 15,
  cellpadding: 300,
  cellradius: 15,
  domainGutter: 15,
  weekStartOnMonday: 0,

});
```

Script link: https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/heatmapsettings.js

3.1.3. Leaderboard

A similar function to the one used for the heatmap was implemented. In this case, the function loops through each object in the array summing up two properties and grouping the result by each student ID. This is how the score for each student is calculated.

```
function sumAll(arr) {
  let sums = {}, value = [], studentID;
  for (var i = 0; i < arr.length; i++) { //loop through array
    studentID = arr[i].ID
    if (!(studentID in sums)) {
      sums[studentID] = 0;
    }
    sums[studentID] += arr[i]['Cycles'];
    sums[studentID]  += arr[i]['Instructions']; //sum Score
    }

  for(studentID in sums) {
    value.push({ student: studentID, score: sums[studentID]}); //push elements to
new array students and score
    }
    return value; //return array of objects
}

const sumPlayers = sumAll(data);
```

The result list of players with scores is then sorted from first to last place in ranking.

```
var sortedPlayers = sumPlayers.sort((aPlayer, bPlayer) => aPlayer.Score -
bPlayer.Score);
//sorts the players by score (ascending)
var topPlayers = sortedPlayers.slice(0, 3); //filters the first 3 players
```

Finally, the resulting array of objects is used as an input to create a table with the best three players.

```
function displayLeaderboard() {
  let theExport = ""; //initialize the export
  var i=1; // counter for ranking position
  topPlayers.forEach((player) => theExport += '<tr><td>' + i++ + '</td><td>' +
player.ID + '</td><td>' + player.Score + '</td></tr>'); //prints the row tables
  document.getElementById("leaderboard").innerHTML = theExport;
}

displayLeaderboard(topPlayers); //call function with 3 top players
```

Script *link:* https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/leaderboard.js

3.1.4. Bar Chart with Dropdown Filter

First, it was necessary to create an update function as stipulated in the official documentation of D3js version 5. This function is essential to create smooth dynamic graphs and it's about removing, entering and updating the graphics with changing data input.

```
//update function
function update(newData) {
// JOIN new data with old elements.
var rects = g.selectAll("rect")
    .data(newData, function(d){
        return d.level;
        });

// EXIT old elements not present in new data.
rects.exit()
    .transition(t) //transition time for the animations
    //insert attributes
    .remove();

// ENTER new elements present in new data.
rects.enter() //enter new rectangles
  .append("rect")
    //insert attributes and return values...
            .merge(rects)//merge new with old rectangles
            .transition(t)
              //inser attribures and return values...

}
```

In order to make the visualization interactive, a dropdown selector was implemented. This dropdown gets the values of every variable defined in the code and it calls the previous update function with the new filtered data every time the user picks a new selection/filter.

```
var selector = d3.select("#drop") //dropdown change selection
    .append("select") //append the actual dropdown to the div
    .attr("id","dropdown") //assign id
    .on("change", function(d){
         selection = document.getElementById("dropdown"); //select by id
            y.domain([0, d3.max(newData, function(d){return
d[selection.value];})]);//change y
                console.log(selection.value);
                update(newData); //call update function with new selected data

});
```

Script link: https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/overview_Student.js

3.1.5. Grouped Bar Chart with Dropdown Filter

At first glance, the grouped bar chart implementation looks very similar to the normal bar chart one. However, it has several differences, starting from how x axis is defined. There are two different domains, x0 represents the grouped bars (the actual categories in the x axis) while x1 represents the individual bars (different stats). Each type, range and padding can be defined separately.

```
//grouped bars
var x0 = d3.scaleBand()
    .range([0, width])
    .padding(0.2);
//individual bars
var x1 = d3.scaleBand()
    .range([0, x0.bandwidth() - 5])
    .padding(0.2);
```

It is worth mentioning that because of this difference, the domain for each variable must be defined differently using keys or a group of variables.

In this case, in order to implement the dropdown selector and make it work correctly with the update function, it was necessary to implement a conditional function for each group required. This means that every time the user picks a filter, another filter for a determined group of variables is created to be passed as values to the domain functions.

```
dropSelector.on("change", function(d){
  selected = document.getElementById("dropdown2");

    var xFilter = ['Loops', 'minL', 'avgL'];
   //Conditional filter for x1 variables and domain
    if(selected.value == 'Loops'){
      var xFilter = ['Loops', 'minL', 'avgL'];
    }
    else if(selected.value == 'Functions'){
      var xFilter = ['Functions', 'minF', 'avgF'];
    }
    //every other condition...
    //...
    }

x0.domain(newData.map(function(d) { return d.level; })); //grouped bars as levels
x1.domain(xFilter).rangeRound([0, x0.bandwidth()]); //individual bars as stats
y2.domain([0, d3.max(newData, function(d) { return d3.max(xFilter, function(key)
{ return d[key]; }); })]).nice();
//domain of each individual bar according to the xFilter
```

Script link: https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/groupedchart.js

3.2. Tutor Dashboard

3.2.1. Probability of Success Bar Chart with Date Filter

This visualization was implemented in a very similar way to the previous bar chart visualization. The difference is that the filter used is about date ranges, for that feature a jQuery range slider was implemented. This slider gets the minimum and maximum date of the data and apply a filter according to a date range slider, calling the update function for each new date.

```
//jQuery UI slider
$("#dateSlider2").slider({
//Different attributes like min, max and step
  values: [minDate.getTime(), maxDate.getTime()],
  slide: function(event, ui){
      $("#dateLabel3").text(formatTime(new Date(ui.values[0]))); //update labels
on the html
      $("#dateLabel4").text(formatTime(new Date(ui.values[1])));
      update3(data.filter(function(d){return ((d.date >= ui.values[0]) && (d.date
<= ui.values[1]));}));
      //call update function with new rects according to date filter
  }
});
```

Script link: https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/barchart_tutor.js

3.2.2. Grouped Bar Chart with Dropdown and Student ID Filter

This visualization was implemented exactly as the grouped bar chart for the student and they share the same script. The difference is that a student filter was added for the tutor making possible to filter data by student ID. The function gets the input element and applies the update function on click.

```
//student filter
$(document).ready(function() {
  $('#filter').click(function() {
    var studentID = Number(document.getElementById('filterID').value); //get
input
    if(studentsArray.includes(studentID)){
      update2(data.filter(function(d){return d.ID == studentID;})) //enter new
rects for the new filtered student
      } else {
      alert("error message")
      }
  });
});
```

3.2.3. Histogram with Double Filter and Date Range Slider

For the histogram visualization, the number of classes was defined by testing different combinations and picking the visually best one for the existing data. Since the true data will vary widely once the game log is officially updated in real time, a function for customizable bin size can be implemented.

```
if (selection2.value != 'All Levels' || (selection.value == 'Functions' ||
selection.value == 'Loops')){
  histogram.thresholds(x.ticks(7)); //Set number of classes
} else {
    histogram.thresholds(x.ticks(21)); //Set number of classes
}
```

Implementing a double filter for rows and columns was tricky, it was necessary to nest the filters by relevance. In this case, the column filter is nested in the row filter, so it has the memory of the selected row data. All this had to be correctly integrated with the different update function calls for every case. Also, a conditional execution was implemented in order to get a visualization for all levels.

```
const levelSelector = d3.select("#drop4") //dropdown change selection
.append("select") //append row filter dropdown
.attr("id","dropdown4")
.on("change", function(d){ // Row Filter
    selection2 = document.getElementById("dropdown4");
    if (selection2.value != 'All Levels'){ //condition for any level
    update(data.filter(function(d){return d.level == [selection2.value];}));
//update histogram columns data
    updatePie(data.filter(function(d){return d.level == [selection2.value];}),
"level"); //update pie
    resetSlider();
        instructionSelector.on("change", function(d){ // Column Filter
            selection = document.getElementById("dropdown3");
            update(data.filter(function(d){return d.level ==
[selection2.value];})); //update histogram row data
            resetSlider(); //reset jQuery slider
        });
    }
    else { //when All levels is selected
      update(data); //update histogram without any filter
      updatePie(data, 'level'); //update pie without any filter
      resetSlider();
          instructionSelector.on("change", function(d){ // Column Filter
              selection = document.getElementById("dropdown3");
              console.log([selection.value]);
              update(data); //again, no filter applied
              resetSlider();
          });
    }
});
```

Secondly, in order to implement the jQuery date range slider properly, it was necessary to integrate in the function the previous filters so it can't mix the selected data. As the other filters, this function has also an execution condition for all levels. In addition, a reset function had to be created to prevent mixing dates with new data whenever the user changes the row and column filter (levels and instructions).

```
//jQuery UI slider
$("#dateSlider").slider({
//attributes...
  slide: function(event, ui){
      $("#dateLabel1").text(formatTime(new Date(ui.values[0])));
      $("#dateLabel2").text(formatTime(new Date(ui.values[1])));
      if (selection2.value != 'All Levels'){
      update(data.filter(function(d){return ((d.date >= ui.values[0]) && (d.date
<= ui.values[1]) && (d.level == [selection2.value] || d.level ==
[selection2]));}));
      updateDate(data.filter(function(d){return ((d.date >= ui.values[0]) &&
(d.date <= ui.values[1]) && (d.level == [selection2.value] || d.level ==
[selection2]));}));
      //must include nested filters from both dropdowns
      }
      else{
      update(data.filter(function(d){return ((d.date >= ui.values[0]) && (d.date
<= ui.values[1]));}));
      updateDate(data.filter(function(d){return ((d.date >= ui.values[0]) &&
(d.date <= ui.values[1]));}));
      }
  }
});
```

A pie chart was implemented as a support visualization, the input data for this graph had to be arranged differently and a similar function as the one used in the heatmap and the leaderboard was implemented, all this with the objective of filtering data and summing properties to get the desired results. It is worth to mention that the update function of the pie chart had to be properly integrated with the above filters allowing both visualizations to be synchronized.

Repository: https://github.com/nicolasleivab/D3-Histogram-JQuery-slider
Script link: https://github.com/nicolasleivab/Interactive-Dashboard/blob/master/js/overview_Tutor.js

4. Testing and Debugging

Jest testing framework was used to implement unitary and integral tests. Several tests were created by importing functions from the different visualization scripts. In general, these tests are better for pure JavaScript functions, therefore the features tested with this framework were related to data filter, data manipulation and format functions. In the following code the test for the heatmap data manipulation function is shown.

```
const { sumAll } = require('./heatmapsettings'); //import function

test('should output date in unix timestamp and sum value', () => {
    const arrOfObj = sumAll([{date: '01/24/2019', "Success Probability": 1},
{date: '01/24/2019', "Success Probability": 1}]);
    expect(arrOfObj).toBe('[(date: 1550962800, value: 2)]'); //expected result
});
```

First, it is necessary to write the export modules in the source script, which can be temporally done for the testing period. Then, the exported function must be imported in the new testing script (scriptname.test.js). The test is written as above and it can be run by typing 'npm test' in the terminal.

Another example for the array and objects average function from the tutor bar chart script is shown.

```
const { average } = require('./barchart_tutor'); //import function

test('should output an array of objects with average success probability per
level ', () => {
    const newAverage = average([{level: 1, 'Success Probability': 1}, {level: 1,
'Success Probability': 0}, {level: 2, 'Success Probability': 1}]);
    expect(newAverage).toBe([{level: 1, average: 0.5}, {level: 2, average: 1}]);
//expected result
});
```

The results of every test indicate if the condition is fulfilled or not (boolean), showing also the real output in comparison with the expected output so fixes can be made afterwards.

For the D3js functions and more complex interactions manual end to end tests were constantly made. This means that for every new feature the application was tested in localhost with the console terminal in order check if any error has appeared. As a support measurement, console.log() function was applied to check different values and function outputs.

5. User Evaluation and Fixes

Once the dashboard is fully deployed, it would be helpful to get some feedback regarding user interface and experience for both student and tutor users. The user requirements presented in this project were carefully thought and complemented with other similar existing applications. However, specific needs can arise with time and it's important to keep track of user feedback.

Several user evaluation techniques exist, but for this case a straightforward approach would suffice, which could consist in making some simple questionnaires or receiving direct feedback about the interaction with the software and how the user's needs are met or not. With this, it is possible to make some modifications that can satisfy the new requirements as well as improve the site performance.

6. Further Development

As mentioned before, the implementation of a function that let the tutor customize the bin sizes for the histogram visualization can be made in the future. It is possible that data will change once the sample of students gets bigger and therefore a more dynamic approach to represent the distribution of the instructions per level would be needed.

For this matter, something like the following code snippet can be implemented

```
function update(numOfBins) {

  var histogram = d3.histogram()
      .value(function(d) { return d.value; })//value of the vector
      .domain(x.domain())
      .thresholds(x.ticks(numOfBins)); //number of bins

      //exit and enter rect function and attributes...
}
```

To successfully implement this feature, a correct integration of this update function with the different filters and update functions of the visualization would be needed.

Finally, an authentication middleware like passport.js could be used to strengthen the security of the web application before its officially deployed.

7. Reference

-D3js official documentation available on: https://github.com/d3/d3/wiki

- Haverbeke, M. (2019). Eloquent Javascript (3rd Edition). Retrieved from https://eloquentjavascript.net/

- Rauschmayer, A. (2015). Exploring ES6. Retrieved from https://exploringjs.com/es6.html

List of completed courses for this project:

-Janes, A. (Producer). (2015). Mastering Data Visualization in D3.js. Available on udemy: https://www.udemy.com/masteringd3js/

-Alicea, A. (Producer). (2015). JavaScript: Understanding the Weird Parts. Available on udemy: https://www.udemy.com/understand-javascript/

- Schwarzmüller, M. (Producer). (2019). Accelerated ES6 JavaScript Training. Available on udemy: https://www.udemy.com/es6-bootcamp-next-generation-javascript

-freeCodeCamp: Javascript Algorithms and Data Structures Certification. Available on: https://www.freecodecamp.org/