

POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree Thesis

**An Educational Application of Computer
Graphics for Enriching Historical
Understanding of the XX Century**



Supervisors

Prof. Bartolomeo MONTRUCCHIO

Dr. Antonio Costantino MARCEDDU

Dr. Jacopo SINI

Candidate

Simone TAVELLA

DICEMBRE 2023

Summary

The World Wide Web acts as a powerful resource for exploring history and studying past facts. However, visualizing the evolution of these events in their spatial context can be a challenging endeavor. Fortunately, new technologies like virtual reality (VR), augmented reality (AR), or simply 3D desktop visualization are being used to create immersive learning experiences, transporting students to different places and times. As technology continues to evolve, we can expect the emergence of even more innovative and effective approaches to enhance learning.

The project under examination in this thesis is an example of this, indeed outlines the plans for improving an existing Unity educational application, carried forward by the Department of Control and Computer Engineering (DAUIN) at the Politecnico di Torino. This software, still in development, allows users to interactively visualize the evolution of Earth's civilizations across centuries. At its core, the application currently includes a 3D globe that users can explore like a conventional map. However, it is enriched with historical events, placed where and when they occurred. These events comprehend significant battles, milestones in arts and science, advancements in technology, crucial treaties, and other occurrences that have shaped the course of civilization. Additionally, the application allows users to filter events based on their type, enabling a focused exploration of specific historical aspects.

The primary objective of this application is to provide a profound understanding of the geographical and chronological context of historical events. To achieve this, it incorporates a slider allowing users to navigate backward and forward through time, for now, spanning half a century in each direction. Furthermore, another important feature is showing the variability of the countries' borders over time. This feature not only facilitates the examination of territorial expansions and conquests but also provides insight into the birth of nations themselves. Besides the events, the application includes also detailed information about the main cities within each state, categorized into capitals, regional capitals, and simple towns. This feature greatly enhances the ability to locate and contextualize historical events, making the learning experience even more immersive and informative.

The enhancements to this application made by me and treated in this thesis focus on amplifying interactivity and introducing new educational and informative features. The primary improvement involves the addition of two different scenes, representing the "1901-1950" and the "1951-2000" periods, each populated with thousands of historical events. Each one is accompanied by a title, a concise description, the date of occurrence, and a link to relevant Wikipedia resources. Another notable feature of the application is the ability to group states sharing the same color. This feature can be used to highlight, for instance, various colonies that persisted into the twentieth century, offering users a unique visual perspective on historical geopolitical relationships.

The outcomes of this thesis will contribute significantly to the growth of this project, which aims to be utilized for history education in schools and as a broader resource for research into the use of virtual reality, augmented reality, and computer graphics for educational purposes. This thesis will demonstrate how these technologies can enhance interactivity and the learning capability of educational applications.

Contents

List of Figures	IV
1 Introduction	1
1.1 The Confluence of Education and Technology	1
1.2 Objective of this Thesis	2
1.3 Competitors and Inspirations	3
1.3.1 GeaCron [3]	4
1.3.2 Omniatlas [4]	5
2 System Architecture	7
2.1 How it Started: World Map Globe Edition 2	7
2.2 Game Engine: Unity	7
2.2.1 Unity Hub	9
2.2.2 TextMesh Pro	9
2.3 Computer Graphics	10
2.4 Code Editing: JetBrains Rider	10
2.4.1 C Sharp	11
2.5 Control Repository: GitHub	11
3 Unity and World Map Globe Edition 2	12
3.1 Unity Base Elements	12
3.2 Unity Interface with World Map Globe Edition 2	13
3.2.1 Hierarchy	13
3.2.2 Project	14
3.2.3 Console	14
3.2.4 Scene View	15
3.2.5 Inspector	15
3.3 Resources Folder	23
3.4 Geodata Files	23
3.4.1 Country File Format	23
3.4.2 City File Format	24
3.4.3 Categories File Format	24
4 Developer Manual	25
4.1 Add a New Scene	25
4.2 Create a New Event Type	28
4.2.1 FilterMenuToggles.cs	29
4.2.2 WorldMapGlobeCategories.cs	30
4.2.3 WPMCategories.cs	34

4.2.4	Menu.cs	40
4.2.5	WPMInternal.cs	41
4.2.6	Category.cs	42
4.3	GUI Development	43
4.3.1	Add a New Toggle	43
4.3.2	Colorize Countries	43
4.3.3	A Dropdown to Colorize Colonies	49
4.4	Editing of a Scene	51
4.5	New Scenes: "1901-1950" and "1951-2000"	52
4.6	New Categories: "Event" and "Tragedy"	52
4.7	Game Mode	53
5	Conclusions	55
5.1	Results	55
5.2	Future Developments	55
5.2.1	Augmented Reality (AR)	57
	Bibliography	58

List of Figures

1.1	<i>The main map in GeaCron</i>	4
1.2	<i>The main map in Omniatlas</i>	6
2.1	<i>Unity logo [5]</i>	8
3.1	<i>Example of Hierarchy window</i>	14
3.2	<i>Section to modify a country</i>	16
3.3	<i>Section to modify a city</i>	17
3.4	<i>Section to modify an event</i>	18
3.5	<i>Editing buttons</i>	18
3.6	<i>The Reshape tools</i>	20
3.7	<i>The Create tools</i>	21
3.8	<i>The Demo Section</i>	22
4.1	<i>”01 GeneralDemo” folder from Project window (Section 3.2.2)</i>	26
4.2	<i>UI Slider in Hierarchy window</i>	27
4.3	<i>changeDate(Slider slider) function</i>	28
4.4	<i>TogglesCategories(Toggle t) method</i>	30
4.5	<i>Methods that show different categories of events</i>	34
4.6	<i>Declaration of materials, layers and spot</i>	34
4.7	<i>Menage of layers for different categories</i>	36
4.8	<i>Menage of the GameObject related to different categories</i>	40
4.9	<i>ChangeFilter(string filter, bool value) method</i>	41
4.10	<i>Prefabs loading</i>	41
4.11	<i>Materials loading</i>	42
4.12	<i>The enum that associates an integer number with each type of event</i>	42
4.13	<i>UI Toggle element in Inspector window</i>	44
4.14	<i>Function to highlight European Union in 1995</i>	45
4.15	<i>Methods to highlight the deployments of the First and the Second World War</i>	46
4.16	<i>WW1 sides colorized in Game mode</i>	47
4.17	<i>Script to change a text</i>	48
4.18	<i>Function that handles a dropdown button and colors the different colonial empires</i>	51
4.19	<i>The icons for categories ”Tragedy” and ”Event”</i>	53
4.20	<i>Game mode</i>	53

Chapter 1

Introduction

In the current digital age, the internet offers an abundance of information, often freely accessible. However, a significant challenge is that this information is scattered across the web, making it challenging to acquire a comprehensive and cohesive understanding. Nevertheless, technological developments can mitigate this problem by changing the way we interface with the enormous amount of information available.

In particular, this thesis focuses on the historical field, aiming to find a simple and effective method to present significant events in the global development of human civilization to history students and enthusiasts.

This first chapter provides an overview of how technology seamlessly integrates with education. It explores the inception, inspirations, and objectives driving the development of the application covered in this thesis.

1.1 The Confluence of Education and Technology

Over the past decade, education and technology started to get closer, primarily due to significant technological advancements and the increased affordability of technology. The use of this medium in education has the potential to revolutionize learning and create new opportunities for all, not only for the total of students and teachers but also, above all, for those who, with traditional teaching methods, were unable to be involved.

Recent years, spurred by the COVID-19 pandemic and lockdowns, have witnessed a significant surge in technology adoption in education. This extreme condition made us think and paved the way for the near future: there is no turning back now, and it is inevitable to use technology in this area as well. That prompted increased investment due to widespread infrastructural and tool shortages in schools and students' homes.

This new development area is commonly referred to as educational technology or "Edtech". That consists of using hardware and software technology to improve both teaching and learning processes. This can include using computers and applications in classrooms to create online courses and learning platforms.

"Edtech" is usually related to another neologism, namely "Gamification", which is the process of applying game dynamics and mechanics to enhance user engagement, problem-solving, and learning in applications and processes. It infuses game design techniques, mechanics, or gameplay styles into various fields. The challenge lies in shifting perceptions, as users may find it difficult to see fun as integral to learning within an educational context or beyond gaming activities. Training is crucial to encourage people to view learning dynamics differently and thus to convince them that these new techniques can only lead to improvements if used correctly.

Edtech has many benefits, including:

- Enhanced interactivity and engagement in the learning process, not only with the new technology tools but also with educators.
- The ability for students to learn at their own pace, also thanks to the fact that course materials are easier to acquire.
- Access to a broader spectrum of resources via the internet, so alternate forms of knowledge representation (e.g. video, audio, text, image, data).
- Personalized learning experiences tailored to each student, which can also help people with disabilities.
- Facilitating collaboration among students and allowing them to post thoughts, ideas, and comments in virtual groups or social networks.
- Equipping students with digital skills for the modern workforce.
- Creating digital content is generally more cost-effective than producing printed material, which often involves higher production expenses.
- Studying becomes more flexible, aided by the asynchronous management of online lessons, useful for situations like illness or scheduling conflicts.

Of course, challenges accompany the integration of technology in education. For example, it can be expensive to purchase and maintain, and it can be difficult to get teachers trained on how to use these technologies effectively. Additionally, not all students have equal access to technology, which can create a digital divide, especially regarding the latest technologies such as Virtual or Augmented Reality (VR and AR), Artificial Intelligence (AI), and the Internet of Things (IoT). Another drawback is the excessive use of technology, which may reduce human contact and contribute to a shortened attention span, particularly in children and teenagers. While not diminishing the effectiveness of new tools, these issues are more of a social problem, addressable through frugality and education in the proper use of technology.

However, despite these obstacles, the potential benefits of using technology in education are profound. To quantify this idea, the global education technology market size was valued at USD 123.40 billion in 2022 and is expected to expand at a compound annual growth rate of 13.6% from 2023 to 2030 [1]. These data fully confirm that, even at an investment level, the market and development of the edtech sector are now a certainty. As technology continues to evolve, it is possible to expect to see even more innovative and effective ways to use it to improve learning, like the project treated in this thesis.

1.2 Objective of this Thesis

After providing a general overview of the software sector and market treated in this thesis, it is crucial to emphasize how it fits into this context, the motivations, and the long-term objectives behind its development.

This project represents a comprehensive strategy for enhancing an existing Unity-based educational application ("World Map Globe Edition 2" [2]), which was initially acquired and subsequently developed by the Department of Control and Computer Engineering (DAUIN) at the Politecnico di Torino. It is essential to note that the original software was not developed for educational purposes. Originally designed as an asset for creating video games or applications

for various platforms, its primary strengths lay in providing a well-structured global map with a focus on geography and cartography rather than historical or educational aspects.

For this reason, modifications and implementations were necessary to make it more compliant with the edtech sector, specifically focusing on the history and the global evolution of human civilization. Offering a comprehensive view of significant events, associating them with both historical periods and geographic locations, enriches the study experience and enhances user awareness.

The primary goal of all the improvements to the Unity application is to make it more engaging and to provide a stronger emphasis on historical content, to have a working prototype base, with enough data to push the project forward. The progress achieved through this thesis work will not only benefit the application's immediate improvement but will also serve as a valuable resource for future researchers and developers who will work on this application and, more generally, for all those who are interested in leveraging computer graphics and emerging technologies for educational purposes.

The main challenges to reach the objective of this thesis included:

1. Researching and curating a comprehensive list of historical events from the twentieth century, ensuring uniform representation across countries and themes. This involved in-depth exploration of less commonly covered regions like Africa, Asia, and Oceania, as well as sourcing information beyond pure historical events, including scientific and artistic aspects.
2. Locating historically reliable and accurate maps depicting world borders in 1901 and 1951.
3. Entering this data as simply and clearly as possible, seeking to enrich the interface and interactivity of the application to make it increasingly attractive to future users.

1.3 Competitors and Inspirations

In the development of the application, various documentation and inspirations were drawn upon. In addition to online sources, traditional history books were consulted to align the project with its educational objective.

The primary online research source utilized was Wikipedia, serving as a valuable reservoir of information and links associated with the events. It proved especially useful for gaining insights into countries that are often on the periphery but contribute significantly to a comprehensive global perspective.

While considering the selection of events, especially for the main ones, there wasn't much struggle. However, comprehending historical country borders, especially in the twentieth century, required consulting multiple sources, with variations across geographical areas. For instance, factors such as tribal presence in Africa or the complexity of desert landscapes introduced ambiguity to territorial divisions, further complicated by varying levels of detail in the sources.

For information on border management, it's necessary to use Geacron and Omniatlas, and their resources to understand and depict historical geopolitical boundaries. In terms of historical events, Omniatlas was the only source of inspiration. This platform not only offered additional information but also provided convenient links to Wikipedia, streamlining the workflow and enhancing the accessibility of supplementary details.

1.3.1 GeaCron [3]

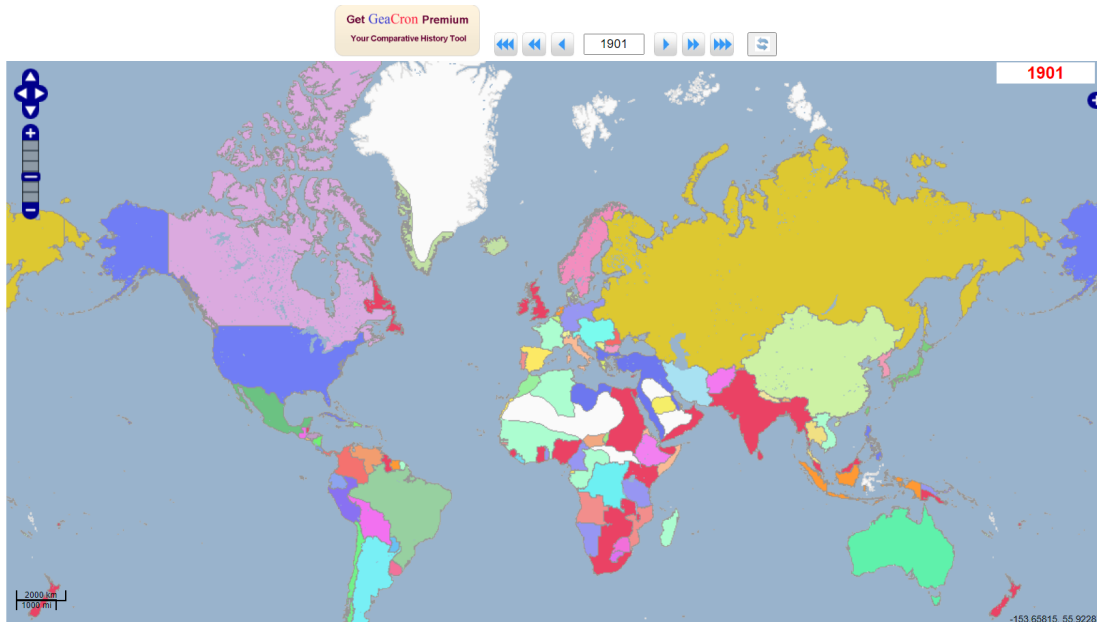


Figure 1.1: *The main map in GeaCron*

GeaCron is a web-based application designed to offer a robust geo-temporal database for historical research, education, and the dissemination of historical findings. Its primary mission is to ensure the accessibility of historical information to a broad audience. It achieves this goal by providing intuitive and visually appealing geo-temporal maps along with configurable timelines.

A standout feature of GeaCron, notably the only one available for free, is its capability to depict the historical state of the world at the commencement of each year. This renders it an invaluable tool for the execution of comparative historical studies. GeaCron aims to simplify and accelerate historical research and analysis by providing a user-friendly interface and access to extensive historical data. As a result, it serves as an invaluable resource for historians, educators, and researchers.

Throughout the thesis, heavy reliance is placed on GeaCron's user-friendly key feature. By simply entering a year in the text box above the map (Figure 1.1), it becomes immediately possible to view the political map of the world in 2D, reflective of the entered year. While it may not be 100% precise, the fact that GeaCron provides a visual representation of historical borders and territories for different years can still offer valuable insights and serve as a useful starting point for historical research.

A premium version of GeaCron is available, resembling the project discussed in this thesis. In fact, it provides a wide array of features and functionalities designed to enhance historical research and exploration.

Here is a summary of the main features of GeaCron, which, if not yet present, can serve as inspiration for future developments:

- Users can click on any part of a political entity on the map for any given date, triggering a popup displaying the name of the selected political entity. Specific information based on the country and date selected is accessible.

- GeaCron provides information on the percentage of the selected political entity's surface area compared to the total area of political entities considered.
- Users can access data on the approximate surface area of the selected political entity for the chosen date. This includes information on the surface area of the largest country at that time, the surface area occupied by the selected country, the surface area of that country occupied by others, and the surface area of other territories.
- GeaCron allows users to explore the reasons for specific border changes related to the selected country for a particular year. Relevant information and links to digital encyclopedias are provided.
- Users can access information about the political context for the selected political entity and date.
- GeaCron provides historical context for the selected date, including information about art, philosophy, literature, science, religion, and other significant events.
- The platform offers links to digital encyclopedia articles related to the country and date selected, as well as articles relevant to the chosen date.
- GeaCron includes a menu with major periods of world history, along with GeaCron timelines for reference.
- The platform provides a list of primary sources that have been consulted to compile historical information.
- Timelines allow a chronological exploration of historical events and developments.
- Users can click on a 3D button to view historical events on a 3D globe. This feature requires HTML5 browser support and offers an interactive way to explore history geographically.

Certainly, the premium version aligns closely with the concept of the application developed during this thesis. Overall, GeaCron provides a comprehensive set of tools and resources for individuals interested in historical research, education, and exploration, rendering it a valuable platform for historians, educators, and history enthusiasts.

1.3.2 Omniatlas [4]

Omniatlas is a comprehensive website that provides a wide range of interactive historical atlases spanning the globe. These maps cover diverse geographical regions and encompass various historical periods, although the majority of them focus on events from the 20th century.

Navigate to the homepage and click on "Maps" in the header. Subsequently, a page displaying a map will appear (Figure 1.2). Here, the user can access more detailed views of different regions of the world, each covering various historical periods. Switching to a specific area view provides additional information, encompassing details related to political geography such as borders, cities, state names, colonial dependencies, and historical events.

Navigation on Omniatlas is facilitated through user-friendly drop-down menus, enabling the selection of specific atlases, years, historical periods, or events of significant importance. Additionally, users can simply choose an area from the main map and then scroll forward or backward in time using straightforward buttons labeled "Next" and "Prev".

In comparison to GeaCron, Omniatlas distinguishes itself with highly detailed maps that offer comprehensive information about events within the selected time frame and geographical area. Events are concisely presented, including their dates (year, month, and sometimes day) and titles,

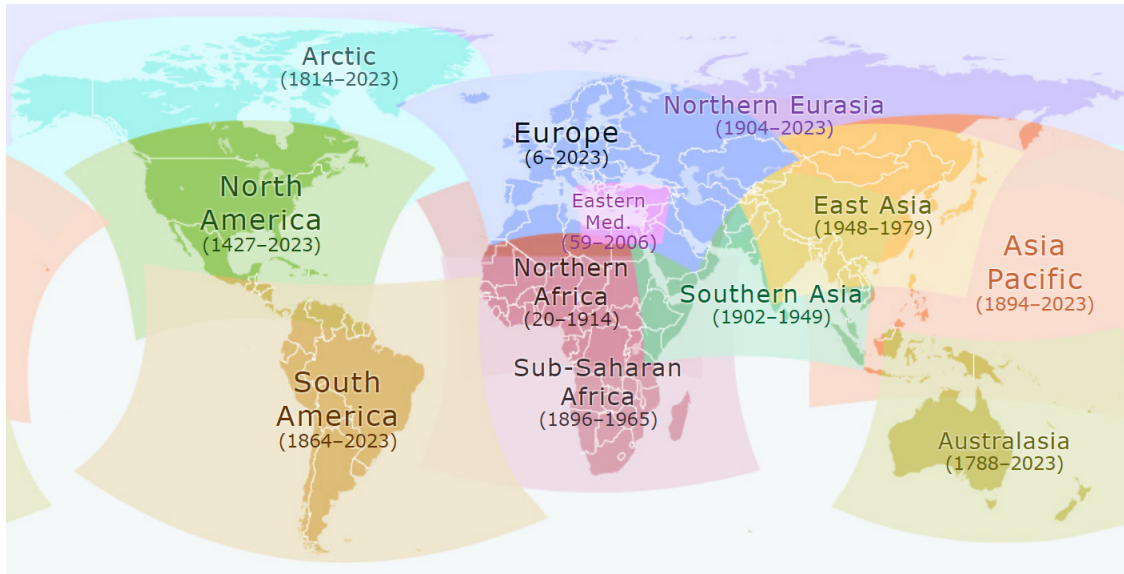


Figure 1.2: *The main map in Omniatlas*

represented on the map by clickable boxes that precisely pinpoint the event location. Clicking on these boxes directs the user to the same page, providing a brief event description along with a Wikipedia icon for additional information. A simple click on the Wikipedia icon redirects users to the corresponding event page on Wikipedia. This feature proved highly beneficial in expediting the search for events, offering clarity on their occurrence locations and providing associated Wikipedia links.

Chapter 2

System Architecture

In this chapter, the key tools and software employed in the thesis work will be outlined. The description will commence with the initial starting point, offering a general overview of all the tools used. Subsequently, a detailed examination of the strengths and, where applicable, any weaknesses associated with these tools will be provided.

2.1 How it Started: World Map Globe Edition 2

A clear starting point is essential for developing an idea. It enables the initial development to concretely visualize the original concept. Using Unity [5] as the game engine, it was crucial to assess and find a suitable asset in the Unity Asset Store that matched the desired features for application growth.

World Map Globe Edition 2 [2] is a 3D world map asset developed on Unity. This was created by Kronnect [6], a software development company known for creating and publishing assets and tools for game development and various applications. They specialize in providing assets for the Unity game engine, which is widely used in the gaming industry. They have been operating since 2015 and have over 200 assets in their catalog, including 3D models, textures, shaders, scripts, and tools.

World Map Globe Edition 2 allows developers to visualize global geography, including country borders, provinces, states, and major cities without an internet connection. It offers integration with online map systems, interactive features like colorization and hover-highlighting, automated country labeling, markers, custom mount points, seamless navigation, and extensive customization options. The asset is compatible with Android and iOS, but also for VR, and serves a wide range of applications, from games to educational software.

The initial version of the asset was 16.2.1, which was the version available at the time of purchase. Kronnect continues to release updates for its application. It is important to monitor the original asset, as they might introduce additional features that can enhance the educational application or streamline development.

2.2 Game Engine: Unity

First, it is necessary to clarify exactly what a game engine is. It is a software framework or platform that provides a set of tools, libraries, and features to facilitate the development, design, and creation of video games. Game engines are used by game developers to streamline the game development process, allowing them to focus on creating the game's content rather than dealing with low-level programming tasks.



Figure 2.1: *Unity logo* [5]

The main characteristics of Game engines are:

- **Graphics Rendering:** they include graphics rendering capabilities to display visuals on the screen. This involves rendering 2D or 3D graphics, handling textures, lighting, and effects.
- **Physics Simulation:** they often incorporate physics engines to simulate realistic physical interactions, such as gravity, collisions, and object dynamics.
- **Audio Support:** they provide tools for managing and implementing audio elements, including sound effects and music, to enhance the gaming experience.
- **Input Handling:** they manage input from various devices, such as keyboards, mice, controllers, touchscreens, and VR or AR headsets allowing players to interact with the game.
- **Scripting and Programming Interface:** Most game engines support scripting or programming interfaces that allow developers to write code to define game logic, behavior, and interactivity. Common programming languages include C# or C++.
- **Scene Graphs:** they often use scene graphs to organize and manage the game world, including objects, characters, and environments.
- **Asset Management:** Game engines provide tools for importing, managing, and optimizing game assets such as 3D models, textures, animations, and audio files.
- **Cross-Platform Development:** Many game engines support cross-platform development, enabling developers to create games that can run on various platforms such as PCs, consoles, mobile devices, and virtual reality systems.
- **Networking and Multiplayer Support:** they include features for managing networking and multiplayer functionality, allowing developers to create online or multiplayer games.
- **Debugging and Profiling Tools:** inside game engines, it is usually present debugging and profiling tools to help developers identify and fix issues in their code, optimize performance, and test game features.

Unity is a versatile cross-platform game engine developed by Unity Technologies and employed for crafting 2D, 3D, virtual reality (VR), and augmented reality (AR) applications. It is important to underline that Unity is free to use for personal projects and provides access to most of Unity's

core features. For the initial development and release of an application, the free version suffices. However, if the revenue grows, upgrading to the Pro version may become necessary.

Unity is useful not only to create a wide variety of games, but also to develop experiences and simulation helpful for everyday life. It is easy to learn and use, even for beginners, also because it has a large community of developers who can provide support and help to create new applications.

There are so many examples of experiences that can be created with Unity, such as:

- Educational application using gamification.
- Virtual training simulations.
- Architectural visualizations.
- Product design prototypes.
- Interactive art installations.

That experiences, thanks to the flexibility of Unity, can run on a variety of devices, including PCs, consoles, mobile phones, VR and AR headsets, so the created application can evolve in different ways.

The Unity version used in this project is 2020.3.25f1, which is essential for current work to make sure to not insert any new bugs. However, for future development and to harness the enhanced capabilities of the game engine, an upgrade to a more recent and stable version may be necessary.

2.2.1 Unity Hub

Unity Hub is a management tool provided by Unity Technologies for developers using the Unity game engine. It offers several features to streamline the development process, making it easier to create, manage, and update Unity projects.

Unity Hub is utilized to promptly open the project and receive immediate feedback on the last time it was accessed. Its effectiveness becomes particularly pronounced when dealing with multiple projects, each employing different versions of the game engine, as it allows for easy selection before opening.

2.2.2 TextMesh Pro

TextMesh Pro (TMP) is a powerful and versatile text rendering solution for Unity.

It offers a wide range of features, including:

- High-quality text rendering: TMP uses advanced text shaders to render text that is both sharp and smooth.
- Flexible text formatting: TMP enables the formatting of text in a wide variety of ways, including the setting of font, size, color, and alignment.
- Support for multiple languages: TMP supports a wide range of languages, including English, French, German, Spanish, and Japanese.
- Customizable text meshes: TMP enables the creation of custom text meshes that can be used to display various types of text, such as buttons, labels, and status bars.
- Easy to use: TMP is user-friendly and can be seamlessly integrated into Unity projects with just a few clicks.

For now, this tool, while supported by the application, sees limited usage since, in an evolving project, graphical aspects take a back seat to functionality. However, it will become more valuable in the future for improving readability and managing language changes, especially since the current content is only in English, it will be surely in Italian too.

2.3 Computer Graphics

Given the predominant role of the visual component, particularly the 3D globe, in this project and generally in many other "Edtech" applications, it is worth taking a moment to grasp the concept of computer graphics.

Computer graphics, a field in computer science and digital art, revolves around creating, manipulating, and displaying visual content, including three-dimensional objects and scenes, using computers. It plays a central role in VR and AR experiences, enabling immersive virtual worlds and enhancing real-world environments with digital elements. This field is essential in numerous industries, such as entertainment, education, design, engineering, medicine, and scientific visualization, driving innovation for more realistic and interactive digital experiences.

In Unity, computer graphics is integral for crafting immersive 2D and 3D experiences in games, simulations, and applications, with the platform providing a robust framework for graphics, rendering, and visual effects.

2.4 Code Editing: JetBrains Rider

Initially, Microsoft Visual Studio was used for code editing. However, based on a recommendation from another thesis student working on the same project, a switch was made to JetBrains Rider. This option is more comprehensive and is available for free to university students.

JetBrains Rider [7] is a cross-platform IDE (integrated development environment) for .NET development, so it uses the .NET framework, a versatile and powerful development platform created by Microsoft. It is based on the IntelliJ IDEA platform (another IDE by JetBrains) and ReSharper (a set of development tools), and it provides a comprehensive set of features for developing .NET applications, including:

- Code editing: Rider provides a powerful code editor with features such as code completion, syntax highlighting, and refactoring.
- Debugging: Rider features a robust debugger that enables users to step through their code, inspect variables, and set breakpoints.
- Unit testing: Rider supports unit testing with NUnit and xUnit.
- Refactoring: Rider offers a variety of refactoring tools designed to assist in improving code quality.
- Profiler: Rider is equipped with a profiler that enables the analysis of code performance.
- Integrations: Rider integrates with a variety of other tools, such as Unity, Docker, and Git.

As for JetBrains Rider's free plan for university students, JetBrains offers a program called "JetBrains Student Pack" or "JetBrains Student License" that provides free access to Rider and other JetBrains tools for students who meet the eligibility criteria.

The primary use of this tool commenced for its simplicity in conducting project-wide searches, covering all the scripts. It greatly aids easy reverse engineering, which is particularly valuable as the project was created by external developers. By left-clicking and highlighting a word in

the code and then left-clicking several interesting tools appear: Find in files, Find Unity usages, Go to Declaration or Usages, etc... These tools make it easier to identify duplicated elements in multiple scripts or pinpoint the source of a method.

2.4.1 C Sharp

C# is a modern, object-oriented, and statically typed programming language developed by Microsoft and is part of the .NET framework. Widely used for developing Windows applications, web applications, and various other types of software, C# shares syntax similarities with C++ and Java, facilitating an easy transition for developers familiar with these languages.

These are the main key Features of C# Programming:

- **Code Structure:** C# projects are built at the code level using .cs files, which are source code files in the C# language. In a .cs file, it is necessary to define namespaces and classes. Namespaces organize the code, while classes define objects and their behavior. Comments in .cs files document the code and can be either single-line (`//`) or multi-line (`/* ... */`).
- **Syntax and Rules:** C# has its syntax and a set of rules governing variables, data types, control structures, and functions. Code is written in a text format following C# language conventions.
- **Cross-Platform Development:** C# has become more cross-platform, allowing developers to create applications for Windows, macOS, and Linux.
- **Library and Framework Support:** C# has access to a wide range of libraries and frameworks provided by the .NET ecosystem. This enables developers to leverage pre-built functionality for various tasks.

In summary, .cs files are a fundamental part of C# development, serving as containers for code that defines the behavior and structure of C# programs. These files are managed within C# development environments and are essential components of software development using C#.

2.5 Control Repository: GitHub

GitHub [8] is a web-based version control repository hosting service that offers software development and version control using Git. It is a central location to store the main project: this makes it easier to find and access code, and also collaborate with other developers. It tracks changes over time and allows us to see how code has evolved, so it is possible to revert to previous versions.

It is free to use for open-source projects, which makes it a great option for small teams and individuals who are working on open-source projects. It is possible to create a branch, that is a separate line of development within a repository. This allows for the development of new features, bug fixes, or experiments without impacting the main or "master" branch.

Chapter 3

Unity and World Map Globe Edition 2

This chapter describes the development setup and environment strictly related to the World Map Globe Edition 2 start point. Most of the project's peculiarities that were utilized and consequently need to be introduced with a general overview pertain to the Unity interface, the code-base architecture, and all other useful components essential for upgrading the project. This chapter and the next one can be a handy starting point for future developers who will continue working on this project. In particular, Chapter 3 is more useful for the figure of the editor who interfaces most with the software that runs on Unity, Chapter 4 instead treats the code part and all the related elements.

3.1 Unity Base Elements

In this section, elements commonly used in Unity application development, which will also be present in this thesis, will be defined.

Game Objects are the fundamental building blocks in Unity and represent entities or objects in the application, like characters, items, scenery, and more. Components are attached to game objects, defining their behavior, appearance, and functionality. Common components include for example Transform, Rigidbody, Collider, and C# Scripts. The last ones are used to add custom functionality to game objects: they control game logic, interactions, and behaviors.

Materials in Unity define a 3D object's visual appearance, specifying surface properties like color, texture, transparency, and how it reacts to light. Textures are 2D images that support various formats, including PNG, JPEG, and TGA, assigned to materials to enhance their details.

Another important element often used in Unity is prefab, a template for creating game objects, promoting reusability and consistent object templates across scenes. A prefab's main property is that all its changes are reflected in all instances throughout the project. In a Unity project, prefab files are represented by a light blue cube icon, while game objects have the same cube but with no specific color, so it is easier to distinguish them.

All those elements are usually put in a scene, a container for organizing the various game objects, components, and assets that comprise different application parts. Like in a movie, to make the scene work it is necessary to insert a camera and lights. Cameras are used to define the viewpoint of the player in Game mode, capture the scene, and render it to the screen, providing different perspectives and effects. Lighting is crucial for setting the mood and atmosphere in the application, so there are different kinds of it, like point lights, directional lights, and spotlights, among others.

Related to the lighting, in Unity, there are elements called shaders. They are programs written in a language called ShaderLab, which is a subset of the CG programming language. Shaders are used to control the rendering process of 3D graphics, determining how surfaces react to light, shadows, and other visual effects. They play a crucial role in creating the visual appearance of objects in a Unity scene.

Last but not least, the "GUI" (Graphical User Interface), is a visual way for users to interact with software applications, including video games. A GUI includes graphical elements like windows, icons, buttons, and menus, making it easier for users to navigate and interact with the software compared to using only text-based or command-line interfaces.

In Unity, a GUI refers to the UI elements that users see and interact with in applications. These elements include buttons, menus, toggles, score displays, on-screen text, and various other graphical components that enhance the user's experience and provide feedback and control options.

Unity provides a range of tools and components for creating GUI elements, allowing game developers to design and implement intuitive interfaces to enhance gameplay and user interaction. GUI elements are often used for displaying information, managing settings, and enabling players to perform actions within the application, such as starting or pausing it, accessing inventory, or adjusting options.

3.2 Unity Interface with World Map Globe Edition 2

The main software used to develop this project is the game engine Unity, so it is really important to know most of the parts of its interface to interact with it. This will be modified due to the presence of the starting project "World Map Globe Edition 2" which will populate it with its various Scenes, Game Objects, Prefabs, and Scripts.

3.2.1 Hierarchy

The Hierarchy window displays every Game Object in a Scene (Figure 3.1), in this case in the "1951-2000" scene), such as models, cameras, lights, or prefabs. When a Game Object is added or removed in the Scene view, the same thing happens from the Hierarchy window. The primary element in the Hierarchy is the "WorldMapGlobe" prefab, which encapsulates the majority of features found in the World Map Globe Edition 2 application, as evident from its name. It represents the 3D globe in the center of the scene and encapsulates within it all the properties that affect it (countries, events, cities, borders, etc...).

The other two prefabs in this window represent key program elements: the interactive menu enhancing the graphic interface during Game mode, and a basic pause menu, currently somewhat bare. There is also the Main Camera which guides the point of view in Game mode and the Sun, the only light source that illuminates the scene. To clarify, the other Game Objects present in the Hierarchy have not been directly used, as they are not relevant to the research conducted in this thesis.

It's useful to sort and group the Game Objects in the Scene: in this scenario, prefabs like PauseMenu and InGameMenu are structured like the UI in the Scene view.

For example, the InGameMenu is subdivided into four main parts:

1. The PanelCategoryToggles, where all the different kinds of event toggles are grouped.
2. The SliderEmpty, which contains the slider object that allows to change the Scene, so also the period, in Game mode.
3. The Panel Left includes all the left parts of the UI (Toggles, Buttons, and a Dropdown Button).

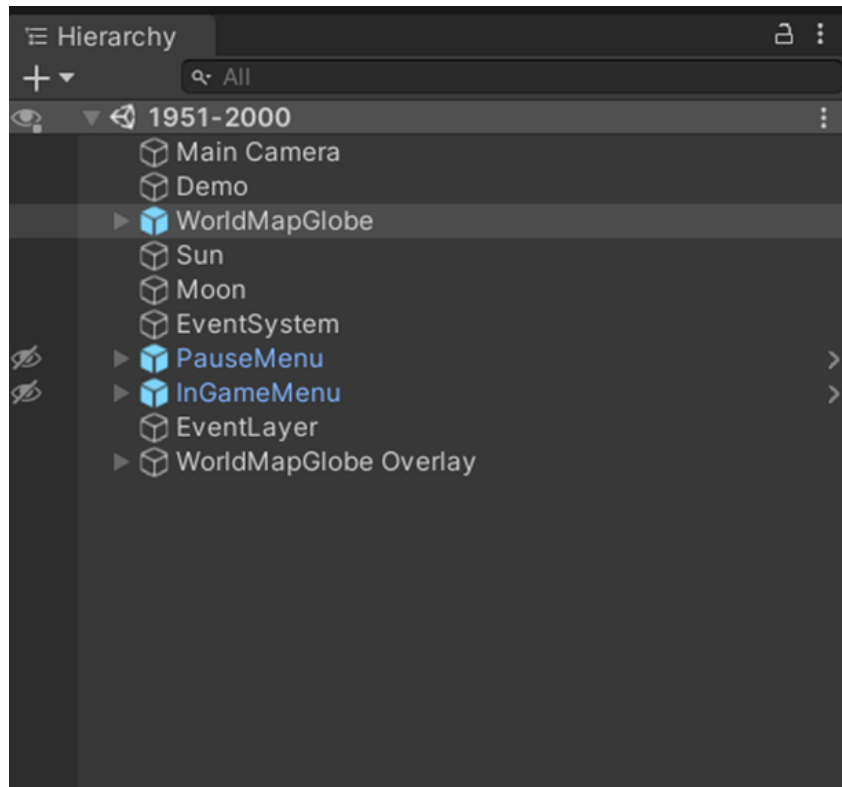


Figure 3.1: *Example of Hierarchy window*

4. The Panel Right Cities Filter, where all the different kinds of cities toggles are grouped.

This subdivision can then be improved as the user interface evolves, the important thing is that it is still a clear hierarchy with easy associations with the elements in the Scene View.

3.2.2 Project

The Project window, typically located below, serves as a direct navigation hub within Unity's project folder. It streamlines the editor workflow by providing user-friendly access and selection of assets. Moreover, it simplifies the process of associating scripts with objects or graphical elements in the scene; the editor can easily drag and drop them on the Inspector window, after selecting the element, for quick utilization. In essence, it is a valuable tool for inserting and linking elements, like textures, without the need for an extra window displaying the project folder, but it also makes navigating and exploring the project easier, also because of the search bar.

3.2.3 Console

The Console window identifies issues, displays errors, warnings, and other messages generated by Unity, tracks game behavior, and debugs code. It can also be used to show the programmer messages in the Console using the `Debug.Log`, `Debug.LogWarning`, and `Debug.LogError` methods inside the IDE JetBrains Rider to understand better how the code works.

When an error occurs, it is easier to understand it through the console window as the message that appears tells all about it: the kind and the number of errors and warnings, in which script,

and in which line comes from. It is also possible to click on it and open the relative script to see immediately where can be the problem.

3.2.4 Scene View

In Unity, the Scene view is a critical component of the Unity Editor, and it serves as a window where interactively create, edit, and navigate the application's scene. Here are the basic controls for navigating in Unity's Scene view:

- Select an object by clicking the Left Mouse Button (LMB).
- Rotate: hold down the Right Mouse Button (RMB) and drag the mouse to pan the view horizontally and vertically. Another way to do this is to hold down the Alt key (Option key on Mac) and the Left Mouse Button (LMB) simultaneously, then drag the mouse to orbit around the center of the view.
- Zoom: use the scroll wheel on the mouse to zoom in and out. Alternatively, it is possible to hold down the Alt key (Option key on Mac) and use the Right Mouse Button (RMB) to zoom in and out.
- Focus on Selection: press F to focus the Scene view on the currently selected object. This centers the view on the selected object.

3.2.5 Inspector

The Inspector in Unity is a window that displays information about the currently selected object or prefab. It can be used to view and edit the properties of objects, such as their position, rotation, scale, and components. In this case, it is really important to analyze the Inspector window after selecting the WorldMapGlobe prefab in Hierarchy, so it is possible to interact with it, changing its properties or those related to some scripts.

So in this window, we have:

1. The object transforms (position, rotation, scale).
2. The World Map Globe script.
3. The World Map Editor script.
4. The Demo script.

Below, an examination will be conducted on the main features that distinguish the latter three elements, which serve as the primary tools for the editor.

The World Map Globe Script

This program section serves as the central hub for managing the design of the entire scene. It encompasses a range of tasks, including the management of the texture applied to the Earth's surface, control over various attributes related to borders (such as color, thickness, and visibility), as well as the customization of countries like them highlighting when the mouse passes on it, and their name characteristics(font, size, and shading).

Furthermore, it extends its reach to cities, allowing adjustments in their icon sizes, visibility, and colors. This section also facilitates cursor management for user interactions, and camera control for navigation, and provides information about the folder where all data related to event categories, cities, and the positioning of control points that define states are stored.

This part of the interface was rarely used for refining the application’s visual appearance, as the primary focus was on content creation. However, with the increasing number of events on the map, the dense presence of cities became a hindrance. Consequently, deactivating them using the “Show cities” toggle found inside this section was deemed more convenient.

Additionally, an issue was encountered with country names occasionally positioned outside territorial borders. In an attempt to address this, changes were made to the relevant settings to assess potential improvements. This issue was associated with the method used for name placement, involving the most extreme vertices of the borders in terms of longitude and latitude. Despite attempts to resolve it, the tools provided by this Inspector section affected all country names, making it challenging to address specific cases. Consequently, the decision was made to maintain the current approach due to the complexity of the required code changes.

The World Map Editor Script

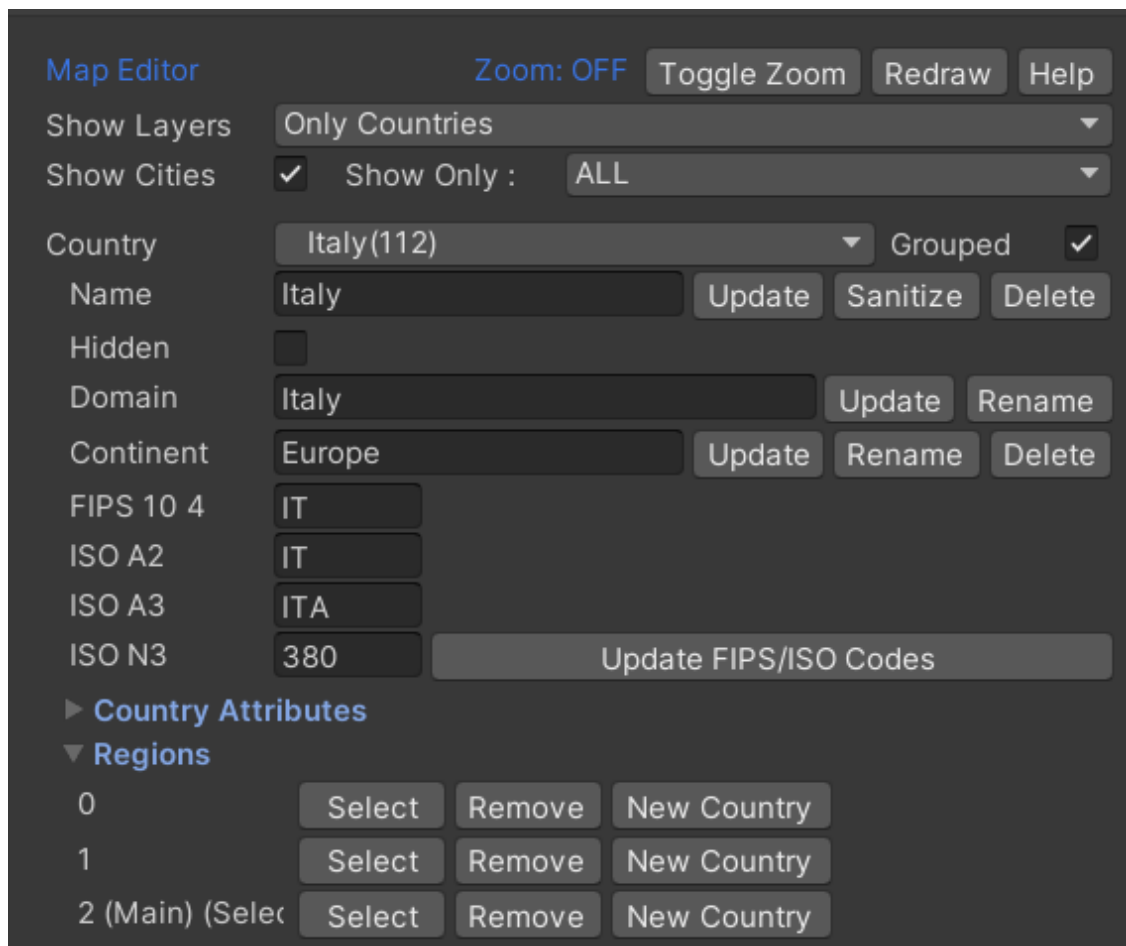


Figure 3.2: Section to modify a country

This section of the interface is dedicated to managing the primary changes that impact the project specifically focusing on the administration of countries, their borders, cities, and most important, historical events.

First, when the editor selects a country, it is initially presented with essential information (Figure 3.2) including the country's name, its domain (which proves particularly useful for managing colonies), and the continent to which it belongs. A dropdown menu is also available, allowing the full list of countries in the scene to be viewed, broken down by continent. Switching between countries can also be accomplished by selecting them via this alternative mode. Although Country ID codes are currently present, they have not been utilized thus far. However, they could potentially be converted into useful fields in the future.

Additionally, there is a list of the distinct regions that constitute the country: it is important to note that these regions refer to continuous land areas within the country rather than distinct administrative regions. For instance, Italy is made up of three continuous regions, namely the mainland and the two smaller islands, Sicily and Sardinia. Of these, the editor can either delete them individually from here or make them autonomous by transforming them into individual countries.

It is important to highlight that next to the text fields, there are update buttons, critical for applying changes on the Scene view, although it is impossible to visualize the alterations. Below, there is a button dedicated to permanently saving all the updates made during editing, a topic that will be discussed later.

After the section where it is possible to modify the generic attributes of a country, there is, more or less in the same format, the one to modify the selected city (Figure 3.3) or the one nearest to where the editor has clicked on the map on the Scene View. It is possible to change various attributes of the city, including its name, type (Country Capital, Region Capital, or Normal City), population, the province it belongs to, and its precise location defined by latitude and longitude. Once more, a dropdown menu is available, offering a comprehensive list of cities belonging to the same country. This facilitates navigation between selected cities within the same country using this list. It is really important to underline that to apply any changes to the Scene view, the editor must press the "Update City" button at the end of this section.

City	Rome (11191)	
Name	Rome	
Class	Country Capital	
Population	3339000	
Province	Roma	
Lat/Lon	X 41.89593	Y 12.48327
Update City		
▶ City Attributes		

Figure 3.3: Section to modify a city

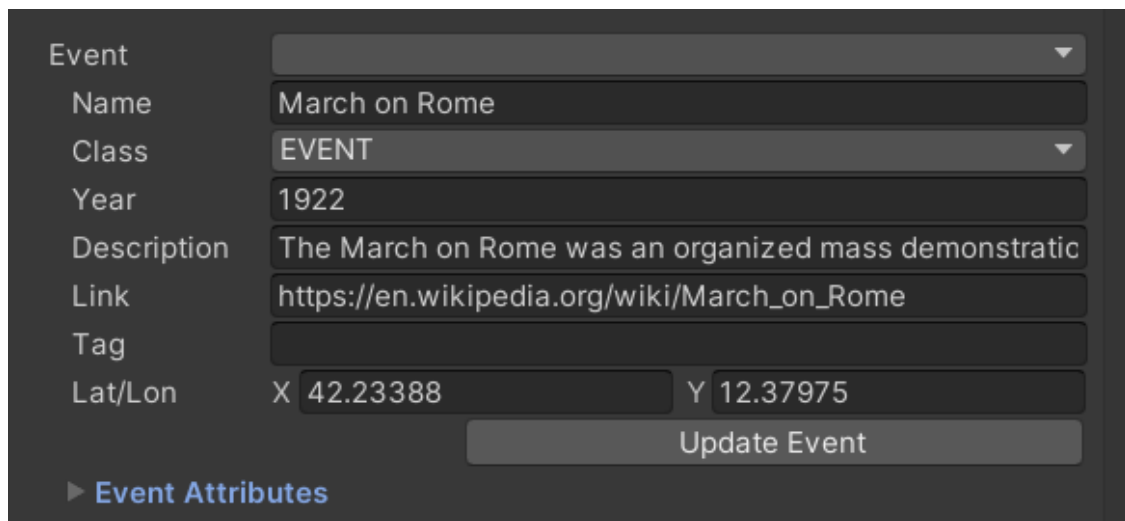
Under that part, after selecting an event (or if not, the application shows the nearest event where the mouse clicked the last time on the map on the Scene View), the editor can customize its attributes here (Figure 3.4). Starting from the beginning, the editor can change the event title, followed by a drop-down menu where it is possible to choose from various event types, such as battle, art, discovery, science, treaty, fun fact, human rights, tragedy, and general historical event. Changing the event type will also update its icon on the map, which by default is assigned the type and icon of the nearest event to it.

Next, the editor can modify the year in which the event took place. Currently, it is only

possible to enter a whole number. Therefore, for events that have lasted for an extended period, the practice is to indicate the year in which they began. Conversely, for events specific to a particular day, it is useful to enter the exact date in the next field. This provides a brief description of what exactly the event is about, to present it more fully to the end user in Game mode.

Below that, there is a field where it is possible to add a link, usually from Wikipedia, to create an interactive connection to the icon in Game mode. A tag field has also been added to facilitate the tagging of events belonging to different categories but connected by a common logical thread (e.g., "WW2" for events related to World War II). Similar to cities, there are also the event's coordinates, specifying its longitude and latitude.

Finally, at the bottom, there is a button enabling the update of the event within the scene without saving it permanently.



Event	<input type="text"/>	
Name	March on Rome	
Class	EVENT	
Year	1922	
Description	The March on Rome was an organized mass demonstratic	
Link	https://en.wikipedia.org/wiki/March_on_Rome	
Tag	<input type="text"/>	
Lat/Lon	X 42.23388	Y 12.37975
<input type="button" value="Update Event"/>		
▶ Event Attributes		

Figure 3.4: Section to modify an event

After the three sections for modifying the scene's core elements, the editor will find buttons (Figure 3.5) that enable direct interactions with the map on the Scene View and the creation of new elements within it. To be clear and concise, it is necessary to dissect every single button one by one, focusing on its characteristics and what it allows the editor to do.

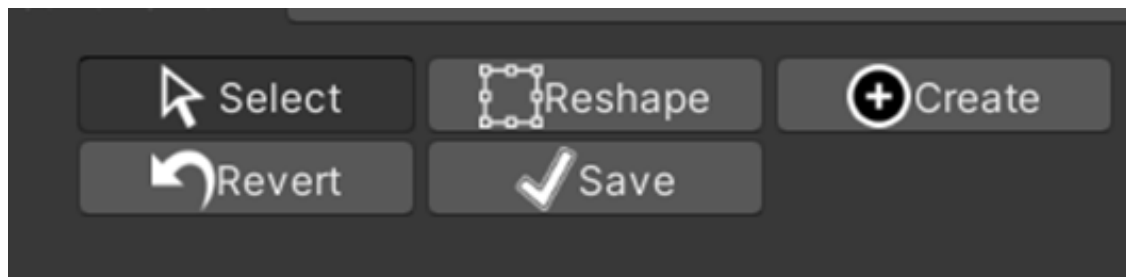


Figure 3.5: Editing buttons

- **Select**

First, this tool allows selecting a specific country with a left click. Then automatically, depending on where the click occurred, it also selects the nearest city and, if present, the nearest related event. After that, it is possible to click on another city or event in the same county or select another one. When a county is selected the area changes to red and all the borders' control points become visible, but not changeable. It is possible to change the highlight color in the World Map Globe script [3.2.5](#).

The Select tool enables the interaction and modification of the countries, the cities, or the events, as, before using the editing tools indicated above it is necessary to select exactly what the editor wants to interact with. This disrupts the workflow, particularly when editing multiple items consecutively. Essentially, after finishing an edit on one element, it is necessary to click the Select button again to choose another one and then click Reshape to reactivate the edit mode.

- **Reshape**

By clicking the Reshape button, new features for editing appear below (Figure [3.6](#)), but this happens only if a country has previously been selected. If it is not, it is necessary to click on the Select button. Even in this case, it is better to analyze them one by one to understand what they are allowed to do.

- Point: this tool permits the movement of every control point of the selected country's border.
- Circle: after selecting a point of the border, a circle appears centered on it. Under the buttons, appears a slider that can change the circle's radius and determines the controlled area. When it's enough big, if the editor starts moving the point, so all the others inside the circle will move.
- SplitV: splits the country with a vertical border that can be controlled with a slider, changing its position to the left or to the right.
SplitH: splits the country with a horizontal border that can be controlled with a slider, changing its position up or down.
Both, SplitV and SplitH create two countries: the larger one retains the same name and characteristics as the original. The other is created with the name of the original country preceded by "New." The issue is that all the cities and events within it are inherited from the other half, making it challenging to select and edit them independently.
- Magnet: This tool works by clicking repeatedly over a group of points that belong to different regions. It will try to make them join, fixing the frontier. Note that there must be a sufficient number of free points inside the circle's tool, so they can be fused.
- Smooth: creates a new point between two that already exist. It is really useful to make borders more precise and to fit them with the nearer ones.
- Erase: cancel control points of the selected country, also in this case under the buttons it appears a slider that can change the circle's radius and determine the area in which points will be deleted.
- Delete: with this button selected it is possible to delete the entire region selected, or the entire country if it has no other regions.

In general, buttons are always present below to change the position or delete the selected city and event in this country like in Figure [3.6](#).

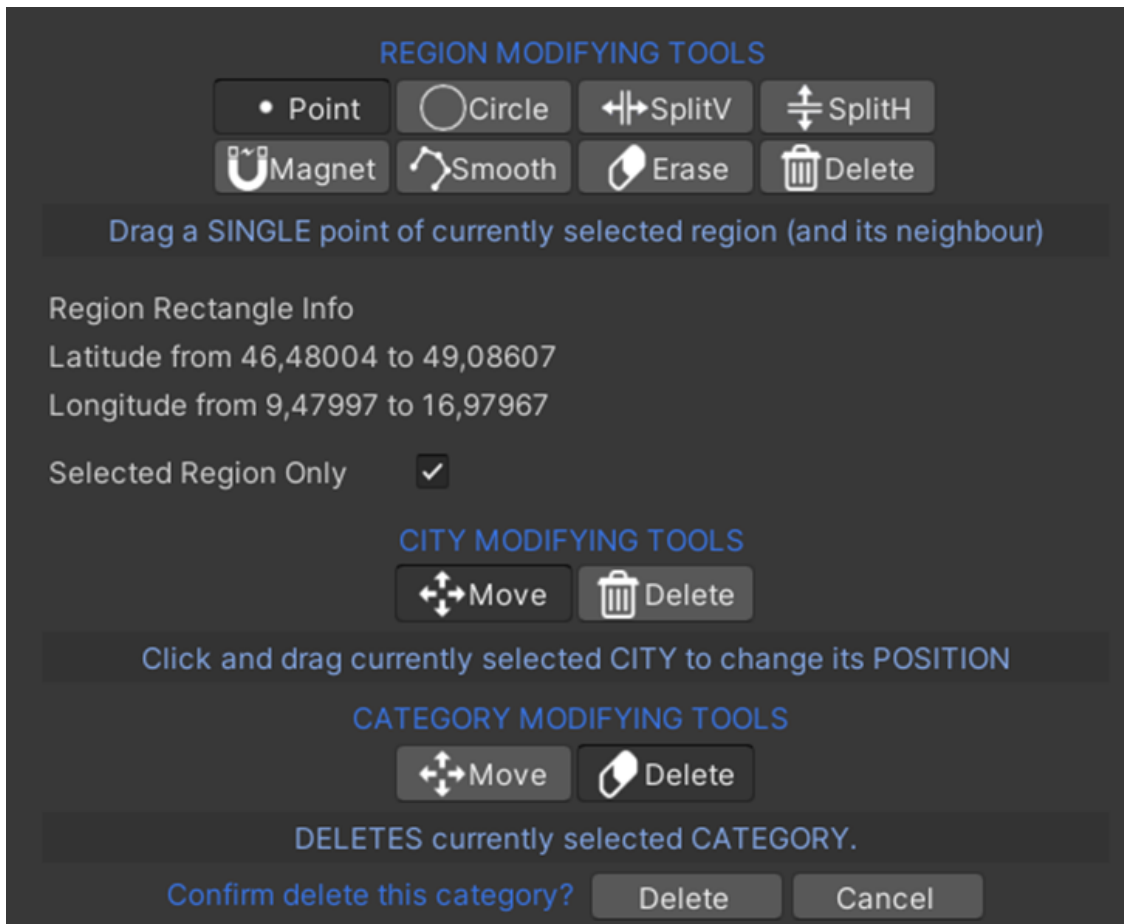


Figure 3.6: The Reshape tools

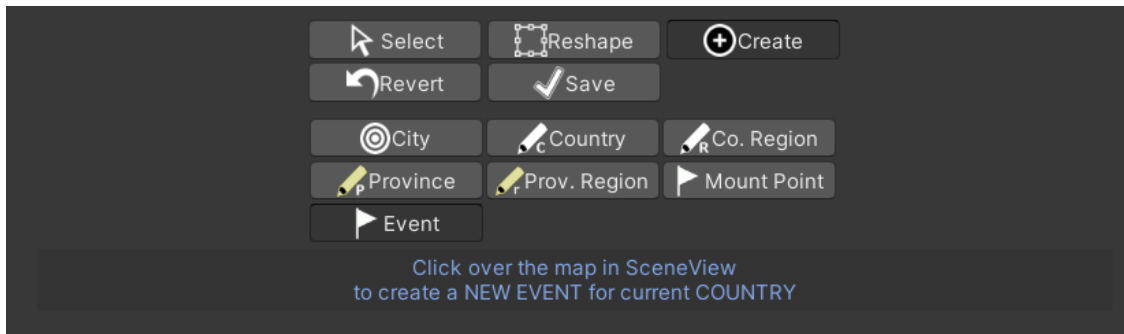


Figure 3.7: The Create tools

• Create

Once the Create button is pressed, several tools for creation will appear below as can be seen in Figure 3.7. The various creation methods are discussed one by one below:

- City: passing the mouse over the Scene View, a flashing dot appears which indicates the activation of the Create tool. Then, by clicking the left mouse button in the position where the editor wants to insert the new city, it will be immediately created. By default, the city is already populated with some graphical and not elements related to the nearest city in the same country to make it already visible in the scene, so it can be easily edited.

The new city obviously belongs to the country selected before creation, but not necessarily within its borders. This can be advantageous in some cases, for example, concerning colonialism. However, the problem arises in Game mode. Here, the icon of the city in question is displayed, but it isn't possible to interact with it using the cursor since it is outside the borders of the country to which it belongs.

- Country: this time, passing the mouse over the Scene View, a flashing square appears, indicating the activation of the Create tool to produce borders. Now, by clicking with the left mouse button at the position where the editor wants to insert the first point of the country's border, it is possible to follow the outline of the country to be created. It is of paramount importance that at a minimum, the boundary consists of at least five points.

This mode presents the following hotkeys:

- * Shift + C = Close polygon (only if there are more than five vertices);
- * Shift + X = remove the last point;
- * Shift + S = snap to the nearest vertex;
- * Esc = clear all.

- Country Region: after selecting a created country it is possible to add a region outside of it. This tool is useful to make the borders of the islands or pieces of land disconnected, related to a specific country. The new region created will be added to the country's section and the way it is created is the same as in the main country.
- Province and Province Region: These tools have never been utilized as they pertain to more detailed subdivision levels within a country. The application, given its intended functionality, demands a high level of detail for international borders but does not require the same level of detail for internal country boundaries.

- **Mount Point:** Mount Points are editor-defined markers on the map created in the Map Editor which indicate a special location that includes a name, a class identifier (a number), and a collection of tags. Mount Points are useful for adding strategic locations, like airports, military units, resources, and other landmarks useful for the application.
- **Event:** as is the case with cities, events also by passing the mouse over the Scene View, a flashing dot appears which indicates the activation of the Create tool. Then, by clicking the left mouse button in the position where the editor wants to insert the new event, it will be immediately created and related to the nearest city. By default, the event is already populated with some graphical and non-graphical elements related to the nearest event in the same country to make it already visible in the scene, so it can be easily edited.

Again, the event does not have to be placed within the country's borders. However, this leads to a problem in Game mode, where the event icon is visible, but not interacting. For this, it is still necessary to surround it with a boundary that makes it active.

- **Revert**

Delete all new updates, that have not been made definitive using the Save button. Once the Revert button is clicked, the question "Discard current changes?" will appear below with two options, "Discard" and "Cancel". This will limit possible wrong clicks.

- **Save**

This button is fundamental to saving all the changes applied by updating each element in the section related to the World Map Editor script. Once the Save button is clicked, the question "Save changes?" will appear below with two options, Save and Cancel. This limits possible erroneous clicks, especially if incorrect changes have been applied.

The Demo Script

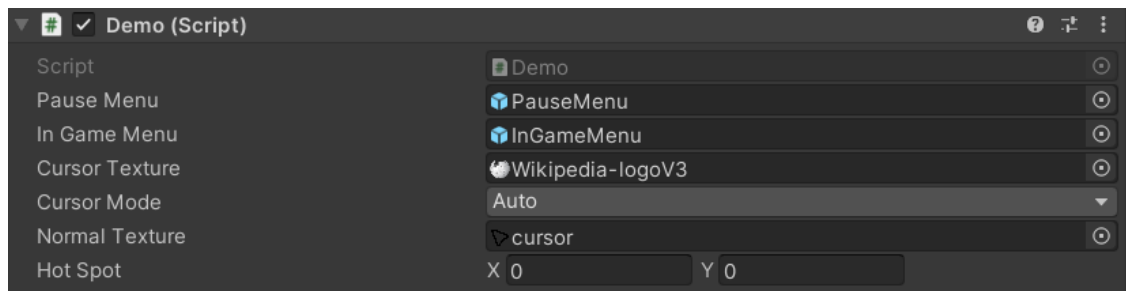


Figure 3.8: *The Demo Section*

This section (Figure 3.8) is shorter but considers the main script of the application as far as the graphical customization aspect is concerned, namely Demo.cs. Here the editor can pass to it the prefabs of the PauseMenu, but especially of the user interface that is the InGameMenu, which can in this way be edited separately to be inserted later within the Game mode. Also present are the two main cursor textures, namely the arrow and the Wikipedia logo, when it passes over an icon of an event.

3.3 Resources Folder

This folder is vital for the application's development, as it contains graphic elements and data related to countries, cities, and events. Graphic elements are divided between fonts, the Game Object materials, the shaders, the meshes, and the textures.

It is essential to emphasize this last aspect because, in addition to the textures related to the objects in the scene, such as the Earth and the Moon, there is an "Other" folder that consolidates all the textures added for the icons. Among these, there are all the textures of the different event categories, all named as "category-name", of the various types of cities and those that concern the appearance of the cursor.

Within the Resources folder, another crucial aspect pertains to the Geodata folder, which will be the focus of the next section.

3.4 Geodata Files

The data for categories, cities, and countries are automatically generated after being created using the Create tool present in the software on Unity (see 3.2.5) and are subsequently stored in files .txt, located in the WorldMapPoliticalMapGlobeEdition/Resources/Geodata folder.

Then they are subdivided based on the different scenes present in the application. So, at the moment, there are only three folders that represent "1901-1950", "1951-2000" and "2001-2021" scenes. In each of these, there are several .txt files; each file must be renamed according to the folder in which they are. Thus, where dates are present that determine the historical period of the scene, the editor must adapt them as with the folder name.

Specifically, the main .txt files have the following format:

- **categories_scene_LANGUAGE.txt**: data for all the categories of events.
- **cities_scene_LANGUAGE.txt**: data for cities.
- **countries10_scene_LANGUAGE.txt**: data for country frontiers in high resolution.
- **countries110_scene_LANGUAGE.txt**: data for country frontiers in low resolution.

Where scenes correspond to the name of the scene under consideration and LANGUAGE for now can only be English or Italian, although, those in the Italian language for the moment have not yet been modified.

All data is packed in string fields using separators, that are |, \$, *, comma, and semicolon. In general, the first separates the elements as a whole, while the others are used within each one to divide the main characteristics that make up the basic data.

3.4.1 Country File Format

Contains a list of countries separated by |, for example: *Country0|Country1|Country2*

Each Country entry is a series of fields separated by \$:

Name \$ Continent \$ REGIONS \$ Domain (optional) \$ HIDDEN \$ FIPS 10.4 CODE \$ ISO A2 \$ ISO A3 \$ ISO N3 \$ LABEL VISIBILITY

Name = name of the country, as displayed on the map.

Continent = continent name, useful to group countries.

REGIONS is a list of polygons, separated by *: REGION0*REGION1*REGION2.

This parameter corresponds to the regions that make up a country and refers to continuous land areas within it. Each REGION is a list of latitude/longitude coordinates in the format like

lat0, lon0; lat1, lon1; lat2, lon 2. These are clearly the positions of the various points that manage the country's borders.

For certain countries, an indicator has also been added immediately after the REGIONS section regarding the Domain. This indicator is only present if the considered country is a colony or a colonizer. Its presence allows for easy identification to highlight the associated colonial empire.

FIPS / ISO codes = standard ID codes for the country.

LABEL VISIBILITY = 0 (visible) or 1 (invisible).

It is important to note that in WPMCountries.cs there's a method called SetCountryGeoData(), which is responsible for loading and extracting this file data.

3.4.2 City File Format

The city .txt file contains a list of cities separated by |, for example: *CITY0|CITY1|CITY2*.

Each CITY entry is a series of fields separated by \$:

NAME \$ PROVINCE_NAME \$ COUNTRY_NAME \$ POPULATION \$ X \$ Y \$ Z \$ CLASS

Name = name of the city.

Province_Name = the name of the province in which is located.

Country_Name = the name of the country to which the city belongs to.

X, Y, and Z are the sphere coordinates of the city.

Population = metropolitan population, that can be useful to filter the biggest city.

Class = 1 corresponds to the normal city, 2 to the region capital, and 4 to the country capital.

Also in this case it is useful to underline that the method SetCityGeoData() in WPMCities.cs is responsible for loading and extracting the file data.

3.4.3 Categories File Format

As regards the categories, in the dedicated folder there are only two .txt files, one for the English version, and the other for the Italian one. Each event is separated from the others by a |, while inside it is composed as follows:

Name \$ Country \$ Year \$ X \$ Y \$ Z \$ Class \$ URL \$ Tag (optional)

Name: the title of the event.

Country: the country in which the event is located.

Year: when the event happened, for now, it's possible only to put a whole number.

X, Y, and Z = local position on the x-axis, y-axis, and z-axis.

Class = class to which the event belongs translated as integer, so BATTLE corresponds to 0, ART to 3, DISCOVERY to 5, SCIENCE to 7, TREATY to 9, FUNFACT to 11, HUMAN RIGHTS to 13, TRAGEDY to 15, and EVENT to 17.

URL = usually a Wikipedia link to the event.

For certain categories, an indicator has also been added immediately after the URL section regarding the Tag. This indicator is only present if the considered category is related to an important subcategory of events, for example, that related to one of the World Wars.

In the WMPCategories.cs file there is the method that populates the aforementioned .txt file, namely SetCategoryGeoData().

Chapter 4

Developer Manual

In this chapter, reference will be made to the three pivotal figures crucial for the development of the application: the editor, the programmer, and the user. The editor is primarily engaged with the Unity interface, as previously detailed in the preceding chapter (Section 3.2). The programmer assumes the responsibility of managing the code within JetBrains Rider or any Integrated Development Environments (IDEs), ensuring alignment with the requirements of the other two figures. Finally, the user plays a vital role in testing the application's functionality, identifying issues, recognizing shortcomings, and suggesting potential enhancements.

The contributions to this application made by the author revolve around amplifying interactivity, incorporating extensive historical material, and introducing new educational and informative features. In this multifaceted role, the author personified all three figures. In the capacity of an editor, a profound understanding of the Unity application was acquired, as extensively discussed previously; however, in this chapter, a more personal exploration of this work will be presented. From the perspective of a programmer, the author delved into the intricacies of various scripts, introducing diverse lines of code through a process facilitated by reverse engineering work.

Finally, in the user role, emphasis was placed on testing and observing outcomes in Game mode within Unity, with the aim of discerning necessary improvements.

Summing up in concrete terms the main activities carried out are:

1. **Add a new Scene.**
2. **Create a new event type.**
3. **Gui development.**
4. **Editing of a Scene.**

4.1 Add a New Scene

Emphasizing this procedure is crucial, as it is anticipated that future developers of this application will frequently need to perform it. Presently, the project encompasses only three scenes for the twentieth and twenty-first centuries. However, for the project to become truly engaging and valuable for historical studies, additional scenes should be incorporated to represent various historical periods. This expansion may also involve refining transitions between scenes, particularly to highlight the frequent changes in territorial borders within the covered time frame.

The creation of a new scene in this project and its integration with existing ones involves several steps. Initially, it is necessary to duplicate an existing well-defined scene and place it in the "Assets/WorldPoliticalMapGlobeEdition/Demos/01 GeneralDemo" folder, renaming it based

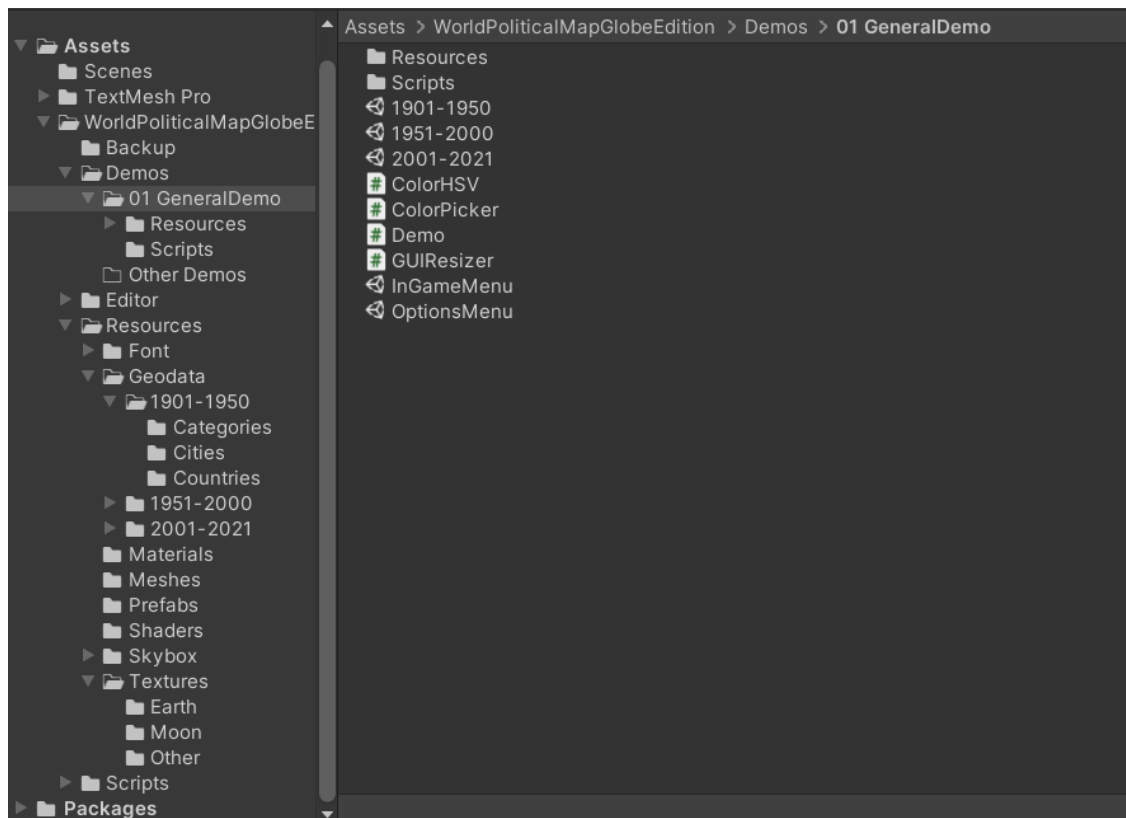


Figure 4.1: "01 GeneralDemo" folder from Project window (Section 3.2.2)

on the historical period it represents. For simplicity's sake, the chosen starting point is a scene that depicts the later historical period, aligning with the project's progression from the present to the more distant past.

Following this, it is imperative to duplicate the folder structure containing scene data, encompassing countries, cities, and events. This duplicated folder should bear the same name as the scene and be located in "Assets/WorldPoliticalMapGlobeEdition/Resources/Geodata". Notably, copying the .txt files inside the data folder related to countries and cities is essential to establish a foundational structure for the new scene. These files can be subsequently modified to accommodate necessary changes, especially concerning countries. It is recommended to leave the .txt file related to cities mostly unaltered, as their historical evolution over the centuries is complex and not explicitly beneficial for the application's objectives. Cities primarily serve as current reference points to enhance the understanding of where historical events occurred. For the categories, it is advisable to create an empty .txt file, intending to automatically populate it later through the Create tool. Ensuring that these files are renamed using the established format, as explained in Section 3.4, is crucial.

Upon completion of these steps, the new scene exists and is already populated with cities and borders. To make it accessible from other scenes in the Game View, the editor should modify the slider in the Hierarchy window, navigating through the path "InGameMenu/InGameMenu-Canvas/SliderEmpty/Panel/Slider". By clicking on it and inspecting the Inspector, the Unity UI object with editable features can be found, as illustrated in Figure 4.2.

In this case, it is really important the Slider object's maximum value, that needs to be adjusted.

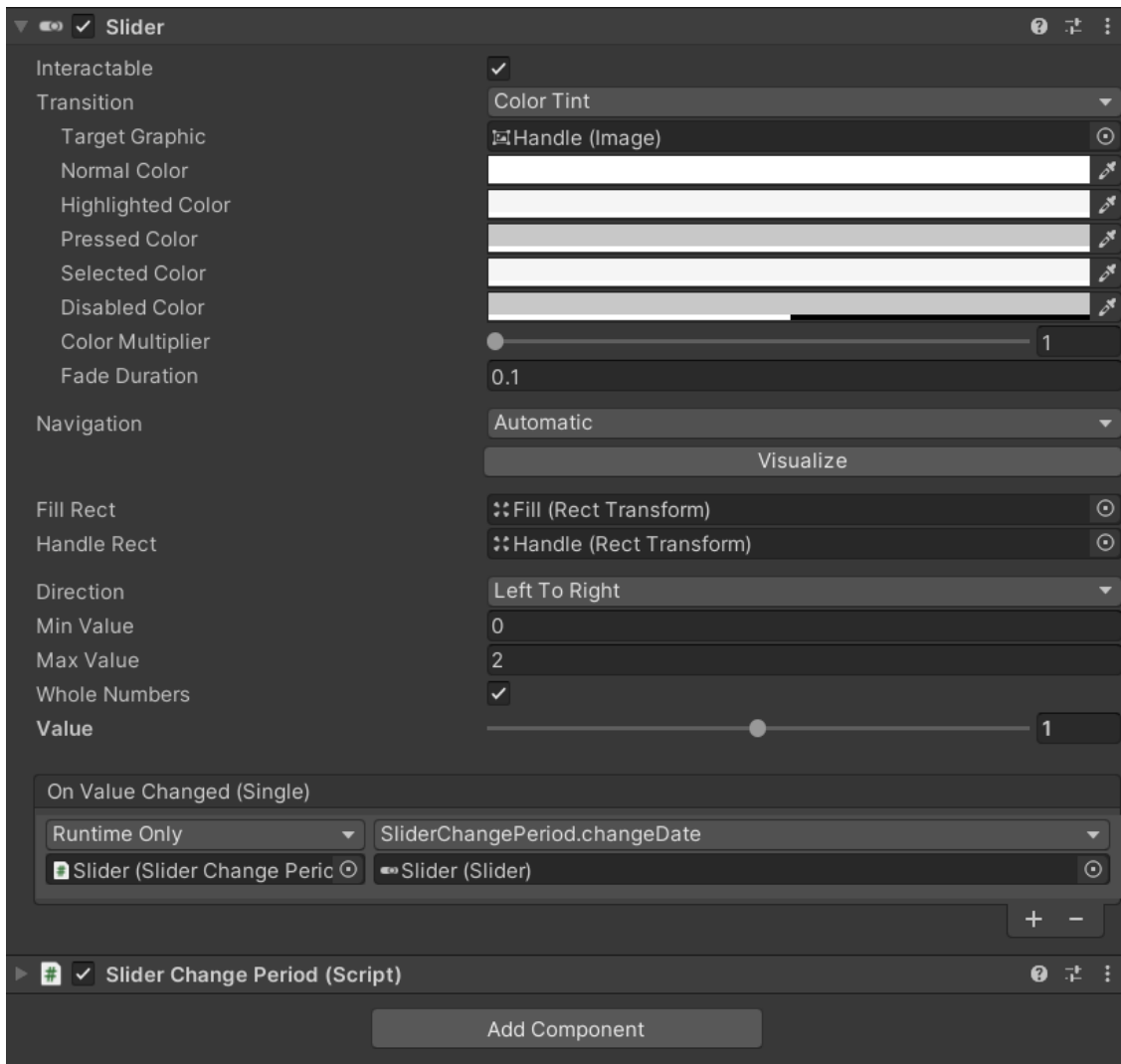


Figure 4.2: UI Slider in Hierarchy window

Therefore, each time a new scene is added, this value needs to be increased by one. Another key aspect to emphasize, it is what script is passed to it in the On Value Changed function, a specific characteristic of the Slider object. This is associated with the "SliderChangePeriod" script that contains a function called "changeDate()". As can be seen in Figure 4.3, this method takes the object Slider to know what its current value is. Then, having taken the current value, the function, via an appropriate switch, loads the scene corresponding to it.

```
private List<string> options =
new List<string>() {"1901-1950", "1951-2000", "2001-2021"};

public void changeDate(Slider slider){

    int numericSliderValue = (int) slider.value;

    if (activeScene != options[numericSliderValue])
    {
        switch (options[numericSliderValue])
        {
            case "1901-1950":
                textComponent.text = options[numericSliderValue];
                SceneManager.LoadScene("1901-1950");
                break;
            case "1951-2000":
                textComponent.text = options[numericSliderValue];
                SceneManager.LoadScene("1951-2000");
                break;
            case "2001-2021":
                textComponent.text = options[numericSliderValue];
                SceneManager.LoadScene("2001-2021");
                break;
        }
    }
}
```

Figure 4.3: *changeDate(Slider slider) function*

4.2 Create a New Event Type

The creation of a new event type involves predominantly adding graphical elements in Unity for the editor, while the programmer is tasked with implementing code in C# to manage its instantiation and developments. This collaborative effort ensures a comprehensive and well-organized representation of various event types within the application.

What the editor must do is:

1. Search on the internet or create a 64x64 RGBA image texture 2D named "category-name" (for example "category-tragedy") and put it in the folder "Assets/WorldPoliticalMap-GlobeEdition/Resources/Textures/Other". If downloaded from the internet, it is necessary

that it has a free Creative Commons license for any use, or if necessary, mention the author by copying and pasting it when the application is distributed.

2. Create a corresponding material labeled "CategoryName" (for example "CategoryTragedy") and settle it in the folder "Assets/WorldPoliticalMapGlobeEdition/Resources/Materials". To create a new material the editor has to right-click in the "Materials" folder, selecting "Create" and then choosing "Material". Then, after selecting the new material, in the Inspector window, there will be an empty square to the right of the texture section with a 'Select' button at the bottom right. Pressing this opens a window where the previously loaded texture can be selected to assign it to the material.
3. Add a prefab asset named "CategorSpotName" (for example "CategorySpotTragedy") in the folder "Assets/WorldPoliticalMapGlobeEdition/Resources/Prefabs". To do it, it is necessary to right-click, then select "Create" and choose "Prefab". After that, two components must be added to the basic version of the prefab to make it ready to become an icon. At the bottom, there is an "Add Component" button. After clicking it, the editor needs first to select the Mesh Filter to add it to the Inspector, and then assign the Mesh Quad in the corresponding field. That is a square where it will be possible to assign the material and the texture corresponding to the new type of event. Once that's done, the editor needs to add the Mesh Renderer component on which it's possible to assign the corresponding material.
4. At the top right of the Unity interface there is a dropdown menu that says "Layer". In Unity, layers are a way to categorize and group objects in a scene and they are primarily used for controlling the visibility and interaction of objects in the scene, especially with regards to cameras and collision detection. In this case, it is necessary to select the "Edit layers" item at the bottom and then add the layer corresponding to the new category, naming it for example "TragedyLayer" for the Tragedy category.
5. Create a new toggle to activate and deactivate the events corresponding to the new category created during Game mode. In this context, attention should be directed to [Section 4.3.1](#) elucidating the procedure for incorporating a new Toggle.

This instead is the list of scripts that the programmer has to edit to add a new event type:

- **FilterMenuToggles.cs.**
- **WorldMapGlobeCategories.cs.**
- **WPMCategories.cs.**
- **Menu.cs.**
- **WPMInternal.cs.**
- **Category.cs.**

4.2.1 FilterMenuToggles.cs

These lines of code ([Figure 4.4](#)) govern the visibility of all events' icons through the InGameMenu. To control the display of the new event type created, this function must be related to the On Value Changed option inside the corresponding Toggle object, selecting then the associated switch case ([Figure 4.13](#)).

```
public void TogglesCategories(Toggle t)
{
    map = WorldMapGlobe.instance;

    switch (t.name)
    {
        case "BATTLE":
            map.showBattles = t.isOn;
            break;
        case "ART":
            map.showArts = t.isOn;
            break;
        case "DISCOVERY":
            map.showDiscoveries = t.isOn;
            break;
        case "SCIENCE":
            map.showSciences = t.isOn;
            break;
        case "TREATY":
            map.showTreaties = t.isOn;
            break;
        case "FUN FACT":
            map.showFunFact = t.isOn;
            break;
        case "HUMAN RIGHTS":
            map.showHumanRights = t.isOn;
            break;
        case "TRAGEDY":
            map.showTragedy = t.isOn;
            break;
        case "EVENT":
            map.showEvent = t.isOn;
            break;
    }
}
```

Figure 4.4: *TogglesCategories(Toggle t) method*

4.2.2 WorldMapGlobeCategories.cs

In this script, there are methods such as `showEvent()` (Figure 4.5), where the category name should be entered instead of `Event`. These functions are the triggers that make the category icons visible, in fact, they are used on `Toggles`' property `"isOn"`, as evident in Figure 4.4.

```
[SerializeField]
bool
    _showBattles = true;

public bool showBattles {
get {
    return _showBattles;
}
set {
    if (_showBattles != value) {
        _showBattles = value;
        isDirty = true;
        if (battleLayer != null) {
            //battleLayer.SetActive(_showBattles);
            battleLayer.SetActive(!battleLayer.activeSelf);
        }
    }
}
}
```

```
[SerializeField]
bool
    _showArts = true;

public bool showArts {
get {
    return _showArts;
}
set {
    if (_showArts != value) {
        _showArts = value;
        isDirty = true;
        if (artLayer != null) {
            artLayer.SetActive(_showArts);
        }
    }
}
}
```

```
[SerializeField]
bool
    _showDiscoveries = true;

public bool showDiscoveries {
get {
    return _showDiscoveries;
}
set {
    if (_showDiscoveries != value) {
        _showDiscoveries = value;
    }
}
```

```
        isDirty = true;
        if (discoveryLayer != null) {
            discoveryLayer.SetActive(_showDiscoveries);
        }
    }
}
```

```
[SerializeField]
bool
    _showSciences = true;

public bool showSciences {
get {
    return _showSciences;
}
set {
    if ( _showSciences!= value) {
        _showSciences = value;
        isDirty = true;
        if (scienceLayer != null) {
            scienceLayer.SetActive(_showSciences);
        }
    }
}
}
```

```
[SerializeField]
bool
    _showTreaties = true;

public bool showTreaties {
get {
    return _showTreaties;
}
set {
    if ( _showTreaties!= value) {
        _showTreaties = value;
        isDirty = true;
        if (treatyLayer != null) {
            treatyLayer.SetActive(_showTreaties);
        }
    }
}
}
```

```
[SerializeField]
bool
    _showHumanRights = true;
```

```
public bool showHumanRights {
get {
    return _showHumanRights;
}
set {
    if ( _showHumanRights!= value) {
        _showHumanRights = value;
        isDirty = true;
        if (humanRightsLayer != null) {
            humanRightsLayer.SetActive(_showHumanRights);
        }
    }
}
}
```

```
[SerializeField]
```

```
bool
```

```
    _showFunFact = true;
```

```
public bool showFunFact {
get {
    return _showFunFact;
}
set {
    if ( _showFunFact!= value) {
        _showFunFact = value;
        isDirty = true;
        if (funFactLayer != null) {
            funFactLayer.SetActive(_showFunFact);
        }
    }
}
}
```

```
[SerializeField]
```

```
bool
```

```
    _showTragedy = true;
```

```
public bool showTragedy {
get {
    return _showTragedy;
}
set {
    if ( _showTragedy!= value) {
        _showTragedy = value;
        isDirty = true;
        if (tragedyLayer != null) {
            tragedyLayer.SetActive(_showTragedy);
        }
    }
}
```

```
    }  
  }  
}  
  
[SerializeField]  
bool  
    _showEvent = true;  
public bool showEvent {  
  get {  
    return _showEvent;  
  }  
  set {  
    if ( _showEvent!= value) {  
      _showEvent = value;  
      isDirty = true;  
      if (eventLayer != null) {  
        eventLayer.SetActive(_showEvent);  
      }  
    }  
  }  
}  
}
```

Figure 4.5: Methods that show different categories of events

4.2.3 WPMCategories.cs

First, in this script, the variables that represent materials and rendering objects associated are declared (Figure 4.6). These variables are associated with different layers and specific spots for various categories.

```
Material categoriesBattleMat, categoriesArtMat, categoriesScienceMat,  
    categoriesTreatyMat, categoriesDiscoveryMat, categoriesFunFactMat,  
    categoriesHumanRightsMat, categoriesTragedyMat, categoriesEventMat;  
  
GameObject categoriesLayer, artLayer, battleLayer, discoveryLayer,  
    humanRightsLayer, scienceLayer, treatyLayer, funFactLayer, tragedyLayer,  
    eventLayer, categorySpotBattle, categorySpotArt, categorySpotScience,  
    categorySpotTreaty, categorySpotFunFact,  
    categorySpotDiscovery, categorySpotHumanRights, categorySpotTragedy,  
    categorySpotEvent;
```

Figure 4.6: Declaration of materials, layers and spot

In summary, this code is responsible for managing the various layers (Figure 4.7). It first checks if an existing one is present, and if so, it is destroyed. Then, a new one is created, configured with specific flags, set as a child of the "categoriesLayer," and assigned the same layer as the current GameObject. This is often used for managing temporary or dynamically created objects within a scene, usually icons.


```
// Create categories ALL layer
Transform t = transform.Find("Categories");
if (t != null)
    DestroyImmediate(t.gameObject);
categoriesLayer = new GameObject("Categories");
categoriesLayer.transform.SetParent(transform, false);
categoriesLayer.layer = gameObject.layer;
if (_earthInvertedMode) {
    categoriesLayer.transform.localScale *= 0.99f;
}

// Create category class parents
//ART
t = transform.Find("ArtLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
artLayer = new GameObject("ArtLayer");
artLayer.hideFlags = HideFlags.DontSave;
artLayer.transform.SetParent(transform, false);
artLayer.layer = gameObject.layer;

//BATTLE
t = transform.Find("BattleLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
battleLayer = new GameObject("BattleLayer");
battleLayer.hideFlags = HideFlags.DontSave;
battleLayer.transform.SetParent(transform, false);
battleLayer.layer = gameObject.layer;

//DISCOVERY
t = transform.Find("DiscoveryLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
discoveryLayer = new GameObject("DiscoveryLayer");
discoveryLayer.hideFlags = HideFlags.DontSave;
discoveryLayer.transform.SetParent(categoriesLayer.transform, false);
discoveryLayer.layer = gameObject.layer;

//HUMAN RIGHTS
t = transform.Find("HumanRightsLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
humanRightsLayer = new GameObject("HumanRightsLayer");
humanRightsLayer.hideFlags = HideFlags.DontSave;
humanRightsLayer.transform.SetParent(categoriesLayer.transform, false);
humanRightsLayer.layer = gameObject.layer;

//SCIENCE
t = transform.Find("ScienceLayer");
```

```
if (t != null)
    DestroyImmediate(t.gameObject);
scienceLayer = new GameObject("ScienceLayer");
scienceLayer.hideFlags = HideFlags.DontSave;
scienceLayer.transform.SetParent(transform, false);
scienceLayer.layer = gameObject.layer;

//TREATY
t = transform.Find("TreatyLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
treatyLayer = new GameObject("TreatyLayer");
treatyLayer.hideFlags = HideFlags.DontSave;
treatyLayer.transform.SetParent(categoriesLayer.transform, false);
treatyLayer.layer = gameObject.layer;

//TRAGEDY
t = transform.Find("TragedyLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
tragedyLayer = new GameObject("TragedyLayer");
tragedyLayer.hideFlags = HideFlags.DontSave;
tragedyLayer.transform.SetParent(categoriesLayer.transform, false);
tragedyLayer.layer = gameObject.layer;

//FUN FACT
t = transform.Find("FunFactLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
funFactLayer = new GameObject("FunFactLayer");
funFactLayer.hideFlags = HideFlags.DontSave;
funFactLayer.transform.SetParent(categoriesLayer.transform, false);
funFactLayer.layer = gameObject.layer;

//EVENT
t = transform.Find("EventLayer");
if (t != null)
    DestroyImmediate(t.gameObject);
eventLayer = new GameObject("EventLayer");
eventLayer.hideFlags = HideFlags.DontSave;
eventLayer.transform.SetParent(categoriesLayer.transform, false);
eventLayer.layer = gameObject.layer;
```

Figure 4.7: Menage of layers for different categories

In the end, these last code lines (Figure 4.8) seem to handle the instantiation, configuration, and visibility of GameObjects categories based on which one "isShown".

```
int categoryCount = categories.Count;
int layer = gameObject.layer;
for (int k = 0; k < categoryCount; k++) {
    Category category = categories[k];
    Country country = countries[category.countryIndex];
    category.isShown = !country.hidden &&
        /*(((int)category.categoryClass &
            _categoryClassAlwaysShow) != 0) ||
            (string.Equals(minDescription, _minDescription) ||
                category.description.Length >= minDescription.Length));*/
        (((int)category.categoryClass) != 0) ||
            (string.Equals(minDescription, _minDescription) ||
                category.description.Length >= minDescription.Length));

    if (category.isShown)
    {
        GameObject categoryObj = null, categoryParent = null;

        switch (category.categoryClass)
        {
            case CATEGORY_CLASS.BATTLE:
                if(WorldMapGlobe.instance.showBattles)
                {
                    categoryObj = Instantiate(categorySpotBattle);
                    if (!combineMeshesActive)
                    {
                        categoryObj.GetComponent<Renderer>().sharedMaterial =
                            categoriesBattleMat;
                    }

                    categoryParent = battleLayer;
                }

                break;
            case CATEGORY_CLASS.ART:
                categoryObj = Instantiate(categorySpotArt);
                if (!combineMeshesActive)
                {
                    categoryObj.GetComponent<Renderer>().sharedMaterial =
                        categoriesArtMat;
                }

                categoryParent = artLayer;
                break;
            case CATEGORY_CLASS.SCIENCE:
                categoryObj = Instantiate(categorySpotScience);
                if (!combineMeshesActive)
                {
                    categoryObj.GetComponent<Renderer>().sharedMaterial =
                        categoriesScienceMat;
                }
            }
        }
    }
}
```

```
    }

    categoryParent = scienceLayer;
    break;
case CATEGORY_CLASS.TREATY:
    categoryObj = Instantiate(categorySpotTreaty);
    if (!combineMeshesActive)
    {
        categoryObj.GetComponent<Renderer>().sharedMaterial =
            categoriesTreatyMat;
    }

    categoryParent = treatyLayer;
    break;
case CATEGORY_CLASS.DISCOVERY:
    categoryObj = Instantiate(categorySpotDiscovery);
    if (!combineMeshesActive)
    {
        categoryObj.GetComponent<Renderer>().sharedMaterial =
            categoriesDiscoveryMat;
    }

    categoryParent = discoveryLayer;

    break;

case CATEGORY_CLASS.FUNFACT:
    categoryObj = Instantiate(categorySpotFunFact);
    if (!combineMeshesActive)
    {
        categoryObj.GetComponent<Renderer>().sharedMaterial =
            categoriesFunFactMat;
    }

    categoryParent = funFactLayer;
    break;

case CATEGORY_CLASS.HUMANRIGHTS:
    categoryObj = Instantiate(categorySpotHumanRights);
    if (!combineMeshesActive)
    {
        categoryObj.GetComponent<Renderer>().sharedMaterial =
            categoriesHumanRightsMat;
    }

    categoryParent = humanRightsLayer;

    break;

case CATEGORY_CLASS.TRAGEDY:
```

```
        categoryObj = Instantiate(categorySpotTragedy);
        if (!combineMeshesActive)
        {
            categoryObj.GetComponent<Renderer>().sharedMaterial =
                categoriesTragedyMat;
        }

        categoryParent = tragedyLayer;
        break;

    case CATEGORY_CLASS.EVENT:
        categoryObj = Instantiate(categorySpotEvent);
        if (!combineMeshesActive)
        {
            categoryObj.GetComponent<Renderer>().sharedMaterial =
                categoriesEventMat;
        }

        categoryParent = eventLayer;
        break;
}

categoryObj.layer = layer;
categoryObj.hideFlags = HideFlags.DontSave | HideFlags.HideInHierarchy;
categoryObj.transform.SetParent(categoryParent.transform, false);
categoryObj.transform.localPosition = category.localPosition;
categoryObj.transform.localScale = categoryScale;
if (!_earthInvertedMode)
{
    categoryObj.transform.LookAt(transform.TransformPoint(category.localPosition
        * 2f));
}
else
{
    categoryObj.transform.LookAt(transform.position);
}

category.gameObject = categoryObj;
_numCategoriesDrawn++;
visibleCount++;

}
else
{
    category.gameObject = null;
}
}

if (combineMeshesActive) {
    DestroyImmediate(categoryScaler);
}
```

```
        CombineCatMeshes(battleLayer, categoriesBattleMat);
        CombineCatMeshes(artLayer, categoriesArtMat);
        CombineCatMeshes(scienceLayer, categoriesScienceMat);
        CombineCatMeshes(treatyLayer, categoriesTreatyMat);
        CombineCatMeshes(discoveryLayer, categoriesDiscoveryMat);
        CombineCatMeshes(funFactLayer, categoriesFunFactMat);
        CombineCatMeshes(humanRightsLayer, categoriesHumanRightsMat);
        CombineCatMeshes(tragedyLayer, categoriesTragedyMat);
        CombineCatMeshes(eventLayer, categoriesEventMat);
    }
```

Figure 4.8: Menage of the GameObject related to different categories

4.2.4 Menu.cs

Inside this script, there is the `ChangeFilter` method (Figure 4.9). It is designed to toggle the visibility of specific layers based on a given filter string and a boolean value. Each case in the switch statement corresponds to a different filter string, and the associated layer's visibility is set to the specified boolean value.

```
public void ChangeFilter(string filter, bool value)
{
    switch (filter)
    {
        case "BATTLE":
            battleLayer.SetActive(value);
            break;
        case "ART":
            artLayer.SetActive(value);
            break;
        case "DISCOVERY":
            discoveryLayer.SetActive(value);
            break;
        case "SCIENCE":
            scienceLayer.SetActive(value);
            break;
        case "TREATY":
            treatyLayer.SetActive(value);
            break;
        case "FUN FACT":
            funFactLayer.SetActive(value);
            break;
        case "HUMAN RIGHTS":
            humanRightsLayer.SetActive(value);
            break;
        case "TRAGEDY":
            tragedyLayer.SetActive(value);
            break;
    }
}
```

```
        case "EVENT":
            eventLayer.SetActive(value);
            break;
    }
}
```

Figure 4.9: ChangeFilter(string filter, bool value) method

4.2.5 WPMInternal.cs

This script is accountable for two main things. First, the provided code segment (Figure 4.10) is responsible for loading `GameObject` prefabs from the "Resources" folder. These prefabs are associated with different categories.

```
//Categories Prefabs Loading
categorySpotBattle = Resources.Load<GameObject>("Prefabs/CategorySpotBattle");
categorySpotArt = Resources.Load<GameObject>("Prefabs/CategorySpotArt");
categorySpotDiscovery =
    Resources.Load<GameObject>("Prefabs/CategorySpotDiscovery");
categorySpotScience =
    Resources.Load<GameObject>("Prefabs/CategorySpotScience");
categorySpotTreaty = Resources.Load<GameObject>("Prefabs/CategorySpotTreaty");
categorySpotFunFact =
    Resources.Load<GameObject>("Prefabs/CategorySpotFunFact");
categorySpotHumanRights =
    Resources.Load<GameObject>("Prefabs/CategorySpotHumanRights");
categorySpotTragedy =
    Resources.Load<GameObject>("Prefabs/CategorySpotTragedy");
categorySpotEvent = Resources.Load<GameObject>("Prefabs/CategorySpotEvent");
```

Figure 4.10: Prefabs loading

Second, these other code lines (Figure 4.11) are responsible for loading Material assets from the "Resources" folder in Unity. They instantiate the materials and assign them to various Material variables associated with different categories.

```
//Materials loading
categoriesBattleMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryBattle"));
categoriesBattleMat.name = "CategoriesBattle";
categoriesArtMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryArt"));
categoriesArtMat.name = "CategoriesArt";
categoriesDiscoveryMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryDiscovery"));
categoriesDiscoveryMat.name = "CategoriesDiscovery";
```

```
categoriesScienceMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryScience"));
categoriesScienceMat.name = "CategoriesScience";
categoriesTreatyMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryTreaty"));
categoriesTreatyMat.name = "CategoriesTreaty";
categoriesFunFactMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryFunFact"));
categoriesFunFactMat.name = "CategoriesFunFact";
categoriesHumanRightsMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryHumanRights"));
categoriesHumanRightsMat.name = "CategoriesHumanRights";
categoriesTragedyMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryTragedy"));
categoriesTragedyMat.name = "CategoriesTragedy";
categoriesEventMat =
    Instantiate(Resources.Load<Material>("Materials/CategoryEvent"));
categoriesEventMat.name = "CategoriesEvent";
```

Figure 4.11: Materials loading

4.2.6 Category.cs

In this script, methods are defining the structure of a generic event. To add a new event type, simply assign a numerical reference to the new event type in the CATEGORY_CLASS enum, incrementing the corresponding number by 2 (Figure 4.12). It is worth noting that the code lines in this script provide insight into customizing an event interface in Unity. For instance, I've included the 'tag' string within the Category constructor.

```
public enum CATEGORY_CLASS {
    ALL = -1,
    BATTLE = 0,
    ART = 3,
    DISCOVERY = 5,
    SCIENCE = 7,
    TREATY = 9,
    FUNFACT = 11,
    HUMANRIGHTS = 13,
    TRAGEDY = 15,
    EVENT = 17
}
```

Figure 4.12: *The enum that associates an integer number with each type of event*

4.3 GUI Development

The objective of the GUI development was to enhance the interface without a complete overhaul, maintaining its simplicity. The key tasks included:

- Incorporating a dropdown menu to represent major colonial empires of the twentieth century. Selection of one from the list would result in the coloring of all its colonies and the country itself on the map.
- Introducing new toggles for the event types that were added to the application.
- Inserting buttons in the "1901-1950" Scene to color the two sides of the World Wars, and in the "1951-2000" scene for the first European Union member countries.

The main reference elements for modifying the graphical interface of this application are "InGameMenu" (prefab) and "FilterMenuToggles.cs" (script). These components serve as the foundation for implementing the desired changes and improving the overall user experience.

4.3.1 Add a New Toggle

A secondary but fundamentally important aspect of user interaction involves the creation of a new toggle. Primarily, new toggles needed to be inserted to connect them to the visualization of the new event categories. Additionally, exploring their functionality for other purposes, such as initially deactivating the toggles managing the display of Region Capitals and Normal Cities, proved to be useful.

To add a toggle in Unity, the editor must first create a canvas. In Unity, Canvas is a component of the UI system that acts as a container for all UI elements in a scene. The Canvas is used to define a plane where UI elements like buttons, text, images, and other controls are rendered. The UI elements within a Canvas are then positioned and organized based on pixel coordinates, providing a way to create and manage the user interface of a game or application. In this context, for the toggles related to the categories within the InGameMenu prefab, the PanelCategoriesToggles canvas serves as the container. To add a new toggle inside it, the editor can simply right-click on it, select the UI item, and then choose Toggle. So the section in Figure 4.13 is therefore added inside the Inspector. Subsequently, customization is possible by adjusting the label, text size, color, font, and the selectable toggle box. Notably, the `isOn` toggle, if selected, makes it active by default.

The next step involves adding a function to characterize the toggle. To connect it to the object, after selecting it, the editor can drag the `.cs` file containing the lines of code managing the Boolean value of the toggle onto the Inspector, in this case, "FilterMenuToggles". Following this, in the "On Value Change" section of the toggle, it becomes possible to first select the name of the inserted script, and then indicate the exact method intended for managing the Boolean. This process ensures that the toggle functions as intended within the user interface.

4.3.2 Colorize Countries

The incorporation of an existing function from the initial software that randomly colors a country upon button click was integral to the development process. This functionality underwent modification and adaptation to enhance the user experience across various historical contexts, as detailed in the following cases.

Within the "1951-2000" Scene, a button was integrated into the User Interface to facilitate the colorization of the members of the European Union in 1995, marking its inception. Figure 4.14 illustrates the code lines responsible for coloring the member countries of the European

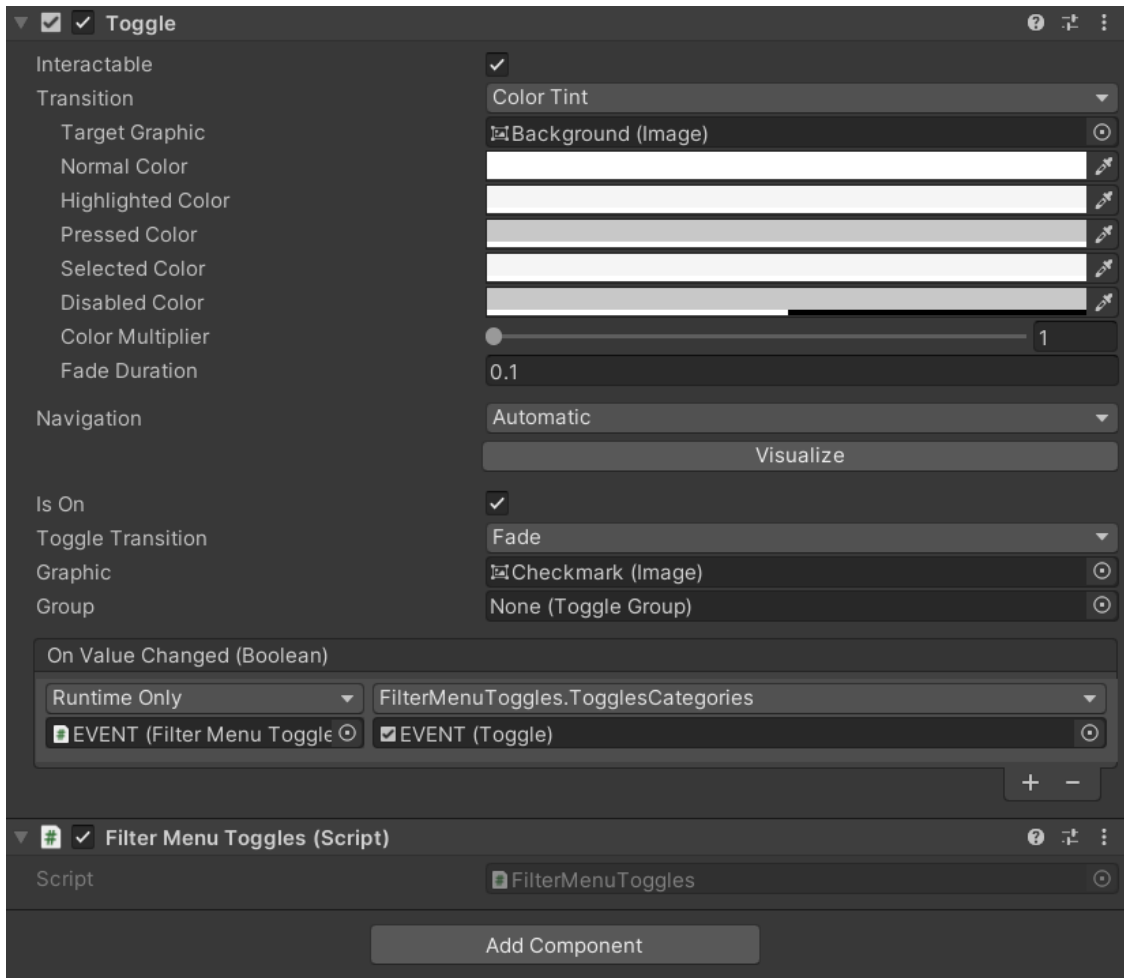


Figure 4.13: UI Toggle element in Inspector window

Union specifically within the context of the "1951-2000" scene. It's crucial to note that applying this method in other scenarios may result in the coloring of European countries as a whole.

```
public void ColorizeEUButton()
{
    if (activeScene.Equals("1951-2000"))
    {
        map.HideCountrySurfaces();
        map = WorldMapGlobe.instance;
        map.FlyToCountry("Italy");
        for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
            colorizeIndex++)
        {
            if (map.countries[colorizeIndex].continent.Equals("EU"))
            {
                Color color = new Color(0, 0, 1);
```

```

        map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
            color);
    }
}
else
{
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Italy");
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++)
    {
        if (map.countries[colorizeIndex].continent.Equals("Europe"))
        {
            Color color = new Color(UnityEngine.Random.Range(0.0f, 1.0f),
                UnityEngine.Random.Range(0.0f, 1.0f),
                UnityEngine.Random.Range(0.0f, 1.0f));
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                color);
        }
    }
}
}}
```

Figure 4.14: *Function to highlight European Union in 1995*

Instead, in the "1901-1950" Scene, two distinct buttons were introduced. Each button was designated to colorize the sides of the First and Second World Wars with specific colors. In the first scenario, countries belonging to the Allied Powers were highlighted in blue, while those aligned with the Central Powers were colored in red (refer to Figure 4.16). In the second scenario, the Allies were represented in blue, and the Axis countries in red.

```

public void ColorizeWW1(){

    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Italy");
    text.textChanger1();
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++) {
        if(map.countries[colorizeIndex].name.Equals("France")
        ||map.countries[colorizeIndex].name.Equals("United
            Kingdom")||map.countries[colorizeIndex].name.Equals("Russian Empire")
            ||map.countries[colorizeIndex].name.Equals("Japanese
            Empire")||map.countries[colorizeIndex].name.Equals("Italy")
        ||map.countries[colorizeIndex].name.Equals("United States of America")) {

            Color color = new Color(0, 0, 255);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true, color);
        }
    }
}
```

```

        } else if
            (map.countries[colorizeIndex].name.Equals("German Empire")
            ||map.countries[colorizeIndex].name.Equals("Austria-Hungary")
            ||map.countries[colorizeIndex].name.Equals("Ottoman Empire")
            ||map.countries[colorizeIndex].name.Equals("Bulgaria")) {
            Color color = new Color(255, 0, 0);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
            color);
        }
    }
}
public void ColorizeWW2()
{
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Italy");
    text.textChanger2();
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++) {
        if (map.countries[colorizeIndex].name.Equals("France")
        ||map.countries[colorizeIndex].name.Equals("United Kingdom")
        ||map.countries[colorizeIndex].name.Equals("Russian Empire") ||
            map.countries[colorizeIndex].name.Equals("United States of
            America")) {
            Color color = new Color(0, 0, 255);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
            color);
        } else if (map.countries[colorizeIndex].name.Equals("German
            Empire")||map.countries[colorizeIndex].name.Equals("Italy")
            ||map.countries[colorizeIndex].name.Equals("Japanese Empire")) {
            Color color = new Color(255, 0, 0);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
            color);
        }
    }
}
}

```

Figure 4.15: *Methods to highlight the deployments of the First and the Second World War*

In the desire to incorporate the lines of code handling the "WW1" and "WW2" buttons into Figure 4.15, the initial action involves concealing all previously colored elements using the `HideCountrySurfaces()` method.

The unique feature of these buttons is that, in addition to coloring the countries, they also modify the text displayed on the side of the two red and blue squares below, so different text will appear based on the button pressed.

This particular property attached to the button click was added thanks to the `textChanger1()` and `textChanger2()` methods created in a special `Change_text.cs` file, as can be seen in Figure 4.17.

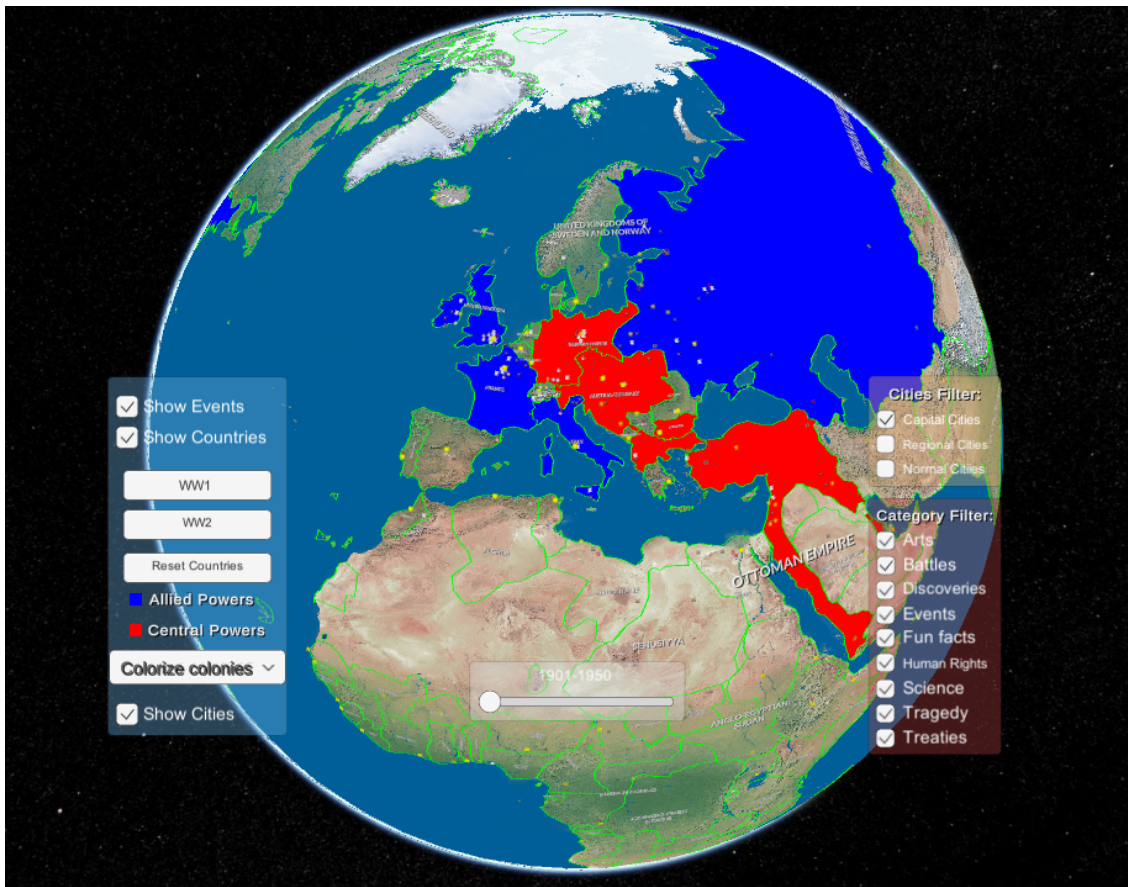


Figure 4.16: WW1 sides colorized in Game mode

```
using UnityEngine;
using TMPro;

public class Change_text : MonoBehaviour
{
    public TMP_Text alliesText;

    public TMP_Text enemiesText;

    static Change_text _instance;

    public static Change_text instance {
        get {
            if (_instance == null) {
                _instance = FindObjectOfType<Change_text> ();
                if (_instance == null) {
                    Debug.LogWarning ("'Change_text' GameObject could not be found
                    in the scene. Make sure it is created with this name
                    before using any map functionality.");
                }
            }
            return _instance;
        }
    }

    void Start()
    {
        alliesText.text = "";
        enemiesText.text = "";
    }

    public void textChanger1()
    {
        alliesText.text = "Allied Powers";
        enemiesText.text = "Central Powers";
    }

    public void textChanger2()
    {
        alliesText.text = "Allies";
        enemiesText.text = "Axes";
    }
}
```

Figure 4.17: *Script to change a text*

4.3.3 A Dropdown to Colorize Colonies

In addressing the division of colonies among various countries in the twentieth century, the identified and implemented graphic element to efficiently display a single country's colonies was the dropdown button. Each list element within the dropdown was linked to a distinct switch case. Within each case, the code was structured to undo previous alterations, reposition the camera to center on the selected country and apply a reference color derived from its flag to both the country and its colonies.

This feature of the application is frequently employed to outline the different colonial empires of the 20th century and to accommodate the substantial shifts in state names that occurred during this period, particularly in Africa and Asia.

```
public void HandleInputData(int index){
    switch (index)
    {
    case 0:
        map.HideCountrySurfaces();
        break;

    case 1:
        map.HideCountrySurfaces();
        map = WorldMapGlobe.instance;
        map.FlyToCountry("France");
        for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
            colorizeIndex++)
        {
            if (map.countries[colorizeIndex].domain.Equals("France"))
            {
                Color color = new Color(0, (float)0.45, (float)0.73);
                map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                    color);
            }
        }
        break;

    case 2:
        map.HideCountrySurfaces();
        map = WorldMapGlobe.instance;
        map.FlyToCountry("United Kingdom");
        for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
            colorizeIndex++)
        {
            if (map.countries[colorizeIndex].domain.Equals("Uk"))
            {
                Color color = new Color((float)0.784, (float)0.063, (float)0.18);
                map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                    color);
            }
        }
        break;
    }
```

```
case 3:
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Spain");
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++)
    {
        if (map.countries[colorizeIndex].domain.Equals("Spain"))
        {
            Color color = new Color((float)0.902, 0, (float)0.15);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                color);
        }
    }
    break;

case 4:
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Portugal");
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++)
    {
        if (map.countries[colorizeIndex].domain.Equals("Portugal"))
        {
            Color color = new Color(1, 0, 0);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                color);
        }
    }
    break;

case 5:
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Belgium");
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++)
    {
        if (map.countries[colorizeIndex].domain.Equals("Belgium"))
        {
            Color color = new Color(1, (float)0.804, 0);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                color);
        }
    }
    break;

case 6:
```



```
map.HideCountrySurfaces();
map = WorldMapGlobe.instance;
map.FlyToCountry("Germany Empire");
for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
    colorizeIndex++)
{
    if (map.countries[colorizeIndex].domain.Equals("Germany"))
    {
        Color color = new Color(0, 0, 0);
        map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
            color);
    }
}
break;

case 7:
    map.HideCountrySurfaces();
    map = WorldMapGlobe.instance;
    map.FlyToCountry("Italy");
    for (int colorizeIndex = 0; colorizeIndex < map.countries.Length;
        colorizeIndex++)
    {
        if (map.countries[colorizeIndex].domain.Equals("Italy"))
        {
            Color color = new Color(0, (float)0.55, (float)0.27);
            map.ToggleCountrySurface(map.countries[colorizeIndex].name, true,
                color);
        }
    }
    break;
}
}
```

Figure 4.18: *Function that handles a dropdown button and colors the different colonial empires*

4.4 Editing of a Scene

Given a clearer idea of the more technical aspects, much of the work was manual and mechanical. The process involved editing the scene, including changing country boundaries, adding historical events, and introducing new countries (see Section 3.2.5 and 3.2.5 for the tools), which was time-consuming. Detailed attention was given to border management, especially for regions involving multiple countries. Additionally, smaller states, especially islands, were added, even if they were not present in the original version. This inclusion was deemed necessary due to their historical significance or geographical relevance. Examples include Malta, Cape Verde, French Guiana, Singapore, Macau, Micronesia, and others.

Another critical aspect of editing involved the addition of events, which was a more mechanical task after the process of searching and selecting them. Typically, information within the fields

comprising the Event element was copied, edited if necessary, and pasted from the Wikipedia page. Additionally, the URL for linking was extracted from the same source.

Inserting events, however, posed various challenges. First and foremost was ensuring spatial uniformity by seeking events outside major world countries, emphasizing the importance of including events often overlooked by Western schoolbooks. Another difficulty involved accurately placing the event within the country, especially when it spanned multiple countries, requiring a decision on which one to select.

Lastly, the task involved technical and mechanical aspects, such as the need to click on the update button and then on the save button to finalize the changes, including adjustments to borders.

4.5 New Scenes: "1901-1950" and "1951-2000"

The primary enhancement encompasses the addition of two distinct scenes, representing the "1901-1950" and the "1951-2000" periods, both meticulously populated with numerous historical events. The process initiated by utilizing an existing global scene as a foundation and subsequently creating the new '1951-2000' scene, following the outlined steps in Section 4.1. Subsequent to extensive work, including border modifications detailed in Section 3.2.5 to closely align with the geopolitical landscape of 1951, this scene served as the basis for constructing the "1901-1950" scene. This step proves crucial, as it often necessitated making parallel changes to both scenes. For instance, the addition of a minor country like Malta, absent in the initial countries package, required inclusion in both scenes. Conversely, scenarios may arise where a country, non-existent before a specific date, must be removed from both scenes.

Each scene is accompanied by a title, a brief description, the date of occurrence, and a link to relevant Wikipedia resources. A notable feature of the application is the capability to group states sharing the same color. This feature can be leveraged to highlight, for example, various colonies persisting into the twentieth century, providing users with a unique visual perspective on historical geopolitical relationships.

The outcomes of this thesis will significantly contribute to the expansion of this project, intended for utilization in history education within schools and as a broader resource for research exploring virtual reality, augmented reality, and computer graphics for educational purposes. The thesis showcases how these technologies can elevate interactivity and augment the educational value of applications in the educational domain. The addition of two new scenes, one for the "1901-1950" period and another for the "1951-2000" period, marks a substantial advancement in the project's scope and historical coverage.

4.6 New Categories: "Event" and "Tragedy"

In the process of creating new events, a classification dilemma based on their types was encountered. Certain events did not neatly fit into the existing main categories, prompting the introduction of a new 'Event' category for generic occurrences. Furthermore, considering the prevalence of tragedies, often linked to wars or persecutions in the twentieth century, a decision was made to establish a new category called 'Tragedy'. The instructions for both categories can be found in the previous chapter, specifically in Section 4.2. When selecting a texture for a new category icon, it is crucial to ensure its commercial usability and adherence to Creative Commons licensing. For the 'Tragedy' and 'Event' icons (Figure 4.19), flaticon.com was utilized, providing a free license for both personal and commercial use with attribution. The relevant lines for future public availability are retained below.



Figure 4.19: The icons for categories "Tragedy" and "Event"

```
<a href="https://www.flaticon.com/free-icons/sad" title="sad icons">Sad icons
  created by Freepik - Flaticon</a>
<a href="https://www.flaticon.com/free-icons/history" title="history
  icons">History icons created by Aldo Cervantes - Flaticon</a>
```

4.7 Game Mode

Figure 4.20: *Game mode*

For a comprehensive exploration of the newly integrated features, the developer could switch to Game Mode within Unity. In this mode, the project actively simulates the application. Upon initiation, the camera is positioned inside the scene, initially centered over Europe, emphasizing the focus on historical events in that region. To navigate the globe, it's necessary to press and hold the left mouse button while dragging. The scroll wheel facilitates zooming in and out. Holding down the right mouse button and moving it enables an anti-clockwise rotation of the world, although this feature is rarely used for exploring map events.

The InGameMenu, acting as the primary graphical interface for the majority of interactions within the application, is the initial prominent element noticeable in the Game view. This graphical user interface (GUI) is compartmentalized into three main parts:

1. To the right there are the two main filters

- Cities Filter: since city types are limited to capitals, regional capitals, and regular cities, there are three toggles to control their visibility. Additionally, the sheer quantity of cities in the application poses another issue. At the start of Game mode, only the capitals are active. Enabling visibility for all other cities can lead to events being less prominent and make user interaction more complex.
 - Category Filter: this is a list of toggles controlling the visibility of event icons by type. At the beginning, they are all activated.
2. At the center bottom there is the slider to change the scene, so the time period.
 3. To the left part there are different kinds of UI elements:
 - Buttons that colorized countries, reset all the highlighted ones, and there's still the functionality "Fly to a random country" that was present in the original software.
 - Drop-down button: with colonialism playing a central role in the 20th century, particularly in Africa, a drop-down button has been incorporated into the graphic interface. This feature enabled the coloring of territories controlled by major European states as they pursued conquest and domination. The functionality is present in both added scenes, effectively emphasizing significant historical shifts, including mid-20th-century declarations of independence, as observed in India, for example. Furthermore, this feature can be applied to scenes that will be added in the next development of this application, representing earlier centuries, as colonialism has historical roots dating back.
 - Show events, Show countries and Show cities are simple toggles that manage whether or not the corresponding elements are displayed

In Game mode, there are other important interactions worth highlighting, particularly with cities and events. Hovering the mouse over cities reveals their names, while for events, an information box appears, including the name, year, and description. Notably, the cursor transforms from the arrow shape to the Wikipedia logo, signifying that clicking the left mouse button will open the browser to the page corresponding to that event.

Chapter 5

Conclusions

The work conducted during this thesis has undeniably contributed positively to the application's development. The data entry process has illuminated the potential for future software enhancements, paving the way for practical use in education. The comprehensive historical perspective offers deeper insights, and the interactive nature has the potential to make learning more engaging and enjoyable.

5.1 Results

Summarizing, the results obtained are:

1. The incorporation of two additional scenes, spanning the first and second halves of the twentieth century, involved adjusting the borders of countries globally to reflect the geopolitical landscape in 1901 and 1951.
2. Through online research and consultation with school books, a list of significant historical events from the twentieth century was compiled. This comprehensive compilation not only included historical events but also encompassed various facets of human culture, spanning art, science, literature, music, and more. Each event was meticulously integrated into the application, categorized by type, to achieve a satisfactory and cohesive representation on the map.
3. An improved user interface, using tools already present within the software reworked both in terms of graphical editing and associated code.

5.2 Future Developments

In terms of future developments, there are two primary areas of focus. The first involves technical improvements, encompassing enhancements at the code level as well as improvements to the interface and graphic design. The second area revolves around practical usability, with a focus on incorporating new features to enrich the overall user experience.

One first thing that can be noticed already in the explanation of the software editing interface on Unity in Chapter 3 is the overabundance of tools present. This is due to the generalist nature of the software underlying the project. "World Map Globe Edition 2" provides elements that in the context of an educational application have no relevance, but might have it for a video game, for example.

Addressing this issue at the code level is crucial. It involves a comprehensive review of the codebase, both quantitatively and qualitatively. This entails identifying and removing any unused or redundant lines of code, ensuring that the scripts compile correctly, and optimizing the overall code structure for better efficiency. This process aims to streamline the codebase, making it more manageable and improving the overall performance of the application.

Improving the efficiency of the data entry process, especially in aspects like updating and saving events, is a significant goal for future developments. Enhancing the workflow in these areas not only increases productivity by reducing data entry times but also allows for the inclusion of a larger volume of historical events, contributing to a more comprehensive and detailed representation of global history within the application.

The improvements at the interactive level of the application, however, are numerous, for instance:

- Simplifying the user interface while expanding the interactive capabilities is a noteworthy goal for future improvements. Creating a more intuitive and streamlined interface enhances user experience, making it easier for individuals to engage with the application. Simultaneously, introducing diverse interaction options beyond event and city viewing adds depth and versatility, providing users with a richer and more engaging experience as they explore global historical content. For example, an initial development might involve exploiting the tag field to group events from different categories with a single theme in common.
- Integrating a quiz feature into the application is a valuable idea for enhancing user engagement and promoting active learning. By incorporating quizzes, users can test their knowledge of historical events in an interactive and educational manner. The quizzes can cover various topics, time periods, and geographical regions, offering a comprehensive way for users to reinforce what they've learned while using the application. This addition not only contributes to the educational aspect of the application but also makes the learning experience more enjoyable and interactive.
- Implementing a feature that allows users to save their favorite events and create custom timelines adds a personalization element to the application. This feature can enhance user engagement and cater to individual interests. Users can curate their own timelines based on specific themes, regions, or time periods, providing a more tailored and meaningful experience. It also encourages users to revisit and explore events that are of particular interest to them, fostering a deeper connection with the historical content available in the application.
- Incorporating multimedia elements such as text, images, and videos into historical events can significantly enhance the interactive and educational aspects of the application. By leveraging various sources, including reputable historical archives, documentaries, and scholarly articles, the application can provide users with a comprehensive and immersive learning experience. Adding relevant text descriptions, historical images, and video clips associated with each event can offer users a deeper understanding of the context, significance, and impact of historical occurrences. This multimedia integration not only enriches the educational content but also caters to different learning preferences, making the application more engaging and informative.
- Introducing a multiplayer mode to the application opens up exciting opportunities for collaborative learning and exploration. Users can connect with others, share insights, and collectively delve into historical events. This feature could enable collaborative activities such as exploring the map together, discussing specific events, and collectively creating timelines.

Moreover, allowing users to save their favorite events and create custom timelines enhances personalization and organization. Users can curate their own historical journeys, focusing on specific themes, time periods, or regions of interest. This not only facilitates a tailored learning experience but also encourages users to actively engage with the content and create a personalized historical narrative.

Incorporating these features contributes to a more dynamic, interactive, and community-driven educational experience within the application.

A final aspect regarding the future developments of the application is the introduction of augmented reality and smartglasses which will be able to make the software even more unique in its kind.

5.2.1 Augmented Reality (AR)

A significant technical development, implemented by a recent intern on the project, involves transposing the application from desktop to Augmented Reality using smart glasses, particularly the HoloLens.

AR overlays digital information in the real world. This can be done using a headset that displays digital information over a live video feed or using a smartphone or tablet that displays digital information over the camera view.

This project will use augmented reality to distinguish itself from other similar ones. Augmented reality will allow us to visualize data or information more intuitively, making it more engaging, interactive, and innovative. The best solution currently available for commercial use in a variety of industries, including manufacturing, healthcare, and education is HoloLens. It is also being used by some developers to create new and innovative applications. These models can be used to create interactive learning experiences or to provide students with a visual representation of complex concepts.

So these are the grooves that have been mapped out that will lead to future developments of this innovative application.

Bibliography

- [1] Education Technology Market Size, Share & Trends Analysis Report By Sector, <https://www.grandviewresearch.com/industry-analysis/education-technology-market>
[Online]. Accessed on Nov. 16, 2023
- [2] World Map Globe Edition 2, <https://assetstore.unity.com/packages/tools/gui/world-map-globe-edition-2-150643#description>
[Online] Accessed on Nov. 24, 2023
- [3] Geacron, <https://geacron.com/home-en/>
[Online] Accessed on Nov. 12, 2023
- [4] Omniatlas, <https://omniatlas.com/maps/>
[Online] Accessed on Nov. 12, 2023
- [5] Unity official website, <https://unity.com/>
[Online] Accessed on Nov. 15, 2023
- [6] Kronnect official website, <https://kronnect.com/>
[Online] Accessed on Nov. 24, 2023
- [7] JetBrains Rider official website, <https://www.jetbrains.com/rider/>
[Online] Accessed on Nov. 24, 2023
- [8] GitHub official website, <https://github.com/>
[Online] Accessed on Nov. 24, 2023