FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Authoring tool for multiplatform interactive digital storytelling

João Cardoso

DISSERTATION PLANNING

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Leonel Morgado

Second Supervisor: Prof. Maria van Zeller

July 28, 2023

Authoring tool for multiplatform interactive digital storytelling

João Cardoso

Mestrado em Engenharia Informática e Computação

July 28, 2023

Abstract

The creation of interactive editors for 3d digital stories is an area which has been widely explored in the last couple of decades, leading to their application in many different areas. This thorough exploration of techniques has allowed the creation of interactive story editors applied in many different areas, such as education and video game development. Despite that, the application of this area in the context of historical recreation is still widely unexplored. Most implementations did not require prior programming knowledge, however, the vast majority of implementations did not include the ability to use the same created story in multiple different scenarios. This dissertation focused on the development of an interactive 3D story editing web application for non-programmers as end users, applying previous technologies and techniques to this new area, including techniques for virtual choreographies developed at INESC TEC (Institute for Systems and Computer Engineering, Technology and Science). The application has the goal of allowing historians to, without external assistance, create stories that allow the displaying and discussing of historical doubts and conflicting sources of information more effectively. The application was applied to the project FronTowns, whose goal is a 3D historical recreation of two medieval towns on the border between Portugal and Castille. This dissertation details multiple prototypes of two different applications: A story editor and a story player. The story editor allowed users to create stories that could be played using the story player. It also describes the development process, from requirements definition and the research on the state of the art, to the development and evaluation of the multiple prototypes. This development was done in iterations, using the DSR (Design science research) (Hevner, 2007) methodology. The results from this thesis will contribute towards the acceleration of historical understanding by enabling a faster and cheaper method of creating virtual 3d historical recreations which highlight points of doubt in them. It may further contribute for historical dissemination in wider contexts, such as education and public awareness.

Keywords: Interactive storytelling; Interactive narrative; Authoring tool; no-code

Acknowledgements

Firstly, I would like to thank my family, especially my parents and my sister, for helping me all throughout my life. I would like to thank my friends, who gave me the support I needed to complete this course. I would like to thank my thesis advisors, professor Leonel Morgado and professor Maria Van Zeller, as well as the members of the FronTowns project, especially Demetrius, for providing this opportunity and assisting me with this dissertation and with the associated work. I would also like to thank the members at INESC TEC for allowing me this opportunity. This work is financed by the ERDF –European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT -Fundação para a Ciência e a Tecnologia, within project PTDC/HAR-HIS/3024/2020

Contents

List of Figures

List of Tables

Abbreviations and symbols

API	Application Programming Interface
DSR	Design Science Research
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
IoT	Internet of Things
PICOC	Population, Intervention, Comparison, Outcome, Context
xAPI	Experience API
2D	Two Dimensional
3D	Three Dimensional

Chapter 1

Introduction

1.1 Background

Storytelling is the process of creating or engaging with narrative structures (Sharaha and Dweik, 2016). It is both a long-established cultural tradition (Lugmayr et al. 2017) and a form of communication and learning (Tharrenos et al, 2015). Storytelling exists through many mediums, such as oral, written or visual (Riedl and Bulitko, 2013) and in many forms, such as physical or digital, interactive or not.

Interactive stories are a form of digital narrative which attempts to engage the player (Riedl et al. 2011), where the user can affect the way that the story progresses (Kim and Kim, 2016). This interactivity ranges from directly presenting multiple choices for the player to select to the use of an open world where the player's actions impact the story.

Given the widespread use of stories and their ability to convey information, many tools have been created to allow people to more easily and effectively create narratives, whether physical or digital. Authoring tools fit into many categories, including simple text editors (e.g.: Notepad++ [1]), video editing tools (e.g.: Sony Vegas Pro [2]), ones based on slideshow presentations (e.g.: Pow Toon [3]), ones which represent stories as graphs (e.g.: Twine [4]) and others. Authoring tools have been used in many contexts, such as education (e.g.: Subramaniam & Gokhale, 1995), video games (e.g.: Göbel et al. 2008), personalized visitor experiences (e.g.: Not & Petrelli, 2019), virtual reality (e.g.: Winterbottom and Blake, 2004) and video game development (Moallem & Raffe, 2020, pp. 305-311). Despite that, the application of this area in the context of historical recreations is still widely unexplored.

The authoring tool that was developed in this dissertation was part of the FronTowns [5] project. The project intends to discover the role employed by a group of small villages in the frontier between Portugal and Castille from the thirteenth century until and including the sixteenth century.

[1]Notepad++: https://notepad-plus-plus.org/

[2]Sony Vegas Pro: https://www.vegascreativesoftware.com/pt/vegas-pro/

[3]https://www.powtoon.com/

[4]http://twinery.org/

[5]https://frontowns.fcsh.unl.pt/

This project intends to create accurate 3D historical reconstructions of these villages. When presented, these reconstructions should include information points, which highlight doubts that the historians have about certain details of the reconstruction. Considering this, part of the process of creating the authoring tool was obtaining the requirements from the historians, whether that's through testing using a mock-up of the final application or through a simple meeting.

1.2 Objective

The goal of this dissertation is to contribute towards the acceleration of historical understanding by enabling a faster and cheaper method of creating virtual 3d historical recreations which highlight points of doubt in them. This dissertation details the development of an interactive story creation tool, together with a story player which runs stories exported from the authoring tool. The goal of the application is to allow people without prior development knowledge to create stories which can then be played in a three-dimensional scenario.

There are two main usage scenarios for the application. First, and most important, to provide historians the ability to freely create virtual reconstructions of narratives. These narratives have the goal of being highly historically accurate to enable the testing of historical hypothesis inside the tool. This can prove useful, for example, when testing conflicting hypothesis or verifying accounts of historical scenarios. Another context the tool can be used in is the authoring of (primarily fictional) narratives as a teaching tool. Teachers could utilize the authoring tool to create interactive scenarios where students can learn about a certain subject. For the first context, being able to utilize the same story in different scenarios/landscapes is a must, as historians must be able to utilize different versions of the scenario because the overall consensus of the landscape of the place at the corresponding historical period changes over time and, as such, so must the historical recreation. In both contexts, in order for the authoring tool to provide value, the stories have to be able to be created quickly. The authoring tool has to follow certain requirements. These requirements include:

- Enabling the creation and editing of interactive multimedia stories by non-programmers.
- The story editor must work in a web environment.
- Stories created inside the application must be usable in different 3D environments. This is enabled by the use of a virtual choreography standard developed by INESC TEC (Cassola et al. 2022).

1.3 Thesis outline

In addition to this Introduction (chapter 1), this dissertation contains 4 more chapters. Chapter 2 contains a description of some innovations in the creation of storytelling tools. In addition, it describes the process of executing the systematic review. Finally, it contains an analysis of which functionalities should be included in the authoring tool. Chapter 3 details the methodology, the development process, and the different prototypes of the applications developed for this thesis. Chapter 5 includes a short summary of the previous chapters, whilst also detailing the goals this project obtained and the reasearch questions it has answered. Chapter 5.3 details functionalities and parts of the development process that were not implemented or executed, but could be useful for future applications and research. Appendix A provides definitions for terminology used throughout this dissertation. Appendix B provides screenshots of the final prototype of the story editor created for this project. There is no annex for the final version of the story player, as there were no necessary images that were not included in the rest of the thesis. Appendix C displays the file structures used in the final story editor prototype and in the JSON files exported from that story editor.

Chapter 2

Literature Review

This chapter describes the review process, along with the results from that process. Before the review process began, there were a few questions posed from prior meetings:

- What degree of freedom should be given to the users (historians)?
- How could positions be defined in a way that was both platform independent and (preferably) customizable by the developers? Note that this question could be extended to other story elements, such as characters.

2.1 Methodology

The review process was divided into two different successive steps. The first step was the literature review and the second step was the systematic review.

The goal of the literature review was threefold. Firstly, to define the search terms used in the systematic review's search. Secondly, to define more proper inclusion criteria for the systematic review. Finally, to acquire information about different authoring tools.

The goal of the systematic review was to acquire information about a multitude of authoring tools and their functionalities, in order to ensure that this work was innovative in some form and to consider using some of those functionalities in my work.

A diagram showcasing the process can be seen in Figure 2.1.

Before beginning the literature review phase, the author instantiated a protocol for conducting both the literature and systematic reviews, which the author could follow. The instantiated protocol used was based on Morgado and Beck (2020).

2.1.1 Literature review

The protocol instantiation began by defining the concept (what the application should be), context (who the application was designed for) and perspective (the type of solutions that were intended) of this work. The author defined the concept as "Solutions to create/edit interactive 3D stories",

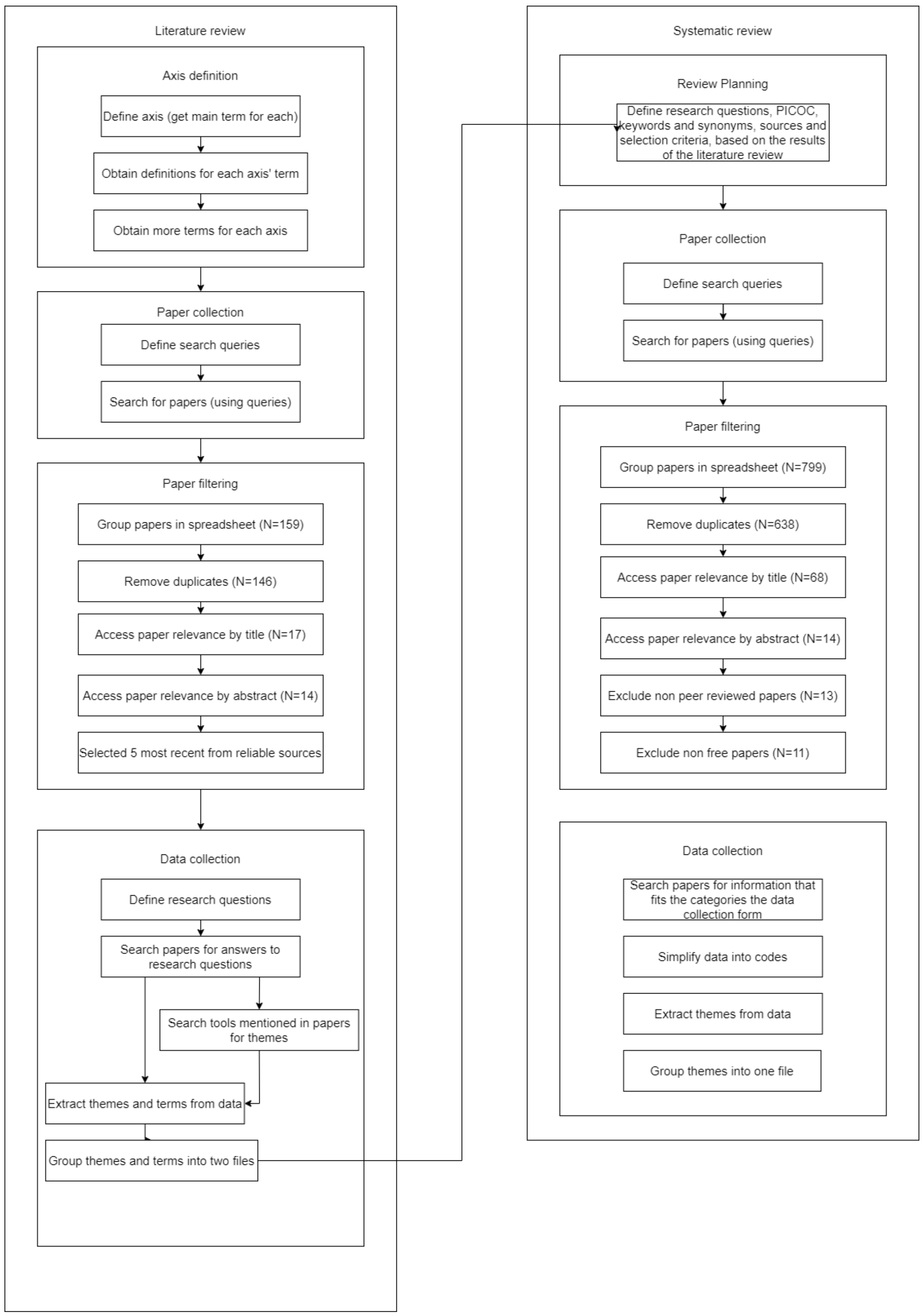

Literature review
Axis definition
Define axis (get main term for each)
Obtain definitions for each axis' term
Obtain more terms for each axis
Paper collection
Define search queries
Search for papers (using queries)
Paper filtering
Group papers in spreadsheet (N=159)
Remove duplicates (N=146)
Access paper relevance by title (N=17)
Access paper relevance by abstract (N=14)
Selected 5 most recent from reliable sources
Data collection
Define research questions
Search papers for answers to research questions
Search tools mentioned in papers for themes
Extract themes and terms from data
Group themes and terms into two files
Systematic review
Review Planning
Define research questions, PICOC, keywords and synonyms, sources and selection criteria, based on the results of the literature review
Paper collection
Define search queries
Search for papers (using queries)
Paper filtering
Group papers in spreadsheet (N=799)
Remove duplicates (N=638)
Access paper relevance by title (N=68)
Access paper relevance by abstract (N=14)
Exclude non peer reviewed papers (N=13)
Exclude non free papers (N=11)
Data collection
Search papers for information that fits the categories the data collection form
Simplify data into codes
Extract themes from data
Group themes into one file

Figure 2.1: Diagram representing literature and systematic reviews

the context as "End users without programming/designing experience" and the perspective as "Autonomous stories which can be created from scratch. These stories must follow a certain structure, unlike, for example, writing a story in a text editor."

Firstly, the author defined what "axes" were relevant for the search. An axis was defined as one or more terms which convey the same idea. Additionally, these terms should be ones frequently used in papers, so that the searches result in as many relevant results as possible. For example, *"Story"* was defined as an axis, and its terms are *“story”, “drama”, “storytelling”, “narrative”*. For the intents of this research and considering the concept, context and perspective, the author created the following axes:

- "Story"
- "Editor"
- "End user"
- "Interactive"
- "3D"

Following the creation of the axes, the author searched for papers including definitions for each of the axis.

The "Story" axis was given the following definition: Attempts to engage the player under the condition that the players’ behaviours and actions can directly impact the direction and/or outcome of the story. Balance player choices with the coherence and quality of the story. (Riedl et al. 2011), (Sgouros et al. 1996).

For each of the axis, the author defined a list of similar terms. They defined these terms by searching in papers, obtaining synonyms to the terms already in the axes and, in some cases, searching through the websites of well known authoring tools.

The following list contains the terms defined for each of the axis. Note that this list does not include pluralized terms, however, the search queries use them, as the search engine used does not automatically include them:

- "Interactive" - "Interactive"
- "Story" - "Story", "Drama", "Storytelling", "Narrative"
- "Editor" - "Editor", "Creator", "Creation", "Modeling", "Authoring Tool"
- "3D" - "3D", "Virtual"
- "End user" - "End user", "audience", "non programmer"

The author used Publish or Perish 8 [1] to perform the search and obtain the search results. They used "Google Scholar" [2] as the search engine.

For the literature review, the entire search query had to be in the title of the results, to improve their relevance. Additionally, ("survey or review") was included in the search strings, since, for the literature review, our target was surveys or reviews.

Because of the 256 character limit for Google Scholar queries (only counting terms, not spaces or operators), some of the terms belonging to each axis were removed. With our criteria, including every axis resulted in zero results before processing. As such, only some axes were used.

The following search queries were used, together with the number of results for each one:

- (story OR storytelling OR drama OR narrative) (editor OR editors OR creator OR creators OR authoring OR modeling OR meta-model OR tool) (survey OR review) - 82 results (as of November 7th, 2022)
- interactive (editor OR editors OR creator OR creators OR authoring OR modeling OR meta-model OR tool) (survey OR review) - 39 results (as of November 7th, 2022)
- interactive (story OR storytelling OR drama OR narrative) (survey OR review) - 38 results (as of November 7th, 2022)

The extracted results contained the "Number of Citations", the "Citations per Year", "Google Scholar Rank", "Title", "Year", "Publication", "Publisher" and "Type" (Example: "CITATION", "PDF", "HTML").

The results were downloaded into the comma-separated values file format (.csv). The results were merged into a single spreadsheet.

The initial corpus was thus N = 82 + 38 + 39 = 159. Of those, 13 duplicate results were removed (N = 146) by first marking papers with duplicate names and then manually verifying if those papers were duplicates. Afterwards, the process included reading the title of each paper, marking them as relevant or irrelevant (N = 17). Finally, the author read the abstract of each paper they previously deemed relevant and marked them as relevant or irrelevant (N = 14). Out of those articles, since the goal of this stage was to analyse surveys to attain a panorama of the field, in support of a subsequent systematic review, using the five most recent papers (>= 2018) from reliable sources was deemed an adequate compromise.

- "A systematic literature review on digital storytelling authoring tool in education: January 2010 to January 2020" (Quah and Ng, 2022)
- "A review of agency architectures in interactive drama systems" (Moallem and Raffe, 2020)
- "A systematic review of research undertaken into the use of digital storytelling as a pedagogical tool in the language classroom" (Abderrahim and Cigerci 2018, 113-124)

[1] Publish or Perish 8: https://harzing.com/resources/publish-or-perish
[2] Google scholar: https://scholar.google.com/

- "Define “authoring tool”: a survey of interactive narrative authoring tools (Green et al. 2018)"
- "The changing panorama of interactive storytelling: a review from locative to transmedia (Nisi, 2018)"

The author created a spreadsheet to store the extracted data.

The extracted data included a line for each category. The first three, "Authoring Tools", "Interactive Story Creation Methods" and "Authoring Tool Attributes/Features", were used to obtain themes for the systematic review. The other three, "Storytelling Terms", "Interactivity Terms" and "Authoring Tool Therms", are used to extend the list of terms from already existing axes.

The author read through each of the articles previously selected. Information relevant to one of the previous categories was underlined and added to the spreadsheet. Sections deemed irrelevant (Example: Answering questions not relevant to any category, review planning and conducting) were crossed out and were not read. The data was organized by paper and by category. The author stored the data in the form of small excerpts quoted from the papers. For example, the category "Authoring Tool Attributes/Features" included the excerpt "computational technology should be made invisible to the participants".

This step of data collection resulted in 29 text excerpts about Authoring Tools, 31 about Interactive Story Creation Methods, 63 about Authoring Tool Attributes/Features, 10 about Storytelling Terms, 1 about Interactivity Terms and 13 about Authoring Tool Terms.

The list of authoring tools found during data collection was placed onto another spreadsheet. Each authoring tool included a short description, which was used later. Authoring tools could be mentioned by name (Example: "Vegas Pro 11"), or through a short excerpt that described them (Example: "Green et al. (2018) (...) categorized taxonomy of authoring tools").

The author standardized each of the "Interactive Story Creation Methods", "Authoring Tool Attributes/Features" and authoring tool descriptions into themes (small excerpts of text) , which could be used to guide the systematic review. For example, "manipulate virtual puppetry for assisting narration" (Quah and Ng, 2022) was standardized into "mixing physical and digital media/interactions". Finally, the author compiled the themes and terms 2.1, removing duplicates, into two different text files.

The author reached some conclusions through the literature review process. Firstly, the authoring tools they intended to further analyse utilized a "flowchart" style of visualization inside the story creator. Additionally, they have obtained a list of terms which could be used in the Systematic Review 2.1.2. Finally, the other themes extracted could be used to guide the development of the final application.

These themes were divided into categories. The list of theme categories created from the themes obtained during the literature review can be seen below. Rows in bold represent categories of themes which the rows below belong to. Note that these themes are only used to guide the systematic review and are not used as part of the results.

This list details all the themes obtained from the literature review process

Interactive	**Story**	**Editor**
digital storytelling	interactive fiction	authoring tool
telling stories		design guidelines
creating stories		writing a script
narrative		storytelling environments
digital stories		composing a digital story
multimedia stories		digital storytelling environment
immersive storytelling		story making
virtual puppetry		prerecorded stories
story grammars		assisting narration
interactive comic		production of (...) stories
		design framework
		story plan
		story structure

Table 2.1: List of terms obtained from the data extraction phase of the literature review

Digital or physical stories:

- Creation of digital stories
- Natural language processing for communicating with characters
- Mixing physical and digital media/interactions

Methods for presenting and editing story fragments inside a story:

- Flowchart story creation tool (Most common)
- Slide presentation style creation tool (e.g.: Pow Toon)
- Timeline story creation tool (e.g.: Sony Vegas Pro)
- Keyword action based story creation tool (e.g.: TextureWriter)

Types of story editors (by freedom):

- Story meta-modelling
- Drawing approach to story creation (Without a model/set of restrictions)
- Modelling approach to story creation (instead of drawing approach)
- Provide a group of pre-selected functionalities for story creation

Element relationship visualization

- Visual relationships between elements
- Event based relationships between elements

- Internal relationships between elements

Types of interactivity:

- Procedural story plot generation with user input
- Only selection between multiple choice

Types of end users:

- Tool targetted at end-users with programming knowledge
- Targetting non-technical users

How are story elements fragmented inside a story? (for editing purposes):

- Story fragmentation by episodes (example of an episode: Chapter of a story)
- Fragmentation of stories by scenes (examples of a scene: virtual location in a game's location; Individual conversations/interactions between users)
- Story fragmentation by events (Examples: Character A says something to character B; Character C appears)
- Other (Example: In a timeline-based video editor, such as Movie Maker, stories can be considered as fragmented by individual video/audio clips)
- Story fragmentation by characters (Character-based story editing) (e.g.: Systems which can imbue certain personality traits to or relationships between virtual characters)

Multi-step story creation process:

- Single tool that allows doing multiple steps of the multi-step story creation process
- Use of multiple tools for the story construction process.

Individual features:

- Publishing of stories online
- Consistent iconography within the tool
- Consider user experience dimensions
- Use of popular characters
- Use of pre-made scenarios to teach new creators enable undoing of mistakes
- Collaborative narrative creation
- Create short stories to get an overview of the story

2.1.2 Systematic review

The systematic review was organized utilizing "Parsifal". [3]

Part of the structure included defining the objectives, the PICOC (Population, Intervention, Comparison, Outcome, Context), the questions, the list of axes and their terms, the search strings, the sources, the selection criteria and the data extraction form.

The goal of the systematic review was to explore the area of interactive story creation for non-programmers by conducting a small review of existing tools which use flowcharts to represent a story. The author defined the population as "non-programmer end-users", the intervention as "Flowchart story creation tool", the outcomes as "Find solutions which use flowcharts for story authoring" and the context as "Non-developer/programmer adults".

This review process attempted to answer the following questions:

- QSR1 - What are the main methods for creating authoring tools using flowchart-based tools?
- QSR2 - What flowchart-based authoring tools are there?
- QSR3 - What features are included in previous story authoring tools?
- QSR4 - What are the limitations caused by the lack of programmer experience from the end-user?
- QSR5 - What type of population does the study consider?

Before the search, the author obtained terms for the newly defined "flowchart" axis and added some of the terms defined in the Literature review 2.1.1 section.

The following list contains the terms defined for each of the axis, for the systematic review. Note that, once again, this list does not include pluralized terms, however, the search queries use them, as the search engine used does not automatically include them:

- "Flowchart" - "Flowchart", "Branching", "Plot"
- "Interactive" - "Interactive", "Dynamic"
- "Story" - "Story", "Drama", "Storytelling", "Narrative"
- "Editor" - "Editor", "Creator", "Creation", "Modeling", "Authoring Tool"
- "3D" - "3D", "Virtual"
- "End user" - "End user", "audience", "non programmer"

The author used "Publish or Perish 8" [4] to perform the search and obtain the search results. They used "Google Scholar" [5] as the search engine.

[3]Parsifal: https://parsif.al/
[4]Publish or Perish 8: https://harzing.com/resources/publish-or-perish
[5]Google scholar: https://scholar.google.com/

For the Systematic Review, some terms only had to be included in the "Keywords" section of the search, instead of having to be in the title.

The following search queries were used, together with the number of results for each one:

- **Title words:** (story OR storytelling OR stories OR narrative OR drama OR authoring); **Keywords:** (flowchart OR "non-linear" OR "branching") AND ("authoring tool" OR (interactive AND (editor OR tool OR creator OR creation OR creating))) AND ("no-code" OR "no code" OR "end-user" OR "end user") - 605 results (as of January 1st, 2023)

- **Title words:** interactive AND (story OR storytelling OR stories OR narrative OR drama OR authoring); Keywords: (flowchart OR "non-linear" OR "branching") AND (((editor OR tool OR creator OR creation OR creating))) AND ("no-code" OR "no code" OR "end-user" OR "end user") - 84 results (as of January 1st, 2023)

- **Title words:** (flowchart OR "non-linear" OR "branching"); Keywords: (interactive AND (story OR storytelling OR narrative OR authoring) AND (editor OR tool OR creator OR creation OR creating)) AND ("no-code" OR "no code" OR "end-user" OR "end user") AND (scene OR chapter) - 59 results (as of January 2nd, 2023)

- **Title words:** interactive AND (editor OR tool OR creator OR creation OR creating); Keywords: (flowchart OR "non-linear" OR "branching") AND (story OR storytelling OR stories OR narrative OR drama OR authoring) AND ("no-code" OR "no code" OR "end-user" OR "end user") - 51 results (as of January 2nd, 2023)

As for data filtering, this review only included papers which were in the English language whose authoring tools used flowcharts for story visualization during the editing process and were targetted for non-developer adults. Note that the population restriction only applies for the story authoring tool and not for the readers of the stories. Additionally, only peer reviewed papers were accepted.

There was a total of N=799 papers. 161 duplicate results were removed (N = 638) by first marking papers with duplicate names and then manually verifying if those papers were duplicates. Afterwards, reading the title of each paper and marking them as relevant or irrelevant resulted in the removal of 570 papers (N = 68). Finally, the author read the abstract of each paper previously deemed relevant and marked them as relevant or irrelevant (N = 14). Out of those articles, 13 were peer-reviewed. Out of those 13 articles, the author was able to access 11 of them.

In order to effectively select and organize the information from the papers, the author created a data extraction form. There was one line of information for each review question:

- Authoring tool story creation method(s) (22 results)

- Story authoring tool name/identifier (30 results)

- Story authoring feature(s) (106 results)

- Limitations/requirements due to lack of programming experience (5 results)

- Population description

For each article, the author noted information belonging to each of the five categories from the data extraction form. After that, the data was moved into a spreadsheet with one page per category. Note that the "Population description" category was ignored. For each of the strings inside each page, the author identified the main concept of that string and coded it into a more concise string. For example, "minimize the technical nature of error messages, themselves" (Kauhanen et al. 2007) becomes "Less technical error messages". The purpose of this process was to simplify the information used in the next step. Afterwards, those codes are grouped into themes, which are a topic that groups repeating ideas/codes (Vaismoradi et al. 2016).

2.2 Results

This section details the findings from the systematic review. It is subdivided by the categories defined during the systematic review, and those categories are subdivided by topic.

The following two tables (2.2, 2.3) indicate the obtained themes and their prevalence. Note that the third column, "Accommodations due to lack of programming experience", had to be included in a different table due to lack of horizontal space:

2.2.1 Authoring tool story creation method

This subsection details the main methods utilized in authoring tools to create or otherwise represent stories.

2.2.1.1 Story representation

From the reviewed papers, the most common way to represent a story inside an authoring tool was through the use of a graph. A graph is a data structure composed of a set of nodes and edges, where the edges point from one node to another (and vice-versa, if the graph is not directed). Nodes generally represent points in a story, and edges frequently represent choices by the story player. This approach was present in 55% of papers. These implementations include more traditional graph structures, some more limited structures (e.g.: Narawi et al. 2020), where certain features, such as backtracking (the ability to return to a previous node) or branching (the ability to have multiple different possible story paths). One paper (Schenk et al. 2013) presents a hierarchical story graph composed of two layers.

An alternative method to visually represent a story was to utilize a "text-based [...] indented tree-like structure" (Schenk et al. 2013). This approach was mentioned in 9% of papers.

One paper (9%) (Winterbottom and Blake, 2004) considered the use of a timeline visualization for interactions between predictable objects.

One paper (9%) (Winterbottom and Blake, 2004) considered the use of floorplans to detail space-related story occurrences.

Authoring tool story creation method	**Story authoring features**
Story representation (55%)	Colour-coding nodes (9%)
Collaboration in story authoring (9%)	Semantic zoom (9%)
	Freely place graph nodes in editor (9%)
	Variable to entity name glossary (9%)
	Interactive overlay graph map (9%)
	Different levels of abstraction by user (9%)
	Different views for different story parts (9%)
	Debugging/testing/error finding (45%)
	Play random story branch (9%)
	Monitor state of the game (9%)
	Error search engine (9%)
	Export scenarios to debug tool (9%)
	Automatic feedback on story structure (9%)
	One start node and one or more end nodes (18%)
	Repeat parts of story node (9%)
	Create/save/load story (18%)
	Component search filter (9%)
	Use of custom media (18%)
	Relational database for saving stories (9%)
	Files for saving stories (9%)
	Use of custom story resources (18%)
	Library of common resources (18%)
	Keep track of components inside exported resource (9%)
	Objects are treated as individual story components (9%)
	Ability to target any game engine (9%)
	Pluggable framework (9%)
	Drag objects to area map + edit properties (9%)
	Annotative flow chart for event handling (9%)
	Basic text editing functionalities (18%)
	Replacing objects with others in a story (9%)
	Auto generated facial animations (9%)
	Real-time rendering engine (9%)
	Use of IoT (9%)
	Methods to reduce file size (9%)

Table 2.2: List of Themes and their Prevalences - Part 1

Accommodations due to lack of programming experience
Keeping track of story resources (9%)
Terminology in authoring tools (9%)
Error message intrusion (9%)

Table 2.3: List of Themes and their Prevalences - Part 2

One paper (Torres et al. 2022) describes three additional types of simulation structures compatible with stories. Firstly, exploratory simulations, which are open-ended allow the player to decide on what information to gather. Secondly, responsive simulations, which are open-ended simulations powered by an engine, where the users can pick assignments and the simulator will respond in real-time. Finally, communication simulations, which use natural language processing and artificial intelligence to interpret questions asked by the user and respond appropriately.

In graph-based implementations, transitions (detailed by edges) can be caused by different types of events. Most commonly, transitions are caused by decisions made by the player (e.g.: Not and Petrelli, 2019). This can be caused by directly presenting a player with a multiple-choice question (e.g.: Narawi et al. 2020) or by the player doing some action inside the story while at a certain point in the story (e.g.: Schenk et al. 2013). Alternatively, graph transitions can be caused not just by player choice, but also, for example, by random chance (e.g.: Erdem et al. 2009). Schenk et al. (2013) describe an additional restriction where nodes (story points), when visited, can succeed, fail or neither. Succeeding enables the immediate successors to that node, failing disables the current node and neither means that nothing happens and that node can be revisited. In the case of merge points, in addition to its predecessor having to succeed, a certain amount of previous nodes have to have succeeded for them to be enabled.

2.2.1.2 Collaboration in story authoring

One potential issue with story authoring tools was the handling of collaborative development, especially with a multidisciplinary team. 9% of papers mentioned this issue, suggesting the use of agile processes (Laurent et al. 2022).

2.2.2 Story authoring features

This subsection details a group of features identified in the papers during the systematic review. They range from essential to complimentary, or even features specific to certain types of authoring feature.

2.2.2.1 Story authoring assistance (visual)

One paper (9%) described an authoring tool which colour coded nodes depending on their functionalities. Start and end nodes are coloured blue and red, respectively, while grey nodes present information or choices to the player. Green nodes indicate a change in scenario, such as changing the room. (Torres et al. 2022)

The ability to zoom inside the story authoring tool was mentioned in a single paper (Göbel et al, 2008). This feature represented story elements with different levels of detail, such as using icons or preview images, depending on the level of zoom.

One paper (Göbel et al. 2008) mentions a tool which facilitates easily understanding the story as a glance by allowing the author to freely place the story graph's scenes in the editor. In addition, the same tool allows background images to be placed in the editor. Göbel et al. (2008) exemplify by mentioning that, for a site plan, the author could arrange the story's scenes by their geographic positions.

A single paper (Subramaniam and Gokhale, 1995) referenced the use of a glossary which links variable names to entity names (the ones which are shown onscreen.

An interactive overlay map was used in the tool referenced by Torres et al. (2022). This map shows the current story's entire node structure and allows the user to navigate through the story.

One paper (Laurent et al. 2022) suggests that, in order to build a tool which was both versatile and easy to use, that different levels of abstraction of the authoring tool are used depending on the user's competencies. This level of abstraction can be implemented by providing the ability to hide advanced features from the author. For example, computer scientists can be allowed more flexibility at the cost of complexity, while teachers with little computer skills would prefer a simpler tool.

One paper (Winterbottom and Blake, 2004) suggests the use of different views to convey different aspects of the story, such as a time-based view and a space-based view.

2.2.2.2 Story authoring assistance (debugging)

A common trend found in authoring tools was the ability to debug, or otherwise, facilitate the testing of stories or finding of errors created in them. This feature was present in 45% of papers. These debug features differ depending on the program/paper.

Erdem et al. (2009) detailed an authoring tool which includes the ability to play a random branch from the story. They further elaborated by declaring that it was an efficient way to discover possible dead-ends in a test scenario.

Laurent et. al (2022) provided a small list of testing and debugging features. These include monitoring the current state of the game through the use of an interactive preview, and being able to find errors and links between story objects through the use of a search engine.

According to Erdem et al. (2009), the authoring tool referenced in their paper includes a separate "Testing and Debugging Tool", which allows for the design verification of the non-linear stories. Additionally, it provides the ability to export scenarios to the testing and debugging tool, regardless of whether they are complete or not.

Winterbottom and Blake (2004) state that the tool detailed in their paper should give feedback on the structure of the created stories. For example, finding holes in a sequence of interactions in a story.

2.2.2.3 Basic functionalities

Within the papers which mentioned the use of a graph structure, some mention the use of specific types of nodes. For example, 18% of them mention the existence of a starting, one or more end nodes and a node which indicates a scene change (e.g.: Narawi et al. 2020). Narawi et al. (2020) additionally mention the use of two entirely optional nodes. One of them repeats a single part in a story while the other repeats an entire scene in the story.

Two papers (18%) mentioned a group of basic features for an authoring tool. These include the ability to create a new story (e.g.: Torres et al. 2022), the ability to save and load a story (e.g.: Torres et al. 2022) inside the authoring tool, the ability to retrieve a list of all stories (e.g.: Erdem et al. 2009) and the ability to search for a specific story (e.g.: Erdem, 2009).

2.2.2.4 Additional functionalities

One paper (Schenk et al. 2013) details a tool which includes a search filter to assist finding certain story components from the story component library.

A functionality found in some authoring tools was the ability to utilize custom media (e.g.: audio, images, video, 3D models) inside the story. This functionality was found in 18% of papers. There were two different methods, one allowed for URLs (e.g.: Torres et al. 2022) and another which used the file's name (e.g.: Göbel et al. 2008). In the case of Torres et al. (2022), in addition to being able to play dialogue audio using a URL, a piece of dialogue can be played as audio through the use of a text-to-speech program.

A total of 18% of papers referred a specific method of saving/exporting story files. One paper (Edrem, 2009) mentions the use of a relational database to store both intermediate and output data. The other paper (Schenk et al. 2013) states the use of files to store the story scenes, elaborating by stating that these files are specific to a game-dependent tool.

The ability to use custom resources was an important feature in authoring tools. Two papers (18%) mentioned this feature. Those resources differed from individual resources, such as text, scripts (Kauhanen et al. 2007) or images (Laurent et al. 2022) to composite resources, such as characters, which included names, descriptions, a picture (e.g.: Laurent et al. 2022), and map features (e.g.: Kauhanen et al. 2007). This included the ability to import and export them to facilitate reuse and sharing (e.g.: Laurent et al. 2022). Both papers additionally suggested that a library of commonly used resources should be provided. Laurent et al. (2022) suggested that these common components should be inferred based on usage. Alternatively, Kauhanen et al. (2007) suggested that users should be able to add their custom components to that library.

Kauhanen et al. (2007) recommended implementing a method to keep track of the components inside an exported resource. They state that it facilitates understanding a story before opening it inside the authoring tool.

Kauhanen et al. (2007) detailed an authoring tool where objects were treated as individual story components. These objects contained a group of parameters (e.g.: movement speed).

ScriptEaseII's authoring tool had the ability to target any game engine or API (Application Programming Interface) as long as a translator was created for it (Schenk et al. 2013). This meant that the stories created in the authoring tool are not locked to a certain platform.

Göbel et al. (2008) describe a tool which uses a pluggable framework composed of five different tools: The Story Editor, the Stage Editor, the Action Set Editor, the Property Editor and the Asset Manager. The fact that it's pluggable means that different editors can be used for different applications. For example, different stage editors can be used for the editing of 2D (Two Dimensional) and 3D (Three Dimensional) stories.

The authoring tool mentioned in Kauhanen et al. (2007) contains a higher level language for creating authoring tools. It includes the ability for authors to drag objects onto an area map and then edit their proprieties through a property screen.

ScriptEaseII (Schenk et al. 2013) utilized an annotative flow chart to trigger the selected behaviour based on what event was fired.

2.2.2.5 Context-specific functionalities

In total, 18% of papers highlight the inclusion of basic text-editing functionalities, such as word or line deletion and automatic word wrapping (Subramaniam and Gokhale, 1995), and rich text formatting (Torres et al. 2022).

One paper included the ability to replace individual objects in a story with another object, without needing to change the story itself (Abawi et al. 2004).

Erdem et al. (2009) wrote about a program which includes the ability to automatically generate facial animations for characters using recorded speech as input.

Erdem et al. (2009) mentioned the integration of a real-time rendering engine in an authoring tool. This allowed authors to create 3d environments using a modelling tool and use the tool to render them in real time while playing.

One paper (Not and Petrelli, 2019, 67-120) mentioned the use of IoT (Internet of Things) devices to combine physical interactions with a virtual story. Not and Petrelli (2019, 67-120) mention a method to facilitate the use of Internet of Things by end-users. They suggested abstracting away the set-up of interconnected devices by, for example, setting up parameters to set when a sensor is triggered (abstracting the sensor layer). This suggestion had the additional upside of facilitating the debugging of sensor events while testing. Not and Petrelli (2019, 67-120) additionally described the ability to, using a menu selection or a drag-and-drop, associate multimedia items to physical objects.

One tool provided two methods of reducing file size. Textures were reduced to a maximum size of 512x512, a low polygon count is adopted for 3d models and the use of build reports to identify unused components and ones which have a high resource demand. This was useful for web applications, to prevent users from having to download large files over the web to use the application.

2.2.3 Accommodations due to lack of programming experience

The systematic review only included authoring tools designed for non-developers. As such, there were a number of problems caused by the lack of programming experience by the end users of these applications. This subsection covers some of them, including accommodations included in the authoring tools to minimize these issues.

2.2.3.1 Keeping track of story resources

Keeping track of story resources could be difficult, especially if the author needed to know all the details about them and/or if there were multiple versions of them (Kauhanen et al. 2007). As such, Kauhanen et al. (2007) recommended the use of a library of commonly used resources, which preferably allowed the addition of custom content.

2.2.3.2 Terminology in authoring tools

The terminology used can impact the end user's understanding of the application. Kauhanen et al. (2007) recommended the use of simple terminology, both for commands in the story authoring tool and in error messages, alternatively suggesting the use of terms related to story creation. For example, using the term "Go" instead of the more technical "compile-run" (Kauhanen et al. 2007).

2.2.3.3 Error message intrusion

For errors, not only should the error messages be easy to understand by the end-users, they should not be intrusive, as they are considered a part of the development process (Winterbottom and Blake, 2004). 18% of papers mentioned one or more of these concerns. For example, instead of a pop-up error, errors acted as status indicators, similarly to compiler warnings (Winterbottom and Blake, 2004).

2.3 Discussion

This section includes a discussion of the points raised in section 2.2. The criteria used for likelihood of future adoption was based on the current understanding of the end-users' requirements and the opportunity cost of implementing such a feature, due to the limited time available for implementarion.

2.3.1 Authoring tool story creation method

2.3.1.1 Story representation

In this section, the author discussed the results obtained in the 2.2 and decided on the applicability of certain features to our project.

Out of all the story representations, the story graph was considered the best for this project, as the nodes of a graph could represent different states of the same story and the edges could represent choices from the player. The ability to backtrack to a previous point in the story graph can be important for some types of stories. For example, virtual visits through a virtual recreation of a real place. A hierarchical multi-layered story graph was considered and may be used under a different context (e.g.: scenes detail single interactions between characters and complex scenes detail all the possible interactions in one location).

2.3.1.2 Collaboration in story authoring

The use of agile processes for authoring can be useful, however, without having fully specified the requirements by communicating with the end-users yet, the relevance of this consideration is currently unknown.

2.3.2 Story authoring features

2.3.2.1 Story authoring assistance (visual)

The colour coding of nodes was an easy feature to create and had a significant upside. As such, it would be likely implemented in a flowchart-based story editor. Additional care had to be placed in ensuring that the colours used are easily distinguishable by people with colour blindness.

The ability to zoom in/out inside the story graph view was important to help the author visualize the story as a whole, with the added upside of being quick to implement. However, a semantic zoom feature which displayed story nodes with a lower level of detail the more zoomed out they are would take a significantly longer amount of time at a likely lower benefit. As such, a non-semantic zoom feature was likely to be implemented.

The ability to reorganize nodes within the story editor was considered important for the same reason as the zoom in/out feature. As such, it was implemented in the final version of the story editor, as well as included in the first story editor prototype.

The ability to upload background images to use inside the graph view was only useful in specific contexts. Additionally, it had the significant downside of requiring whatever system saves the story files to also allow saving images and their position/scale/rotation (if applicable), which also likely increased their file size and possibly load/save times in the authoring tool.

The use of a glossary to link variable names to entity names could be potentially useful. However, this firstly implies that some sort of story component would have names shown onscreen and secondly implies that some names onscreen should not be the same as the ones stored inside the story files.

The use of an overlay minimap showing the entire node structure of the graph would significantly reduce the time taken navigating through a long story. As such, it is a potentially useful feature to implement, depending on the end-user requirements.

Due to the limited time frame for implementing the application, the use of different views for different users or for conveying different aspects of the story was not considered. Additionally, in the former's case, the end-users are historians and, as such, they most likely do not have very heterogeneous programming skills, this making the upside irrelevant.

2.3.2.2 Story authoring assistance (debugging)

The authoring tool will most likely have some form of testing feature included, with the added caveat of having to be easy to use, even for non-programmers.

If the authoring tool is designed in such a way that dead-ends in a story are not easily perceptible, then random scenario testing can prove to be useful.

The ability to monitor the state of the story while playing is likely useful, as it helps story authors know where they are in a story.

2.3.2.3 Basic functionalities

The authoring tool would most likely have (one) start and (one or more) end nodes. However, the addition of mid-story scene changes requires a lot of time to implement and, through my current knowledge of the end-user requirements, does not seem overly useful. As such, scene change nodes were not included in the story editor prototypes.

The ability to create a new story and save/load a story was included, as well as the ability to search for and list stories.

2.3.2.4 Additional functionalities

A search filter for story components may be included, however, that feature's utility heavily depends on the implementation of the authoring tool and the scope of the stories intended to be built using it. As such, it is difficult to predict whether this feature is useful enough to implement.

The use of custom images, audio, and videos will be included in the authoring tool, as they have been previously defined as requirements for the application.

Depending on the scale of usage and the size of the story files of the authoring tool, a relational database becomes more useful to store them relative to file system storage. However, as the files will have to be stored on a web server, there is no reason not to use a relational database.

2.3.2.5 Context-specific functionalities

The ability to replace objects with others in a story, while originally stated for a mixed reality context, may prove useful, as the idea may be extended to allow for the same story to be used in entirely different scenarios with different objects, so long as there is a translator between objects in both scenes (e.g.: A dictionary with one scene's objects as keys and another's as their values).

2.3.3 Accommodations due to lack of programming experience

2.3.3.1 Keeping track of story resources

As said previously, a library of commonly used resources would likely be too time-consuming to implement. As such, this issue is unlikely to be addressed

2.3.3.2 Terminology in authoring tools

The authoring tool will use non-technical terminology whenever possible. Additionally, terms with meaning specific to a programming context should not be used.

2.3.3.3 Error message intrusion

It is important that error messages are not made intrusive, as has been previously explained.

The creation of a library of common resources with the ability to add custom resources, although useful, would be very difficult to implement in a short period of time. As such, it is unlikely to be included in the authoring tool

A method of keeping track of story resources inside a story would be exporting the stories in a human-readable format (e.g.: properly formatted JSON). This has the negative implication of occupying more storage space.

Chapter 3

Development process

This chapter provides the context for the development of these applications, as well as a general outline of the development process of the applications and prototypes mentioned in this thesis.

3.1 Project context

The application was developed for use in the FronTowns (NOVA FCSH, n.d) project, whose goal is a 3D historical recreation of villages in the border between Portugal and Castille.

The applications whose development is described in this thesis were created for the purposes of the FronTowns project. One of the goals of the FronTowns project is to recreate and study the evolution of two villages, Castelo de Vide and Cáceres. This thesis will focus on the creation of the wall in Castelo de Vide.

Considering the context of the FronTowns project, an application was created to be able to create stories inside the historical reconstitutions. The editor would have the purpose of enabling/-facilitating the development of stories by the historians. These stories' main purpose would be to assist the historians in testing their hypothesis on the historical reconstitutions of those places.

3.2 Methodology

This subsection describes a list of methods which were taken into consideration while researching and developing the project.

3.2.1 Design Science Research

The methodology used was adapted from DSR (Hevner, 2007) to structure the future work plan.

Hevner (2007) presents a view of the design science research as three closely related activity cycles.

The Relevance cycle entails the development of prototypes based on the requirements from the Design cycle and the findings from the Rigour cycle. For this project, this translates to the presentation of the prototypes to the team members and thesis advisors, which can be found in

4.3, and 4.5. Unfortunately, due to time constraints, the presentation of the prototypes to potential end users is limited. Additionally, it also includes the feedback about the application, and any requirements obtained from that feedback.

The Design cycle involves not only the development of Design artefacts (e.g: paper prototypes and sketches), but also the evaluation process of the prototypes. The Design cycle takes as input the requirements discovered by the Relevance cycle and the evaluation theories and methods from the Rigour cycle (Hevner, 2007). For this project, this translates to creating implementable solutions to the requirements, and the development of the prototypes that would be evaluated in the future. This can be found in 4.1, 4.4, 4.6, 4.8, and 4.9.

The Rigour cycle serves, in part, to produce a strong, grounded argument for the construction of the design artifacts (Hevner, 2007). In this project, this was executed through the review process, which can be found here 2. The conclusions from the review guided, in part, the rest of the project. Note that the review process was only executed once (the literature review determines), instead of being iterative. The review was done partway through the development of the story player, and the beginning of the story editor.

The Design Science Research (DSR) methodology describes a set of practices that must be done to carry a design science research in information systems (Peffers et al., 2007).

The starting point of the DSR Process Model for this project was the problem. Firstly, the problem was identified, both through the early meetings, described in 3.3 and the literature review described in 2. Afterwards, the goals of a solution were defined, and, based on further meetings, an application prototype was developed and finally evaluated in more meetings. After that prototype is evaluated, potential problems are identified. Solutions were identified to those problems and a new prototype was developed based on the solutions that are viable (e.g: Ones that do not require too much development time to be sustainable). This process was repeated until there was no more time to continue. This process, as of currently, was stopped at the development phase of the last version of the prototype.

3.2.2 xAPI and JSON structure used

The experience API file structure was used in this project for the story JSON files, which are imported/exported by the editor and imported by the story player.

The xAPI (Experience API) uses the following file structure. Each field's information has been replaced with "[...]" for this example:

verb: [...] actor: [...] place: [...] context: [...]

In the applications made for this project, the verb, actor, and place are a string representing their names, and the context is a JSON object containing a fixed group of fields, some of which are always empty, depending on the type of action. The reason those fields are still included in the JSON file is because the default Unity JSON serializer does not serialize fields inside subclass objects in other classes.

The following example explains this problem:

```
public Class1{
    Class2 class2Object;
}

public Class2{
    float val1;
}

public Class3: Class2{
    string val2;
}
```

If the Unity application were to attempt to serialize the following object with the default serializer:

```

 Class2 class3Object = new Class3();
 class3Object.val1 = 4.5f;
 class3Object.val2 = "Hello, world!"
 Class1 class1Object = new Class1();
 class1.class2Object = class3Object;
```

The output JSON file would not include the field val2.

3.3 Development Process

The diagram in 3.1 provides an overview of the development process:

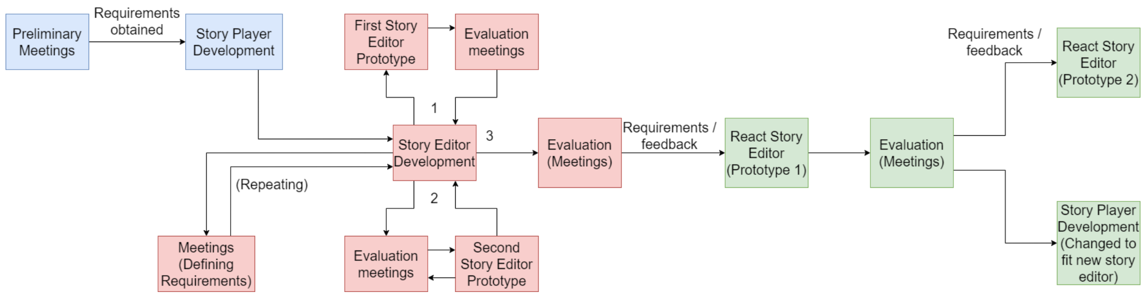


Figure 3.1: Development Process

The work presented in this thesis was developed over the course of about a year, starting in July 2022 and finishing in June 2023 (including August 2022 and June 2023).

As mentioned previously, the development process consisted of a sequence of iterations, each of which included obtaining requirements through meetings, using those requirements to obtain solutions, creating a prototype implementing those solutions, evaluating that prototype by presenting it to members of the team to obtain feedback and requirements. Since the development of the story player and editor were partially intertwined, the development cycle was modified to fit this. Finally, there were periodic meetings over the course of the development.

As a preliminary work to this thesis, some requirements for the story player were obtained, though a number of meetings with the thesis advisors and the team. These meetings had the goal of explaining the context of the project and its current state to the author, as well as the project's goals. Additionally, a team member provided the source code for their version of the story editor and player, along with some feedback pertaining that story editor and player.

The story player was created according to the requirements defined and based on the the previously existing story player. Over the course of this development, multiple meetings with the thesis advisors and a team member resulted in feedback.

After that, development began on the story editor, in Unity. A number of requirements for this editor were already known, due to the overlap in requirements with the story player. This story editor was based on the previously existing story editor. A paper prototype was created, based on the work-in-progress story editor, and then shown to members of the team to obtain feedback about its flaws. After some more development of the story editor, based on that feedback, another prototype was created. This prototype was presented in a meeting to members of the team, which generated more requirements. One team member proposed a number of different layouts for the story editor. One of the designs was accepted and a new prototype was build, using React. This prototype was evaluated partway throgh development by members of the team, after which, the possible improvements noted in that meeting were implemented in the story editor. Finally, the story player was modified to fit the new functionalities introduced in the new editor.

3.4 Requirements Elicitation and Objectives

The requirements noted in this section were obtained through meetings with the thesis advisors and members of the FronTowns team. These meetings occurred prior to the literature review process 2.

The goal of these applications is to create a story editor that minimizes the amount of programmer and designer assistance necessary for the historians to make modifications to the stories. For example, if a historical simulator uses a certain military strategy or formation, and then the historians discover that the tactics or formations used in that situation were different, then that software would not be used, or they would require programmer assistance.

Stories must be able to be represented/played in different story players using different representations. For example, the same story created by the editor should be playable in both the 3D story player developed and a future story player, which plays the story as a comic book. This does not mean that the story can't contain fields exclusive to a certain player or representation, instead, the story has to follow a certain structure. The xAPI (Corbí and Solans, 2014) structure was used. The way that the xAPI structure was used was explained in the 3.2.2 section.

The platform has to allow for different 3D environments to be used. This does not mean that the same story has to be playable in multiple environments, it means that the user can select the scenario which a story is made for.

The previous version of the story editor scaled poorly with story size, meaning that medium or large stories worked poorly.

Despite the previous version of the story player being in a 3D environment, characters could only move in a plane. This version should adapt to 3D environments.

The target end-users of this application are historians. This application is intended mainly for the creation of historical recreation stories.

One important question was the degree of freedom that should be given to the story creators. The greater degree of freedom given to the user, the more they will be able to do without requiring the application to be modified, however, that increases the development time, can increase the story creation time and having the story editor include more features can confuse the users.

One of the main concerns was defining positions in such a way that they are platform independent and preferably customizable by the story creators.

3.5 Existing Story Editor and Player

The designs of the story editor and player were based on a previous story editor and player made by a member of the team.

3.5.1 Story Editor

The story editor used a hierarchical tree structure, similarly to the first prototype of the story editor presented in this thesis.

The application began in the main screen, as seen in 3.2. The user could either press the new story button to create a new story, the open story button to open an existing story, the translate button to create a copy of an existing, or the close button to close the application.

If the user pressed the new story button, they will be prompted to select the recipe from a fixed list of available recipes, each entry containing the name of the corresponding recipe. A recipe defines the list of characters and places that its stories can use, as well as the 3D environment that is used to play them in the story player.

If the user presses the open story menu, they will be prompted to select the story they want to open, each entry containing the name of the story.

The new story menu and open story menu can be seen in figure 3.3.

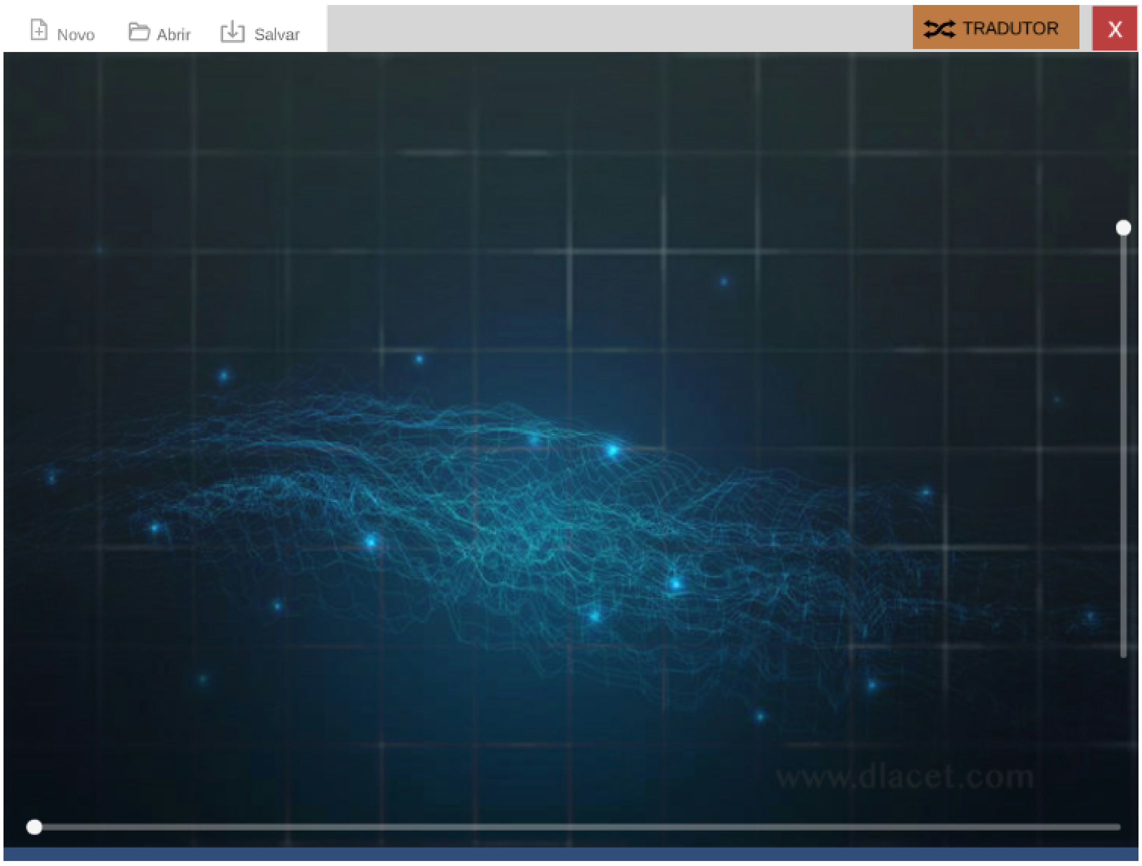
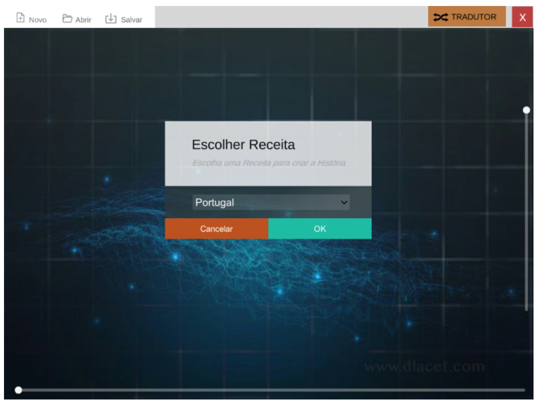
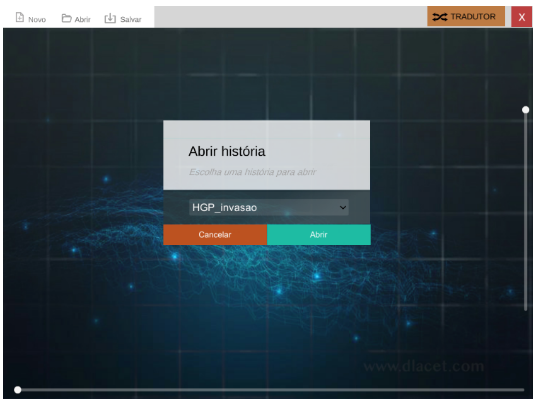


Figure 3.2: Team member's story editor: Starting State



(a) Team member's story editor: Create Story Menu



(b) Team member's story editor: Open Story Menu

Figure 3.3: New story and open story menus

When a story is selected (either by loading an existing story or creating a new one), the user can do the previously described actions, along with saving the story, translating the story to a different recipe, creating a new action, or modifying an existing action. When the user entered the save menu, they could write the name of the story they're about to save, as seen in 3.4.

When the user entered the translate menu, they would be requested to select a story to translate.

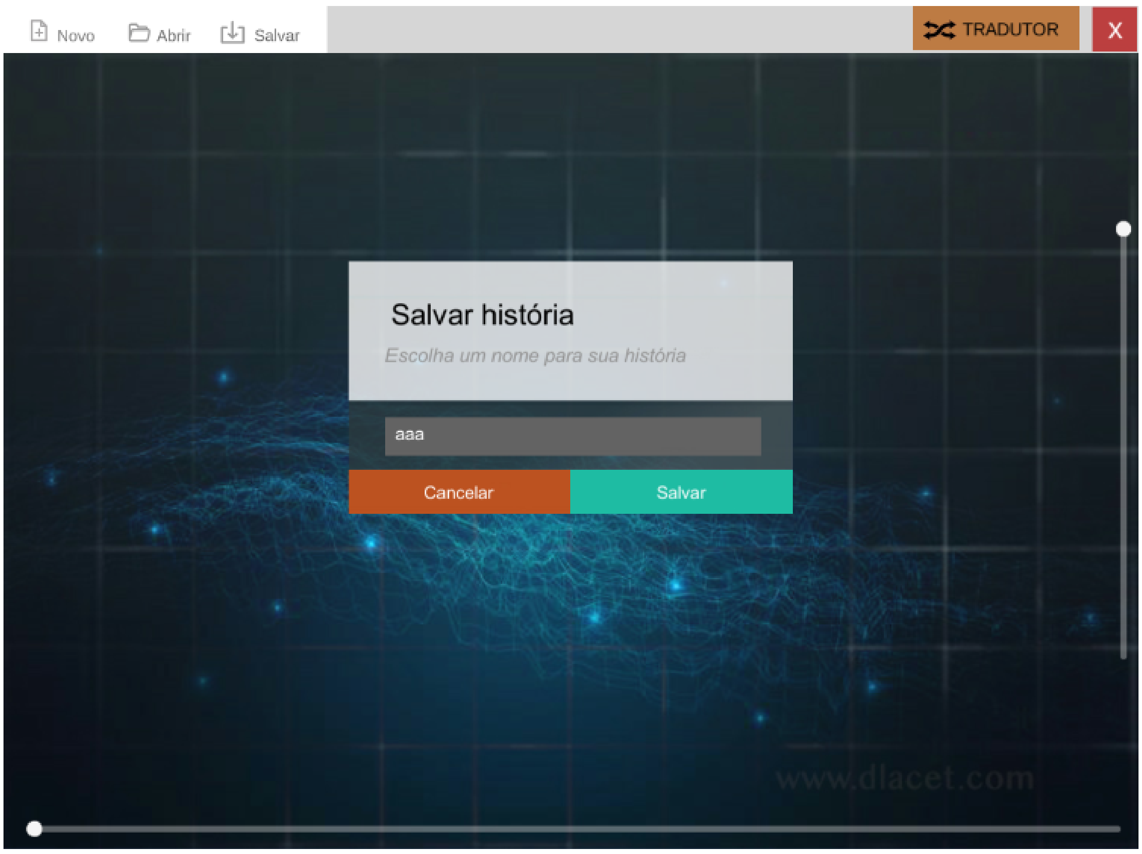
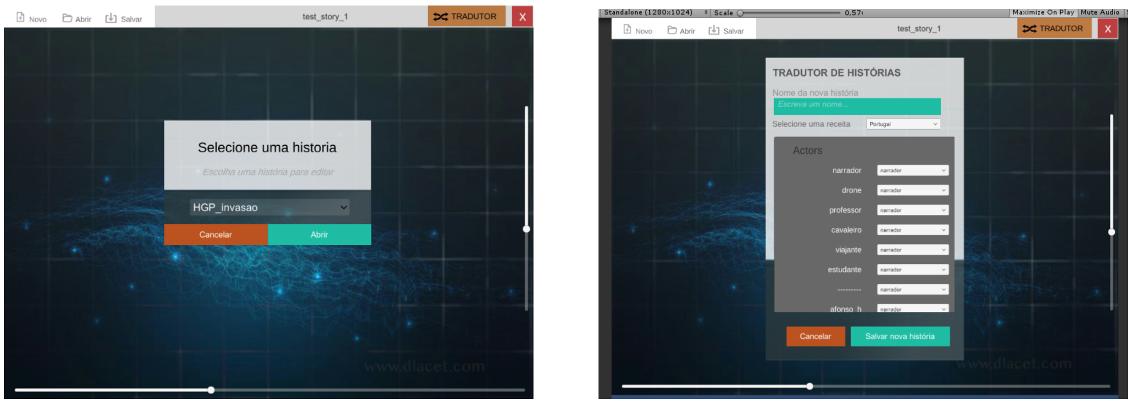


Figure 3.4: Team member's story editor: Save Story

Once the user selects the story to translate and clicks the open button inside that menu, another menu will appear, which includes a text field for the name of the new story, a dropdown to select the recipe to use, and a scrollable window with three lists of dropdowns, each with an accompanying text, one for actors, another for places, and another for types of actions. The text on the left indicates the name of the old actor/place/action type, while the dropdown on the right's selected text represents the name of the new one. There is one field per actor/place/action that exists in the old recipe. Once the user clicks the save button inside that menu, a new story was created with that name, which replaced the old actors/places/actions with the ones the user selected. The translate menu can be seen in figure 3.5.



(a) Team member's story editor: Translate Story Menu: Story Selection

(b) Team member's story editor: Translate Story Menu: Story Translation

Figure 3.5: Translate story menu

There were 7 types of actions. The following list contains the names of the actions, along with their names translated to English and what they do.

- esta (Appear/Stay). This action places the specified character at the specified place.
- irpara (Move [Towards]). This action made a character move to the specified place. The character moved towards the destination in a straight line, meaning that there was no destination.
- dialogo (Dialogue). This action specifies a dialogue, between multiple characters.
- escolher (Choice/Choose). This action presented a choice to the user with two different possible answers. Depending on the answer chosen by the user, the story would continue at a different point. For choice actions, actions are identified by name.
- mostrar (Show). This action displayed an image.
- esconder (Hide). This action removed a character from the story. Note that the character could be added back into the story if a future action required so.
- play (Play). This action played the specified video.

Actions were represented by rectangles including their fields. Each action contained a name, represented by a string, an author, an action type, and a place, all of which were represented by dropdowns, where each entry contained the corresponding author, action type, or place's name, an input (unless it is the starting action) and an output, and, optionally, a duration, which was a number, in seconds. An action's input could be connected with another action's output (not vice-versa) by first clicking on the input left mouse button, and then clicking on the output with the left mouse button. The actions in the editor are shown in figure 3.6.

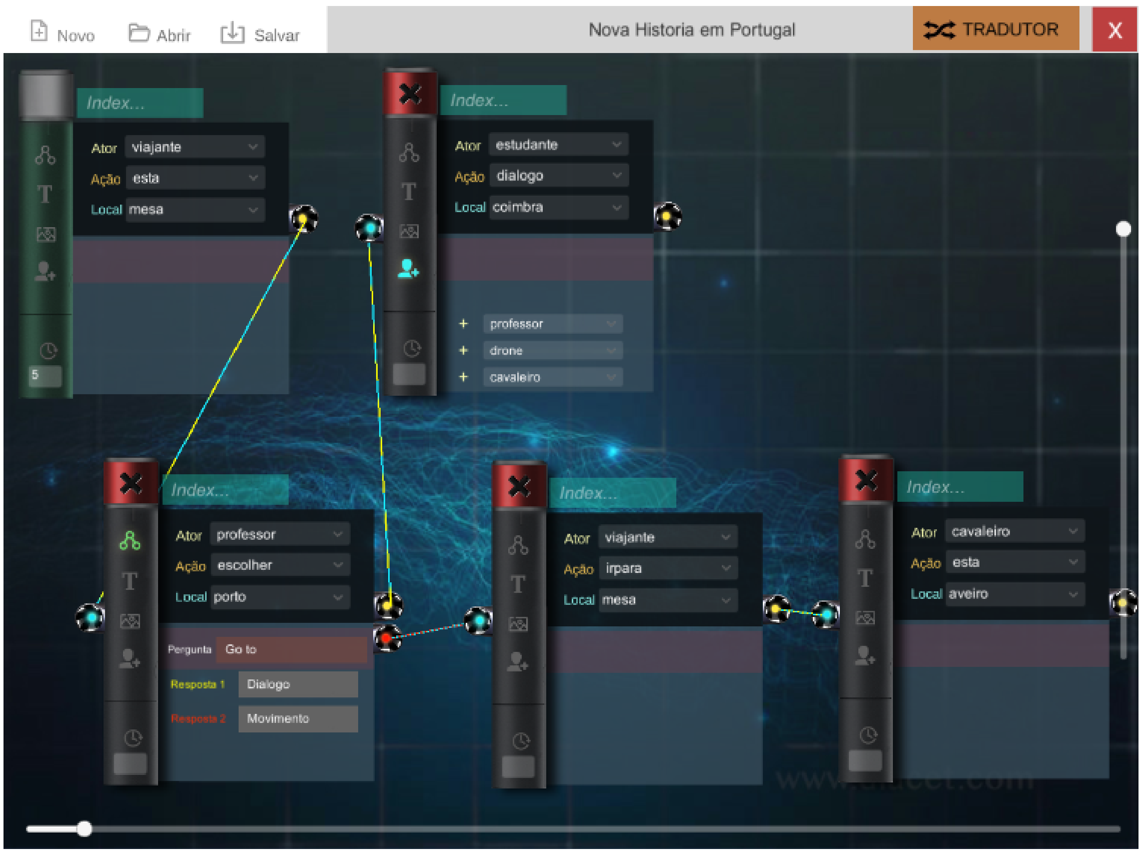


Figure 3.6: Team member's story editor: Example Group Of Actions

Each action contained a group of optional selectable fields, which were displayed by clicking on their corresponding buttons, which were located to the left of their action. Only one of these optional actions can exist per action, except for the last option, which can coexist with the previous three. These optional fields will be discussed from top to bottom, considering the position of their corresponding button. The first option included a question and two answers, and one additional output. The first answer used the pre-existing output, while the second answer used the new output. The second option included a text and a URL for a sound clip (.mp3 or .wav extension), which could be used, for example, as a narration. The third option included two checkboxes and a URL text field for the intended resource. If the first checkbox was ticked, the URL would represent a video (.mp4 extension), otherwise, the URL would represent an image. If both checkbox were ticked, the video would appear in fullscreen. The fourth option included a dropdown that the user could use to select one or more additional actors for the action, up to a maximum of 3 more. These additional actors would be placed next to the "main" actor of the action.

3.5.2 Story Player

This version of the story player was a web application created using the Unity engine.

The story player begins in the start menu. The user could select a story to play from the dropdown. Afterwards, they could click the play button to play the story. Alternatively, the user could click on the bottom button to access the story editor. The story selection menu can be seen in 3.7

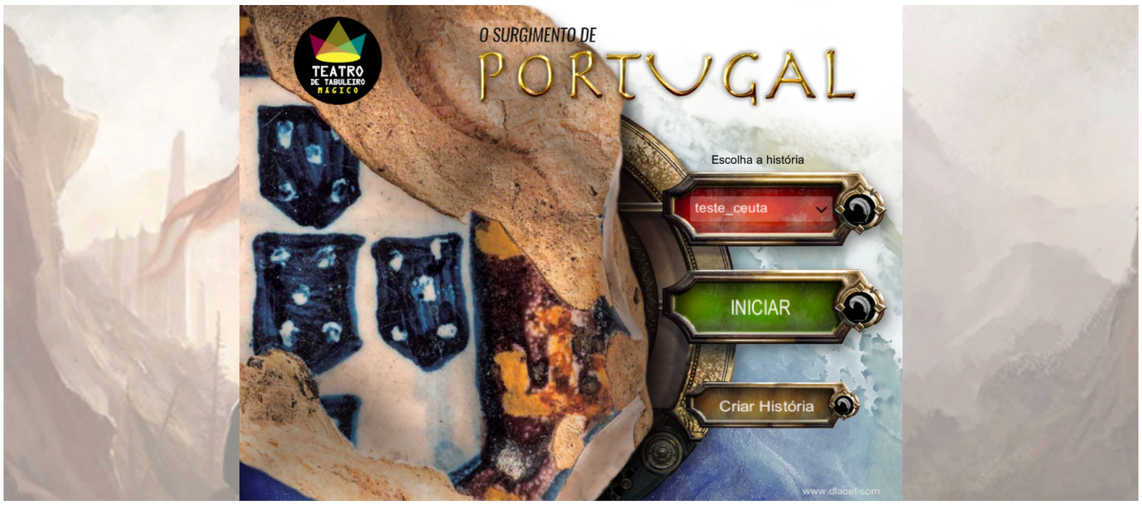


Figure 3.7: Team member's story player: Story selection

Once the user clicks the play button, the scene beings playing. First, the camera pans around the scene, quickly displaying the entire scene to the user. After that, the actions in the story being playing sequentially, except for choice actions, which divert the flow of the story to a different action, depending on the user's choice. The camera followed the characters in the current action. The user could freely rotate the camera.

An example of an action can be seen in figure 3.8.

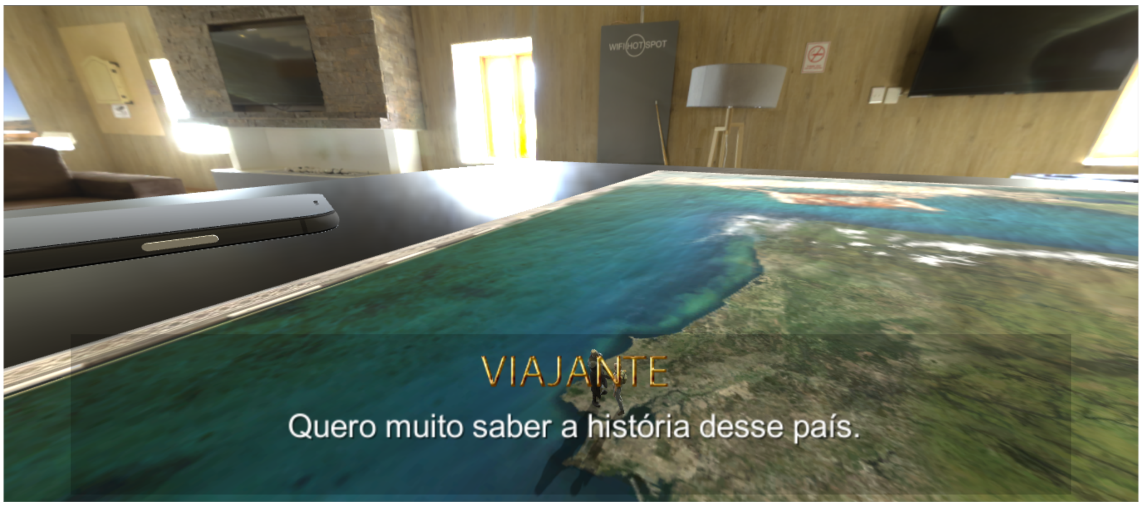


Figure 3.8: Team member's story player: Dialogue action

Chapter 4

Proposed solution

4.1 First Player and Editor - Design & Development

This chapter details the first iteration of the player and editor developed for this dissertation. The development of the story player started before the story editor's, as the author believed that potential limitations brought by programming constraints would be more likely found earlier by developing the story editor first. Additionally, creating a low fidelity diagram of the story player would be much more difficult and much less useful than doing the same for the story editor. Certain factors present in the story player that can be improved through evaluation seemed difficult or impossible to gauge with a low fidelity prototype (e.g.: a screen prototype drawn on paper). The characters' placements in dialogue actions, how well the camera tracks the characters in the current action and the general look of the environments and characters (e.g.: The lighting being appropriate or not) are examples of those factors. The story editor could have fewer factors which negatively impact the effectiveness of a low fidelity prototype. Finally, the story player was better defined at the start of development, when compared to the story editor.

The stories used actions as individual story segments. A story is composed of a sequence of actions.

This prototype contained four types of available actions:

- "Estar" (Stay)
- "Andar" (Move)
- "Diálogo" (Dialogue)
- "Escolha" (Choice)

The stay action made the specified character appear at the specified place. The duration field represented the maximum time (in seconds) that this action could last. If the duration was less than 0 or the field did not exist, the duration was treated as if it were infinite.

The Move action made the specified character move from the start place (one of the fields inside the context) to the end place. The action ended once the character had reached close enough to the end place. The character moving played a walking animation, whose speed depended on the character's speed. If the speed was high enough, the character would play a running animation instead. This animation was from the Unity third person template. If no start place is specified, then the character moves from their current position to the end place. If a maximum duration is specified, then the action will stop after that amount of time has elapsed. If the speed is specified, then that multiplier is applied to the character's speed. Otherwise, the multiplier is set to 1.

The Dialogue action showed a conversation between two people. A text box appeared, showing the dialogue in the text field. Additionally, an image was shown if one was included, and a sound clip is played, which was intended as a narration. If a volume was specified, then the sound clip was played at that volume.

The Choice action showed a text box with a question, whose text was from the choice text field, and showed a list of possible responses. When the player selected one of the options, the story would continue at the action specified by that option. Each option was displayed as a text box. If there were more than 3 options, a scrollbar would appear, so that the user can select those options.

Any action could be skipped by pressing the left mouse button, except for Choice actions, which require choosing an option to continue.

4.1.1 Story Player

This version of the story player was a web application created using the Unity engine.

The story editor is composed of two different screens: The main menu, which allows the user to select a story, and the story scene, which plays the story that the user selected, or allows for free exploration of the place, if the user opted for free exploration.

After the story selection menu, the application had 3 main states.

- Loading the story, where the application obtained the file from the server and converted it to a story (Cutscene class).
- Playing a story, where story actions were executed in succession until the story ended.
- Exploration mode, where the user could control a specific character in the scene.

A diagram displaying these states can be seen in figure 4.1.

There was no way of returning to the main menu after playing the story, however, that was planned to be added in the future, since the story player this was based on had that functionality.

4.1.1.1 Story Loading and Playing

The story loading functioned in the following way:

The stories were saved in JSON files in a locally hosted PHP server. Once the user selected a story, the Cutscene Translator class requested the server for the JSON file for that story. When the story player receives the file's contents, they are converted to a story (Cutscene class). If it doesn't

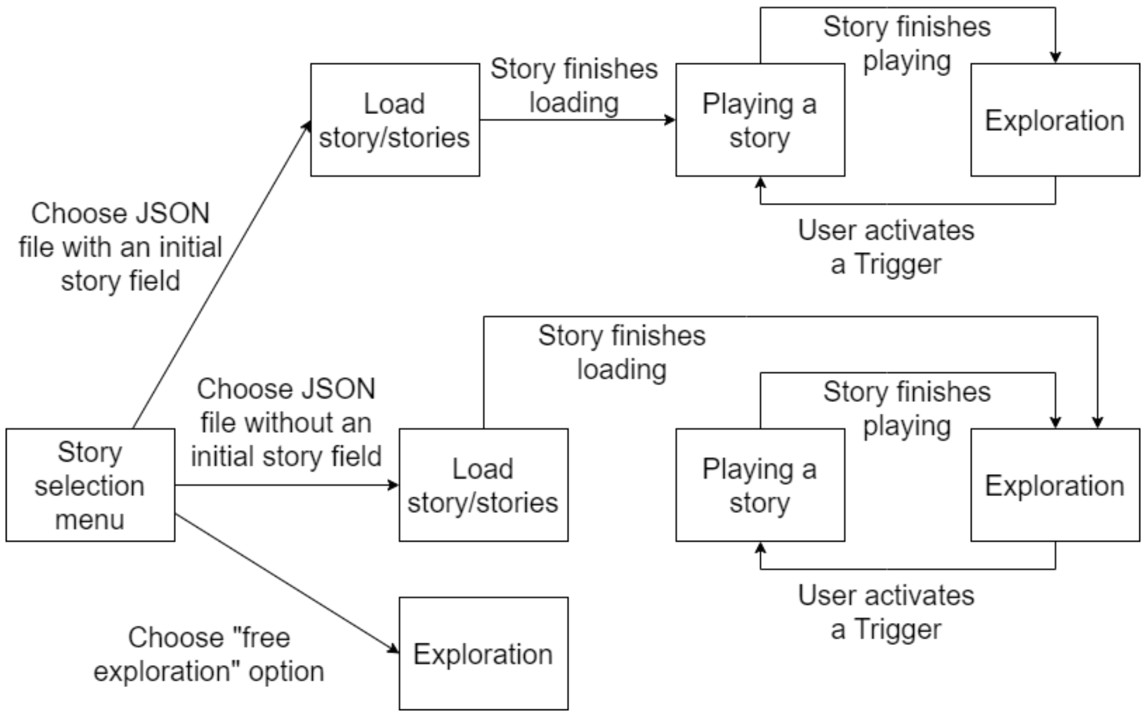


Figure 4.1: First version of the story player: Application states

receive the file, it attempts up to two more times. Each action is represented as a Cutscene Action. Each type of action has a subclass of Cutscene Action which corresponds to that type of action: CutsceneStay (Stay), CutsceneMove (Move), CutsceneDialogue (Dialogue), and CutsceneChoice (Choice).

Figure 4.2 is a diagram describing the architecture for the story loader created in the story player.

When the story finished loading, the story began playing, and certain functionalities were enabled disabled. For example, the controls related to stories were enabled (E.g: Skipping an action by clicking the left mouse button), and the ones related to the free movement were disabled (example: Moving, freely rotating the camera). When the story finished playing, the opposite happened, while also having the necessary steps to change the current state of the application to free exploration occur, such as making the camera follow the character controlled by the user. Each action inside a story contained an ID integer, and an ID integer for the next action. Choice type actions included an ID for the next action for each possible answer. The first action in the story was one whose ID matched the start ID field. The Cutscene Action class contained a constructor, a start action function, an end action function, and a skip function. The first one was called when the action was loaded, which set the action's fields specified in the story JSON, the second one was called when the action started playing, the third one when the action ended, and the fourth one when the user skipped the action. Note that the skip function called the end action function as well. Whenever an action is about to start, a component (script) that extends MonoBehaviour was attached to the actor(s) who that action belonged to. If that character already

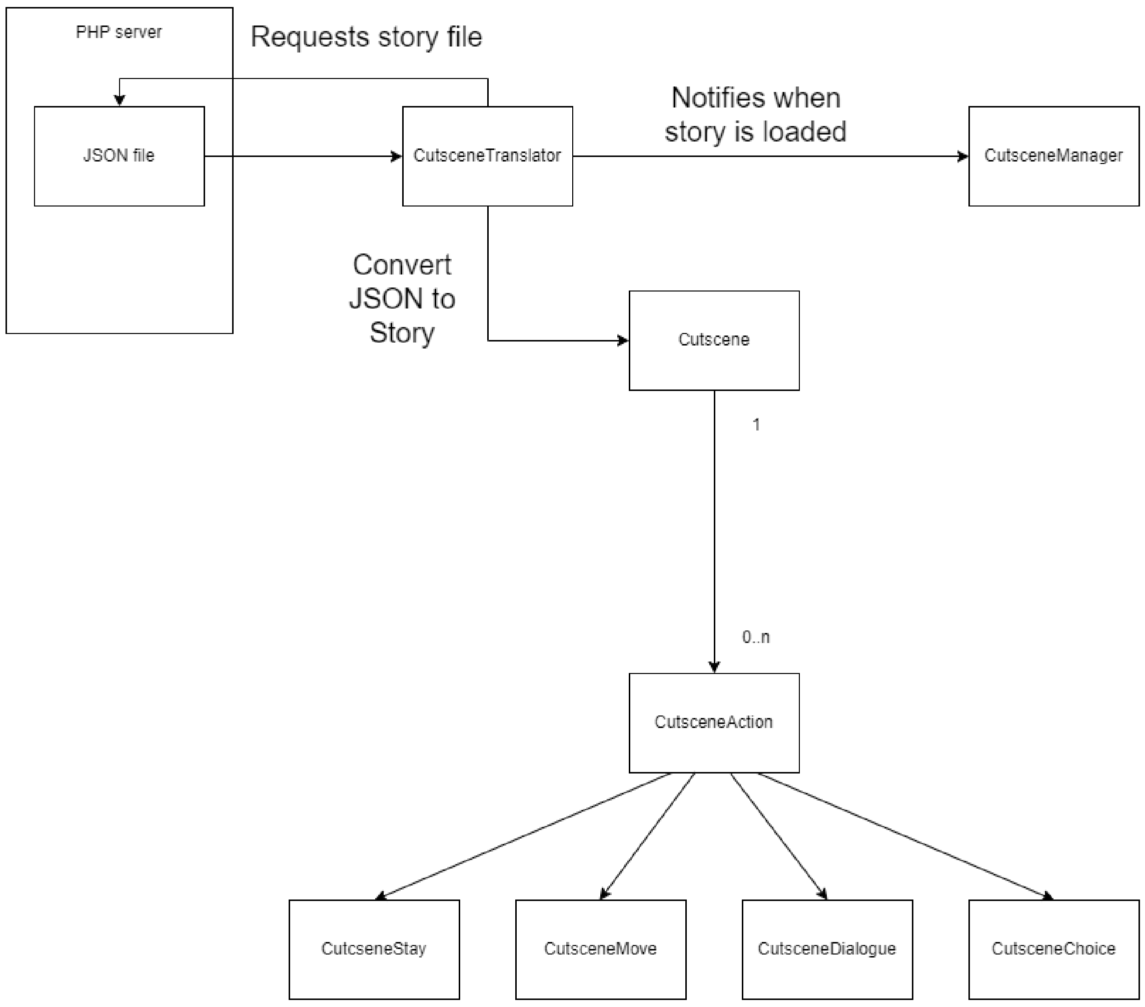


Figure 4.2: First version of the story player: Story loading diagram

had a component with the same name attached, that component was used instead. There was one of these MonoBehaviour components for each type of action (named MBCutsceneStay, MB-CutsceneMove, MBCutsceneDialogue, and MBCutsceneChoice). These scripts are necessary for actions that require MonoBehvaiour functions, such as time based occurances (using the Update function with Time.DeltaTime), or moving characters. These components' start action functions are called by the corresponding action scripts' start functions, and they call the corresponding action scripts' end action functions. Afterwards, the specified character is placed in the specified position, and the rest of the behaviour relative to that type of action is executed. The story ends when an action whose next action ID is -1 ends. A diagram representing the playing of an example story can be seen in figure 4.3.

When the story ends, the user is able to control a certain character. If that character was not set in the story, the character would be placed at (0, 0, 0) xyz coordinates. This was intended to be changed in the future, by, for example, being able to set the character in the story, or by using the last character in a story, or by some other method. The exploration can be seen in figure 4.4.

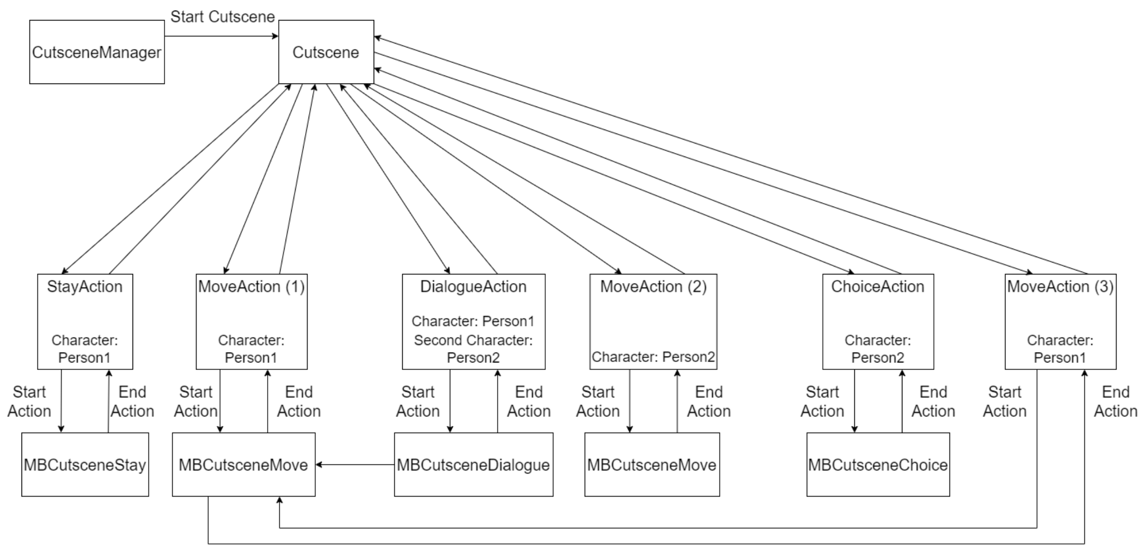


Figure 4.3: First version of the story player: Story playing diagram

4.1.1.2 Story Activation Triggers

After some reunions, it was determined that adding some method for users to interact during the exploration state to play more stories was necessary. As such, triggers were added. Triggers were a series of Unity colliders which could play stories when their conditions were met. These colliders had they only detected collisions, and did not attempt to push overlapping objects outside its collider. The stories that would be played and the conditions that these triggers would activate those stories could be set in the trigger to cutscene JSON file.

There was a new file used for the stories, the trigger to cutscene JSON file. This file contains the start cutscene field, which contains the name of the story should be played at the start, and a list of triggers that, when their conditions were activated, their corresponding stories would be played. In the main menu, instead of the user selecting the name of story directly, the user selects the name of trigger to cutscene file they want to use.

There are two types of story triggers. The first one activated when the user entered it. The second one activated when the user used an item on them.

Below is an example of a Trigger to Cutscene file, along with their corresponding story files.

TriggerToCutscene trigger file:

```
{
    "startCutsceneName" : "Story1.json",
    "triggers": [
      {
        "triggerType" : "place",
        "cutsceneName" : "Story2.json",
        "context" : {
            "triggerName" : "CastleEntrance"
        }
```

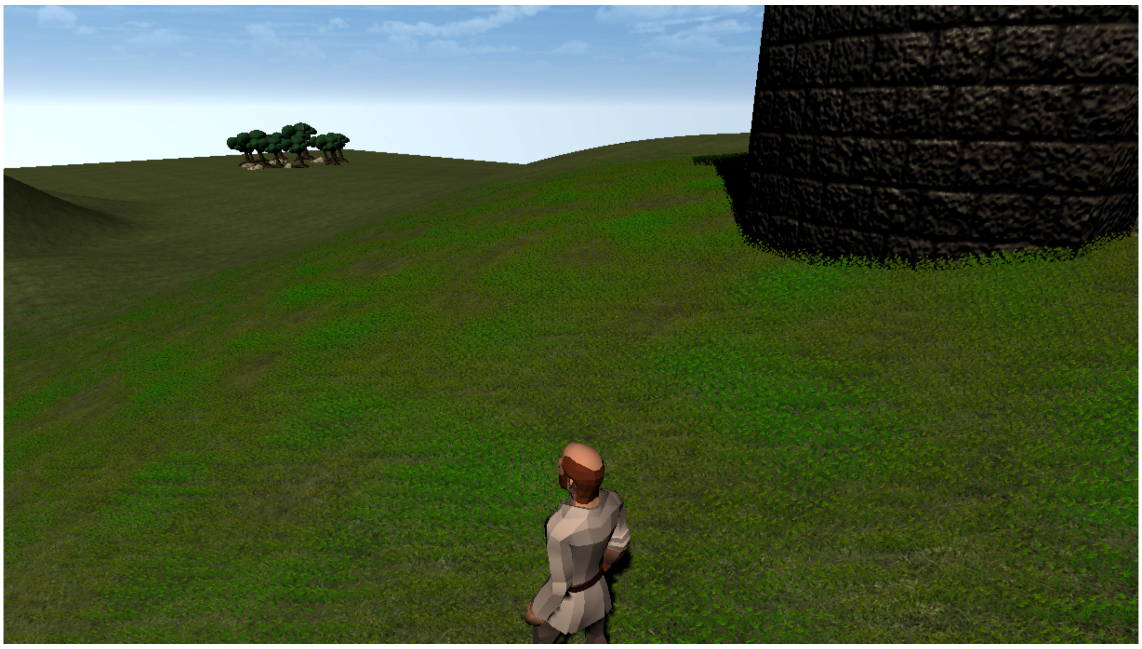


Figure 4.4: First version of the story player: Exploration

```
      },
      {
        "triggerType" : "place",
        "cutsceneName" : "Story3.json",
        "context" : {
            "triggerName" : "TestTrigger"
        }
      }
    ]
}
```

Story1.json story file:

```
{
  "startId" : 1,
  "historia": [
    {
      "verb" : "irpara",
      "actor" : "Knight",
      "place" : "poiTopOfStairs",
      "context": {
          "id" : 1,
          "nextActionId" : 2,
          "speed" : 99,
          "maxDuration" : 15,
          "startPlace" : "poi1"
      }
    },
    {
      "verb" : "irpara",
      "actor" : "Knight2",
      "place" : "poi3",
      "context": {
          "id" : 2,
          "nextActionId" : 3,
          "speed" : 15,
          "maxDuration" : 20,
          "startPlace" : "poiTopOfStairs"
        }
    },
    {
      "verb" : "estar",
      "actor" : "Knight",
      "place" : "poi2",
      "context": {
          "id" : 3,
          "nextActionId" : 4,
          "duration" : 3
        }
    },
    {
      "verb" : "dialogo",
      "actor" : "Knight",
      "place" : "poi2",
      "context": {
          "id" : 4,
          "secondCharacter":  "Knight2",
          "nextActionId" : 5,
          "dialogueText":  "texto testes"
        }
    },
    {
      "verb" : "irpara",
      "actor" : "Knight",
      "place" : "poi1",
      "context": {
          "id" : 5,
          "secondCharacter": "Knight2",
          "nextActionId" : -1,
          "speed" : 15,
          "maxDuration" : 15,
          "startPlace" : "poi2"
        }
    }
  ]
}
```

Story2.JSON story file:

```json
{
  "startId" : 1,
  "historia": [
    {
      "verb" : "irpara",
      "actor" : "Knight",
      "place" : "poiTopOfStairs",
      "context": {
          "id" : 1,
          "nextActionId" : 2,
          "speed" : 10,
          "maxDuration" : 15,
          "startPlace" : "poi1"
      }
    },
    {
      "verb" : "irpara",
      "actor" : "Knight2",
      "place" : "poi3",
      "context": {
          "id" : 2,
          "nextActionId" : 3,
          "speed" : 20,
          "maxDuration" : 20,
          "startPlace" : "poiTopOfStairs"
        }
    },
    {
      "verb" : "estar",
      "actor" : "Knight",
      "place" : "poi2",
      "context": {
          "id" : 3,
          "nextActionId" : 4,
          "duration" : 3
        }
    },
    {
      "verb" : "dialogo",
      "actor" : "Knight",
      "place" : "poi2",
      "context": {
          "id" : 4,
          "secondCharacter":  "Knight2",
          "nextActionId" : 5,
          "dialogueText":  "texto testes"
        }
    },
    {
      "verb" : "irpara",
      "actor" : "Knight",
      "place" : "poi1",
      "context": {
          "id" : 5,
          "secondCharacter": "Knight2",
          "nextActionId" : -1,
          "speed" : 15,
          "maxDuration" : 15,
          "startPlace" : "poi2"
        }
    }
  ]
}
```

Story3.JSON story file:

```
{
    "startId" : 1,
    "historia": [
      {
        "verb" : "irpara",
        "actor" : "Knight",
        "place" : "poiTopOfStairs",
        "context": {
            "id" : 1,
            "nextActionId" : -1,
            "speed" : 50,
            "maxDuration" : 25
        }
      }
    ]
  }
```

The story loading process is different as well. The Cutscene Translator firstly requested the server for the trigger to cutscene JSON file. Once that file was loaded, it parsed that file and requested all story JSON files that are either in the start cutscene field or any of the trigger fields. Once all those files are loaded, the image and sound files included in those stories are loaded. Once all those files are loaded, the story defined as the starting story began playing. If there was no start story, then the game begins in the exploration state. If the user clicks on the free exploration, the game beings in the exploration state and all triggers are disabled.

The player character had an inventory. The user could add items to the inventory by pressing the E key. These items were static and placed on the ground. The user could then use those items by clicking on them in the inventory. The inventory could be opened by pressing the I key. If the correct item was used inside the correct trigger, the story indicated in that entry of the trigger to cutscene file would begin to play. There was no way to place items in the story in the story file structure.

For the camera, this project used the Cinemachine Unity package. The Cinemachine package was used to automatically control the camera during stories and exploration. During stories, the camera had two different presets, one for dialogue actions, which had two characters, and one for the other type of actions, which only have one. Most actions used a cinemachine virtual camera with a third person follow following the action's actor. For dialogue actions, it used a group composer looking at a Cinemachine Target Group that contained both characters in the scene. This version did not attempt to force the camera to not be in such a position and angle that the actors would not be visible, due to an object being in front of them. During the exploration phase, it used a third person follow camera following the actor of the current scene. The user could rotate the camera in any direction by moving the mouse (that code was adapted from the Third Person Unity template).

4.1.2 Editor description

The first iteration of the story editor was a screen prototype made using sheets of paper. It included the following functionalities:

- Creating a new story.
- Loading a story.
- Saving a story.
- Translating a story to another scenario.
- Adding a new action.
- Changing the value of fields inside an existing action.
- Connecting two actions.
- A top-down image containing the story recipe's 3D scene, which can be zoomed in and out.
- A window containing a more zoomed out view of the story

4.1.2.1 Story Related Functionalities

For the new story, open, save, and translate menus, there were two buttons at the bottom, one to cancel the corresponding operation, and another to confirm it. Those will not be mentioned in the individual descriptions of each menu.

The create new story menu included a few elements. Firstly, a text input field for the new story's name. Under the text input field, there was another text input field and a dropdown to select the recipe. The text written by the user on the former was used as a filter for the recipes included in the ladder. This menu can be seen in figure 4.5.

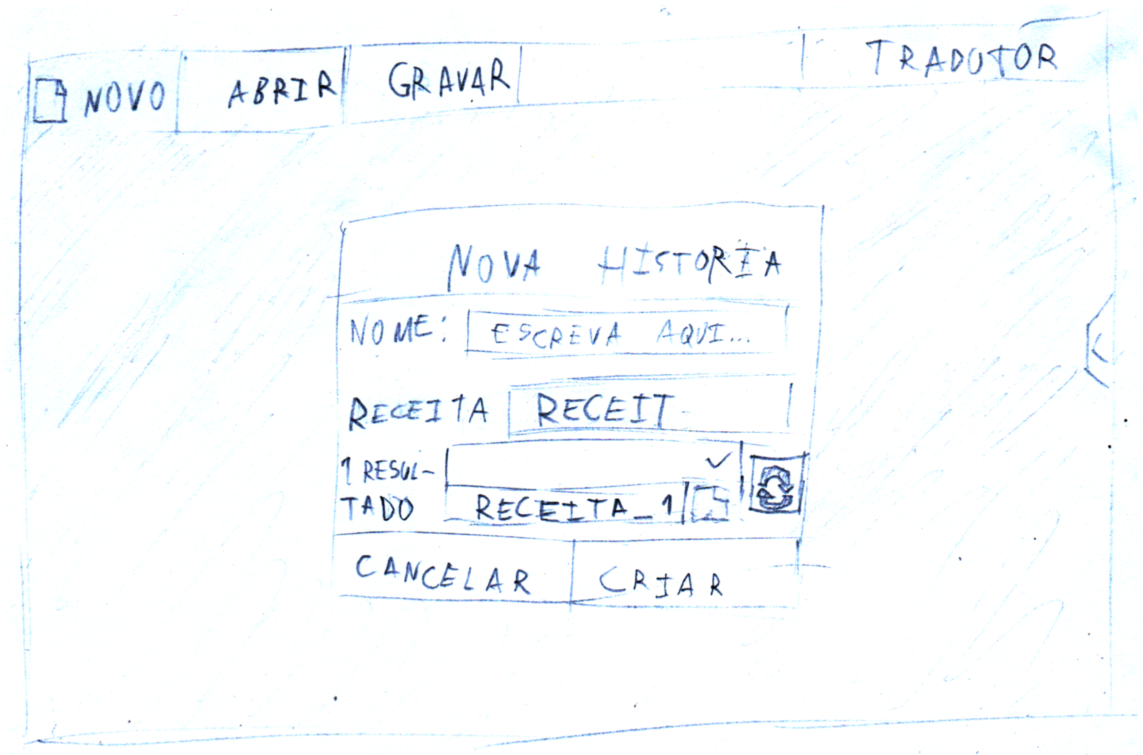


Figure 4.5: First story editor prototype: New story menu

The open story menu included a dropdown to list the stories and a button that reloads the content on the dropdown. Each entry in the dropdown contained its story's name and image, as well as its recipe's name and image. This menu can be seen in figure 4.6.

The save story menu included a text input field, which was used to name the story being saved. If the user had chosen a name that had already been used for another story, then another menu would appear, asking the user if they would like to replace that story with the one they are currently saving, or to go back to the save menu to choose a different name for the story. The save and save confirmation menus can be seen in figures 4.7 and 4.8.

The "translate" menu allowed users to change the recipe associated with the current story. When this action is complete, if the old story had been saved previously, it will still be saved using the recipe, in addition with the version of the story using the new recipe. The translate menu includes a text field for the name of the translated story, a text field to search for a recipe, and a dropdown, which lists all the recipes whose name includes the text in the recipe text field. When the user input the new story's name and selected a recipe from the dropdown, they would be able to press the translate button, which would bring them to another menu. In that menu, in order to translate the story, the user would have to manually associate each character used in the current recipe to a character, respectively, from the target recipe. The same thing applied to places and

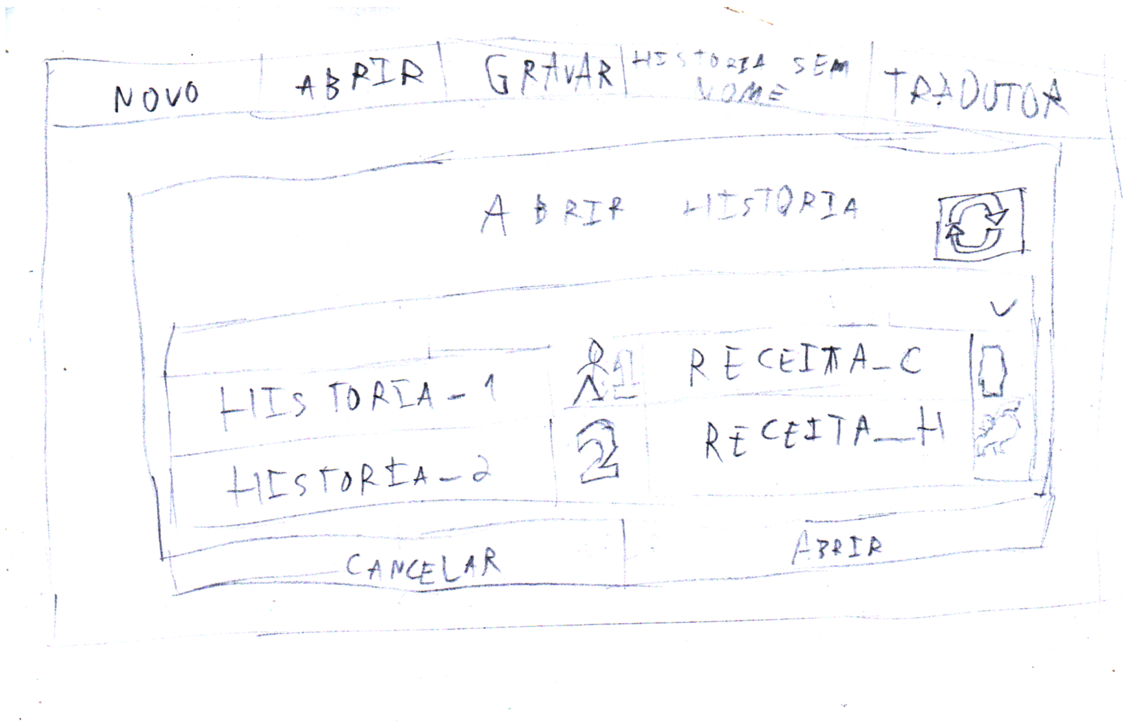


Figure 4.6: First story editor prototype: Open story menu

action types. This had to be enforced because different recipes included different sets of characters and places. The translation menu can be seen in figures 4.9 and 4.10.

This prototype included a tab on the right which could be opened or closed by clicking on the arrow shaped button to open it. This tab included the button to close it as well as a smaller view of the story and a top-down view of the map from the story editor. The place map was able to be zoomed in and out through two buttons located under the map. The place map had a button above it, which allowed the user to add a new location. This prototype did not specify how that location would be added, however, the idea was that clicking the button first then clicking somewhere on the map would cause a new place to be created. The x and z coordinate of that place would be stored somewhere in the story. There was no intended method of being able to name that place. The coordinates that would be used inside the story would be determined when the story was played. First, a Raycast would be done at those x and z coordinates and a y coordinate above the entire scene pointing towards negative y, which would then collide with the point with the highest y coordinate and the corresponding x and z coordinates. After that, the nearest point that is inside a NavMesh would be found and used for that place. The map menu can be seen in 4.11.

4.1.2.2 List of Actions

This prototype included six different actions: Stay ("Estar"), Move ("Andar"), Dialogue ("Dialogo"), Choice ("Escolha"), Hide ("Esconder") and Show ("Mostrar"). Each action contained a group of fields, which were text inputs (some which accepted any text; others which only accepted numbers), dropdowns with an image, in the case of fields which selected a character or place, or

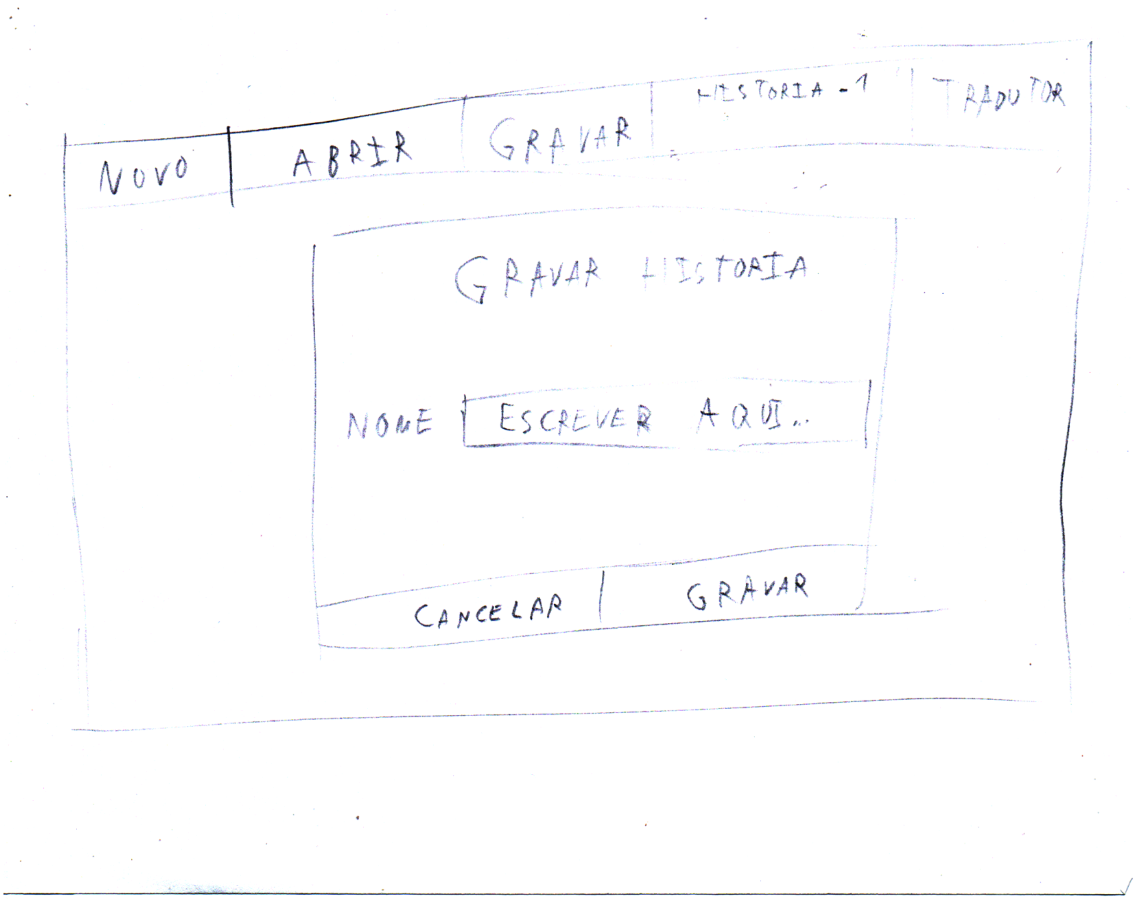
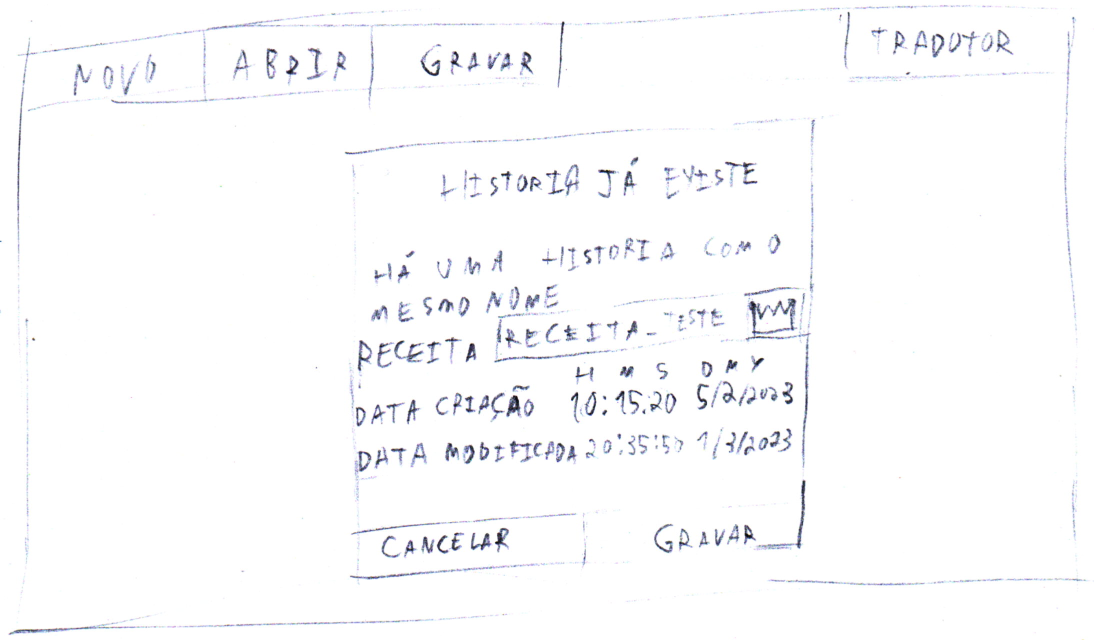


Figure 4.7: First story editor prototype: Save story menu

buttons, in the case of Choice type actions. The stay, move, dialogue and show actions could be skipped (meaning that the action immediately stops playing and simulates being completed) by clicking the left mouse button. The choice action led to different parts of the story depending on the option chosen by the player and, as such, could not be skipped while the Hide action was instantaneous and, as such, could also not be skipped. The Stay ("Estar") action placed the actor at the place dropdown ("Local"), while also making the camera show the actor ("Ator") dropdown. The action would last until the user pressed the left mouse button. Every field in this action is obligatory. The Move action moved the actor from the place to the destination ("Destino"). If there was no (start) place and the character was already in a place, the character would commence moving from their previous location. The maximum time ("Tempo Max") field, if specified, determined how much time the character would until the action automatically ended. The speed ("Velocidade") field determined the speed at which the character would walk towards the destination. The maximum time and speed fields were optional, and the place was optional, as long as the character was already in a certain place before. The Dialogue action represents a conversation between two characters. The actor at the place rotates towards the second character (and vice-versa), while the text from the text ("Texto") field appeared and, if specified, the image from the Image URL ("URL Imagem") field appears and the sound in the Narration URL ("URL Narração") field



HISTORIA JÁ EXISTE

HÁ UMA HISTORIA COM O MESMO NOME

RECEITA RECEITA_TESTE

H M S D M Y

DATA CRIAÇÃO 10:15:20 5/2/2023

DATA MODIFICADA 20:35:50 1/3/2023

Figure 4.8: First story editor prototype: Save confirmation menu

plays once, when that action begins playing. The camera pointed from the character talking to the character listening. This prototype did not include a second character field due to a lapse. The text, image URL and narration URL fields were optional. The Choice action placed the actor in the target place and displayed both a question, whose text was from the text field (the top field named "Texto") and a sequence of buttons containing the text in the corresponding text field (the bottom field(s) named "Texto"). Under the question text field, there were two buttons. The one of the left appended a new empty answer to the question, while the button on the right deleted the bottom most answer. Each choice contained a text and an output. One of these answers' output directly connecting to a certain action means that that action would be played when the player selected that option. The question text field is optional, as well as the number of answers being variable (min. 1). The choice action itself included an output, like the other types of actions, however, it did not serve any purpose. Note that, in this prototype, there was no way to reorder the answers to the quiz field. This means that, in order to remove the first answer, every other answer would have to be removed and re-added. The hide action removed the specified character from the story. That character could still be readded to the story by being used in an action. The show action displayed an image on the screen for a certain amount of time. If no time was specified, then it would be interpreted as infinite time, meaning that the action would last until it was skipped.

The actions included in the editor can be seen in figure 4.12.

Every action had one input and one output, except for Choice actions, which have one output for each answer they have. Lines could be placed between an option's input and an option's output. If an action's ouput was directly connected to another action's input, that meant that the ladder action succeeded the former. For choice actions, the same applied, however, the next action

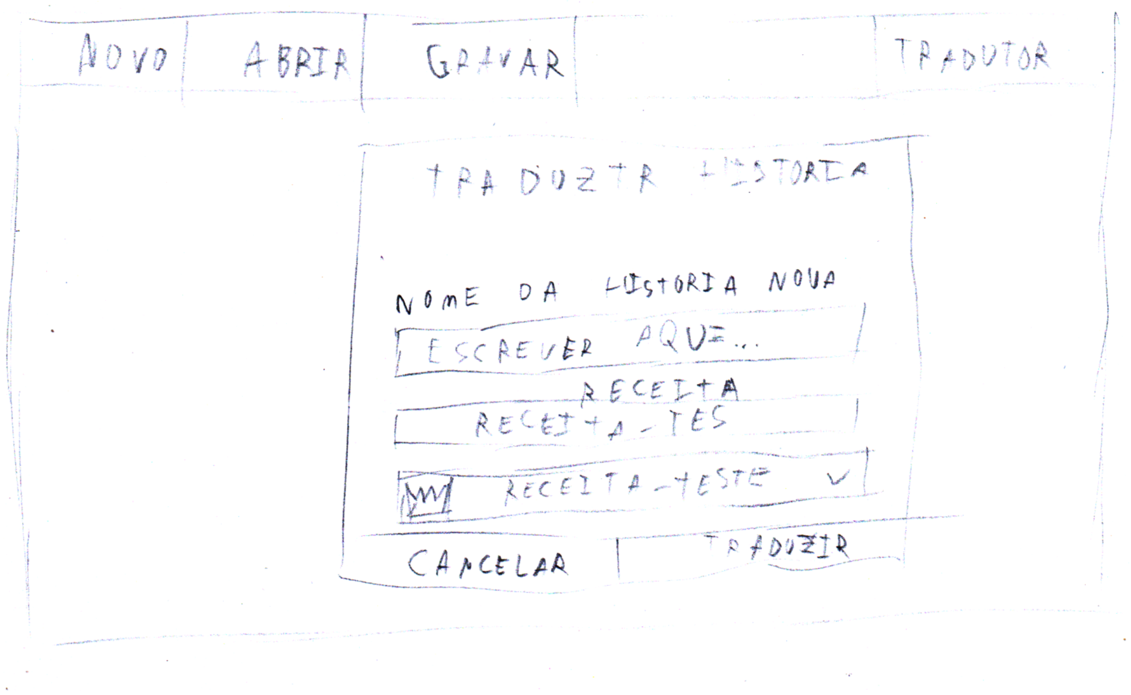


Figure 4.9: First story editor prototype: Translate menu - Recipe selection

would be the action connected to that answer's output. The user could click an input or output to begin the line, which would follow the user's cursor until they either clicked the left mouse button on an output or input, respectively, or clicked the right mouse button, which erases the line. Clicking on a connected input or output with the right mouse button undid that connection and erased the line. For a story to function, there should be exactly one action without an input, which is the start action. Actions that do not have outputs end the story when they finish. There can be multiple endings to a story, since the Choice action allows for branching. Note that this prototype was not planned to check for infinite loops.

4.2 Unity Story Editor Prototype

A partial Unity implementation of the story editor prototypes was created. This implementation did not include all the features from those prototypes, as a new design for the story editor was suggested, and because using Unity for a for the story editor, which is a web application, was not ideal.

It should be noted that recipes were not implemented in this prototype.

4.2.0.1 Main Screen

The main screen of the application contains a header containing buttons to erase, save, and load a story, along with a button to add an empty action to the story and duplicate the selected action in the story. There were scrollbars at the sides of the screen which allows the user to move through

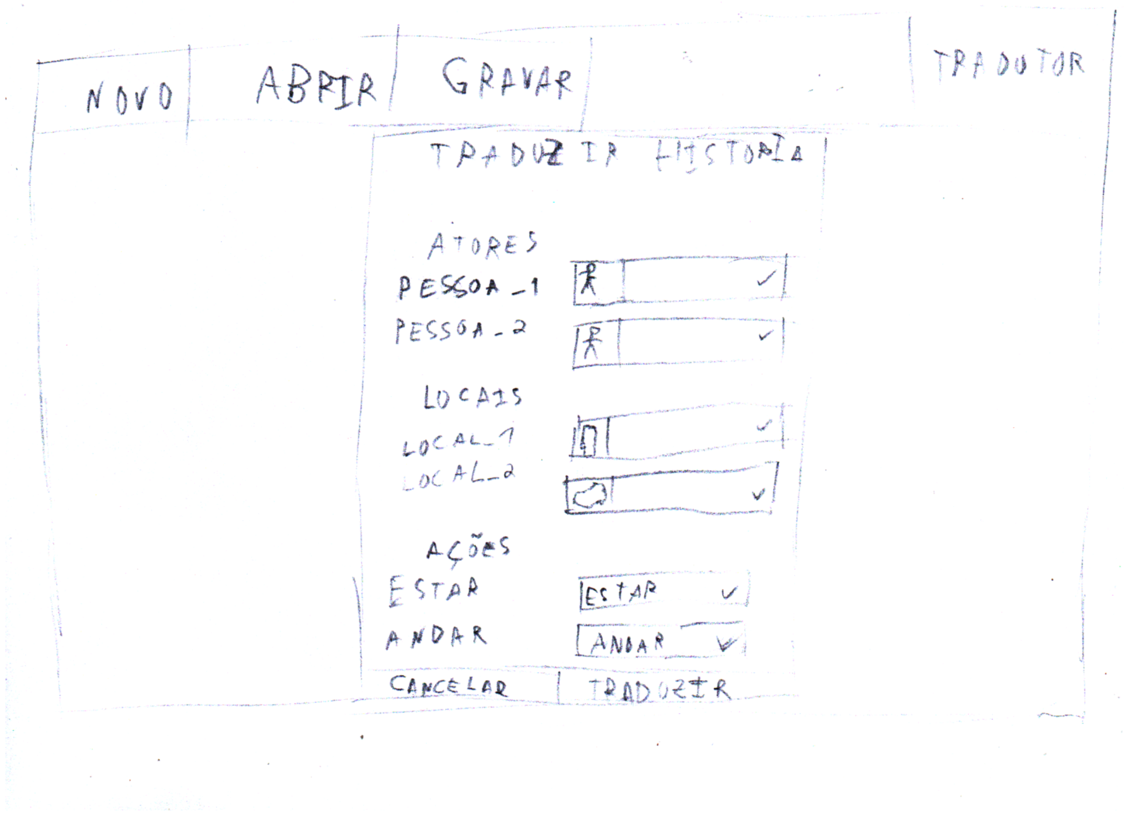


Figure 4.10: First story editor prototype: Translate story menu - Field translation

the story panel and assist in displaying the size of the panel and the position in that panel. The panel can also be moved by moving the mouse while holding the left mouse button, in the opposite direction of the movement. The main screen can be seen in figure 4.13.

Additionally, the main screen contained a scrollable panel displaying the current story, as well as buttons in the bottom right corner to zoom in and out, as well as a button that, when clicked, opened a menu that shows the characters, places, and objects available. The menu that displays story contents can be seen in figure 4.14.

4.2.1 Actions

The action type selected for each action determined which fields appear in that action. This was done by enabling the game object that contains the fields for the currently selected option and disabling all the others. This implementation had two consequences. Firstly, for a certain arbitrary action, if the user set a field from a certain action, then changed the type of that action, that field will no longer be shown, but that field will still have the value stored. For example, if the This value would not be saved, since, for each action, only the fields relative to that action's current action type are saved. Additionally, if two types of actions included the same field, if an action was changed from one of those types to the another, that field would have to be filled in again. For

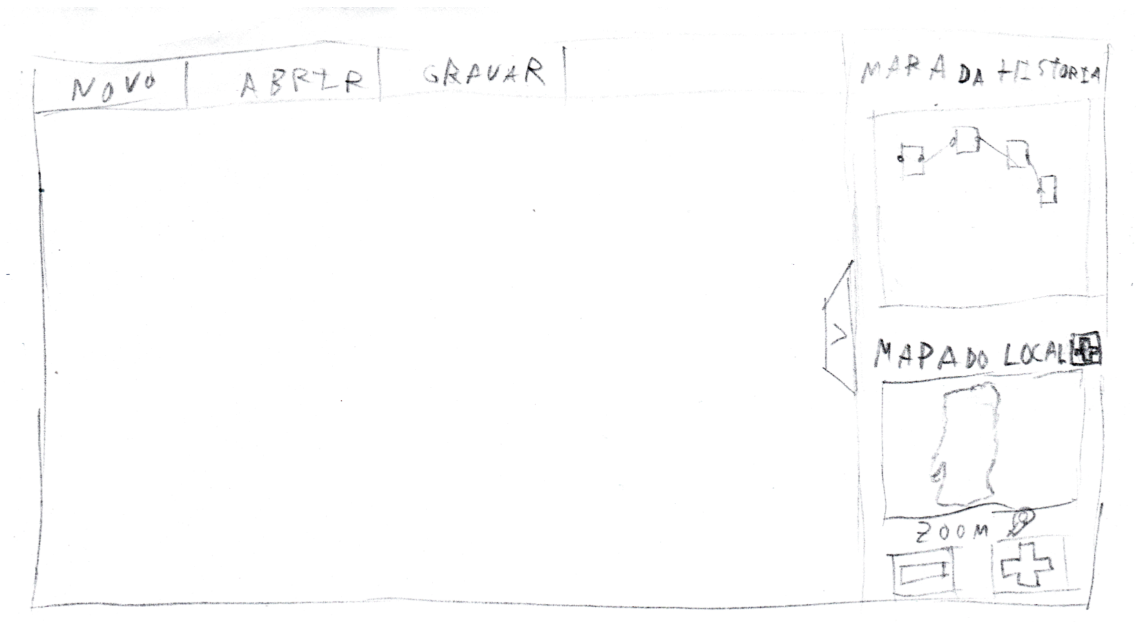


Figure 4.11: First story editor prototype: Map menu

example, if stay type actions and dialogue type actions both had a text field, and the user changes the action type of an action from stay to dialogue, the text field would have to be filled in again.

The user can drag an action by clicking the left mouse button over that action, and then moving the mouse while holding the left mouse button. The action that was last clicked with the left mouse button by the user was considered "selected" and had a white border surrounding it. This is used for the clone button, which duplicates the currently selected option.

The actions in this story editor prototype can be seen in 4.15.

This editor does not have a method to "erase" a story. Additionally, this editor does not include the ability to create a new story. The only ways users can create a new story is from loading an existing story, modifying it, and then saving it under a different name, or by closing and reopening the story editor.

4.2.2 Story Loading

The load menu contained a dropdown that listed all the available stories. Once the user selected a story they wanted to load, they could click the load button to open the story. This menu can be seen in figure 4.16.

4.2.3 Story Saving

The save menu contained a text field for the story's name, a button to save the story, and text that displays whether the story has been saved successfully, failed saving, or is currently saving. The text field was empty until the user clicked on the save button. This menu can be seen in figure 4.17.

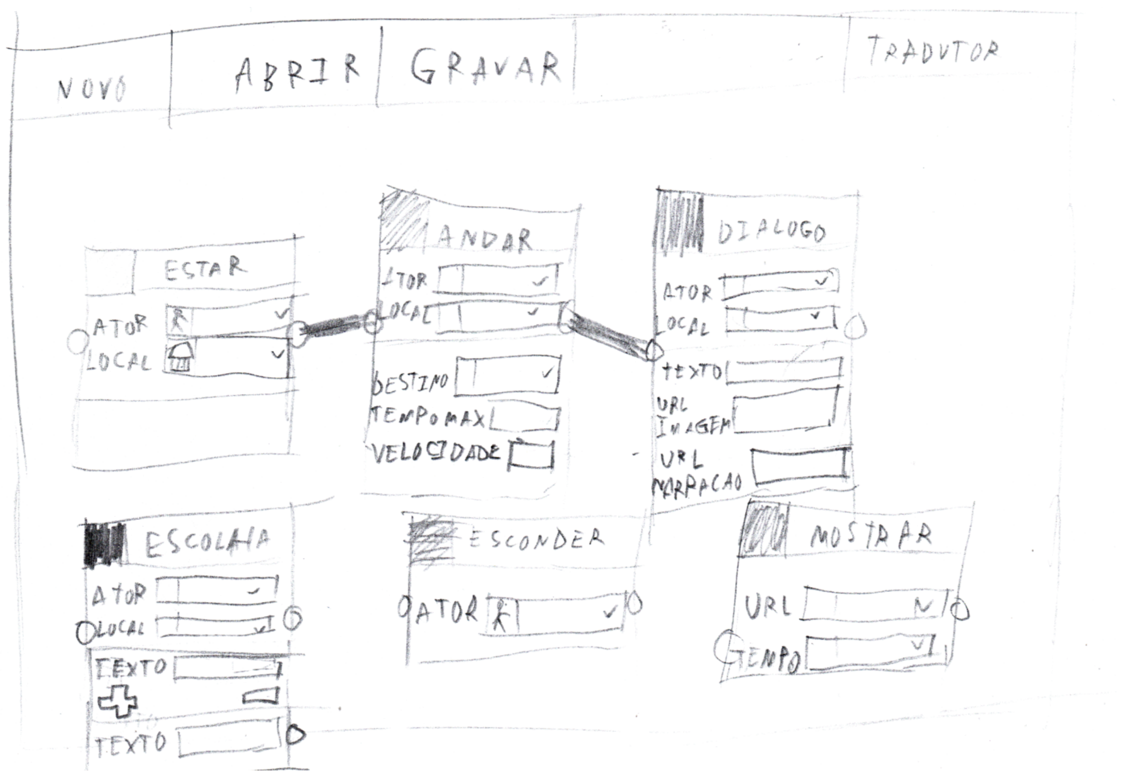


Figure 4.12: First story editor prototype: List of actions and their fields

If a story had the name the user chose for the story, a prompt would appear, asking the user if they would still like to save. This prompt included the name, creation date, and last modified date of the story that was already saved. If the user clicked the "Save Anyway" button, the story would overwrite the previous story with the same name. Otherwise, the story would not be saved. This menu can be seen in figure 4.18.

The stories were saved into two JSON files. The story file was the JSON file that contained the story that was loaded by the story player. The placement file contained the placement of each file in the editor. When the user saved a story, its actions would be saved in order of creation. The fields from the action (only including the fields that belong to any type of action and for the selected action type for the corresponding action) were placed in a fixed order.

The classes used to represent a story are presented in the listing below. These classes are directly serialized to JSON when the story is saved and are directly deserialized into those classes.

```

[System.Serializable]
public class StoryJSON
{
    public int startId;
    public ActionJSON[] historia;
}

```

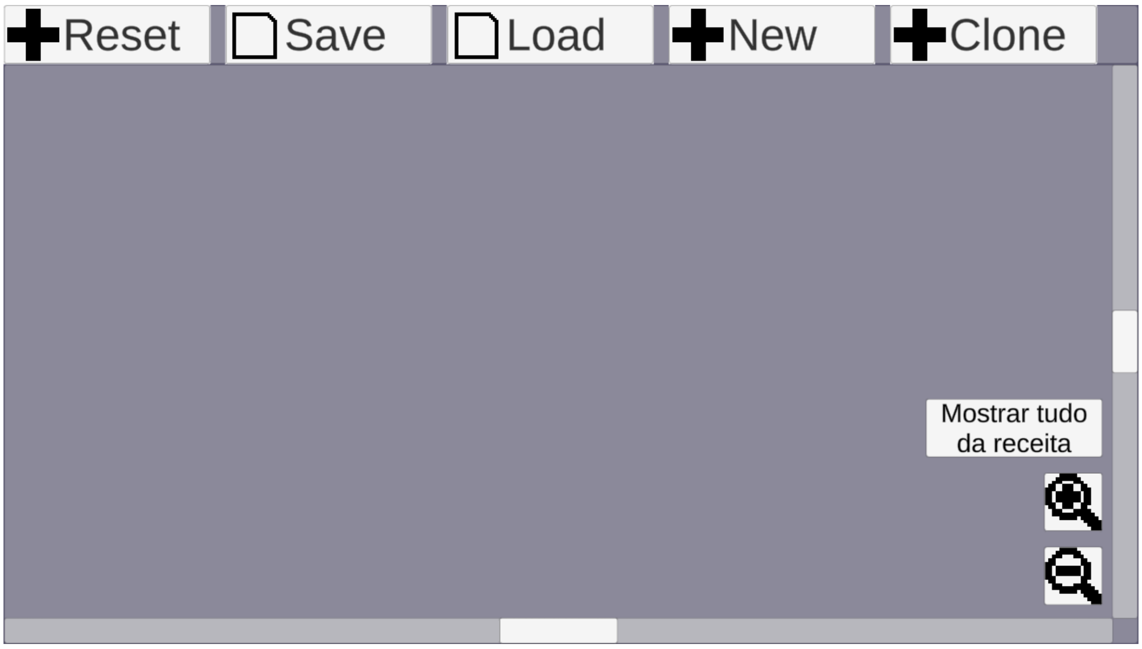


Figure 4.13: Unity Story Editor: Main Menu

```
[System.Serializable]
public class ActionJSON
{
    public string verb;
    public string actor;
    public string place;
    public ActionAllContextJSON context;
}

[System.Serializable]
public class EstarActionContextJSON: ActionContextJSON{


    public int nextActionId;

    // Appear
    public float duration;
}

[System.Serializable]
public class IrParaActionContextJSON: ActionContextJSON{
    public int nextActionId;
    public string startPlace;
    public float speed;
    public float maxDuration;
}

[System.Serializable]
```

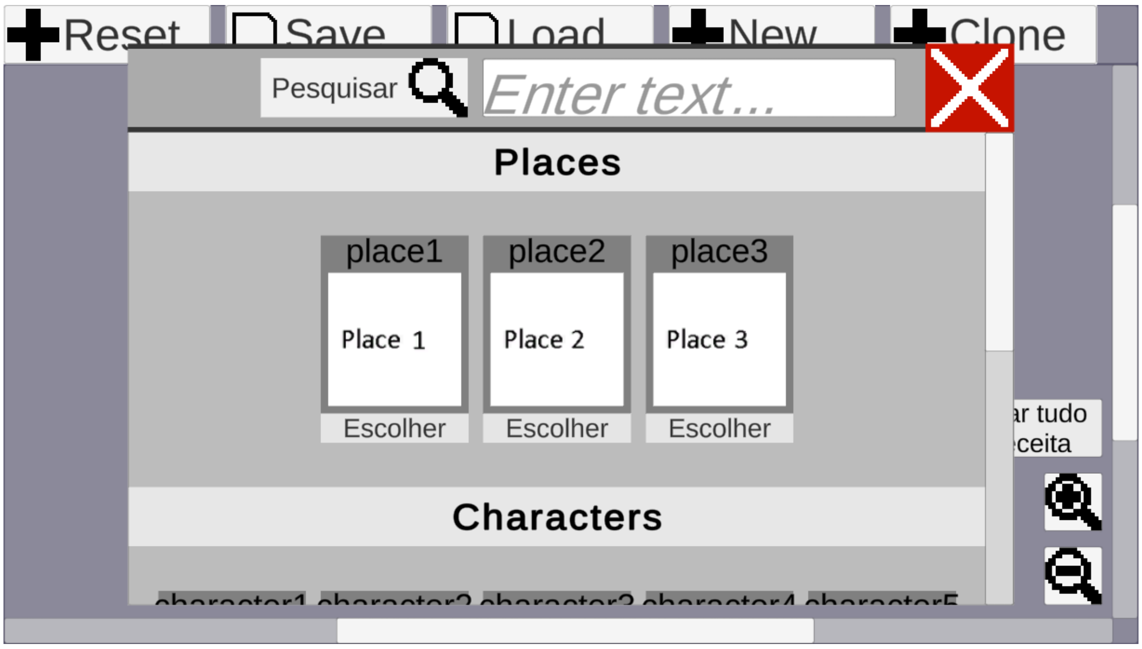


Figure 4.14: Unity Story Editor: Display Story Contents

```
public class DialogoActionContextJSON: ActionContextJSON{
    public int nextActionId;
    public string dialogueText;
    public string secondCharacter;
    public bool isResponse;
    public string imageURL;
    public string dialogueSoundURL;
    public string volume;
}

[System.Serializable]
public class EscolhaActionContextJSON: ActionContextJSON{
    public int nextActionId;
    public string choiceText;
    public ActionContextChoiceJSON[] choices;
}

[System.Serializable]
public class EsconderActionContextJSON: ActionContextJSON{
    public int nextActionId;
}

[System.Serializable]
public class ActionContextJSON{
    public int id;
}

[System.Serializable]
```

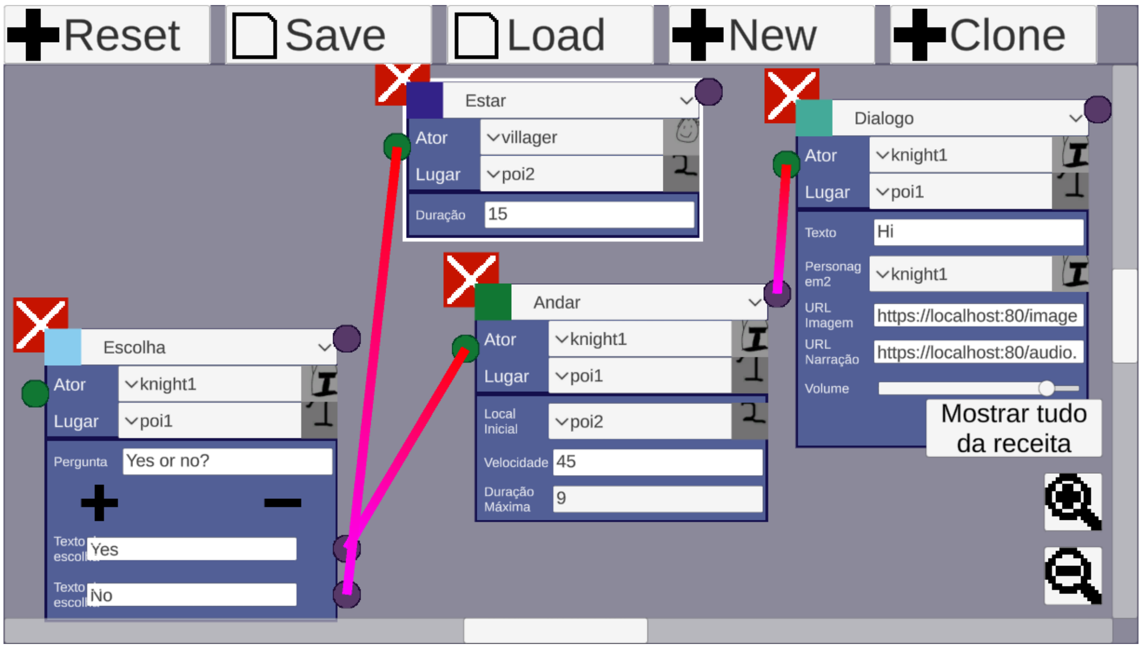


Figure 4.15: Unity Story Editor: Main Menu with Actions

```
public class ActionContextChoiceJSON{
    public string optionText;
    public int choiceActionId;
}

[System.Serializable]
public class ActionAllContextJSON{
    public ActionAllContextJSON(string actionType){
        this.actionType = actionType;
    }
    public string actionType;
    public int id;
    public float duration;
    public string startPlace;
    public float speed;
    public float maxDuration;
    public string dialogueText;
    public string secondCharacter;
    public bool isResponse;
    public string imageURL;
    public string dialogueSoundURL;
    public string volume;
    public string embedLink;
    public string choiceText;
    public ActionContextChoiceJSON[] choices;
    public int nextActionId;
}
```

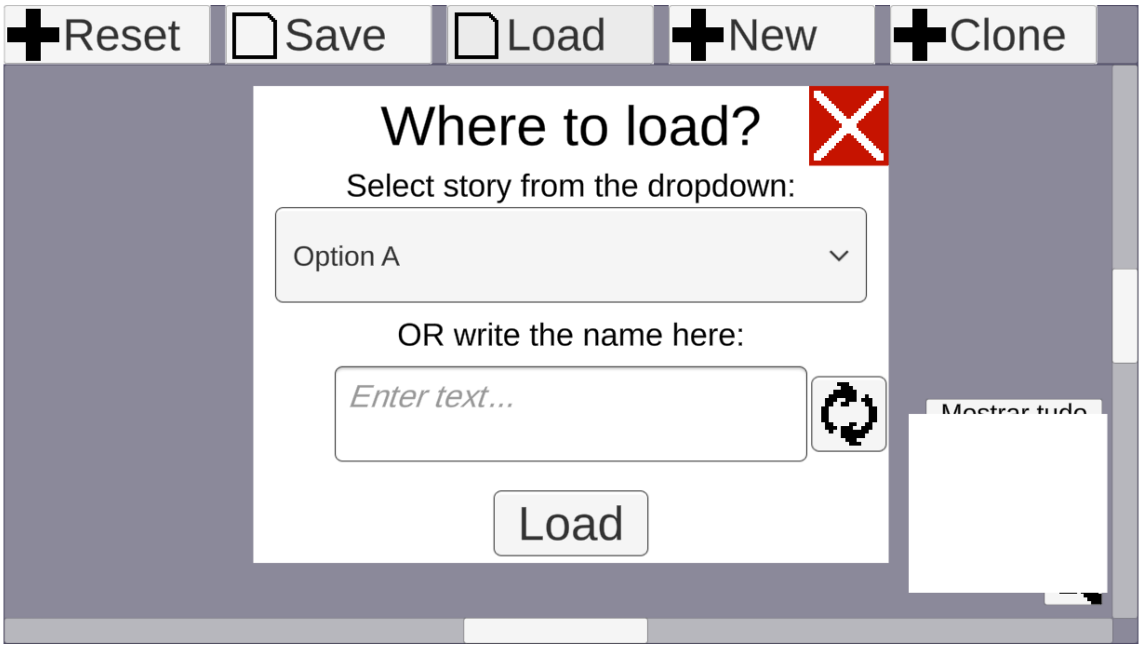


Figure 4.16: Unity Story Editor: Load Menu

4.3 First Story Editor - Evaluation and Feedback

This prototype of the story editor was first shown in an in-person meeting to 2 people. Afterwards, more feedback and requirements were given by one of the aforementioned people as well as thesis advisors.

4.3.1 Feedback

This subsection presents the feedback obtained from the meetings where the story editor.

The ability for the users to add and use videos and 3D models should be added, in some form.

When opening a story, the dropdown that shows the stories should also show what recipe they belong to. Possibly also date of creation and last modification. Additionally, an image of the story and recipe, which would be supplied by the user and the recipe creator, respectively, could be shown.

Objects should also be included in the stories. As such, there could be a list of "objects" and a list of "characters". Within the story player, they would have the same behaviour and abilities, however, they would belong to two different lists inside the editor.

Users should be able to name actions. This point was raised because it was difficult to quickly identify what each individual action represented in a story, especially without looking at the entire story. For example, a stay action with a character "Maria" at the place "Park" tells us little about the context of the story at that point.

At this point in time, the content, including the images, was stored in a database. A member of the team suggested that it would be easier to store the files in the server's filesystem.

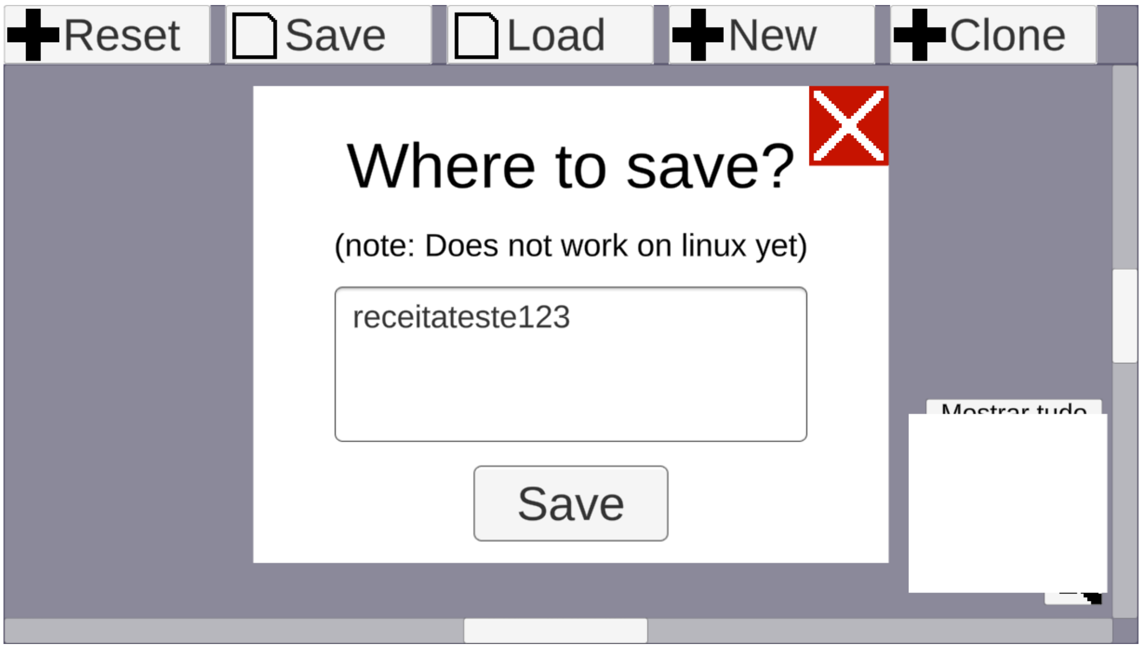


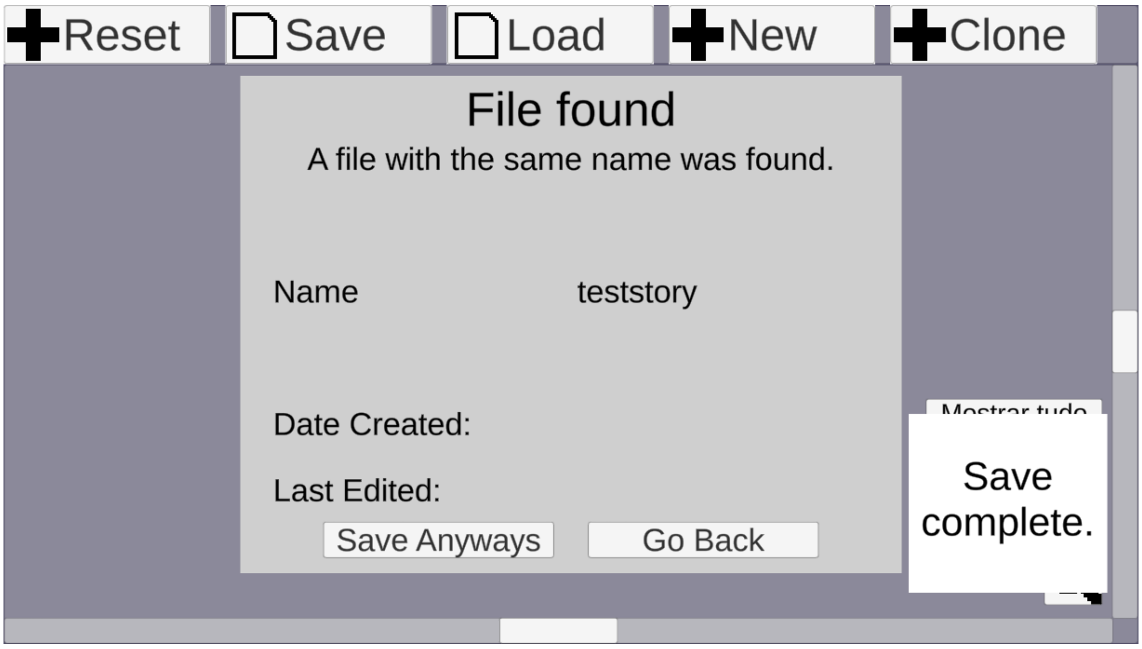
Figure 4.17: Unity Story Editor: Save Story

4.4 Improved story editor: Development

The second prototype demonstrated was also created on paper, however, this time it was supplemented with a partially completed version of the editor in the Unity engine. This prototype improved on the prototype described in section 4.3, meaning that it included the functionalities described in that subsection.

The following features were added from the first version of the editor:

- Buttons were added at the bottom right of the screen to zoom in and out, respectively.
- When the user tried to create a new story while the story that was already open had unsaved changes, they would be taken to a screen asking if the user is sure they want to create another story without saving the current story.
- The open story menu's dropdown now lists the date of creation and last modification, in addition to the previous fields. The same dropdown does not display the story's image, since there would no longer be one. Additionally, the layout has been slightly altered to make the intent of the search text field more obvious.
- A button was added to see all the content in a recipe (list of places and characters).
- A dropdown was added at the bottom of the main screen which the user can use to select the starting action.
- The previously existing actions had additional fields added to them.



File found

A file with the same name was found.

Name teststory

Date Created:

Last Edited:

Save complete.

Figure 4.18: Unity Story Editor: Save Story: Story with the same name already exists

4.5 Improved story editor: Evaluation and feedback

At the time, it was discussed whether the zoom in and out feature should be assigned to clickable buttons or to hotkeys. It was determined that hotkeys could be too complicated for the end user, and, as such, should not be the only method of zooming in/out.

The characters and places should have images associated with them, and those images should be visible both in the dropdown entries when selecting a character/place, and as part of the currently selected character(s)/place for each action. These images do not have to be very big, they should specifically be around the size of discord profile pictures.

Each story should have an image associated with it, which would be shown in the list of stories that appears in the open story menu.

Multiple characters should be able to do actions at the same time. For example, two characters should be able to move from one spot to another.

There should be the ability to show two images, the first showing a "before" of a place and the second showing an "after". These images would overlap each other and have a vertical bar that the user can move where, to the left of the bar, the corresponding portion of the "before" image is displayed and, to the right of the bar, the corresponding portion of the "after" image is displayed.

In addition to the zoom in and out buttons, there could be a button that sets the zoom to the default amount.

One suggestion was splitting the characters into characters and objects. Characters would now only represent people or animals, while objects would represent objects (which don't move). This was suggested to not confuse the end users by having objects be a character.

One problem with the current actions is that it's difficult to group these actions together and, as such, when reading the story, it is difficult to know which groups of successive actions happen in the same place, for example. Additionally, it is difficult to understand what each individual action translates to in the context of the story. One of the solutions was that actions should be able to be named. This would facilitate the reading of a story. The user could be able to add notes (similar to post-its) anywhere in the story. The most important suggestion was changing the layout so that, instead of using actions as the individual components, actions would be grouped into acts. These acts would include a certain pool of characters and objects, specified by the player. Actions inside that act would play in the order that they were placed inside the act. Inside an act, actions could be added, removed or reordered. These acts could be named by the user.

The following image, 4.19, shows a variation of the concept. This variation included places in the acts, as well as characters and objects.

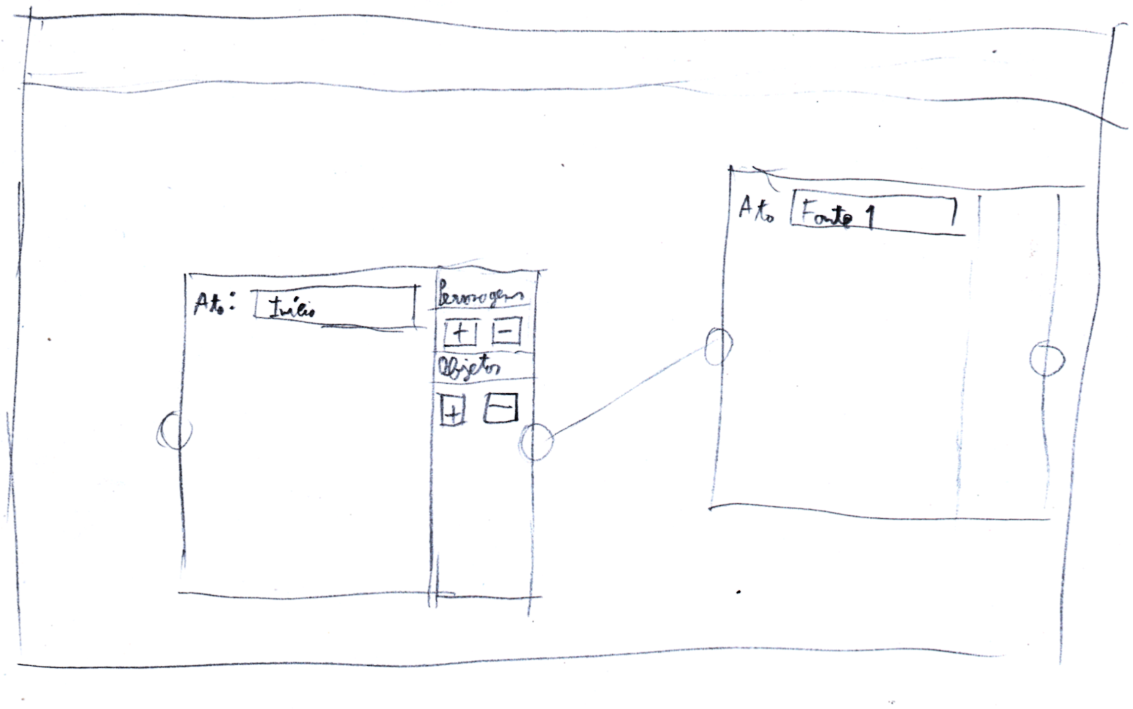


Figure 4.19: New layout suggestion

Another layout was proposed by a member of the team:

Scenes are now the individual draggable components of stories. Each scene contains a place, a list of characters, a list of objects and a list of actions. These actions happen at the specified place using some of the specified characters and/or objects. Each scene has one input and one output. Non-choice actions appear in a scrollable sidebar, similarly as to how they appeared in the last 2 editors, but without the input and output buttons. The user can add a new action by clicking the button at the bottom of the action list. The user can change the input of the connection by clicking the connected output and then clicking on another input, both with the left mouse button. At the end of the action list, a choice action could be added. Since the action list was on a scrollable bar, instead of having a connectable output, the choice action would include two dropdowns for each answer (without images) where the user could select the scene and action where that answer goes to. The historians provided an example story to a member of the team. Using the previous editor format, the story would be represented in 10 actions. Using the new editor design, the story would be represented in 2 scenes, with 10 actions total.

4.5.0.1 Visit to Castelo de Vide

After visiting Castelo de Vide, the place where the 3D scenario is based on, the following potential terrain-based problems were raised:

Character navigation on steep slopes. The slopes were found in the path towards the entrance to the castle inside Castelo de Vide. Character navigation on narrow corridors, especially in conjunction with stairs. This situation happens at the entrance to the wall next to the castle. Interactions with doors opening next to stairs. This situation happens at the entrance to the wall next to the castle.

4.5.0.2 New Story Editor Idea

A member of the team provided a screen prototype for a new story editor design. That design was implemented in the first prototype story editor that was made in React, described in section 4.6.

4.6 React Story Editor (First iteration) - Conceptualization and Development

The final version of the story editor was made in React with Typescript. The tree-like format of the story was replaced with a more linear approach.

A story was divided into sequential scenes, and a scene was divided into sequential actions (called slides in this version, but they will be referred to as actions in this dissertation for consistency).

The user could append a new scene, or a new slide to any scene, but they could neither delete nor reorder slides, or add scenes/slides to anywhere but the end of the list. The user could select a scene/slide by clicking on them. The list of slides shown was for the scene that was selected.

Each action contained the following fields:

- A background image.
- A place, including an image and a name.
- A character, including an image and its associated name
- A text input.
- An additional field button.

The main menu of this application can be seen in figure 4.20.

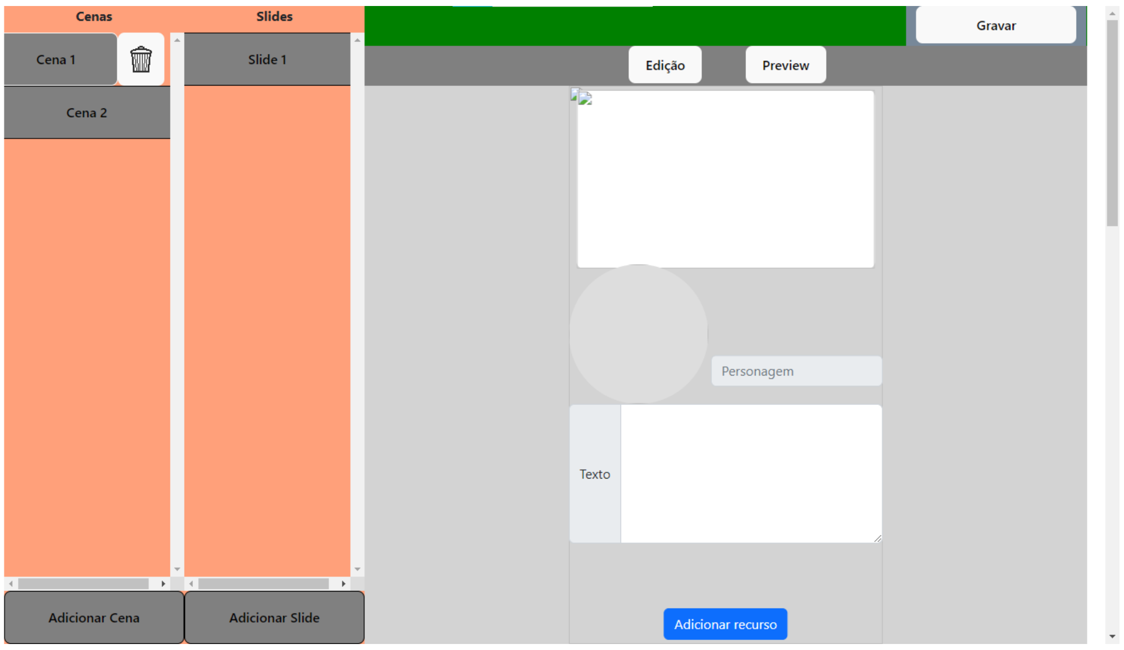


Figure 4.20: First React story editor prototype: Main menu

Each action could have a single additional field, except for image fields, which could also have text associated with them. Additional fields could be an additional text, an image with a text, a set of "before" and "after" images, a 3D model embed from sketchfab, a youtube video embed, and a quiz, which presents a choice to the user and leads to different actions depending on the choice.

This editor included a preview for individual actions, as if they were being viewed in a 2D application. However, clicking on buttons which would normally cause transitions to other scenes/actions did not do anything. Inside this preview, the main fields (background image, character image and name, place image and text) were viewed the same way they would be viewed in the editing menu, except that the clickable buttons in the editing menu which allow the user to add those fields were no longer clickable. At the bottom of the preview there was a button which, when clicked, allowed the user to see the content of the additional field set for that action. If there was no field set for that action, or if it was a destination field, the button would not be displayed. For destination actions, text was displayed, indicating where the user should go, and which object they should look at.

Whenever the character image was clicked, it opened a menu which allowed the user to select the character they want to use for that slide. This menu can be seen in figure 4.21. This menu was a Modal that appeared over the rest of the content. included a top bar with a search bar, a button to add a new character, and a button which closes the menu. Under that top bar, the characters whose names contained the search string were shown. Each character included their image on top, their name under the image, and a button to select it under the name. There was a bar at the bottom which included two buttons, both of which only closed the menu. The "Save menu" button was not meant to be included in this menu. Note that every character and their images was loaded at once in this list, which would cause problems with large lists.

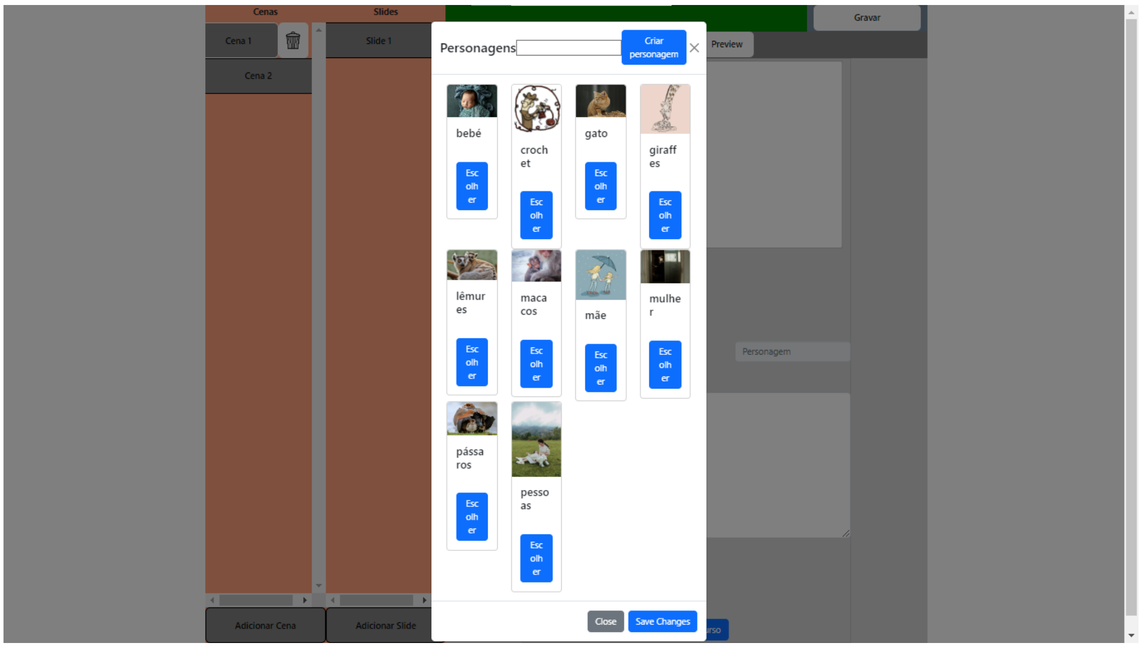


Figure 4.21: First React story editor prototype: Character menu

The create character menu included a top header with the name of the menu and a close button. The middle included a text field for the name and an image field for the character's image. At the bottom, there were two buttons, one to create the character and another to close the menu without creating the character. Note that, in this prototype, a character with no name and no image could be created. This menu can be seen in figure 4.22.

4.7 React Story Editor (First iteration) - Evaluation and Feedback

For image fields, users should have the ability to both use a URL for an image, and to select an image from the server to use. Additionally, the user should be able to save images to the server to then use in those fields. Images in the server should have a name, so that the user can search for them.

A potential 2D story player, if created, should be created with the intent of being used inside Castelo de Vide (or, in case another scene was made in the future, in the corresponding place), and could be used as a complement or guide to the exploration of the region. As such, that player would have to not only have good performance, but the stories, which would be loaded from the server, would have to occupy little space, so that they could be loaded quickly from the server with a poor internet connection, which is to be expected when moving from one place to another. As such, a few suggestions had been made:

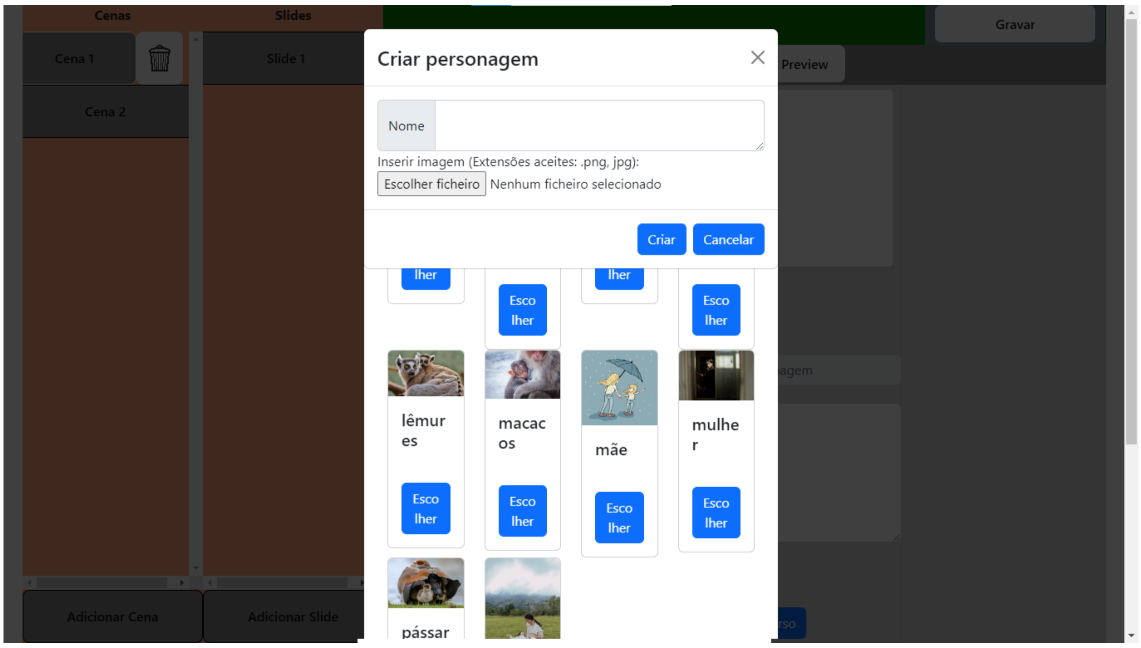


Figure 4.22: First React story editor prototype: Create character menu

- Images should be resized for their necessary purposes. In the case that different resolutions are needed for different purposes, images should be saved to the server in those different resolutions and then loaded accordingly.

- Images inside the story editor and players should not all be preloaded at the beginning of the story. Since the stories have choice type actions, which can lead to any number of branching story paths, loading every image inside the story at the beginning would be impractical with any medium size story. As such, images should either only be loaded when necessary. Since the story editor is made in React, this becomes trivial, as images are lazy loaded by default.

- If images are only loaded for the current action being played, users would have to wait for the images to be retrieved from the server to be able to see them. This problem is exacerbated with slow internet connections. One solution raised was that images should be preloaded for the current action, as well as a certain fixed number of following actions. This number would preferably be relatively small, since some images for actions for paths which the user did not choose would be loaded. Another alternative would be to preload the images for the current action plus a certain fixed number of images from the next actions.

Some method would have to be used to incorporate videos and 3D images in the story player. The main suggestion was that a link to a YouTube video or a Sketchfab 3d model could be input by the user, which would then be embed by the applications playing them. A 2D story player could be made as a web application, so that those embeds can be displayed. However, since the 3D story player is made in Unity and, according to the author, YouTube or Sketchfab embeds cannot be displayed on Unity web application, the embeds would have to be displayed in the page that the instance of the Unity player is being run. Fortunately, Unity supports function calls to JavaScript functions with arguments. Since JavaScript functions can manipulate the DOM (the content in the web page running the Unity application), this means that the Unity application can call a JavaScript function to place the embeds in the webpage, on top of the Unity application. Note that this would not work when the Unity application is in full screen, as the Unity application is placed at the highest possible z-index, so that no other page element could be ahead of it.

Newly created stories should begin with a single slide already created.

Submenus inside an action were originally opened by double-clicking on them with the left mouse button, however, it was decided that a single left click was enough.

Images can have a fixed size for preview/editing. However, it was realized that the size of the image in the preview and editing menus depend on the size of the window that the webpage is on. Images can still be resized to an arbitrary maximum size, though.

The editor should include a menu where the user can add images. More specifically, there should be a menu to add place images, another to add character images and another for objects. Each of these should use a different group of images.

Quiz actions should indicate which scene and which action they point to.

This version of the editor lacked the ability to reorganize scenes/slides or add a scene/slide in any point except the end.

Scenes had no practical purpose, except for allowing the user to divide the story into arbitrary segments. After some discussion, it was proposed that scenes could divide the story by place. All the actions inside a scene would share the same place. At first, the possibility of being able to change the place of actions inside a scene, with an action's place being its scene's place by default, however, after further discussion, the idea was deemed unnecessary, or at least low priority. Places only being able to be defined in scenes instead of the slides was suggested because of the example stories created by the historians in the team frequently containing many actions in a row within the same scene. The name scene was chosen because of its theatrical context, where a scene change is determined by a place change.

Frequently, historical scenarios involve not only individual people, but also crowds of people. As such, it was suggested that single crowds of people should be treated as characters. Additionally, it was suggested that these crowds of people should be able to be placed and move in certain formations, to display historical army formations and how they changed over time. A formation would be defined as a group of points forming a single polygon Finally, it was also suggested that the number of people in the crowd should be able to be defined for each action, since historical army formations change in not only shape but also size.

The user should be able to change a character's name from their default name (for a certain action). A character with the same name and model but named different characters by the user would count as two different characters for the 3D application.

It was suggested that there should be a new type of action, which plays another story from the beginning. This action could be placed at any point in the story. If this action was only able to be placed at the end of a story, this could be recreated by finishing the story and asking the user to play another story, it was decided that too low priority for the time remaining. If this were able to be placed at any point in the story, it would require many changes in the current story player, in addition to potentially creating uncaught bugs.

It was suggested that stories should be able to be triggered by some action caused by the user, such as going into a certain place. However, after some discussion, it was concluded that, since the stories produced from this editor are intended to be played in different story players, adding an action exclusive to the 3D story player that the user would have to specify would go against this project's goals.

4.8 React Story Editor (Final iteration) - Conceptualization and Development

This section describes the changes from first prototype of the new story editor design to the final version of the story editor.

4.8.0.1 Recipes

Recipes have been implemented in this version of the editor. Each story belongs to a recipe, which is selected at the beginning, or when a new story is created. Each recipe contains the list of characters and places that can be used in the story. Recipes are similar to the ones used in the application referred in appendix C, however, in this new story editor, stories cannot be translated.

4.8.0.2 Action changes

The editor now allowed for the creation of quiz actions with the quiz field. The add quiz field menu includes a text field for the question asked to the user and a button to add a new answer. Each answer contains a text field for the text included in the answer, a text field for a URL for the image included in that answer, a delete button and a place to select the scene and slide which selecting that action leads to.

4.8.0.3 Creating stories

This version of the editor starts in the create story menu. In order to be able to create a story, a recipe must be selected, since the recipe determines the environment in which the story will be played. The recipe's name is always displayed at the top of the main menu.

This version of the editor included the ability to create a new story. Whenever the user clicks the new story button, a menu appeared which allows them to select the recipe they want to use for the new story. This menu includes a search bar, which allowed the user to filter for recipes whose name included the string they typed. After the user selects the story, if they click the create button, a new empty story is created, and the menu closes. This menu can be seen in figure 4.23.

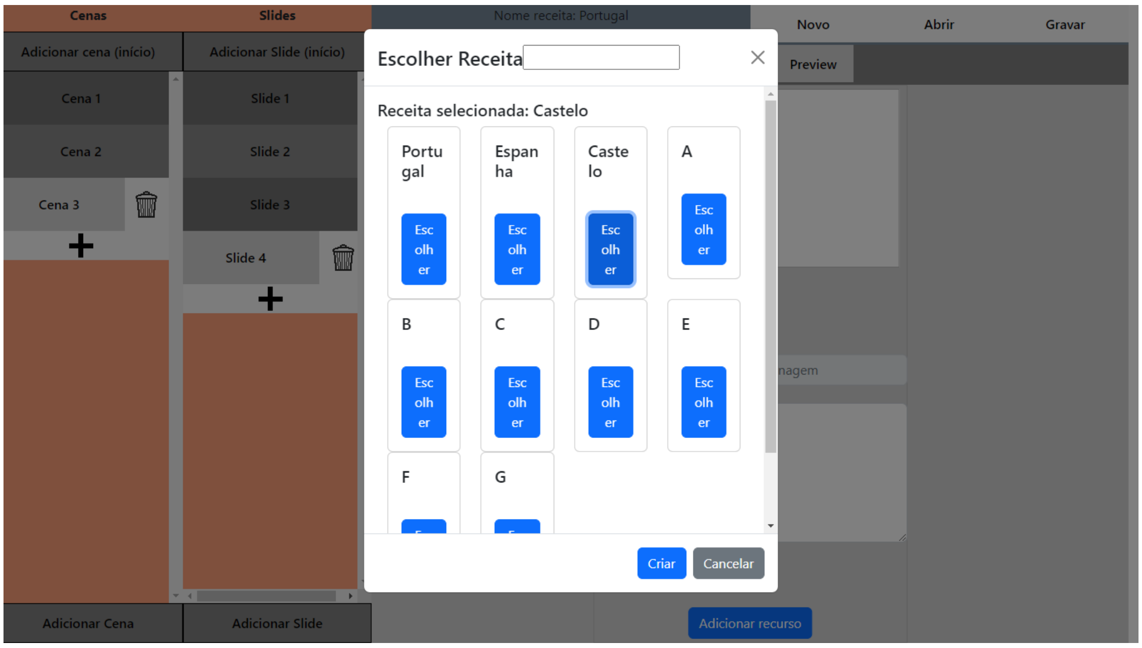


Figure 4.23: Second React story editor prototype: New story menu

4.8.0.4 Saving stories

The save feature now functions properly.

When the user clicks on the save button, a menu appears. If the user writes the name they want to save the story with, they can click the save button, which saves the story. Once the saving is completed, the user can close the save story menu. This menu can be seen in figure 4.24. Stories are converted from the format they are stored in the application to a different class instance. Afterwards, that instance is serialized to a JSON file string and then that string is sent to the server to be saved, along with the story's name and the recipe's name.

4.8.0.5 Loading stories

The user can load a story by clicking the load story button. Clicking the button opens a menu, which allows the user to select the recipe they want to use. This menu can be seen in figure 4.25.

After the user selects the recipe they want to use, the application displays a menu which allows the user to select the story they want to load. When this menu is opened, the application makes a request to the server to get the name and last modified date of the list of stories which use that recipe. Once the list of stories is loaded, the application displays that list. Each element of the list contains the name of the story. The stories are sorted alphabetically in ascending order (A to Z). The user can use the search bar at the top to filter the stories by name. Once the user has selected a story, the user can click the open button to load that story. This menu can be seen in figure 4.26.

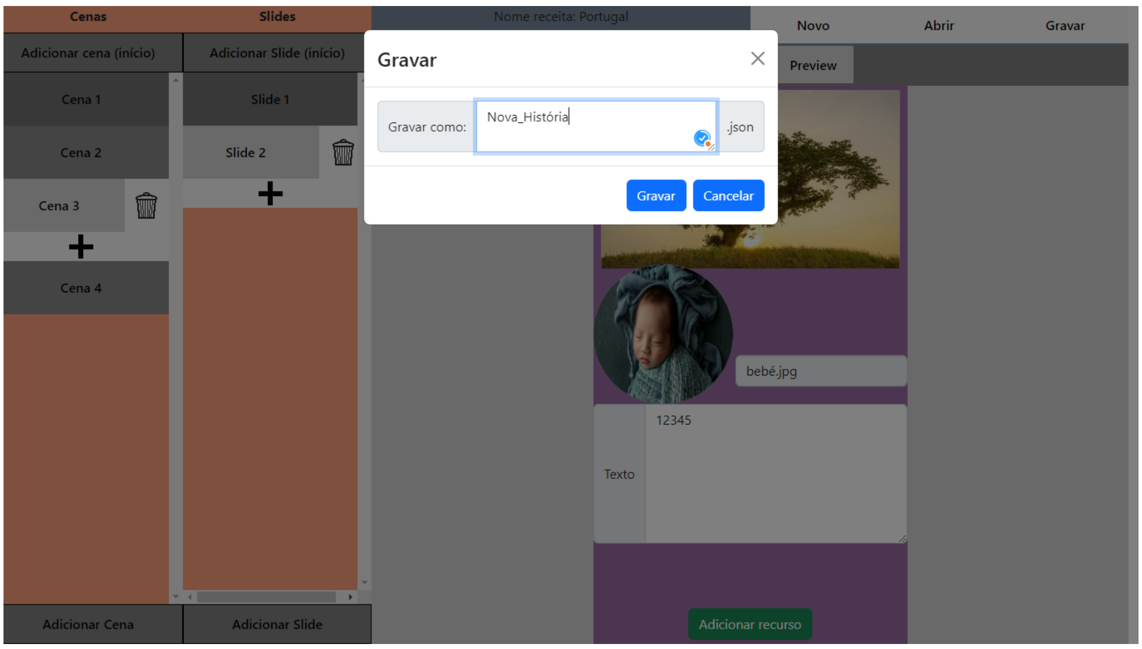


Figure 4.24: Second React story editor prototype: Save story menu

4.9 Story Player (Final iteration) - Conceptualization and Development

4.9.0.1 Exclusions from previous version

When compared to the previous version of the story player, certain features were removed. These previous additions were considered not relevant enough, too time-consuming to implement in the story editor, not fitting most possible story representations, or because of the redesign to the story structure.

Triggers were removed for two reasons. Firstly, they would not function in story players which would not have controllable characters. Secondly, the editor would have to undergo major design and implementation changes to accommodate for them.

Items were removed since their previous use (activating trigger) no longer existed.

The volume field remained unused from dialogue actions, since it was not included in the editor.

Because of the xAPI structure, in the story JSON file, the story should merely be a list of actions. This means that there should be no fields outside of actions. The fields in the previous version of the story editor that did not follow that standard were either removed or moved inside some action.

4.9.0.2 Main Menu

The story player begins at the main menu. This menu includes a dropdown with a list of stories with a refresh button, two buttons, one to play the story, and another one to enter the scene without

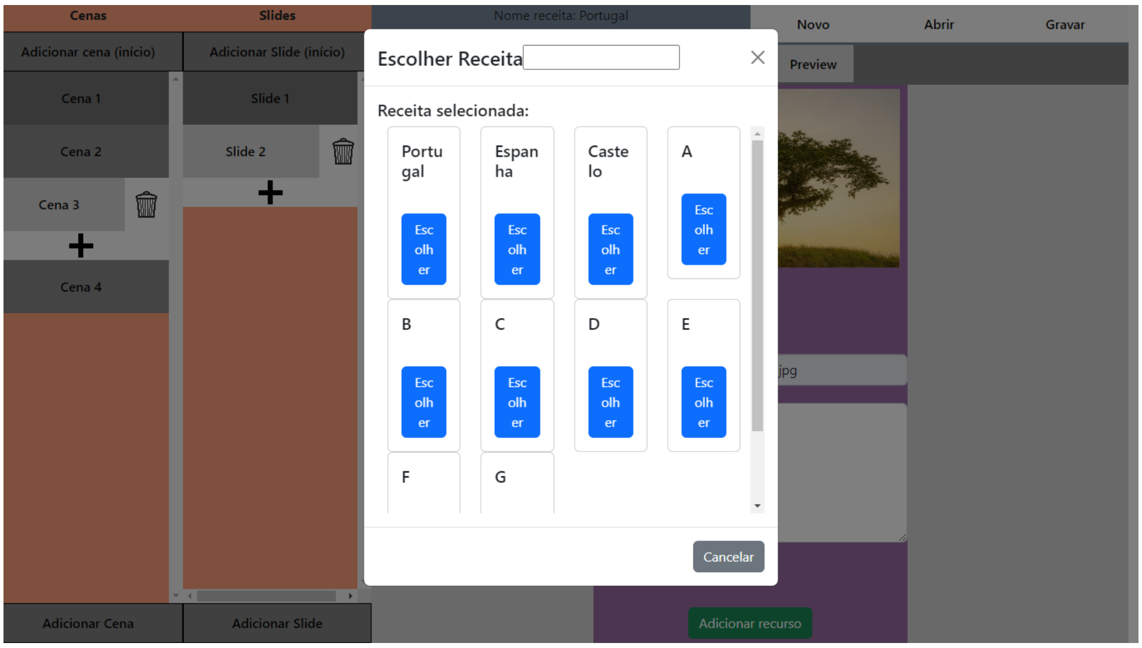


Figure 4.25: Second React story editor prototype: Load story menu (Recipe Selection)

playing a story, to allow for free exploration. There is a checkbox, which, when clicked, toggles whether characters move from one place to another. There is an additional button which shows the controls. The main menu can be seen in figure 4.27.

4.9.0.3 Controls

R - Return to the main menu. Note: Returning to the main menu unloads the story.

Left mouse button - Skip action. The current action immediately finishes, and the next one is played. Only works while playing the story.

1, 2 - Change camera presets to a certain camera. Only works in dialogue actions.

W, A, S, D - Move the character. Does not work while playing the story.

Moving the mouse in a direction moves the camera. Does not work while playing the story.

The controls menu can be seen in figure 4.28.

4.9.0.4 Stories

Stories played are loaded when the play story button is clicked. Additionally, other necessary files, such as images, are loaded at the same time. After all the files are loaded, the starting story file is converted into a story object, which is represented as a sequence of actions.

Unlike the previous version, instead of each story having a start ID field that determined the action that played first, that action is now the first action in the file. The previous method did not respect the xAPI, as the story file would no longer be only composed of a list of actions.

The following format did not respect the rules set by the xAPI "[...]" is replacing the fields inside the action's context that are not relevant to this example:

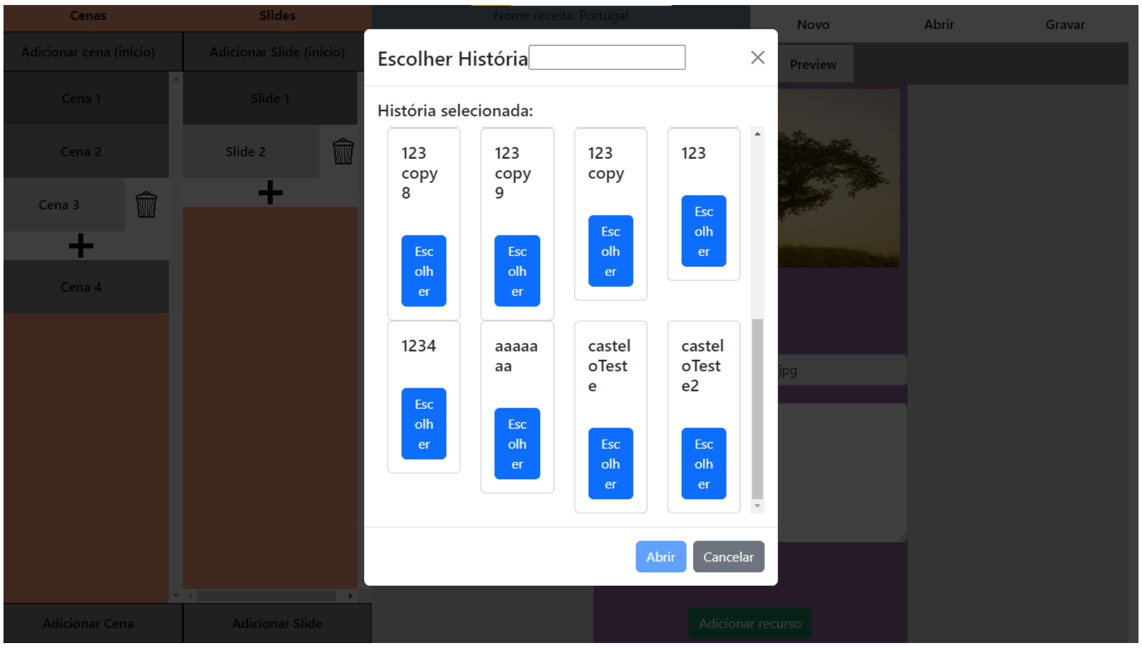


Figure 4.26: Second React story editor prototype: Load Story Menu (Story Selection)

```
{
  "startId" : 1,
  "historia": [
    {
      "verb" : "irpara",
      "actor" : "actor1",
      "place" : "place1",
      "context": {
          id: 1,
          [...]
      }
    }
  ]
}
```

The following format respected the rules set by the xAPI. "[...]" is replacing the fields inside the action's context that are not relevant to this example:

```
{
  "historia": [
    {
      "verb" : "irpara",
      "actor" : "actor1",
      "place" : "place1",
      "context": {
          id: 0,
```

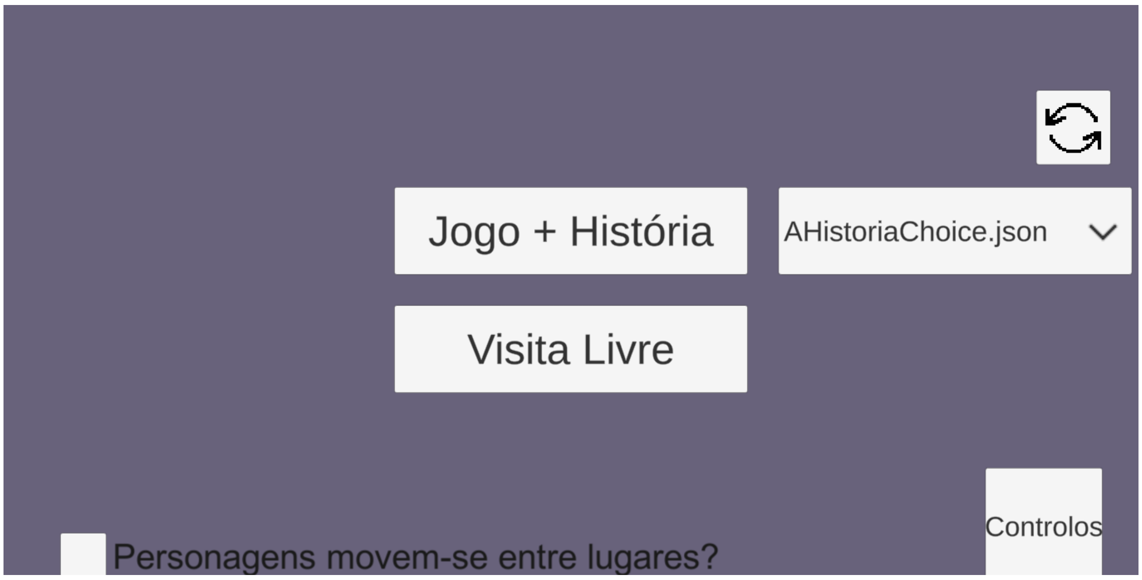


Figure 4.27: Story Player (Final Iteration): Main Menu

```
          [...]
      }
    }
  ]
}
```

An instance of a Cutscene class represents a story. The cutscene class includes a whole number representing the ID of the first action played, a list of actions and a boolean representing whether the story is playing or not. These fields are set when the story is loaded.

The story player now supports recipes, as intended. A story's recipe determines the lists of characters and places that can be used, and the environment that they are played in. For example, if a story's recipe recipe was "Forest", then the characters and places had to be from the recipe with that name and the environment in the 3D editor where that story played would likely be a forest. Story JSON files now include the recipe they exist in, in the first action of the story.

The start id field from the Story JSON files can now be ommited. If it is ommited, then the default value is set to 0. Stories exported by the story editor naturally begin at the action with ID 0, so there was no need to include it.

4.9.0.5 Actions

The new version of the story player had two actions:

- "Diálogo" (Speech)
- "Escolha" (Choice)

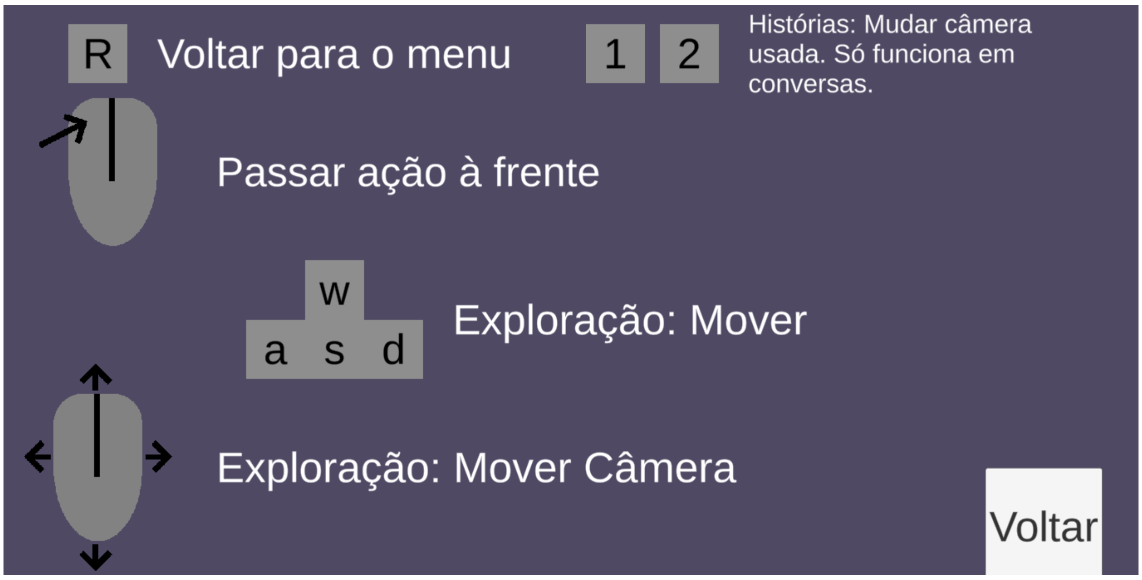


Figure 4.28: Story Player (Final Iteration): Controls

The stay and move actions were removed. These actions did not fit many types of story players (eg: a comic book style 2D representation). Characters can still appear, through the use of other actions.

The dialogue action places one or two characters in a certain place, looking at each other. A text box is displayed containing the text in the text field, as well as an image (retrieved from a URL defined in the image URL field). Dialogue actions can only be ended by skipping them, through pressing the left mouse button. An example of a dialogue action can be seen in figure 4.29.

Choice actions present the user with a question, in a text box, which is a string included in the choice text field, with one or more possible response strings, which are included in the choices field, which is an array of objects each containing a string as the question text and an integer as the ID of the action played after the user picks that choice. An example of a choice action can be seen in figure 4.30.

4.9.0.6 Places

Inside the Unity project, places are denoted by a game object. That game object's transform's position indicates the place's position in the application's world. For each of those points, an icon Gizmo is drawn. These are only shown in the editor's scene view, and, as such, are not shown in the final product. These were used to more easily place and locate the points.

If two different characters are to be placed in the same position at the same time, then a nearby position is found. The positions used are stored, together with their names and the characters that they are using. These are stored as 2 dictionaries, to make searching for the character or the position not scale with time. operation. When an action is about to begin, for each character, a function is called to provide a nearby available place. Firstly, this function removes the character from the dictionaries of occupied places. Afterwards, if the place intended for the character is

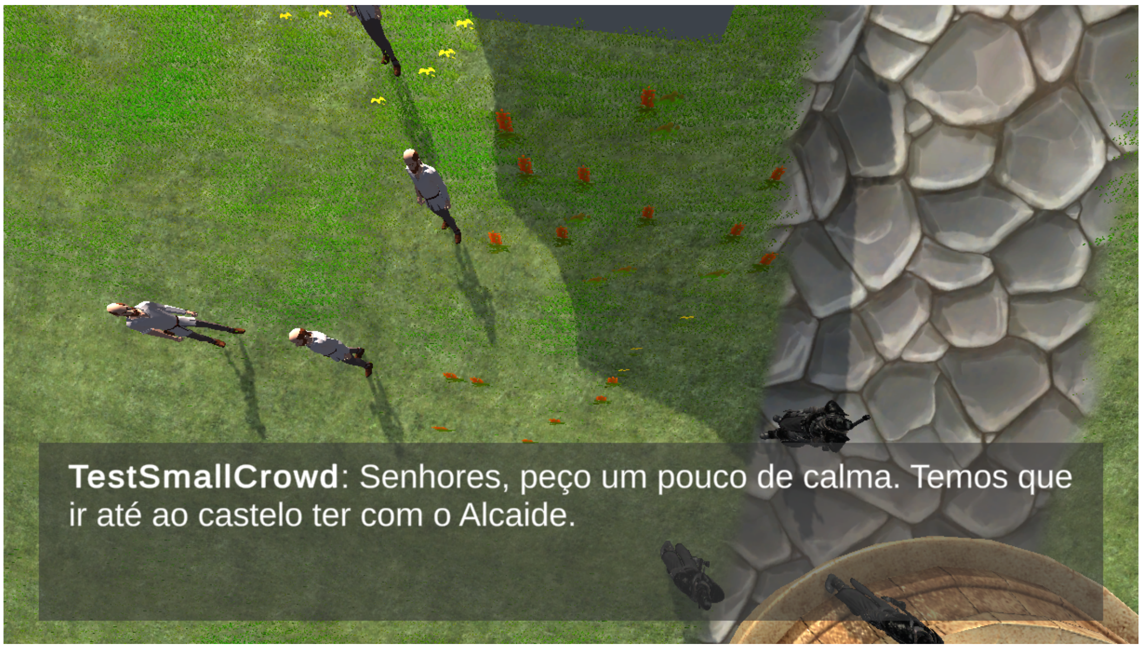

TestSmallCrowd: Senhores, peço um pouco de calma. Temos que ir até ao castelo ter com o Alcaide.

Figure 4.29: Story Player (Final Iteration): Dialogue Action

free, it is chosen and added to the dictionaries. If the place is occupied, it tries to find nearby places. This works by searching for nearby positions in multiple circles with different distances. For each place, it checks if it's not too close to another place. Finally, if the character is not part of a crowd, it does a raycast to check if the line of sight from the original place to the new place is uninterrupted. Having a raycast for each point caused lag with very large crowds. For crowds, the radius of the circles is smaller than for individual characters, so that the characters in a crowd can remain closer.

4.9.0.7 Characters

Each character is associated with a name. Additionally, each character is associated with a Unity prefab, whose name is the same as the character's name. That prefab contains, in its root game object, multiple components, which are required to play the actions. Each prefab's root game object contains two children. The first child contains a transform positioned in the face of the character. This object is used by the camera scripts, so that the camera can attempt to keep that point onscreen, when attempting to keep that character onscreen.

Every character that belongs to a story was loaded when the story begins playing. Note that, because of the way that characters work, two characters cannot exist with the same name. When an action was called with the same character name as a previous action, both actions would use the same object. For example, if the story began with a Stay action with an actor named "Carina" at the place named "Park" and, later on, there was a dialogue type action with an actor named "Carina" at a place named "Coffee shop", the character named "Carina" would be translated to the

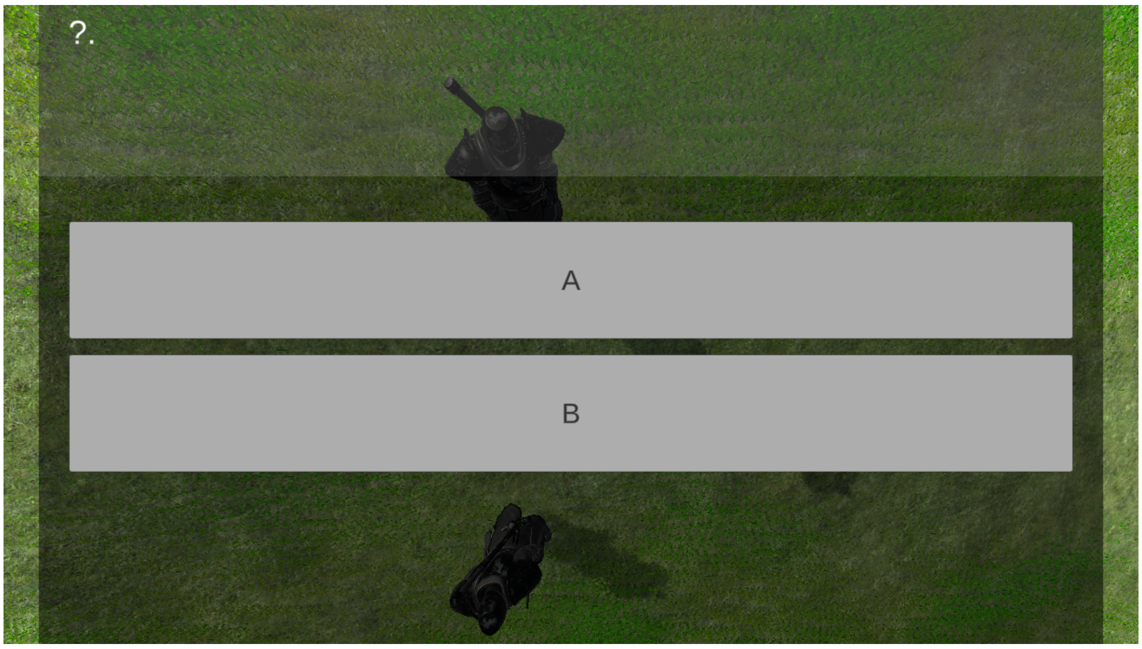


Figure 4.30: Story Player (Final Iteration): Choice Action

place named "Park" for the first action, and, when the second action plays, that character would be translated to the place named "Coffee Shop".

Characters had a NavMeshAgent attached to them, which allowed them to find and go through the nearest path from their current position to any other place within the NavMesh's area. This was used for movement actions. A character did not have to be a single person. A crowd of people could be considered as a single character as well. In that case, each of the characters have a NavMeshAgent attached to them, and, in movement actions, all of them moved towards the target destination. There was a problem where characters in a crowd could be placed in non-connected parts of the navmesh. Those characters did not go towards the destinations.

4.9.0.8 Points of interest

Points of interest are points in the game world, indicated by Game Objects. A point of interest contains three strings: a name (the name of its Game Object), a title and a description. Whenever the player is close enough to a point of interest, its title and description will be shown on the screen.

The points of interest, their locations, names, and descriptions cannot be changed by the story creators. An example of the points of interest system can be seen in figure 4.31.

4.9.0.9 Doors

Doors automatically open when a character gets close enough to them, both during stories and in the exploration mode. Whenever every character that was in the door's area leaves that area, the

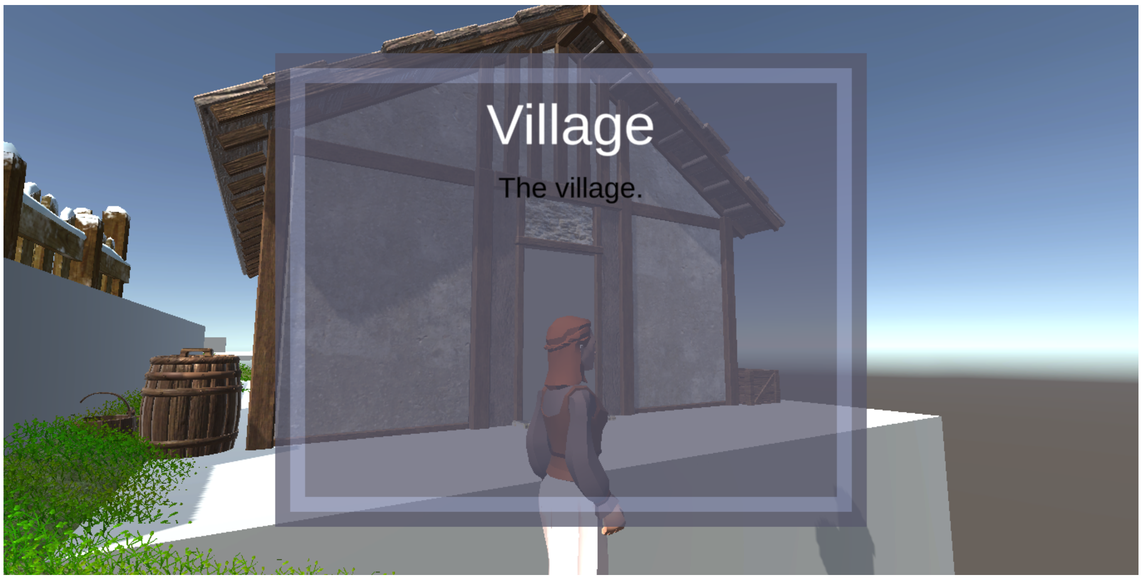


Figure 4.31: Story Player (Final Iteration): Point Of Interest

door closes. Doors open and close by rotating ninety degrees. An example of these doors can be seen in figure 4.32.

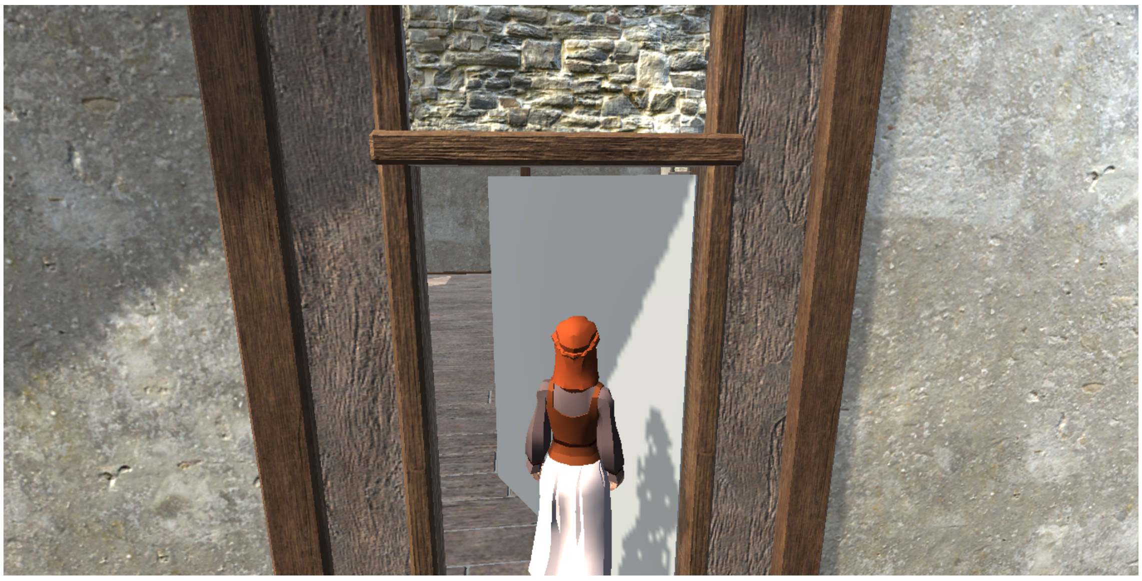


Figure 4.32: Story Player (Final Iteration): Door

4.9.0.10 Camera

As stated previously, this project uses the Cinemachine Unity package. The Cinemachine package was used to automatically control the camera during stories and during exploration. At any time during the story, depending on the action that was currently playing, the camera would be set to a certain preset. In the case of dialogue actions, there are two different presets which could be set, depending on the option selected by the player. During these actions, the player could set the camera used by pressing 1 or 2 on the keyboard's numpad.

During non-dialogue actions, the camera attempted to follow the corresponding action's player.

During dialogue actions, the camera attempted to aim at the one or two characters in the action, and to keep both characters in the camera's view.

If the dialogue action involved two characters, the camera's position was manually calculated to be a certain distance away from the centre point between the characters. If either or both of the characters in the action were crowds instead of singular characters, the camera attempted to include all of those characters. The dialogue camera includeed two different presets. The user could select them by pressing the 1 or 2 keys in the keyboard. If the first preset was chosen, then the camera would have a field of view (FoV) ranging from 30 to 90, and the cinemachine group composer would only try to adjust itself by zooming in or out. If the second preset was chosen, then the same minimum and maximum fields of view would be enforced, but the camera would also attempt to dolly to obtain the best angle, adjusting its distance by a maximum or 30 units in or out, and keeping its distance between 5 and 60.

During the exploration, the camera follows the character that the player is controlling. This is achieved by adding a Cinemachine3rdPersonFollow component to the Cinemachine's body property and setting its follow property to the player character's camera pivot. That camera pivot rotates when the player moves the mouse in any direction, which then rotates the camera.

Chapter 5

Conclusions

This chapter begins by outlining the work done. Then, it answers the questions posed at the beginning of the review process.

5.1 Summary

This project was created over the course of a year as part of the FronTowns project (NOVA FCSH, n.d), which intends to, among other things, recreate the historical evolution of two villages, Castelo de Vide and Cáceres. This project intends to assist in that by allowing users to create and play stories in those historical scenarios. From the systematic review process, The realization of the systematic review process generated some goals for this project. Combined with meetings with team members and the thesis advisors, some requirements were obtained for the project. Using the (iterative) Design Science Research (DSR) methodology, (Peffers et al., 2007), the author developed this application in iterations of creating requirements, proposing solutions, developing on those solutions, evaluating that prototype and finding problems. The story editor was made in React and creates a JSON file (which follows the rules of the xAPI) composed of a sequence of actions from the story created by the user. The 3D story player was made in Unity and read the JSON file that the user wanted to use, playing the story, using the characters and places included in the Unity application, specified in the story by their names. Stories used a recipe, which determined the 3D environment of the story in the 3D story player and the list of characters and places available. Story actions could be of two different types: Dialogue and (multiple) Choice. The former represented actions without choices, while the ladder represented actions with choices. Actions could have a group of fields added to them, such as text or 3D models.

5.2 Answers to questions posed

5.2.1 What degree of freedom was given to the users?

The applications were structured in such a way that the creators had as much freedom as possible, considering both the time frame of creating these applications and the lack of computer skill of the target end users. Users were able to create, save, load and modify stories in the story editor, while they were able to play the stories in the story player. In the story player, they were allowed to determine if characters move between places, or if they just teleported, through the use of a checkbox. In terms of each individual action, the user could change the actor and the place, as well as the action's text, the background image (for the 2D preview), along with one of the additional fields. Note that the user could not create a new actor or place, since, for a 3D application, that would mean including the 3D model, along with a rig compatible with the animations used. This was considered too complex for the target end users, who do not necessarily have computer skills. For the place, as mentioned in 5.3 and included in the first story editor screen prototype, the user was intended to have the ability to create a new place by clicking on the desired location in an image representing the top down view of the recipe. However, in the end, this was not included due to time constraints.

5.2.2 How can the application define platform-independent and (preferably) customizable story elements?

Positions and characters were defined by their names. Additionally, they include images, which were used for the 2D preview. In the 3D platform, those names were associated with positions for places, and with game objects for characters.

5.3 Future work

This section covers some points that could or should be done in the future.

First and foremost, since the iteration process finished in the development phase, these prototypes for the story player and editor should be shown to and evaluated by the team, so that feedback can be obtained on them.

More bug testing should be done, so that this could be used as a better final product.

For both the story editor and player, the necessary recipes should be added, with their corresponding list of characters, places and their names and images.

Characters could be separated into characters and objects, where the former can move while the ladder can't. This could help reduce confusion, since people without computer/programming experience may not associate objects as being "characters".

A volume field could be added.

For the story player, the following list of improvements could be done:

- For the story player, there are a few improvements to be made.
- Displaying 3D models and videos in the story editor (through Sketchfab and YouTube links/embeds). Including 3D models and videos is a functionality already included in the story editor and, as such, it should be included in the 3D story player. Unfortunately, it is not possible to add YouTube or Sketchfab embeds to a Unity application in WebGL. Having

the user upload the videos directly to the story editor is an option, however, it occupies a lot of server space and requires a good internet connection to load. Additionally, these applications' target end users would likely not know how to download these, as they are not guarenteed to have computer experience. Despite not being able to add embeds to the Unity application, the embeds could be added to the page running the Unity application. Unity exported to WebGl allows calling Javascript functions with arguments from the C# code. As such, assuming that these function calls can change the page's DOM, embeds could be added onto the web page and placed in front of the Unity application. Whenever the application needed to display a 3D model or youtube video, a function call would be made to insert a link into those embeds. Whenever those embeds needed to be hidden, another function call could be done for that.

- Crowds should be able to be placed in some formation. This would be particularly helpful for stories focused on progression of one or more armies over a certain war.
- The camera could be allowed to move in 360º in every axis in the middle of stories, to assist the user with the visualization.
- Images and sounds from a story should not be loaded all at the beginning. They should instead be loaded during the course of the story.
- When a story ends, the user could be asked which character they would like to control from a list of characters. This list could either come from the recipe, from the story, or include every character from any recipe. Alternatively, instead of having a character, the user could move the camera, without controlling any character.
- An undo action button could be added to the stories, which would return the story to the previous action.
- For free exploration mode, the user should be able to select which recipe they want to explore.
- Characters who are talking in an action could have their images displayed.
- Story creators should be able to add, modify, and remove points of interest.

For the story editor, the following list of improvements could be done:

- Most importantly, a second character should be able to be added to scenes, so that dialogue actions work properly.
- Being able to reorder scenes, or slides from the same scene.
- Being able to reorder quiz answers in the editor.

- Being able to add "pseudonyms" to existing characters in the editor, for specific actions. The same character with two different pseudonyms would be considered as two different characters, so that the same model could be used for multiple characters.
- Being able to add new places to be used in the story player. This was considered lower priority due to two reasons. Firstly, this would likely require a different file, or alternatively violating the guidelines of the xAPI by placing the places' names and coordinates outside of the list of actions.
- Adding error messages to the story editor when a story had some inconsistency which would not allow it to run properly in one or more story players. These error messages would be displayed either while the story was being edited in a non-intrusive manner (eg: a bar at the bottom), or as a pop-up while saving, which would prevent the story from saving.
- The ability to translate a story to a different recipe, allowing the user to manually change the actors, places, and actions as necessary. This feature was included in the story editor this one was based on.
- Being able to see the contents of the recipe without having to create a story, and look through the list of characters and places could be helpful.
- Images should be resized to the size they are needed as, to save internet data while playing the story.
- The server should images and files that are no longer being used in any story.

References

[1] Frontowns website. https://frontowns.fcsh.unl.pt/.

[2] HTML5 Game Development. https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/HTML5/.

[3] Daniel F. Abawi, Silvan Reinhold, and Ralf Dörner. A Toolkit for Authoring Non-linear Storytelling Environments Using Mixed Reality. In Stefan Göbel, Ulrike Spierling, Anja Hoffmann, Ido Iurgel, Oliver Schneider, Johanna Dechau, and Axel Feix, editors, *Technologies for Interactive Digital Storytelling and Entertainment*, Lecture Notes in Computer Science, pages 113–118, Berlin, Heidelberg, 2004. Springer.

[4] Lizzie Abderrahim and Faith Mehmet Cigerci. A Systematic Review Of Research Undertaken Into The Use Of Digital Storytelling As A Pedagogical Tool In The Language Classroom. pages 113–124. 2018.

[5] Fernando Cassola, Daniel Mendes, Manuel Pinto, Leonel Morgado, Sara Costa, Luís Anjos, David Marques, Filipe Rosa, Ana Maia, Helga Tavares, António Coelho, and Hugo Paredes. Design and Evaluation of a Choreography-Based Virtual Reality Authoring Tool for Experiential Learning in Industrial Training. *IEEE Transactions on Learning Technologies*, 15(5):526–539, October 2022. Conference Name: IEEE Transactions on Learning Technologies.

[6] Chia Yi Quah and Kher Hui Ng. A Systematic Literature Review on Digital Storytelling Authoring Tool in Education: January 2010 to January 2020. *International Journal of Human–Computer Interaction*, 38(9):851–867, May 2022.

[7] A. Tanju Erdem, Bora Utku, Tolga Abaci, and Çiğdem Eroğlu Erdem. Advanced authoring tools for game-based training. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCSC '09, pages 95–102, Vista, CA, July 2009. Society for Modeling & Simulation International.

[8] Daniel Green, Charlie Hargood, and Fred Charles. Define "Authoring Tool": A Survey of Interactive Narrative Authoring Tools. December 2018.

[9] Stefan Göbel, Luca Salvatore, and Robert Konrad. StoryTec: A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-Linear Stories. In *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*, pages 103–110, November 2008.

[10] Alan Hevner. A Three Cycle View of Design Science Research, January 2007.

[11] Marty Kauhanen, Chris Eaket, and Robert Biddle. Patterns for Story Authoring Tools. January 2007. Pages: 36.

[12] S.-K Kim and SeongKi Kim. Survey on the narrative structure of the interactive storytelling authoring system. *International Journal of Applied Engineering Research*, 11:9236–9239, January 2016.

[13] Demetrius Lacet, Filipe Penicheiro, Leonel Morgado, and António Coelho. Preserving story choreographies across multiple platforms: An approach to platform-independent reuse of characters' behaviors for games, simulations, animations and interactive videos. In *Proceedings of the 9th International Conference on Digital and Interactive Arts*, pages 1–4, Braga Portugal, October 2019. ACM.

[14] Maxence Laurent, Sandra Monnier, Audrey Huguenin, Pierre-Benjamin Monaco, and Dominique Jaccard. Design Principles for Serious Games Authoring Tools. *International Journal of Serious Games*, 9(4):63–87, November 2022. Number: 4.

[15] Artur Lugmayr, Erkki Sutinen, Jarkko Suhonen, Carolina Islas Sedano, Helmut Hlavacs, and Calkin Suero Montero. Serious storytelling – a first definition and review. *Multimedia Tools and Applications*, 76(14):15707–15733, July 2017.

[16] Jonathan D. Moallem and William L. Raffe. A Review of Agency Architectures in Interactive Drama Systems. In *2020 IEEE Conference on Games (CoG)*, pages 305–311, August 2020. ISSN: 2325-4289.

[17] Leonel Morgado and Dennis Beck. Unifying Protocols for Conducting Systematic Scoping Reviews with Application to Immersive Learning Research. In *2020 6th International Conference of the Immersive Learning Research Network (iLRN)*, pages 155–162, June 2020.

[18] Nur Azima Alya Narawi, Hayati Abd Rahman, Nazrul Azha Mohamed Shaari, and Wan Ya Wan Hussin. Interactive story graph structure for digital storytelling / Nur Azima Alya Narawi ... [et al.]. *Malaysian Journal of Computing (MJoC)*, 5:619–634, October 2020.

[19] Valentina Nisi. The changing panorama of interactive storytelling: a review from locative to transmedia. *DOC On-line*, 0(2017SI), January 2018. Number: 2017SI.

[20] Elena Not and Daniela Petrelli. Empowering cultural heritage professionals with tools for authoring and deploying personalised visitor experiences. *User Modeling and User-Adapted Interaction*, 29(1):67–120, March 2019.

[21] Mark Riedl, Boyang Li, Hua Ai, and Ashwin Ram. Robust and Authorable Multiplayer Storytelling Experiences. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 7(1):189–194, October 2011.

[22] Mark Riedl, Boyang Li, Hua Ai, and Ashwin Ram. Robust and Authorable Multiplayer Storytelling Experiences. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 7(1):189–194, October 2011.

[23] Mark Owen Riedl and Vadim Bulitko. Interactive Narrative: An Intelligent Systems Approach. *AI Magazine*, 34(1):67–67, 2013. Number: 1.

[24] Kevin Schenk, Adel Lari, Matthew Church, Eric Graves, Jason Duncan, Robin Miller, Neesha Desai, Richard Zhao, Duane Szafron, Mike Carbonaro, and Jonathan Schaeffer. ScriptEase II: Platform Independent Story Creation Using High-Level Patterns. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, November 2013.

[25] P. P. Ganapathi Subramaniam and S. S. Gokhale. Development of an Authoring System for Engineering Education for PC-DOS Applications. 1995.

[26] Unity Technologies. Unity - Manual: Unity User Manual 2021.3 (LTS). `https://docs.unity3d.com/Manual/index.html`.

[27] Unity Technologies. Unity - Manual: WebGL browser compatibility. `https://docs.unity3d.com/Manual/webgl-browsercompatibility.html`.

[28] Andrei Torres, Bill Kapralos, Celina Da Silva, Eva Peisachovich, and Adam Dubrowski. Moirai: A No-Code Virtual Serious Game Authoring Platform. *Virtual Worlds*, 1(2):147–171, December 2022. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[29] Marcus Trenton, Duane Szafron, Josh Friesen, and Curtis Onuczko. Quest Patterns for Story-Based Computer Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 6(1):204–209, October 2010. Number: 1.

[30] Mojtaba Vaismoradi, Jacqueline Jones, Hannele Turunen, and Sherrill Snelgrove. Theme development in qualitative content analysis and thematic analysis. *Journal of Nursing Education and Practice*, 6(5):p100, January 2016.

[31] Cara Winterbottom and Edwin Blake. Designing a VR interaction authoring tool using constructivist practices. In *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, AFRIGRAPH '04, pages 67–71, New York, NY, USA, November 2004. Association for Computing Machinery.

Appendix A

Terminology

A.1 Unity Terminology

- GameObject - A Game Object in Unity is an entity inside a Unity scene. Note that a GameObject has to either be inside a scene or a Prefab. GameObjects have a group of Components attached to them, and can have any number of child GameObjects. A scene can be seen as a tree (the data structure), where every node, except the root, is a GameObject.
- Prefab - The prefab system allows users to store GameObjects as reusable Assets.
- Component - A component is the base "class for all entities in Unity Scenes".
- Transform - A component which contains a GameObject's position, rotation and scale. Each GameObject has to contain exactly one Transform component.
- Gizmo - "Gizmos are used to give visual debugging or setup aids in the Scene view.". Since gizmos are only visible in the Scene view, they are not visible while playing the game.
- Collider - A Component which detects collisions with its area. If isTrigger is set to false, it acts on those collisions (Eg: Pushing the other object out); otherwise, it sends an OnTriggerEnter, OnTriggerExit, and OnTriggerStay, when a rigidbody enters or exits the trigger volume. For a collision to be detected, at least one of the colliders' GameObject has to have a Rigidbody component. Note: Colliders with isTrigger on will be called "trigger colliders" for simplicity.

A.2 C# Terminology

- Abstract class - An abstract class cannot be instantiated. In other words, an object of the type of that class cannot be created. An abstract class often has subclasses, which can access the variables and functions of the parent abstract class (as long as they aren't private)
- Subclass - For example, there can be an Animal class, which has three other classes, Dog, Cat, Zebra as subclasses.

A.3 React Terminology

- Lazy loading - Content is only loaded whenever it's necessary. For example, a lazy loaded image is only loaded whenever it's about to be shown.

Appendix B

Second version of the react story editor: Screenshots

B.1 Main screen

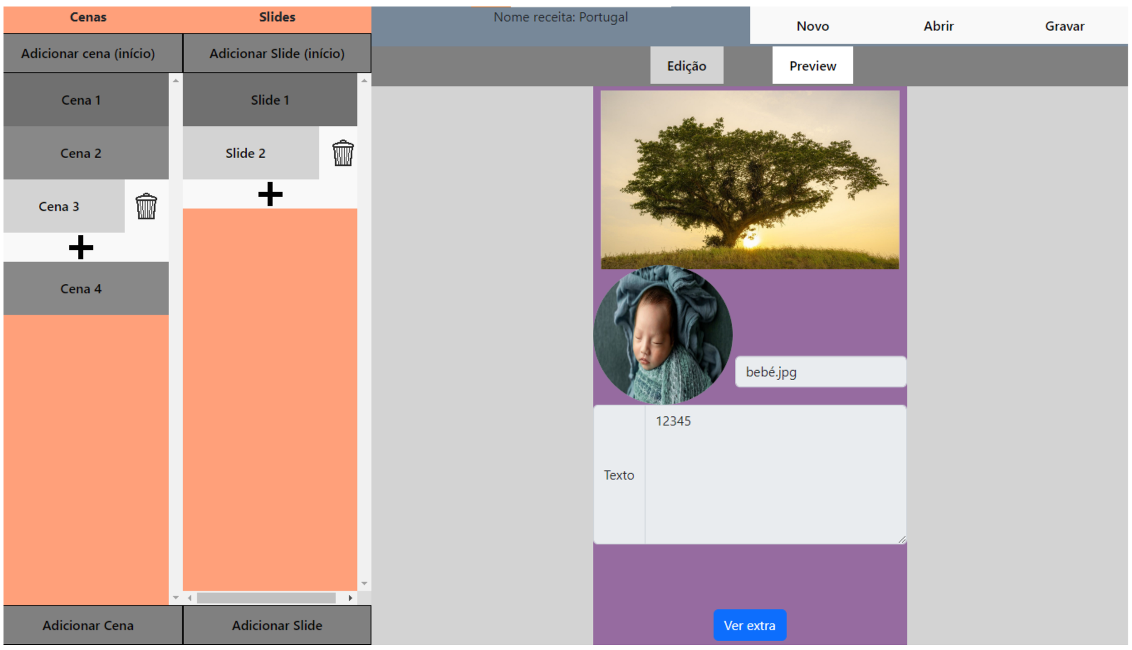


Figure B.1: Second React story editor prototype: Story Preview Menu

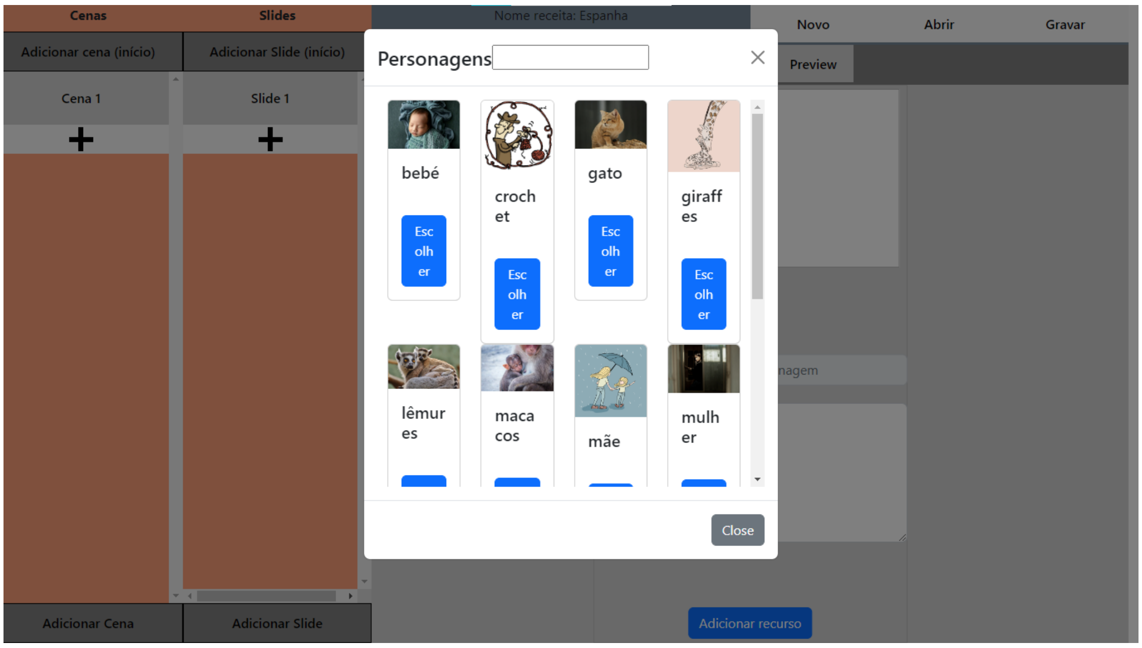


Figure B.2: Second React story editor prototype: Select Character Menu

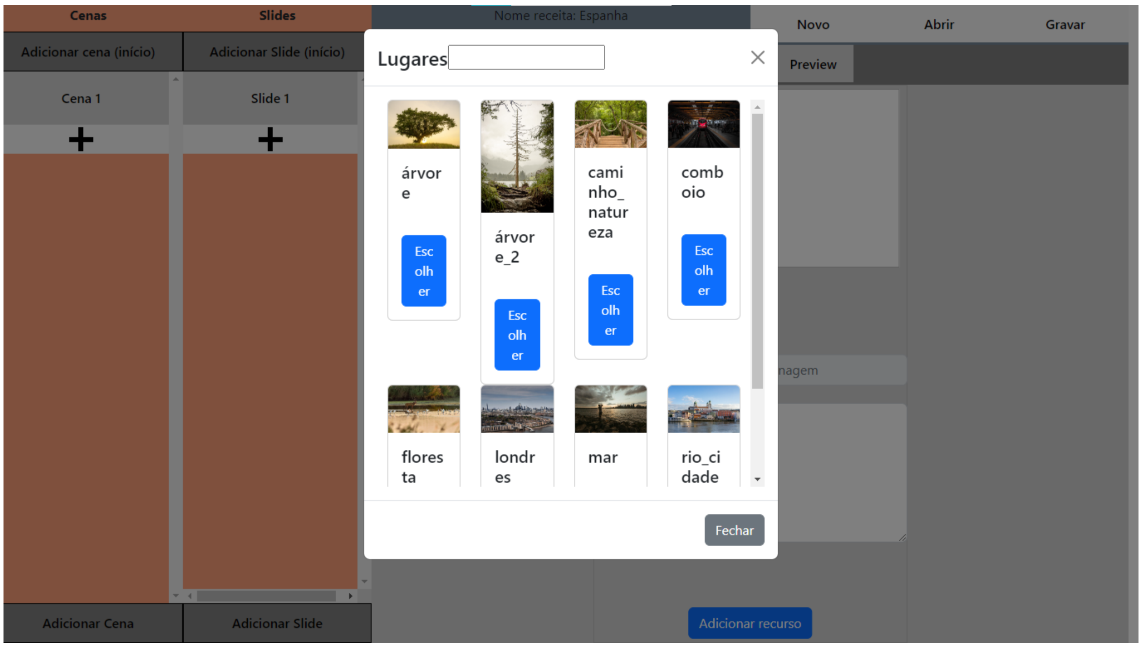


Figure B.3: Second React story editor prototype: Select Place Menu

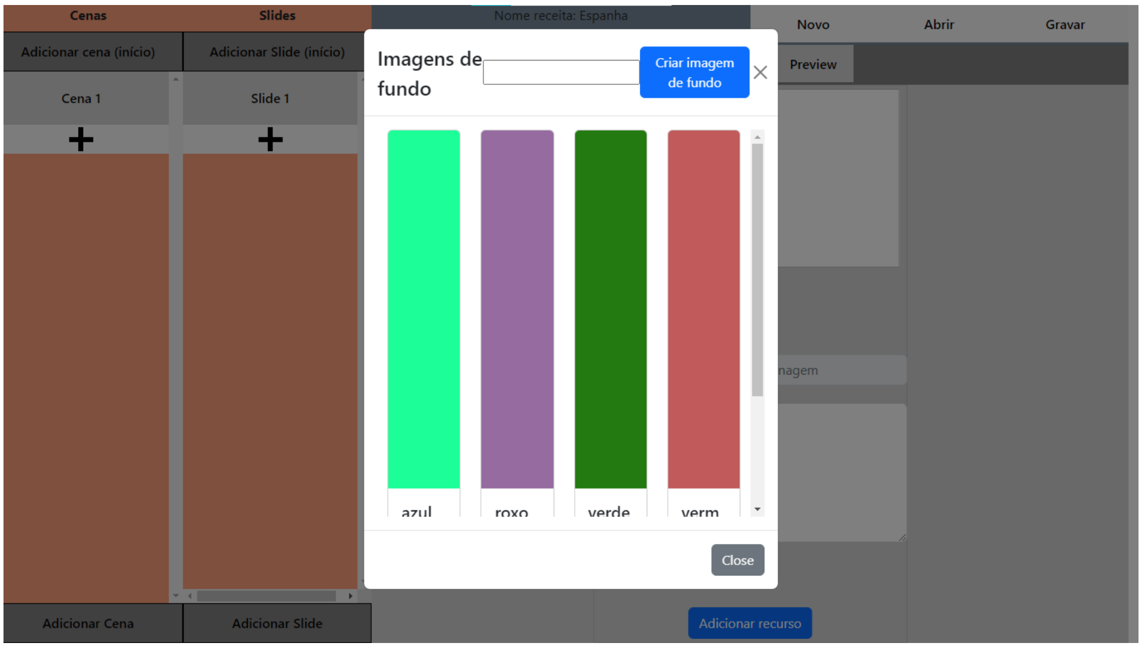


Figure B.4: Second React story editor prototype: Select Background Image Menu

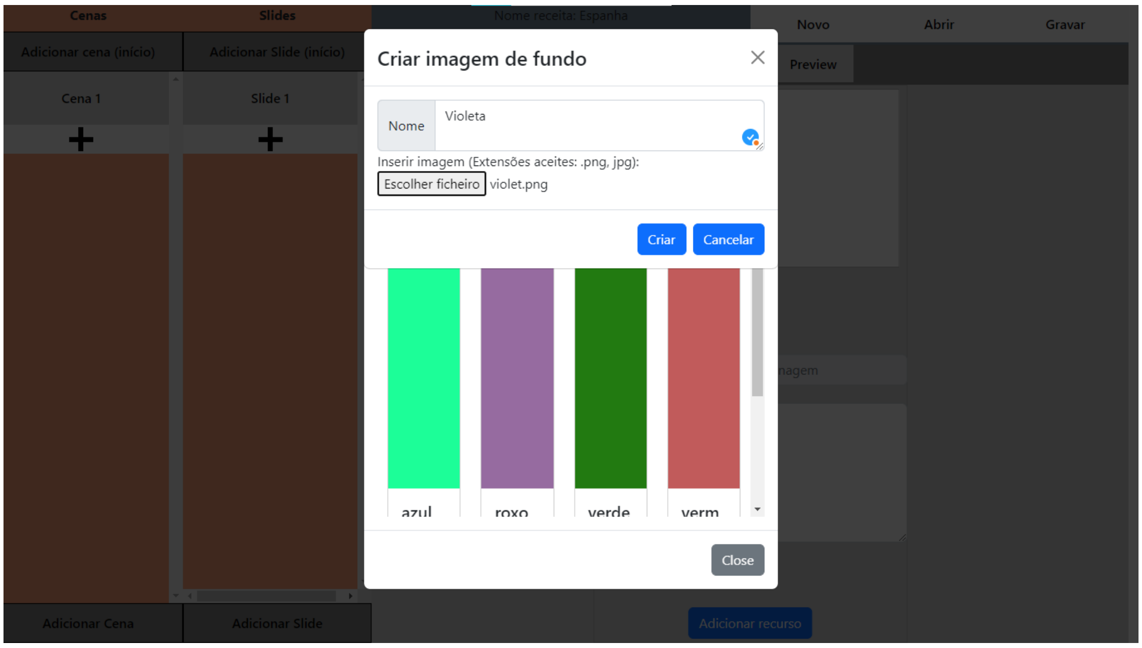


Figure B.5: Second React story editor prototype: Create Background Image Menu

B.2 Add resource menus

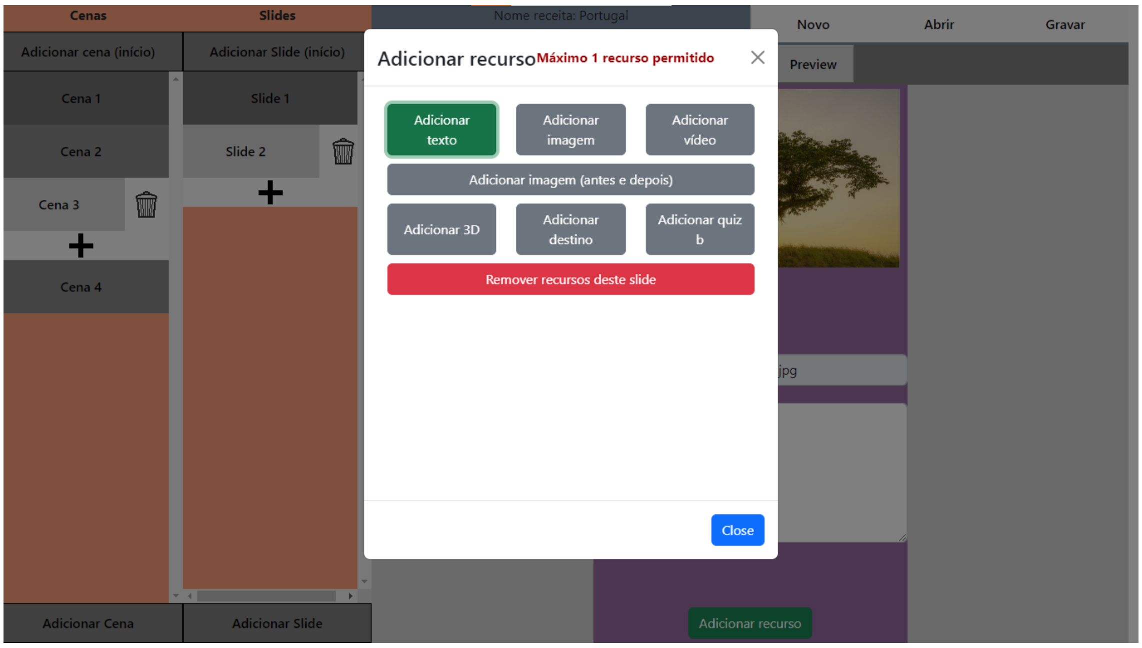


Figure B.6: Second React story editor prototype: Add Resource Menu

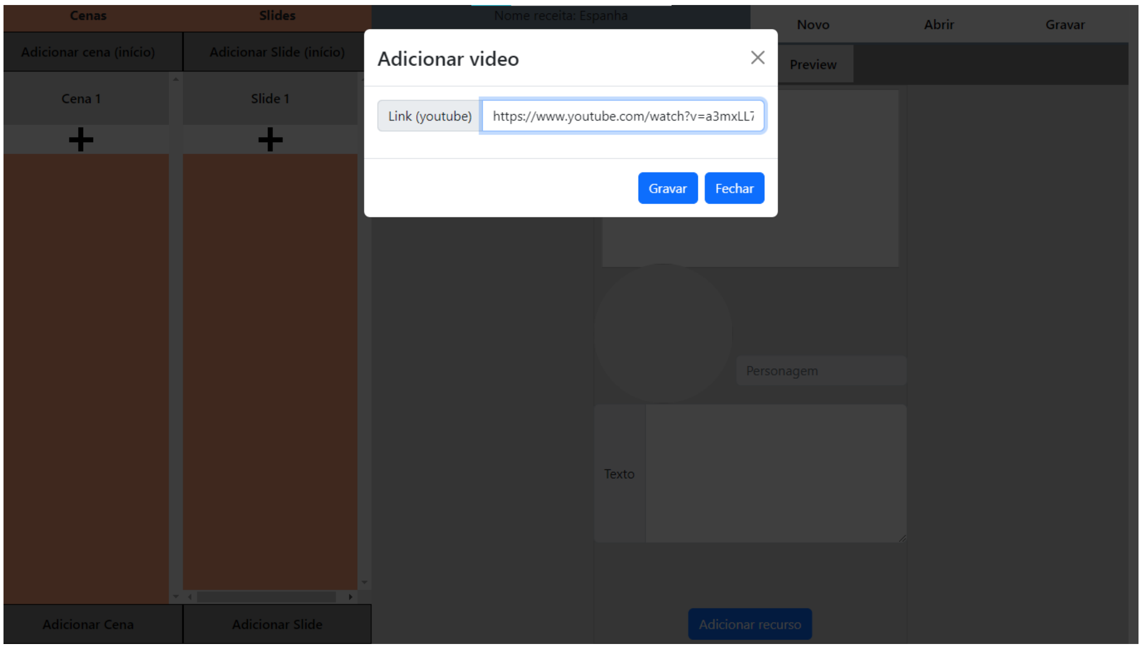
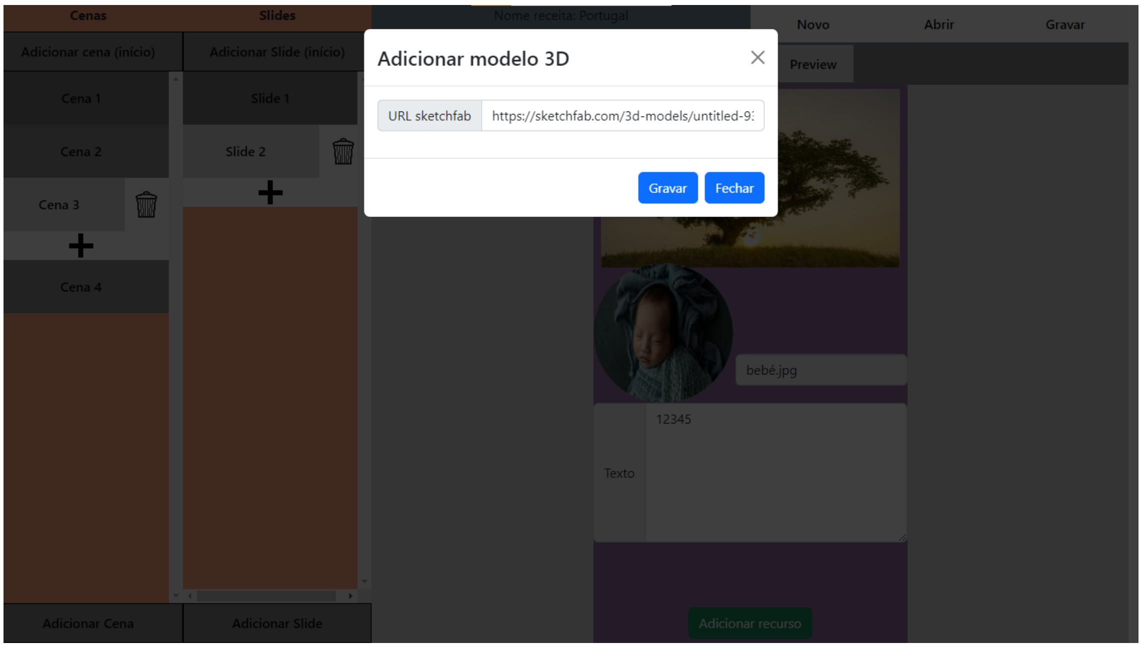


Figure B.7: Second React story editor prototype: Add Video Menu

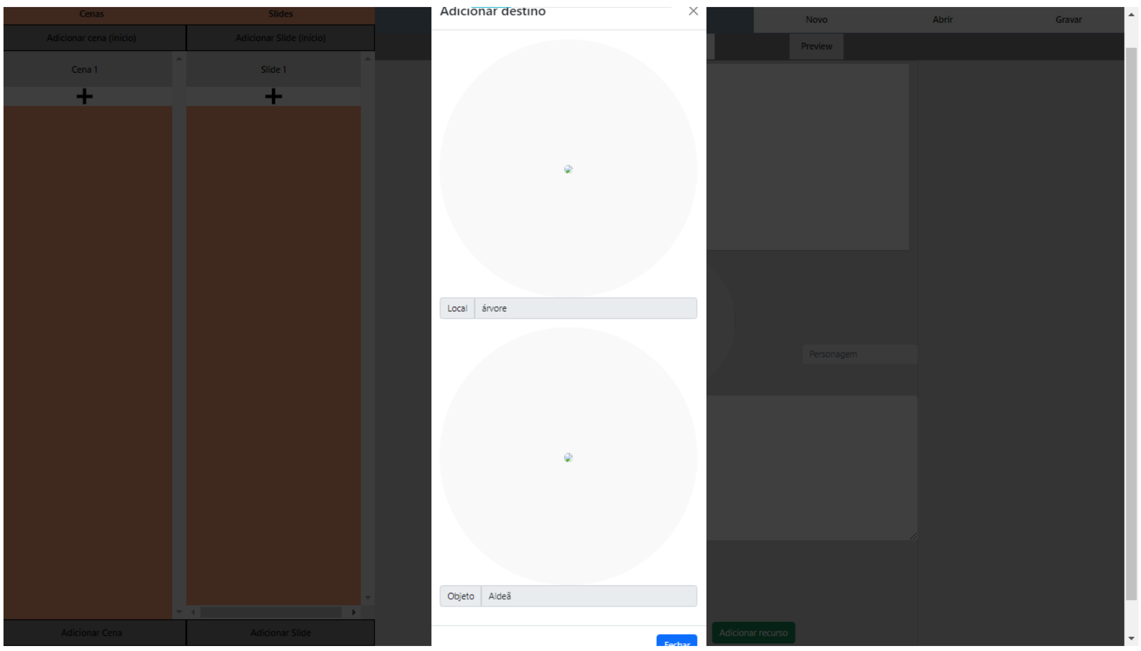


Adicionar modelo 3D

URL sketchfab https://sketchfab.com/3d-models/untitled-9

Gravar Fechar

Figure B.8: Second React story editor prototype: Add 3D Model Menu



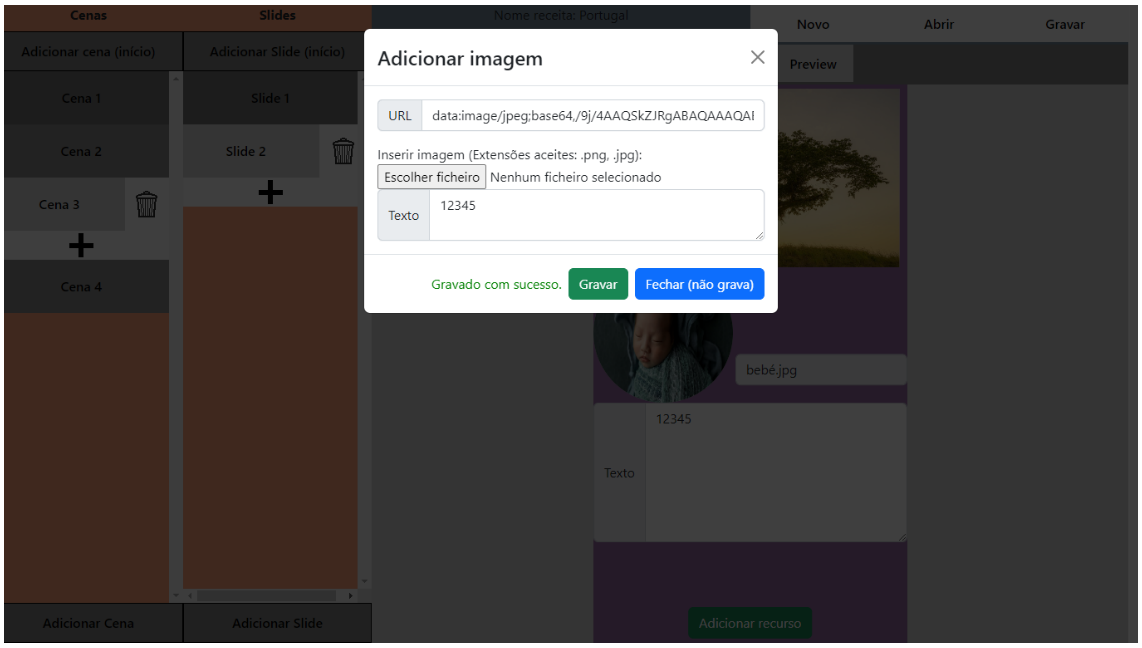
Adicionar destino

Local árvore

Objeto Aldeã

Fechar

Figure B.9: Second React story editor prototype: Add Destination menu



Adicionar imagem

URL data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQAI

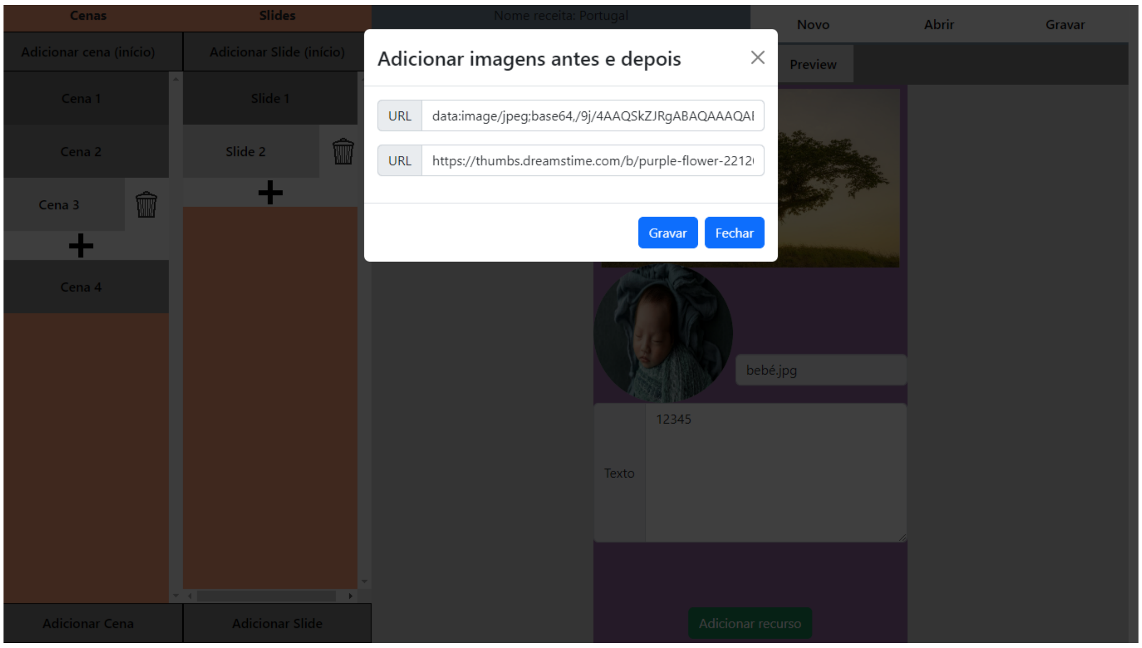
Inserir imagem (Extensões aceites: .png, .jpg):

Escolher ficheiro Nenhum ficheiro selecionado

Texto 12345

Gravado com sucesso. Gravar Fechar (não grava)

Figure B.10: Second React story editor prototype: Add Image Menu



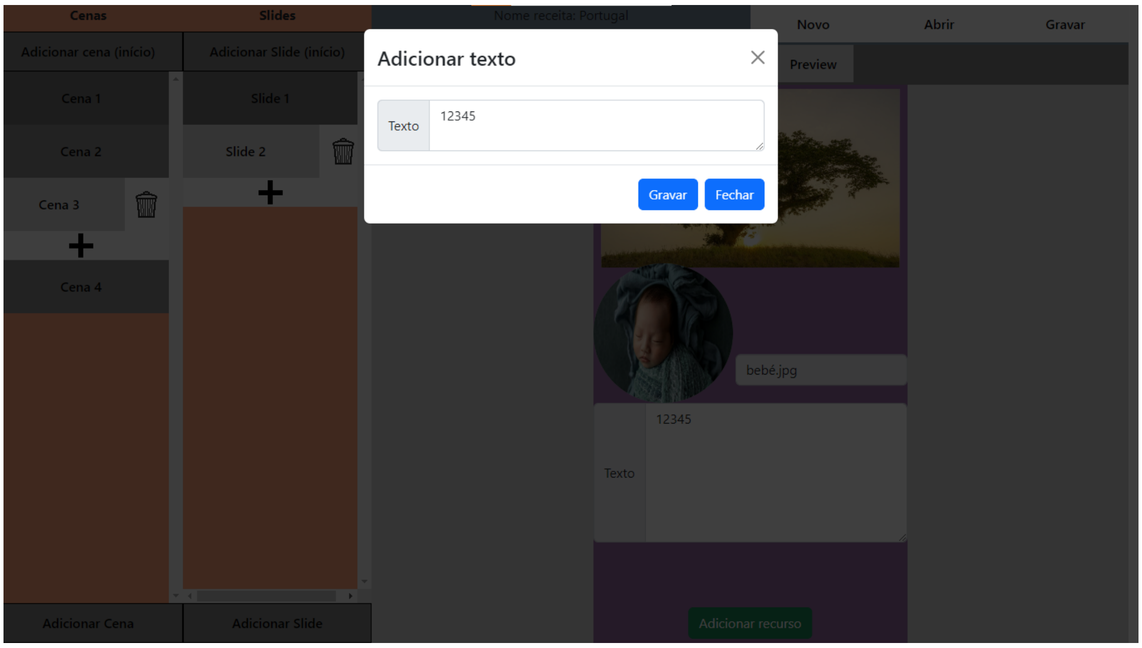
Adicionar imagens antes e depois

URL data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQAI

URL https://thumbs.dreamstime.com/b/purple-flower-2212

Gravar Fechar

Figure B.11: Second React story editor prototype: Add Image Before and After Menu

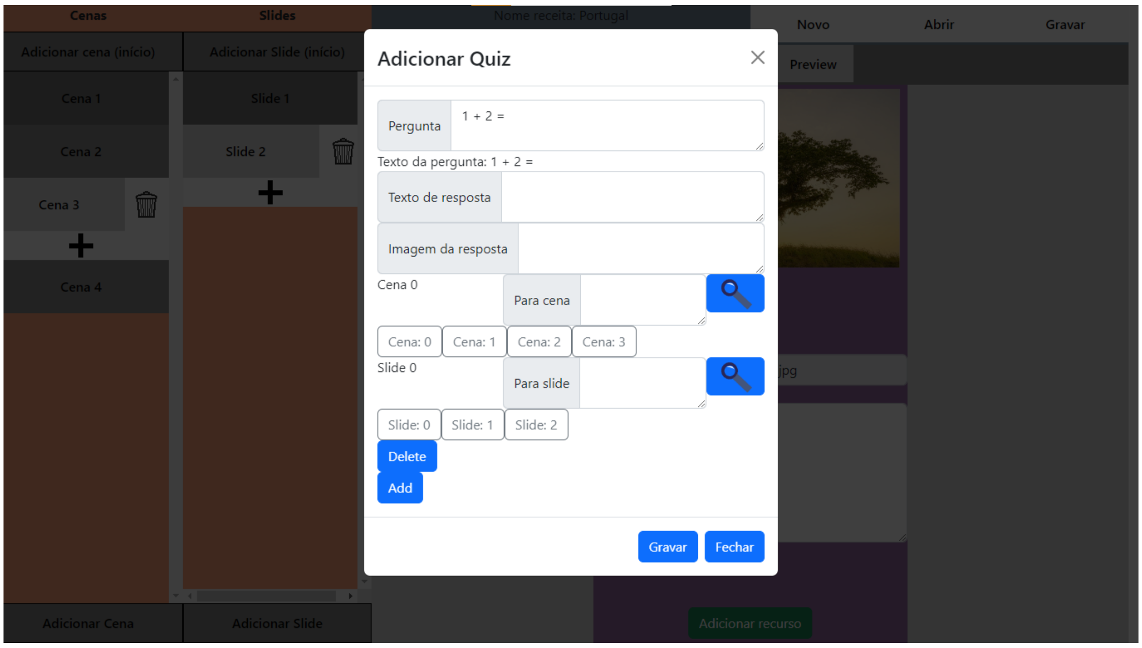


Adicionar texto

Texto: 12345

Gravar Fechar

Figure B.12: Second React story editor prototype: Add Text Menu



Adicionar Quiz

Pergunta: 1 + 2 =

Texto da pergunta: 1 + 2 =

Texto de resposta

Imagem da resposta

Cena 0

Para cena

Cena: 0 Cena: 1 Cena: 2 Cena: 3

Slide 0

Para slide

Slide: 0 Slide: 1 Slide: 2

Delete

Add

Gravar Fechar

Figure B.13: Second React story editor prototype: Add Quiz Menu

B.3 View resource menus

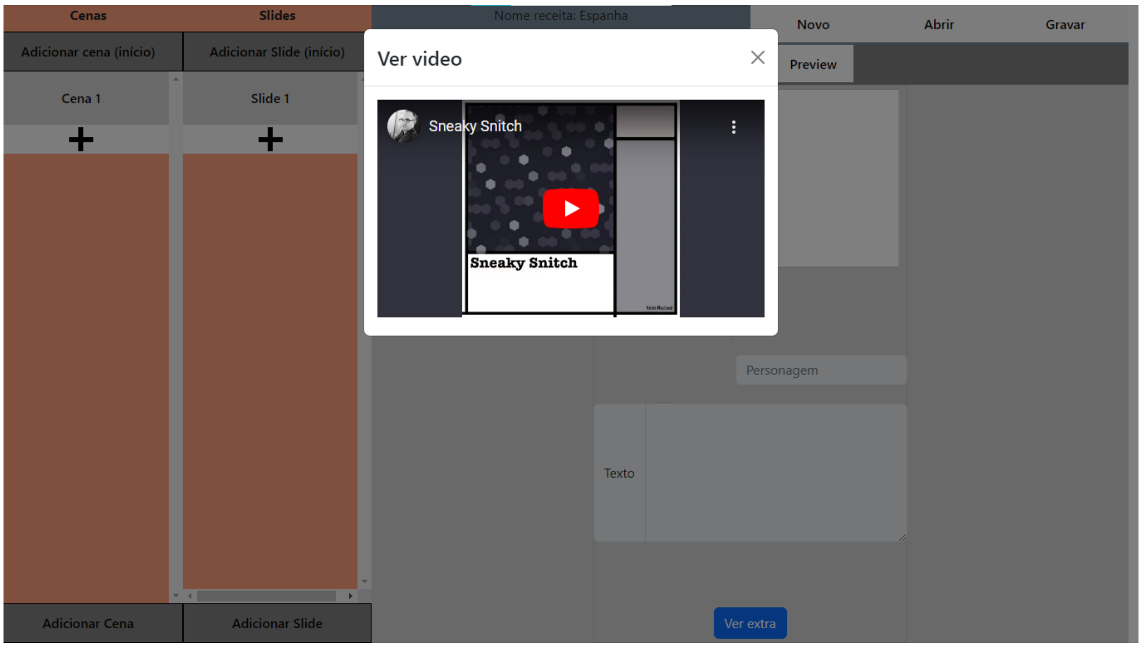


Figure B.14: Second React story editor prototype: View Video Menu

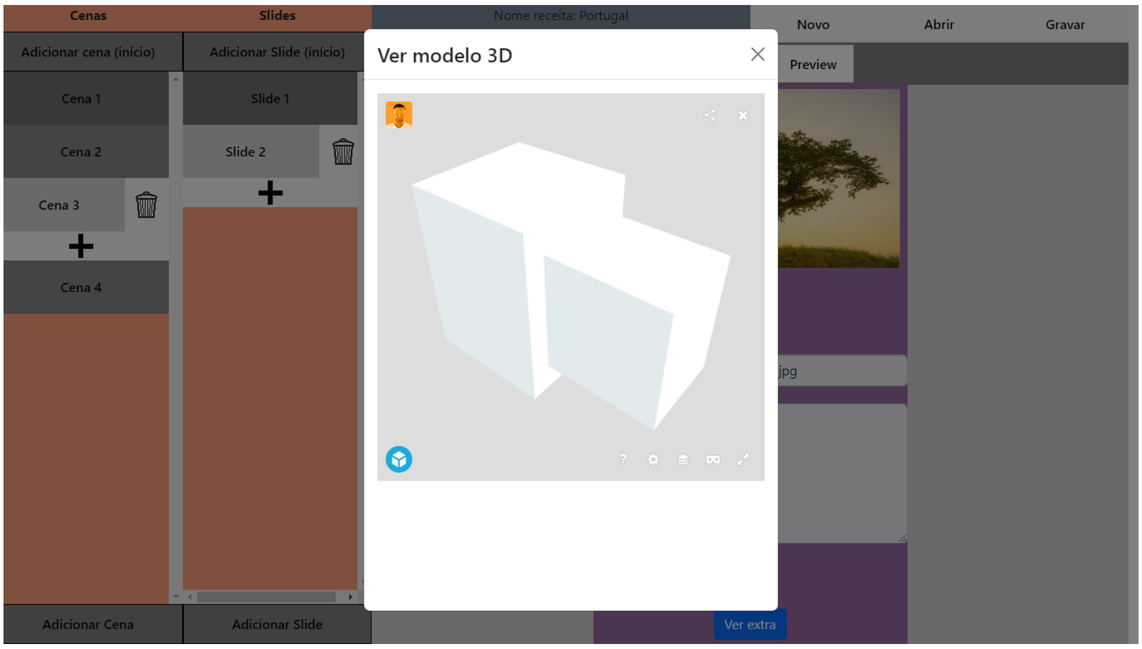


Figure B.15: Second React story editor prototype: View 3D Model Menu

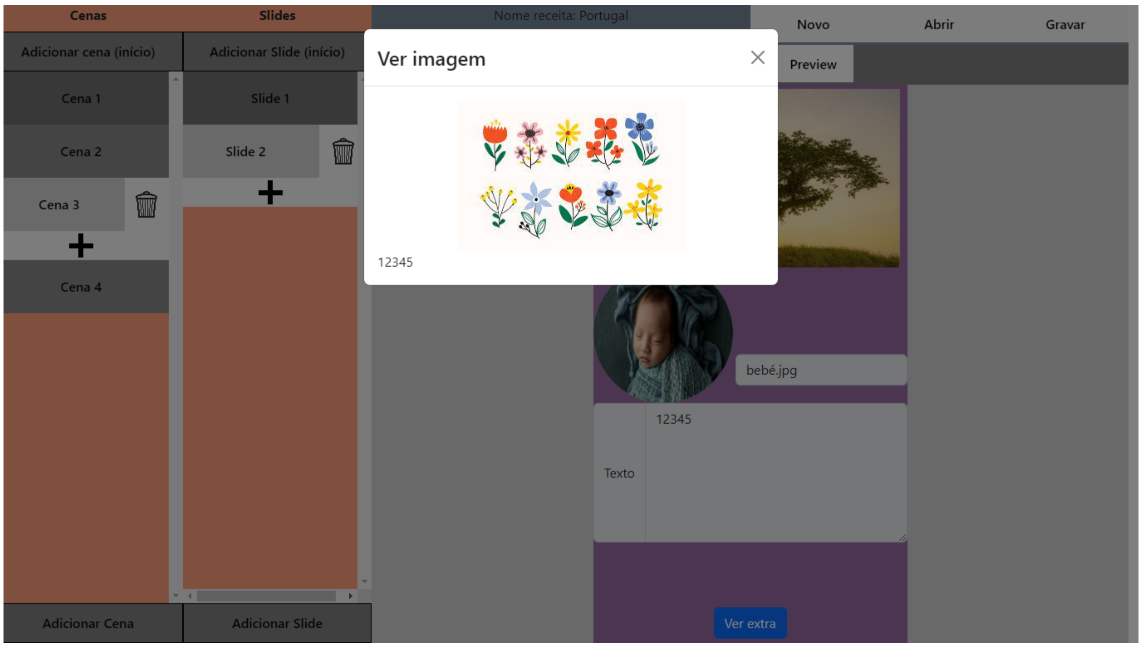


Figure B.16: Second React story editor prototype: View Image Menu

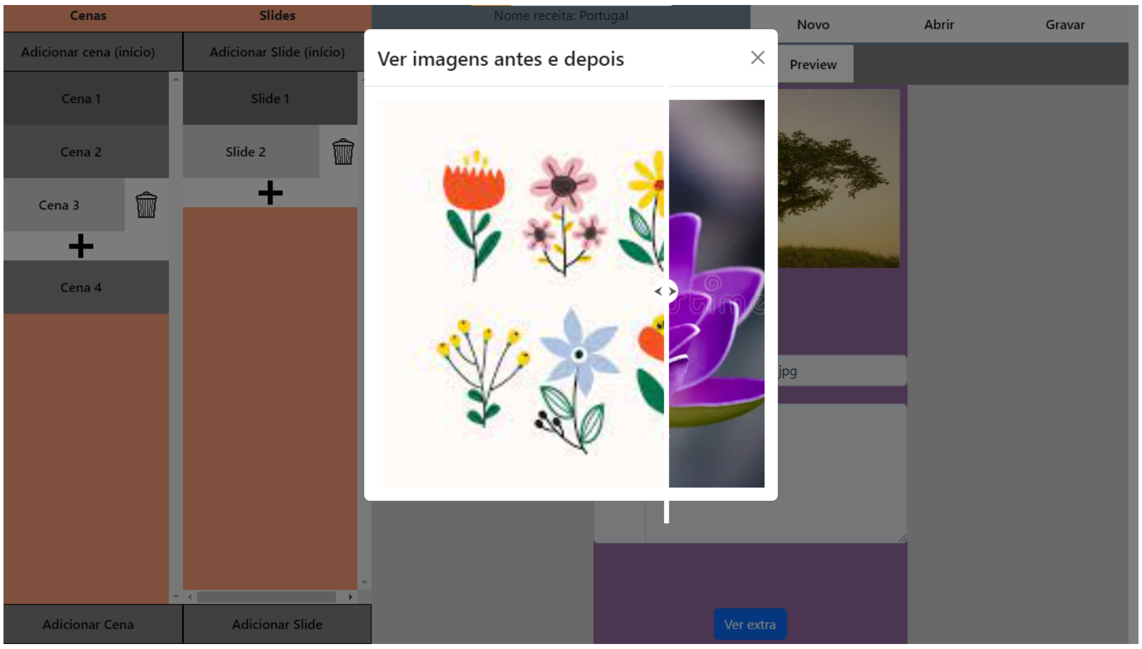


Figure B.17: Second React story editor prototype: View Image Before and After Menu

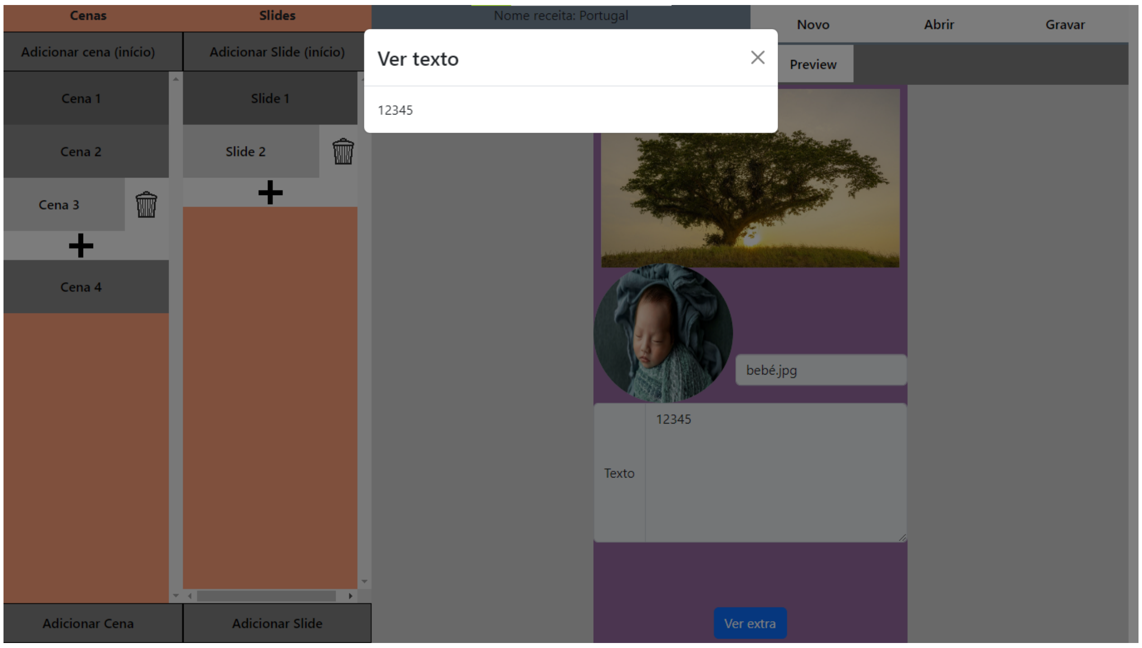


Figure B.18: Second React story editor prototype: View Text Menu

Appendix C

File formats

C.1 React story editor

C.1.1 Story structure (in editor)

```
export interface Story
    scenes: Slides[];
    currentSlideIndex: number;
    currentSceneIndex: number;
    recipeName: string;
    export interface Slides
    slides: Slide[];
    sceneIdentifier: number;
    export interface Slide
    // Random Identifier
    slideIdentifier: number;
    id: number;
    backgroundImageName: string;
    placeName: string;
    placeImageName: string;
    characterName: string;
    characterImageName: string; text: string;
    // Video fields
    videoURL: string;
    videoName: string;
    // Text fields
    // Text that comes with the following extra resources: image, text
    extraText: string;
    // Image field
    imageURL: string;
```

imageName: string;

// Image before and after fields

imageBeforeURL: string;

imageBeforeName: string;

imageAfterURL: string;

imageAfterName: string;

// 3D fields

modelURL: string;

// Destination fields

objectName: string;

objectImageName: string;

destPlaceName: string;

destPlaceImageName: string;

// Quiz fields

quizText: string;

quizOptions: Choice[]; // String indicating which resource is currently added. Possible values: "video", "image", "text", "imageBeforeAfter", "model", "destination", "quiz"

resourceType: string;

export interface Choice

choiceText: string;

choiceImageName: string;

sceneId: number;

slideId: number;

C.1.2 Story structure exported JSON file

export interface ExportStory

startId: number;

historia: ExportSlide[];

recipeName: string;

export interface ExportSlide

verb: string;

actor: string;

place: string;

context: ExportContext;

export interface ExportContext

dialogueButton: string; // String that appears in the button that shows the additional field

embedLink: string; // Youtube video URL for video type actions or Sketchfab URL for 3D model type actions

goto: string; // Name of this destination type action's place lookobject: string; // Name of this destination type action's object

choiceText: string; // Question text to a quiz type action choices: ExportChoice[]; // List of answers to a quiz type action

nextActionId: number; // ID for the next action. Unused for quiz actions

imageBeforeURL: string; // Additional image before and after field before image URL

imageAfterURL: string; // Additional image before and after field after image URL

videoURL: string; // Additional video field Youtube URL

modelURL: string; // Additional model field Sketchfab URL

resourceText: string; // Additional text field text

destinationPlace: string; // Destination actions: The name of the place

destinationObjectPlace: string; // Destination actions: The name of the object

actorImageName: string;

placeImageName: string;

backgroundImageName: string;

destinationImageName: string; // Destination actions: The image of the place

destinationObjectImageName: string; // Destination actions: The image of the object

slideId: number; // Corresponding slide ID

sceneId: number; // Corresponding scene ID

resourceType: string; // String indicating which resource is currently added. Possible values: "video", "image", "text", "imageBeforeAfter", "model", "destination", "quiz"

// Each of these represent an answer to a quiz type action

export interface ExportChoice

choiceText: string; // (The text of the answer)

choiceImageName: string; // (The image of the answer)

choiceId: number; // (The id which the answer points to)