NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

TIAGO JOÃO FERREIRA NUNES

Bachelor in Computer Science and Informatics Engineering

CREATING AN IMPROVED AND IMMERSIVE VISITOR EXPERIENCE FOR MUSEUMS

COMPLEMENTING EXTENDED REALITY EXHIBITS WITH CUSTOM EXHIBIT CREATION

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CREATING AN IMPROVED AND IMMERSIVE VISITOR EXPERIENCE FOR MUSEUMS

COMPLEMENTING EXTENDED REALITY EXHIBITS WITH CUSTOM EXHIBIT CREATION

TIAGO JOÃO FERREIRA NUNES

Bachelor in Computer Science and Informatics Engineering

Adviser: Armanda Rodrigues
Associate Professor, NOVA University Lisbon

Co-adviser: Nuno Correia
Full Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2023

Creating an improved and immersive visitor experience for museums

Acknowledgements

This thesis would not be possible without my Adviser Armanda Rodrigues and Co-Adviser Nuno Correia. I would like to thank them for their commitment and dedication towards the elaboration of this thesis, as they provided me with the invaluable help I needed while carrying out the developed work, as well as writing the document itself.

In addition, I would like to extend my sincere thanks to NOVA School of Science and Technology, for providing me with the resources and conditions that enabled me to partake in this project.

Lastly, I cannot express enough how grateful I am to my parents and grandparents, who have always supported me, both financially and morally, and enabled me to be where I am today. An additional thank you goes to Professor João Lourenço for freely providing the template for this document.

"The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do." (Ted Nelson)

Abstract

Visiting museums or other spaces of cultural interest may prove to be prohibitive due to numerous factors, such as health problems, distance, mobility issues or schedule incompatibility. As such, there is a need to develop virtual alternatives to on-site visits, as well as to improve the experience of said visits through the addition of virtual elements, which provide functionalities that would otherwise be impossible to experience. On the other hand, it is also necessary to implement a platform that facilitates the creation of complete exhibitions by curators and staff of cultural institutions. This is because existing tools suffer from some usability and ease of use problems, and are not suitable for a more generic domain.

This is a challenging problem, as it concerns the merging of several software components to create a complex system. Additionally, it is also necessary to develop a modular and adaptable solution that can be used in multiple cultural institutions, regardless of their context.

This thesis proposes the creation of such a solution, which would allow users to interact with works on display in an innovative and immersive way, as well as help to promote museums and organize visits. The solution seeks to expand an existing system, consisting of a Virtual Reality (VR) Application, an Augmented Reality (AR) application and an administrative Desktop application.

The work carried out during the duration of this thesis consisted of the implementation of a navigation aid, in the form of a "mini-map", as well as an additional interface where users can navigate, search, and locate works within the exhibition. An advanced editor was also developed within the desktop application, which allows curators to manipulate and import personalized objects in order to compose complete exhibitions that can be viewed in the VR application.

The work also included the design and development of the components to be added, as well as the overall evaluation of the system through tests with system users and domain experts. This evaluation made it possible to draw positive conclusions regarding the developed system, as well as identify areas for potential future improvements.

Keywords: Virtual Reality, Augmented Reality, Extended Reality, Digital Twin, Museology, Object Manipulation

Resumo

A visita a museus ou outros espaços de interesse cultural pode revelar-se proibitiva devido a inúmeros fatores, como saúde, distância, mobilidade ou incompatibilidade de horários. Como tal, surge a necessidade de desenvolver alternativas virtuais a visitas presenciais, bem como melhorar a experiência das visitas presenciais através da adição de elementos virtuais, os quais fornecem funcionalidades que de outro modo seriam impossíveis de experienciar. Por outro lado, é também necessária a implementação de uma plataforma que facilite a criação de exposições completas por parte de curadores e staff das instituições culturais. Isto porque ferramentas existentes sofrem de problemas de usabilidade e de facilidade de utilização, não sendo adequadas a um domínio mais genérico.

Trata-se de um problema desafiante, visto que diz respeito à união de vários componentes de software para a criação de um sistema complexo. Adicionalmente, é igualmente necessário o desenvolvimento de uma solução modular e adaptável, que possa ser utilizada em múltiplas instituições culturais, independentemente do contexto.

A proposta desta tese é a criação de uma tal solução, a qual permita a utilizadores interagir com obras em exposição de forma inovadora e imersiva, bem como auxiliar na divulgação dos museus e na organização de visitas. A solução procura expandir um sistema existente, composto por uma Aplicação VR, uma aplicação AR e uma aplicação Desktop administrativa.

O trabalho realizado durante a duração desta tese consiste na implementação de um auxílio à navegação, sob a forma de um "minimapa", bem como uma interface adicional onde os utilizadores podem navegar por, pesquisar, e localizar obras dentro da exposição. Foi também desenvolvido um editor avançado, dentro da aplicação de desktop, que permite a curadores manipularem e importarem objetos personalizados de forma a compor exposições completas que poderão ser visualizadas na aplicação VR.

O trabalho incluiu ainda o design e desenvolvimento dos componentes a adicionar, bem como a avaliação global do sistema através de testes com utilizadores do sistema e peritos do domínio. Esta avaliação permitiu tirar conclusões positivas relativamente ao sistema desenvolvido, bem como identificar áreas para potenciais melhoramentos futuros.

Palavras-chave: Realidade Virtual, Realidade Aumentada, Realidade Extendida, Digital Twin, Gémeo Digital, Museologia, Manipulação de Objetos

Contents

List of Figures

List of Listings

Glossary

backend The Internal Structure and data of an Application. Responsible for calculations and most processing. 21

C# Multi-paradigm object-oriented programming language developed by Microsoft in 2000. 21, 29

Earth and Fire (Not to be confused with Earth, Wind and Fire) Dutch Rock band active from 1968 to 1983 and from 1987 to 1990. 17

Emoji Pictogram used in social media and on the internet as emotional cues. 28

frontend The graphical part of an application the user interacts with. Sometimes a synonym for Graphical User Interface. 21

GitHub Internet platform for software version control based on Git. 21

Java Widely used object-oriented programming language originally developed in 1995. xvii, 21

Kotlin General purpose object-oriented programming language developed by JetBrains in 2011, inter-operable with Java. 21

Mixtec Indigenous Mesoamerican Civilization inhabiting the "La Mixteca" region of Mexico. 15

QR-Code Two dimensional barcode used to quickly link information to a mobile device. 1

Software Development Kit Suite of software development tools and libraries. Often include a compiler and a debugger. xix, 12

Unity Cross-platform game and simulation engine launched in 2005. 16, 21, 26, 30, 35, 80

UV Mapping 3D Modelling Process of transforming a 3D Model's surface onto a 2D Image. 15

VRChat Virtual Reality Social Platform originally released in 2014 where users interact with each other in a virtual environment. 18

XML Extensible Markup Language, a markup language first defined in 1998. 25, 26, 29

Acronyms

2D Bi-dimensional 27

3D Tri-dimensional 8, 15, 16, 26, 27

AR Augmented Reality vi, viii, 3, 4, 8, 9, 10, 11, 12, 14, 15, 16, 17, 19, 20, 23, 26, 28

CLI Command Line Interface 7, 13

GUI Graphical User Interface 7, 8, 11

HMD Head Mounted Display 11, 12, 27, 28

HTTP HyperText Transfer Protocol 29

SDK Software Development Kit 12, 14, 27

TUI Tangible User Interface 8

UI User Interface 7, 8, 41

VICARTE NOVA's Glass and Ceramic for the Arts Research Unit 2

VR Virtual Reality vi, viii, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 17, 18, 20, 22, 24, 26, 27, 28, 31, 61, 63

XR Extended Reality 11, 15

1

Introduction

This first chapter aims to discuss the context (Section 1.2), and motivation (Section 1.1) of this thesis, as well as a description of the problem to be tackled, and its main objectives (Section 1.3). Additionally, it provides a brief description of the developed solution (Section 1.4) and a summary description of the overall structure of the document (Section 1.5).

1.1 Motivation

The global COVID-19 pandemic demonstrated the need to use digital alternatives to support several facets of our daily lives, from shopping, to banking, from work and learning, to leisure. It has also tested the viability and utility of the systems already in place.

This period also showed that some sectors were severely impacted by reduced attendance, such as tourism and culture. However, other factors contributed to this decrease in the number of visitors, besides the global pandemic. In addition to it, difficulties related to global access to artefacts, spread through multiple cultural institutions, can be detrimental to the overall user experience in museums. Furthermore, there is a need to motivate younger audiences to visit museums and other cultural institutions. Users in this age range enjoy sharing their experiences, interacting with the world around them, and expressing their creativity.

Nonetheless, some efforts have been made to provide more immersive and interesting experiences for museum visitors. Among these, for instance, are NOVA's own PASEV[1] or LxConventos[1] projects.

There have also been some small-scale experiments related to the virtualization of tourist destinations or museum exhibitions. From QR-Code based experiences, to fully fleshed out Virtual Reality experiences, such as Uffizi Galleries'[2] or Louvre's[3] virtual tours, a few steps have already been taken to bring the latest technologies to these sectors.

[1] https://pasev.hcommons. org/
[2] https://www.uffizi.it/en/online-exhibitions/uffizi-virtual-tour
[3] https://www.louvre.fr/en/online-tours

These technologies not only can increase attendance to museums and cultural sites' by providing an improved experience to on-site visits, by integrating virtual components in a real environment, but can also provide a virtual alternative via Virtual Reality exhibits[2]. Visitors can partake in activities and experiences which they would not be able to in a physical environment, as well as examine works or monuments that cannot be publicly exposed due to preservation issues or that have already been damaged or destroyed[3]. This can also help to preserve artifacts that are in risk of disappearing in the near future, either due to the aforementioned causes or due to external factors such as war, theft, vandalism, or private acquisition. Another possible use is to link to and display works that are present in other museums, creating a larger, unified collection with works from different institutions, which can be viewed simultaneously in distinct locations.

To quell any possible reluctance by cultural institutions towards the use of these new technologies, an effort has to be made to ensure that the transition to said technologies is as seamless and as effortless as possible. Tools that do not allow museum personnel the creative freedom to assemble their own custom exhibits, or that are too complicated to use, can lead to some friction related to the adoption of VR and AR technologies. Thus, it is paramount to offer these institutions a tool that is both easy to use, and that allows the creation of custom content to be used in whatever context might require it.

1.2 Context

Currently, in colaboration with NOVA's NOVA's Glass and Ceramic for the Arts Research Unit (VICARTE)[4], development is underway on the creation of a framework and a platform to enable the creation and visualization of virtual museum exhibits, as well as the augmentation of in-situ visits. This platform is comprised of a set components developed as part of the work of a previous thesis written by Pedro Lourenço in 2022[4].

The work of this thesis intends to expand on this existing platform by implementing a system to enable the creation of virtual exhibits, in a simplified and efficient way. This was accomplished by extending existing functionalities to make them compatible with multiple exhibits, as well as the development of a virtual exhibit editor, with the ability to import 3D models of objects and rooms.

1.3 Problem Description and Objectives

The problem of the work carried out as part of this thesis was the extension of the previous framework by providing intuitive and simple ways for curators in cultural institutions, such as museums, art galleries, or other cultural heritage sites, to assemble custom exhibits based on digital representations of real-world locations and real-world artefacts, without requiring significant computer knowledge. Additionally, it is also important to

[4]https://www.vicarte.fct.unl.pt/

develop functionalities that can improve visit experience in museums and cultural sites, within the existing system.

This previous system consisted of a mobile AR application, to be used by guests visiting the physical museum space, and a VR application to be used with a dedicated headset (Figure 1.1).

The Mobile Application allowed users to access additional information about a specific art piece by pointing their device at it, as well as viewing comments and feedback from other users.

Figure 1.1: Example of Normal AR Application Usage (Previous Framework).

The VR Application is geared towards people that cannot physically visit a museum. It consists of a virtual representation of the museum space, with digital copies of each work. Upon approaching a work, the user can execute the same activities as in the Mobile Application (Figure 1.2).

Figure 1.2: Example of Normal VR Application Usage (Previous Framework).

In addition to these, there is an administrative application designed to supervise the state of the system.

The developed work focused on improving on the previous system by adding further functionality to the base system, and implement support for different exhibits via exhibit creation.

These functionalities are as a navigational aid in the form of a minimap, an artwork search menu, support for three-dimensional artworks and artefacts, and dynamic instantiation of artefacts from the .XML files.

To support different rooms and different exhibits, so that the system can be used in different museums, it was necessary to improve the Administrative application by developing a built-in exhibit editor that offers museum staff tools to create and edit exhibits. This editor allows museum personnel to import 3D Rooms and 3D objects, as well as the 2D objects that were already supported by the existing system, to assemble a custom exhibit.

1.4 Main Contributions

The core contributions produced as part of this thesis were distributed between three different categories, each relating to which part of the system was extended by their implementation. The focus of the contributions were mainly the VR and Administrative applications. As such, no significant modifications were implemented exclusively in the AR Application.

1.4.1 Global Changes

1. Added a bypass to server checks so development and debugging can be conducted in offline mode. This bypass consists of a single global boolean variable that enables/disables server connections, thus forcing the system to use local storage at all times, if these connections are disabled.

2. XML File structure altered to enable compatibility with dynamic instantialization and the additional artwork types.

1.4.2 VR Application

1. A navigation aid in the form of a minimap was implemented as part of the user's model in the VR Application, displaying the user's location and the location of nearby artworks. It can additionally

2. An artefact search and browse menu, openable via a specific controller button, that enables users so view currently exhibited artworks and locate them, by clicking on them or selecting them and following visual and audio cues.

3. Implemented dynamic artwork instantiation, replacing the preset static arworks. The gallery scene starts empty, and builds the exhibit from the artefact data present

on the .XML, generating the corresponding artefacts according with their properties.

1.4.3 Administrative Application

1. All improvements made in the VR Application were ported and modified to the Adminstrative Application's gallery scene. The minimap is displayed in the upper right corner of the screen in the administrative application, and the artefact search and browse menu is opened with the "HOME"key.

2. Creation of a feature-rich editor application that allows museum personnel to import custom 3D rooms and artefacts, as well as 2D artefacts, to assemble custom exhibits that can be viewed in the VR application..

1.4.4 DAACH Eurographics Paper

The work of this thesis was the subject of a Paper to be published on the Digital Applications in Archaeology and Cultural Heritage (DAACH) journal as part of the 2023 Eurographics Annual Conference. Writing this paper was an opportunity to consolidate all knowledge obtained during the entire development process into a single article. The contents of this article would then serve as the base text upon which this present document was written.

1.5 Document Structure

This thesis document is organized into four different chapters:

1. **Introduction** - This first chapter serves as an introduction to the theme, context, and motivation of this thesis, as well as providing a short description of the developed solution.

2. **Fundamental Concepts** - This chapter works to explain and describe several theoretical concepts related to the theme of this thesis, to provide readers with a basic knowledge and understanding of some of the core elements and ideas shown.

3. **Related Work** - The third chapter examines several systems that have some degree of commonality with the work that was developed as part of this thesis. Apart from giving readers a general idea of the current status quo regarding the technologies to be used, it also highlights approaches and methodologies that served as an inspiration and starting point for the developed system.

4. **System Design and Implementation** - This fourth chapter details the approach that was followed during the course of this thesis, from technologies used, to a description of the development process, as well as a time schedule where the original

workplan can be compared to the actual work carried out throughout the development process.

5. **User Evaluation** - This chapter describes the evaluation process undertaken by users. It explains the evaluation methods used, the steps taken during the process, and the results obtained from this process, as well as their impact and significance to the work of this thesis.

6. **Conclusions** - The last chapter focuses on gathering conclusions from the work of this thesis. It identifies shortcomings, lessons learned, and user feedback gathered during the course of system development, as well as future improvements and modifications which can improve and provide additional content to the system, making it suitable to a larger group of potential clients.

2

Fundamental Concepts

To acquire a better understanding of the context and technologies of the work carried out as part of this thesis, some fundamental concepts require some further explanation so that their presence throughout this document does not cause unnecessary confusion to the reader.

2.1 User Interface (UI)

In Human-Computer Interaction, a User Interface (UI) is defined as any means the user and the computer have to communicate among themselves. That is, whenever a certain input is provided to the machine, the corresponding output is calculated and the user is informed of the result of this operation.[5]

There are various types of UIs, each geared towards a specific application. Of special note are the following:

- **Command Line Interface (CLI)** Perhaps one of the most basic types of Human-Computer Interaction, in CLIs the user interacts with the machine by inputting command via a text command prompt. Owning to crude graphics and prioritizing performance, the output is often presented in plain text. These interfaces are often used in administrative and enterprise environments, mostly in software development or maintenance.[6]

- **Graphical User Interface (GUI)** The most widely used type of interface, it bases itself on visual elements to guide user interaction. Unlike CLIs, GUIs are much more expressive, as well as easier to use for users unfamiliar with technology. GUIs work via user interaction with the graphical elements such as clicking buttons, dragging icons, scrolling, and several other actions. In contrast with CLIs, one drawback is that rendering all visual elements introduces a higher performance cost. This prevents GUIs from being the interface of choice in performance critical environments.[6]

- **Tangible User Interface (TUI)** A more recent development, TUIs characterize themselves for relying heavily in physical interactions. The user manipulates digital information my interacting with physical elements. These elements are then interpreted by the system as a combination of inputs, and the system can then display a corresponding output visually, based on the input parameters[7]. ReacTable, from Kaltenbrunner et al.[8] is a very well known example of this type of interface in action, in which the user places small tangible cubes on a table top to compose music via an electronic synthesizer.

2.1.1 User Interfaces in the context of the work carried out

The Virtual Reality and Augmented Reality applications in the previous system possessed a Graphical User Interface, in the form of the Buttons the user must press to access the settings, the drawing functionality on VR and the overlaid comments and reactions on artworks. Apart from GUIs, it can also be argued that the user interacting with the artworks in an AR environment might be considered a TUI, as the physical artworks are part of the interface itself.

Additionally, the navigational element and artwork browse menus implemented as part of this thesis are considered part of the Virtual Reality (VR) Application's GUI. They are visual elements that provide the user with information regarding the state of the system, and allow the user to interact with it, thus enabling Human-Computer Interaction.

Moreover, the Exhibit Editor contains its own UI, divided into two categories, two dimensional GUI, represented by the menu, buttons, and tooltips, an example of a GUI in a more traditional sense, and the three-dimensional UI, being the Tri-dimensional (3D) compnents the user can interact with, the gizmo and the imported objects themselves.

2.2 Augmented Reality (AR)

According to Azuma et al., Augmented Reality (Augmented Reality (AR)) consists in overlaying digital information over a real environment, thus giving the illusion that both worlds are one and the same.[9]

Use of AR on an everyday basis has several applications, from information retrieval, to navigation, social interaction and, of course, leisure and cultural applications[10].

Traditionally, the usual means to experience AR were handheld visors or Head Mounted Displays. However, with the development of smartphones, it was soon attempted to use these devices as AR tools. Equipped with a camera and a large screen, smartphones and tablet computers allow users to instantly interact with their surrounding environment through AR[11].

While convenient, smartphone-based AR is not as immersive as using a dedicated Head Mounted Display (HMD) like Microsoft's HoloLens or Google's Glass (pictured in figure 2.2 below). Due to lack of interest, the prohibitive price of some of the prototypes,

Figure 2.1: Example of an Augmented Reality application running on a mobile device (Tablet Computer) (Public Domain Image).

and technology restrictions, this technology has gone largely undeveloped. However, recent advances in computing and miniaturization have allowed for improvements in headset size and price[12].

However, since Head Mounted Displays are a single purpose device, unlike smartphones, it is expected that wide scale adoption by users might not happen in the near future.

Figure 2.2: User wearing a Google Glass Head Mounted Display (Public Domain Image).

2.3 Virtual Reality (VR)

Unlike AR, which uses real world imagery to overlay virtual elements, Virtual Reality (VR) consists in immersing the user in an entirely virtual world.

To achieve this, the user is detached from any notion of the real environment by means of an enclosed Head Mounted Display (HMD). This HMD consists of two screens, one

for each eye, that give the user a stereoscopic view of the virtual world, while at the same time blocking any view of the real world. The user is, thus, fully immersed in the VR environment. This means that VR can only be properly experienced via the use of a specialized closed HMD, as mentioned in section 2.5 (pictured below, in figure 2.4).

A downside of this system is that, due to the lack of situational awareness, the user can sometimes feel some motion sickness when moving around in the virtual world.[13] A common method to mitigate this is to limit the user's movement to teleportation to the target location, thus ensuring the user is always stationary. Another disadvantage of VR is that inside the enclosed HMD there is no apparent way to guarantee a steady flow of air, which can result in a significant build up of heat and cause increased discomfort. Additionally, long exposure to the screens in the HMD can result in eye irritation due to reduced blinking. This can then lead to headaches and blurred vision. [14]

Since the device does not need to visually track specific points of the real world environment and the technology to display virtual information can be more intrusive (as the wearer does not need to see the real and virtual worlds simultaneously), unlike an AR device, VR headsets started to be developed and experimented with before their more complex AR counterparts. Nintendo's 1995 Virtual Boy is an early commercially failed example of this concept (Pictured in figure 2.3.).

Figure 2.3: Nintendo Virtual Boy (Public Domain Image).

Figure 2.4: Consumer-grade Virtual Reality Headset (Public Domain Image).

Due to being a more mature technology when compared to AR, since the 1980's VR has been used in numerous contexts, from scientific research[15], to engineering, to military training[16]. These early experiments led to an increased awareness and development of the technology that culminated in the first commercially successful VR headset being released in 2012 by Oculus, now part of Meta. More than a decade later, a niche market for VR gaming and entertainment appeared. Due to the emergent necessity to have global access to cultural experiences, this market has grown in recent times.

2.4 Extended Reality (XR)

Extended Reality (XR) is a commonly-used umbrella term used to describe VR, AR and Mixed Reality (MR). The latter is often synonymous with AR, being a further enhancement of the concept, and the terms are frequently used interchangeably.[17]

2.5 Head Mounted Display (HMD)

A Head Mounted Display (HMD) is a peripheral device that consists of one or two small displays located close to the user's eyes and worn on the user's head as part of a helmet or an headset.

These displays present images and/or GUIs near the wearer's eyes, providing a greater sense of immersion. HMDs allow users to have improved control over their virtual movements since they are not restrained by the limitations of their input devices, such as computer mice, keyboards or controllers.[18] HMDs allow six degrees of freedom in a single device, unlike other peripherals. This means that the user possesses the ability to rotate or translate in each of the three dimensional axes (Three degrees of rotation, three degrees of translation). [19]

HMDs have a multitude of applications in military, industrial, gaming, training and entertainment environments. Some HMDs include several sensors that monitor several aspects of user behaviour, such as:

- **Head Tracking**, to track the position and orientation of the user's head, to determine what needs to be shown in the user's field of view at any given time and to update the user's position within the virtual content[20];
- **Eye Tracking**, provides a way for the user to use their eyes as a pointing device, allowing for interactions with the interface, either by navigating the interface, similar to a mouse cursor, where an action such as blinking or fixating the gaze can be perceived as clicking, or by highlighting specific areas of the GUI according to where the user is looking at[21];
- **Hand or Body Tracking**, is present in some devices that possess numerous cameras to capture footage around the user. This footage is analyzed in real time in an attempt to identify the location of several key points in the human body, such as fingers, arms, legs, or torso[22]. A relatively commercially successful example of this is XBox's Kinect system, although Kinect differs in the aspect of being a set of cameras mounted in a fixed location, as opposed as being integrated within the HMD (pictured below, figure 2.5)[23];

There are several types of HMDs, designed for different applications, such as traditional closed HMDs (represented in figure 2.4), mainly used in VR, to open Optical HMDs that resemble eyeglasses (example pictured in figure 2.2), used in AR. HMDs are often

Figure 2.5: XBox Kinect (Public Domain Image).

used in conjunction with advanced motion controllers so that the user's hands can be detected and interact within the virtual environment. While not part of the headsets themselves, they are required for most VR and some AR applications.

Figure 2.6: Motion Controllers used with Oculus Rift Virtual Reality HMDs (Public Domain Image).

2.6 Technology Relevance

It is important to analyze how the aforementioned technologies were used in the work carried out during the course of this thesis.

VR and AR are critical components of the developed system, since one of the client application makes use of VR through Oculus' Software Development Kit (SDK), and other of the applications uses Vuforia to enable visitors to experience AR [4]. Concerning HMDs, it is the only way to experience VR properly, and as such it is also a fundamental component of the work developed as part of this thesis.

With this said, however, no significant focus was placed in improvements to the AR application. As such, AR technologies did not prove of extreme relevance to the

developed work. In contrast, VR technologies were relevant to the navigational and browsing improvements carried out, as these improvements were made to both the VR application and the desktop application.

Furthermore, as previously mentioned in subsection 2.1.1, graphical users interfaces were an integral part of the developed system, both in the VR and administrative applications. As it is not a console based system, but rather a graphical one, CLIs were not used or developed as part of the work conducted during this thesis.

3

Related Work

This Chapter will examine some projects which develop or use technologies related to the theme of this thesis, such as VR or AR, and Object Manipulation in a cultural exhibition context. It also discusses some studies related with said technologies and analyzes the viability of some of methods and technologies in the context of the thesis.

3.1 Augmented Reality

As mentioned in section 2.2, Augmented Reality is the least mature Extended Reality technology. With this said, however, there are still numerous projects aimed at developing the technology and applying it to a multitude of different contexts.

3.1.1 A Mobile Augmented Reality System for Exhibition Halls Based on Vuforia (Fuguo and Zhai, 2017)

This system has many similarities with the AR component of the current system[4], in that it uses the Vuforia[1] SDK to allow user interaction with virtual objects. Although the technology used is the same, this system presents just a proof of concept example, and does not apply the technology to any particular context or environment.

The purpose of this particular system is to simply demonstrate Vuforia's capabilities by displaying a small model of a city over a real-world map of the city. The article also describes the internal architecture for an AR application, and compares various tracking technologies, such as marker tracking and markerless tracking.

The article reported that the AR demonstration performed very well with no stability nor performance issues. However, it was also reported that the amount of content needed to use this type of application in a relevant context is relatively high. This problem can be easily solved by having different models and different trackers to provide users with a more complete system[24].

[1] https://developer.vuforia.com/

Figure 3.1: Fuguo and Zhai's Application Demonstration[24].

3.1.2 Design of interactive Museographic Exhibits using Augmented Reality (Ramírez et al., 2013)

This system covers the same use cases as we do.

The goal of the system is to provide an immersive museum experience by implementing an AR artifact visualization application to improve museum interactivity.

The accompanying article compares attendance numbers from select museums in Mexico and establishes a relationship between museum attendance and the interactivity of the exhibits in the same museum. It focuses on the Museo Regional de Huajuapan (MureH), one of the most important museums in regards to Mixtec Artifacts. The authors argue that the low attendance levels in this particular museum are due to the lack of interactivity, such as XR experiences. This factor is what motivated the authors to develop their own system to increase attendance and improve interactivity in the Museo Regional de Huajuapan.

The system consists of an application that can guide users through a museum and displays adequate information whenever the users approach any of the works being exhibited. The museum opted to manually model each of its Mixtec culture works instead of 3D scanning, using Blender[2]. The textures, on the other hand, were UV Mapped (UV Mapping) from real world photographs.

This approach is similar yet not the same as our AR application[4], since MureH's application renders pre-modelled objects on an otherwise empty environment upon the user scanning a certain marker or element. In our current application, the AR component is simply responsible for displaying contextual information over a physical artifact.

[2]https://www.blender.org/

The application runs using Vuforia[3] (a free, non-commercial version) and the NyARToolKit[4] library. This means that the system was developed using free (albeit not entirely open-source since Vuforia is a closed source product) software, which can be a deciding factor in museums with limited financial resources.[25]

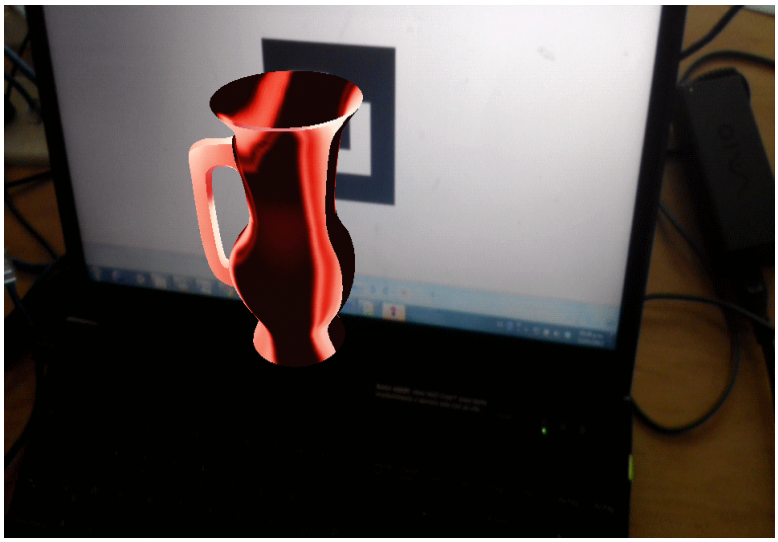

Figure 3.2: Example of Augmented Reality in Ramirez et al.[25], showing a modelled vase in front of a marker.

3.1.3 Implementation of Augmented Reality Technology in Sangiran Museum with Vuforia (Purnomo et al., 2018)

This is another system which shares several similarities with the work of this thesis.

It consists of an application designed for an archaeological museum, namely the Sangiran Museum in Indonesia, which uses Vuforia[5] to digitally display information about around 81 artifacts providing additional context and constantly updated information about them. This contrasts with the previous project where virtual 3D objects would be rendered in a real space. Just like the AR application present in the framework being developed, the goal of Sangiran Museum's system is to overlay digital information over a real object, such as a painting or a fossil.

This system is Unity-based, meaning that the Unity game engine is responsible for merging the virtual information with the real-world footage from the device's camera.

The article also researches the ability of mobile phones to recognize "markerless" markers depending on factors such as lighting conditions and viewing angles. The study found that lighting conditions and viewing angles do, in fact, play a part in Vuforia's capability to detect markers correctly[26].

[3] https://developer.vuforia.com/
[4] https://nyatla.jp/nyartoolkit/wp/?page_id=198
[5] https://developer.vuforia.com/

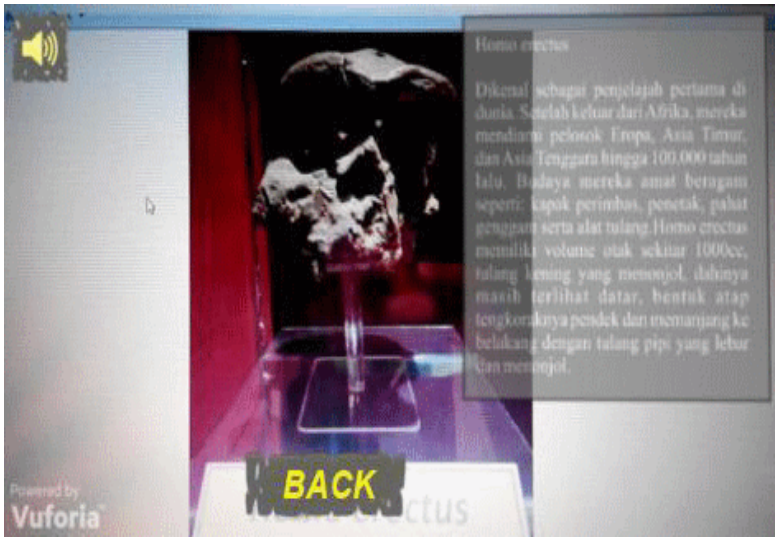

Figure 3.3: Example of Purnomo et al.'s[26] application showing additional information related to an archaeological artifact.

3.2 Virtual Reality

As previously stated, Virtual Reality is a more mature technology when compared to Augmented Reality. Nevertheless, its use in the context of improving visit experience and interactivity in a museum or in a cultural heritage site is still relatively limited.

3.2.1 Mediascape XR: A Cultural Heritage Experience in Social VR (Reimat et al., 2022)

Another recent system, Mediascape XR is one of the few researched systems which focuses on the VR aspect of the application, instead of AR.

It consists of a multi-user experience where one or more users can explore a virtual museum which contains one single artifact, a dress used by the lead singer of the Dutch rock band Earth and Fire, Jerney Kaagman. Users can interact with the dress by moving and rotating it, visualize all sorts of information related to it and see themselves in a virtual mirror, "wearing" that dress. In another room, an interactive concert can be recreated where users are able to interact with several instruments. In a social VR context, multiple users can play together to form a virtual band, just as it would happen in the real world[27].

This collaborative aspect is an important aspect of the virtual experience, and thus something that should be investigated as part of this thesis. Furthermore, the inspection and analysis of heritage artifacts, such as the dress, is something that is included in the subject of this thesis.

Figure 3.4: Structure of the Mediascape XR application[27].

3.2.2 Remote and Collaborative Virtual Reality Experiments via Social VR Platforms (Saffo et al., 2021)

This is a recent article that questions whether if widely available Social VR platforms (VRChat, in this particular instance) are effective tools to conduct remote individual or collaborative studies. This can have several implications in the field of VR research, as traditional crowd sourcing methods required prior training of test subjects to familiarize them with the technology, before conducting the actual tests. Using commercially available platforms would allow researchers to gather participants with a higher level of technological knowledge in Virtual Reality, thus decreasing or completely eliminating any training time, enabling researchers to focus on the studies themselves.

In the article, the authors create two elaborate tests to be compared to real life Human Computer Interaction (HCI) tests. One of the tests involves replicating Fitts' Law, where a participant must move a cursor to a designated target. In this case, the authors used an extension of Fitts' Law developed by Clark et al.[28] adapted to a three dimensional space. Subjects were tasked with moving a cubical cursor to a randomized target and then selecting said target. Task time was recorded and then compared against other studies to determine if the VR environment had any visible effect in the realization of the test.

The other test revolves around two participants collaborating together to draw paths on a networked graph in order to get the least expensive path. Each participant had their own "data layer"with individual path costs, one for monetary cost and another for time. This test involved two phases: One where the two test subjects would compete against one another to draw the lowest cost path in their own path layer, and another phase where both subjects would compromise and draw the path with the smallest cost and time.

The article concludes that VR is a useful tool to perform these types of tests, and that using widely available Social VR platforms such as VRChat provides an easy way to set up said tests[29].

3.3 Effects of Extended Reality

It is also important to study the effects Virtual Reality and Augmented Reality have in improving the user experience and interactivity in a psychological sense.

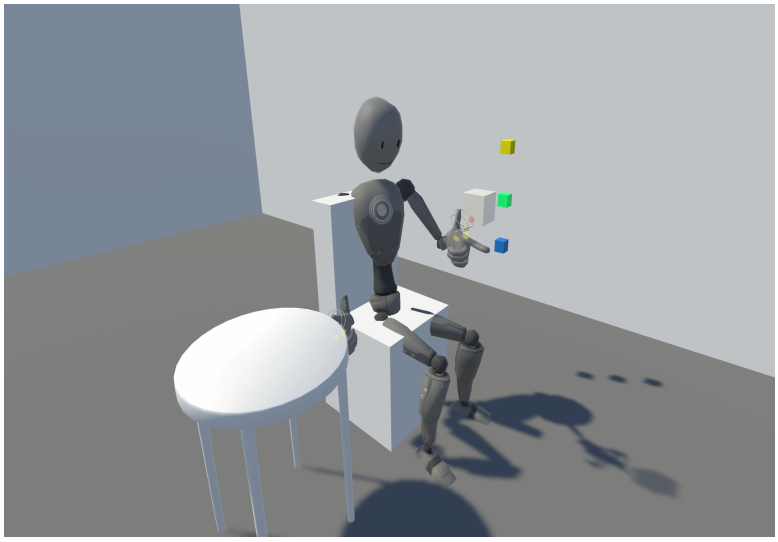

Figure 3.5: One of the two tests conducted during Saffo et al.'s research[29].

3.3.1 Tourists' intention to visit a destination: The role of augmented reality (AR) application for a heritage site (Chung, Han and Joun, 2015)

This paper studies the psychological and societal impact that an AR application has on its users, when applied to a cultural heritage context.

Chung, Han and Joun[30] created an AR application designed for Deoksugung Palace, in Korea, and studied its role in respect with several factors concerning user experience, such as perceived usefulness, ease of use, and intentions to visit cultural heritage sites.

It was found that, if an AR application has an appealing design and a support infrastructure behind it (e.g. from the local site or museum where it is located), it can vastly improve a tourist's desire to visit that heritage site. The study concluded that the perceived ease of use and usefulness that application can provide to them were also significantly improved by these factors.

The result of this study is relevant for the subject of this thesis since it reveals potential benefits of implementing the technology in the desired context[30].

3.3.2 Effects of Virtual Reality and Augmented Reality on Visitor Experiences in Museum (Jung et al., 2016)

In another paper concerned with the effects of Virtual and Augmented reality in a tourist destination or a museum context, Jung et al.[2] investigate whether Extended Reality technologies (Augmented and Virtual Reality) can observably impact the overall visitor experience in a museum or a tourist attraction.

The researchers used, as basis for their scientific method, Short et al.'s Social Presence Theory[31] and Pine and Gilmore's Experience Economy concept[32].

Experience Economy, as defined by Pine and Gilmore, is a set of four classifications

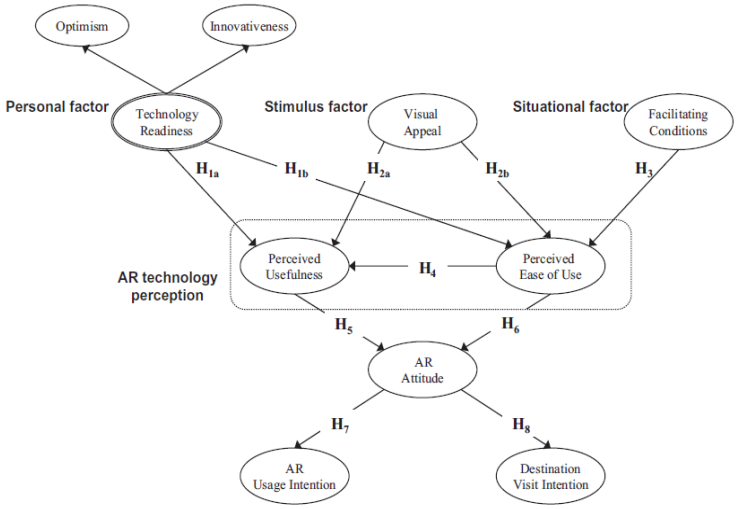

Figure 3.6: Diagram illustrating Chung, Han and Joun's research model[30].

attributed to user experience: Entertainment, Education, (A)esthetic and Escapist[32]. These are the main categories in which experiences generally fall.

This paper selected the Geevor Tin Mine Museum, in Cornwall, United Kingdom, as the location to conduct the study. 163 visitors agreed to participate in the study and were given the opportunity to experiment with an Augmented Reality and a Virtual Reality application centered around the theme and history of the museum. The Virtual Reality application had the advantage of allowing users to experience activities that would otherwise be impossible to take part on, since as riding the mine shaft elevator, which was closed for tourists due to safety concerns.

The paper concluded that the presence of a VR and an AR application at the museum can improve user experience across all four realms of Pine and Gilmore's Experience Economy. It was also found that, apart from aesthetic experience, all realms significantly impacted the overall visitor experience and, consequently, the visitor's will to visit the museum again.

The results of this paper are significant to the theme of this thesis since they demonstrate a clear relationship between VR/AR and an improvement in visitor experience. This can lead to a greater visitor satisfaction and intention to visit again in the future[2].

3.4 Exhibit Editing

Concerning the Exhibit Editor, it was important to have a base system to analyse and to take design and implementation clues from.

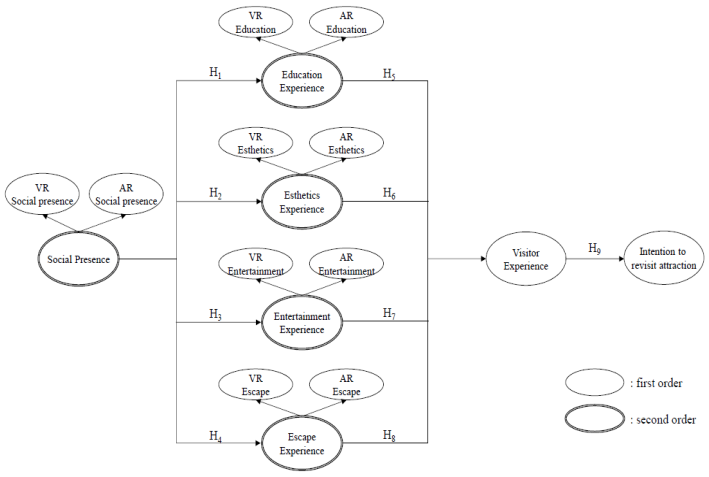

Figure 3.7: Jung et al's proposed research model[2].

3.4.1 VIRTUE — A Virtual Reality Museum Experience (Giangreco et al., 2019)

VIRTUE is a system developed by the University of Basel, Switzerland, that possesses a conceptual and technological overlap with the Virtual Reality component of our application.

VIRTUE consists of a modular exhibition creation and visualization system, with a backend Java/Kotlin application aimed at creating museum exhibits, and a frontend C# Unity application responsible for displaying said exhibits, so that visitors can examine the exposed works.

In the backend application, originally written in Java but since ported to Kotlin, museum curators can create custom exhibitions, by selecting the works to be displayed, customizing wall, floor and ceiling textures, setting lighting and audio conditions, and choosing the number and layout of rooms, which can be connected with teleport points. Works can be placed freely around the rooms, and the entire exhibit is saved to a file and stored in a MongoDB[6] database.

The frontend application is, as the name of the system suggests and as previously stated, a Virtual Reality application written in C# and making use of the Unity engine's capabilities. The visitors can explore the entire exhibition space without much restraint either by walking or teleporting, and approach displayed works in order to examine or interact with them. Visitors can also receive visual or audio feedback based on the current exhibit, if so specified by the curators[33].

VIRTUE is an open-source project and the entirety of its source code is available on GitHub , divided between frontend and backend[7].

[6]https://www.mongodb.com/
[7]https://github.com/VIRTUE-DBIS

Figure 3.8: Screenshot of the VIRTUE Virtual Reality application[33].

3.5 Summary, Analysis and Discussion

It is important to determine which facets of the aforementioned related work were incorporated into the work to be developed during the period of this thesis.

3.5.1 Augmented Reality

Regarding Augmented Reality, the systems presented by Ramirez et al. [25] and Purnomo et al. [26] were developed in the same museum context as our framework[4]. However, as the main focus of the developed system was not Augmented Reality, these systems were not explored for insights related implementation procedures or details.

3.5.2 Virtual Reality

With respect to Virtual Reality, Mediascape XR[27] was of great technological value for the work to be carried out, since it demonstrates possible methods and use cases for implementing user interaction with 3D artefacts in a Virtual Reality environment. However, the collaborative aspect of this system was not looked into as a base for the system developed as part of this thesis, as it was decided not to focus on collaborative experiences. In addition, the testing process conducted by Saffo et. al served as a general guideline into how the VR application should be evaluated.

3.5.3 Exhibit Editing

In this regard, VIRTUE[33] is of particular interest due to how the creation of exhibitions is implemented, since this is a similar concept to what was proposed for the work to be carried out (More information in section 4.1).

3.5.4 Effects of Extended Reality

Despite not including technologies, the results of the studies mentioned in this section are relevant to keep in mind when developing the application, as they provide valuable information regarding what effects Virtual and Augmented Reality have on visitor experiences. Chung et al.'s research [30] also includes a collaboration component, in the form of live support from museum or heritage site staff, which it concludes to improve visitors' experience and their intention to re-visit the museum or heritage site again in the future.

3.5.5 Application Evaluation

Another aspect that cannot be overlooked was how the application was going to evaluated. What components would be evaluated and according to what metrics?

Most applications mentioned in this chapter included usability tests upon concluding their respective development. These tests mainly focused on user experience, with an emphasis on qualitative results, such as ease of use or visual appeal [30]. This type of testing was part of the user evaluation that was carried out during this thesis, in the form of User Experience Questionnaires.

However, some of the projects tested the technical performance of the technologies under study, as was the case with Fuguo and Zhai's application [24]. Fuguo and Zhai tested the Vuforia engine for stability and performance issues, as well as the amount of content needed for a large scale application [24]. Ramirez et al. [25] demonstrated the viability of mobile phones as AR platforms [25], while Purnomo et al. [26] experimented with different lighting conditions to determine their effect on AR tracking. While this type of testing was not conduction during the system evaluation, users were to test the developed framework for usability issues, potential bugs, and performance mishaps.

Further details concerning how the evaluation process was conducted can be found in chapter 5.

4

System Design and Implementation

This chapter provides an in depth description of the work developed over the course of this thesis. It also presents an overall summary of the system's architecture and the previous system's architecture, describing which modifications had to be carried out to enable the implementation of the new features.

4.1 Extended Platform

Since the developed system extends upon the original system, the base technologies did not change between what had already been completed in the original system and what was accomplished during the course of this thesis.

As previously mentioned in 1.3 and 1.4, the main goal of the work carried out was the extension of the existing platform by implementing additional functionalities. Due to the need to integrate these functionalities, a decision was made to shift the focus of the developed work towards the VR application and the Exhibit Editor within the Administrative Application.

4.1.1 Navigation System

The Navigation System allows museum goers to possess a basic understanding of their surroundings. It informs visitors of their current location within the exhibit, as well as the location of any nearby artefacts. The types of artefacts also need to be distinguishable among themselves to further improve this environmental awareness. Furthermore, the exhibit rooms themselves include an additional floorplan or a blueprint that may contain further information related to the exhibit, or simply the shape of the rooms themselves.

Taking inspiration from video games and modern navigation apps, an easy and effective way to implement this type of navigation system was the creation of a minimap. A minimap consists of a small navigational aid, generally in one of the corners of the interface, that displays the user's location, relative to the environment around them, generally in the form of a map[34]. In this minimap, users are be able to visualize their current position, on which the minimap is centered, as well as any artefacts around them and

their type. The user could also observe the physical layout of the exhibit room they are currently on, due to the presence of the aforementioned floorplan or blueprint.

4.1.2 Artwork Search and Browse

The fundamental purpose of a search and browse menu is to facilitate user navigation and awareness of the exhibit.

Complementary to the navigation aid previously described, this menu provides users with a list of all artefacts currently in display in the exhibit. Therein, the user can search for a specific artefact, or select one of the visible ones. Upon searching, any artworks or artefacts matching the requested search parameters are shown in the list.

Upon clicking one of the artworks being displayed on the list, said piece is highlighted on the user's minimap, and a visual and/or audio aid will guide users towards it.

A more advanced navigation solution could be implemented, such as a dynamic path that would go from the user's current location at any time to the selected artefact. However, this would have to take into consideration the geometry of the room, and would have to involve complex path calculations that might have an impact of performance, albeit minimal, depending on the device, when compared to the "simpler"solutions in the form of a visual arrow pointing towards the general direction of the artefact, at all times, or an audio cue using spatial 3D audio, which is generally less visually intrusive than the visual aid, but nonetheless directs the user towards the vicinity of the artefact.

Upon reaching the overall location of the artefact, another visual indicator indicates its exact location, by hovering over it until the user approaches it. After this, it disappears and the artefact is internally deselected.

4.1.3 Exhibit Editor

This functionality involved the creation of an "exhibition editing mode"in the Administrative application. This will allow museum creators to create a custom exhibition by importing a number of 3D rooms, in the form of .obj files, and placing them accordingly to their needs. These could then be modified in terms of dimensions and rotation, and saved on an XML file so that the entire museum configuration could be loaded later, either by the curators to perform modifications, or by visitors when visiting the exhibition.

These rooms also have, as mentioned in the Navigation System subsection, the option for an additional blueprint or floorplan in order to be represented in the navigational minimap.

The loading and saving of the rooms from and to XML files requires storing them in the Web Server, using the htdocs folder used to store user drawings, comments, and reactions. Using an identical system to the one previously developed for comments and drawings [4], the rooms' positions, rotations and scales are stored in order to be accessed by museum goers in their client applications.

After importing the custom rooms, museum staff can import custom 3D artefact objects into the scene, as well as 2D artefacts in the form of images. These can be resized and placed where it is the most relevant, and these coordinate changes can then be saved in the XML file.

4.1.4 3D Artworks

Implementing 3D Artwork support in AR can be accomplished using Vuforia's Native 3D Marker Tracking. However,the user still needs to be located in a particular angle in relation to the desired artwork to initiate tracking. However, due to the scope of the developed work not not including significant changes to the AR application, this was not investigated further.

In VR this process is much simpler, as it only involved the creation of an additional prefab to support 3D .obj objects. This type of object is natively supported by Unity itself, so no additional tools were required. Furthermore, the aforementioned Exhibit Editor was developed with 3D object support in mind in order to allow importing custom rooms and artefacts.

However, to dynamically import the .obj files and their corresponding .mtl files and textures in runtime, an external plugin was required, as well as a plugin to export copies of the .obj models in each exhibit when the exhibit is saved using the editor, so they get packaged with the rest of the exhibit instead of being dependent on models present in external storage.

4.2 System Implementation

As mentioned in section 4.1, the focus of the work developed as part of this thesis was the improvement of the existing applications by implementing additional functionalities, and the development of an Editor application within the Administrative app to allow Exhibit curators to create and edit exhibits using digital copies of real-world artefacts. This section will go into detail to describe the stages and processes undertaken throughout the entire development period.

4.2.1 Previous System Architecture

The system, as previously stated, is divided into three client applications, and a dedicated server running the web server, the Ant Media Server, and the filesystem within the XAMPP Apache Web Server, where data is stored.

Both the AR and the VR applications were developed using the Unity[1] Game Engine.

The existing AR application allows users to interact with exhibited works by providing additional information about them, in a way that would not be possible in the real

[1] https://unity.com/

environment, such as rotating the artworks, drawing on said artworks or leaving a comment related to them. The Application makes use of Vuforia[2], a Markerless Augmented Reality Engine originally developed by Qualcomm, which is responsible for displaying the virtual content over the real world markers, which in the context of this application are Bi-dimensional (2D) pieces or 3D artifacts.

Figure 4.1: Screenshot depicting the AR Application (Previous Framework).

Vuforia natively supports using 3D Objects as targets. This process, according to Official Vuforia Documentation, requires 3D models to be converted into 3D targets using Vuforia's Model Target Generator App[3]. This application checks the model for suitability and configures it for optimal tracking. The output model target still requires users to be at a particular angle when viewing, in order to initialize the tracking. After this, the user can move freely around the model.

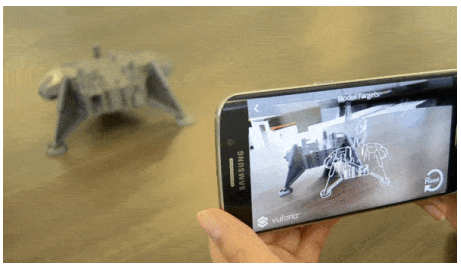

Figure 4.2: Vuforia 3D Markerless tracking example

The developed VR HMD application allows absent users to visit a virtual exhibition, thus providing a substitute experience that preserves a certain sense of presence within the virtual environment. The Application uses Meta's Oculus SDK[4], a set of tools designed

[2]https://developer.vuforia.com/
[3]https://library.vuforia.com/ objects/model-targets
[4]https://developer.oculus.com/downloads/

to enable compatibility with Meta's VR HMDs. This is what enables the application to function properly when using one of said Head Mounted Displays.

Both applications allow for an improved museum experience by displaying supplementary information related to an artwork and enabling visitors to classify exhibited pieces using a selection of *Emojis* to react, and view reactions by other users. It is also possible to leave a comment on an artwork and see comments posted by other users, as well as draw on works and view drawings made by other museum visitors, attached to their respective comments.

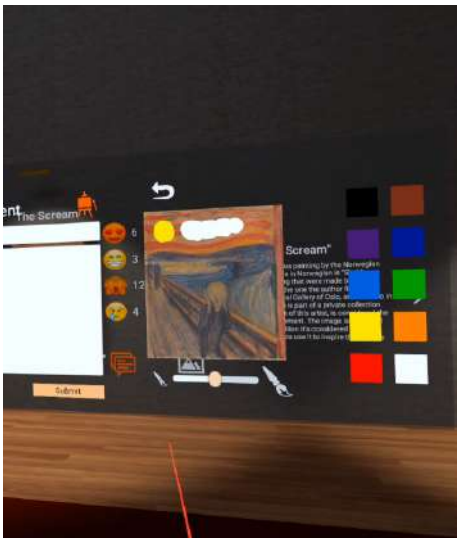

Figure 4.3: Screenshot depicting the VR Application (Previous Framework).

In addition to these functionalities, the AR application allows the user to live broadcast their visit to the media server, whereas the VR application allows users to spectate current museum visits, by watching one of the many livestreams made available by users of the AR application.

Apart from the AR and VR application, there is also an administrative desktop application which allows museum curators to monitor and moderate the current state of the system, such as comments and drawings created by users, or current livestreams.

Ant Media Server was used to enable users to livestream their visits, or watch other users' livestreams. This runs on the dedicated server machine and, since the previous server was a Windows machine, a virtual Ubuntu machine had to be created to run the server on. Ports had to be forwarded so that the Virtual Machine's ports could communicate directly with the outside world via the host machine and SSL (Secure Sockets Layer) encryption had to be enabled in order to protect data. To do this, the machines need a dedicated hostname, which can be obtained via the website "no-ip.com". With this setup complete, curators have access to a dedicated web-based dashboard where server properties can be closely monitored. Additionally, using WebRTC, the And Media Server allows curators to view current livestreams via their web browser.

Figure 4.4: Screenshot depicting the Streaming Menu within the VR Application (Previous Framework).

To store all information regarding user generated data, such as reactions, comments, drawings and their respective strokes, Lourenço made use of an XML file format. Objects such as User drawings or their individual strokes, were converted from XML to C# and from C# to XML when needed. These XML files were stored in the same VM as the Ant Media Server, making use of XAMPP, or in this case LAMPP, the Linux distribution of XAMPP. Upon setting up XAMPP, a folder called lampp was created in the path /opt/lampp/. Inside this folder, another folder called htdocs is located. This is where the files sent and received via HyperText Transfer Protocol (HTTP) requests, such as the aforementioned user generated XML files, were stored.

An overall scheme of the system architecture, as defined by Lourenço in his thesis [4], is provided below:

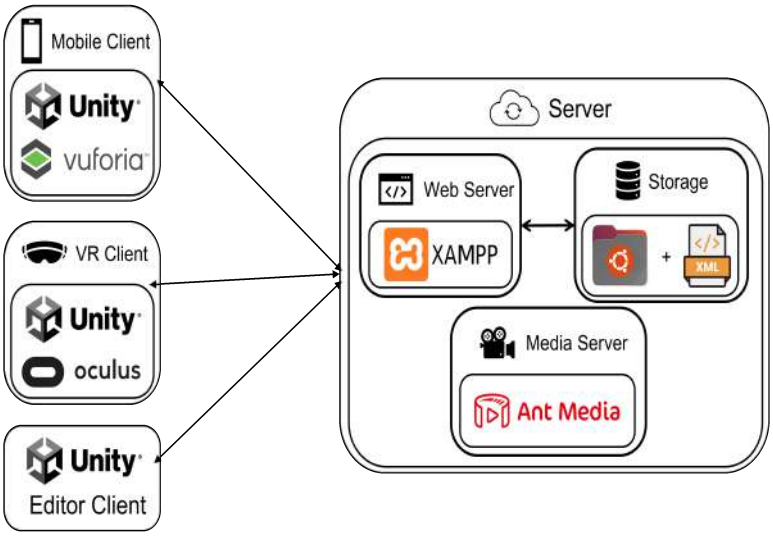

Figure 4.5: System Architecture as defined by Lourenço [4].

4.2.2 Adopted Technologies

Apart from the technologies already being used by the existing applications, it was important to take into consideration which tools were still necessary in order to fulfill the proposed requirements.

4.2.2.1 Dummiesman's Runtime OBJ Importer

Dummiesman's Runtime OBJ Importer is a special, free plugin available to download in the Unity Asset Store [5]. This plugin allows the import of a chosen .obj into unity, at runtime.

The reason this plugin was needed was due to the fact that the editor needs to handle custom .obj models at runtime, and not within the Unity editor itself, as that is a development tool the end users will not have access to when using the editor application. As such, a way to be able to import custom .obj models from external storage at runtime was required in order allow for complete editor application functionality.

4.2.2.2 Dummiesman's Scene OBJ Exporter

Conversely, it was also crucial to export the objects used within the editor as .obj files to a specific storage location within the Application's internal storage. This approach was chosen over simply copying the models being used, as it severs any dependency on external files. If the mesh and texture files are automatically stored in the same internal directory as the remaining exhibit files (such as the .XML files), this makes the entire exhibit portable, as all contents are centralized in one location. Furthermore, this approach has the added advantage of preserving any changes performed on the imported mesh, when compared to the original mesh. Thus, the exported mesh will not undergo the same changes as an external mesh when re-imported.

This is where Dummiesman's Scene OBJ Exporter [6]is relevant. This plugin, also free and available in Unity's Asset Store, exports a selected mesh present within a Unity scene as a .OBJ file, which can be stored in a specific location. In the case of the developed Editor Application, the ideal location is within the folder that contains the .XML files and the 2D artefact canvas textures.

4.2.2.3 Yasirkula's Runtime File Browser

To enable importing and exporting of rooms and artefacts, as well as opening and saving of exhibits, a file browser dialog is required so users can select the exact directory to perform their operation in. Unity contains a default solution for this issue, in the form of Native File Browsers present within the *EditorUtility* package. However, this package is

[5]https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547
[6]https://assetstore.unity.com/packages/tools/utilities/scene-obj-exporter-22250

designed exclusively for use within the Editor. As such, an existing alternative had to be procured for the release appliction itself.

Yasirkula's Runtime File Browser [7] was the chosen alternative. It was designed to resemble Windows' default File Browser, and can work without requiring the *EditorUtility* package.

4.2.3 Data Model Changes

In addition to changes to individual applications, the global data model to be altered in order to provide support for said changes.

The .XML file structure was modified with the purpose of supporting dynamic artefact instantiation, 3D artefacts, and custom rooms.

Artefacts now contain additional *type*, *model*, *height*, *width*, *and image* fields. The *type* field allows the application to distinguish between three different types of scene objects: *3D Artefacts*, *2D Artefacts*, *and 3D Rooms*. This informs the program which fields should be read and taken into consideration when assembling the scene. The *height and width* fields are used with *2D artefacts*, and correspond to its real-world height and width dimensions, respectively. *Model and image* fields link to the exact directory locations where the artefact's 3D model (in case the artefact is a *3D Object* or a *Room*) and image textures (in case the artefact is a *2D Object*) are located. *Image* fields are not required when the artefact is a 3D Object or a room, since the .OBJ importer plugin also imports the associated .MTL file and textures. It is also important to note that the only supported model files are .obj files, and the only supported image files are .jpg and .png files. However, the respective import dialogs have been preemptively filtered to only display .obj, .jpg, or .png files. This was carried out in order to prevent possible compatibility issues with unsupported file types. An updated UML diagram with the VR Application data model can be found in figure 4.6.

The editor application, within the administrative tool, was developed from the ground up, and thus it features an entirely independent and unique data model, which can be viewed in Figure 4.7. It consists of a main *editor* class and contains several component data structures to hold different types of information.

The core building block within the editor is the *Gizmo*, which informs the user of the *position*, *rotation and scale* of selected *objects* and facilitates their manipulation by the user. The *gizmo* also acts as a facilitator for interaction The *gizmo* may be handled in four different modes: select, move/translate, rotate, or scale); so that it may perform the corresponding operation at the user's request.

Additionally, it also keeps track of the user's original click point and of whether the cursor is currently being dragged by the user, to calculate the distance between its current position and the click point. Moreover, the *Gizmo* needs to store coordinate values associated with the *Scale and Rotation* of temporary parent objects, created when manipulating

[7] https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006

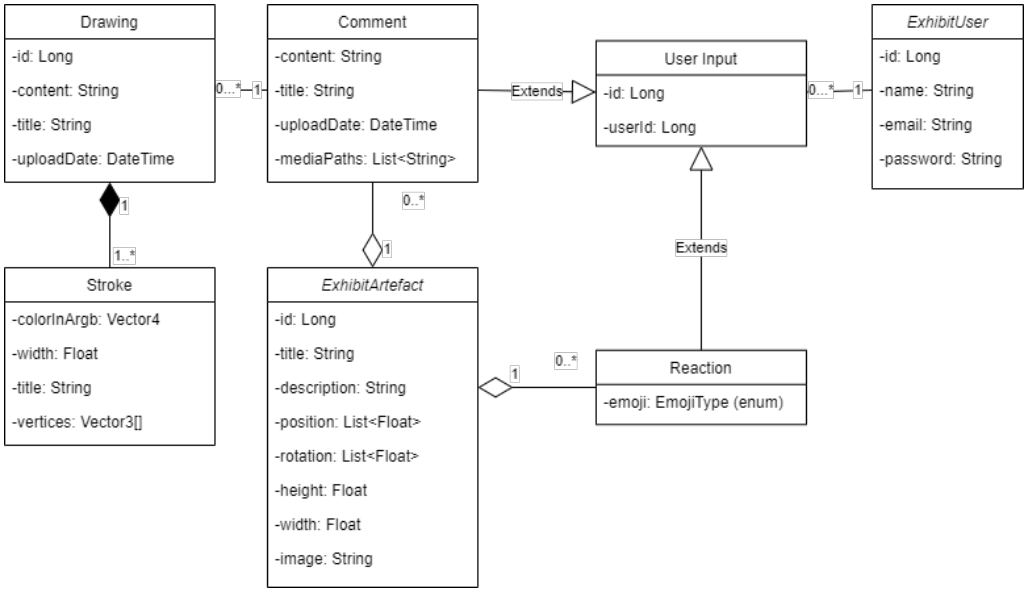

Figure 4.6: Class diagram representing how VR Exhibit data is structured.

more than one object at once. This parent object is created when the interaction starts and is destroyed when it ends. The local rotation and scale values are then passed to its children. It is also important to note that the *gizmo* is composed of three different *Axis* objects, one for each dimensional axis. Each *axis* stores a vector representing its normal, and a center point, upon which movement is processed. The reason for having three different axes, instead of using a single *object*, relates to the need to independently interact with each axis to inform the *editor* that a change on the current *object* will be applied on the selected *axis*. When the user hovers their mouse over one of the *axes*, it highlights, informing that this dimension is currently selected, allowing for better visual feedback.

Another important aspect of the editor are the *Editor Objects* themselves. Every *object* that is generated in the scene is automatically provided with an *EditorObject* class script. This script contains identifying information about the object, such as an ID, the object's title and type. According to the object's type, it can be classified either as an *Artefact Object* (divided into *3D Artefacts* and *2D Artefacts*), or as a *Room Object*. A *Room Object* contains an image that serves as a map, to be visualized in the VR Application's *minimap*, and this map's dimensions (length and width), so that the image can be properly scaled, according to the scale of the room. In contrast, an *Artefact object* contains an additional description and, if it is a *2D artefact*, two optional *height* and *width* fields.

An *ExhibitState* object represents the state of the entire scene at a specific moment in time. It is used for saving, undoing, redoing, and in clipboard actions. Inside an *ExhibitState* there is a *Dictionary* that contains Key,Value pairs pertaining to every object in the scene. The key is the *EditorObject* in the *GameObject* itself, and the value is an *ObjectProperties* class. This class stores an object's *position*, *rotation*, *and scale* coordinates.

These cannot be stored inside the *GameObject*'s transform due to how Unity handles

GameObjects. Unity treats *GameObjects* by reference, not by value. This means that, as long as an object exists in the scene, all references to it in data structures will have their values synchronized with it. To keep old values, it is necessary to store them.

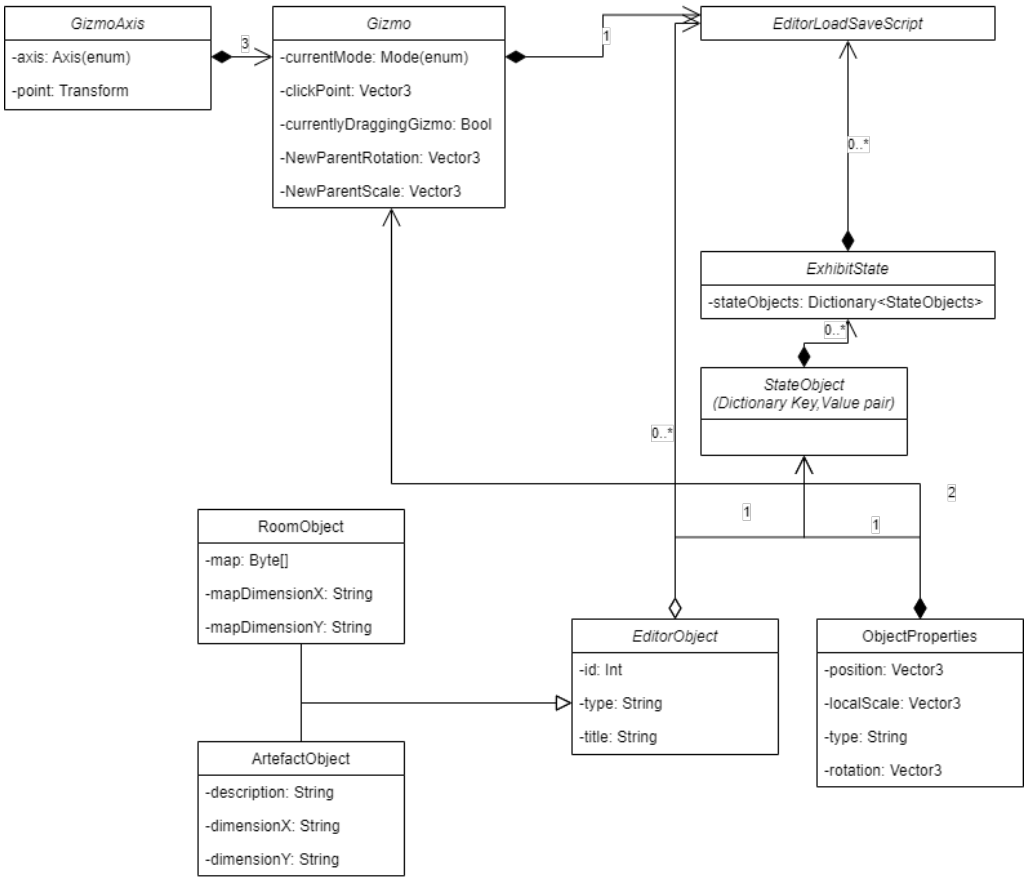

Figure 4.7: Class diagram representing how editor data is structured.

4.2.4 Remote Virtual Reality Application

To improve the overall user experience while using the VR Application, numerous features were developed and implemented.

As previously mentioned, the art pieces are instantiated dynamically based on the contents of the XML file. A "generic" art piece *prefab* was created, replacing the original *ArtPieceCanvas*, which were unique prefabs for each painting, and were manually inserted into the gallery scene. This new "generic"*prefab* includes default dimensions and no canvas texture. For every painting present in the exhibit's XML file, a clone of this *prefab* is instantiated and placed according to the existing coordinates and rotation. The *prefab* is then re-scaled according to the painting's dimensions, and the canvas is retextured with the image located in the path provided by the XML file.

To enable 3D Object compatibility, a separate *prefab* was created, similar to the new *ArtPieceCanvas* used with 2D Objects. 3D Art pieces only include the *User Content* UI

Canvas and a variation of the *"Focus Detection"plane*, as the model itself is loaded from the location described in the XML file.

A small snippet of the code that dynamically instantiates 2D Artefacts can be observed below:

```
1
2
3  foreach (ExhibitArtefact dArt in allArtefacts.exhibitArtefacts)
4            {
5                if (dArt.type.equals("2dartefact")) {
6                //properly positioning the artefact
7                GameObject createdCanvas = Instantiate(defaultPainting, ListConvert(dArt.
                     ↪ position), Quaternion.Euler(ListConvert(dArt.rotation)));
8
9                //Setting the name of the created object
10               createdCanvas.name = dArt.title + "␣Art␣Piece";
11
12               //changing artefact scale according to xml parameters
13               Transform canvasTransform = createdCanvas.transform.Find("Frame/PIC-FRAME");
                     ↪
14               canvasTransform.localScale = new Vector3(dArt.width*frameToCanvasRatio,dArt.
                     ↪ height*frameToCanvasRatio,1) ;
15
16               //setting script variables
17               ArtPiece newArtPiece = createdCanvas.GetComponent<ArtPiece>();
18               newArtPiece.setId(dArt.id);
19               ArtefactUIVisibilityHandler newCanvasVisibilityHandler = createdCanvas.
                     ↪ GetComponentInChildren<ArtefactUIVisibilityHandler>();
20               newCanvasVisibilityHandler.mainCamera = mainCamera;
21
22               Transform canvasObject = createdCanvas.transform.Find("Canvas");
23               Canvas canvasComponent = canvasObject.GetComponent<Canvas>();
24
25               //this is only needed while offline mode is enabled
26               EmojiCounterHandler emojiCounterHandler = canvasComponent.GetComponent<
                     ↪ EmojiCounterHandler>();
27               emojiCounterHandler.globalVars = globalVars;
28
29
30               canvasComponent.worldCamera = mainCamera.GetComponent<Camera>();
31
32               Transform expandHitbox = createdCanvas.transform.Find("Expand␣Hitbox");
33               Canvas hitboxCanvas = expandHitbox.GetComponent<Canvas>();
34               hitboxCanvas.worldCamera = mainCamera.GetComponent<Camera>();
35
36               //this is only needed while offline mode is enabled
37               Transform currentCommentObject = createdCanvas.transform.Find("Canvas/
                     ↪ Current␣Comment");
38               CommentUpdateHandler currentCommentComponent = currentCommentObject.
                     ↪ GetComponent<CommentUpdateHandler>();
```

```
39              currentCommentComponent.globalVars = globalVars;
40
41              IndicatorToggle canvasIndicatorScript = createdCanvas.GetComponent<
                    ↪ IndicatorToggle>();
42              canvasIndicatorScript.user = mainCamera;
43
44              //applying the canvas texture
45              GameObject canvasPainting = createdCanvas.transform.Find("Frame/PIC-FRAME/
                    ↪ painting").gameObject;
46
47              Renderer paintingRenderer = canvasPainting.transform.GetComponent<Renderer
                    ↪ >();
48              Material newMat = new Material(genericPaintingMaterial);
49
50              newMat.name = (dArt.title + "␣mat");
51              //the first line is needed for the line below it to work
52              newMat.EnableKeyword("_EMISSION");
53              newMat.SetColor("_EmissionColor", Color.black);
54
55              newMat.SetColor("_Color", Color.white);
56
57              //the first line is needed for the line below it to work
58              newMat.EnableKeyword("_NORMALMAP");
59              newMat.SetTexture("_MainTex", ImageLoad(dArt.image));
60
61              paintingRenderer.material = newMat;
62
63              }
```

The above code, as the comments describe, checks, for each artifact in the XML file, if it is a 2D artefact. If so, it then positions the artefact properly within the exhibit, according to its coordinates, by instantiating a copy of the generic *ArtPieceCanvas* Prefab on those coordinates. It sets the name of the created Unity *GameObject* to the name of the Artefact, for debugging purposes, and changes its scale according to the values present in the file. Afterwards, it sets the ArtPiece script variables to their proper values, such as object ID, the *ArtPieceCanvas* visibility handler, and the main Camera, so the artefact can be properly interacted with. If the system is currently in offline mode, it needs to sync the offline global variables with its sub-components, namely the number of reactions and its comments. Lastly, it needs to apply the canvas texture located in the path referenced in the XML file to the generic art piece canvas model. For that, a new material has to be created with specific emission properties.

To help users navigate through an exhibit and locate the works they are searching for, a new user interface that is present in both the VR application and the Administrative Application was developed. This is the *Artwork browse menu*. Upon pressing a button on their controller/keyboard (for reference, on a keyboard it is the Home Key), the user is presented with a scrollable list of all existing artworks, as well as a search box. The

user can select one of the available works on the list to locate it. Doing so will mark the selected artwork/artefact with an animated visual indicator, which also emits a 3D sound effect to help visitors locate the artwork (Figure 4.10). Each artwork present in the list is represented by a clickable button *prefab*, with an image of the artwork (using the artwork's own texture) and its title.

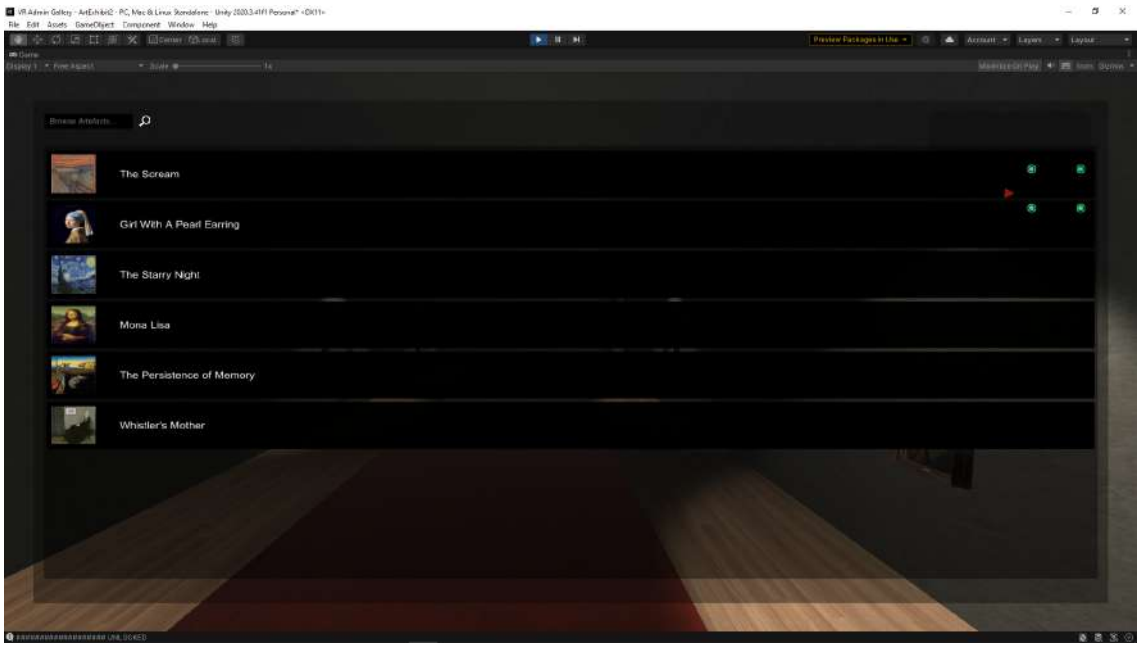

Figure 4.8: The artwork browse menu as it appears in the Administrative application.

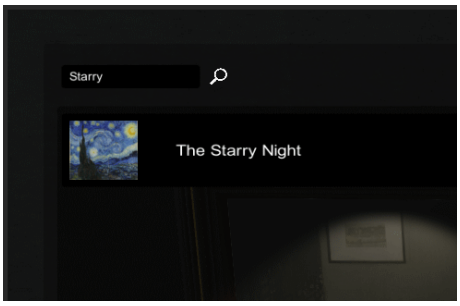

Figure 4.9: The artwork browse menu search bar with an inserted search query. Its results can be seen in the below list.

Due to limitations with *Unity*'s built-in UI tools, most of the visual behavior of the Artwork Search and Browse menu had to be manually defined. The scrollable list within the UI is cleared of any children at the start of the scene, and after every search.

The browsing function works by deleting all button *prefabs*, browsing through all active artworks, and creating a new button *prefab* for each artwork whose title matches the currently queried text. The buttons are then dynamically positioned inside the *scrollView* element, which is also re-scaled according to the number of results. The dynamic position

of the buttons had to be calculated manually since *Unity* does not provide tools for easy placement of items within a *scrollView* element (Figure 4.8).

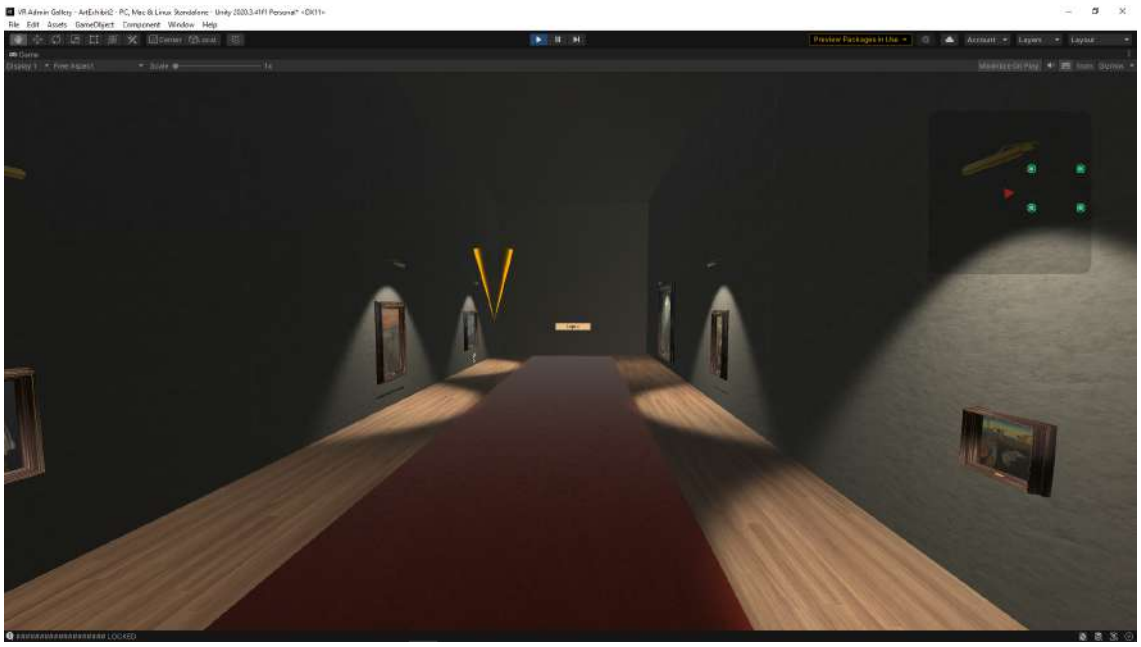

Figure 4.10: The position indicator displayed when an artwork is selected

The function that dynamically re-renders the list's items after each query is the following:

```
1
2   private void RenderItems(string query)
3   {
4
5       Transform contentPage = this.transform.Find("Background/Scroll View/Viewport/
            ↪ Content");
6
7
8
9       //auxiliary counter so it doesn't get stuck in an update loop.
10      int childCounter = contentPage.childCount;
11      while (childCounter > 0)
12      {
13
14          DestroyImmediate(contentPage.GetChild(0).gameObject);
15          childCounter--;
16      }
17
18
19      int count = 0;
20      foreach (ExhibitArtefact dArt in allArtefacts.exhibitArtefacts)
21      {
22          if (dArt.title.ToLower().Contains(query.ToLower())) {
```

```
23          //properly positioning the artefact
24          GameObject createdItem = Instantiate(uiItemButton);
25          createdItem.transform.SetParent(this.transform.Find("Background/Scroll␣View/
                ↪ Viewport/Content"));
26          RectTransform createdItemScreenCoordinates = createdItem.GetComponent<
                ↪ RectTransform>();
27          createdItemScreenCoordinates.SetLeft(5);
28          createdItemScreenCoordinates.SetRight(5);
29          createdItemScreenCoordinates.SetTop(0);
30          createdItemScreenCoordinates.SetBottom(0);
31          createdItemScreenCoordinates.localScale = new Vector3(1, 1, 1);
32          //60 = from top of dialog to beginning of the scrollable area
33          //100 = height of the item box
34          //5 = spacing between item boxes
35          createdItemScreenCoordinates.SetHeight(100);
36          //it has got to be the anchored position, otherwise it would be the global
                ↪ position and thus the count increments would not align properly
37          createdItemScreenCoordinates.anchoredPosition = new Vector3(
                ↪ createdItemScreenCoordinates.anchoredPosition.x, (-60 - ((100 + 5) *
                ↪ count)), 0);
38
39          //Adding the Art Piece texture to the button's icon
40          Transform imageElement = createdItem.transform.Find("Image");
41          Image iconImage = imageElement.GetComponent<Image>();
42
43          //ImageLoad has to be stored in an object instead of being inline since we need
                ↪  to get the height and width values of the imported image.
44          Texture2D loadedTexture = ImageLoad(dArt.image);
45
46          //Images are not Sprites, and Sprites are not Textures, but the types can be
                ↪ converted into one another. The last argument is the texture pivot. By
                ↪ having it be 0,5, it will center it within the image.
47          iconImage.sprite = Sprite.Create(loadedTexture, new Rect(0, 0, loadedTexture.
                ↪ width, loadedTexture.height), new Vector2(0.5f, 0.5f));
48
49          //increase the count
50          count++;
51
52          //Adding the Art Piece title to the button's text
53          Transform textElement = createdItem.transform.Find("Text");
54          Text iconText = textElement.GetComponent<Text>();
55
56          //Setting up the Art Piece ID within the button's identifying script
57          BrowseListItem iconScript = createdItem.transform.GetComponent<BrowseListItem
                ↪ >();
58          iconScript.setId(dArt.id);
59
60          iconText.text = dArt.title;
61
62
```

```
63          //Setting the Button Pressing Action according to the Button's Art Piece ID
64          Button iconButton = createdItem.transform.GetComponent<Button>();
65              iconButton.onClick.AddListener(delegate { ItemClick(dArt.id); });
66
67
68          }
69
70
71
72      }
73
74      //this will scale the scrollable area dynamically, acording to the number of
            ↪ existent artefacts
75      RectTransform scrollableArea = this.transform.Find("Background/Scroll␣View/
            ↪ Viewport/Content").GetComponent<RectTransform>();
76      //the extra 10 represents a bit of extra padding on the last value
77      scrollableArea.SetHeight(Mathf.Max((10 + (100 + 5) * count), 300));
78  }
```

The above code is conducting the following operations: It starts by finding the *GameObject* that corresponds to the background of the scrollable list. It then clears that list of any already existing content, rendering it clean. Next, for each artefact currently present in the exhibit, it will check if its title matches the current search query. If so, it will instantiate a new copy of a generic UI button prefab, and place it under the scrollable list. Afterwards, it modifies the button's padding and scale so it fits properly within the list. The next step will be to move it to its corresponding position within the screen, by moving it vertically (height of button) times (the object index) plus a small tolerance. After the button is correctly positioned, the code will load its corresponding canvas texture and apply it to a thumbnail image. The current button counter is increased and the artwork title is loaded into the button text. Lastly, the artefact ID will be assigned to its button ID, so it can then be used in the corresponding button pressing action, triggering the proper indicator.

Additionally, a navigational aid element, commonly called a "*minimap*", was implemented in the application's UI to provide the user with positional awareness. In this *minimap*, artworks are displayed as circles, with their color depending on the type of artwork, while the user is represented by a red arrow (Figures 4.11 and 4.12). An overall blueprint or map of the room can also be displayed, as long as the current room has been configured with a said map. This *minimap* appears on the upper right corner of the screen in the Administrative App, and as a part of the player model in the VR Application (Figure 4.10). When one of the artworks has been selected through the artwork browse menu, their minimap indicator will display a different color, to alert the user of the location of the selected work.

This minimap was developed through the use of an additional orthographic camera, whose feed is being rendered into a dedicated Material. This material is then applied to a

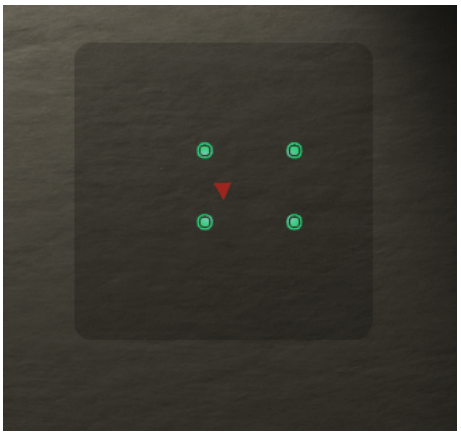

Figure 4.11: An example of the minimap present in the VR Application. The Red arrow pointing down represents the user's position and location, whereas the green dots represent unselected 2D Artefacts. The current room does not possess a custom floorplan, thus explaining the lack of background images in the map.

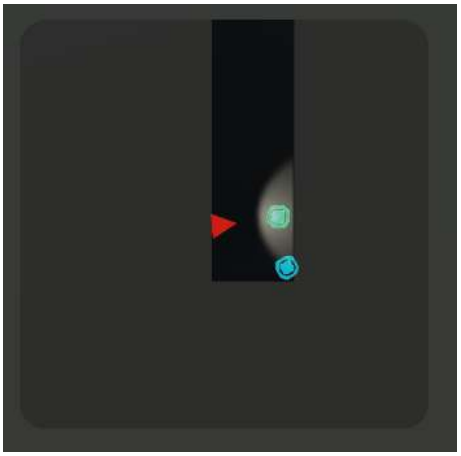

Figure 4.12: Another example of the minimap present in the VR Application. The green dots represent unselected 2D Artefacts, while the blue dots represent unselected 3D Artefacts. The Room also possesses a custom floorplan, as exemplified by the image on the map

UI Panel Element, which is located in the upper right corner of the screen, as previously mentioned. This camera is configured with a culling mask for a special "Minimap"layer, meaning that it ignores all objects on screen except the ones in this layer. The player, the room floorplans, and the artefacts themselves all have special indicators within their prefabs that belong to this special layer, thus being able to be viewed in the minimap. These indicators as not visible in the regular camera, as it has been configured to ignore (or cull) this specific layer.

4.2.5 Exhibit Editor

The Administrative Application was extensively upgraded by, apart from all changes previously mentioned in the VR Application, which were also applied in the Administrative Application, the addition of an *Exhibit Editor*, a feature rich tool that allows museum or gallery personnel to easily create and edit custom exhibits, based on real-world locations, and making use of real-world objects. These objects can be obtained either by 3D Scanning the physical artefacts, photographs, or digital replication via 3D modeling software. It is of note, however, that, as previously mentioned, only .obj 3D models are supported, as well as .jpg and .png images.

4.2.5.1 Scene, UI Layout and Keybinds

To implement this new Exhibit Editor, an entire new scene had to be created. This new scene consists of an empty object staging are, wherein the manipulated objects are placed, and a UI layer with 2 different panels: A mode selection panel, and an action panel. It is worth noting that the current mode can also be changed by pressing keys 1-4, for Select, Translate/Move, Rotate and Scale, respectively. Familiar keyboard shortcuts were also implemented to improve user workflow, such as Control+C, Control+V, Control+X, for clipboard manipulation, or Control+Z and Control+Y for Undoing and Redoing, or Control+S, Control+O and Control+N for Saving, Opening and Creating a new Exhibit, respectively. Deleting Items with Delete is also supported (Figure 4.13). The complete list of controls can be found below:

1. **Keys 1 to 4** - Change editor mode to Selection, Translation, Rotation and Scaling, respectively.
2. **Delete** - Delete an Object.
3. **Control** - Object Selection modifier. Enables multi selection or single deselection.
4. **Control + C** - Copy selected Objects, replaces current clipboard with selection.
5. **Control + N** - New Exhibit, clears entire scene if saved, keeps clipboard.
6. **Control + O** - Open existing Exhibit, replaces existing scene if saved, keeps clipboard.

7. **Control + S** - Save current Exhibit
8. **Control + V** - Pastes the current clipboard on the scene.
9. **Control + X** - Cuts selected objects, deleting the original objects and keeping the copies in the clipboard, replacing its contents.
10. **Control + Y** - Redo last undone action. Up to 5 Redos are stored at any time, and any action done after an Undo clears all Redos.
11. **Control + Z** - Undoes last action. Up to 5 Undos are stored at any time.

For an improved first user experience, *tooltips* were added to each button to clarify its function, as the .svg icons might not be intuitive for every user on their first interaction with the app.(Figure 4.14) These .svg files were taken from Google's Material Icons Library[8]. Additionally, an object's current position, scale or rotation values are displayed in a small UI component on the top right of the screen in their respective mode (Figure 4.15).

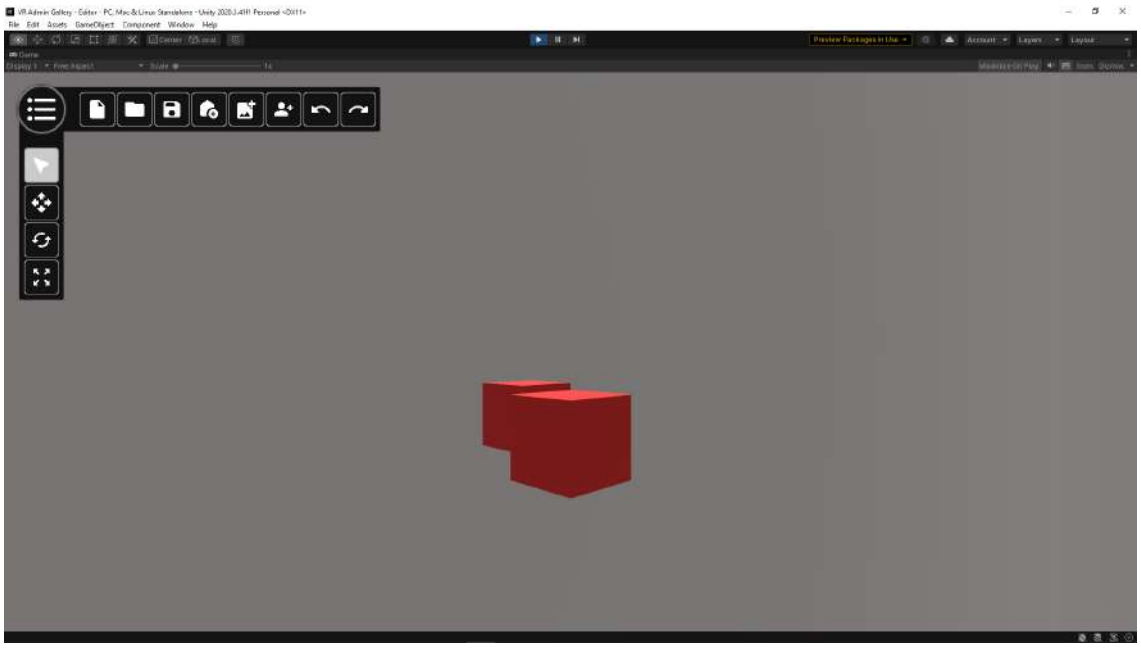

Figure 4.13: The Editor Application's UI

4.2.5.2 Object Manipulation

An important part of any editor application is the ability to manipulate and interact with objects. To this end, a custom made *"gizmo"* object was created and modelled. This *"gizmo"* is actually a set of three different *"gizmo"* objects, one for each mode (Select, Rotate,

[8]https://fonts.google.com/icons

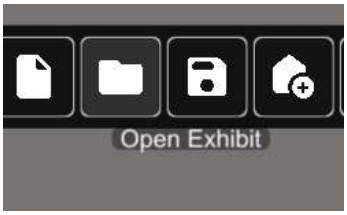

Figure 4.14: An example of UI Tooltips

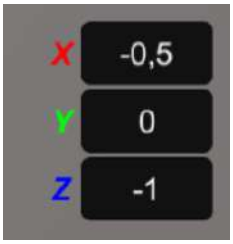

Figure 4.15: Editor's Object Coordinate Display

Move). Each one consists of a central hub and three different and mutually perpendicular *axes*. When one of the *axes* is hovered with the mouse, that *axis'* model is replaced with a larger, lighter colored one, indicating it is selected. The program is then informed of which *axis* is currently selected, so that all future movement is done relative to said axis. *Gizmos* are only visible whenever one or more objects are selected, and its scale changes dynamically in relation to its distance to the camera. This ensures its "apparent size" remains unchanged, when viewed by the user, preventing *gizmos* that are either too small and cannot be interacted with, or too large that take up the entire screen (Code snippet below this paragraph.). The *gizmo's* coordinates are the average value of the coordinates of all selected objects. This allows the user to have a common point around which all operations are executed. When clicking and dragging a *gizmo*, the corresponding action is performed along the currently highlighted axis. Clicking the hub has no effect, as it serves mostly as a pivot indicator. However, clicking and dragging the hub when in "Scale"mode will scale all dimensions equally.

```
1            scaleCoefficient = Vector3.Distance(this.transform.position, editorCamera.
                 ↪ transform.position);
2            this.transform.localScale = new Vector3(scaleCoefficient, scaleCoefficient,
                 ↪  scaleCoefficient);
```

If the current mode is "Move", whenever the mouse is held over one of the *gizmo's axes*, a *Boolean* variable is set to true. This variable tells all *gizmo*-mouse interactions to freeze until this variable is set to false. During this time, the two planes parallel to the *axis* vector are calculated and generated, as one cannot move along the perpendicular plane. Afterwards, the dot product of each of these planes' normal vectors and the camera is calculated, and smallest value is chosen. This will give us the closest plane to the camera, and thus the one along which we wish to perform our operations. Next, a ray to this

plane is cast, and the distance from the intersection point, between this ray and the plane, to the original click point that originated the dragging movement is calculated, and the corresponding value (dependent on the axis that is selected) will be summed to all of the selected objects' position vectors. As the *gizmo* moves automatically to the geometric center of all objects, it requires no coordinate calculations whatsoever. Upon finishing the action, the *Boolean* variable is reverted to false so that all *gizmo*-mouse interactions can resume, and the object's new coordinates are rounded down to prevent precision errors from successive movements (Figure 4.16).

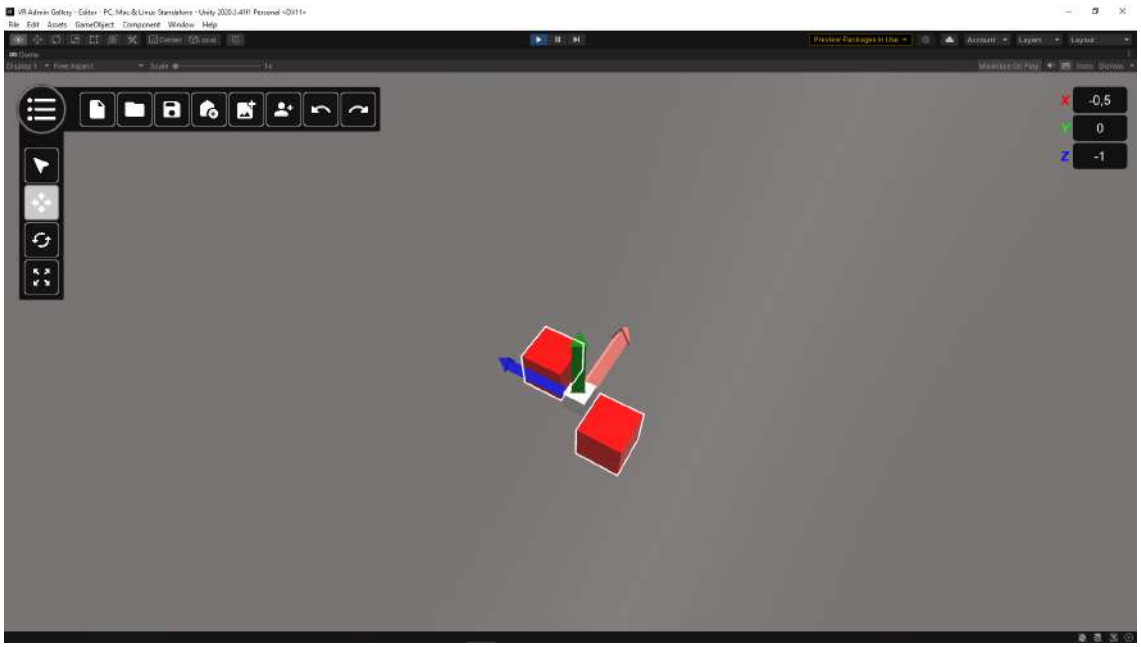

Figure 4.16: Example of Objects being moved while in Moving mode.

A code snippet demonstrating the vector calculations required to calculate the parallel planes and calculate the closest plane to the camera is shown below:

```
1  Plane[] planeCalculation(GizmoAxis axis)
2      {
3          Transform BluePoint = axis.transform.parent.Find("BlueAxis/Point");
4          Transform GreenPoint = axis.transform.parent.Find("GreenAxis/Point");
5          Transform RedPoint = axis.transform.parent.Find("RedAxis/Point");
6          Transform HubPoint = axis.transform.parent.Find("Hub/Point");
7          Plane RedBlueAxisPlane = new Plane(RedPoint.position, HubPoint.position, BluePoint.
               ↪ position);
8          Plane GreenBlueAxisPlane = new Plane(GreenPoint.position, BluePoint.position,
               ↪ HubPoint.position);
9          Plane RedGreenAxisPlane = new Plane(RedPoint.position, GreenPoint.position,
               ↪ HubPoint.position);
10         switch (axis.axis)
11         {
12
13             case Axis.Blue:
```

```
14              return new Plane[] { RedBlueAxisPlane, GreenBlueAxisPlane };
15
16          case Axis.Green:
17              return new Plane[] { RedGreenAxisPlane, GreenBlueAxisPlane };
18
19          case Axis.Red:
20              return new Plane[] { RedBlueAxisPlane, RedGreenAxisPlane };
21
22
23          default:
24              return new Plane[] { RedBlueAxisPlane, GreenBlueAxisPlane,
                    ↪ RedGreenAxisPlane };
25
26      }
27  }
28
29  Plane closestPlaneToCamera(Plane[] planes)
30  {
31      Transform BluePoint = hoveredAxis.transform.parent.Find("BlueAxis/Point");
32      Transform GreenPoint = hoveredAxis.transform.parent.Find("GreenAxis/Point");
33      Transform RedPoint = hoveredAxis.transform.parent.Find("RedAxis/Point");
34      Transform HubPoint = hoveredAxis.transform.parent.Find("Hub/Point");
35      float biggestValue = 0;
36
37      Plane returnPlane = new Plane();
38      foreach (Plane p in planes)
39      {
40          //Since all planes have the same norm, which is the norm of the plane's normal,
                ↪  and the camera is always the same, this means that, the bigger the Dot
                ↪  product, the closer to facing the camera a plane is.
41          float dotProduct = Vector3.Dot(p.normal, editorCamera.transform.position -
                ↪ HubPoint.position);
42          //in case we are on the other side of the plane, the calculation must view the
                ↪ inverted plane as if it were its regular version.
43          float dotProductInverted = Vector3.Dot(p.normal, HubPoint.position -
                ↪ editorCamera.transform.position);
44          if (dotProductInverted > dotProduct)
45          {
46              dotProduct = dotProductInverted;
47          }
48
49          if (dotProduct > biggestValue)
50          {
51              biggestValue = dotProduct;
52              returnPlane = p;
53          }
54      }
55
56      return returnPlane;
57  }
```

Object Rotation, due to its non-Cartesian nature, cannot rely on relative distances to determine the angle of movement of an object. Instead, objects must rotate a certain number of degrees around a common center, or pivot. To achieve this, objects are temporarily stored inside a parent container "Object"for the duration of the rotation, and the "distance" value being used by the program is the Signed Angle between the original click point and the current hit point, in relation to the currently selected *axis*. The parent object containing all selected objects is then rotated around its own center, along the currently selected *axis'* normal vector, "distance" degrees (As with translation, this angle is then rounded down to prevent precision errors) (Figure 4.17).

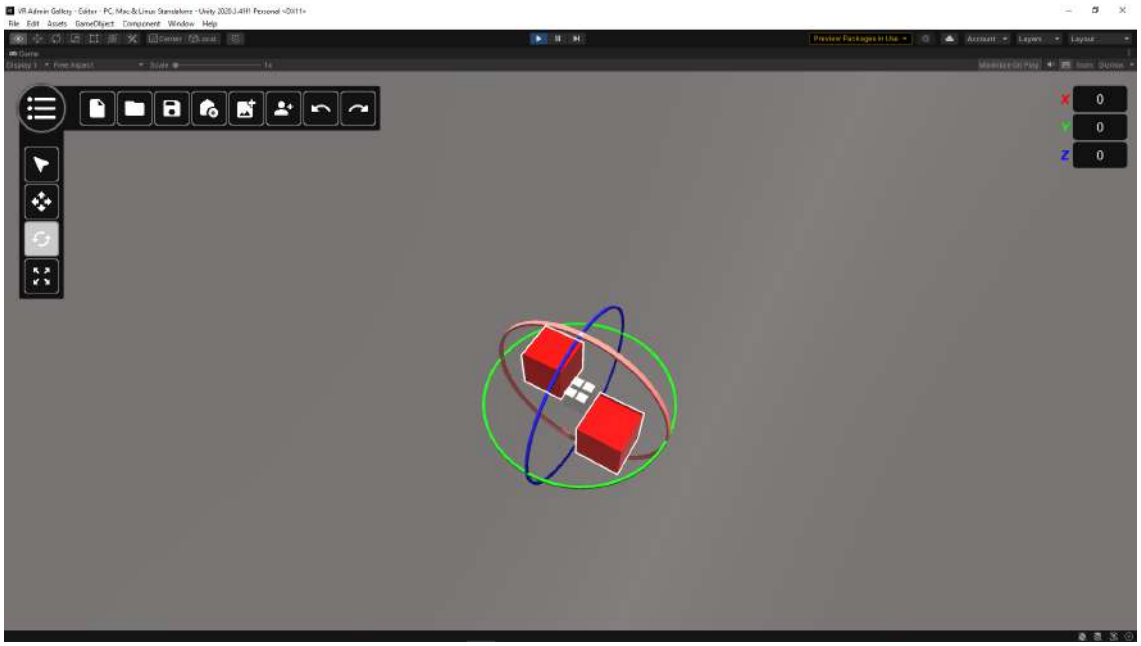

Figure 4.17: Example of Objects being rotated while in Rotating mode.

The following code sample describes the distance equation:

```
distance = Vector3.SignedAngle(clickPoint, hitPoint, hoveredAxis.transform.GetComponent<
    ↪ GizmoAxis>().Forward());
```

The following code sample describes the transform rotation applied to the parent transform:

```
    case Axis.Blue:
                    clickPoint = hitPoint;
                    newParentRot.transform.RotateAround(this.transform.position, new
                        ↪ Vector3(0, 0, 1), Mathf.Round(minimalDistance));
                    break;
```

Object Scaling works in a similar fashion to Object Moving, in that a 3-*axis gizmo* uses the distance between the original click point and the current dragging point. However, just like Object Rotating, all objects are placed in a temporary container object that scales all of them in relation to each other. Additionally, Objects cannot get scaled to 0

or negative values due to how *Unity* processes objects' relative positions to each other. (Figure 4.18). In theory, it would be possible to define a margin of tolerance where a negligibly small positive value would transition over to a negligibly small negative value, thus avoiding the zero value. The reason for *Unity*'s failure in recognizing small values is not a flaw with *Unity* itself, but rather a problem within mathematics. If two 1 meter cubes are 1 meter away from one another, if this distance is scaled down to infinitesimally small values, upon re-scaling both cubes will retain their original size, since *Unity* handles object scales differently, but their coordinates will be rounded down to the same position. This would mean that both cubes will end up in the same position. Thus, any scale that passes through the value zero will suffer from this issue.

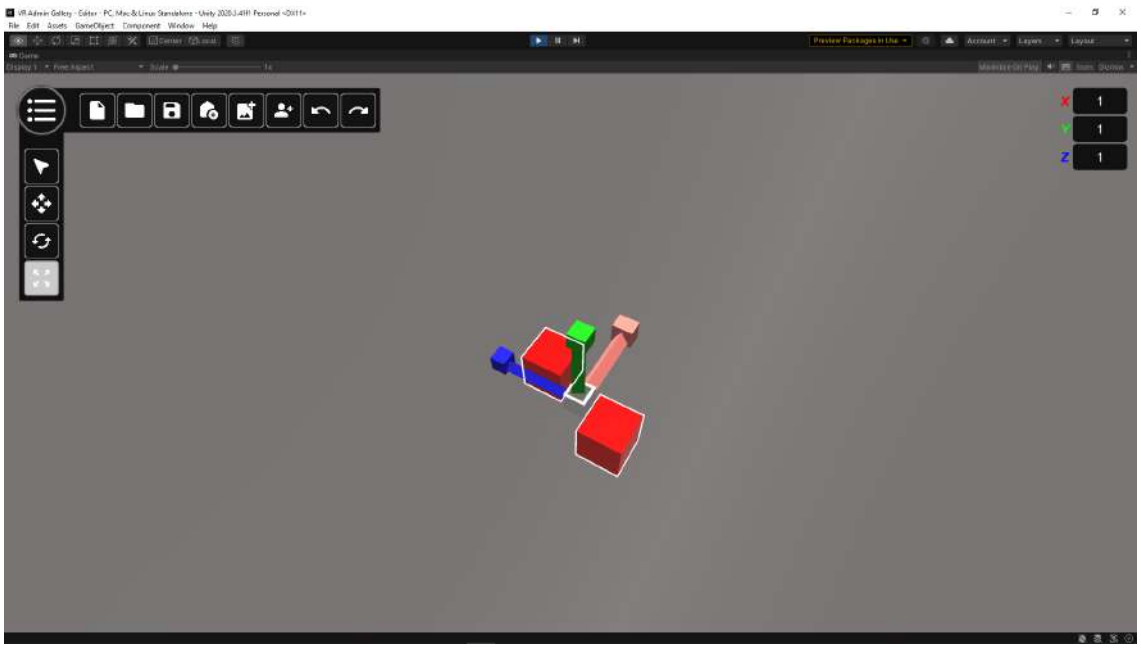

Figure 4.18: Example of Objects being scaled while in Scaling mode.

4.2.5.3 Object Selection

Another essential component to being able to manipulate objects is the selection of those objects itself. To select an object, the user must click it. The object is then added to an array of *SelectedObjects* until it is deselected. Per standard editor behavior, if no modifier key is pressed, any other selected objects are deselected. If the same object is pressed again, it is deselected and removed from the *SelectedObjects* array. If the "void" (that is, if the ray is cast over a practically infinite distance) is hit, all selected objects are deselected, and the *SelectedObjects* array is cleared. It is worth noting that a special mask was applied on the UI to ensure UI clicks are not registered as "void"clicks.

An important aspect of selecting objects is giving the user visual feedback of which objects are currently selected. When an object is selected, a thin, white outline appears around the object, being visible over overlapping objects. This ensures that the user is

always aware of the selected object's location in the scene, even if said object is hidden behind other objects. This effect was achieved using a free Outline shader from the *Unity Asset Store*, developed by Chris Nolet. Whenever an object is imported into the scene, an *Outline* script is added to its *transform*, pre-configured with the required values. This script is disabled by default since newly created objects are not automatically selected, although this might be a possible future addition. All objects are also provided with an *EditorObject* script class, a Data Structure that stores identifying information about the objects in the scene, such as their unique *ID*, *type*, *title*, *dimensions*, and several other types of data. Upon clicking on an object, if this object is not selected, a method in the *EditorObject* class called *Glow()* is called, enabling the inactive *Outline* Script, outlining the object in the scene. Conversely, if the object is already selected, a method called *Unglow()* is called, disabling the object's *Outline* Script, thus visually removing the outline from the object (Figure 4.19).

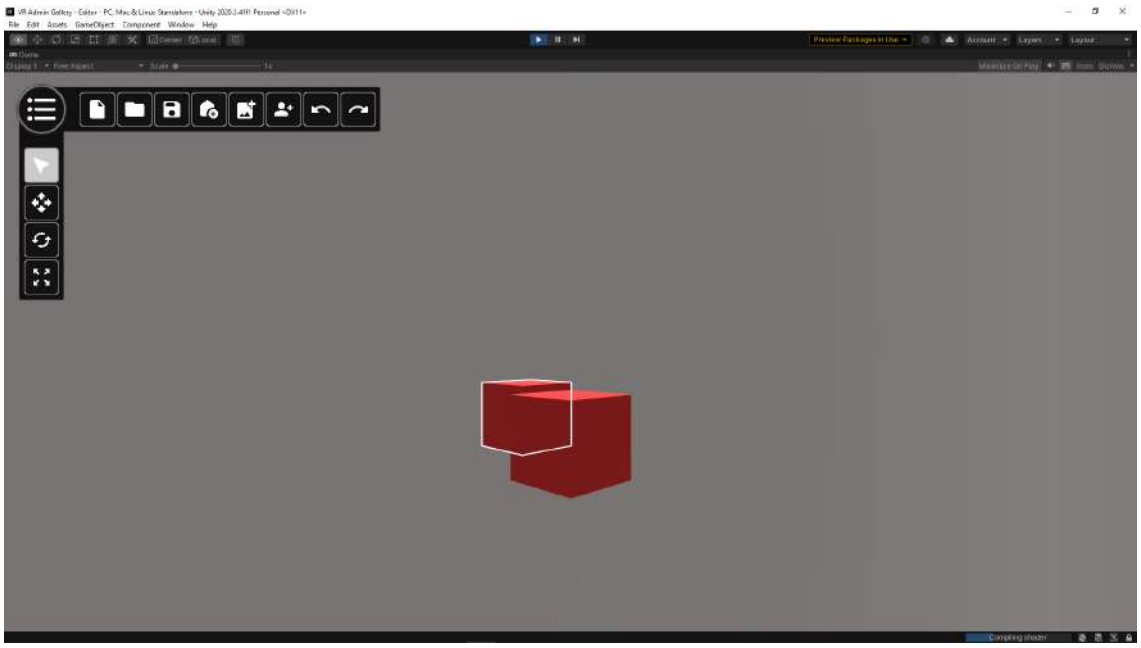

Figure 4.19: A selected object being visible behind another object due to its outline.

```
1    public void Glow()
2    {
3        this.transform.GetComponent<Outline>().enabled = true;
4    }
5    public void Unglow()
6    {
7        this.transform.GetComponent<Outline>().enabled = false;
8    }
```

To select multiple objects, per standard keybinds, the Control Key must be held while

clicking objects. *Objects* selected using this mode are appended to the end of the *SelectedObjects* array, while *Objects* deselected using this mode are removed from the *SelectedObjects* array. If more than one object is selected, selecting one of the selected objects without holding the Control Key will deselect all other objects, leaving the clicked object as the only remaining selected object. Reversely, clicking one of the selected objects while holding the Control Key will result in that object being the only object to be deselected (Figure 4.20).

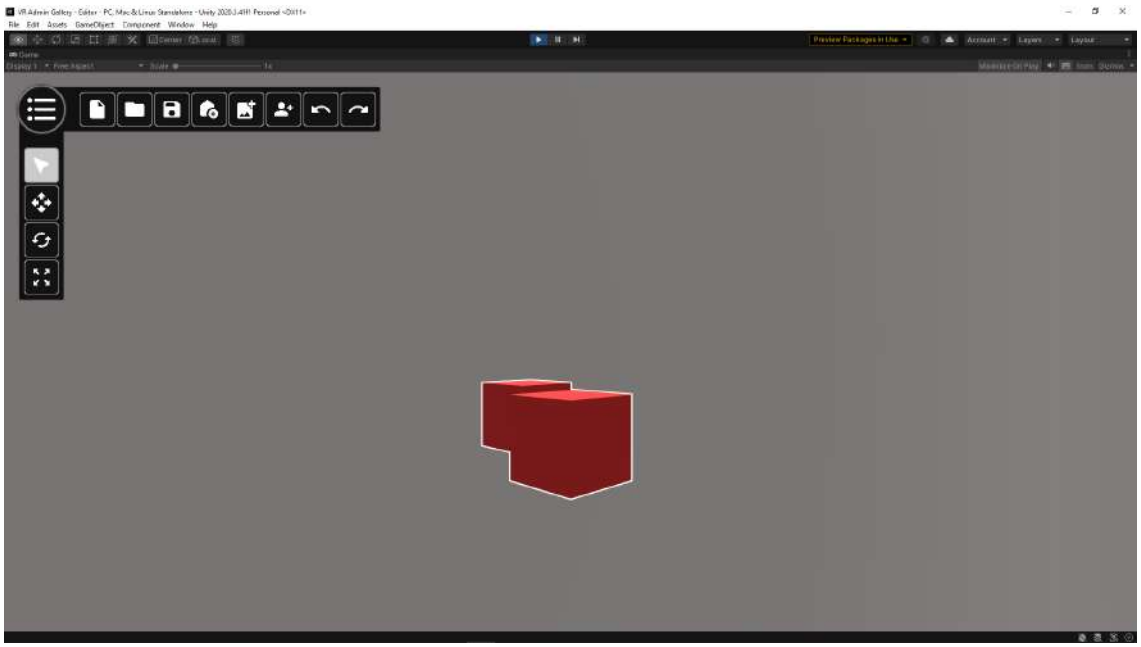

Figure 4.20: Two selected objects. Their outlines fuse into a single shape.

4.2.5.4 Undoing and Redoing

In addition to basic object manipulation and selection, the ability to correct one's mistakes through the use of Undos and Redos is of utmost importance in any editor application, as it considerably improves workflow by saving users time and effort, as unintended changes can be reverted easily, and these reversions can be discarded if proven misguided.

To implement this Undo/Redo functionality, which can be observed upon clicking the Undo and Redo buttons or, as previously stated, by pressing Control + Z and Control + Y, respectively, a set number (5) of *System States* are stored in 2 arrays, one for the previous *System States* (Undos), and another for future *System States* (Redos). These *System States*, under the class name of *ExhibitState*, store all objects in the scene in a *Dictionary*, whose keys are an object's properties. These *ObjectProperties* are data structures that contain an object's *position*, *scale*, *and rotation in the scene*, as well as its *type*. It was necessary to store these values individually since in *Unity* the *GameObjects* themselves are passed through reference, which means that their coordinate values would change. Storing them in memory allows us to access the values at a later date.

Immediately after each object manipulating step, or before an advanced object operation such as cutting, pasting, deleting, and importing objects, a complex chain of events takes place to store the current state in the Undos array. Firstly, as aforementioned, all of the object's position coordinates are rounded down to prevent precision errors that might occur from successive movements, long movements, or composite movements (scaling a rotated object). Subsequently, each active object in the scene is "cloned"and stored inside the same *ExhibitState*. "Cloning"in this particular context means that every object in the scene is precisely duplicated, thus keeping the same attributes and properties. Cloning is required so the original object can be deleted when needed. This state is then stored inside the Undos array, and any existing Redos are cleared, as changing the scene after Undoing will clear any Redos since the latest state of the scene has changed. If the array of Undos is full, it will act like a queue, where the first Undo will be discarded, and the new Undo will occupy the last index of the array.

When Undoing, the current state is saved to the Redos array so it can be Redone later if needed. All scene objects are deleted and removed from the list of *Selected Objects*, *Room Objects*, and *Artefact Objects*. The last Undo *ExhibitState* is then fetched and "unpacked". The process of unpacking works as follows: For every object in the *ExhibitState*, said object is activated, turning it visible. Next, the corresponding *ObjectProperties* are retrieved. From these properties, all necessary coordinate values are extracted, and applied to the object. Lastly, depending on the type of object, it is added to the *Artefact List*, or *Room List*.

A similar but opposite process occurs upon Redoing, where the previous state that was Undone is restored and the current state, after the Undo, is stored back in the Undos array.

The following code block represents the main Undo() function. Auxiliary methods are not shown:

```
1  public void Undo()
2    {
3       if (!dialogLock)
4       {
5          if (numberOfUndos > 0)
6          {
7             //create a redo savepoint with all objects in the scene (and add it as the
                  ↪ last redo in the redo list)
8
9             numberOfSteps++;
10            List<GameObject> allObjects = roomObjects.Concat(artefactObjects).ToList();
11            ExhibitState currentState = new ExhibitState(allObjects.ToArray(),
                  ↪ numberOfSteps);
12            AddRedo(currentState);
13
14
15            //delete all objects in the scene
16            selectedObjects = new GameObject[0];
```

```
17              gizmo.UpdateObjects(selectedObjects);
18              roomObjects.Clear();
19              artefactObjects.Clear();
20
21              Transform objectsStage = this.transform.Find("Objects");
22
23
24
25              //auxiliary list so it's easier to check for "activeness"
26              List<GameObject> objectStageChildren = new List<GameObject>();
27              for (int i = 0; i < objectsStage.childCount; i++)
28              {
29                  objectStageChildren.Add(objectsStage.GetChild(i).gameObject);
30              }
31              foreach (GameObject obj in objectStageChildren)
32              {
33                  if (obj.activeInHierarchy)
34                  {
35                      string objname = obj.name;
36
37                      DestroyImmediate(obj);
38
39
40                  }
41              }
42
43              gizmo.UpdateObjects(selectedObjects);
44
45              //get the last undo
46              ExhibitState lastUndo = GetLastUndo();
47
48              //for every object in the undo, create it under the staging area and
49              //for every object, check its type, and add it to the correct list
50              foreach (GameObject obj in lastUndo.ReturnState().Keys)
51              {
52                  obj.SetActive(true);
53                  obj.transform.SetParent(objectsStage);
54                  ObjectProperties objProperties = lastUndo.ReturnState()[obj];
55
56                  obj.transform.position = objProperties.GetPosition();
57                  obj.transform.localEulerAngles = objProperties.GetRotation();
58                  obj.transform.localScale = objProperties.GetScale();
59                  string objType = objProperties.GetObjectType();
60                  if (objType.Trim().ToLower().Equals("2dartefact") || objType.Trim().
                        ↪ ToLower().Equals("3dartefact"))
61                  {
62                      artefactObjects.Add(obj);
63                  }
64                  else
65                  {
```

```
66                      roomObjects.Add(obj);
67                  }
68              }
69
70          }
71          UpdateCurrentState();
72      }
73  }
```

4.2.5.5 Clipboard Operations

The functionalities mentioned thus far allow for basic manipulation and interaction within a fixed set of objects. They do not, however, change the number of objects present in the scene (at least, not visibly). Nonetheless, a user might want to duplicate an object by copying and pasting it, or remove it from the scene entirely by cutting it or deleting it, the former allowing for future recovery by the means of pasting said object, if it is present in the *clipboard*.

To make this possible, 4 object operations were implemented in the editor: Deleting, Copying, Cutting and Pasting Objects. As previously mentioned near the start of this section, they can be performed using standard Keybinds, those being Delete, Control + C, Control + X and Control + V, respectively. It also involved creating a new *clipboard* data structure that stores *gameObjects* and their *type*, so that they can be added to their respective lists afterwards.

Deleting is the simplest of the 4 operations. It creates an Undo step, with all objects in the scene in the shape of an *ExhibitState*, and deletes all *selected objects*. Depending on the *type* of object, it is removed from the appropriate scene list as well.

Cut and Copy are essentially the same operation, with the distinction that Cut also deletes all *selected objects* in the end, and thus creates an Undo step. The methods clear the existing *clipboard*, create new *clipboard* copies of all *selected objects*, add them to the respective *roomList* or *artefactList*, and disable them.

The Paste operation, compatible with Undos and Redos, starts by saving the current *exhibit state* to an Undo. Then, for each object in the *clipboard*, it will:

1. Duplicate it (so that unlimited pastes can be made from a single copy).

2. Activate/Enable it.

3. Set its *ID* to the largest unused *ID*.

4. Check the object's *type* and add it to the respective list.

5. Move the pasted objects by (1,1,1) in relation to the original object's position, so that the user has some visual feedback of existence of the new object.

This last step is not standard editor behavior, and thus could be replaced with simply selecting the newly pasted object.

An example of multiple copies of the same object being manipulated in different ways can be found in Figure 4.21.

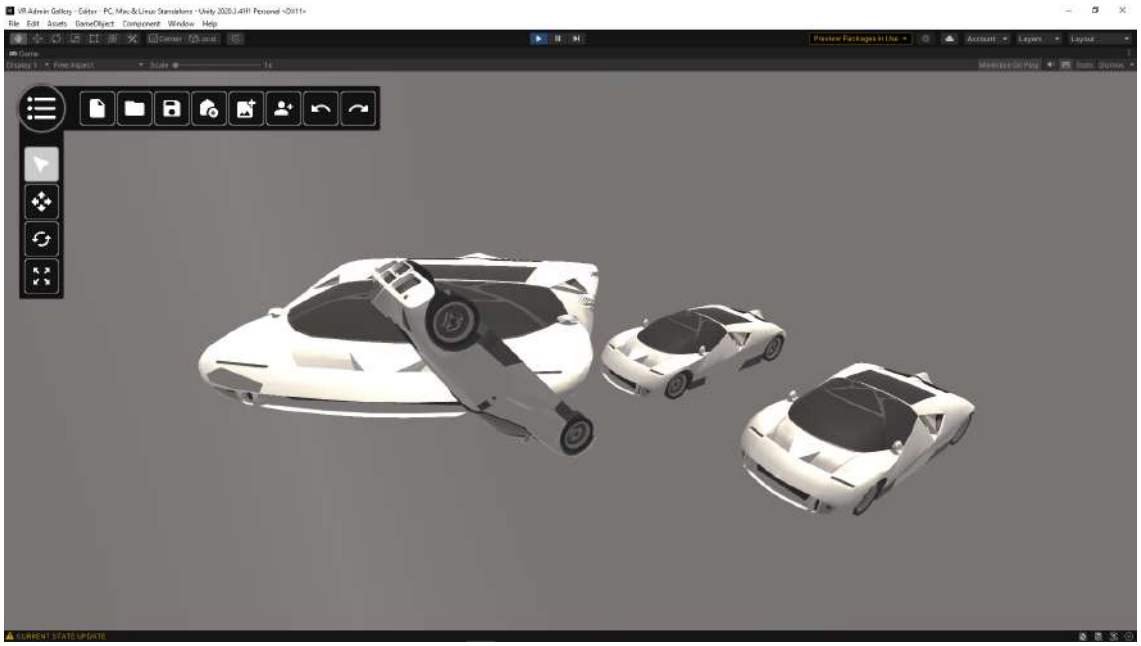

Figure 4.21: Several copies of the original object, manipulated in distinct ways such as translation, rotation, and scaling.

4.2.5.6 Global Operations

To ensure exhibit changes are preserved, methods that track the global state of the Exhibit and allow the user to interact with the XML files used by the VR and AR applications are required. To this end, the following operations were implemented: New Exhibit, Open Exhibit, Save Exhibit.

Creating a New Exhibit can be done by pressing Control+N, or by clicking the "New Exhibit"button. If the scene has been previously saved, the entire scene is cleared, the Undos and Redos are cleared, along with the *selectedObjects*, *roomObjects* and *artefactObjects* lists. The editor mode is reverted to select, and the camera position is reverted to its default state. Per standard behavior, the *clipboard* is preserved, so that its contents can be used on a new exhibit. Otherwise, the user is prompted with a dialog window asking whether they want to save the current exhibit before creating a new one.

Opening an Existing Exhibit, similarly, can be accomplished by clicking the respective button, or by pressing Control+O. A system explorer dialog will appear, prompting the user to select an .XML file to open. Upon selection, a similar process to creating a new exhibit takes place, where the scene is cleared, the Undos and Redos are reset, and all Object lists are cleared. Next, the .XML file is read. For each object entry in the .XML

file, a new *transform* is created with its coordinate values. The object is then added to its respective list, depending on whether it is an *artefact Object* or a *room Object*. *Artefact objects* are further discriminated into *2D objects and 3D objects*. If it is a *2D object*, such as a painting, a document or a photograph, its dimensions are taken into consideration and its texture is loaded from the local path specified in the .XML file. Conversely, if it is a *3D object*, the .obj mesh and textures are loaded from the local path. This .obj mesh import was done using Dummiesman's Runtime OBJ Importer, a free plugin asset from *Unity's Asset Store* [9].

Saving an Exhibit, in accordance with standard keybinds, is achieved by pressing Control+S, or by clicking the "Save Exhibit" button. The user is then prompted with a system explorer dialog similar to the one shown when opening an existing exhibit. Upon saving the exhibit, the current *ExhibitState* is saved into a local variable containing the currently saved *ExhibitState*. This will keep track of which *Exhibit State* has been saved, preventing the user from losing any progress by displaying a warning dialog box whenever the user tries to close, open or create a new exhibit. For every object present in the scene, a serialized entry is created in an .XML file, containing its type, ID, properties like name and description, and its coordinates in the scene. Lastly, if it is a 3D Object, its .obj mesh is exported into local storage, together with the textures used in said mesh. If it is a 2D Object, the 2D Canvas texture is exported. These resources are then linked in the .XML file, so that they can be re-imported when opening a scene. Exporting .obj meshes is possible thanks to another free plugin asset from Dummiesman, Scene OBJ Exporter [10].

To compare exhibit states, a new "isEqual"method was created, in order to properly check for similarities among key variables in both states. This method checks if both states have the same number of objects. Then it proceeds to verify whether the names of all objects present in one of the states are the same as the names on the other state. Afterwards, it will check if all objects in both states have the same editorObject script, which contains editor related qualitative information about the object, such as title, description and ID. Lastly, it will compare the objectProperties within each state. These objectProperties contain quantitative information about each object, such as its position, scale and rotation coordinates.

If all these checks pass, then both states are equal.

4.2.5.7 Object Exporting and Importing

The final core component of the editor application is the possibility of importing custom 3D or 2D models. Since the editor itself does not provide the user with premade sample objects, it is essential for the user to be able to import models of exhibit rooms or artefacts in order to be able to set up a virtual exhibit. To fulfill this requirement, it was necessary

[9] https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547
[10] https://assetstore.unity.com/packages/tools/utilities/scene-obj-exporter-22250

to create 3 similar methods, one for each type of importable object, a *3D Room Object*, a *3D Artefact Object*, and a *2D Artefact Object*. Fundamentally, these three methods function in a similar fashion. However, it is important to highlight some key differences between the three of them.

Taking *3D Artefact Object* importing as a baseline example, the user can import this type of objects by clicking the appropriate button on the UI. An explorer dialog window appears. and the user is prompted to select an .obj file. For better results, it is recommended that the .mtl file and textures be in the same directory location as the .obj file. If successful, an undo step is created, the object is imported into the scene, using Dummiesman's script, and given *editorObject* and *outline* scripts. The *editorObject* script will be created with a brand-new *ID*. A new *mesh collider* is created dynamically in the top-level mesh, in case the .obj is composed of several sub-meshes. If all meshes are at the same level, the first mesh is used. (Figure 4.23) Afterwards, an in-editor dialog appears and freezes all functionality until the confirm or cancel buttons are pressed. In this dialog the user can, depending on the type of Object, input information about the it. In the case of a *3D artefact object*, the user can input the name and description of the imported object. Upon confirming the dialog, the imported object is renamed to what was input in the dialog. Furthermore, the *EditorObject* class stores all required data, such as title, description, image dimensions, and image data (when applicable). This is also saved when the dialog is confirmed. Canceling the dialog undoes the object importing and clears the redos (Figure 4.22).

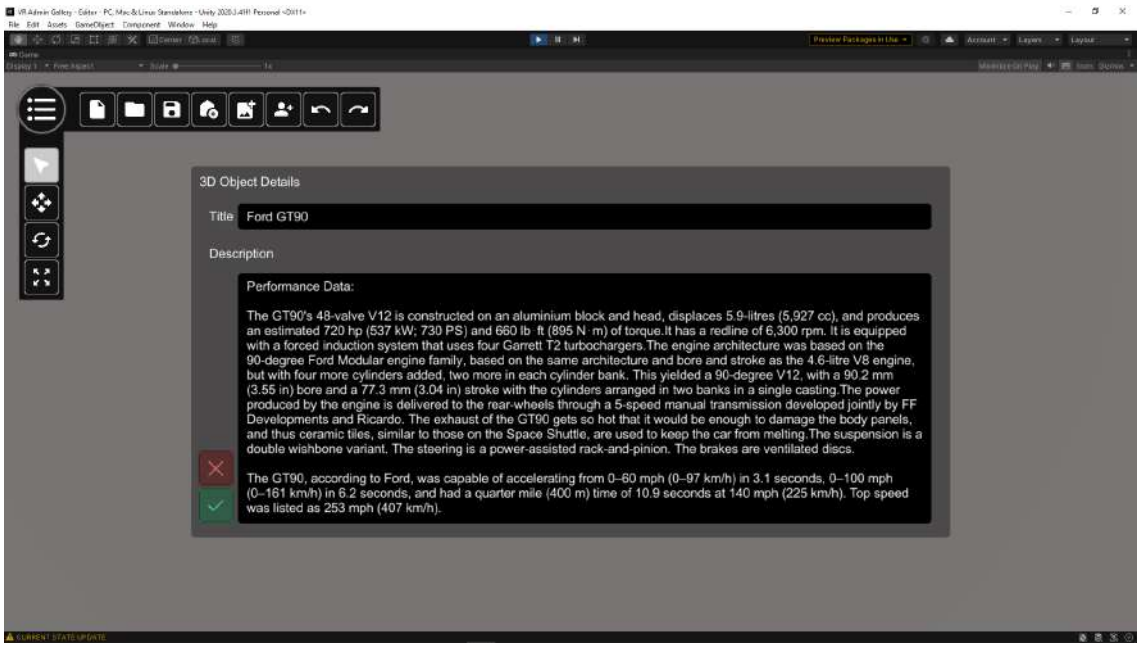

Figure 4.22: The editor dialog that is displayed when Importing a 3D Artefact Object.

As stated before, importing *2D Artefact Objects* and *3D Room Objects* is not too dissimilar from importing 3D artefact objects. However, due to the inherent properties of

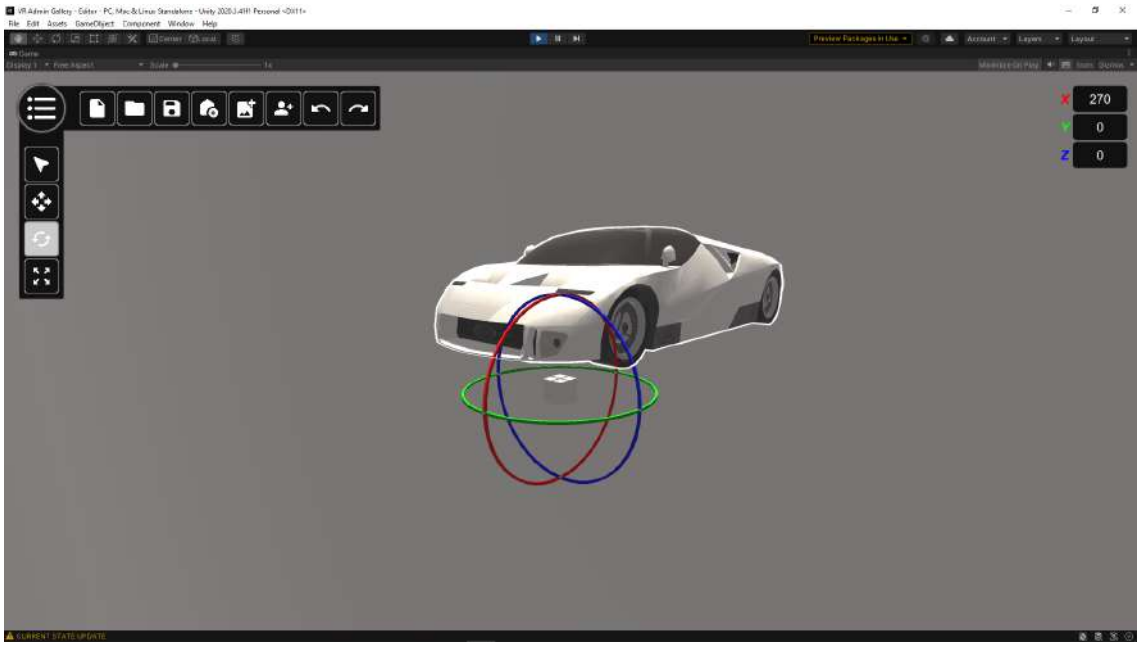

Figure 4.23: An example of an imported 3D Artefact Object.

these types of objects, some modifications had to be performed on the importing process a whole, but also on the import dialog specifically, since the types of data the user needs to input are different.

When importing *2D Artefact Objects*, similarly to what occurs in the main scene when loading the artworks from the .XML file, the desired image, which can be either a .png or a .jpg type of image, is loaded into a texture, that is then applied to a generic canvas-like object (Figure 4.25). The Artefact Import dialog that is displayed upon importing is different from the one used when importing *3D artefact objects*. In addition to title and description, the user must input the object's real-world height and width in centimeters. If no values are introduced, a value of 1 meter is used (Figure 4.24). The artwork is then re-scaled according to these values (although the user can manually re-scale it again later).

The process of importing *3D Room Objects* is the same as importing *3D Artefact objects*. The only differentiating factor being the Import dialog that is displayed. In this Dialog, a small window is shown, with a button that says, "Add Room Plan"(Figure 4.26). This is where the user can supply a complimentary map image, so that it can serve as a basis for the room as seen on the *minimap* in the Exhibit Scene. This image is textured into a plane that is placed at the center of the imported room object. Because of this, the image and the 3D object must have a common center when imported so that the map aligns with its respective room (Figure 4.27). The user can also adjust the dimensions of the room in 2 input fields, to scale the image according to the real life dimensions of the room. If nothing is input, default values are assumed.

The following code block represents the process of importing a 3D Artefct object into

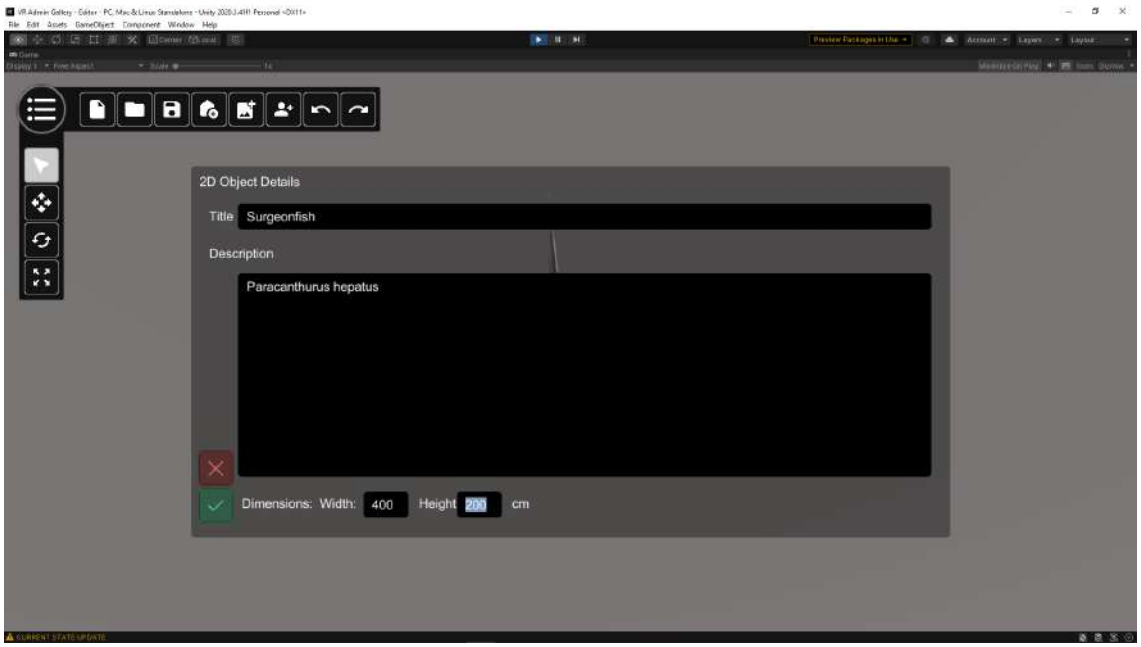

Figure 4.24: The editor dialog that is displayed when Importing a 2D Artefact Object.

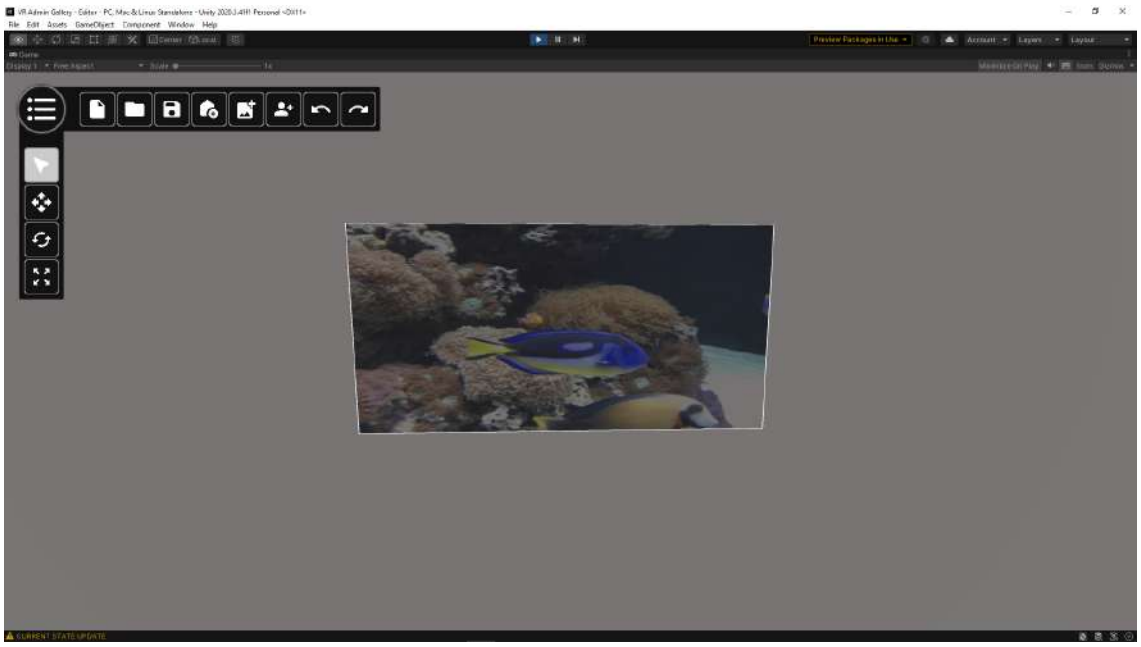

Figure 4.25: An example of an imported 2D Artefact Object.

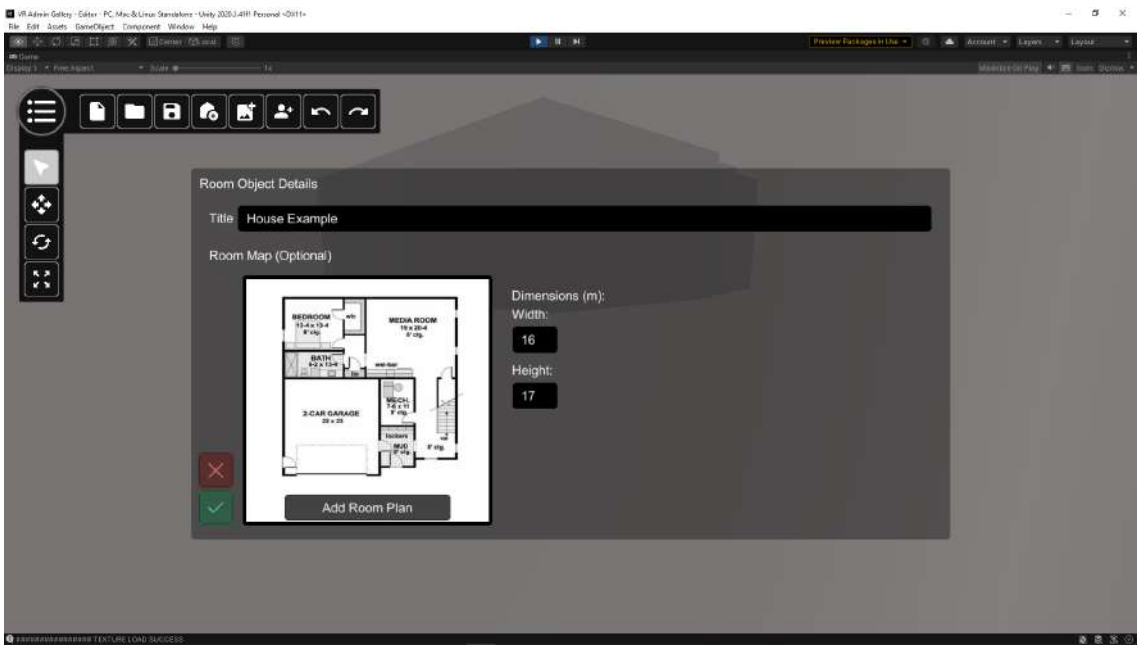

Figure 4.26: The editor dialog that is displayed when Importing a 3D Room Object.

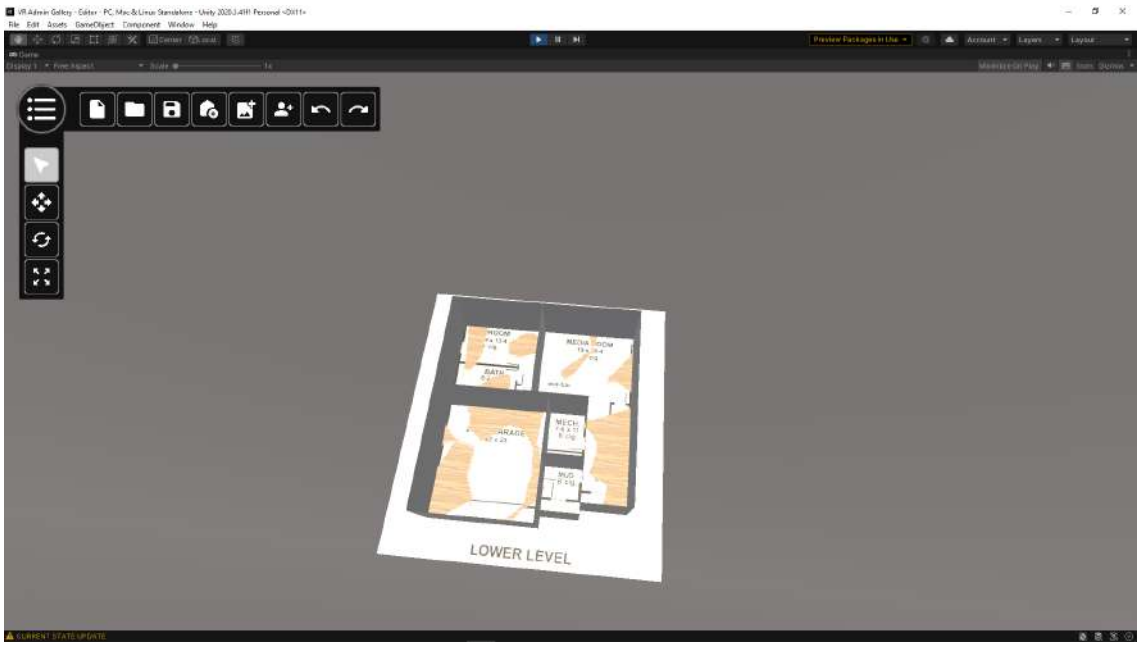

Figure 4.27: An example of an imported 3D Room Object.

Figure 4.28: An example of an imported 3D Room based on data from a 3D Scan.

the scene. As previously mentioned, importing 2D Artefacts and 3D Room Objects follow a similar process.

```
1   // Import a 3D object into the scene
2   public void Import3DObject()
3   {
4       if (!dialogLock)
5       {
6           Transform objectsStage = this.transform.Find("Objects");
7
8           string path = EditorUtility.OpenFilePanel("Import 3D Object (.obj)", "", "obj")
                ↪ ;
9           if (path.Length != 0)
10          {
11              //create an undo step
12              Step();
13
14              //import object and mtl file
15              GameObject loadedObj = new OBJLoader().Load(path);
16              //set the object's parent to the object's stage
17              loadedObj.transform.SetParent(objectsStage);
18              //add the object to the artefactObject list
19              artefactObjects.Add(loadedObj);
20              //create the editorObject script with ID and obj type
21              EditorObject objProperties = loadedObj.AddComponent<EditorObject>();
22              objProperties.SetId(getLatestId() + 1);
23              objProperties.SetObjectType("3dartefact");
24              //create the outline script with the respective properties
25              Outline objOutline = loadedObj.AddComponent<Outline>();
26              objOutline.OutlineColor = Color.white;
27              objOutline.OutlineMode = Outline.Mode.OutlineAll;
```

```
28              objOutline.OutlineWidth = 4f;
29
30              //look for meshfilter, obtain mesh
31              MeshFilter objMeshFilter = loadedObj.GetComponentInChildren<MeshFilter>();
32              Mesh objMesh = objMeshFilter.mesh;
33
34              //use mesh in mesh collider in the top parent so the object can be properly
                    ↪ selected
35              MeshCollider meshCollider = loadedObj.AddComponent<MeshCollider>();
36              meshCollider.sharedMesh = objMesh;
37
38              import3DObjectDialog.gameObject.SetActive(true);
39              import3DObjectDialog.gameObject.GetComponent<ImportDialogScript>().enabled
                    ↪ = true;
40              StartCoroutine(WaitUntilDialogCloses(import3DObjectDialog, loadedObj));
41
42
43          }
44      }
45
46  }
```

5

User Evaluation

This chapter describes the evaluation process, all of its stages, the evaluating user demographics, and the results gathered after thorough testing. Lastly, some preliminary conclusions were taken from this user evaluation process that can be taken into consideration for any future work involving this system.

5.1 Testing Process

Part of the testing process was inspired by the testing process developed for the precursor of this thesis.[4] As its logical successor, some of the evaluation steps were adapted to the developed work. However, due to a higher emphasis oh Augmented and Virtual Reality, the previous testing process included a Presence questionnaire (PQ), which is not as relevant to the work of this thesis as it focuses on small additions to the VR application, which can also be viewed in the Administrative desktop app, and in the Exhibit Editor, which is exclusive to it.

Tests were conducted in person and via voice call with video sharing. Users tested the Exhibit Editor, as well as the changes made in the VR application. In order to obtain a reasonable amount of test data and a larger user base, it was decided to perform the VR application test in the desktop administrative application. Both applications were the target of the same improvements, with the only difference being the platform used. The Desktop application was chosen over the VR Application as it would be available to more test participants, which would allow for a greater sample relevance.

5.1.1 User Questionnaire

This first questionnaire allowed the collection of user background information that helped determined the test user demographics, as seen in section 5.2.

Every user was presented with a small paragraph, written in Portuguese, informing the users about the purposes of the testing they were about to undergo, as well as the types of personal information that would be necessary to assemble a complete demographic study. The users were then asked for their consent in participating in this first

questionnaire.

After consenting to the terms of the questionnaire, the users would be requested to provide the following information:

1. Age
2. Identifying Gender
3. Level of Education
4. Occupation (Optional)
5. Experience with Extended Reality Applications
6. Experience with 3D Editing Applications

5.1.2 Hands-On Test

After submitting the User Questionnaire, the users were given the opportunity to freely interact with the applications, for a limited amount of time. (2 minutes for the Gallery Application, 5 minutes for the Exhibit Editor)

During this period, users were able to explore the entirety of both applications. This process can potentially lead toe the detection of previously undetected bugs or errors within the program. It also serves as way to analyze standard user behavior when using the applications for the first time. It can also help detect design or usability problems, both in the application's UI and in the application itself.

After this hand-on test was completed, users were asked for their personal opinion regarding the applications:

1. Were you able to figure out how most functionalities work?
2. Were the User Interfaces too obstructive?
3. Were the User Interfaces helpful and descriptive?
4. Did the application function as you would expect it to function, from previous experiences or logical deduction?

5.1.3 Feature-specific Evaluation

Upon completing the Hands-On test, users underwent a second phase of testing. This phase consisted of a feature-specific evaluation, wherein users were tasked with performing specific tasks within the applications. This type of evaluation helped gather invaluable information concerning the application's ease of use and intuitiveness.

The users were asked to perform a series of five increasingly complex tasks that should cover all aspects of the application. These tasks were the following:

1. While in VR or in the desktop application in exhibit mode, facing a wall, browse the currently exhibited works and search for "The Starry Night", turn around, and approach the artwork.
2. While in the Exhibit Editor, starting with 3 yellow objects and 3 green objects, select 2 green objects and 1 yellow object.
3. While in the Exhibit Editor, starting with 3 yellow objects and 3 green objects, delete the two yellow objects and replace them with 2 copies of the green objects.
4. While in the Exhibit Editor, starting with 6 equally sized cubes, randomly rotated, construct a 6 step pyramid, with increasingly smaller cubes
5. While in the Exhibit Editor, import an instance of Room A, two instances of the 3D Object A, three instances of the 3D Object B, and two instances and one instance of the 2D Object A. Next, save the exhibit.

After the users completed all five tasks, they were questioned about the following aspects regarding their experience:

1. Did you manage to perform all required tasks?
2. Did you require any help from the developer while performing any of the tasks?
3. Were the tasks straightforward and simple?
4. Did you change your opinion regarding the application as a whole since the previous Hands-On Test questionnaire?
5. If you were able to provide any feedback relative to the application, what would it be?

5.1.4 User Experience Questionnaire

Once all tests are complete, users were presented with a last questionnaire, the User Presence Questionnaire, or UEQ. This questionnaire, originally developed by Laugwitz et al. [35], serves to evaluate usability and user experience aspects of a product or, in this case, a computer application. According to Laugwitz's original paper, these are:

1. Attractiveness
2. Perspicuity
3. Efficiency
4. Dependability
5. Stimulation

Identifier	Age	Gender	Level of Education	VRAR Experience	3D Editing Experience
1	52	Male	High School Graduate	Used at least once	Rarely Uses
2	52	Female	Bachelor's Degree or Above	Used at least once	Never Used
3	23	Male	Bachelor's Degree or Above	Uses Frequently	Uses Frequently
4	72	Female	High School Graduate	Used at least once	Never Used
5	78	Male	High School Graduate	Used at least once	Never Used
6	24	Male	High School Graduate	Used at least once	Rarely Uses
7	24	Male	Bachelor's Degree or Above	Rarely Uses	Uses Regularly
8	23	Male	Bachelor's Degree or Above	Used at least once	Never Used
9	60	Male	High School Graduate	Never Used	Never Used

Table 5.1: Table containing information about the test subjects.

6. Novelty

These aspects were measured using a 7-Point Likert scale, from 1 to 7.

After the entire user evaluation process was concluded, the results were gathered and the data was analyzed. These results can be observed in Section 5.3. From these results, general conclusions about the intuitiveness, usability and possible issues found during the testing process were reached. These conclusions can be found in Section 6.2.

5.2 Test User Demographics

Unlike the tests carried out during the precursor thesis, the tests performed during this evaluation process were comparatively unremarkable in terms of the quantity of test subjects surveyed. Nevertheless, the variety of the smaller group of test subjects is comparable.

While the User Applications are geared towards museum goers, and thus need to accommodate a larger pool of different users, the Exhibit Editor build within the Administrative Application is mostly tailored towards museum personnel, which means it is a smaller, more homogeneous group of users. Regardless, it proved valuable to understand how users with different characteristics and backgrounds interacted with the Editor.

The following table (table 5.1) represents all participants involved in this user evaluation, as well as all personal data provided by them in the first User Questionnaire.

The gender distribution of the test subjects can be observed in figure 5.1.

According to the graph, the gender distribution was relatively homogeneous, with more masculine participants than feminine participants. This is indicative of a disparity when compared to the real world demographic of museum personnel, which is mostly female at around 60%, versus around 40% male.[36][37]

In terms of age distribution, figure 5.2 organizes the participants into 7 groups, each concerning a different age decade. The groups start at the 10-20 age gap and end with

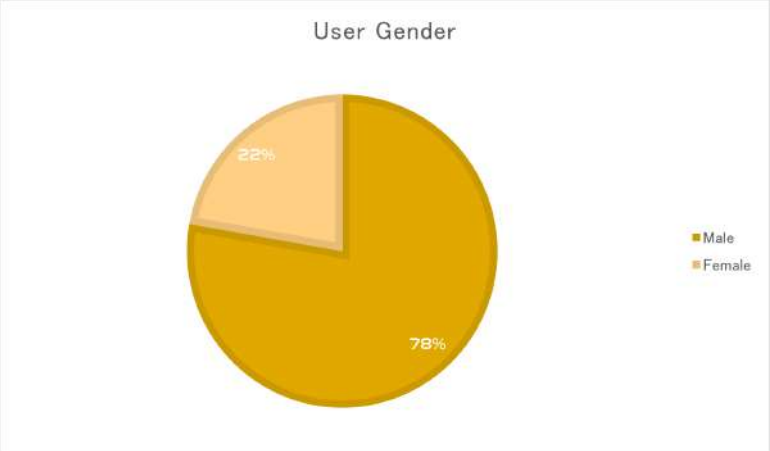

Figure 5.1: Graph displaying the gender distribution among test subjects.

the 70-80 age gap. The average age of all participants was 45,3 years, which is a similar value to real world data. This means that the user sample is representative of real world museum personnel. [36][37]

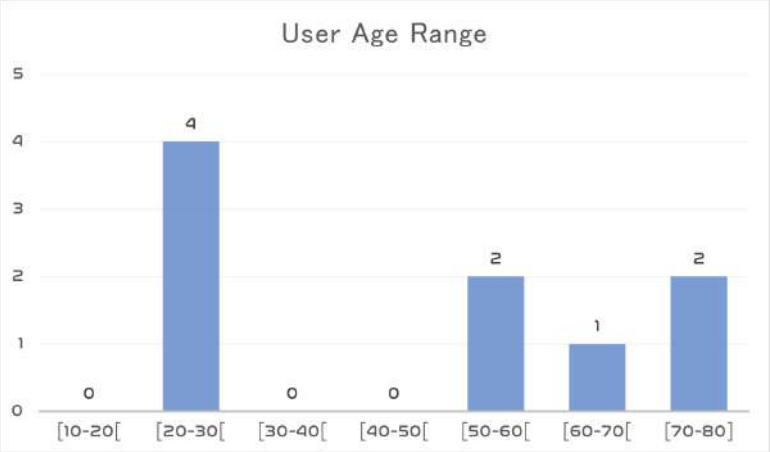

Figure 5.2: Graph displaying the age distribution among test subjects.

Test subject education levels can be observed in figure 5.3.

In regards to this particular characteristic, the subject sample is inevitable more varied than real examples, as education levels among museum curators are relatively consistent, as required by the nature of the job. [36][37]

Lastly, concerning user experience with Extended Reality applications and Editor Applications, the results are as follows:

From what can be seen in figures 5.4 and 5.5, the majority of test subjects have had no prior experience with Editor Applications akin tot he one developed during the course of this thesis. On contrast, perhaps in part due to the COVID-19 pandemic, or a general digitization of society in general, most participants had interacted with virtual reality at least once in their lives.

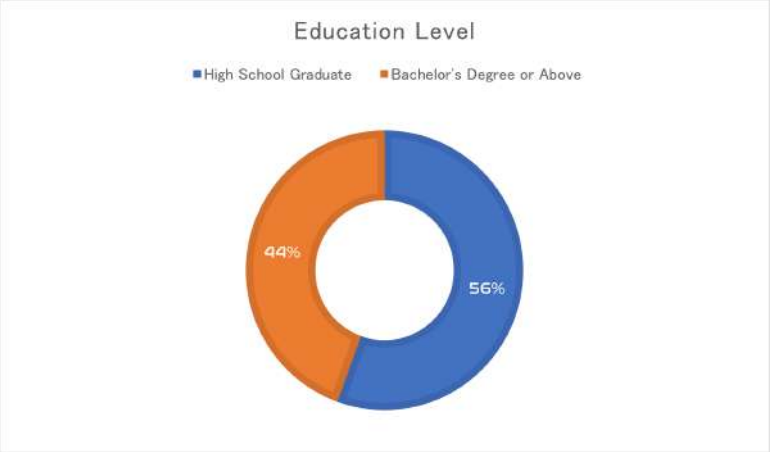

Figure 5.3: Graph displaying the education level distribution among test subjects.

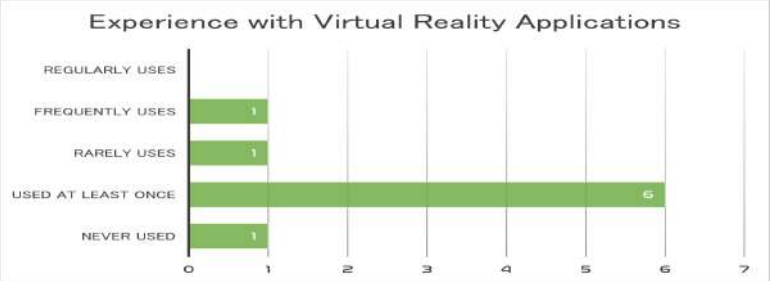

Figure 5.4: Graph displaying the level of familiarity with VR and AR applications among test subjects.

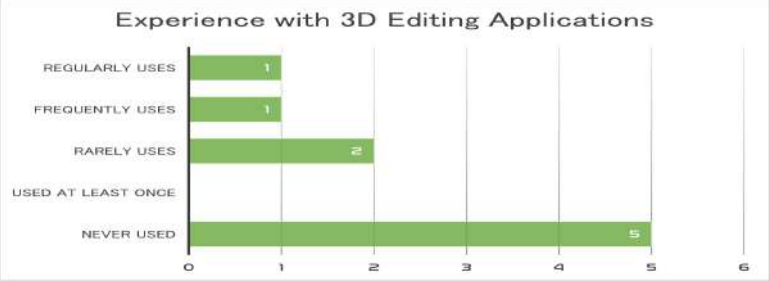

Figure 5.5: Graph displaying the level of familiarity with 3D Modeling applications among test subjects.

5.3 Test Results

This section is intended to present and provide an in depth analysis on the results of the testing regimen described in the previous section. The sample size consisted of 9 users which, despite not being able to provide an accurate assessment of the system, allows for a preliminary evaluation of it.

5.3.1 Hands-On Test

In regards to the hand-on test, where users could freely interact with the applications for a limited amount of time, users unfamiliar with editing applications were able to comprehend some of the basic functionalities of these applications. Users that had some previous experience with this type of applications spent their time trying to understand the limitations and possibilities of the system.

In relation to discovering functionalities, it was found that 100% of test subjects managed to discover most functionalities in both applications. This is an encouraging result, as it indicates both applications possess a certain level of intuitiveness (Figure 5.6).

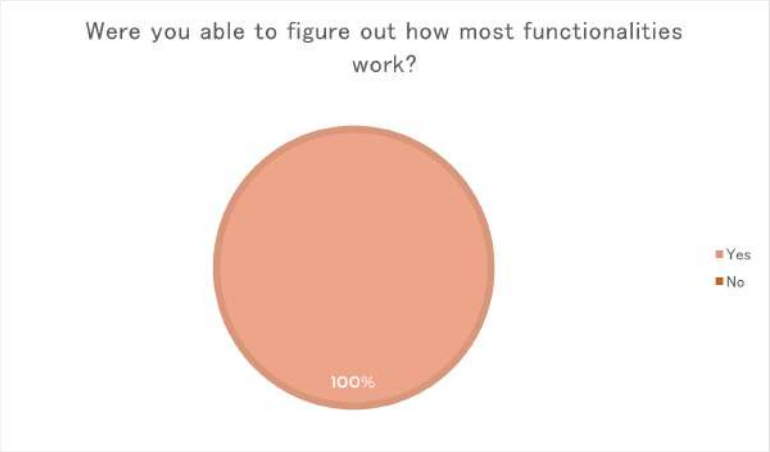

Figure 5.6: Graph showing the first question of the Hands-On Test, related to discovering functionalities.

Regarding User Interface Obtrusion, the results show that 88.9%, or 8 out of 9 users did not consider the User Interfaces to be very obstructive, revealing that the newly developed interfaces do not hinder regular application usage, in both applications (Figure 5.7).

In terms of User Interface Descriptiveness and Helpfulness, it can be observed that 100% of test users found the user interfaces to be helpful and descriptive. This result might prove too optimistic when compared to a real world environment. Nevertheless, it is a good indicator of how the overall interface design might be received by users (Figure 5.8).

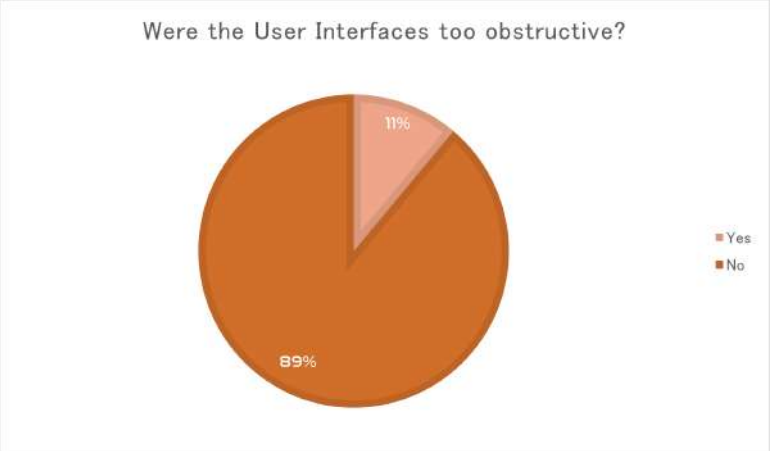

Figure 5.7: Graph showing the second question of the Hands-On Test, related to interface obtrusion.

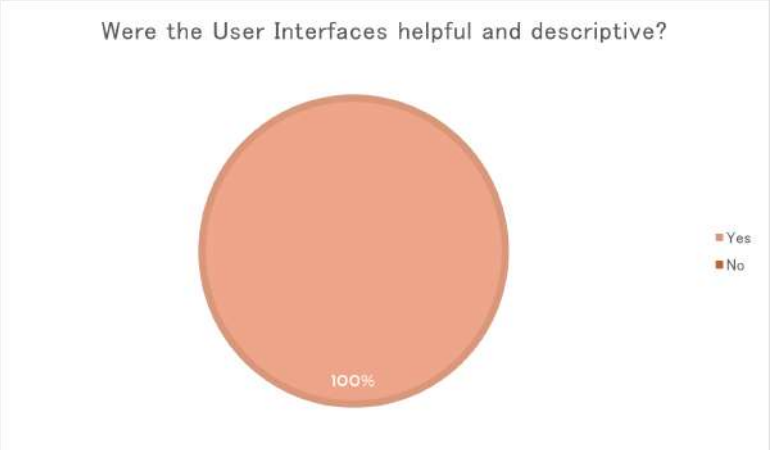

Figure 5.8: Graph showing the third question of the Hands-On Test, related to interface descriptiveness and helpfulness.

Lastly, with respect to application intuitiveness and adherence to convention, is it possible to visualize that two thirds of the test base (66,7%) found the applications to behave in a logical or intuitive way, according to previous experience or logical deduction. However, a third (33,3%) found the application to behave contrary to it. Thus, this result reveals that there is room for improvement relative to how the applications behave. Studying other similar applications or taking examples from the real world might improve this value (Figure 5.9).

5.3.2 Feature-specific Evaluation

Concerning the Feature-specific Evaluation, where users were given increasingly complex tasks to complete within the developed system, users had some previous experience with the applications during the Hands-On Test. The decision to place the feature-specific test

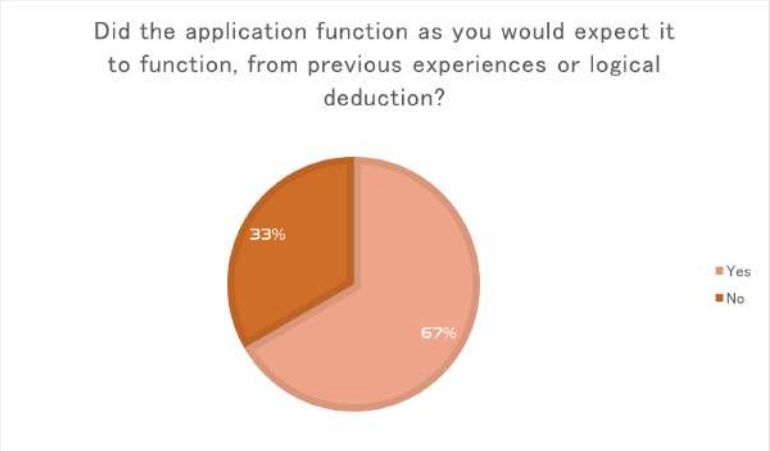

Figure 5.9: Graph showing the fourth and last question of the Hands-On Test, related to application intuitiveness and adherence to convention.

after providing users with some first-hand knowledge of the application, albeit minimal, was deliberate. This is to ensure unfamiliar users have a higher chance of completing most tasks with minimal or no assistance. It also guarantees that users familiar with the technologies are already acclimated to the developed system, thus decreasing task completion time, and possibly an improved user experience.

Regarding whether users were able to complete all required tasks, the results demonstrate that all users (100%) completed all five required tasks, with or without developer help. These results indicate that there are no severe usability problems within both applications, which could not be solved even with developer intervention or guidance (Figure 5.10).

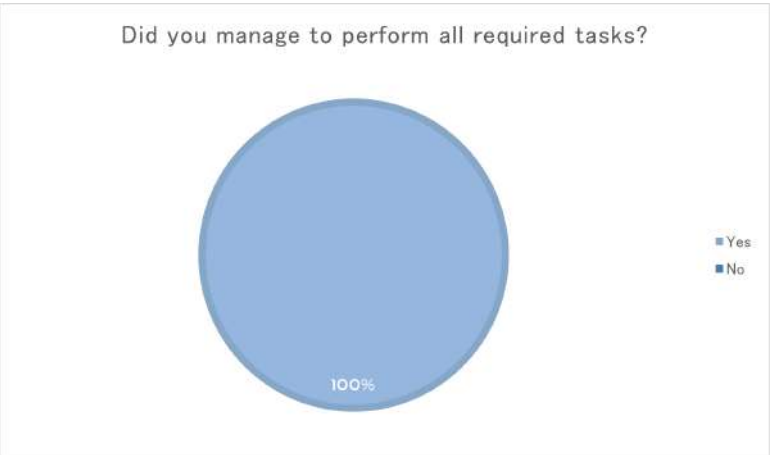

Figure 5.10: Graph showing the first question of the Feature Specific Evaluation, related to task completion.

With regards to users requiring external help while performing the required tasks, it is possible to observe that 44,4% participants (4 out of 9) required developer help to

complete one or more tasks. Due to the divisive nature of this results, users who answered "Yes"regarding needing developer help were asked for further information regarding which tasks proved more complex or less straightforward, or less intuitive (Figure 5.11).

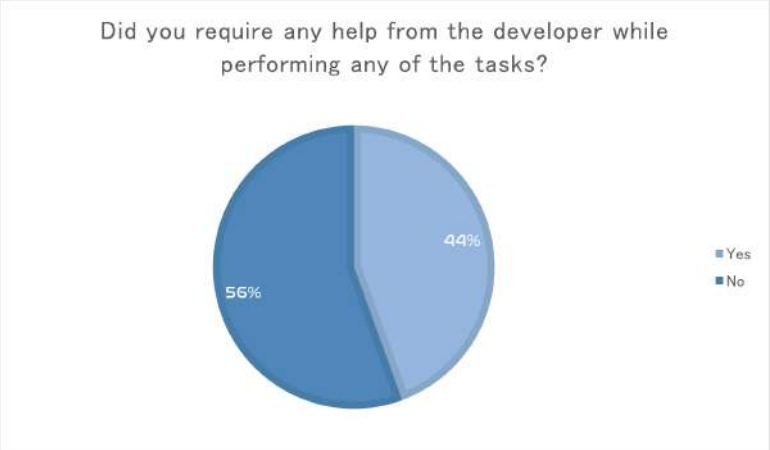

Figure 5.11: Graph showing the second question of the Feature Specific Evaluation, related to external developer help.

Of these 4 users, three of them expressed difficulty performing task 1, one of them required help to complete task 2, and two of them experienced hardships while trying to perform tasks 3, 4, and 5.

These values denote that the first task was not logically obvious upon first interaction to one third of all test participants, whereas tasks 3, 4 and 5 required additional information to two ninths (22.2%) of users. Considering that tasks were developed and ordered by increasing complexity, it would not prove unusual for the last three tasks to cause difficulties to users. However, 33.3% of users requiring help with the first task indicates that the task instructions were not clear enough for the test subjects, or that the available interfaces required to perform the requested task were not clear (Figure 5.12).

In relation to task complexity, the test results reveal that, even when requiring external help, 77.8% of test participants (seven participants) considered the required tasks to be simple and straightforward. To gather additional data about their responses, the two (22.2% of) users who considered the required tasks to be complex were questioned about which tasks they were referring to (Figure 5.13).

One user considered tasks 2 and 3 to be complex, whereas both users regarded tasks 4 and 5 as the most complex. This is in accordance with the original goal of the feature-specific evaluation, which as previously mentioned consisted of increasingly complex tasks. None of the users did, however, classify the first task as complex. This might indicate that, apart from an introductory explanation and help from the developer, users managed to understand what was being requested from them. It might also suggest that the issue with users requiring help in task 1 is related to unclear interfaces.

This would seem to contrast against the hands-on test results, where users did not

Figure 5.12: Graph showing detailed answers to the second question of the Feature Specific Evaluation, related to external developer help.

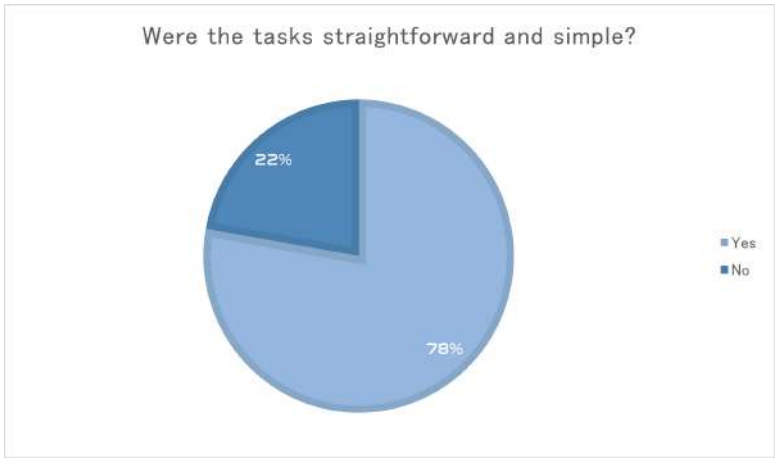

Figure 5.13: Graph showing the third question of the Feature Specific Evaluation, related to task complexity.

report any confusing interfaces. However it is worth noting that no user in said test opened the additional artwork browse interface that was required to complete task 1 (Figure 5.14).

With concern to user opinion changes since the hands-on test, these results prove that users did not change their opinion of the system during the feature-specific evautation, for the most part. 77.8% of users said that their overall opinion of the system remained unchanged since the first test (Figure 5.15).

Finally, in terms of user feedback, the main feedback provided by the users is the following:

1. The mouse sensitivity in the gallery scene was reported to be too high by most participants.

Figure 5.14: Graph showing detailed answers to the third question of the Feature Specific Evaluation, task complexity.

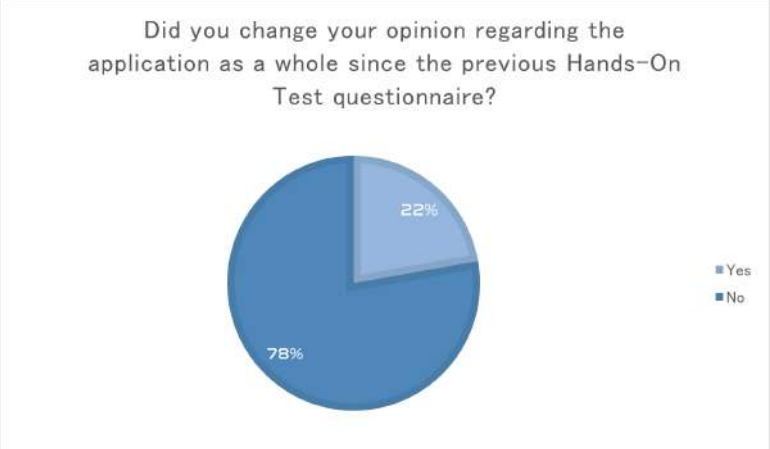

Figure 5.15: Graph showing the fourth and last question of the Feature Specific Evaluation, related to opinion changes.

2. The button that enables the ArtPieceCanvas in the gallery scene needs an improvement in terms of visibility, as users could not tell whether the button was being hovered or not.

3. Large paintings currently overlap over the ArtPieceCanvas text. This is due to the painting's frame which protrudes from the painting by a substantial amount. Re-scaling the frame is a solution to this issue.

4. The Exhibit Editor was shown to be more intuitive and user friendly when compared to the gallery scene.

5. The camera movement within the Exhibit Editor was not entirely obvious or intuitive for the test subjects. A tooltip or a visual aid describing how to perform this movement or if this camera mode is currently active could be implemented.

6. Object Rotation in the Exhibit Editor still suffers from problems related to the camera perspective, as sometimes rotation occurs in a backwards fashion, slower than it should, or erratically. Alternative ways to perform the desired rotation need to be explored.

7. Some users suggested the creation of a right-click context menu for the Exhibit Editor. This would bring the Editor closer to standard behaviour, as file explorers and other editors make user of these context menus as an alternative to the lesser known keyboard shortcuts.

8. One of the users was using a Linux system. Upon trying to import objects into the scene, no files or folders would be shown, possible due to different operating system file systems.

9. Gizmos need to be placed in a layer above the objects, so as to always be visible to the user.

10. It was suggested to make object manipulation operations local instead of global. This would always align gizmos with the object's angle.

11. Another suggestion would be to make the coordinate indicators editable, thus allowing users to manually input coordinates into objects.

5.3.3 User Experience Questionnaire

The User Experience Questionnaire provided valuable user insight concerning overall user experience while interacting with the developed system. It is insightful to analyze what users thought about the system in relation to each of the usability and user experience aspects.

Concerning Attractiveness, the Questionnaire's results indicate that users generally considered the developed system to be an attractive one, as the mode of all attractiveness values was 6 and the median was 6. All evaluations were in the 4 to 7 range, meaning all feedback was neutral to positive (Figure 5.16).

With regards to Perspicuity, it was found that the developed system was also positively received in this regard. The mode of all perspicuity scores was 5, with a median value of 5. As with attractiveness, all feedback was either neutral, at 4, or positive, from 5 to 7 (Figure 5.17).

In relation to efficiency, it is possible to observe, from the results, that the system can provide some efficiency to otherwise difficult workflows. Users have rated the system from 4 to 7 in this regard, with a mode value of 5 and a median of 5 (Figure 5.18).

With respect to Dependability, users considered that the system is mostly dependable, albeit with some issues, as stated in the previous subsection. Results in this aspect were lower than the previous values. The mode value was 5 and the median value was 5. Most results were in the 4 to 6 ranges, with the only outlier being a single 2 result. This result

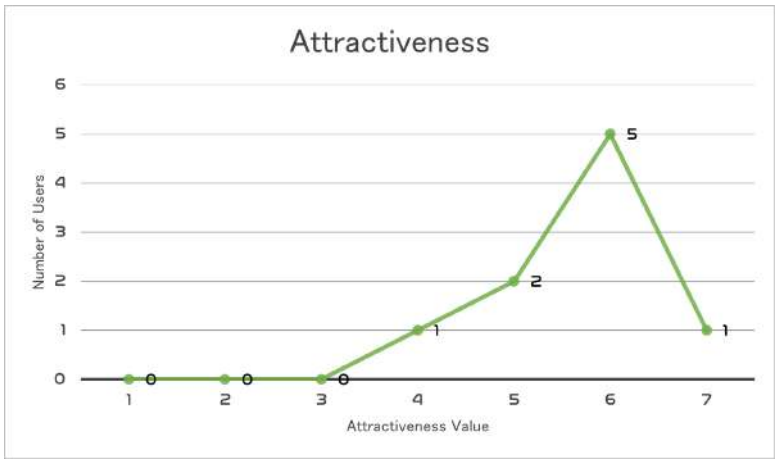

Figure 5.16: Graph showing the Attractiveness Aspect of the User Experience Questionnaire.

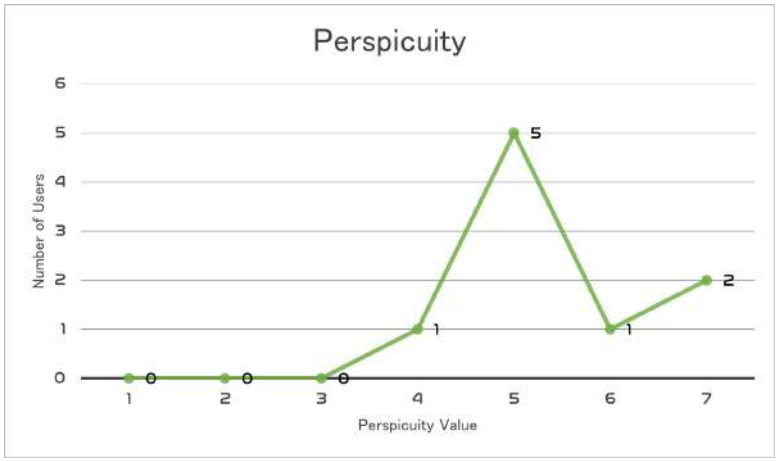

Figure 5.17: Graph showing the Perspicuity Aspect of the User Experience Questionnaire.

Figure 5.18: Graph showing the Efficiency Aspect of the User Experience Questionnaire.

can be explain by some unpredictability and instability in the editor, such as during rotation movements (Figure 5.19).

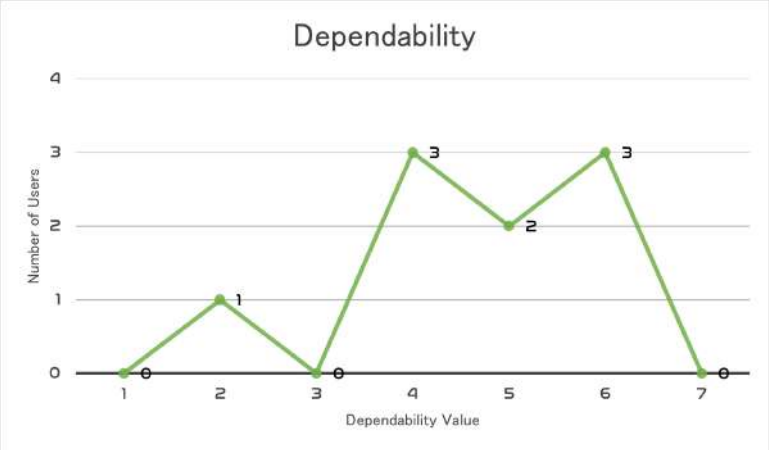

Figure 5.19: Graph showing the Dependability Aspect of the User Experience Questionnaire.

In terms of Stimulation, the results revealed that users feel stimulated while using the developed system. This means the system has the ability to motivate users during regular usage. This can lead to increased efficiency, which might explain the efficiency values described above. The mode of all stimulation values was 4.5, whereas the median value was 5. Values in this category are mostly consistent, with all users attributing a value between 4 and 7 (Figure 5.20).

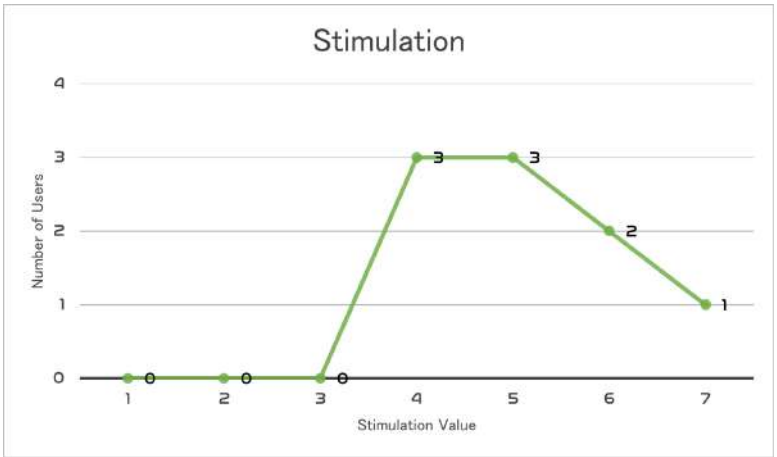

Figure 5.20: Graph showing the Stimulation Aspect of the User Experience Questionnaire.

Lastly, regarding Novelty, it is shown that the developed system is considered novel by most participants. The mode value was 7, with a median of 6. The reason the mode is higher than the median in this attribute is due to the range of values given by users. Most users gave a novelty value between 4 and 7. However, one user gave the system a value of 1, which decreased the median value. This value was given because the user in

question had frequent experience with 3D Editing applications, and thus was familiar with similar applications (Figure 5.21).

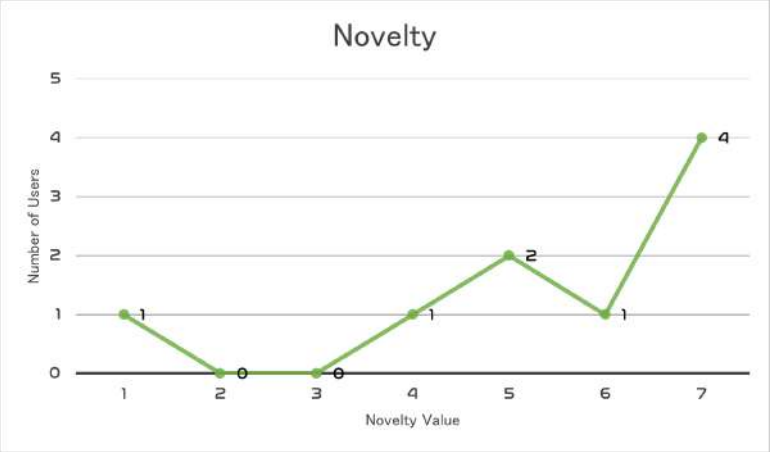

Figure 5.21: Graph showing the Novelty Aspect of the User Experience Questionnaire.

5.3.4 Discussion

From a global standpoint, the results of the application evaluation can be considered to be generally favourable. This represents a promising outlook regarding the future of the application, as it demonstrates that the system does not suffer from any apparent usability flaws and is intuitive and easy to use, even for those who are not familiar with editing applications. However, it is worth referencing the inadequacy of the user sample when compared to a real world environment. It is of note that participants that had never interacted with editing software before encountered some difficulties while performing some of the required tasks. The Hands-On Test results demonstrated that users could quickly navigate within both applications and found the user interfaces to be descriptive. Additionally, two thirds of the test subjects agreed that the application performed in a logically deducible way. On the other hand, while the Feature-specific Evaluation revealed the system does not suffer from severe usability problems, and test users managed to complete all required tasks, it was also concluded that some additional helping features, bugfixing, or quality of life improvements are necessary to improve system usability and ease of use. Lastly, from the results of the User Experience Questionnaire, it can be concluded that the system was positively received by all users in most categories, with attractiveness having the highest average value and dependability demonstrating the lowest average value. These values indicate the user experience while using the system was, in a general sense, optimistic.

6

Conclusions and Future Work

This chapter will consolidate all findings and conclusions related to the work developed during the period of this thesis. It also serves as a starting point for future work involving the developed system as it provides several possibilities concerning which areas of the system need further improvements, as well as possible additions that might be implemented in the future.

6.1 Final Overview

The main objective of this thesis was the improvement of an existing Extended Reality application via the development of additional features, and extending existing ones.

The original application was composed of three Unity applications and a supporting file and media server. The Unity applications consisted of an Augmented Reality application, designed for smartphones, a Virtual Reality Application, designed for specific Virtual Reality headsets, and a desktop Administrative application, aimed at exhibit monitoring and content moderation.

Taking this existing system as a base, the objective of this thesis was to improve on it by adding functionalities that would improve user experience and provide additional capabilities to the system. To this end, some changes to the system architecture were made to allow the development of said capabilities.

In terms of new functionalities, three new functionalities were added to the client applications. These are dynamic artefact instantialization, a navigational aid, and an artefact browse menu.

Furthermore, it was decided to implement a feature-rich exhibit editor, a tool to allow museum or gallery personnel to create and modify their own exhibits using 2D and 3D artefacts that can be traditionally modeled or 3D scanned from real-world objects.

Concerning dynamic artefact instantialization, the .XML data files present in local storage were modified to contain additional information about the works they represent, as well as to support other types of objects, such as 3D artefacts or the exhibit rooms themselves. The VR and administrative app themselves were then changed to, instead

of relying on preset artworks, load these artworks from an .XML file, and place them dynamically across the room, according to the coordinates and properties stored in the .XML file.

In regards to navigation aid, a decision was made to implement this navigational aid in the form of a minimap. This approach was chosen since it is a common element in modern navigation applications and in video games. This minimap displays the user's position, as well as the positions of any nearby artefacts, with a different color for each type of artefact. The user can also view an additional floorplan or blueprint of the exhibit room if the room has been pre-configured to support one such element. This allows the user to obtain more information from the exhibit or locate themselves in the room.

With respect to the artefact menu, this is a new UI element that appears when the user presses an action button on the Oculus controllers, or the Home button on a keyboard. This menu contains a list of all currently exhibited artefacts. An additional search box allows users to search for a specific artwork present within the exhibit. Upon selecting one of the artworks available within the list, the user must now find the chosen artwork. To facilitate this process, its minimap indicator now displays a different color. Additionally, the artefact will emit a three dimensional sound effect to guide users toward it. When nearby, users will notice that the artefact is accompanied by an animated floating indicator.

Lastly, regarding the exhibit editor, this was a time-consuming feature made entirely from scratch. Within the administrative application users will find an additional button that will load a completely new scene. This is the exhibit editor. It consists of an object manipulation program that allows museum or gallery personnel to create and edit custom exhibits. It allows complete control over an objects movement, rotation, and scale. Staff can import 2D or 3D artefact objects, as well as 3D room objects. These can be customized upon import with details about them. Rooms can be further customized by supplying a complimentary floorplan or blueprint, that will show up in the user's minimap in the VR and administrative applications as previously mentioned. Staff and curators can then assemble an entire exhibit using these objects and tools, and export it so it can be loaded in the client applications. It is also possible to load existing exhibits and modify them. It is worth noting that this editor only supports .obj model files and .png and .jpg image files.

From the results of the testing phase, it is possible to conclude that the system performed positively among the user pool in terms of usability, intuitiveness, and ease of use. This is a significant result, as it demonstrates that the developed system continues to harbor the potential to be a successful product. Any bugs encountered during this evaluation process were taken into consideration and added to the bug fix list. User feedback was also an important factor in deciding what should be corrected in the system.

6.2 Issues, Limitations and Future Features

Due to time constraints and technology limitation, some features could not be implemented during the development period, and will thus be considered future developments. Additionally, future improvements to existing systems might also be relevant. Some of these features are:

1. A possible collaborative experience, where VR user can see each other in virtual space. This feature might help users interact among themselves in a virtual environment, as well as enabling Virtual visitors to share experiences with physical visitors. However, constant content monitoring and moderation would be required.

2. Implementing supports for other types of artwork, like video and audio items, can greatly improve user experience in the exhibit, as these types of media generally require specific physical devices to play them, which can lead to a significant crowding of visitors in the same location within the exhibit.

3. Adding dynamic attributes to artefacts, so users can obtain a greater understanding of their story and purpose. For instance, having several preservation stages of an ancient artefact throughout history, since its original conception, to environmental decay, to museum restorations, or the moving inner workings of an old machine, like a mechanical calculator or the enigma machine.

4. The current room floorplan solution works reasonably well. However, it would be a more reliable solution to have a dynamic rendering of any room in the exhibit. Experiments were carried out using several outline filters and methods. Additionally, two dimensional walls (without any depth) can prove quite problematic when seen from above. A derivative of a cell or edge shader is possible, or perhaps a different solution entirely.

5. The addition of preset objects to the exhibit editor might increase the user pool as several institutions lack the financial resources, equipment, or technical know-how to model or scan large parts of their collection. Having preset rooms or objects, like pedestals or lights, that the user can simply add to the scene from a dedicated menu is one possible approach to this issue.

6. Cross-sharing of artwork drawings between the AR and VR/Administrative applications is still not possible, due to the way vertices are calculated in both AR and VR applications. Positional, rotational and scaling coordinates are adjusted according to the application environment.

7. Annotating three dimensional artefacts, as well as interacting with them in VR, by moving and rotating them, was not implemented. The interaction itself is not a complex task. However, the act of drawing in 3D surfaces might prove challenging.

A possible approach would be to treat the texture of the object as the canvas in a 2D artefact. Then, using Unity's RaycastHit.textureCoord, it would be possible to fetch the raycast hit coordinate. Using Texture2D.SetPixels would change the color of the hit pixels to the user's choice. The texture could then be exported to the .XML file in a similar fashion to 2D artefacts. Upon loading 3D object drawings, these textures would be overlaid on the 3D model.

8. From taking user feedback into account, the features the most users requested were the inclusion of a right-click context menu, the ability to edit the coordinate widget to manually input coordinates, and modifying the coordinate system to use local coordinates for object operations instead of global coordinates.

9. Additionally, it is imperative to develop an improved way to perform object rotation, as the current system is not functioning as intended.

10. However, 2D artefact rotation can be bypassed altogether if an alternative placement method is developed. This method would involve the 2D artefacts "sticking"to their nearest wall. This would ensure curators would always place the artefacts on the wall, preventing clipping or floating artefacts. It would also make said placement easier and straightforward, which is beneficial for curators without significant experience with 3D Editing software.

    In addition to these unimplemented features, another challenge faced during the development of this system was interacting with an existing framework, with unfamiliar code which needs to be expanded and enhanced with the new features. While not a significant impediment, it nonetheless proved more difficult to overcome when compared to working with newly developed code.

    Lastly, to ensure a greater understanding of end-users' needs, a new, larger evaluation process must be conducted, with museum curators and other domain experts. This new evaluation will have as a main objective to gather information about system requirements, as well as gather more accurate information about the system's performance and user acceptance relative to its ease of use and general usability.

Bibliography

[1] M. R. H. da Silva, A. M. Reis, A. R. d. A. F. P. Mégre, A. M. F. S. V. Pires, A. Ribeiro, A. H. R. Miranda, A. Rodrigues, D. M. M. Rijo, D. Folgado, E. Alberto, E. Gama, H. Silva, J. Alegria, J. M. C. R. Garcia, M. d. F. D. I. Ó. Ramos, P. Alho, and P. O. Ramos. *PROJECT LXCONVENTOS - From sacred city to secular town. The extinction of religious orders and the dynamics of urban transformation in the nineteenthe century Lisbon*. Feb. 2023. URL: https://institutodehistoriadaarte.wordpress.com/research/fctfunded/cidadesacra/ (cit. on p. 1).

[2] T. Jung, M. C. tom Dieck, H. Lee, and N. Chung. "Effects of Virtual Reality and Augmented Reality on Visitor Experiences in Museum". In: *Information and Communication Technologies in Tourism 2016*. Springer International Publishing, 2016, pp. 621–635. DOI: 10.1007/978-3-319-28231-2_45. URL: http://link.springer.com/10.1007/978-3-319-28231-2_45 (cit. on pp. 2, 19–21).

[3] G. A. Lee, A. Dunser, S. Kim, and M. Billinghurst. "CityViewAR: A mobile outdoor AR application for city visualization". In: *2012 IEEE International Symposium on Mixed and Augmented Reality - Arts, Media, and Humanities (ISMAR-AMH)*. IEEE, Nov. 2012, pp. 57–64. ISBN: 978-1-4673-4665-8. DOI: 10.1109/ISMAR-AMH.2012.6483989. URL: http://ieeexplore.ieee.org/document/6483989/ (cit. on p. 2).

[4] P. Lourenço. "Augmented and Virtual Reality for Enhanced Presence in Cultural Institutions". MSc Dissertation in Computer Science. Universidade NOVA de Lisboa, 2022 (cit. on pp. 2, 12, 14, 15, 22, 25, 29, 61).

[5] F. Churchville. *What is User Interface (UI)?* Sept. 2021. URL: https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI (cit. on p. 7).

[6] Cybrary. *Command line interface (CLI) vs. Graphical User Interface (GUI)*. Dec. 2020. URL: https://www.cybrary.it/blog/0p3n/command-line-interface-cli-vs-graphical-user-interface-gui/ (cit. on p. 7).

[7] H. Ishii. "Tangible Bits: Beyond Pixels". In: *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*. TEI '08. Bonn, Germany: Association for Computing Machinery, 2008, pp. xv–xxv. ISBN: 9781605580043. DOI: 10.1145/1347390.1347392. URL: https://doi.org/10.1145/1347390.1347392 (cit. on p. 8).

[8] M. Kaltenbrunner, S. Jorda, G. Geiger, and M. Alonso. "The reacTable*: A Collaborative Musical Instrument". In: *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'06)*. 2006, pp. 406–411. DOI: 10.1109/WETICE.2006.68 (cit. on p. 8).

[9] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. "Recent advances in augmented reality". In: *IEEE Computer Graphics and Applications* 21.6 (2001), pp. 34–47. DOI: 10.1109/38.963459 (cit. on p. 8).

[10] K. Dupzyk. *I saw the future through Microsoft's hololens*. Sept. 2016. URL: https://www.popularmechanics.com/technology/a22384/hololens-ar-breakthrough-awards/ (cit. on p. 8).

[11] D. Wagner and D. Schmalstieg. "Handheld Augmented Reality Displays". In: *IEEE Virtual Reality Conference (VR 2006), Alexandria, VA, USA*. 2006, pp. 321–321. DOI: 10.1109/VR.2006.67 (cit. on p. 8).

[12] K. Vi. *Why augmented reality does not receive full public acceptance?* Oct. 2021. URL: https://medium.com/vibentec-it/why-augmented-reality-does-not-receive-full-public-acceptance-324308b8f90b (cit. on p. 9).

[13] B. Lawson. "Motion Sickness Symptomatology and Origins". In: *Handbook of Virtual Environments*. CRC Press, Sept. 2014, pp. 532–587. DOI: 10.1201/b17360-29 (cit. on p. 10).

[14] A. D. Souchet, S. Philippe, D. Lourdeaux, and L. Leroy. "Measuring Visual Fatigue and Cognitive Load via Eye Tracking while Learning with Virtual Reality Head-Mounted Displays: A Review". In: *International Journal of Human–Computer Interaction* 38.9 (2022), pp. 801–824. DOI: 10.1080/10447318.2021.1976509 (cit. on p. 10).

[15] D. Stredney, D. Sessanna, J. S. McDonald, L. Hiemenz, and L. B. Rosenberg. "A virtual simulation environment for learning epidural anesthesia". In: *Medicine Meets Virtual Reality*. IOS Press. 1996, pp. 164–175 (cit. on p. 10).

[16] S. Parkin. *How VR is training the perfect soldier*. Dec. 2015. URL: https://www.wareable.com/vr/how-vr-is-training-the-perfect-soldier-1757 (cit. on p. 10).

[17] M. Casini. "Extended Reality for Smart Building Operation and Maintenance: A Review". In: *Energies* 15.10 (2022). ISSN: 1996-1073. DOI: 10.3390/en15103785. URL: https://www.mdpi.com/1996-1073/15/10/3785 (cit. on p. 11).

[18] T. Shibata. "Head mounted display". In: *Displays* 23.1 (2002), pp. 57–64. ISSN: 0141-9382. DOI: https://doi.org/10.1016/S0141-9382(02)00010-0. URL: https://www.sciencedirect.com/science/article/pii/S0141938202000100 (cit. on p. 11).

[19] Google. *Degrees of freedom ||GoogleVR||googledevelopers*. Sept. 2018. URL: https://developers.google.com/vr/discover/degrees-of-freedom (cit. on p. 11).

[20] Mechatech. *How do common virtual reality tracking systems work?* Feb. 2023. URL: https://www.mechatech.co.uk/journal/how-do-common-virtual-reality-tracking-systems-work (cit. on p. 11).

[21] D. W. Hansen and Q. Ji. "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 32.3 (Mar. 2010), pp. 478–500. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2009.30. URL: https://doi.org/10.1109/TPAMI.2009.30 (cit. on p. 11).

[22] Ultraleap. *What is hand tracking in VR?* Feb. 2023. URL: https://www.ultraleap.com/company/news/blog/hand-tracking-in-vr/ (cit. on p. 11).

[23] T. Carmody. *How motion detection works in Xbox Kinect*. Nov. 2010. URL: https://www.wired.com/2010/11/tonights-release-xbox-kinect-how-does-it-work/ (cit. on p. 11).

[24] P. Fuguo and J. Zhai. "A mobile augmented reality system for exhibition hall based on Vuforia". In: *2017 2nd International Conference on Image, Vision and Computing, ICIVC 2017*. IEEE, July 2017, pp. 1049–1052. ISBN: 9781509062379. DOI: 10.1109/ICIVC.2017.7984714 (cit. on pp. 14, 15, 23).

[25] M. Ramirez, E. Ramos, O. Cruz, J. Hernandez, E. Perez-Cordoba, and M. Garcia. "Design of interactive museographic exhibits using Augmented reality". In: *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*. IEEE, Mar. 2013, pp. 1–6. ISBN: 978-1-4673-6155-2. DOI: 10.1109/CONIELECOMP.2013.6525747. URL: http://ieeexplore.ieee.org/document/6525747/ (cit. on pp. 16, 22, 23).

[26] F. A. Purnomo, P. I. Santosa, R. Hartanto, E. H. Pratisto, and A. Purbayu. "Implementation of Augmented Reality Technology in Sangiran Museum with Vuforia". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 333. Institute of Physics Publishing, Apr. 2018. DOI: 10.1088/1757-899X/333/1/012103 (cit. on pp. 16, 17, 22, 23).

[27] I. Reimat, Y. Mei, E. Alexiou, J. Jansen, J. Li, S. Subramanyam, I. Viola, J. Oomen, and P. Cesar. "Mediascape XR: A Cultural Heritage Experience in Social VR". In: *Proceedings of the 30th ACM International Conference on Multimedia*. MM '22. Lisboa, Portugal: Association for Computing Machinery, 2022, pp. 6955–6957. ISBN: 9781450392037. DOI: 10.1145/3503161.3547732. URL: https://doi.org/10.1145/3503161.3547732 (cit. on pp. 17, 18, 22).

[28] L. D. Clark, A. B. Bhagat, and S. L. Riggs. "Extending Fitts' law in three-dimensional virtual environments with current low-cost virtual reality technology". In: *International Journal of Human-Computer Studies* 139 (2020), p. 102413. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2020.102413. URL: https://www.sciencedirect.com/science/article/pii/S1071581920300173 (cit. on p. 18).

[29] D. Saffo, S. Di Bartolomeo, C. Yildirim, and C. Dunne. "Remote and Collaborative Virtual Reality Experiments via Social VR Platforms". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445426. URL: https://doi.org/10.1145/3411764.3445426 (cit. on pp. 18, 19).

[30] N. Chung, H. Han, and Y. Joun. "Tourists' intention to visit a destination: The role of augmented reality (AR) application for a heritage site". In: *Computers in Human Behavior* 50 (2015), pp. 588–599. ISSN: 0747-5632. DOI: https://doi.org/10.1016/j.chb.2015.02.068. URL: https://www.sciencedirect.com/science/article/pii/S0747563215002101 (cit. on pp. 19, 20, 23).

[31] J. Short, E. Williams, and B. Christie. *The Social Psychology of Telecommunications*. Wiley, 1976. ISBN: 9780471015819. URL: https://books.google.pt/books?id=Ze63AAAAIAAJ (cit. on p. 19).

[32] B. J. Pine, J. H. Gilmore, et al. *Welcome to the experience economy*. Vol. 76. 4. Harvard Business Review Press, 1998, pp. 97–105 (cit. on pp. 19, 20).

[33] I. Giangreco, L. Sauter, M. A. Parian, R. Gasser, S. Heller, L. Rossetto, and H. Schuldt. "VIRTUE: A Virtual Reality Museum Experience". In: *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*. IUI '19. Marina del Ray, California: Association for Computing Machinery, 2019, pp. 119–120. ISBN: 9781450366731. DOI: 10.1145/3308557.3308706. URL: https://doi.org/10.1145/3308557.3308706 (cit. on pp. 21, 22).

[34] K. Zagata and B. Medynska-Gulij. "Mini-Map Design Features as a Navigation Aid in the Virtual Geographical Space Based on Video Games". In: *ISPRS International Journal of Geo-Information* 12.2 (2023). ISSN: 2220-9964. DOI: 10.3390/ijgi12020058. URL: https://www.mdpi.com/2220-9964/12/2/58 (cit. on p. 24).

[35] B. Laugwitz, T. Held, and M. Schrepp. "Construction and Evaluation of a User Experience Questionnaire". In: *HCI and Usability for Education and Work*. Ed. by A. Holzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 63–76. ISBN: 978-3-540-89350-9 (cit. on p. 63).

[36] *Zippia - Museum Curator Demographics and Statistics in the US*. Sept. 2023. URL: https://zippia.com/museum-curator-jobs/demographics/ (cit. on pp. 64, 65).

[37] L. Sweeney, D. Harkins, and J. Dressel. "Art Museum Staff Demographic Survey 2022". In: *American Alliance of Museums - Art Museum Staff Demographic Survey 2022* (Nov. 2022). DOI: 10.18665/sr.317927. URL: https://www.aam-us.org/2022/11/22/art-museum-staff-demographic-survey-2022/ (cit. on pp. 64, 65).

A

Portuguese Version of the User Evaluation Questionnaire

This Appendix depicts an example of the Google Forms questionnaire users encountered during the User Evaluation phase described in Chapter 5.

Questionário de Avaliação de Sistema Informático em desenvolvimento.

O seguinte questionário está enquadrado no contexto de uma tese de mestrado em Engenharia Informática na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. O objetivo deste questionário é a obtenção de dados de utilização das aplicações desenvolvidas durante o período da tese suprarreferida, para apurar métricas relacionadas com a eficácia e usabilidade das aplicações.

Os dados obtidos neste questionário serão exclusivamente utilizados para efeitos estatísticos e de investigação, e não serão partilhados com qualquer outra entidade. Nenhum elemento identificativo será pedido ao inquirido, como nome, morada, contacto ou documento de identificação. *

O inquirido aceita participar neste estudo e permite a utilização dos dados que fornece de forma voluntária, confiando em que apenas serão utilizados para esta investigação e nas garantias de confidencialidade e anonimato que lhe foram dadas pelo investigador.

- (●) Sim
- ( ) Não

Dados Pessoais

Para obter uma perceção global da demografia da amostra de teste, bem como comparar com o público alvo das aplicações, os seguintes dados pessoais do utilizador são necessários:

Idade *

0

Género *

- ○ Masculino
- ○ Feminino
- ◉ Outra: None

Nível de Educação *

- ◉ 1° Ciclo do Ensino Básico
- ○ 2° Ciclo do Ensino Básico
- ○ 3° Ciclo do Ensino Básico
- ○ Ensino Secundário
- ○ Ensino Superior (Licenciatura e Posterior)

Ocupação (Opcional)

Sample Form

Experiência com Realidade Virtual e/ou Aumentada *

- ◉ Nunca utilizou
- ○ Já utilizou pelo menos uma vez
- ○ Utiliza raramente
- ○ Utiliza frequentemente
- ○ Utiliza regularmente

Experiência com Aplicações de Edição 3D *

- (●) Nunca utilizou
- ( ) Já utilizou pelo menos uma vez
- ( ) Utiliza raramente
- ( ) Utiliza frequentemente
- ( ) Utiliza regularmente

Teste 1 - Teste Livre

O utilizador vai ter a oportunidade de interagir com ambas as aplicações livremente um período de tempo limitado. (2 minutos na aplicação de Realidade Virtual, 5 minutos no Editor de Exposições) No final deste teste, o utilizador será questionado sobre a sua opinião pessoal relativamente à natureza das aplicações.

Questionário do Teste Livre

O utilizador conseguiu a maioria das funcionalidades são executadas? *

- (●) Sim
- ( ) Não

As interfaces das aplicações foram intrusivas? *

- (●) Sim
- ( ) Não

As interfaces das aplicações foram úteis e descritivas? *

- (●) Sim
- ( ) Não

As aplicações funcionaram como o utilizador esperaria que funcionassem, baseado experiência prévia ou dedução lógica? *

- (●) Sim
- ( ) Não

Teste 2 - Avaliação de Funcionalidades Específicas

O utilizador será solicitado a realizar uma série de cinco tarefas crescentemente complexas que abrangerão todos os aspetos das aplicações.

1 - Na aplicação da galeria, virado para uma parede, o utilizador deve navegar pelas obras atualmente em exposição (Botão HOME para abrir o menu) e pesquisar por "The Starry Night". De seguida, o utilizador deve voltar-se para as obras, e aproximar-se da obra desejada.

2 - No Editor de Exposições, começando com 3 objetos amarelos e 3 objetos verdes, o utilizador deve selecionar 2 objetos verdes e 1 objeto amarelo em simultâneo.

3 - No Editor de Exposições, começando com 3 objetos amarelos e 3 objetos verdes, o utilizador deve eliminar 2 dos objetos amarelos e substituí-los por 2 cópias dos objetos verdes.

4 - No Editor de Exposições, começando com 6 cubos das mesmas dimensões, girados aleatoriamente, o utilizador deve construir uma pirâmide de 6 degraus, com cubos cada vez menores.

5 - No Editor de Exposições, o utilizador deve importar uma instância da Sala "Room A.obj", duas instâncias do Objeto 3D "Model A.obj", três instâncias do Objeto 3D "Model B.obj" e uma instância do Objeto 2D "Object A.png". Ao importar o Objeto 2D "Object A.png", as dimensões do objeto devem ser 120 por 120 cm. Por fim, o utilizador deve guardar a exposição.

Questionário da Avaliação de Funcionalidades Específicas

O utilizador conseguiu realizar todas as tarefas necessárias? *

- (●) Sim
- ( ) Não

O utilizador necessitou de ajuda do programador para realizar alguma das tarefas? *

- (●) Sim
- ( ) Não

Se respondeu "Sim", quais?

- [ ] Tarefa 1
- [ ] Tarefa 2
- [ ] Tarefa 3
- [ ] Tarefa 4
- [ ] Tarefa 5

As tarefas foram simples e diretas? *

- (•) Sim
- ( ) Não

Se respondeu "Não", quais destas tarefas é que se revelaram mais desafiantes?

- [ ] Tarefa 1
- [ ] Tarefa 2
- [ ] Tarefa 3
- [ ] Tarefa 4
- [ ] Tarefa 5

O utilizador mudou de opinião relativamente à aplicação desde o Teste Livre anterior? *

- (•) Sim
- ( ) Não

Se o utilizador pudesse fornecer qualquer feedback relativo à aplicação, qual seria? (Opcional)

Sample Form

Questionário de Experiência do Utilizador (QEU)

Este questionário serve para avaliar aspetos de usabilidade e de experiência do utilizador das aplicações.

É constituído por pares de
opostos relativos às propriedades que o sistema possa ter. As graduações entre os opostos são
representadas por círculos.

Atratividade *

	1	2	3	4	5	6	7	
Desinteressante	◉	○	○	○	○	○	○	Atrativo

Clareza *

	1	2	3	4	5	6	7	
Incompreensível	◉	○	○	○	○	○	○	Compreensível e Evidente

Eficiência *

	1	2	3	4	5	6	7	
Ineficiente	◉	○	○	○	○	○	○	Eficiente

Previsibilidade *

	1	2	3	4	5	6	7	
Imprevisível	◉	○	○	○	○	○	○	Previsível

Estimulação *

	1	2	3	4	5	6	7	
Desmotivante	◉	○	○	○	○	○	○	Motivante

Novidade *

	1	2	3	4	5	6	7	
Convencional	◉	○	○	○	○	○	○	Original

Questionário Concluído

Obrigado pelas respostas!