



UNIVERSIDADE D  
COIMBRA

Gustavo José Fernandes Morgado

LIVE PREVIEW AND CODING ASSISTANCE TOOL  
FOR VIRTUAL REALITY PROGRAMMING WITH  
A-FRAME

Dissertation in the context of the Master in Informatics Engineering,  
Specialization in Software Engineering advised by Professor Jorge Carlos dos  
Santos Cardoso and Professor António José Mendes and presented to the  
Department of Informatics Engineering of the Faculty of Sciences and  
Technology of the University of Coimbra.

July 2023





DEPARTAMENTO DE  
ENGENHARIA INFORMÁTICA  
FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

Gustavo José Fernandes Morgado

**LIVE PREVIEW AND CODING ASSISTANCE  
TOOL FOR VIRTUAL REALITY PROGRAMMING  
WITH A-FRAME**

Dissertation in the context of the Master in Informatics Engineering,  
Specialization in Software Engineering advised by Professor Jorge  
Carlos dos Santos Cardoso and Professor António José Mendes and  
presented to the Department of Informatics Engineering of the Faculty  
of Sciences and Technology of the University of Coimbra.

July 2023





DEPARTAMENTO DE  
ENGENHARIA INFORMÁTICA  
FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

Gustavo José Fernandes Morgado

FERRAMENTA DE PRÉ-VISUALIZAÇÃO EM  
DIRETO E ASSISTÊNCIA DE PROGRAMAÇÃO  
PARA REALIDADE VIRTUAL COM A-FRAME

Dissertação no âmbito do Mestrado em Engenharia Informática,  
especialização em Engenharia de Software, orientada pelo Professor  
Jorge Carlos dos Santos Cardoso e Professor António José Mendes e  
apresentada ao Departamento de Engenharia Informática da Faculdade  
de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2023



## Abstract

The process of learning how to program in virtual reality can be an arduous one. Not only does it involve the programmer to think in the three-dimensional space, it also obligates the user to view the results of their coding in separate times and spaces, introducing a break in the momentum of programming and slowing down the working speed.

Due to this, there is a need for a coding assistance tool in order to help programmers more clearly visualize the changes that they make. We chose to work on the Glitch platform, a popular editor that supports the A-frame framework used for programming virtual reality. In addition, it will also be added features that can help beginners get better used to the framework, alongside more veteran programmers who can appreciate an improvement on the experience of programming in A-Frame.

This report will present the different phases of the development process of the coding tool, starting with some testing of the A-Frame framework, followed by a research on how the Glitch platform works, a review of the state of art, followed by the decision of the methodologies implemented on the project, the planning of the project, and all the decisions taken in the development process and the usability testing. By following this approach, the aim is to deliver a robust and user-friendly coding assistance tool for virtual reality programming with A-Frame.

This tool will not only facilitate the visualization of code changes but also enhance the overall programming experience, empowering programmers of varying expertise levels to harness the full potential of A-Frame in their virtual reality applications.

## Keywords

Live Preview, A-Frame, Glitch, Programming, IDE





## Resumo

O processo de aprender a programar em realidade virtual pode ser árduo. Não só obriga o programador a pensar no espaço tridimensional, como também obriga o utilizador a visualizar os resultados da sua programação em tempos e espaços separados, introduzindo uma quebra no ritmo da programação e diminuindo a velocidade de trabalho. Por este motivo, existe a necessidade de uma ferramenta de assistência à programação para ajudar os programadores a visualizar mais claramente as alterações que efectuam.

Optámos por trabalhar na plataforma Glitch, um editor popular que suporta a framework A-Frame, que é utilizada para a programação de realidade virtual. Para além disso, também serão adicionadas funcionalidades que podem ajudar os principiantes a habituarem-se melhor à framework, ao lado de programadores mais veteranos que podem apreciar uma melhoria na experiência de programação em A-Frame.

Este relatório apresentará as diferentes fases do processo de desenvolvimento da ferramenta de programação, começando com alguns testes da framework A-Frame, seguidos de uma pesquisa sobre o funcionamento da plataforma Glitch, uma revisão do estado da arte, sucedida pela decisão das metodologias implementadas no projeto, do planeamento do projeto e de todas as decisões tomadas no processo de desenvolvimento e nos testes de usabilidade. Seguindo esta abordagem, o objetivo é fornecer uma ferramenta de assistência à programação robusta e de fácil utilização para a programação de realidade virtual com A-Frame.

Esta ferramenta não só facilitará a visualização das alterações ao código, como também melhorará a experiência geral de programação, permitindo que programadores de diferentes níveis de especialização aproveitem todo o potencial do A-Frame nas suas aplicações de realidade virtual.

## Palavras-Chave

Pré-visualização, A-Frame, Glitch, Programação, IDE



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Objectives . . . . .	2
1.3	Report Structure . . . . .	2
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	A-Frame . . . . .	5
2.2	Glitch . . . . .	9
<b>3</b>	<b>State of the Art</b>	<b>17</b>
3.1	Programming Editors . . . . .	17
3.2	Functionalities . . . . .	18
3.2.1	Code analyzer . . . . .	18
3.2.2	Support for web frameworks . . . . .	19
3.2.3	Multi-Edit and Multi-Selecting . . . . .	20
3.2.4	Code-folding . . . . .	20
3.2.5	Goto button . . . . .	21
3.2.6	Tags explanation . . . . .	22
3.2.7	Live Preview . . . . .	22
3.2.8	Extension Manager . . . . .	23
3.2.9	Auto-completion . . . . .	24
3.2.10	Highlighting . . . . .	27
3.2.11	Color Picker . . . . .	27
<b>4</b>	<b>Project planning</b>	<b>31</b>
4.1	Methodology . . . . .	31
4.2	Technology . . . . .	31
4.3	Tools . . . . .	32
4.4	Planning . . . . .	32
4.5	Risk Planning . . . . .	33
<b>5</b>	<b>System Description</b>	<b>35</b>
5.1	Requirements . . . . .	35
5.1.1	Survey . . . . .	35
5.2	Use Cases . . . . .	36
5.2.1	Functional Requirements . . . . .	37
5.2.2	Non Functional Requirements . . . . .	38
5.3	Architecture . . . . .	39

---

<b>6</b>	<b>Development</b>	<b>41</b>
6.1	A-Frame Helper . . . . .	41
6.1.1	Color Picker . . . . .	41
6.1.2	Documentation Display . . . . .	42
6.1.3	Component Installer . . . . .	43
6.1.4	Live preview . . . . .	44
<b>7</b>	<b>Testing</b>	<b>47</b>
7.1	Usability testing . . . . .	47
7.1.1	Planning and Procedure . . . . .	47
7.1.2	Results/Discussion . . . . .	48
<b>8</b>	<b>Conclusion</b>	<b>51</b>
	<b>References</b>	<b>53</b>
	<b>Appendix A Mockups</b>	<b>57</b>
	<b>Appendix B Testing Script</b>	<b>61</b>

# Acronyms

**API** Application Programming Interface.

**CSS** Cascading Style Sheets.

**HTML** Hypertext Markup Language.

**IDE** Integrated Development Environment.

**SUS** System Usability Scale.

**VR** Virtual Reality.



# List of Figures

2.1	Result of A-Frame testing . . . . .	7
2.2	Glitch . . . . .	9
3.1	Visual Studio Code . . . . .	18
3.2	Sublime Editor . . . . .	18
3.3	Example of manually analyzing code in IntelliJ Idea [10] . . . . .	19
3.4	Creating a Django project in PyCharm[11] . . . . .	20
3.5	Example of multi-selecting in Visual Studio Code . . . . .	20
3.6	Code folding before being activated . . . . .	21
3.7	Code folding after being activated . . . . .	21
3.8	Goto tool being used in Sublime to find a symbol . . . . .	22
3.9	Explanation of tag when being hovered . . . . .	22
3.10	Live Preview in the Glitch editor . . . . .	23
3.11	Example of an Extension Manager . . . . .	23
3.12	Example of basic code completion [13] . . . . .	24
3.13	Example of type-matching completion [13] . . . . .	25
3.14	Example of hippie completion [13] . . . . .	25
3.15	Example of postfix code completion [13] . . . . .	26
3.16	Example of machine-learning-assisted completion [13] . . . . .	27
3.17	Visual Studio Code Syntax highlighting . . . . .	27
3.18	RJ Text ED Color Picker, that only opens up when the user opens the tool . . . . .	28
3.19	Visual Studio Code Color Picker, shows up immediately next to back- ground attribute . . . . .	28
4.1	Gantt chart for the first semester . . . . .	32
4.2	Gantt chart for the second semester . . . . .	33
5.1	Software Architecture level 1 . . . . .	39
5.2	Software Architecture level 2 . . . . .	40
6.1	Example of the color picker being used . . . . .	42
6.2	Example of Documentation Display . . . . .	43
6.3	Example of Component Installer . . . . .	44
A.1	Mockup of the Color picker . . . . .	57
A.2	Mockup of the Autocomplete function . . . . .	58
A.3	Mockup of the Hyperlink function . . . . .	58
A.4	Mockup of the Component manager . . . . .	59





# List of Tables

3.1	Intersection between the IDEs and the features researched . . . . .	29
4.1	Risks . . . . .	34
5.1	Get live preview of coding . . . . .	36
5.2	Pick color for object . . . . .	36
5.3	Get documentation of respective element . . . . .	37
5.4	Install external component . . . . .	37
5.5	Requirements . . . . .	38
7.1	Average of the results of the questions about the features of A-Frame Helper . . . . .	49



# Chapter 1

## Introduction

The subject of this report is the development of a coding assistance tool with live preview for the A-Frame Virtual Reality (VR) framework. A-Frame is a Web-based Hypertext Markup Language (HTML)/JavaScript framework for creating 3D worlds for VR. A-Frame code can be written with any Integrated Development Environment (IDE) or text editor, but the Glitch platform and editor are often used to make A-Frame projects publicly available.

### 1.1 Motivation

Programming in VR is fundamentally different from programming normal software. Not only the user needs to take into account that the environment now has 3 dimensions, they also need to take into account all the interactions on that 3D space. When a programmer makes changes to the code, they have to execute it, to then have to open a new window to preview the results. This introduces a break to the momentum, and can be considered tedious to the majority of programmers, even becoming harmful to beginner programmers, since it ruins their first impressions. Tools that implement live preview already exist, however these tools are not optimal since they refresh the entire 3D environment. This is not optimal to the user, since they then need to relocate the point of view to the section in code where they were working on, especially in bigger projects where there are a huge quantity of objects.

These types of tools can also be helpful in educating new programmers about new languages that involve the VR area, like the framework A-Frame. By making the connection between the written code and the visual result more clear we can speed up the learning process of the students, since this tool will be mainly used to teach the A-Frame framework to students that are learning VR programming, one example being the students in the course Licenciatura em Design e Multimédia, with the main purpose being to make the process of writing code easier. Another purpose is making it so that they depend less on visual editing of the objects and more of a written approach, through code.

Other types of help that can be useful to programmers are tools that, for example, display different types of parameters for the programmer to choose, picking a

color from a color wheel instead of searching the correspondent hexadecimal code, or selecting a value from a value range. These tools would add more fluidity to the programming process and turn the programmer's job into a more pleasant experience.

## 1.2 Objectives

The main goal of this project is to improve the programmer's experience while coding in the VR framework A-Frame in the Glitch editor, by allowing them to see the results of their changes in the preview window, without having to reload the entire VR environment. This tool will be available only for the Glitch web editor, since it's the most common web editor for A-Frame.

The implementation of this tool requires a set of UserScripts that can run on the user's browser automatically when the user opens up the Glitch editor. These UserScripts will function as an add-on to Glitch, providing the live preview feature, which will reflect immediately the changes made in the Glitch preview page. Following it, other changes to the editor that can improve the interaction between the user and the IDE will be implemented. These tools will be a color picker, a window that displays the documentation, and a component manager.

Lastly, an evaluation of the tool will be conducted, in order to get a good understanding of the possible difficulties that a beginner programmer may have when installing and executing the tools. The evaluation will also provide us with an understanding of the impact that these types of tools have on novice A-Frame programmers.

## 1.3 Report Structure

The report will follow the structure below:

- **Background:** This chapter gives an idea of how the tools of the project work. It's focused on the framework that the coding assistance tool will run upon, researching on how to work with the framework and its architecture, and it will also be focused on the functionalities of the web editor, and how it works.
- **State of Art:** The goal of this chapter is to give an insight of the current state of programming editors and the assistance that they provide to the user. This will be accomplished by researching the most popular editors, as well as what types of coding assistance do they provide, and how can they fit in the project
- **Project Planning:** This chapter will introduce the planning for the project, including decisions in software processes, tools that are gonna be used, and risk analysis.
- **System Description:** This chapter will be about the scope of the project, which will be done by explaining the use cases of the system, as well as a set of

functional and non functional requirements. It will also include the system's architecture.

- **A-Frame Helper:** This chapter focuses on the development of the product, explaining how the features are implemented, along with problems and decisions that were made.
- **Testing:** This chapter showcases the planning, execution and results of the conducted usability test, as well as an analysis of those results.
- **Conclusion:** The final chapter will be a presentation of the final conclusion of the report.



# Chapter 2

## Background

This chapter will provide a background on the tools used in this project, namely the A-Frame Virtual Reality (VR) framework and the Glitch programming platform.

### 2.1 A-Frame

A-Frame [1] is a web framework that permits the user to program VR environments in Hypertext Markup Language (HTML). This framework is based on Three.js [2], which is a JavaScript library that is used to create animations of 3D objects in a web browser, and it also follows an entity-component structure [3], which is an architectural pattern used on video game development, based on the principle of composition over inheritance. It supports the vast majority of VR headsets in the market while being an open-source project, which lead it to have one of the biggest VR communities[4].

When starting to program in A-frame, after adding the framework file to the head section of the HTML file, the first step is to create the virtual environment. This can be done by using the element `<a-scene>`, which will hold all the VR components that we will use. After creating it, we can now add a sky element, which is the `<a-sky>` tag, and an object, which in this example will be a cylinder. These elements are what we call primitives. Primitives are entity-component patterns made to be more accessible to beginners. These primitives can also be created by developers [5]. Examples of these primitives are geometrical shapes, like cylinders and rectangles. This creation can be seen in the code below.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Primitives - 01 - Position</title>
5     <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
6   </head>
7   <body>
8     <a-scene>
9       <a-cylinder position="0 0 -2" height="5" radius="2" rotation="0 45 0"
10        color="red"></a-cylinder>
```

```

11     <a-sky color="#ADD8FF"></a-sky>
12
13
14   </a-scene>
15 </body>
16 </html>

```

After creating the scene and the object, we can now add an environment. This environment however is an external component. External components are components that other members of the community created for public use. They can be added into the project by importing them to the head section of the HTML code, using their native link. After adding the component, we can now add its respective element, which is done by using the element `<a-entity environment=X>`, that will setup an environment that covers the whole preview of the object, with X being the preset chosen for the project. An example of this can be seen in the code below.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Primitives - 01 - Position</title>
5     <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
6     <script src="https://unpkg.com/aframe-environment-component@1.1.0/dist/
7       aframe-environment-component.min.js">
8     </script>
9   </head>
10  <body>
11    <a-scene>
12      <a-entity environment="preset: forest"></a-entity>
13
14      <a-cylinder position="0 0 -2" height="5" radius="2" rotation="0 45 0"
15        color="red"></a-cylinder>
16
17      <a-sky color="#ADD8FF"></a-sky>
18
19    </a-scene>
20  </body>
21 </html>

```

After this, we can also add a texture to the object, by creating a tag called `<a-assets>`, which will include all the textures that will be used in the project. We can declare a texture by creating an `<img>` tag, with the source being either a direct link to the image, or a path to the directory where that image resides. Finally, we can link this image to the object by adding the field `src` to the primitive, which will be filled with the id of the texture. The final product can be seen in 2.1.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Primitives - 01 - Position</title>
5     <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
6     <script src="https://unpkg.com/aframe-environment-component@1.1.0/dist/
7       aframe-environment-component.min.js">
8     </script>
9   </head>

```



```

10 <body>
11   <a-scene>
12     <a-entity environment="preset: forest"></a-entity>
13
14     <a-assets>
15       
16     </a-assets>
17
18     <a-cylinder src="#boxTexture" position="0 0 -2" height="5" radius="2"
19       rotation="0 45 0" color="red"></a-cylinder>
20
21     <a-sky color="#ADD8FF"></a-sky>
22
23   </a-scene>
24 </body>
</html>

```

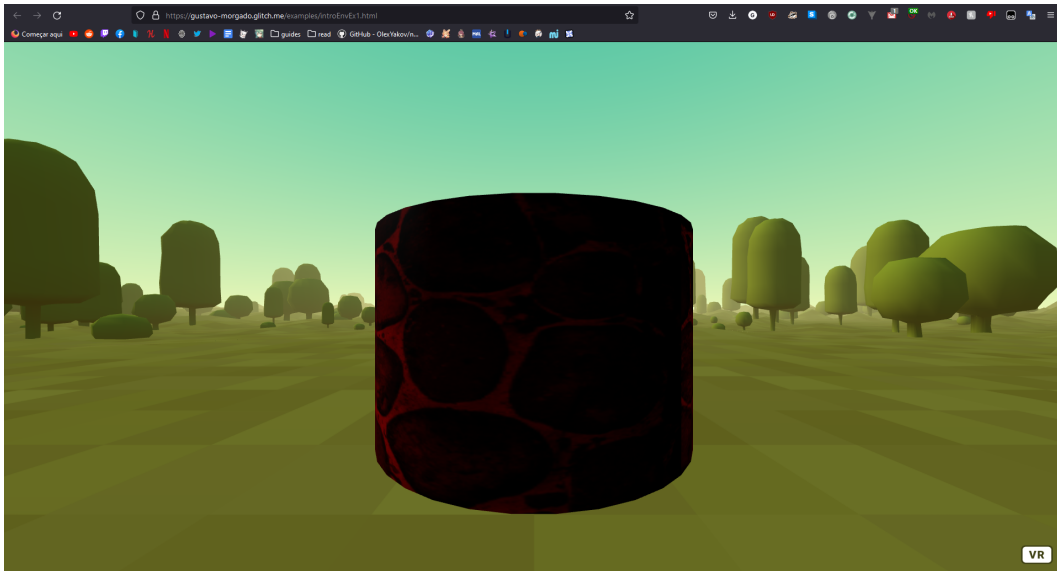


Figure 2.1: Result of A-Frame testing

According to the documentation, a component is a JavaScript module that can be configured, reused and shareable. These components can be used in other entities to expand on behavior and appearances.

When creating a component, the user must declare it before the a-scene element. After declaring it, the user can register a component by using the method `registerComponent()`, where it will take as arguments the name of the component and the component definition. The component definition is where all the methods of the component will be. This definition will include the `init` function, which will be called once the component connects to its entity. Afterwards, the user can call this component on the a-scene, like it's shown below.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Component Creation</title>
5     <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
6   </head>

```

```
7 <body>
8   <script>
9     AFRAME.registerComponent('hello-world', {
10       init: function () {
11         console.log('Hello, World!');
12       }
13     });
14   </script>
15
16   <a-scene>
17     <a-entity hello-world></a-entity>
18   </a-scene>
19 </body>
20 </html>
```

There are two ways to set a component. By using HTML, which is the process previously demonstrated, where the user simply calls the component on the static HTML, and by using JavaScript, where instead, the user summons the component by using the `setAttribute()` method, calling it on the JavaScript file, where the user must enter it like so:

```
1 document.querySelector('a-scene').setAttribute('hello-world', '');
```

To add more into the component, the user can pass data into it by first defining its properties via the schema. This schema will include the arguments of the component, Each property will include the default value, the name, and the type of property, which will dictate how the data will be parsed.

In the example below, we can see a log component, that can handle an argument that will be a string. This string will then be printed into the console. This component is simple, however it serves as a base to understand how one could implement more components, with more complex methods. Some of these methods include the `tick()` method, where every frame of the scene's render loop will execute an action, and `update()`, where it's called every time the content of the component updates.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Component Creation</title>
5     <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
6   </head>
7   <body>
8     <script>
9       AFRAME.registerComponent('log', {
10         schema: {
11           message: {type: 'string', default: 'testing'}
12         },
13
14         init: function () {
15           console.log(this.data.message);
16         }
17       });
18     </script>
19
```

```

20   <a-scene>
21     <a-entity log="message: This is a test!"></a-entity>
22   </a-scene>
23 </body>
24 </html>

```

## 2.2 Glitch

Glitch [6] is a web platform, that allows the programmer to create full-stack web applications. This means that the programmer can program both frontend and backend on this service. It provides a platform to host those applications, it's own JavaScript web server, and it has its own build preview. This editor also has a terminal, and a tool to check the logs and status of the app, along with a project directory, that stores all the HTML and JavaScript files, along with other assets that will be needed for the project, like images.

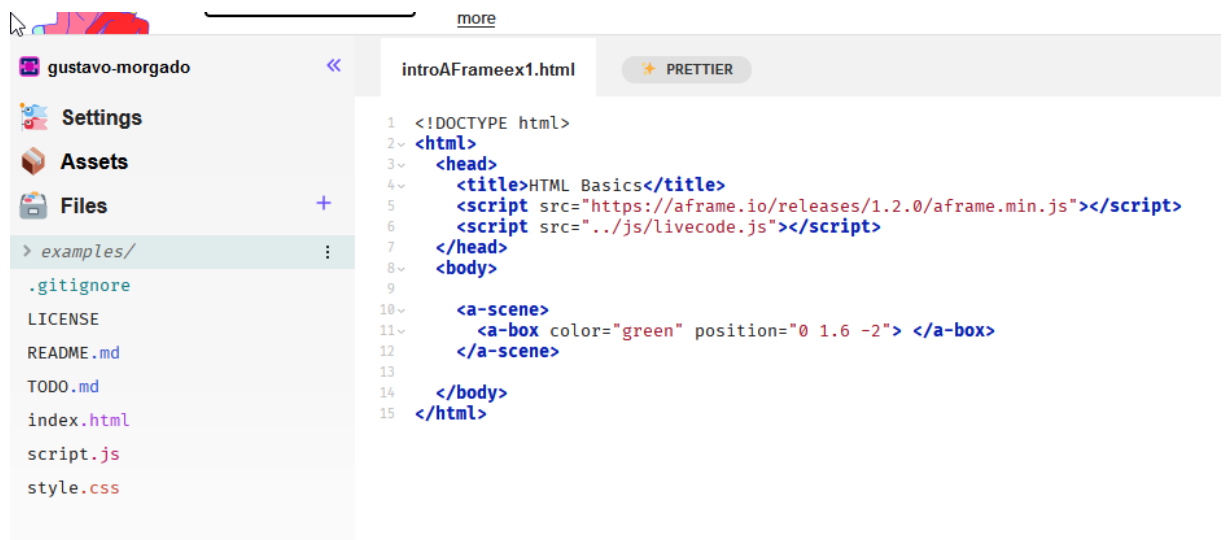


Figure 2.2: Glitch

The Glitch platform editor comes with several helpful functionalities to enhance the programmer's experience. These include highlighting HTML tags and a text wrapping feature. With text wrapping, when the user types in the editor and reaches the end of the text window, the text automatically continues on the line below.

Apart from the visible features in the figure 2.2, the editor also includes other helpful capabilities. It automatically inserts closing tags whenever an element is created, ensuring proper code structure. Additionally, there is an auto-refresh feature that activates whenever the programmer makes code edits. When the auto-refresh occurs, it resets the project's viewpoint to the central coordinates, which can be disruptive and negatively impact the programmer's experience. These changes can be seen either on the preview panel provided by Glitch, that is placed to the right of the Integrated Development Environment (IDE), or in a new tab.

For the Glitch editor to be able to compile in the browser, it needs to have a code-editor component in the web page. This component is called CodeMirror[7], it's a text editor implemented in JavaScript for the browser, being specialized for editing code, and capable of handling multiple languages, like PHP, C, SQL and Ruby. It also includes an Application Programming Interface (API) and a Cascading Style Sheets (CSS) theming system, that allows the user to customize and extend the functionalities of CodeMirror.

To understand better the inner workings of CodeMirror, a small prototype was created, where the user could input HTML code into a box and click the submit button, where the preview of the code would appear. This component also requires jQuery to be installed. To use the CodeMirror in an HTML page, so that the editor itself is working, the programmer must declare the scripts related to the libraries, and they can now setup the editor. This can be done by creating a textarea element, that can be used as an argument to create a CodeMirror instance, using the method `fromTextArea`, like the code snippet below.

```
1 $(document).ready(function(){
2   var code = $(".codemirror-textarea")[0];
3
4   var editor = CodeMirror.fromTextArea(code, {
5     lineNumbers : true
6
7   });
8
9 });
```

With the editor created, it is now possible to enter code in CodeMirror and run it, however it will not show the result of the code. One method that can be used to create a preview of the result is a PHP form, that receives the result of the HTML code and post it on another location in the web page, just like how it's implemented in the code below.

```
1 <?php
2
3 $comment = null;
4
5 if($_SERVER['REQUEST_METHOD'] === 'POST' &&
6   !empty($_POST['preview-form-comment'])){
7   $comment = $_POST['preview-form-comment'];
8 }
9 ?>
10
11 <!DOCTYPE html>
12 <html>
13   <head>
14     <title>CodeMirror - Form</title>
15     <link rel="stylesheet" type="text/css"
16       href="plugin/codemirror/codemirror-5.65.5/lib/codemirror.css">
17     <link rel="stylesheet" type="text/css" href="css/default.css">
18   </head>
19   <body>
20     <form id="preview-form" method="post" action="<?php echo
```

```

20     $_SERVER['PHP_SELF']; ?>>
    <textarea class="codemirror-textarea" name="preview-form-comment"
21         id="preview-form-comment"><?php echo $comment; ?></textarea>
22     <br>
    <input type="submit" name="preview-form-submit" id="preview-form-submit"
23         value="Submit">
24 </form>
25 <div id="preview-comment" style="width: 640px;">
26     <?php echo $comment; ?>
27 </div>
28 <!--JavaScript-->
29 <script type="text/javascript" src="js/jquery.min.js"></script>
30 <script type="text/javascript"
    src="plugin/codemirror/codemirror-5.65.5/lib/codemirror.js"></script>
31 <script type="text/javascript" src="js/default.js"></script>
32
33 </body>
34 </html>

```

Other features that CodeMirror includes are addons that supplement the editor, more specifically, functionalities that are going to be studied in the state of art. When creating an editor instance on CodeMirror, the user can then define some parameters. These parameters will dictate on what the editors contains. For example, the programmer can decide if they want the editor to incorporate syntax highlighting, an automatic closure of brackets and tags, or a pop-up that presents multiple choices, which works as an auto-complete, and other multiple arguments. An implementation of these addons can be seen below.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Code Mirror</title>
6     <link rel="stylesheet" type="text/css"
7         href="plugin/codemirror/codemirror-5.65.5/lib/codemirror.css">
8     <script type="text/javascript"
9         src="plugin/codemirror/codemirror-5.65.5/lib/codemirror.js"></script>
10    <script type="text/javascript"
11        src="plugin/codemirror/codemirror-5.65.5/mode/javascript/javascript.js"></script>
12    <script type="text/javascript"
13        src="plugin/codemirror/codemirror-5.65.5/mode/xml/xml.js"></script>
14    <script type="text/javascript"
15        src="plugin/codemirror/codemirror-5.65.5/mode/css/css.js"></script>
16    <script type="text/javascript"
17        src="plugin/codemirror/codemirror-5.65.5/mode/htmlmixed/htmlmixed.js"></script>
18    <script type="text/javascript"
19        src="plugin/codemirror/codemirror-5.65.5/addon/edit/closetag.js"></script>
20    <script type="text/javascript"
21        src="plugin/codemirror/codemirror-5.65.5/addon/edit/closebrackets.js"></script>
22  </head>

```

```
20 <body>
21
22 <textarea id='code'>
23
24 </textarea>
25
26 <script type="text/javascript">
27     var editor = CodeMirror.fromTextArea(document.getElementById("code"), {
28         lineNumbers: true,
29         mode: "htmlmixed",
30         autoCloseTags: true,
31         autoCloseBrackets: true
32     });
33
34 </script>
35 </body>
36 </html>
```

For the user to turn on a feature, all they need to do is search for what feature they would like for the editor to have, and then add that field to the declaration of the editor instance. For these features to work, it is needed to load the correct mode for the editor. Since the editor can only handle one mode, when working on the editor the user must go to the documentation and search through the list of modes that exist. In this example, it was picked the `htmlmixed` [8]. This mode is a wrap of the XML, JavaScript and CSS modes, being dependent on those three. After loading this mode, it was also needed to load the correspondent JavaScript files of those features, with the exception of `lineNumbers`, which was automatically loaded on `CodeMirror`. After this, it was implemented the auto-complete feature, which the coding can be seen below.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Code Mirror</title>
6 <link rel="stylesheet" type="text/css"
7     href="plugin/codemirror/codemirror-5.65.5/lib/codemirror.css">
8 <script type="text/javascript"
9     src="plugin/codemirror/codemirror-5.65.5/lib/codemirror.js"></script>
10 <script type="text/javascript"
11     src="plugin/codemirror/codemirror-5.65.5/mode/javascript/javascript.js"></script>
12 <script type="text/javascript"
13     src="plugin/codemirror/codemirror-5.65.5/mode/xml/xml.js"></script>
14 <script type="text/javascript"
15     src="plugin/codemirror/codemirror-5.65.5/mode/css/css.js"></script>
16 <script type="text/javascript"
17     src="plugin/codemirror/codemirror-5.65.5/mode/htmlmixed/htmlmixed.js"></script>
18 <script type="text/javascript"
19     src="plugin/codemirror/codemirror-5.65.5/addon/edit/closetag.js"></script>
20 <script type="text/javascript"
21     src="plugin/codemirror/codemirror-5.65.5/addon/edit/closebrackets.js"></script>
```

```
18 <script type="text/javascript"  
    src="plugin/codemirror/codemirror-5.65.5/addon/hint/show-hint.js"></script>  
19 <script type="text/javascript"  
    src="plugin/codemirror/codemirror-5.65.5/addon/hint/css-hint.js"></script>  
20 <link rel="stylesheet" type="text/css"  
    href="plugin/codemirror/codemirror-5.65.5/addon/hint/show-hint.css">  
21 </head>  
22  
23 <body>  
24  
25 <textarea id='code'>  
26  
27 </textarea>  
28  
29 <script type="text/javascript">  
30     var editor = CodeMirror.fromTextArea(document.getElementById("code"), {  
31         lineNumbers: true,  
32         mode: "htmlmixed",  
33         autoCloseTags: true,  
34         autoCloseBrackets: true,  
35         extraKeys: {"Ctrl-Space": "autocomplete"}  
36     });  
37  
38 </script>  
39 </body>  
40 </html>
```

From the code, it can be seen that the method for the auto-complete implementation is different from the prior ones. This is because there is no instant code completion, possibly due to it just being an API. What CodeMirror does offer is another type of auto-complete, that can be summoned using hotkeys. After discovering this, the Glitch editor was booted up, in order to see if it actually had this feature, and that it wasn't noticed simply because of being controlled by hotkeys. However, it still didn't work, which lead to believe that the glitch developers chose to purposely not include this function.

Going back to the implementation, CodeMirror includes a field on the editor instance called `ExtraKeys`, which assigns extra features of the CodeMirror to hotkeys selected by the user. Since the argument for this field is a map, it has lead the author to believe that multiple hotkeys can be handled. For this feature to work, it needs to load the correspondent CSS file, in order for the popup to appear.

Next, it was changed the process of showing the output of the code in CodeMirror. In the previous example, the prototype had been created using PHP language. This time, the use of PHP was discarded, instead looking for a solution that only involved HTML and JavaScript, using the method called `getValue()`, that returns the content of the editor. With that matter sorted, we had to make sure that it could handle the A-Frame framework. After running some HTML code that included a A-Frame scene, the resulting output covered the entire HTML page, instead of just below the editor. While researching how to show this output in a more constricted way, one solution was found, which was the insertion of an `iframe` element. When using this element, the corresponding result of the code was limited to the window of the `iframe`, while also being able to interact.

Finally, there was an attempt to try to bring the live preview aspect to the prototype. After researching the various events that the CodeMirror editor handled, one of them brought attention, which was the "change" event [9]. This event triggers every time that the content of the editor has changed, where it will receive the instance of the editor as of that moment and a object, that includes the properties *from*, *to*, *text*, *removed* and *origin*. The *from* and *to* properties include the position of the character that is being changed, having the line and character number. *text* will be an array of strings that will include the text that will replace the previous one, with that text being in the *removed* property. Finally, the *origin* property includes the nature of the change, whether it's an input, a delete action, paste, and others.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Prototype</title>
6     <link rel="stylesheet" type="text/css"
7       href="plugin/codemirror/codemirror-5.65.5/lib/codemirror.css">
8     <link rel="stylesheet" type="text/css"
9       href="plugin/codemirror/codemirror-5.65.5/addon/hint/show-hint.css">
10
11    <script type="text/javascript"
12      src="plugin/codemirror/codemirror-5.65.5/lib/codemirror.js"></script>
13
14    <script type="text/javascript"
15      src="plugin/codemirror/codemirror-5.65.5/mode/javascript/javascript.js"></script>
16
17    <script type="text/javascript"
18      src="plugin/codemirror/codemirror-5.65.5/mode/xml/xml.js"></script>
19
20    <script type="text/javascript"
21      src="plugin/codemirror/codemirror-5.65.5/mode/htmlmixed/htmlmixed.js"></script>
22
23    <script type="text/javascript"
24      src="plugin/codemirror/codemirror-5.65.5/mode/css/css.js"></script>
25
26    <script type="text/javascript"
27      src="plugin/codemirror/codemirror-5.65.5/addon/edit/closetag.js"></script>
28
29    <script type="text/javascript"
30      src="plugin/codemirror/codemirror-5.65.5/addon/edit/closebrackets.js"></script>
31
32    <script type="text/javascript"
33      src="plugin/codemirror/codemirror-5.65.5/addon/hint/show-hint.js"></script>
34
35    <script type="text/javascript"
36      src="plugin/codemirror/codemirror-5.65.5/addon/hint/css-hint.js"></script>
37
38  </head>
39  <style type="text/css">
40    #test {
41      position: static;
42      height: 500px;
43      width: 1000px;
44    }
45  </style>
46
47  <body>
48
49    <textarea id='code'>
```



```
35 </textarea>
36
37 <iframe id="test"></iframe>
38 <script type="text/javascript">
39
40     var ifrm;
41
42     var editor = CodeMirror.fromTextArea(document.getElementById("code"), {
43         lineNumbers: true,
44         mode: "htmlmixed",
45         autoCloseTags: true,
46         autoCloseBrackets: true,
47         extraKeys: {"Ctrl-Space": "autocomplete"}
48     });
49
50     editor.on("change", function(instance,object){
51         //console.log(instance);
52
53         if(object.origin === "+input" || object.origin==="+delete" ||
54            object.origin==="paste"){
55             //DEBUG
56             console.log(object);
57             console.log("CONTENT")
58             console.log(editor.getLine(object.from.line));
59             console.log((editor.getLine(object.from.line)).charAt(object.from.ch));
60
61             ifrm = document.getElementById('test');
62             ifrm = ifrm.contentWindow || ifrm.contentDocument.document ||
63                 ifrm.contentDocument;
64             ifrm.document.open();
65             ifrm.document.write(instance.getValue());
66             ifrm.document.close();
67
68         }
69     });
70 </script>
71
72
73 </body>
74 </html>
```

From the code above, we can see the inclusion of the iframe below the area for the text editor, whose size was managed with CSS, since the default size was too small. After that, the event handler was added, which will detect if the user wrote, pasted or deleted code, refreshing the content of the iframe with the new value that is in the text area. However, this process will refresh the entire environment, which is what the tool will fix.



# Chapter 3

## State of the Art

This chapter will present the research made on the state of the art. Due to the theme of the project, it was decided to first research the existing IDE, taking notes of what functionalities do they have and what improvements do they make. After that, those functionalities were separated into categories, based on their purpose by the IDE.

### 3.1 Programming Editors

The first action was to search for the most popular IDEs for HTML programming and investigate them. This helps in terms of researching what type of functionalities exist in IDE that are missing in the Glitch editor. The following IDEs were chosen based on a variety of websites that ranked the best IDE for HTML programming.

Some IDE were downloaded and tested in person, while others were analyzed by looking at what features they had on their website. The testing that was done was with HTML and JavaScript files, trying to see what help did the IDE provide to the user when writing code that involves those languages, mostly by erasing snippets of the code and seeing if they gave any suggestion, deleting parameters, and seeing what tools they provided. After that, it was researched on what other features did the IDE have that were possibly missed during the testing. The IDE that were tested were Visual Studio Code (Figure 3.1), Sublime Editor (Figure 3.2), Atom, Brackets, AWS Cloud 9, IntelliJ IDEA, PyCharm, PHPStorm, Code::Blocks, RJ TextED, Komodo Edit and Light Table.

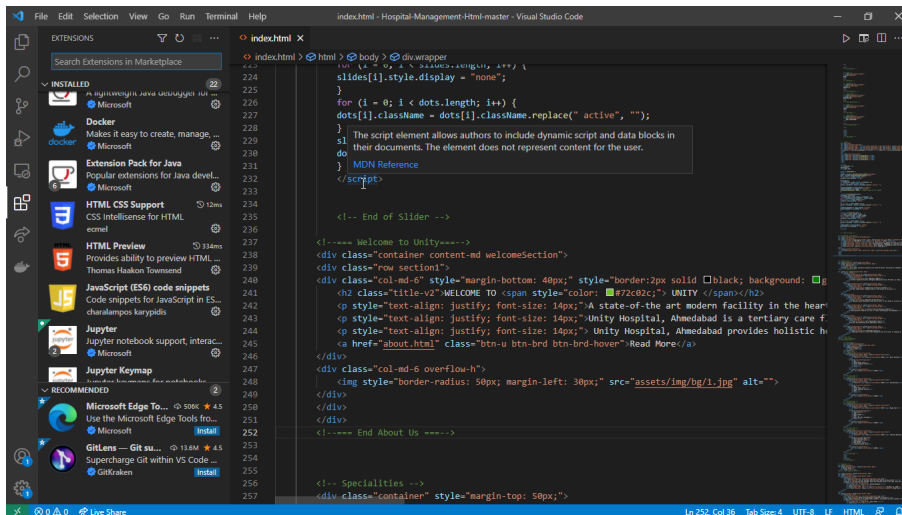


Figure 3.1: Visual Studio Code

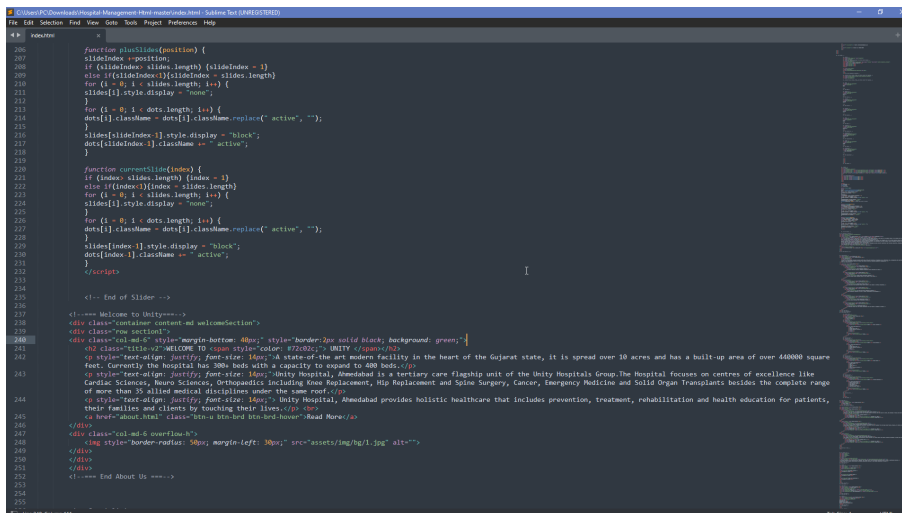


Figure 3.2: Sublime Editor

## 3.2 Functionalities

After researching some of the most used IDE that exist in the market, we organized the functionalities identified and looked deeper into their purpose.

### 3.2.1 Code analyzer

Code analyzing is the process of the IDE analyzing the code as it is being written, highlighting any syntax errors that appear instantly. This process can also be done manually, which will run the code analyzer through all the code existent in the project, instead of just the one being written in the editor. By running it manually, the user can get a full report of the existent problems. This feature is in VS Code,

IntelliJ IDEA, PyCharm, PHPStorm, Code::Blocks. An example of this feature is shown in the Figure 3.3.

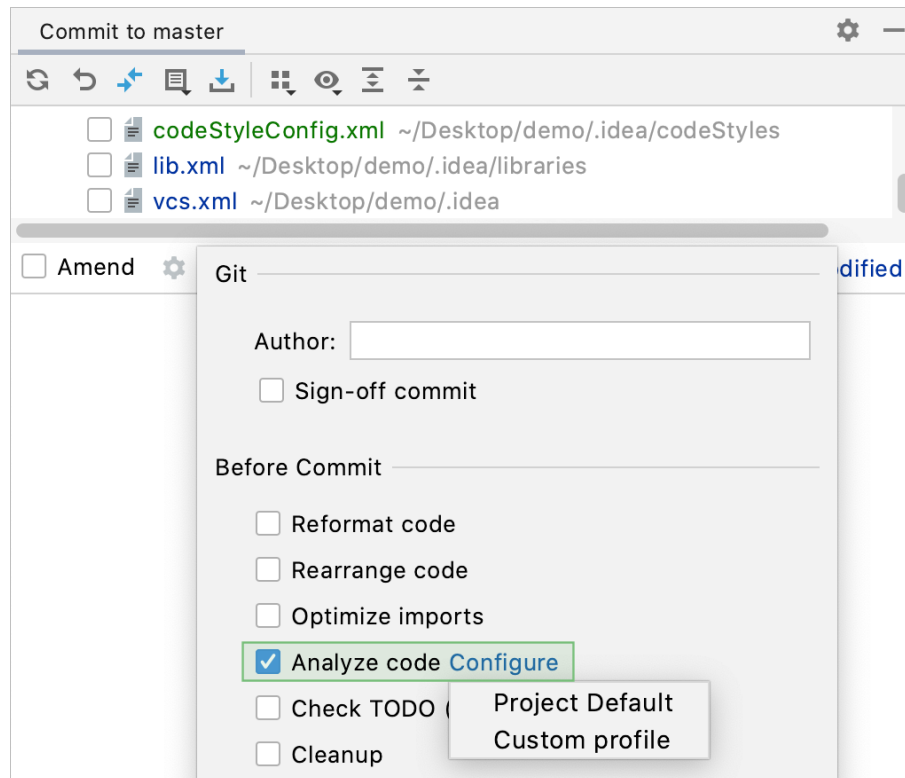


Figure 3.3: Example of manually analyzing code in IntelliJ Idea [10]

### 3.2.2 Support for web frameworks

Some IDE offer framework-specific support for modern web development frameworks, which include Django, Google App Engine, and others. These frameworks are vastly used, and every modern IDE needs to be able to handle these frameworks. This support is based upon having special debuggers, code completion, navigation between views and templates, and the inclusion of dedicated specific project types to those frameworks. IDE that have this feature are IntelliJ IDEA, PyCharm, PHPStorm, VS Code. In figure 3.4, we can see an example of the creation process of a project that uses other frameworks.

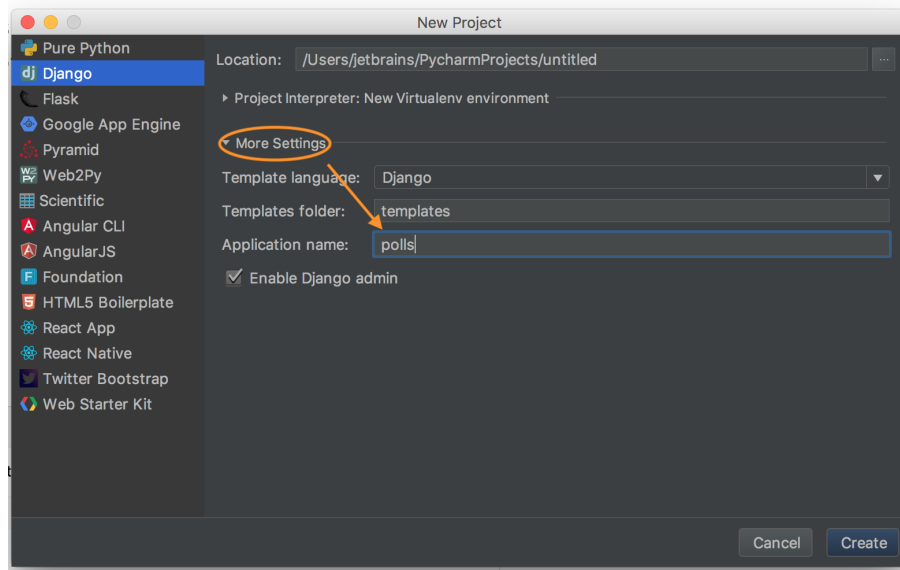


Figure 3.4: Creating a Django project in PyCharm[11]

### 3.2.3 Multi-Edit and Multi-Selecting

Multi-selecting is about selecting multiple instances of a keyword, making it easier to edit multiple lines in where that keyword was used, instead of manually searching for every instance of that word and editing one by one. After selecting it, the user can then use Multi-edit to change all the selected words at the same time. All IDEs use this feature. We can see how this tool is used in figure 3.5.

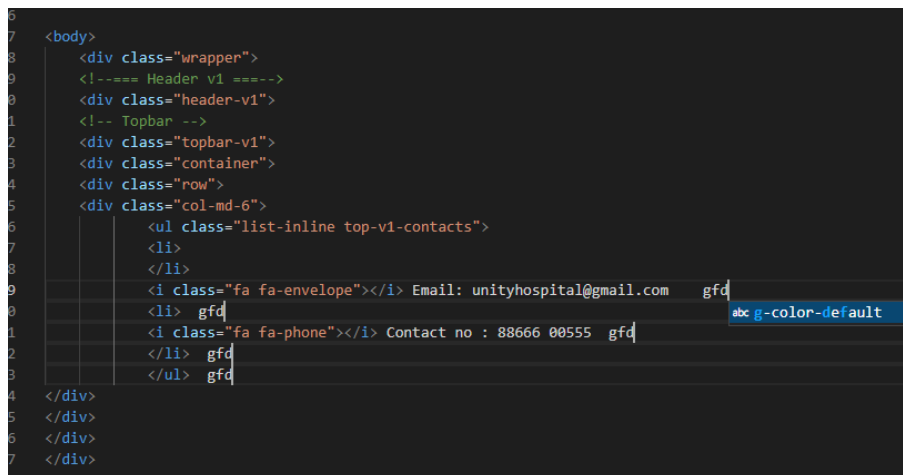


Figure 3.5: Example of multi-selecting in Visual Studio Code

### 3.2.4 Code-folding

Code-folding is the act of being able to hide methods, with the purpose of helping the user manage multiple functions and other sections of code, letting them focus more easily on the section they're working at. This feature is normally implemented

by inserting a button next to every method call, allowing the user to collapse specific parts of the code, to ensure a more clean workspace. Every editor seemed to include this feature. The way this feature works is presented in figure 3.6 and in figure 3.7.

```

$('.modal-opener').on('click', function()
{
  if( $('#sky-form-modal-overlay').length )
  {
    $('body').append('<div id="sky-form-modal-overlay" class="sky-form-modal-overlay"></div>');
  }

  $('#sky-form-modal-overlay').on('click', function()
  {
    $('#sky-form-modal-overlay').fadeOut();
    $('#sky-form-modal').fadeOut();
  });

  form = $('#this').attr('href');
  $('#sky-form-modal-overlay').fadeIn();
  form.css('top', '50%').css('left', '50%').css('margin-top', -form.outerHeight()/2).css('margin-left', -form.outerWidth()/2).fadeIn();

  return false;
});

```

Figure 3.6: Code folding before being activated

```

<> blog.html > html
1 <!DOCTYPE html>
2 <html>
3 > <head>...
28 </head>
29 <body>
30 <div class="wrapper">
31 <!--== Header v1 ==-->
32 > <div class="header-v1">...
164 </div>
165 <!-- End Navbar -->
166
167 <!-- Image title -->
169
170 > <div style="text-align: center; margin-top: 50px; margin-bottom: 40px;">...
172 </div>
173 <!-- End title -->
174
175 <!--== Content Part ==-->
176 <!-- BLOG 1 -->
177 > <div class="container content">...
287 </div>
288
289 <!--== Footer ==-->
291 > <div class="footer-v1">...
400 </div>
401 <!--== End Footer ==-->
402 </div><!--/wrapper-->
403
404
405 </body>
406 </html>

```

Figure 3.7: Code folding after being activated

### 3.2.5 Goto button

The Goto button is a tool that lets the user navigate to a specific line or element declared in the code. This feature is more helpful the more code there is, since it removes the necessity to scroll down the entire window to search for the intended element. There are other places that you can use the Goto button, like definitions, references, and lines. This feature was present in VS code, Sublime, Atom, Brackets, Pycharm, IntelliJ IDEA, PHPStorm, RJ TextED and Bluefish, and it can be seen in figure 3.8.

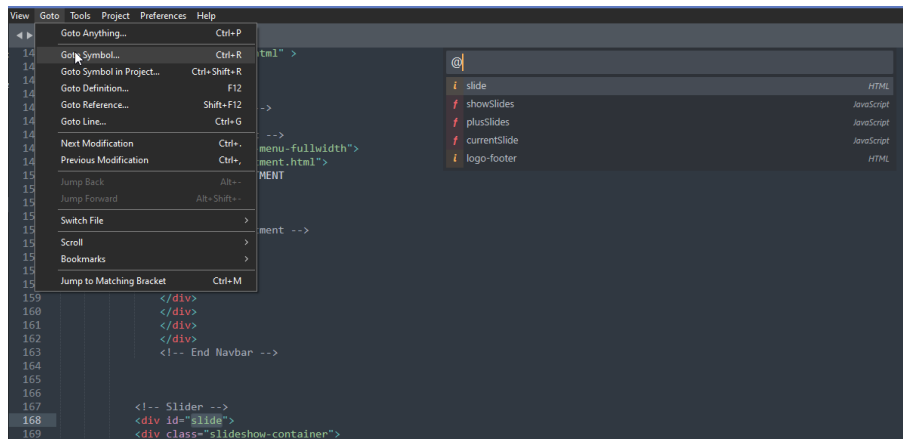


Figure 3.8: Goto tool being used in Sublime to find a symbol

### 3.2.6 Tags explanation

This feature helps the programmer know what keywords they are writing and how they work. It is mostly based upon showing the entry to the keyword in the documentation of the programming language, sometimes even showing the hyperlink to that page. Other information that it can show is the available parameters from that method. IDEs that have this feature are VS Code, IntelliJ IDEA, PyCharm, and PhpStorm.

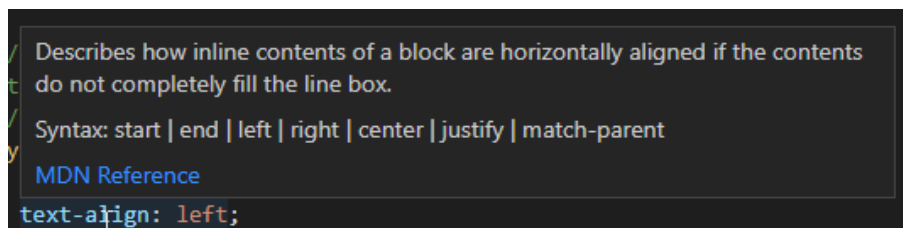


Figure 3.9: Explanation of tag when being hovered

### 3.2.7 Live Preview

Live Preview is the presentation of the expected web page looks. It allows the user to see the result of the changes they make to the code as if they launched the web page as is, allowing them to always have a real-time view of the web page. IDEs that use this feature are VS Code, PyCharm, IntelliJ IDEA, PhpStorm, Atom, Brackets, and the Glitch Editor. An example of this feature can be seen in figure 3.10.



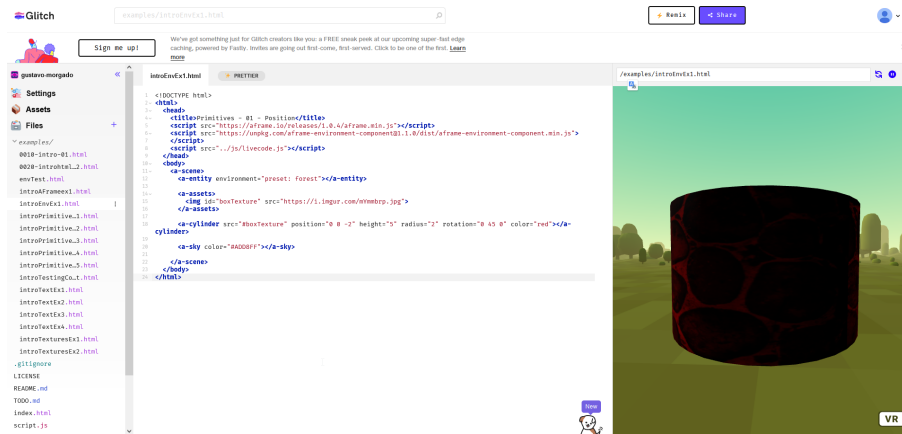


Figure 3.10: Live Preview in the Glitch editor

### 3.2.8 Extension Manager

Lots of editors now have extension managers. These managers host multiple extra functionalities that complement the editor, some of them are the ones mentioned below, and are created by either the community or the creators themselves. All the IDE present use this feature.

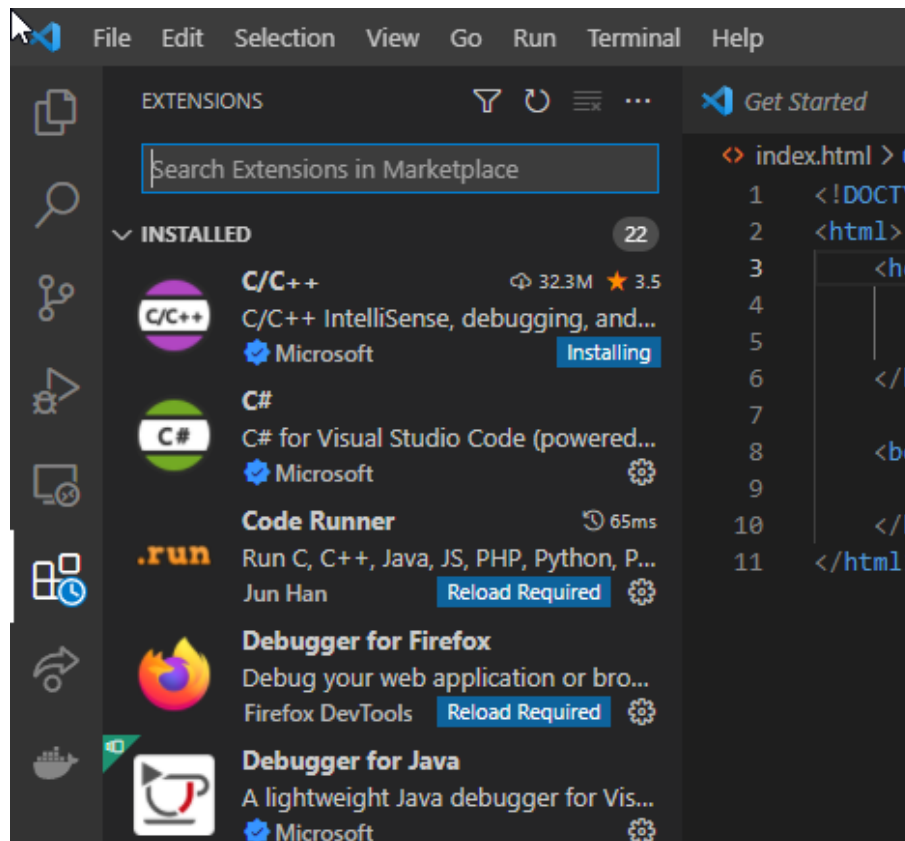


Figure 3.11: Example of an Extension Manager

### 3.2.9 Auto-completion

Auto-completion is a feature where the editor presents several keywords that match with what the programmer is writing. This feature is quite helpful since it speeds up the programming. This feature normally comes with a showcase of the starting arguments, so it also helps at reducing the number of interruptions. It also is one of the features that every IDE tested had. Studies have also been done on the auto-complete function, and its impact on beginner level programmers. It was determined that the use of an auto-complete function was useful to a beginner, since it allowed a positive correlation between the user's needs and the frequency of the use of auto-complete [12]. There are multiple types of auto-completion, which shall be explained in more detail.

#### Basic code completion

Basic code completion is the presentation of all possible options that correspond to what the user is writing, applied to classes, keywords, methods and fields [13].

```
} catch (NumberFormatException nfe) {  
  if (nfe) {  
    nfe  
    conversionFactor  
  }  
  ^↓ and ^↑ will move caret down and up in the editor Next Tip
```

Figure 3.12: Example of basic code completion [13]

#### Type-matching completion

Type-matching completion, in which the IDE will first look at the context of the code, showing more specific suggestions, that can be applied to the current context [13]. This type of completion is mainly useful on situations where it's possible to determine the type, more specifically:

- Variable Initializers.
- Return Statements.
- List of arguments of a method call.
- In the right part of assignment statements.
- After the **new** keyword, in an object declaration
- Chained expressions

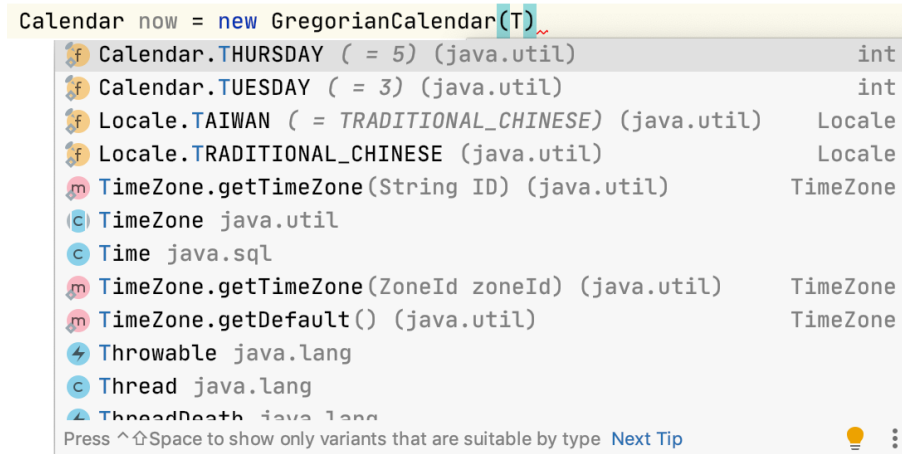


Figure 3.13: Example of type-matching completion [13]

## Statement completion

There is also statement completion, that involves inserting syntax elements (parentheses, brackets, and semicolons) on the correct position, and repositioning the position of the user's cursor to the next statement [13].

## Hippie completion

Hippie completion is a completion text engine that analyzes the text in the current scope, giving suggestion to the user based on the current context the open files of the project [13].

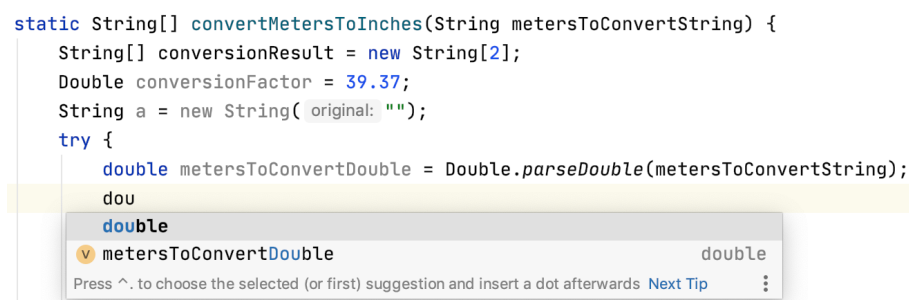


Figure 3.14: Example of hippie completion [13]

## Postfix code completion

Postfix code completion allows the user to put a dot after an expression, followed by a postfix [13]. By doing this, the IDE will wrap the expression based on the postfix used, transforming the expression that was typed into a new one.

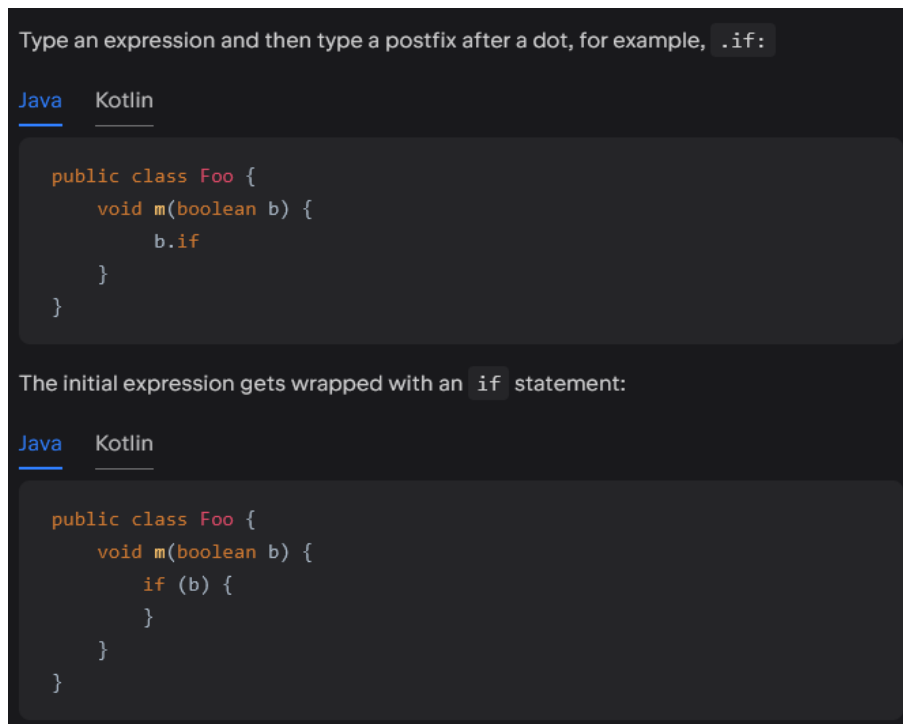


Figure 3.15: Example of postfix code completion [13]

## Completion of tags and attributes

This type of code completion will automatically complete names and values of tags and attributes [13]. In IntelliJ IDEA, this works based on the DTD or schema file attached to the file is associated with, and in case of the non existence of that schema, it will use the file content to complete the input. The files that this IDE supports are:

- HTML/XHTML.
- XML/XSL.
- JSP/JSPX.
- GSP.
- JSON

## Machine-learning-associated code completion

Machine-learning-associated code completion is based on giving completion suggestion based on what other users wrote in similar situations. This data is collected locally, and it's not public.

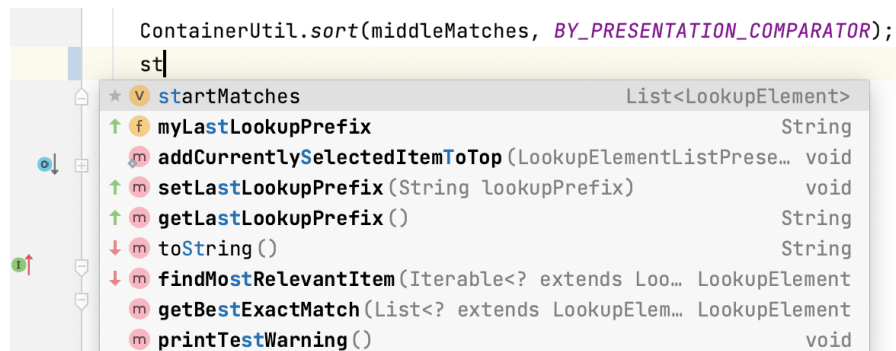


Figure 3.16: Example of machine-learning-assisted completion [13]

### 3.2.10 Highlighting

Syntax highlighting is a tool that helps differentiate keywords in code, turning it less monotonous to read and also shadowing less important parts, like comments. An example of this tool is shown in 3.17. This feature was present in all IDE used, including the Glitch editor.



Figure 3.17: Visual Studio Code Syntax highlighting

### 3.2.11 Color Picker

Color picker is one of the main features that exists in today's IDE, due to its massive help in automatically choosing the color that the user wants, instead of having to search the correspondent hexadecimal code and editing the values to become lighter or darker.

There are multiple ways to implement a color picker, either by having on the the tools of the IDE be a color wheel, that allows the user to select a color from the wheel and get the correspondent hexadecimal code, like in figure 3.18, or automatically

detect if the code has a color attribute, inserting in that place a button that opens up the color wheel, like in figure 3.19. The latter can also show up next to the correspondent line number.

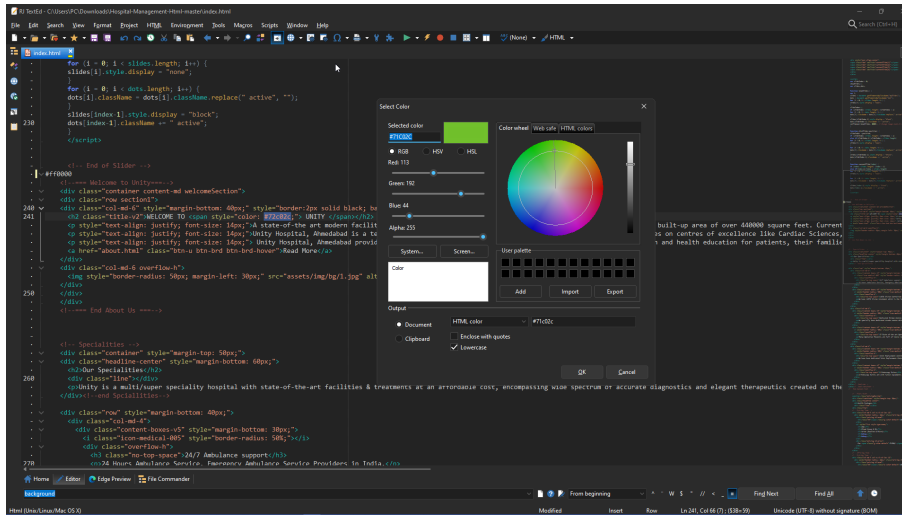


Figure 3.18: RJ Text ED Color Picker, that only opens up when the user opens the tool

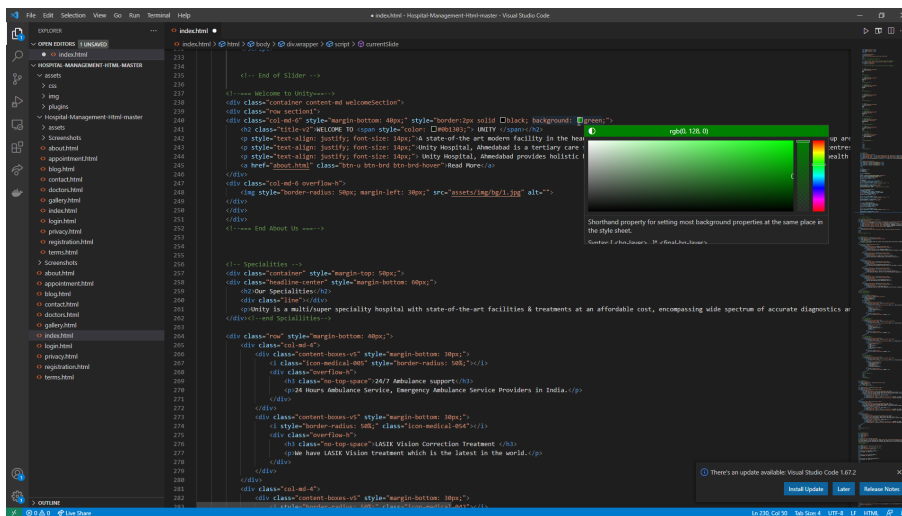


Figure 3.19: Visual Studio Code Color Picker, shows up immediately next to background attribute

To better analyze what features to implement on the tool, a table was created, to see what are the more common functionalities. The table 3.1 is an intersection of what features each editor has. From the table, we can see the most popular features are code folding, plugin manager, auto-completion, highlighting and color picker. With the exception from code folding, it was decided to add all those features to the tool, alongside the Tags Explanation feature, since can be helpful for beginning A-Frame programmers.

Functionalities											
IDE	Code Analyzer	Support for Web Frameworks	Multi-edit	Code-folding	Goto button	Tags Explanation	Live Preview	Plugin Manager	Auto-completion	Highlighting	Color Picker
VS Code	X	X	X	X	X	X	X	X	X	X	X
Atom			X	X	X		X	X	X	X	X
Sublime			X	X	X			X	X	X	X
PyCharm	X	X	X	X	X	X	X	X	X	X	X
IntelliJ Idea	X	X	X	X	X	X	X	X	X	X	X
PHPStorm	X	X	X	X	X	X	X	X	X	X	X
Komodo Edit			X	X				X	X	X	X
RJ Text ED			X	X	X			X	X	X	X
Brackets			X	X	X		X	X	X	X	X
AWS Cloud 9				X				X	X	X	X
Code::Blocks	X		X	X	X			X	X	X	X
Light Table			X	X	X			X	X	X	X

Table 3.1: Intersection between the IDEs and the features researched





# Chapter 4

## Project planning

The objective of this chapter is to present the aspects related to project management. They will be presented first by the methodology used, followed by the chosen technology, and lastly by the risk identification and evaluation.

### 4.1 Methodology

For the process of selecting a methodology, a study was conducted to compare the various methodologies currently available and determine the most suitable one for this project. After careful consideration, the chosen methodology was the Iterative Waterfall Model [14] due to the project's nature. Since the author will be undertaking the project independently, there is no need to implement team-focused software development methodologies. Furthermore, since there is no outside client, one of the disadvantages of the Waterfall and Iterative Waterfall model, which is the restricted customer interaction, will be negligible. Lastly, to ensure greater flexibility during programming, particularly in error correction, it was decided to follow the Iterative Waterfall Model. This model offers feedback pathways from each development phase, allowing the programmer to address and incorporate error corrections into subsequent steps. In contrast, the original Waterfall model lacks such feedback pathways.

### 4.2 Technology

Once the methodology was selected, a choice was made regarding the technology to be employed for project implementation. Due to the project working as an add-on to the Glitch editor, there were two options available. The first one was to create a browser extension, while the other option was to create a script that could be loaded by an existent extension called Tampermonkey [15], which allows the user to load scripts. After studying the two options, it was decided to follow the second option, since if it was chosen to create an extension, there would always be a risk of the browser changing the extension implementation method. There would also be the

need to take into account all of the existent web browsers and create an extension for every single one, in order for this solution to be easily reachable. Tampermonkey also has risks, being one of them the termination of the extension. However, since it's an already existent extension that works with the most popular web browsers, and has over 10 million users, it reduces the risk of not updating the extension, and making it so there can be a full focus on the script.

### 4.3 Tools

After deciding the technology, a list of tools to be used during the development of the project was created:

- **Tampermonkey:** Tool that will be used for the development for the project.
- **Sublime Editor:** Tool that was used to create the prototype.
- **Overleaf:** LaTeX editor that is used to create the documentation for this project.
- **Visual Paradigm:** Tool to create any chart needed for the planning of the project
- **GanttProject:** Tool to create the Gantt charts.

### 4.4 Planning

In the beginning of the semester, an initial meeting was made to introduce the author to the theme of the internship, followed by bi-weekly meetings whose purpose was to review the work that had been made, and see what should be done until the next meeting. The planning for both semesters is presented in the following Gantt charts, being respectively figure 4.1 and figure 4.2.

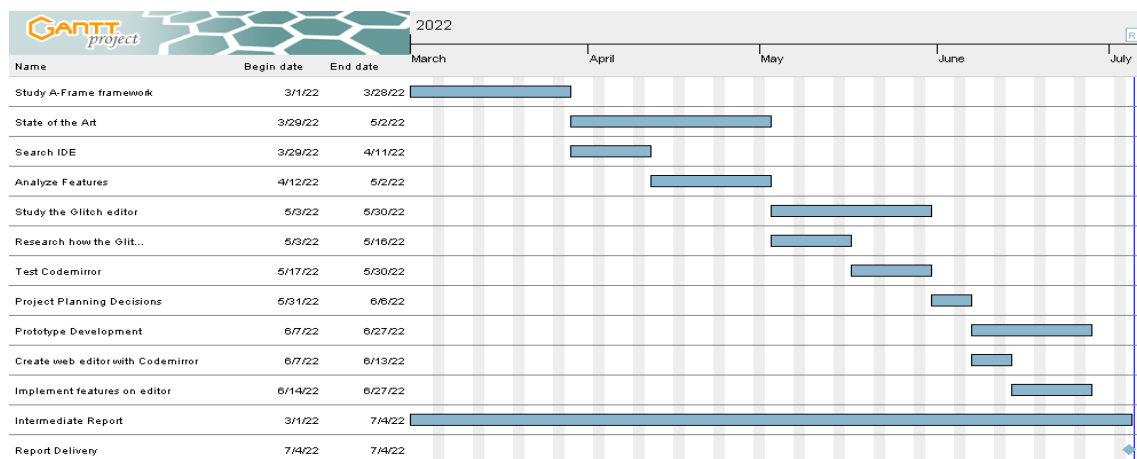


Figure 4.1: Gantt chart for the first semester

For the first semester the first task that was decided was to experiment with the A-Frame framework and the Glitch editor, to get some understanding of how it worked. After that, an extensive research was done on multiple IDE, and their functionalities. Next, a more through experimentation with the Glitch editor and it's inner workings were made. Following that, the decisions relative to the project in terms of features and implementation were made. Finally, a prototype was built that included some of the features that were talked about.

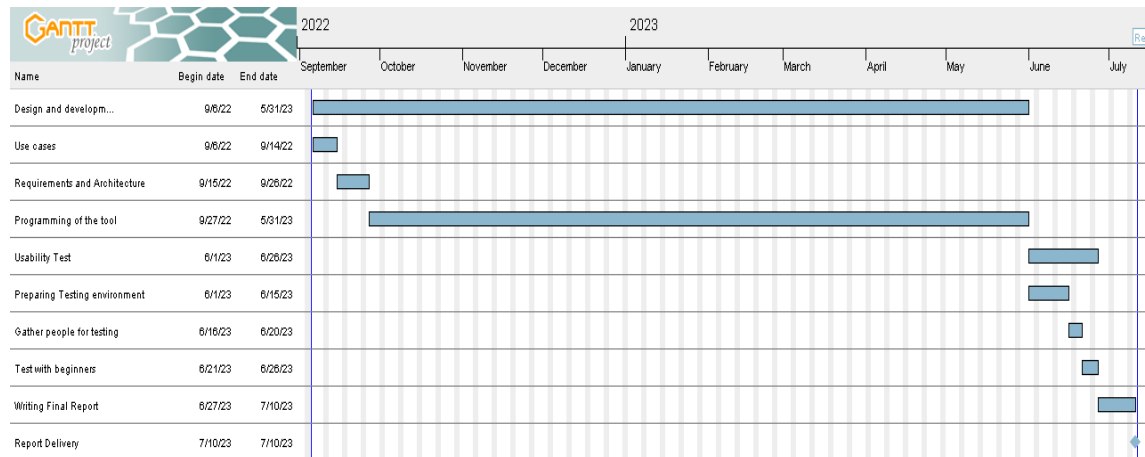


Figure 4.2: Gantt chart for the second semester

In the second semester, the first task was the design of the tool. It involved the creation of the use cases, followed by the production of the functional and non-functional requirements. After this, the development of the tool started. This process took a severe amount of time due to the author not being able to commit full-time to the project. Following that, the testing period started, where the majority of the time was spent gathering enough testers. The final period was dedicated to writing the final report.

## 4.5 Risk Planning

Risk planning is the process of identifying what can go wrong during the project. This will help handling some possible obstacles to the project that can interfere with its realization.

- **R1:** Incorrect task estimation time
- **R2:** Inexperience with the Glitch editor causes delays in the project
- **R3:** Problems that appear in the testing phase take more time to solve
- **R4:** Not enough people to run a usability test
- **R5:** Additional requirements are detected later

After identifying the risks, the next phase is determining the likelihood of those risks becoming real. It was divided in three categories, **High**, which means there is more than 70% chance of happening, **Low**, that means that there is a 40% chance of that risk appearing, and **Medium**, that will include the percentages between those two. After that, it was measured the possible impact of those risks happening, along with a mitigation plan that will allow the minimization of the impact made by the risks.

Risks			
Risk Number	Probability	Impact	Mitigation Plan
R1	Medium	Medium	Allocate time from other tasks
R2	Medium	Low	Read documentation and tutorials
R3	Medium	Medium	Allocate more time to development
R4	Low	High	Choose a longer period for testing
R5	Low	High	Pick requirements from the "Could have" section

Table 4.1: Risks

# Chapter 5

## System Description

This chapter presents all functional and non-functional requirements, as well as providing more context on the scope of the project through use cases and constraints. It will also describe the system's architecture.

### 5.1 Requirements

First, a decision was made on the requirements for the project. These requirements are divided in two groups. Functional requirements, which will define what the system does, and non functional requirements, which will describe the properties that the system will have. The features that were initially decided were the color picker, the documentation display, the component installer and the auto-complete.

#### 5.1.1 Survey

First, a survey was created, in order to understand how would these features be received by programmers of the A-Frame community. It also had the objective of helping decide what features should be kept, and find any other ideas for the script. The survey was composed of 9 questions, where 4 of them where about the features planned, followed by a mockup, where the person answering the survey would give their opinion on the feature and rate it from 1 to 6, with 1 being Very Bad and 6 being Very Good. The mockups are present in Appendix A, while the survey is present in Appendix B. Each of those question were then followed by an open question about suggestions to the feature, specifically in terms of design and access ability. Finally, there was a question about other suggestions that the respondents had in mind. After making the survey available for 1 week, it received 5 answers, due to some of the communities rejecting the request to post the survey, making it hard to reach. However, it still gave some valuable knowledge. Looking at the answers, they were pretty divisive. On the auto-complete, it was positively received, with every respondent giving a positive answer. For the other however, it's where the divisiveness started to appear. For the color picker, the respondents answered the two extremes of the scale, with 2 of them voting 6, and the other 3 voting 1 and

2, which gave the sense of being controversial. For the Documentation Display and the Component Installer, every respondent picked a different answer, leading to a difficult analysis. In terms of suggestions, only one was given, which was, instead of just showing the hyperlink, the system would show the documentation itself, making it more effective and simple. After looking at all answers, it was decided to proceed with the features, and keep the suggestions made.

## 5.2 Use Cases

Use cases are used in system analysis to describe the interactions and behaviors of the system from the perspective of an actor, which can be the user or a stakeholder. The list of all use cases are represented below. One thing to note is the idea of implementing the auto-complete was discarded. This was due to discovering that the Glitch platform didn't contain the required files for it to be activated, and also because it only worked for normal HTML, which would then make it useless for A-Frame. This was only discovered after the survey was done.

Name	Get live preview of coding
Actor	User
Description	User will enter the desired changes, thus seeing them instantly on the preview page
Preconditions	User must have some code already written and they have to have a preview page from Glitch open
Basic flow	1. User writes code
Postconditions	Preview will show immediately the results

Table 5.1: Get live preview of coding

For the live preview, as we can see from 5.1, the process is very simple. As the user writes code on the editor, the changes are made automatically, without input from the user. One of the preconditions is for the user to have a preview page open already. This is due to a need for communication between the two pages, and thus being able to communicate and exchange information.

Name	Pick color for object
Actor	User
Description	User will open the color picker by pressing Alt-C and select one of the colors
Preconditions	User must have an object that has the property color
Basic flow	1. User opens color picker 2. User uses the color wheel to select the desired tone 3. User clicks the apply button
Postconditions	The code now has the color that the user picked
Alternate Course	1.a Attribute isn't color 1.a.1 Color picker doesn't open

Table 5.2: Pick color for object

For the color picker, the interactions needed are simple. When a user desires to active the color picker, they simply need to have an existent object that has the color attribute, since without that the feature won't activate. After that, by activating the right hotkey, the color picker appears minimized, where the user can then click the hexadecimal tag that appears and change the color, maximizing the color picker, that then shows the color wheel.

Name	Get documentation of respective element
Actor	User
Description	By pressing Alt+D, a preview of the entry of the keyword in the documentation will appear. This preview will be the contents of the web page
Preconditions	User must be selecting a valid keyword from the original A-Frame framework
Basic flow	1. User opens the web page
Postconditions	The web page is displayed

Table 5.3: Get documentation of respective element

The use case 5.3 is also simple in terms of execution. A user will select with the cursor a keyword, which must be a valid primitive of the A-Frame frame work. This selection can also include the "<" sign. After activating the hotkey, a new browser window will appear, with the A-Frame documentation open on the respective entry.

Name	Install external component
Actor	User
Description	User will press alt-M and that will open up a search bar, which will contain multiple components available to the user
Preconditions	User must be in the head section of the HTML file
Basic flow	1. User opens the component Installer 2. User inputs the name of the component he wishes to install 3. User then clicks the Install button
Postconditions	The component is installed correctly

Table 5.4: Install external component

The final use case, the Component installer, is based on the user being able to install A-Frame components that exist on the npm repository. The user activates the hotkey, and a selection box appears. The user can then search for the component they want, either through scrolling through the options or writing the name of the component. After that, the user then clicks the install button and the tool applies the respective script element on the HTML file.

### 5.2.1 Functional Requirements

Functional requirements are features that must be implemented for the user to be able to complete a certain task. It is important for them to be clear, since they describe what the system should do when operating. To evaluate the priority of the

requirements it was applied the MoSCoW[16] method, which gives every requirement on of the following designations:

- Must Have - Requirements that are vital for the system
- Should Have - Requirements that add significant value
- Could Have - Requirements that are nice to have, and don't have an impact if left out
- Will not Have - Requirements that will not be implemented

After analyzing the features that were decided, it was then attributed those designations, which can be seen on table 5.5. Every one of them was given the Must Have designation, since all of them were vital to the tool.

Table 5.5: Requirements

Requirements		
Requirement ID	Description	Action
1	Preview changes to the code	Must
2	Choose wanted color	Must
3	Show documentation	Must
4	Install scripts/libraries	Must

## 5.2.2 Non Functional Requirements

Non functional requirements are the criteria that define the quality attributes for the software, rather than specific qualities of the system, focusing more on how well it should perform certain functions or qualities. The following requirements are ones that are crucial to the project and are considered a must have.

### Usability

The usability requirement focuses more on how intuitive and user-friendly the system is, which is a core aspect for programmers that are new to A-Frame. To ensure that this requirement is in practice, it will be tested with programmers that have a considerable amount of experience with the framework, since it will mean that they can identify possible problems with the interaction.

### Compatibility

The compatibility requirement is the capacity of the system to be compatible with different browsers. To make sure that this requirement is well implemented, the script will be tested on multiple browsers, while at the same time try to reduce the number of requests that can go against the policies that the browser has, like the Cross Origin Policy.



## Performance

The performance requirement is to ensure that the preview is updated smoothly, minimizing loading times. To ensure that this requirement is enforced, the number of operations will spread out, making it so the tool doesn't need to execute multiple functions at once.

## 5.3 Architecture

A system's architecture refers to the structure and organization of a software system. It allows an overview of all connections that the system might have, in order to manage the system complexity. The architecture is presented below, in the C4 model[17], which allows for a better view of the entire system, by providing different levels of depth. For this project, it was used 2 levels.

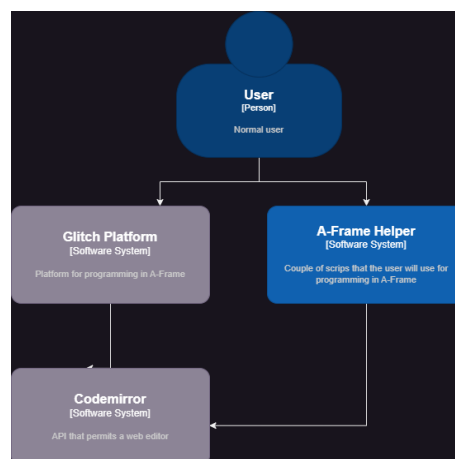


Figure 5.1: Software Architecture level 1

The first level is the more basic one, and it serves to provide a simple overview of how the system works. In this project, the user will be interacting with the Glitch platform and the Userscript simultaneously, since the purpose of the project is to complement the platform, not to replace. Other than that, both of them will interact with Codemirror, in order to execute their necessary actions.

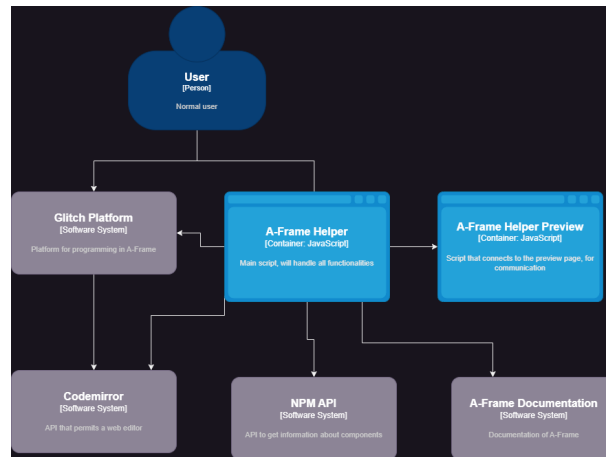


Figure 5.2: Software Architecture level 2

For the second level, more detail is shown about the project itself, and how will the functionalities be implemented. The tool will be divided in two Userscripts, in order to make changes to both web pages, the editor and the preview. The scripts will have to be connected to the npm API and to the A-Frame documentation page, in order to get the necessary data for the features.

# Chapter 6

## Development

This chapter is focused on the development of the tool, whose name was decided to be A-Frame Helper.

### 6.1 A-Frame Helper

The first step was to figure out how to communicate with Codemirror. By looking at how the prototype was made, a general idea was that Codemirror has a global variable that holds the editor itself, which then lead to searching that variable's name, in the case of the Glitch platform. After searching through the Glitch forums, the name was found, and after testing, communication was established. One problem that occurred during this was that sometimes the page would load too fast, which lead to the script not being able to recognize the editor, since it wasn't loaded yet. To fix this a wait function was added, to make sure that there was a viable Codemirror editor.

After this, the implementation of the functionalities began. First, it was decided to implement the more basic ones, before approaching the live preview.

#### 6.1.1 Color Picker

For the Color Picker, first there was a search for different types of color pickers that are available for Javascript. The one that was chosen was a script that involved color.js, which has the permissions to be added into other scripts. After importing it, it was a simple matter of adding it into an HTML element and inserting it into the page.

Next was the part of checking when to make the color picker appear. First, it was decided to run a lexical test on the selected keyword, to check if it's a color or not. However, after some consideration and use of the tool, it proved to be a bit bothersome having to highlight the color field repeatedly. Because of that, instead of highlighting the color, the user now only had to open up the color picker in the correct place, which was after the "=" sign, and also the color attribute must be

before it. After adding those checks, the lexical test was scrapped, since it wasn't needed anymore, and now the tool automatically replaces the previous color if it existed. The end result is shown in the figure 6.1.

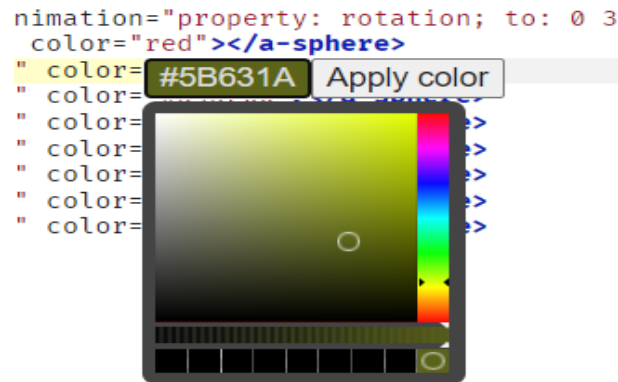


Figure 6.1: Example of the color picker being used

## 6.1.2 Documentation Display

For the Documentation Display, there were some changes from the initial mockups. The suggestion that was received from the survey said that the documentation should be shown immediately on the web page, instead of just showing the link. After some brainstorming, one of the ideas was to make an iframe appear, with the content of the corresponding element appearing on the frame. However, some problems occurred while implementing this idea. One was that the script didn't have permissions to insert the A-Frame documentation on the iframe, due to the Cross Origin Policy. One idea to bypass this was to make a http request to the documentation website, and receive the corresponding HTML. However, the result was that the CSS wasn't displayed, which lead to a non-visually appealing feature.

Since the results weren't what was desired, it was decided to brainstorm another way of showing the documentation. After some consideration, it was decided to simply open up a window with the documentation, instead of the previous idea. This fixes the problem that the previous implementation had, and surpasses it, since it allows the user to resize the window as they like it, or move it to another monitor.

In terms of how the element is detected, the process is similar to the color picker. It will check what has been selected, and if it coincides with the existing A-Frame elements, then it will open up a website that links directly to the page containing that element. To make sure that there is no need to update the script regularly, the tool will open the website on the master version. To get the correct link, the tool adds the name of the element that the user desires to search about to the base link of the A-Frame documentation website, which is "<https://aframe.io/docs/master/primitives/>". A final version can be seen in the figure 6.2.

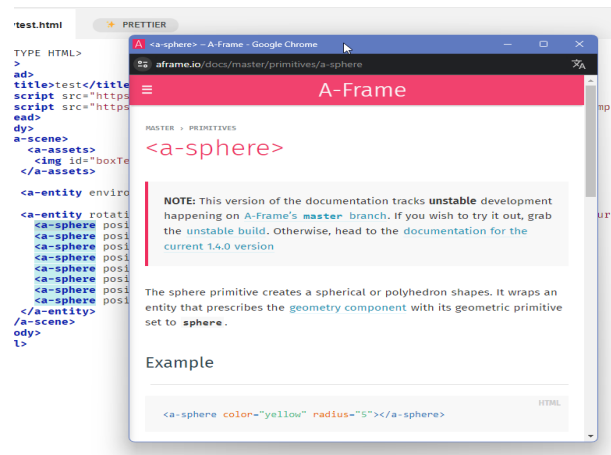


Figure 6.2: Example of Documentation Display

### 6.1.3 Component Installer

For the Component Installer, first it was made a research on how to access the npm API. From that, some relevant endpoints were recorded in order to access the information, mainly the one that gives all package information that resides in npm. This proved to be quite exasperating, since the npm repository deprecated those endpoints. However, after some research, it was discovered a hidden endpoint that was still in use, on `http://registry.npmjs.com/-/v1/`. After that, a query was created in order to only receive the packages that were related to A-Frame, which was done by adding the term `search?text=a-frame` to the url, resulting in the url `http://registry.npmjs.com/-/v1/search?text=a-frame`. That data was then parsed from JSON, storing into an array all the names of the packages, and showcasing that array into a select box.

By using a select box, when testing it, all the results were shown. However, the user could not search by a specific term, which made it inconvenient to use, especially because there were a numerous amount of components. To fix this, the select box was switched to a datalist, that allows the user to search for the component that they want. To install the components, it was simply adding to the editor variable the html line with the corresponding unpkg link. Since the API doesn't provide the link itself, it has to be crafted, by following the normal template that npm packages have, which is `"https://unpkg.com/"`, plus the name of the package, plus `"/dist/"`, plus the name of the package again, plus `".min.js"`. the final result can be seen in the figure 6.3.

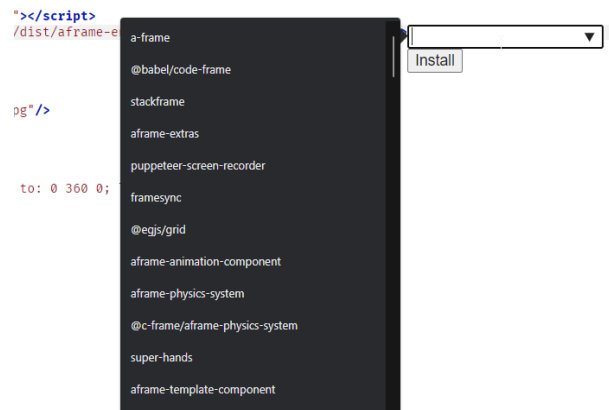


Figure 6.3: Example of Component Installer

### 6.1.4 Live preview

Finally, the development of the live preview started. This was the most problematic task, which involved several setbacks, where some of them needed rethinking. First, there was an attempt to make it as the prototype, by changing the HTML inside of the iframe of the preview. This idea was scrapped since first, as it was written above, the browser didn't allow for communication between iframes due to policies, and second, since the users could want to view the preview in another tab, the iframe method wouldn't work anymore. So instead, another way of communicating was found, which was by sending messages between two web pages, the Glitch editor and the preview page. For this to work, another script had to be created, in order to handle the communication on the preview side.

After establishing messages between the two scripts, it was time to work on the algorithm. Two event handlers were created, where both detected changes that were made to the editor, where one of them showed the state of the editor before and the other one showed after. The reason for this is so that the tool can know, in the case of a user deleting an attribute, where was the deleted section in the Codemirror, since the object doesn't save the range of the deletion after the change was made, only before.

For the algorithm, the final version of it involved doing multiple verification on the fields of the element, that will be treated as tokens. When first detecting a usable token, it would be called into action, receiving the position of the token as arguments. After that, it would start backtracking the code, counting the number of open and closed brackets, in order to determine the indentation, until it reached the body or head tag. It would also take track of the number of sibling elements that were being called first, to get the position. This way, when sending the path to the preview page, it could correctly access to the element being changed.

Next, it checks the corresponding tokens of the line where the user was writing, and filters out the irrelevant ones (tag brackets, null tokens, etc.), keeping the attributes and values of the HTML element saved in an array, with the tag name. This is then sent to the preview script, where that data is read and added into the data tree of

the project. Down below is the code of when the A-Frame Helper finds a tag bracket in the tokens, more specifically the bracket that indicates the end of an element.

```

1   if(allTokens[i].type === "tag bracket"){
2       if(allTokens[i].string === ">"){
3           var arr = [];
4           if(first){
5               console.log("entering higher limit same line");
6               arr = findHigherLimit(curLine,i-1);
7               if(arr[arr.length-1] === "invalid tag"){
8                   return;
9               }
10          arr = arr.concat(token);
11      }
12      if(token.length != 0){
13          token.push(elementCounter(curLine,i));
14          arr = arr.concat(token);
15          tokensToChange.push(arr);
16          token = [];
17          arr = [];
18          tokenPath.push(pathParser(curLine,i));
19      }
20      continue;
21  }
22  else{
23      continue;
24  }
25  }

```

It first starts by checking if it's the first token that has been checked. If so, it will then go backwards in the code, through the function `findHigherLimit`, and add every previous token that belongs to the element, until it finds its initial tag. If it returns "invalid tag", that means it's not eligible for preview, so it ignores the changes made. After joining those tokens to the selected one, it will then check if there are any tokens present on the original token array, that contains all previous tokens added, before entering this bracket. If the check is true, then it will add at the end of the array the position of the element in the hierarchy of the code, and also add the corresponding path to the list of paths. Finally, it will then continue and analyze the next token, until it reaches the end of the committed changes.

For the preview to know when to delete, add or change an object, another algorithm was created to count the total amount of elements there were at the same level of the change. The logic was similar to the previous verification, where the tool would backtrack the code and analyze the elements, however this one has the capability of searching down the code, stopping when there were more closed brackets than open brackets, when searching downwards, and the opposite when searching upwards. This number was then sent to the preview, where it would be compared to its counterpart, making it clear which of the operations were made. An exception was made to the a-scene element, since when calling it, it automatically creates other elements inside it, which led to having a special check in the code for that specific scenario.

The tool also has multiple checks, in order to address special cases like per example,

separating a single HTML element in multiple lines of code. These checks were made by creating specific functions that check the upper and lower limit of the token that is being seen. They also automatically add possible unseen tokens, that are out of range of the recorded changes. With this, it could be assured that all affected elements would be detected, which could then be sent to the preview page.

In the sending process, there would be a list of the elements and their respective paths, which would be sent one by one, in order to avoid issues when handling multiple actions at once. This helps because then the preview then only needs to care about adding, deleting or changing one element.

For the preview part, after receiving all that data, it now needs to parse and make the changes to the HTML tree. Since the data is all sent in a array, ready to be treated, the process is simple. It first reads the path, starting to cycle through the elements in the body part, thanks to one of the HTML default methods, that allows to ignore null elements. After it reaches the intended node, the tool then reads the first box of the array that contains the variables, called "elements". The first index contains a strings that says if it has to delete or to change. If it's the first case, it simply removes the node. If it's the other, then it does the calculations explained in the second last paragraph, and performs the changes.

This feature, while functional, has some limitations. One being that it can't update scripts and assets in real time, since for the first one it's impossible to predict how many elements does a script contain when loading, while the other one is pre-loaded by the browser, which means that it can't be changed. Also, it only recognizes elements if they have the correct closing brackets (`</>` or `</>`), since it's needed for counting the elements. This means that if the user inserts an element that doesn't contain those brackets, it will produce errors that will affect the entire process, since this tool only requires one simple error on adding to the tree to start impacting other future changes. Finally, since it requires a connection between the two web pages, when too much time has passed, they disconnect, requiring the user to then have to manually create another preview page.



# Chapter 7

## Testing

This section is about the testing process used on this project. It will explain the usability test conducted, where other users participated and used the tool, along with its results and analysis.

### 7.1 Usability testing

For the usability testing, it was divided in three phases. The planning of the tests, the execution of the tests, and the analysis of the results.

#### 7.1.1 Planning and Procedure

For the planning, first there was a brainstorm on what kind of task to give to the users. It couldn't be too simple because the tool would be barely used, but it couldn't be too complex because it would be too cumbersome for a test. In the end, it was decided to make the users create a solar system with A-Frame, with the help of some components. This task would be complemented with some rules, to incite the users to use the features of the tool, which were:

- The planet Earth had to have a texture, whose link was provided, and the other planets must have solid colors, which should be similar to their real life counterparts.
- The planets in the solar system must make a rotation around the Sun and be able to spin.
- When the sun was clicked, the animations must stop, which would require the `afree-event-set-component`.
- The user also had to implement an environment that complemented the system, with the component `afree-environment-component`.

Other than those guidelines, the testers had free reign to complete the task however they desired, to have a larger detection range of problems with the live preview.

Before the task, the testers were presented with an explanation of the tool and how to install it, along with an initial survey to get a hold of the experience of the user in A-Frame. After the task was done, the user would then answer another survey, this time about the tool. The survey first uses the System Usability Scale (SUS)[18] template questions, which focuses on a quick way to evaluate the usability of a system. One of the major key points which helped choosing this template was that it could be used on small sample sizes, and still get reliable results. After that, they had to answer specific questions about the different features the tool has, which were about the simplicity, usefulness, and how easy it made the programming process of the respective functionalities. These questions also followed scale from 1 to 5, where 1 would be the worst option and 5 the best option, followed by an open question about any problems found. At the end of the questionnaire, they could give suggestions and comments in an open question box about future improvements.

Initially, it was intended for the tests to be open only for people with experience in A-Frame, to see the tools being used more efficiently. However, there weren't enough testers with that kind of experience, so the range expanded to people that didn't have experience in A-Frame. Due to this, the task was simplified for the testers with no experience. This was made by making the guideline that required the `aframe-event-set` optional.

### 7.1.2 Results/Discussion

There were 5 testers for this tool. This number was picked because it was getting harder to notice any new problems the more it was tested, thus stopping the tests. All the testers had programming experience, however only one had experience in A-Frame, since he used it in his job. They also were either employed or writing thesis, which makes sense given that they were around 24 years old.

In terms of the results, first the SUS was analyzed, using the respective method, which involves giving a score from 0 to 4 to all questions. These scores were obtained by looking at what answer did the user pick, which had a 1 to 5 scale, ranging from Strongly agree to Strongly Disagree. For the questions 1,3,5,7 and 9, it was the answer minus 1, while for the questions 2,4,6,8 and 10, it was 5 minus the answer, since the questions had a negative nature[19]. Lastly, it was made the sum of all scores, and then multiplied by 2.5, to convert the original score of 0-40 to 0-100. After getting the scores, it was also needed to make an average of all scores of the testers. All of this resulted in the score of 87 in 100, which was then compared to the adjective ratings that were created for the SUS. According to these ratings, A-Frame Helper has excellent usability, which is the second highest rank possible.

Next was the analysis of the questions about the features. To keep things simple, it was decided, for every group, to attribute a score of 0 to 4 to the questions, and then add them up. After adding them, we had to make the average score of that group, to get the general answer. This process was then repeated for every question. After making this calculation, it was then recorded, to make an analysis of the success of

the features.

Results			
Feature	Useful	Simple	Easier
Live Preview	4	3.6	3.6
Color Picker	4	4	4
Documentation Display	3.6	3.2	3.6
Component installer	3.8	3.8	3.8

Table 7.1: Average of the results of the questions about the features of A-Frame Helper

As we can see in the table 7.1, the color picker was an extreme success, managing to get maximum positive reviews from every user. The other features were also received positively, with all participants exclaiming that they would recommend the use of this tool. The less popular one was the Documentation Display, which was mainly due to a problem while using a different browser that couldn't handle http requests from Tampermonkey, which made the feature simply unavailable for use. Looking now at the suggestions made by the testers, there are some interesting ideas, and problems that had to be fixed. Interesting enough, most of the problems the testers had with the tool was with the help button, since in some laptops it was overriding with some of the buttons of Glitch, and the text color used was not the most appealing. Also, some of the content in the help button was lacking, specifically on the Documentation Display and the Component installer.

In terms of the features, as it was expected, some bugs appeared when using live preview. Some were easily noticed, along with the respective cause, which were then promptly fixed. Others were more difficult, including one where if you changed the planet rotation, instead of changing the axis, it would use both, which was fixed by updating another field of the element. Some bugs however were probably because of Glitch itself, since they didn't get fixed even after a hard refresh on the preview page, and the feature was working on the same browser in all the other testers. For the other features, no bugs were detected, aside from the difficulties when using the Opera GX browser, which was the browser that had the problems mentioned in the paragraph above, alongside not being able to search for the desired component in the component installer. However, the user could write the name of the component and it would still install, which gave some degree of usability. Of the other suggestions, one was implemented, which was more liberty in activating the color picker, since the tester claimed it was frustrating having to activate it on one position, instead of just on the color itself, which after some consideration, was then added to the tool. The other was an expansion of the features available, like a diagram to choose position and rotation. However, due to having a shortage of time, these features cannot be added currently. The last one was a change to the hotkey to summon the Documentation Display, since the current one (Alt+B) didn't make much sense. After some thought, it was changed to Alt+H, of "help".

Overall, the tool was received with major success, with the results indicating that it was easy to use to people without experience in A-Frame, and even helped them program with it, even though the majority of them were beginners. It showed to be easy to use and useful at the same time, which is crucial for these types of

programmers, and it was also simple enough that they can get used to it's features pretty easily.

# Chapter 8

## Conclusion

The main topic of the project was about the implementation of a coding assistance tool for the VR framework A-Frame, that would be able to work with the Glitch platform. Before starting the implementation, it was studied how the A-Frame framework worked, starting out by solving multiple A-Frame exercises, then later trying to write a new component. After that, a review of the state of art was carried out, starting with a research on some of the present IDE in the market, followed by an analysis on their features. Next, a more deep insight into the Glitch editor was made, in order to analyze on how the editor worked and how can it be possible to create a script.

Following the study, that stage was the project planning stage. First, it was decided on what methodology should be used for the creation of the software. Next, the decision between creating a browser extension and a user script was made, the final decision being the script for a already existing web extension. After that, it was outlined the project timeline, through Gantt charts. Next, it was decided the tools used for the project, which were Tampermonkey and Sublime. Finally, it was decided the extra features to implement in the tool, in addition to the live preview feature. The features that were decided were a color picker, a new windows that displays the documentation entry of the element, and a manager for components made by the community. From that stage onwards, the requirements were finalized, along with their use cases. Other stages of software engineering were also implemented, like the software architecture, to ensure a more smooth programming process. Next, the tool was developed, using the knowledge acquired previously. Some tests were made to verify the functionality of the tool, along with an usability test to see how the tool performed in action.

In terms of the future, the author decided to make the tool open source, and give permissions for anyone to use it. The tool will be accessible through Github, and also through GreasyFork, for easier installation. For future improvements, more features can be added, like a diagram to change the value of attributes such as position and rotation, in order to make the programming process even more flexible.

In conclusion, most of the goals of this project were reached, even though there were major setbacks. However, due to support of the advisor teachers, some of them were overcome, which contributed to the growth of the author in terms of knowledge in

programming, since it's through mistakes that we learn and evolve. One important piece of knowledge that was learned was the importance of project planning, since due to the author working on this project alone, instead of a team, there were no roles like Scrum Master or Product Owner, that could help with the project planning and task management. In the end, although it was tiring, it was overall a rewarding experience learning new tools and strengthening some of the author's weak points.

# References

- [1] A-frame. <https://aframe.io/>. Accessed: 2022-03-10.
- [2] Three.js. <https://threejs.org/>. Accessed: 2022-03-16.
- [3] Entity component system. <https://aframe.io/docs/1.3.0/introduction/entity-component-system.html>. Accessed: 2022-03-20.
- [4] A-frame introduction. <https://aframe.io/docs/1.3.0/introduction/>. Accessed: 2022-03-10.
- [5] A-frame primitives and elements. <https://aframe.io/docs/1.3.0/introduction/html-and-primitives.html>. Accessed: 2022-03-10.
- [6] Glitch editor. <https://glitch.com/>. Accessed: 2022-03-10.
- [7] Codemirror. <https://codemirror.net/5/index.html>. Accessed: 2022-05-10.
- [8] Html mixed - codemirror. <https://codemirror.net/5/mode/htmlmixed/>. Accessed: 2022-06-09.
- [9] Events - codemirror. <https://codemirror.net/5/doc/manual.html#events>. Accessed: 2022-06-20.
- [10] IntelliJ idea code analyzer. <https://www.jetbrains.com/help/idea/running-inspections.html#run-inspections-manually>. Accessed: 2022-04-20.
- [11] Creating and run your first django project with pycharm. <https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-django-project.html#create-project>. Accessed: 2022-04-28.
- [12] Motoki Miura. Effect of auto-complete function on processing web ide for novice programmers. In *13th International Conference on Knowledge, Information and Creativity Support System*, pages 166–171, 2018.
- [13] IntelliJ writing source code. <https://www.jetbrains.com/help/idea/working-with-source-code.html>. Accessed: 2022-04-20.
- [14] What is iterative waterfall model. <https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/>. Accessed: 2022-06-02.

- 
- [15] Tampermonkey. <https://www.tampermonkey.net/>. Accessed: 2022-06-03.
- [16] Moscow prioritization. <https://www.productplan.com/glossary/moscow-prioritization>. Accessed: 2022-09-17.
- [17] C4 model. <https://c4model.com>. Accessed: 2022-09-18.
- [18] System usability scale. <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. Accessed: 2023-06-10.
- [19] System usability scale analysis. <https://uxpajournal.org/determining-what-individual-sus-scores-mean-adding-an-adjective-rating-scale/>. Accessed: 2023-06-28.



# Appendices



# Appendix A

## Mockups

In this appendix, the complete list of mockups is displayed.

```
introPrimitivesEx5.html  ✨ PRETTIER

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Primitives - 01 - Position</title>
5     <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
6   </head>
7   <body>
8     <a-scene>
9
10    <a-box
11      position = "0 2 -2"
12      height="1"
13      width = "1"
14      depth = "1"
15      color=|
16    ><a-spher
17    <a-sphere
18    <a-sphere
19    <a-sphere
20
21    <a-sky co
22
23    </a-scene>
24  </body>
25 </html>
```

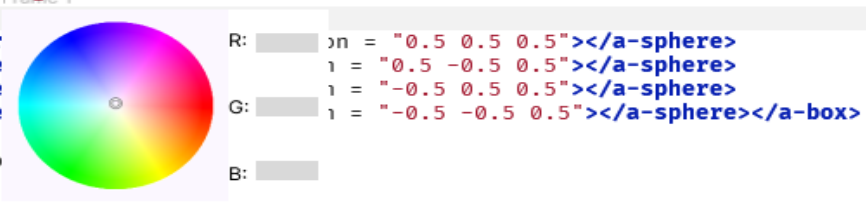


Figure A.1: Mockup of the Color picker

```

<!DOCTYPE html>
<html>
  <head>
    <title>Primitives - 01 - Position</title>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box >
        <a-s radius = "0.1" position = "0.5 0.5 0.5"></a-sphere>
        <a-s color radius = "0.1" position = "0.5 -0.5 0.5"></a-sphere>
        <a-s height radius = "0.1" position = "-0.5 0.5 0.5"></a-sphere>
        <a-s width radius = "0.1" position = "-0.5 -0.5 0.5"></a-sphere>
      </a-bo depth

    <a-sky color="#ADD8FF"></a-sky>

  </a-scene>
</body>
</html>

```

Figure A.2: Mockup of the Autocomplete function

```

<!DOCTYPE html>
<html>
  <head>
    <title>Primitives - 01 - Position</title>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
  </head>
  <body> https://aframe.io/docs/1.3.0/core/scene.html
    <a-scene>
      <a-box
        position = "0 2 -2"
        height="1"
        width = "1"
        depth = "1"
        color="red"
      >
        <a-sphere radius = "0.1" position = "0.5 0.5 0.5"></a-sphere>
        <a-sphere radius = "0.1" position = "0.5 -0.5 0.5"></a-sphere>
        <a-sphere radius = "0.1" position = "-0.5 0.5 0.5"></a-sphere>
        <a-sphere radius = "0.1" position = "-0.5 -0.5 0.5"></a-sphere>
      </a-box>

    <a-sky color="#ADD8FF"></a-sky>

  </a-scene>
</body>
</html>

```

Figure A.3: Mockup of the Hyperlink function



Figure A.4: Mockup of the Component manager



# Appendix B

## Testing Script

Guia de Testes

Apresentação:

Agradecer ao participante;

Descrição breve da ferramenta:

Esta ferramenta serve para ajudar no processo de programação de A-Frame na plataforma Glitch. Ela inclui suporte para live preview, onde o utilizador pode ver as suas alterações em real time, um color picker, um documentation displayer e um script installer. Para terem esta ferramenta, têm de ter o tampermonkey instalado

Disclaimer: este teste é para testar a ferramenta, por isso pedia para vocês usarem o mais possível as funcionalidades

Mostrar como instalar os scripts:

1º Instalar Tampermonkey se não já estiver instalado

2º Adicionar A-Frame Helper Script e A-Frame Helper Preview Script

3º Verificar se os scripts estão ativados

Breve descrição de como vai ocorrer os testes:

Primeiro vai ser feito um questionário inicial para ter uma ideia da experiência do tester em A-Frame, depois vai ser feita uma tarefa com potencialmente 45 minutos, que vai ser descrita mais à frente, para poder testar a ferramenta, e no fim vai haver um questionário final onde o tester avalia a usabilidade da ferramenta e relata problemas que encontrou ou sugestões adicionais.

Enviar projeto no questionário

QUESTIONÁRIO INICIAL(5 min):

<https://forms.gle/sbCYrWijSHStWAS47>

Tarefa a realizar(até 45 min):

For this test, you will be invited to create a representation of the solar system on

the Glitch platform using A-Frame. The following guidelines are encouraged to be followed:

-The planet Earth must have a texture, whose link will be provided. The other planets must have solid colors, which should be similar to their real life counterparts.

-The planets in the solar system must make a rotation around the Sun and be able to spin.

-When the sun is clicked, the animations must be stopped. This can be done with the component `aframe-event-set-component`

-The user must also implement an environment that complements the system, with the component `aframe-environment-component`.

Apart from those rules, there will be no limits to how the system is created, which means the user can be free to build it however they like it, if it doesn't exceed the time limit.

QUESTIONARIO FINAL(10 min):

<https://forms.gle/wwLMob8a5gjg3L7G6>