Technical University of Crete

DIPLOMA THESIS

AR-Apt: Architectural Participatory Design of Buildings in Augmented Reality

Author:
Tzortzi Maria-Eleni

Committee:
Mania Aikaterini
Vazakas Alexandros
Giatrakos Nikolaos

A thesis submitted in fulfillment of the requirements
for the degree of Electrical and Computer Engineering

Abstract

Maria Eleni Tzortzi

AR-Apt: Architectural Participatory Design of Buildings in Augmented Reality

This thesis presents the development of AR-Apt, an Augmented Reality (AR) application designed to facilitate architectural design and participatory planning of apartments and buildings. The application enables real-time collaboration between expert users, such as architects, and non-expert users, including future residents, allowing them to contribute to the design process. By utilizing Microsoft HoloLens 2, the system provides an interactive and immersive design experience, where users can visualize, customize, and refine building structures collaboratively.

The AR-Apt system follows a three-phase design process: (1) defining the abstract structure, where architects establish the initial framework of the building; (2) user participation in spatial design, enabling civilians to modify and organize living modules in an augmented environment; and (3) customization and finalization, allowing users to refine individual layouts and facades to enhance personalization.

A visualization interface is implemented to offer scaled-down model views, improving comprehension and interaction. The Mixed Reality Toolkit (MRTK3) and Unity Engine were chosen for system development, ensuring seamless interaction, user-friendly controls, and a scalable platform for future enhancements.

The implementation of AR-Apt was built using the Unity Engine, leveraging its robust real-time rendering capabilities and seamless integration with AR technologies. The Mixed Reality Toolkit (MRTK3) was employed to facilitate user interactions, including hand gestures, eye tracking, and spatial mapping, ensuring an intuitive user experience. The system architecture utilizes Unity's Netcode for GameObjects for real-time synchronization between multiple users, enabling seamless collaborative interactions in a shared AR space.

Additionally, Unity's Relay and Lobby services were integrated to establish reliable multiplayer networking, allowing users to create and join design sessions without complex network configurations. For data persistence and retrieval, Unity Cloud Save was utilized, ensuring that user modifications and design iterations could be stored and accessed across sessions. The application's interface was developed with MRTK's UI components, providing a volumetric and adaptable interface for immersive user interaction. The application also implements custom-built scripts for scene

management, user authentication, and real-time modification of architectural models. By combining these tools and frameworks, AR-Apt delivers a high-fidelity, scalable, and efficient AR experience tailored for collaborative architectural design.

A user evaluation was conducted to assess usability, interaction techniques, and collaboration efficiency. The results indicate that AR-Apt successfully enhances participatory planning by bridging the gap between architectural professionals and non-expert users, fostering a more inclusive and democratic approach to urban development.

Future work will explore real-scale walkthroughs, advanced customization options, and improved real-time collaboration features to further enhance user experience and architectural engagement.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Mania Aikaterini, for her guidance, patience, encouragement and support throughout the course of this research.

I would also like to thank, Vazakas Alexandros, for selflessly providing his architectural expertise and guidance and invaluably helping shape this thesis.

I would also like to express my sincerest gratitude to the architect Eleftheria Papadosifou for endlessly brainstorming with me and helping shape this work. Also for her beautiful 3D designs, fundimental to this thesis.

My deep gratitude to the entire SURREAL TEAM for their acceptance, help and guidance and encouragement.

My thanks to the people that housed me for the duration of this thesis. Thank you Michalis, Eirini, Christina and Giannis for opening up your house to me.

A massive thank you to Perakis Giorgos for his insight, support, encouragement, testing and more testing. For always being there to help and being a constant to depend on.

Finally, I wish to express my heartfelt appreciation to my family for their continued sacrifices, support, love and confidence in me. Without all of you this wouldn't be possible.

Contents

List of Figures

List of Abbreviations

AR	**A**ugmented **R**eality
VR	**V**irtual **R**eality
HMDs	**H**ead **M**ounted **D**isplays
CSCW	**C**omputer **S**upported **C**ollaborative **W**ork
VST	**V**ideo **S**ee **T**hrough
OST	**O**ptical **S**ee **T**hrough
DR	**D**iminishing **R**eality
UI	**U**ser **I**nterface
UX	**U**ser e**X**perience
MR	**M**ixed **R**eality
2D	2**D**imentional
3D	3**D**imentional
SDK	**S**oftware **D**evepoment **K**it
DOF	**D**egrees **O**f **F**reedom
MRTK	**M**ixed **R**eality **T**oolkit
MRTK3	**M**ixed **R**eality **T**oolkit 3
PBR	**P**hysically **B**ased **R**endering
API	**A**pplication **P**rogramming **I**nterface
PUN	**P**hoton **U**nity **N**etworking
NAT	**N**etwork **A**ddress **T**ranslation
DR	**D**iminishing **R**eality
RT	**R**eal **T**ime
IDE	**I**ntegrated **D**evelopment **E**nvironment
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
RPC	**R**emote **P**rocedure **C**all
SUS	**S**ystem **U**sability **S**cale
P2P	**P**eer **T**o **P**eer

Introduction

Augmented Reality (AR) is revolutionizing how individuals interact with the world, seamlessly blending digital elements into the physical environment. From gaming and entertainment to industrial applications and education, AR is transforming various sectors, enhancing productivity, and creating immersive experiences. The manufacturing industry, for instance, is increasingly adopting AR technologies to streamline processes, improve training, and enhance worker safety. In the realm of education, AR has proven to be a valuable tool, offering interactive and engaging learning experiences that cater to diverse learning styles.

This thesis explores AR on the field of architectural design and participatory planning. An innovative AR application is proposed to facilitate collaboration between architects and civilians in the context of collective housing projects. By harnessing the capabilities of cutting-edge AR headsets like the Microsoft HoloLens 2, the application enables users to actively participate in the design process, visualizing ideas in real-time and fostering a more inclusive and democratic approach to architectural development.

1.1 Brief Outline

AR-Apt, an Augmented Reality (AR) application, is designed for architectural design and participatory planning of apartments and buildings. The application enables real-time collaboration between expert users, such as architects, and non-expert users, including future residents, allowing them to contribute to the design process. By utilizing Microsoft HoloLens 2, the system provides an interactive and immersive design experience, where users can visualize, customize, and refine building structures collaboratively.

The AR-Apt system follows a three-phase design process:(1) Defining the abstract structure, where architects establish the initial framework of the building. User participation in spatial design, enabling civilians to modify and organize living modules in an augmented environment. (2) Customization and finalization 1.1, allowing users to refine individual layouts and facades to enhance personalization.

(3) A Visualization and Rate interface, is implemented to offer scaled-down model views, improving comprehension and interaction. The Mixed Reality Toolkit (MRTK3) and Unity Engine were chosen for system development, ensuring seamless interaction, user-friendly controls, and a scalable platform for future enhancements.

AR-Apt offers a variety of features and functionalities to enhance the architectural design process and promote user participation. Multiple users, including architects

FIGURE 1.1: Apartment Customization.

and future residents, can simultaneously interact and contribute to the design process, fostering communication and shared understanding. This collaborative approach allows for diverse perspectives and ideas to be incorporated into the design, leading to more inclusive and user-centered outcomes. The application leverages Unity Engine and MRTK3 to enable real-time collaboration between expert and non-expert users in the design process. Users can visualize and manipulate architectural elements in real-time, enhancing communication between designers and stakeholders. They can also provide feedback on proposed layouts through rating and comment functionalities, fostering a dynamic participatory design process.

The application leverages the advanced capabilities of Microsoft HoloLens 2, such as hand tracking, eye tracking, and spatial mapping, to enable natural and immersive manipulation of architectural elements. Users can intuitively interact with virtual objects, making the design process more engaging and accessible. Gesture-based controls and eye-tracking enhance interactivity, ensuring intuitive manipulation of design elements.

AR-Apt provides a visualization feature, allowing users to view a scaled-down architectural model. This offers an in-depth perspective of individual living spaces, enhancing comprehension and facilitating better design decisions. Users have the freedom to modify and arrange apartment layouts, customize interior designs to suit individual preferences.This level of customization empowers users to actively participate in shaping their living spaces, fostering a sense of ownership and personalization. The application allows architects to place foundational elements in AR, guiding users through structured customization. The interface offers different levels of detail, from broad structural adjustments to fine-tuned interior modifications. These steps collectively contribute to a comprehensive participatory design framework, ensuring both professionals and non-experts actively shape the final product.

The application incorporates a feedback and rating system 1.2, allowing users to provide input on proposed layouts through rating and comment functionalities. This fosters a dynamic participatory design process, where user feedback is actively sought and incorporated into design iterations. Thus, users can provide feedback on proposed layouts through rating and comment functionalities, fostering a dynamic participatory design process.

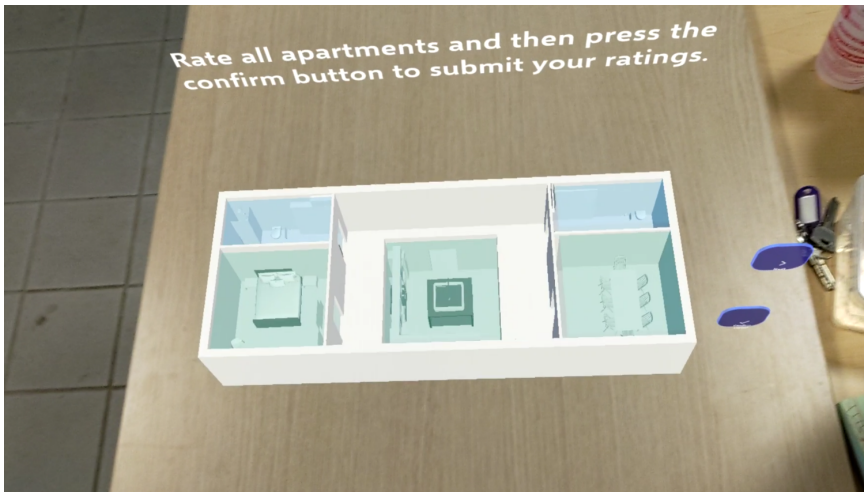


FIGURE 1.2: Visualize and Rate Interface.

FIGURE 1.3: Collaborators' Awareness Cues.

Collaborator awareness cues 1.3 were implemented to enhance user presence and understanding within the mixed reality environment by incorporating customizable avatars, animated user hands, and real-time interaction feedback. Avatars allow users to visually identify each other with synchronized positioning and color customization, while animated hands dynamically update based on user actions, displaying gestures for selecting, manipulating, or interacting with objects.

The development and implementation of AR-Apt involved utilizing cutting-edge technologies and tools to create a robust and user-friendly application. The application is built using the Unity Engine, leveraging its powerful real-time rendering capabilities and seamless integration with AR technologies. The Mixed Reality Toolkit (MRTK3) is employed to facilitate user interactions, including hand gestures, eye tracking, and spatial mapping, ensuring an intuitive user experience. Unity Engine and MRTK3 serve as the foundation for rendering and input processing, enabling hand gesture recognition, spatial mapping, and real-time adjustments. To enable real-time collaboration, AR-Apt utilizes Unity's Netcode for GameObjects, which ensures seamless synchronization between multiple users interacting in a shared AR space. Additionally, Unity's Relay and Lobby services are integrated to establish reliable multiplayer networking, allowing users to create and join design sessions

without complex network configurations. The use of Unity's Netcode for GameObjects ensures seamless synchronization across multiple users, allowing for efficient collaborative sessions. Networking services like Unity Relay and Lobby streamline the connection process, facilitating session-based design discussions without the need for complex server configurations.

For data persistence and retrieval, AR-Apt utilizes Unity Cloud Save, ensuring that user modifications and design iterations can be stored and accessed across sessions. This allows users to continue their design process from where they left off, without losing any progress. Data persistence is managed through Unity Cloud Save, allowing design iterations to be stored and retrieved, ensuring that users can continue their design process across multiple sessions without losing progress.

To assess the usability, interaction techniques, and collaboration efficiency of AR-Apt, a user evaluation was conducted. The evaluation involved architects and non-expert participants engaging with the system to test its functionality in real-world scenarios. Participants reported improved engagement and comprehension compared to traditional architectural planning methods. The ability to visualize spatial arrangements in AR significantly enhanced communication between professionals and users. Findings from the study highlighted several benefits, including an increase in user confidence when making design choices and a reduction in misunderstandings that typically arise from 2D blueprints. The results indicate that AR-Apt successfully enhances participatory planning by bridging the gap between architectural professionals and non-expert users, fostering a more inclusive and democratic approach to urban development.

Future work on AR-Apt will explore the implementation of real-scale walkthroughs, allowing users to experience their designs in life-size AR environments. Advanced customization options, such as the ability to modify textures, materials, and furniture, will be explored to further enhance user engagement. Improved real-time collaboration features, including more precise synchronization and enhanced feedback mechanisms, will be investigated to ensure a smoother and more intuitive collaborative design experience. Future research and development efforts will focus on increasing customization capabilities, refining real-world integration, and further enhancing the scalability of multi-user collaboration to ensure that AR-Apt continues to evolve as an innovative solution for participatory design.

1.2 Thesis Structure

Introduction

This chapter provides an overview of the thesis, including the problem statement, objectives, and motivation behind the development of an augmented reality-based participatory design system. It introduces the challenges of participatory planning in architecture and highlights the benefits of integrating AR for real-time collaboration between expert and non-expert users. The chapter also presents the goals of the AR-Apt system and the intended contributions of this research.

Background and Research Overview

This chapter introduces key concepts related to augmented reality, collaborative design, and participatory planning. It provides an overview of the evolution of AR

technologies, discussing their applications in architecture and urban planning. Additionally, it reviews interaction techniques such as hand gestures, gaze tracking, and voice commands. Relevant research studies on collaborative AR and participatory design methodologies are also analyzed.

Technological Background and System Components

This chapter delves into the tools and technologies used in the development of the AR-Apt application. It discusses game engines, particularly Unity, and the integration of the Mixed Reality Toolkit (MRTK3) for enhanced AR interactions. Additionally, it examines the role of Microsoft HoloLens 2 in providing an immersive and interactive experience, alongside relevant software development kits (SDKs) used for spatial mapping and real-time rendering.

Collaborative System Architecture and Networking

This chapter explains the architecture of the collaborative AR system, focusing on how multiple users interact and communicate within the AR environment. It details the networking solutions, including Unity's Netcode for GameObjects, Unity Relay, and Lobby services, to enable real-time synchronization and multiplayer interaction. The chapter also discusses data management strategies and workspace consistency in a networked environment.

Implementation

This chapter provides an in-depth discussion of the AR system's implementation. It covers the system's UI and interaction models, including the design interface, dwelling customization tools, and dual-visualization features. Technical details of the application's functionalities, such as module placement, spatial awareness, and user authentication, are explained. The chapter also presents the challenges encountered during implementation and the solutions developed.

Evaluation

This chapter presents the methodology and results of the user evaluation conducted to assess system usability and interaction effectiveness. It includes feedback from participants, focusing on the intuitive nature of the interface, ease of collaboration, and overall user experience. The chapter also discusses identified challenges, such as interaction accuracy, UI responsiveness, and real-time synchronization.

Conclusion and Future Work

The final chapter summarizes the key findings of the research, highlighting the contributions of AR-Apt in enhancing participatory architectural design. It reflects on the system's impact on collaborative planning and outlines potential improvements. Future work includes further customization options, real-scale walkthrough implementations, and additional interaction enhancements to improve usability and engagement.

Research Overview

2.1 Augmented Reality

2.1.1 Introduction to Augmented Reality

Augmented reality (AR) is the technology in which virtual objects are superimposed in real environments. In contrast to Virtual Reality (VR), where the user is immersed in a synthetic world, in AR the user interacts with a version of the reality enriched with virtual artifacts.

While early research in the field strictly associated AR with Head Mounted Displays (HMDs), latest research [1] [2] defines AR in a technology independent manner, as all systems that:

- merge and align virtual and real environments,
- render objects in real time,
- run interactively in real time.

2.1.2 Collaborative Augmented Reality

Collaborative Augmented reality is a subcategory of Computer-Supported Collaborative Work (CSCW). Although the field of CSCW is an old concept, the utilization of AR for CSCW has only recently been explored by researchers.

Technology advancements in smartphones and AR-HMDs have enabled researchers to develop more sophisticated, accurate and easy to use applications, both in learning, co-design and in industry settings. Also, the pandemic emphasized the need for remote collaboration and AR is proving to be a very powerful tool, upgrading collaborative experiences worldwide.

Why Collaborative Augmented Reality?

Augmented reality is a field that increases interactivity, enabling users to use gestures, voice commands and glance and gaze queues to interact with virtual objects all the while being able to interact with their physical environment.

Utilizing AR and its many capabilities in a collaborative setting enables users to manipulate virtual Objects, control their own independent viewpoint. Also, real artifacts can be augmented by virtual annotations, enhancing collaboration capabilities.

Collaborative AR taxonomy

Collaborative Augmented Reality applications can be categorised using *time, space, technology* and *role of users* during the collaboration. Thus, the following distinct categories as recognized in [3]:

- **Synchronous/ Asynchronous**
- **Co-Located/ Remote**
- **Symmetrical/ Asymmetrical** both in regard to technology used by each collaborator and collaborator roles

Co-Located Collaborative AR : Users are located in the same physical space. Shared virtual objects are placed in an exocentric manner and located in the same place irrelevant to user position [4].

Remote Collaborative AR : Collaborators are located in different physical environments, yet share virtual objects.

Synchronous Collaborative AR : User collaboration is done *simultaneously* or in synchronous time. The majority of Collaborative AR apps are synchronous.

Asynchronous Collaborative AR : User collaboration is done in asynchronous time. There is little research on the field of asynchronous AR Collaboration.

2.2 Interaction Techniques in Augmented Reality

Natural and easy user interaction is a very important aspect of successful AR application implementation. Thus, interaction techniques play a vital role in user experience and unless they are intuitive users can be fatigued. Researchers have explored many user input modalities for mixed reality applications [5]. Among those are: touch devices, generic input devices (mice, keyboards), tangible devises, gestures, voice commands and head and gaze interaction techniques.

2.2.1 Hand Gestures as an Interaction Technique

Hand gestures are a versatile input modality in mixed reality apps, commonly used for creation, selection, and manipulation tasks [5]. They can be used to manipulate objects both near the user (direct manipulation) and far from the user (point and commit with hands). Combining hand gestures with other interaction techniques, such as gaze or voice cues, can increase their accuracy. However, hand gestures can become ambiguous in collaborative environments, where they may be used for non-verbal communication or private views during collaboration.

The most common hand gestures are designed to be instinctual for users. Some of them include the pinch, open hand, point and swipe gestures.

Direct object manipulation is a hand gesture input model that involves touching holograms directly, allowing objects to behave just as they would in the real world. This input model is user-friendly, with no symbolic gestures to teach users. All interactions are built around visual elements that can be touched or grabbed, making it a "near" input model best used for interacting with content within arms reach.

Finally, **Point and Commit with Hands** is a unique "far" interaction technique in mixed reality that allows users to target, select, and manipulate 2D and 3D content that is out of reach. This technique breaks the physical constraint of the real world, allowing users to interact with far objects as if they were close by. While not something that humans can naturally do, this virtual enhancement makes user interactions in mixed reality more efficient.

2.2.2 Eye Based Interaction Techniques

Eye gaze and head gaze have been explored for a natural hands-free and noise-independent user interaction, in situations where hands are occupied or voice commands are unreliable. Researchers are utilizing gaze as a selection technique [5] and as an interaction technique for user interfaces [6, 7, 8].

Gaze Based Input Models

Gaze and dwell, available in two variants, allows users to easily select targets using head or eye-gaze and perform actions through extended dwell. Gaze and commit, similar to the point and click model with the mouse, involves selecting targets with head or eye-gaze and committing to an action through voice commands, hand gestures, button clicks or extended dwell, making it ideal for manipulating holographic content that is out of reach.

Eye Gaze

Eye gaze is being used to implement adaptable user interfaces with: (1) variable levels of information and (2) natural transitions between the virtual context displayed [6] [8] while minimizing the obstruction of real world content and virtual crowding [7].

Using eye-gaze as an input has several benefits, including high speed pointing, low effort, implicitness, and providing an alternative input channel. Eye-gaze can also help infer what a user is paying attention to, aiding in various application areas. However, there are some challenges to consider when working with eye-gaze input, such as the need to combine it with other inputs to confirm the user's intent, the double role of eye-gaze for input and control, and issues related to target selection and eye tracking accuracy. Designers should consider these challenges and use best practices to create a pleasant and comfortable user experience.

Head Gaze

Head-gaze is an input method that involves targeting an object with a user's head direction. This input method is great in scenarios where a person's hands are busy with other tasks, and voice commands are difficult due to environmental or social constraints.

Head-gaze targeting is controlled and explicit, but it can be fatiguing and cause discomfort to the user. It requires showing a cursor, and it's slower than eye-gaze targeting. However, head-gaze targeting is reliable and can be a great fallback.

Head-gaze and dwell is a familiar and easy-to-master mechanism. However, too much feedback in head-gaze and dwell interactions can induce anxiety, and the yo-yo effect can cause uncomfortable head movements.

2.2.3 Voice as an Interaction Technique

Voice as an input modality is usually utilized in simple tasks such as *activation, selection and deletion* of objects. It can be utilised as confirmation to preform more complex tasks, in combination with other input modalities such as gaze and gestures [5]. In a collaborative setting with audio communication between collaborators, voice commands can prove confusing. Voice cues can be incorporated in applications that support private working environments.

Voice input is a natural and efficient way to communicate with holograms without using hand gestures. It can save time, reduce effort, and minimize cognitive load for users. Voice input is especially useful for traversing complex interfaces and performing tasks while multitasking or when hands are full. However, there are also challenges with using voice input, such as fine-grained control, reliability of detection, and social acceptability in shared spaces. Additionally, learning voice commands and dictating unique or unknown words can be difficult.

2.2.4 Other Interaction Techniques

Input Devices like mice and keyboards are often used in asymmetrical AR applications. **Touch** has also been explored as an input modality by the use of touch devices such as tablets and smartphones or by the projection of user interfaces on the users body. **Tangible devices** are another input modality where designers usually incorporate physical devices for specific gesture needs in applications (i.e scalpel in augmented surgery simulation). More that one input modalities can be combined into **Multimodal Interaction** techniques to increase input accuracy. Some often used combinations of inputs are gaze and voice or gesture and voice.

2.3 Augmented Reality Technologies and Devices

Augmented reality technologies are divided into the following main categories:

- **Video see through (VST)** record the real world and render and project an augmented version of it in a virtual screen. Most common VST AR devices are hand held touch screen devices such as tablets and smartphones.
- **Optical see through (OST)** use transparent devices to project virtual holograms of objects on the real world. OST devices are often mounted on the head of the user and are thus named Head Mounted Displays (HMDs). Some examples of HMDs are the Microsoft Hololens, Hololens 2 and Magic Leap One.
- **Diminishing Reality (DR)** In contrast to the two previous technologies, in diminishing reality (DR) objects from the real environment are removed and the real environment obstructed by them is reconstructed using machine learning. Diminishing Reality is the least researched technology in the field of Augmented Reality.

2.3.1 Augmented Reality Head Mounted Displays (HMDs)

In recent years, the market for AR Head-Mounted Displays (HMDs) has seen significant growth, with several notable devices available to consumers and developers.

These devices offer varying features, capabilities, and price points, catering to different use cases and preferences. Some of the most prominent AR HMDs on the market include:

Microsoft HoloLens

Released in 2016, the first generation HoloLens [9] was a groundbreaking AR headset that paved the way for its successor. It featured a 34-degree diagonal field of view, a resolution of 1280x720 pixels per eye, and an Intel Atom processor. While less powerful than the HoloLens 2, it was still capable of delivering impressive AR experiences and was primarily aimed at enterprise and developer use cases.

FIGURE 2.1: Microsoft HoloLens (1st gen)[9]

Microsoft HoloLens 2

A cutting-edge AR headset designed for enterprise and industrial applications. It boasts advanced hand tracking, eye tracking, and spatial mapping capabilities, enabling users to interact with holograms in a natural and intuitive way. The HoloLens 2 [10] is known for its high-quality visuals, comfortable design, and robust software ecosystem. Compared to the first generation, the Hololens 2 has a wider field of view (52 degrees diagonal), higher resolution (2048x1080 pixels per eye), and a more powerful processor (Qualcomm Snapdragon 850). It also includes eye-tracking and iris scanning for user profiling.

Magic Leap 1

A consumer-focused AR headset that emphasizes immersive experiences and creative applications. It features a unique light field display technology that produces realistic and vibrant holograms. The Magic Leap 1 [11] also offers a wide field of view (up to 50 degrees diagonal) and a comfortable fit, making it suitable for extended use. It is a standalone device with its processing unit (Lightpack) worn on the hip, making the headset itself lighter and more comfortable for extended use.

FIGURE 2.2: Microsoft HoloLens 2 [10]

The Magic Leap Control is a 6-DOF (Degrees Of Freedom) controller that offers precise interaction.

FIGURE 2.3: Magic Leap One HMD [11]

FIGURE 2.4: Meta Quest 3 HMD [12]

Meta Quest 3

A versatile VR headset that also offers mixed reality capabilities through color passthrough. It is a standalone device with a focus on gaming and entertainment, but its mixed reality features make it a potential contender in the AR space. The Quest 3 [12] boasts improved visuals, a more comfortable design, and a wider field of view compared to its predecessors.

Apple Vision Pro

A high-end mixed reality headset that seamlessly blends digital content with the real world. It features a sleek design, advanced eye-tracking and hand-tracking capabilities, and a wide field of view. The Vision Pro [13] is powered by Apple's M2 chip and a dedicated R1 chip for real-time sensor processing, promising exceptional performance and immersive experiences.

2.3.2 Comparison and Device Selection

Each of these AR HMDs has its own strengths and weaknesses, making them suitable for different applications and user preferences. When choosing an AR HMD, it is important to consider factors such as the device's features, capabilities, price, and target audience.

For the purposes of this thesis, the Microsoft HoloLens 2 is the most suitable AR HMD due to its advanced features, robust software ecosystem, and focus on enterprise and industrial applications. The HoloLens 2's hand tracking, eye tracking, and spatial mapping capabilities will enable the development of a highly interactive and

FIGURE 2.5: Apple Vision Pro HMD [13]

intuitive user interface for the collaborative architectural design application. Additionally, the HoloLens 2's comfortable design and high-quality visuals will ensure a positive user experience for both expert and non-expert users.

2.4 Augmented Reality in Architecture

2.4.1 AR Architecture Experiences

In recent years, the advancement of Augmented Reality research has resulted in greater adoption of AR technologies in the architectural engineering field. AR technologies are utilised in architecture education [14] and the architectural engineering industry has incorporated AR technologies in all stages of the architectural process from planning [15] to fabrication and construction [16]. Below are presented some important AR applications on the field.

Holographic Construction

(AR 3D holographic instruction for assembly)

In their work [17] Jahn et.al developed an application that generates holographic construction information from parametric models in AR allowing workers to assemble complex shape brick walls.

FIGURE 2.6: Holographic construction [17] .

Making in Mixed Reality

(AR 3D holographic instruction for fabrication)

Another AR architectural application [18] that enables creation of interactive holographic instructions for complex shape construction. The application assists the design, fabrication and assembly of complex steel designs by using on-site AR fabrication guidance.

Pop Up Factory : Collaborative Design in Mixed Reality

(AR immersive design)

In their work [19] Betti et.al propose a mixed reality collaborative approach to design, fabrication and assembly on site, through a custom, multi-modal interface. User input is live-streamed and channeled to 3D modelling environment, on-demand robotic fabrication and AR-guided assembly.

FIGURE 2.7: Making in Mixed Reality [18].

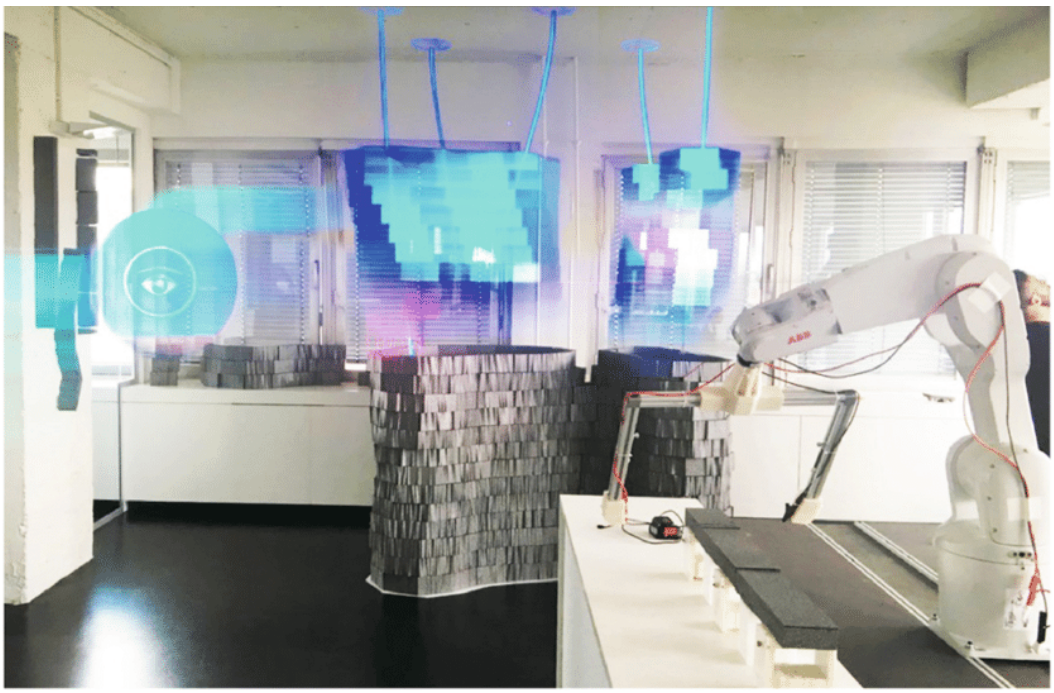


FIGURE 2.8: Pop Up Factory [19]

ARtect, an augmented reality educational prototype for architectural design

(AR architectural design in education)

ARtect [20] is an Augmented Reality visualization tool that enables users to visualize custom made 3D models and 2D graphics in real environments using a custom interface.

An Architecture for Mobile Outdoors Augmented Reality for Cultural Heritage

(AR historical cultural sites visualization)

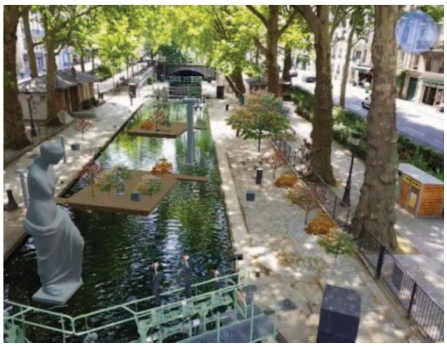


Fig. 7. Augmented scene result evaluation test height and ground tracking from bridge, Canal Saint-Martin, Paris, photo: June 2020.

Fig. 9. AR scene, Back view of Venus of Milos.

FIGURE 2.9: ARtect, An augmented reality prototype for architectural design. [20]

In this work [21] Panou et.al. developed a mobile AR application enabling users to navigate the Venetian part of Chania, Crete, Greece, using gamification, while visualizing historical buildings in their past state and exploring historical information.

2.5 Augmented Reality and Participatory Planning

2.5.1 Participatory Planning.

In traditional urban and architectural planning users are able to provide their input and insight on the resulting designs during late stages of the design process. Thus, altering the design is a painstaking, time-consuming and expensive process. Participatory planning is a democratic process that aims to provide the solution to this problem by involving non-experts in the design proccess. It is a user-oriented approach to urban planning and architectural design during which the users of a space are involved in all stages of the design process collaborating with stakeholders to actively influence the design outcome.

2.5.2 AR in Participatory Planning

Having been familiarized with the concepts of participatory planning and AR collaborative design one can easily infer that combining the two satisfies many specifications of participatory planning and design. Using AR collaborative applications in order to design enables a user-friendly easy design process, bypassing the difficulties users encounter when working with more complicated applications addressed to experts. In addition, on-site and real time design grants non expert users easy visualization of their resulting design during the process that would otherwise need to be gained through experience. Some relevant work on augmented reality participatory planning applications is presented below.

2.6 AR Participatory Planning Experiences

Smart-Phone Augmented Reality for Public Participation in Urban Planning

Figure 2. The Graphical User Interface

Figure 1. Overlay on scale model

FIGURE 2.10: Smart-Phone Augmented Reality for Public Participation in Urban Planning [22]

In this work [22] Allen et al. (2011) propose a smart-phone prototype system as AR architecture instrument for public participation in urban planning. The proposed system focuses on the visualization of proposed 3D architectural designs on existing real-world architecture. Through an interface, the application enables user feedback of their degree of approval on the design.

The Urban CoBuilder – A mobile augmented reality tool for crowd-sourced simulation of emergent urban development patterns

(Asynchronous stakeholder co-design and feedback)

Imottesjo and Kain (2018) developed the mobile MR (Mixed Reality) multiplayer application Urban CoBuilder [23] that enables users to "design urban environment on site", allows crowd sourcing data. Collective results of individual design and planning decisions can be gathered.

Participatory Design Supported with Design System and Augmented Reality

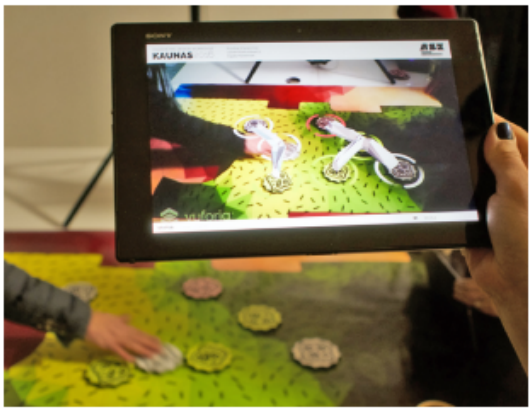


FIGURE 2.11: Participatory Design Supported with Design System and Augmented Reality[24]

In their work [24] Kwieciński et al.proposed a dedicated design system in order to minimize physical interaction between the architect and the users while allowing for customization of design solutions by participants. The design system was made using the GrassHopper software, where a design was created using circle positions as parameters. The design process consisted of a physical model with different colored circles that users could move to customise the resulting design. The design system and the participatory design process were linked with the use of a digital communication interface enabling users to view the resulting augmented design through a tablet.

2.7 Collaborative AR Application Design Specifications

In order to implement a high quality (remote) collaborative AR application, some important aspects, as highlighted by [3, 25], have to be taken into consideration. Some of the issues that need to be addressed include (1)collaborators' awareness and embodiment, (2) action awareness and (3)collaborative workspace consistency and synchronisation.

2.7.1 Awareness In Collaborative AR

In order to achieve seamless collaboration in an AR setting it is crucial for users to have a strong sense of awareness of themselves, their collaborators and their respective actions.

Awareness of Users

During collaboration it is important for every member of the team to be aware of the other team members' placement and intentions in the collaborative space [3], as it can enhance their sense of presence and enrich the remote collaborative experience in unique ways.

Self Awareness of users is very important as it is crucial for a natural collaborative experience. Applications must provide visual cues indicating the direction and target of users' eye gaze [26], especially when it is the main interaction method. Remote users should also be aware of their embodiment and placement in the remote environment.

Collaborator Awareness and Embodiment *Full body avatars, head and hands embodiment* or even *collaborator gaze cues* can be implemented so that users are aware of the position and the direction of their collaborators in the collocated space. Further visual cues can be incorporated to indicate the interfaces or objects on which collaborators are working on. Visual feedback should vary depending on current user collaboration stage or work space (private or public). Designers should utilize the aforementioned methods when necessary so as to avoid information crowding and real world obstruction.

Past work/Action Awareness

Current Action Awareness *Real-time distinct user feedback* for each action has to be provided so users are aware of their current actions and the state of their mixed reality environment at all times. The selection, manipulation or interaction with an object or UI have to each be indicated in a distinct way (blinking of selected objects, highlighting of selected UI buttons, sound effects etc.).

Past Action History Visualisation can be useful in some scenarios, especially on applications with more than one collaborator. Past action history visualization can be implemented in the form of some gaze history indicator [26], in a timeline fashion available on demand or placed on the outskirts of each user vision. Depending on the application, further version control features can be implemented.

2.7.2 Synchronous Workspace Consistency in Collaborative AR

Public/ Shared View Consistency and Synchronization Collaborative AR applications should support synchronous working on public objects and views in real time. Users should be able to work privately, adjust their views and show/hide layers of their environment on demand as pointed out by in [25].

As mentioned above certain version control features can be incorporated in the application to enable easier user collaboration. Undo/redo actions and other workspace manipulation techniques [27] (save, copy, merge workspace or object instance etc.) are just some features that can be implemented.

Real-time Annotation Rendering is a unique capability of augmented reality that enhances remote collaboration. Annotations can: (1) contain crucial information for the collaboration process (creator info, object creation and modification time etc,) (2) aid collaborator communication and guidance in a non-verbal or even asynchronous collaborative environment.

Adaptable and Glancable User Interfaces UI has to be able to adapt based on: (1)user eye gaze focus and lingering [26] (2)user workspace (private or shared) and role. Adaptable user interfaces can be used to display varying levels of information and enable seamless transition between real and virtual content to avoid user fatigue, information crowding and real-world obstruction.

Development Platforms and Networking Solutions for Collaborative AR

This chapter delves into the essential software development kits (SDKs), frameworks, and resources that form the backbone of the AR-Apts MR application. First, the choice of game engines are presented, evaluating platforms such as Unity and the Unreal Engine in terms of their support for MR, ease of integration with collaborative networking, and compatibility with existing AR/VR technologies. Ultimately, Unity was selected for its robust ecosystem, extensive developer community, and dedicated AR/VR toolkit, which streamline the development process and provide access to specialized resources.

Building on Unity's capabilities, a combination of Microsoft's cutting-edge technologies were leveraged, such as the Mixed Reality Toolkit (MRTK3), along with community-driven components and Unity-specific packages. These tools enable a responsive, immersive, and interactive MR experience, facilitating seamless interactions in collaborative environments. Key resources include Unity's Netcode for GameObjects, Unity Relay, Lobby services, Cloud Save and others, which allow for real-time, synchronized multiplayer experiences essential to the application's design.

Through this combination of powerful game engine tools, advanced SDKs, and specialized networking solutions, a comprehensive framework that supports high-performance, user-friendly MR interactions was established.

3.1 3D Graphic Engines for MR Development.

This section provides an analysis of the two most prominent 3D graphics Engines available for Mixed Reality application development: the Unity and Unreal Game Engines. Both engines provide a variety of tools, assets and support for building MR applications and each of them has their own strengths tailored to different project requirements and hardware capabilities. Given the choice of Hololens 2 as the application development device, it is imperative to examine both engines taking into consideration its hardware and processing capabilities and limitations.

3.1.1 Unity Engine

Unity is a widely popular game engine used for developing applications across various platforms, including AR/VR environments. Its extensive cross-platform support, optimization for mobile and standalone devices, and ease of use make it a

favored choice for MR applications, especially those targeting the HoloLens 2.

FIGURE 3.1: Unity Game Engine

Features and Advantages for Collaborative MR on HoloLens 2:

- **Optimized for HoloLens 2:** Unity provides extensive support for the HoloLens 2, including optimizations for its processing limitations and unique sensor array (depth, spatial mapping, hand-tracking). This ensures efficient performance for demanding MR applications.
- **Cross-Platform Compatibility** While optimized for HoloLens 2, Unity supports other platforms (Oculus, Magic Leap, mobile AR), enabling deployment on multiple devices if needed.
- **Rich Ecosystem** Unity's Asset Store offers a vast collection of 3D models, textures, scripts, and plugins, facilitating rapid environment building and optimization for HoloLens 2.
- **Mixed Reality Toolkit (MRTK)** This Microsoft-developed toolkit offers components tailored for HoloLens 2, including hand and eye tracking, spatial awareness, and UI controls optimized for MR. Importantly, MRTK simplifies the development of collaborative features by providing pre-built components for shared experiences, avatar representation, and interaction management.
- **Networking for Collaboration** Unity provides a robust set of tools for creating collaborative MR experiences such as Unity Multiplayer services like Lobby Services, Relay and SaveCloud. These services facilitate real-time multiplayer support, allowing developers to easily implement features like matchmaking and real-time communication within their HoloLens 2 applications. Also, Unity now offers **Netcode for GameObjects** , its own high level API multiplayer networking solution. This package offers a high-level API for building multiplayer functionality, enabling features like shared holographic objects and synchronized interactions between HoloLens 2 users.
- **Performance Optimization** Unity offers tools and resources to optimize MR applications for standalone devices like the HoloLens 2, balancing high visual quality with the limited processing power of wearable MR hardware.

3.1.2 Unreal Engine

Unreal Engine, developed by Epic Games, is renowned for its high-fidelity graphics, often used in applications demanding photorealistic visuals. While it has made strides in AR/VR development, it can be more resource-intensive, potentially posing challenges for standalone MR devices like the HoloLens 2.

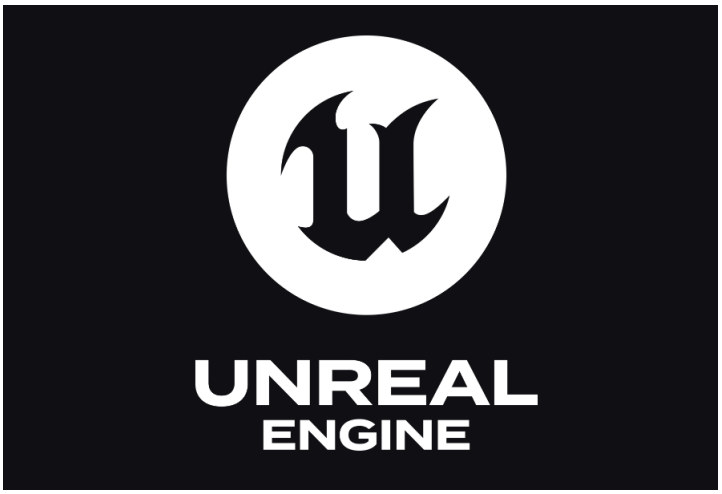


FIGURE 3.2: Unreal Game Engine

Features and Advantages for Collaborative MR on HoloLens 2:

- **Photorealistic Rendering:** Unreal Engine's powerful rendering engine supports physically-based rendering (PBR) and real-time ray tracing, enabling the creation of highly realistic MR environments.
- **Blueprint Visual Scripting:** Unreal Engine's Blueprint system allows for the creation of complex behaviors and interactions through visual scripting, which can be beneficial for designers and artists.
- **AR/ VR Support** Unreal Engine supports ARKit, ARCore, and HoloLens, along with integration with VR devices like Oculus and HTC Vive.
- **Networking and Multiplayer** Unreal Engine's built-in networking capabilities support the development of interactive, synchronized MR experiences across multiple devices, useful for collaborative tools and training simulators.
- **Pixel Streaming** This feature allows Unreal applications to be streamed to devices with lower computing power, potentially enabling high-quality MR experiences on HoloLens 2 even with its hardware limitations.

3.1.3 Comparison and Justification for Choosing Unity

Both Unity and Unreal Engine offer strong support for MR applications, but they cater to different needs and hardware capabilities. Unity is generally preferred for applications on standalone MR devices like the HoloLens 2 due to the following:

- **Ease of Use and Rapid Prototyping:** Unity's user-friendly interface, visual scripting tools, and intuitive networking setup make it easier to learn and use, especially for collaborative MR development. The abundance of collaborative templates and tutorials further accelerates the development process.
- **Strong HoloLens 2 Integration:** The dedicated Mixed Reality Toolkit (MRTK) streamlines HoloLens 2 development by providing pre-built components and interactions optimized for the device. This includes features specifically designed for collaborative experiences, such as shared object manipulation, avatar representation, and spatial audio.

FIGURE 3.3: MRTK for Unity

- **Optimized Performance:** Unity's lightweight architecture and optimization tools are essential for maintaining smooth frame rates and responsiveness in collaborative MR experiences on HoloLens 2, where resource constraints are a major factor. Techniques like culling objects not visible to individual users and optimizing network traffic for shared objects can be readily implemented in Unity.
- **Strong Community and Ecosystem:** Unity boasts a larger and more active community, particularly for HoloLens 2 and MRTK development. This translates to readily available solutions, tutorials, and assistance for collaborative MR projects.

Justification

For this thesis, the Unity engine was deemed the more suitable choice for developing collaborative MR applications on HoloLens 2. Unity's tight integration with the Mixed Reality Toolkit (MRTK) provides a comprehensive framework specifically designed for HoloLens 2, including pre-built components for shared experiences, avatar representation, and optimized interactions. This, coupled with Unity's user-friendly interface, visual scripting tools, and extensive community support focused on collaborative MR, allows for rapid prototyping and efficient development. Furthermore, Unity's lightweight architecture and optimization tools ensure smooth performance and responsiveness, which are critical for collaborative experiences on resource-constrained devices like HoloLens 2.

3.2 Networking Solutions for Multiplayer and Collaborative AR Experiences

This section explores various networking solutions available within the Unity ecosystem for creating multiplayer and collaborative AR experiences, analyzing their strengths, weaknesses, and suitability for different project requirements.

3.2.1 Unity Networking Solutions

Unity offers a suite of native networking tools designed to simplify the development of multiplayer and collaborative applications. These tools include:

- **Unity Authentication:** This service provides a secure and streamlined way to manage user accounts and authentication in collaborative AR applications. It allows users to sign in with various providers (e.g., email/password, Google, Facebook) and handles secure authentication tokens, which can be essential for managing access control and user-specific data in collaborative AR experiences.

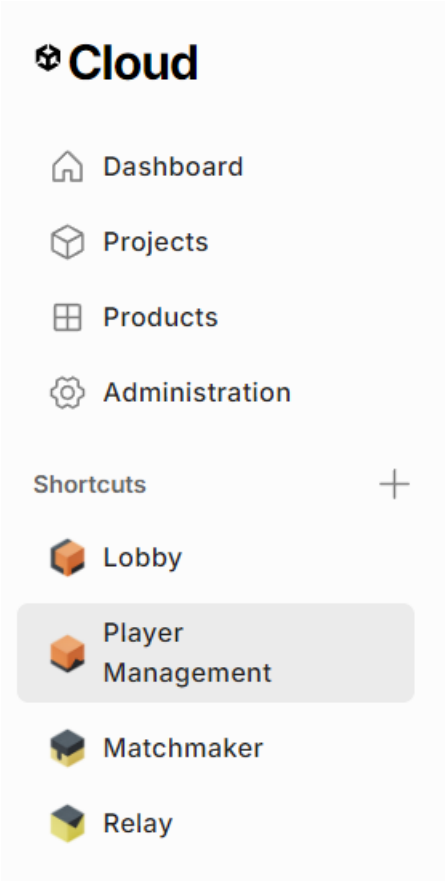


FIGURE 3.4: Unity Services

- **Unity Relay:** A service that facilitates communication between AR devices even when they are behind different network configurations (e.g., firewalls or NAT). This is crucial for establishing robust connections in real-world collaborative AR scenarios..
- **Unity Lobby:** Provides functionalities for creating and managing multiplayer lobbies, enabling users to find and join AR sessions seamlessly
- **Netcode for GameObjects:** A high-level networking API that simplifies the process of synchronizing game objects, player interactions, and game state across multiple AR devices. This allows for shared AR experiences where users can interact with the same virtual objects and environment.
- **Unity Cloud Save:** A cloud-based data storage solution that enables AR applications to securely store and retrieve user data across multiple sessions and devices. It allows for seamless synchronization of user preferences, project states, and design modifications, ensuring continuity in collaborative AR experiences. By leveraging cloud storage, Unity Cloud Save enhances accessibility and persistence, enabling users to resume their work from any device without data loss.

3.2.2 Alternative Networking Solutions

Beyond Unity's built-in tools, several third-party networking solutions provide additional features and flexibility:

- **Photon Unity Networking (PUN):** A popular choice for multiplayer games, Photon offers a simple and efficient solution for establishing connections and synchronizing data between AR clients. However, it may require additional effort to integrate seamlessly with Unity's AR Foundation and MRTK.
- **Mirror Networking:** A community-driven networking solution that builds upon Unity's legacy UNet system. Mirror offers a good balance of ease of use and customization, making it suitable for developers who need more control over network communication and data handling in their collaborative AR applications.
- **Microsoft PlayFab:** A comprehensive cloud-based platform that provides backend services for multiplayer games, including matchmaking, player authentication, and data storage. PlayFab integrates well with Unity and can be a valuable tool for managing the infrastructure of larger-scale collaborative AR experiences.

In this thesis, Lobbies were created using the Unity Lobby Service. Using the lobby information, the relay was created without further user input. A Netcode session was created for the collaborative user aspect, and data was saved using the Cloud Save service. Netcode for GameObjects was chosen over alternative solutions (e.g., Photon, Mirror) primarily due to its ease of use, seamless integration with other Unity tools (such as AR Foundation and MRTK), and the availability of comprehensive documentation and support. This choice allowed for rapid development and efficient implementation of the collaborative features within our AR application.

3.3 Mixed Reality Toolkit (MRTK3)

In the context of AR development for the Microsoft HoloLens 2, the MRTK3 [28] emerges as a powerful and versatile SDK. MRTK3 is an open-source, cross-platform toolkit designed to accelerate the development of mixed reality applications for various devices, including the HoloLens 2. It offers a wide array of features and components that simplify the creation of immersive and interactive experiences, aligning with the requirements for AR application development outlined previously.

FIGURE 3.5: Mixed Reality Toolkit 3

3.3.1 MRTK's Evolution

MRTK has undergone significant evolution since its inception, expanding its capabilities and platform support:

- **MRTK v1:** The first version was specifically tailored for the first Microsoft HoloLens, pioneering new approaches to UX/UI design in mixed reality.
- **MRTK v2:** This iteration introduced a more modular architecture and expanded cross-platform support, enabling developers to create applications for both HoloLens and other AR/VR systems.
- **MRTK3:** The latest version represents a significant leap forward in terms of performance, modularity, and interaction paradigms. It supports features like dynamic scaling, volumetric UI, and improved 3D object manipulation, all while being highly optimized for resource-constrained devices like the HoloLens 2. MRTK3 promotes code reusability across platforms, reducing development time and complexity.

3.3.2 Benefits of MRTK3 for Collaborative AR

MRTK3 offers numerous benefits for developing collaborative AR applications:

- **Simplified Development:** MRTK3 provides a rich set of pre-built UI components (buttons, sliders, hand menus) and interaction building blocks specifically designed for MR environments. This significantly accelerates development and allows developers to focus on the core functionality of their collaborative AR experiences.

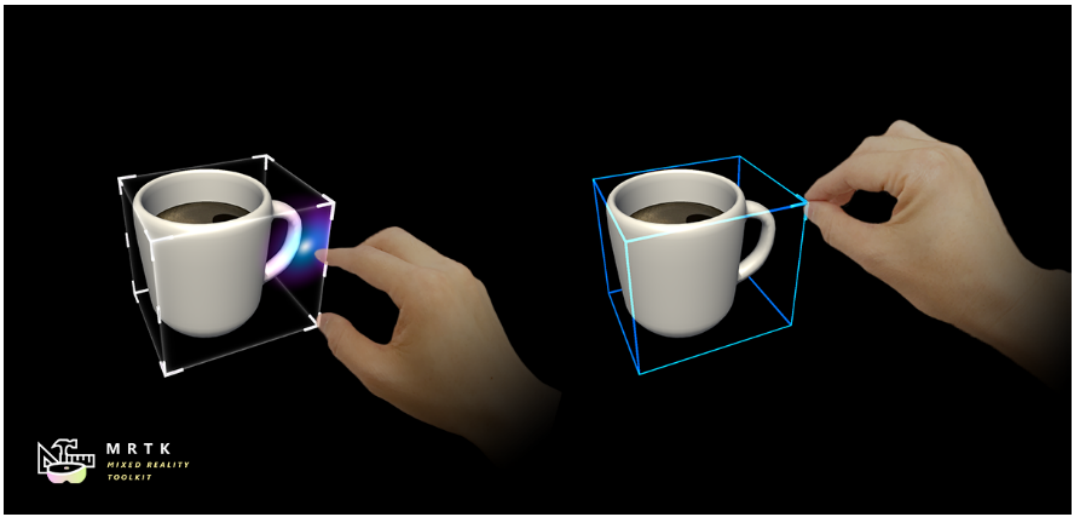


FIGURE 3.6: Mixed Reality Toolkit 3 Manipulations

- **Cross-Platform Compatibility:** MRTK3 supports a wide range of AR/VR platforms, allowing developers to create collaborative experiences that can be easily deployed across different devices, including HoloLens 2, mobile AR devices, and VR headsets.
- **Optimized Performance:** MRTK3 is optimized for performance on resource-constrained devices like the HoloLens 2. This is crucial for maintaining smooth frame rates and responsiveness in collaborative AR applications, where multiple users may be interacting with shared virtual objects and environments.

- **Enhanced User Experience:** MRTK3's support for intuitive interaction models (hand tracking, eye tracking, voice commands) and volumetric UI creates more natural and immersive user experiences in collaborative AR. Users can seamlessly interact with shared virtual objects and UI elements as if they were real physical objects in their environment.
- **Spatial Awareness and Interaction:** MRTK3 provides tools for spatial mapping and scene understanding, enabling developers to create applications that are aware of the physical environment and can seamlessly blend virtual content with the real world. This is essential for collaborative AR experiences where users interact with shared holograms in a common physical space.

3.3.3 Key Features of MRTK3

MRTK3 offers a comprehensive set of features for building collaborative AR applications:

- **UI and Interaction Models:** MRTK3 provides an extensive library of pre-built UI components designed for 3D, volumetric interfaces. These components are optimized for mixed reality interactions, allowing users to manipulate objects using gestures and interact with immersive UIs that are contextually bound to their environment.

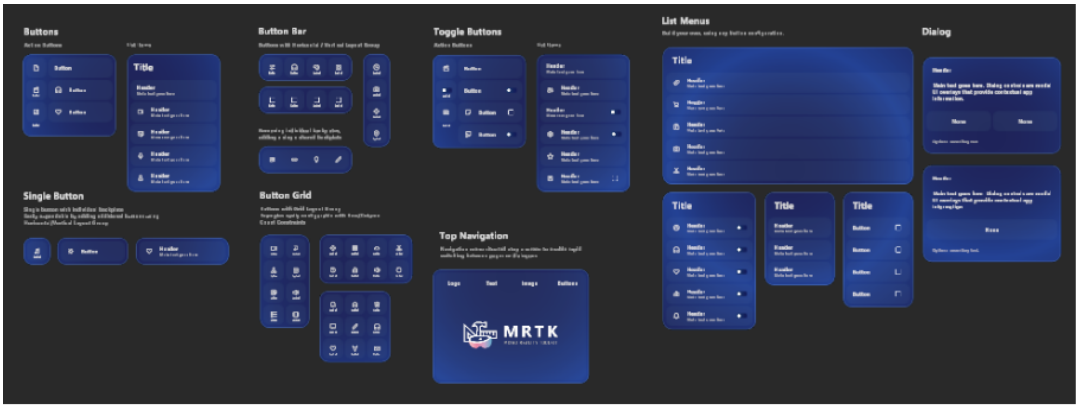


FIGURE 3.7: MRTK3 Volumetric UI

- **Input and Interaction System:** The interaction system supports various input methods, including gaze-pinch, hand rays, speech commands, and motion controllers. This flexibility allows developers to create collaborative AR experiences that adapt to different user preferences and hardware capabilities.
- **Volumetric UI:** MRTK3's support for volumetric UI allows UI elements to be treated as real objects in 3D space. This enhances immersion and enables more intuitive interactions in collaborative AR experiences.
- **Optimized for HoloLens 2:** MRTK3 is highly optimized for HoloLens 2 and other resource-constrained devices, ensuring smooth performance and responsiveness even in complex collaborative AR scenarios.

3.3.4 MRTK3 in this Thesis

MRTK3 played a vital role in the development of the collaborative AR application presented in this thesis. The toolkit's pre-built UI components and interaction building blocks were instrumental in creating an intuitive and user-friendly interface for the HoloLens 2. MRTK3's support for hand tracking, gaze-based interactions, and volumetric UI enabled seamless and natural interactions with the shared virtual environment, enhancing the overall collaborative experience.

3.4 MRTK Dev Template

The MRTK Dev Template [29] is an invaluable tool for both novice and experienced MRTK3 developers. This pre-configured Unity project provides a streamlined starting point, complete with a basic scene setup, sample scripts, and essential interactions already implemented.

In this thesis this template was utilized, so the initial setup hurdles and dive were bypassed and we proceeded straight into prototyping and building the AR-Apt. Furthermore, the template served as an educational resource, showcasing MRTK3's best practices and demonstrating how different components, such as input systems and UI elements, work together.

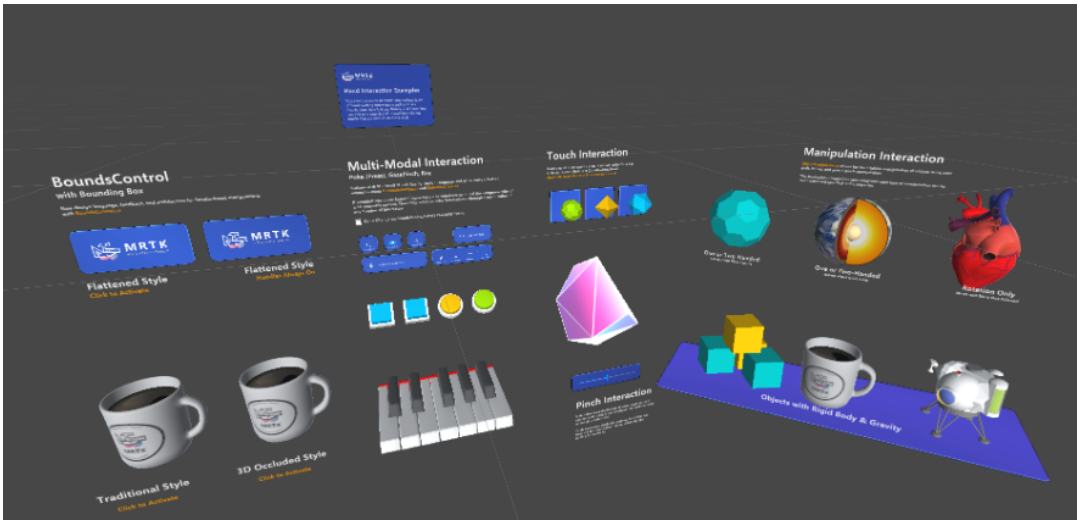


FIGURE 3.8: MRTKDevTemplate Unity Project

3.4.1 Microsoft Visual Studio

In this work, Microsoft Visual Studio, seamlessly integrated with Unity, served as the external script editor for the project. Its robust debugging tools and comprehensive C# support streamlined the development process, allowing for efficient code writing and troubleshooting. The tight integration between Visual Studio and Unity enabled a smooth workflow, where code changes were instantly reflected in the Unity Editor, accelerating iteration and testing cycles. The editor was also used in the later deployment of the application on our Hololens2 device(s).

Requirement Analysis, UX Design and Prototyping

Software and application development is a meticulous procedure and, in order for the application to be successful, development teams have to adhere to certain design steps, methods and practices. This planning and designing stage ensure that the resulting application fulfills its purpose in meeting user needs and requirements and the project adheres to its preset budget and time limits.

In this chapter of the thesis the steps of the UX design process of the application are presented and analyzed. First, a Requirement Analysis is preformed using Personas and Use Cases. After that a prototype user interface and the storyboard of the application were created, using the deducted requirements and use cases. Finally, the User Interface was evaluated by users using the Think Aloud Method, and user feedback was collected and utilized to optimize the application UI.

4.1 Requirement Analysis

Requirement, as defined by the IEEE, is a property (or behavior) that must be exhibited in order to solve some problem of the real world. In order to fulfill the goals of the application all user requirements have to be met.

Requirement Analysis is a nonlinear process, during which the application requirements are specified and analyzed, validated and updated until application stakeholders are satisfied. In the first stage developers can interact with real life users, or create fabricated personas in order to strictly define the application requirements. Following their definition, requirements are validated in cooperation with the application stakeholders, using various methods (use case diagrams, prototyping) and are updated according to user feedback.

4.1.1 Functional and Behavioral Requirements

The main goal of the application is architectural design and participatory planning. The application will enable collaboration between architects (expert users) and civilians (non-expert users) to design and customize apartments and buildings. During each session, one (or more) user(s) will be able to design, while visualizing their design in real time, using a custom design interface deployed on the Microsoft Hololens 2 head mounted display.

During the first phase of the design process, expert users place the hollow building aligning it with a horizontal surface. Following, users can utilize the provided dwelling modules/shapes to collaboratively fill the abstract shape of the building, in a Tetris like fashion, with the dwellings of their choice. Thus, each building is comprised of several modules, each of which can be independently customized by their occupants in collaboration with an architect. During the second phase, users will be able to customize each module layout and facade of the building in collaboration with a consulting architect.

For all described functions a seamless synchronous collaborative process is imperative for the success of the application. **Waiting rooms** will be implemented for easier **user synchronization**. **Internet connection** will be another requirement and when necessary, the **user has to be notified on the status of the internet connection**.

Virtual **content uniformity** and **collaborator awareness** and all other aspects mentioned above for a collaborative application constitute high priority requirements. Another high priority aspect of the application implementation is **intuitive user interaction**, including **easy in app navigation, object placement and general user guidance and cues**.

Glanceable user interfaces and menus, **intuitive user gestures** and a plethora of **visual cues** will be implemented and utilized to enable the user experience and interaction.

In order to achieve this, given the innovative technology on which the AR-Apt app will be deployed, each user can choose to be instructed on both how to use the technology and navigate the application. For this purpose, both a **tutorial on Hololens Gestures and Features** and an **In-app Help feature** have to be implemented and be accessible to the users when necessary via a **Custom Hand Menu** that also has to be implemented.

The application's custom design interface will need to **provide two different visualization techniques**: (1) a real scale, outside view of the building complex on site of the development and (2) a model scale building visualization of the layout of one selected module. Users will be able to toggle on and off both views, scale, anchor them or have them follow them to include whichever view necessary for the achievement of each user task.

4.2 Personas

Personas is a method where software developers create imaginary users of a developing software application, in order to understand their target audience, user needs and motivations. During this phase of developers, fine tune features so they can tailor the application to their users' deduced needs and requirements. At least one persona has to be created for each type of user of the application.

In order to cover the needs of the application of this thesis we need to model people (experts and non experts) looking to co-design a housing or building space, while visualizing it both its facade and inside. Important factors that have to be determined include:

1. **user expertise**: expert user (civil engineer, architect etc.) non expert user (resident of communal housing, stakeholder of a development)

2. **participation in the design**: designers or viewers

Taking into consideration the previous user types, their needs and requirements, the following user personas were crafted, that represent the main user types for the application. These profiles include users' age, gender, general personal information and their general attitude towards technology. Also it will include the requirements the application needs to fill for each of them.

4.2.1 Veloukia, 28 years old, Architect (Expert user, designer)

Veloukia is a young architect that has a Masters Deegree in open building design. She firmly believes user involvement and public participation in building design leads to greater satisfaction rate and greater approval rating of designs. She is sociable and would like to implement her knowledge of open building frameworks to co-design buildings in a participatory manner. She is familiar with the latest technologies and programs for architectural design, but not as familiar with augmented reality headsets. She recognizes the knowledge gap between civilians and architects concerning design software and methods. She would like a software with easier user interaction and more direct visualization to afford users freedom of design and customization while simultaneously operating under a defined framework.

4.2.2 Jason , 19 years old, Student (non-expert user, designer)

Jason is a student in the Technical University of Crete Electrical and Computer Engineering Department and a resident of the university student housing committee. He is an active member of the student body and is interested to design the new student building housing in cooperation with his fellow students and other stakeholders. He has a base knowledge of some architectural design concepts and is a technology enthusiast familiar with the latest developments in AR and VR technologies. He is excited to try the new technologies and input modalities of the application, but would like a simplified version of design software so he can design without their steep learning curve and difficulties of adhering to intricate architectural rules and limitations. He would like to collaborate with his roommates to design their communal space.

4.2.3 John, 55 years old, Executive (non-expert user, viewer)

John is a university executive who oversees housing decisions. He has a heavy workload and limited time to oversee the design process. He is not very familiar with emerging technologies and would like an straightforward interface with intuitive user controls. He wants to oversee the overall design without bothering with detailed design. He is not familiar with architectural model interpretation and can benefit greatly by the visualization of the proposed design.

4.3 Use Cases

Use case of an application is a collection of scenarios that describe main user goals in an application. A use case scenario is a step sequence of user-application interactions necessary to achieve a specific goal.

Use case diagrams are a high-level abstract representation of the main user actions and interactions within a system. They include the main application functionality

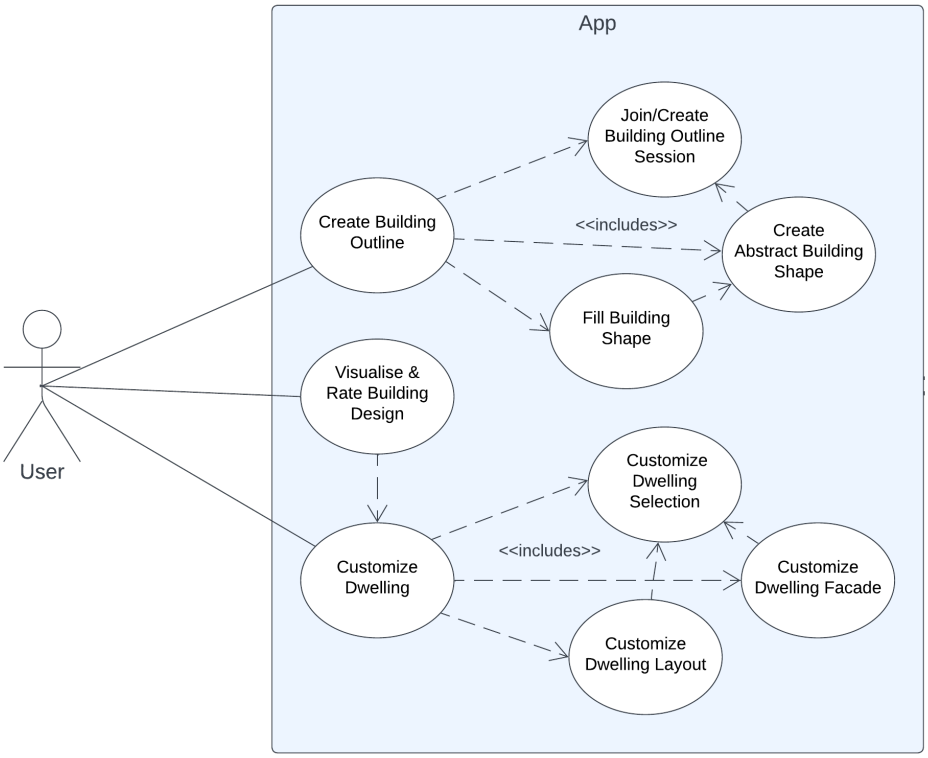


FIGURE 4.1: Application use case diagram for all users.

and its various types of users (actors). They are a simple and straightforward representation of application use cases.

For the purposes of this thesis two types of users of the application can be identified:

- expert users (architects, civil engineers, electricians)
- non-expert users (stakeholders, owners, inhabitants)

A high level use case diagram of the application is presented in fig. 4.1.

A more detailed use case diagram of the application Join/Create Session use case is presented in fig. 4.2.

In this section the main use cases of the AR-Apt application are presented as derived after the application requirement analysis. For each of the use cases a step-by-step description of the actions used to achieve the user goal is provided (menu choices, inputs etc.) as well as the application feedback and reactions to the described user actions.

4.3.1 Design Abstract Building

To enter this use case, users choose the Design Abstract Building option from the Main Menu by pressing the corresponding button. In order to properly describe the steps of this use case, they have been divided, thematically, into three sub-use cases. So users then continue to follow the *Create/Join Design Session* use case where all users create and or join a waiting room, after which expert users can *Place the Building* on

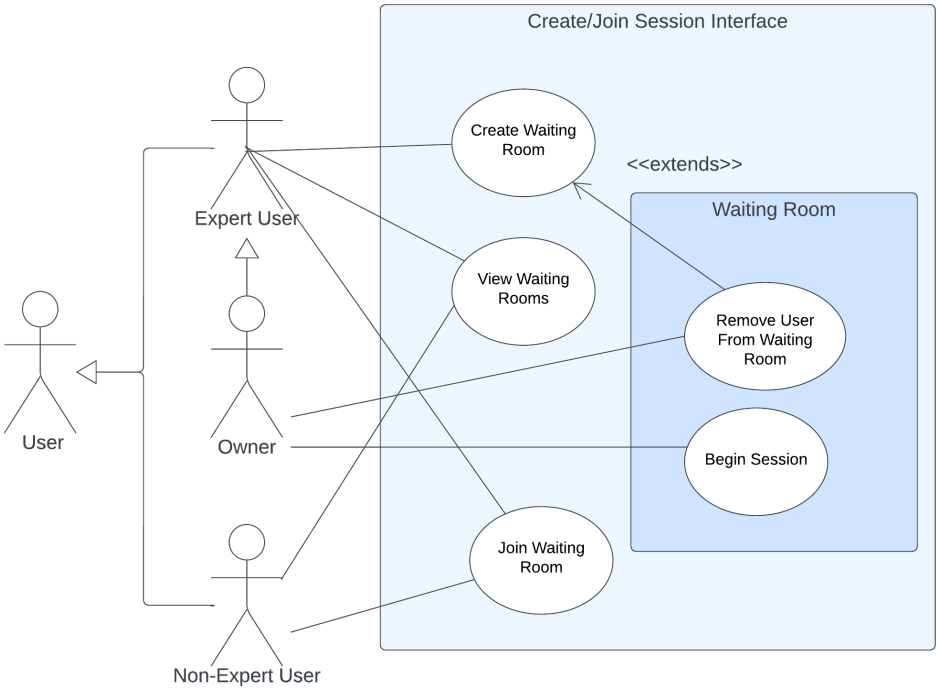


FIGURE 4.2: Join/Create Session use case diagram.

a horizontal surface, as described on the corresponding use case and then all users go on to *Fill Building Shape* use case. When all the steps are done, the end result, a building consisting of various dwellings, is saved and the use case is completed.

4.3.2 Create/ Join Design Session

The application shows users the *Create/ Join Waiting Room interface* where all existing waiting rooms are displayed. Users can either Join an existing Waiting Room or Create a New Waiting Room should they have the authorization to do so.

Expert users only are provided the authorization to Create Waiting rooms whereas non expert users can only Join waiting rooms. Once users are in the *Waiting Room Interface* if they are non expert users they wait for the owner of the session to start the design session. Expert users have the option to remove someone from the waiting room or begin the design session using the corresponding interface buttons. After the owner of the session has started the design process the expert goes on to *Place the Building* on a horizontal surface.

4.3.3 Place the Building

In order to *Place the Building*, users have to have already completed *Create/ Join Building Outline Session* use case as described above. During this phase of the design, expert users can place a 3D building model provided in our design interface in the mixed reality scene. They then can manipulate the building to align it and place it on a horizontal surface. They can press the 'Next' button and finalize the placement. The result of this use case is a hollow building shape, with its common areas marked in red, that will go on to be filled in the next use case.

Application Feedback: The application provides a shared mixed reality space where expert users can manipulate 3D building models using a white bar at its base. The application provides audio feedback when the users interact with the UI.

4.3.4 Fill Building Shape

In order to enter this use case, users have to have already completed the *Place the Building* use case. In this stage, users will synchronously fill the placed hollow building design with a combination of the provided building dwelling modules in a Tetris-like fashion. Each non-expert user will be able to place the module(s) of their choice from a menu of different module (shape) variations. They will be permitted to manipulate their own dwellings but will be required to contribute to the already formed building, as formed by other users, and place their modules on top of them. As mentioned above, expert users will be able to manipulate all modules in order to provide solutions and recommendations to all users. When the building has been successfully filled, the owner of the session will conclude the design session and save the filled building. The end result of this use case is saved as the Finished Building Design and the building module customization is enabled for the corresponding design.

Application Feedback: Users are provided with module variations in a menu. Placing and arranging modules in a Tetris-like fashion is visually indicated. Expert user manipulations and recommendations are visible to all users.

4.3.5 Customize Dwelling Selection

To enter this use case, users choose the *Customize Dwelling* option from the *Main Menu* by pressing the corresponding button. The app responds by showing all lobbies that correspond to designed buildings ready for customization in a folder-like fashion. Users can choose a lobby of the ones displayed by pressing the corresponding button or cancel their choice. When a lobby is selected the *Create/Join Customize Session* use case is identical to the *Create/ Join Design Session*. After the session has begun, a view of the building design as produced at the end of *Design Abstract Building* use case is displayed in virtual scale. The user can choose the dwelling for customization by pointing at it and pinching. This prompts a dialog which ensures the dwelling choice is correct. The users can either confirm their choice or cancel it. When a choice is confirmed the app prompts the user to the *Customize Apartment Layout* where they can choose the layout of the apartment and the room placement of the apartment. After that they are shown the *Room Placement Interface* and can place their desired rooms on the design and save the end result that is the final customization result.

4.3.6 Customize Apartment Layout

The prerequisites of this use case are described in the *Customize Dwelling Selection* use case. After users have begun the customization session, all participants are presented with a menu that contains tabs for each different room type (kitchen, bathroom, living room, bedroom etc.). Users can press at any tab and view the selected room variants. They then can place a selected room on the apartment model. The interface will turn green when a room is at a valid position and red when it cannot be placed in its current position. The users can place any and as many rooms they

wish and then confirm the design. Then, the finished Customize Apartment is saved in the cloud.

This interface contains tabs of each of the main rooms necessary for a living space (bedroom, bathroom, living room, kitchen, etc.). Users are presented with menus containing variations of each room, which can be selected to be placed in the building. At any time, if a user desires so, they can discard or replace selected and placed rooms according to their needs by pressing at the respective room. After users have reached the desired layout, they can save it and exit the *Customize Dwelling Layout* interface.

Application Feedback: Users are guided through module customization with an interface tailored to their selected module. Real-time synchronization and collaborative adjustments are visible to all participants.

4.3.7 Customize Dwelling Facade

The prerequisites of this use case are described in the *Customize Dwelling Selection* use case. After users have begun the customization session, all participants are presented with a *Module Facade Customization Interface* which can be used to synchronously design the facade of the module. In a similar fashion to the building layout, users can collaboratively design the facade of their space independently, customizing its openings and balconies, etc. The corresponding *Façade Menu* will provide users with different components (openings, windows, balconies, shaders, etc.) Users will have to place their desired elements in a predefined grid to complete their building facade. Users can, at any time, discard or modify the facade elements, until the design session is concluded by one of the authorized expert users. When the facade customization process is concluded, the design is saved and the building data is updated.

4.3.8 Visualize and Rate Building Design

To enter this use case, users choose the *Visualize and Rate Building Design* option from the *Main Menu* by pressing the corresponding button. The app responds by showing all lobbies corresponding to buildings ready for visualization in a folder-like fashion. Users can again in a similar fashion choose the apartment they wish to visualize and rate. The user is shown the finished apartments and can rate them through the custom interface. Once the user has rated the designs the results are saved on the cloud.

4.4 Prototyping

Sketching or Rapid Prototyping is a swift and effective method employed by development teams to conceptualize the application's fundamental structure, interfaces, and user interactions across various usage scenarios. This approach serves as a navigational blueprint, guiding users seamlessly toward the successful achievement of their goals within the application.

Prototyping serves as a valuable tool for soliciting rapid user feedback during the initial phases of application development, prioritizing user interaction over high-fidelity details. It constitutes an economical yet indispensable resource for development teams, enabling them to showcase the application's user interface promptly and gather feedback efficiently. This feedback encompasses interface refinements,

feature adjustments, or even the incorporation of new features not initially encompassed in the current prototype iteration.

Initially, a paper prototype was created as a starting point, which later evolved into a more detailed digital prototype within the Figma application. This prototype acts as a tangible representation of the application's envisioned user experience, setting the stage for iterative development and continual improvement.

In the following section, we can witness the final user interface after its transition from the initial paper prototype to a more detailed digital representation in Figma, all of which is based on the use cases described above.

4.4.1 App Start- Hand Menu- Main Menu

In fig. 4.3 we see the app *Start Screen*. A welcoming message is displayed and a button that the user has to press to continue to the app.

In fig. 4.4 we see the *View Tutorial Dialog*, where the user can choose to either view the tutorial or to continue to the app.

In fig. 4.5 the app *Hand Setting Menu* is displayed. The interface consists of five buttons, 2 toggles and a slider. The slider adjusts the volume. The 'Scene Helper' toggle that toggles scene hints. The 'View Action History' toggle that shows the user their previous action history. The 'Undo/ Redo' buttons. The 'Home' button, 'Wireless Network setting' and 'Comment' button, that can be used to leave scene annotations. This menu is accessible at all times and available to the user by turning their palm up.

In fig. 4.6 the app *Choose User Role dialog*, is displayed, where the user chooses between expert and non expert user.

In fig. 4.7 we see the app *Main Menu*, where the there are three buttons with the main app functionalities. The user can choose 'Between Abstract Building Design', 'Building Module Customization' and 'Visualize and Rate' building buttons. There is also a 'Quit App' button on the top right corner of the menu.

4.4.2 Create/ Join Building Outline Session Prototyping

In fig. 4.8 we see the *Create/ Join Waiting Room* Interface as presented to the non-expert user. In this interface the user is presented with as many buttons to join a waiting room as the existing waiting rooms. By pressing a button they join its corresponding waiting room. In top right of the interface there is a 'home' button that, when pressed, returns the user to the main menu.

In fig. 4.9 we see the *Create/ Join Waiting Room* Interface as presented to the expert user. The expert user is presented with the same buttons as the non-expert, plus an additional 'Create Waiting Room Button' on the top right of the interface.

In fig. 4.13 the *No Wi-Fi Connection Screen* is presented. This screen is prompted after any user (expert or not) presses a button to join/ create a waiting room, with no internet connection.

In fig. 4.10 the *Joining Waiting Room Screen* is presented. This screen is prompted after any user (expert or not) presses a button to join a waiting room.

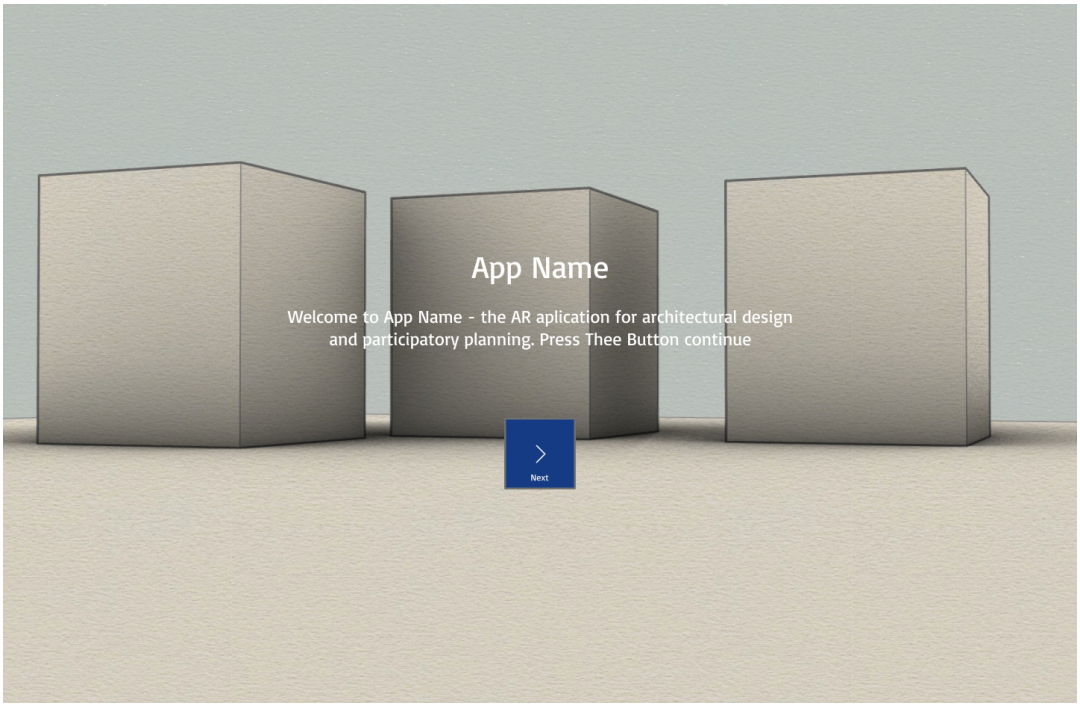


FIGURE 4.3: App Start Screen.

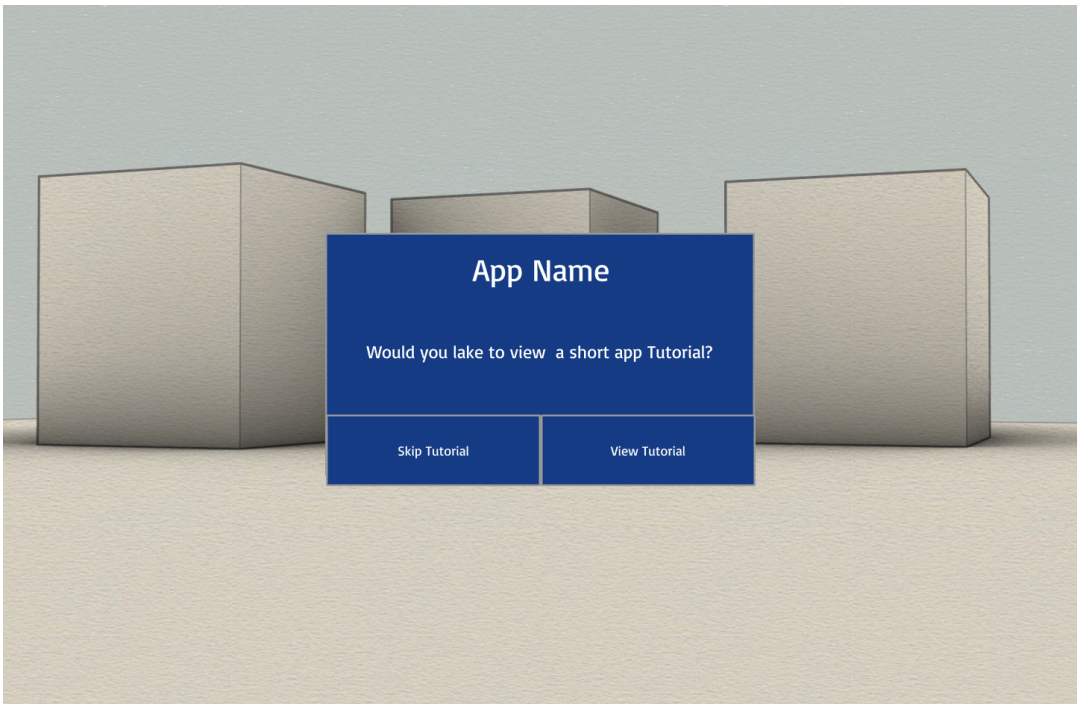


FIGURE 4.4: View Tutorial Dialog.

In fig. 4.11 the *Creating Waiting Room Interface* is presented. This interface consists of text prompting the user to enter the room name, an input field for the name of the waiting room and a 'create' button.

In fig. 4.12 the *Creating Waiting Room Screen* is presented.

In fig. 4.14 *Waiting Room Interface (non-expert)* is presented. This interface Displays

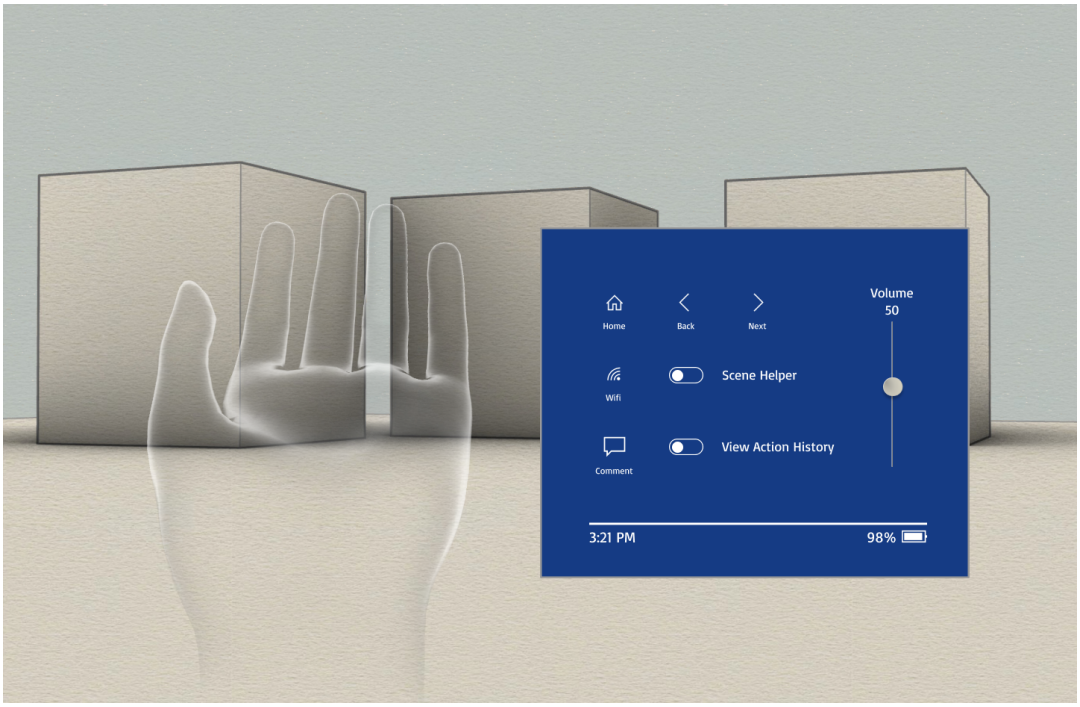


FIGURE 4.5: Hand Setting Menu Interface

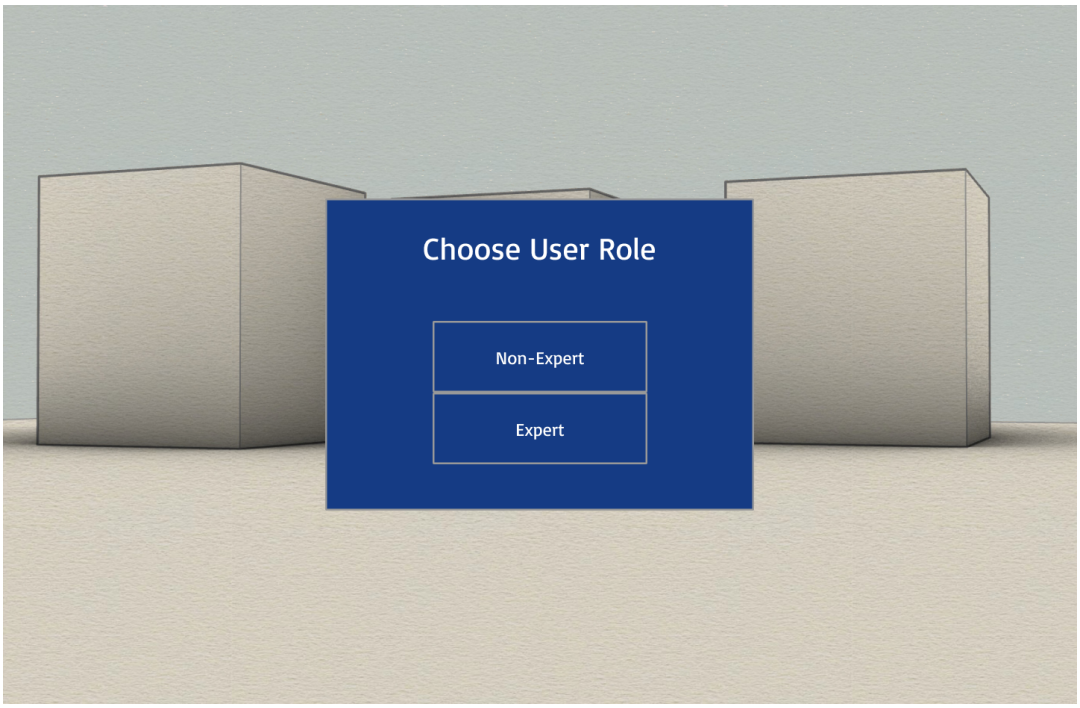


FIGURE 4.6: Choose User Role Dialog.

the room name in the top left. On the top right there is a 'Exit Waiting Room' button and a 'Follow me' button. In the main fields of the interface the users in the waiting room are displayed.

In fig. 4.15 *Waiting Room Interface (expert)* is presented. This interface is the same as the non-expert version with an extra 'Begin Session' button that is only visible to the owner of the room. Additionally, the owner of the room can remove any user from

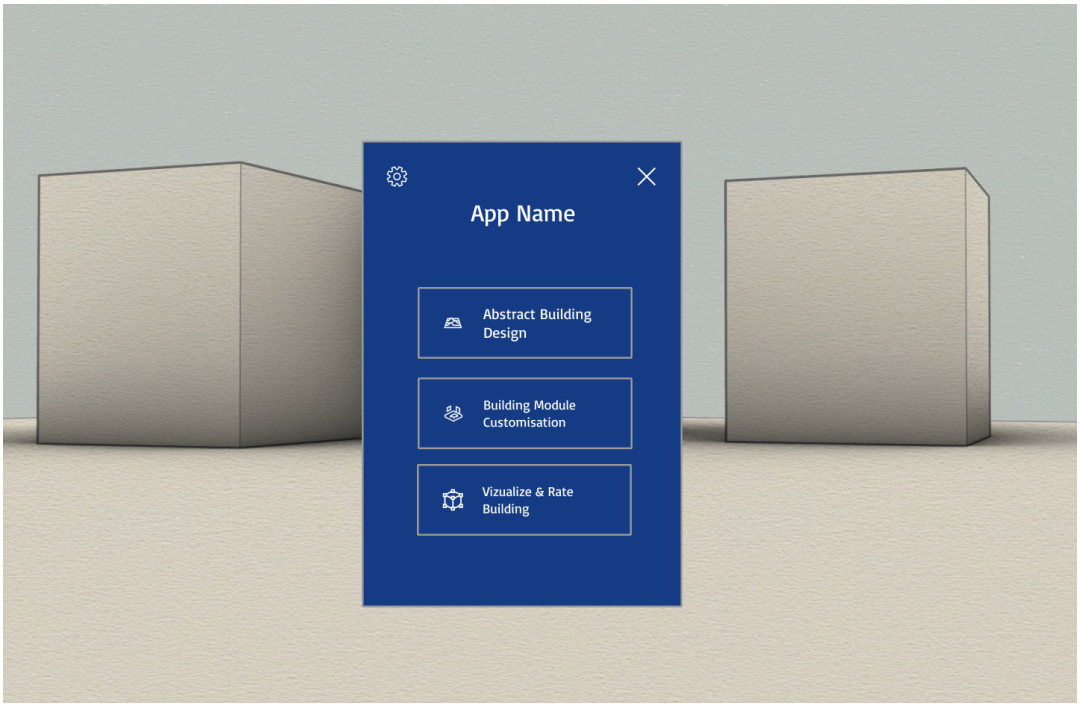


FIGURE 4.7: Application Main Menu Interface.

the room by pressing their user button.

In fig. 4.16 the *Finalizing Waiting Room Interface* is presented.

FIGURE 4.8: Create/ Join Waiting Room Interface (Non-Expert User)

FIGURE 4.9: Create/ Join Waiting Room Interface (Expert User)

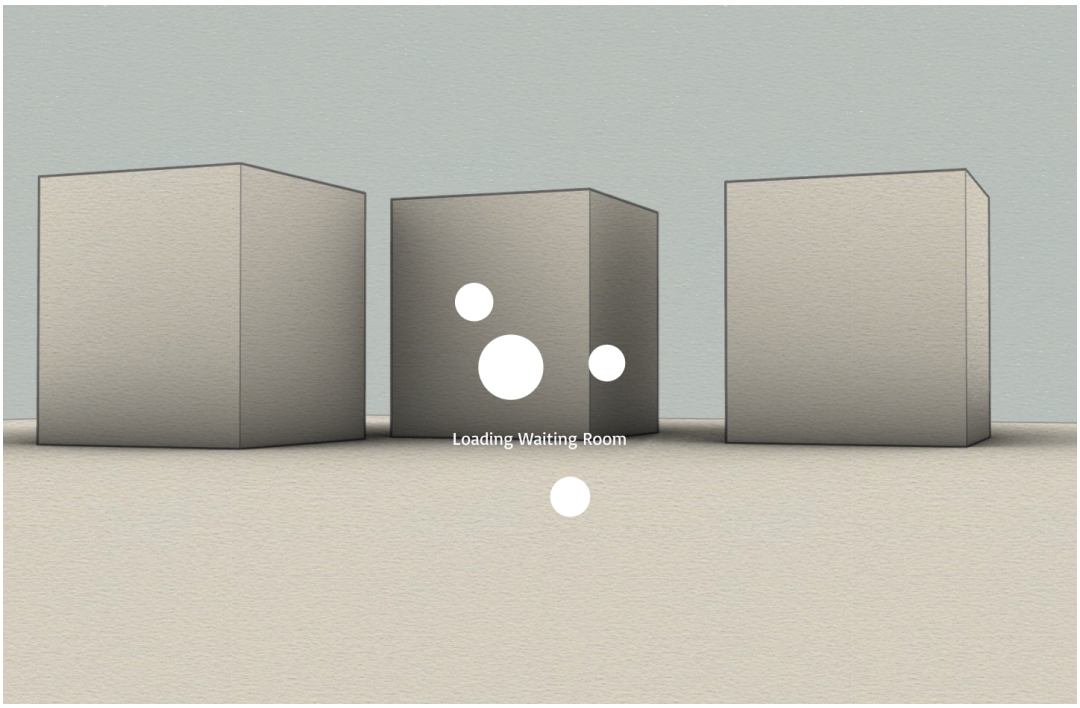


FIGURE 4.10: Joining Waiting Room Screen

4.4.3 Create & Fill Abstract Building Shape Prototyping

In fig. 4.17 the *Create New Abstract Building Interface* is displayed. It has a 'New Building' and a 'Next phase Button'.

In fig. 4.18 the *Building Placement Interface* is displayed. It has a single 'Place' button.

In fig. 4.19 the *Building Menu Interface* is displayed. It has a 'Hide' button that hides

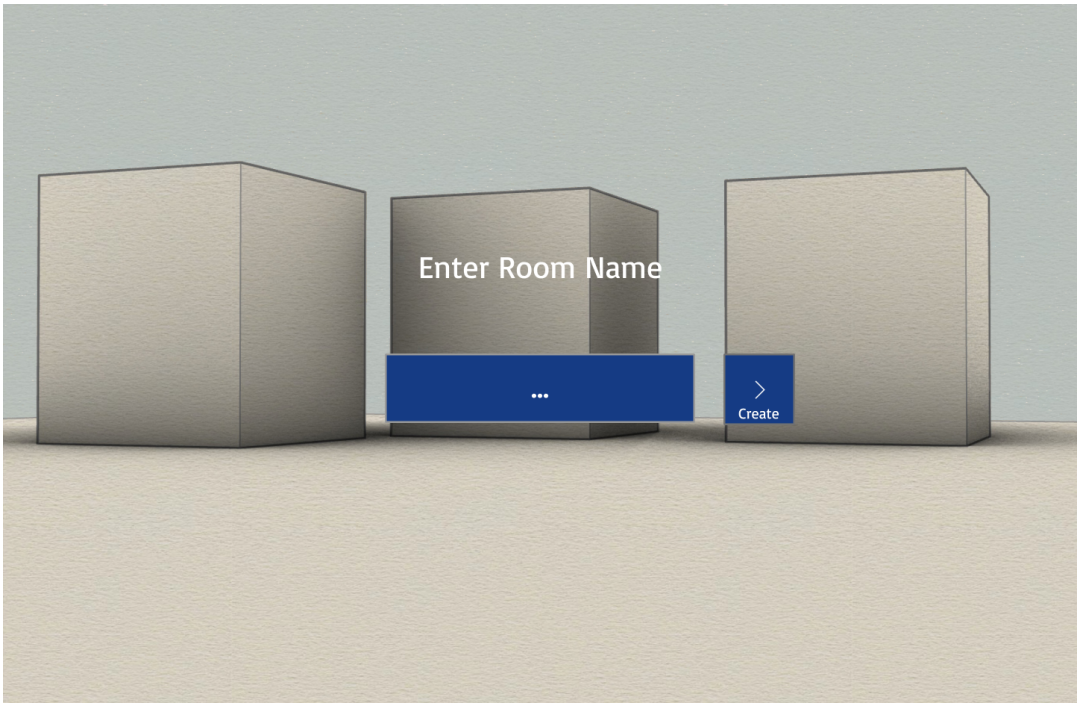


FIGURE 4.11: Creating Waiting Room Interface

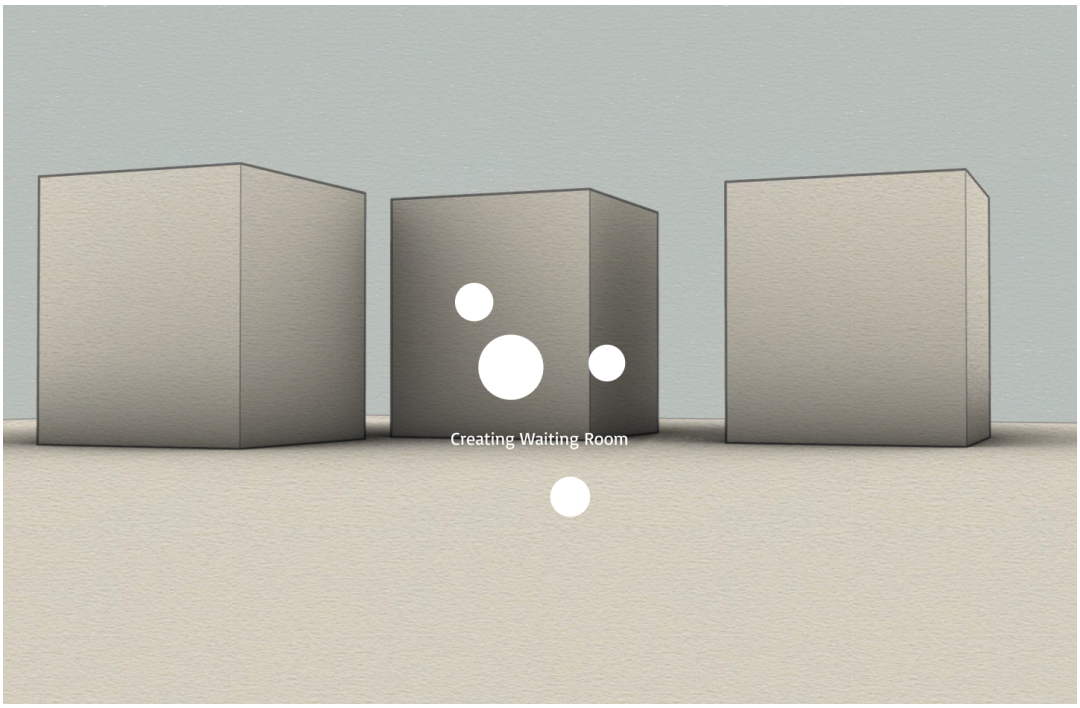


FIGURE 4.12: Creating Waiting Room Screen

the building menu, a 'Resize' Button to resize the building model, a 'Block Space' button to establish the building common areas, and a 'Remove' button to delete the building.

In fig. 4.20 the *Module Placement Interface* is displayed while the building filling is incomplete. The interface consists of a menu containing the different dwelling modules. Users can use the modules to fill the abstract building shape.

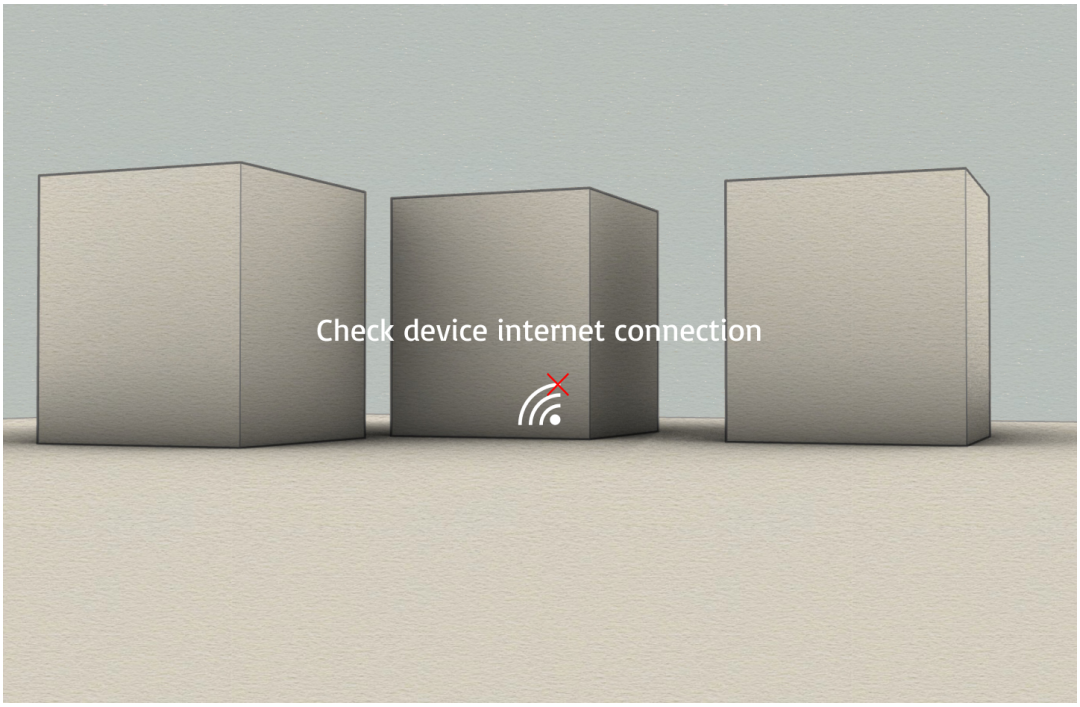


FIGURE 4.13: No Wi-Fi Connecetion Message Screen

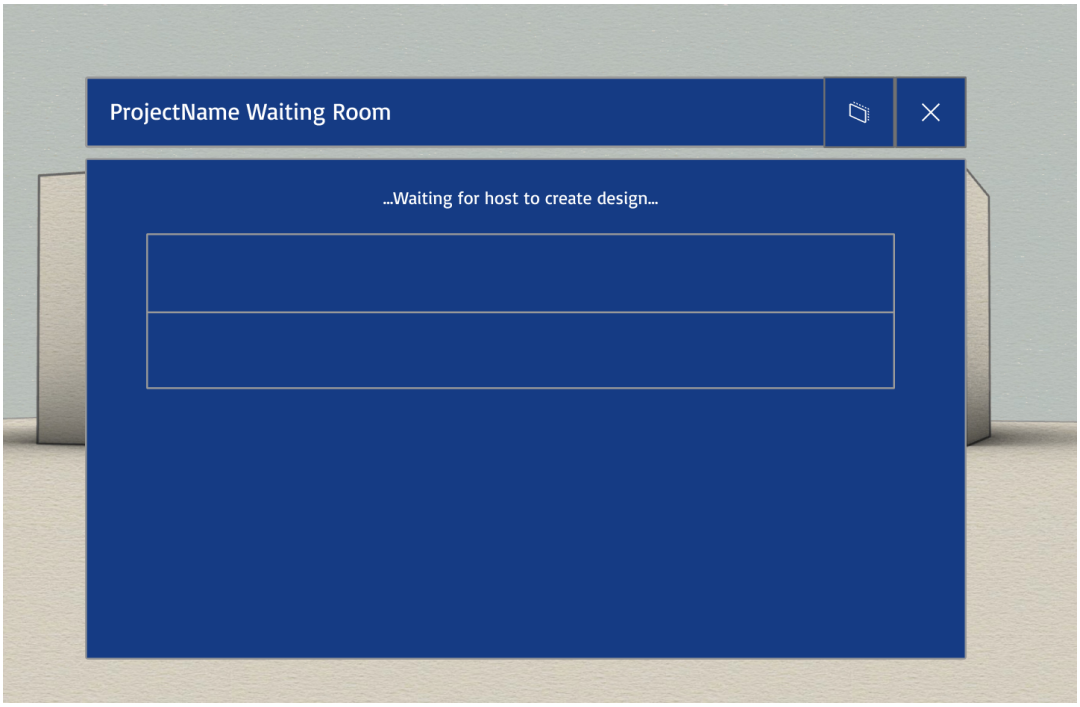


FIGURE 4.14: Waiting Room Interface (non-expert)

In fig. 4.21 the *Module Placement Interface* is displayed when the building filling is complete. The interface consists of a menu containing the different dwelling modules and a 'Finalize Design' button.

In fig. 4.22 the *Building Filling Complete Dialog* is displayed. This dialog displays the text 'Are you sure you want to finish the abstract design process' and a 'Yes' button and a 'No' button.

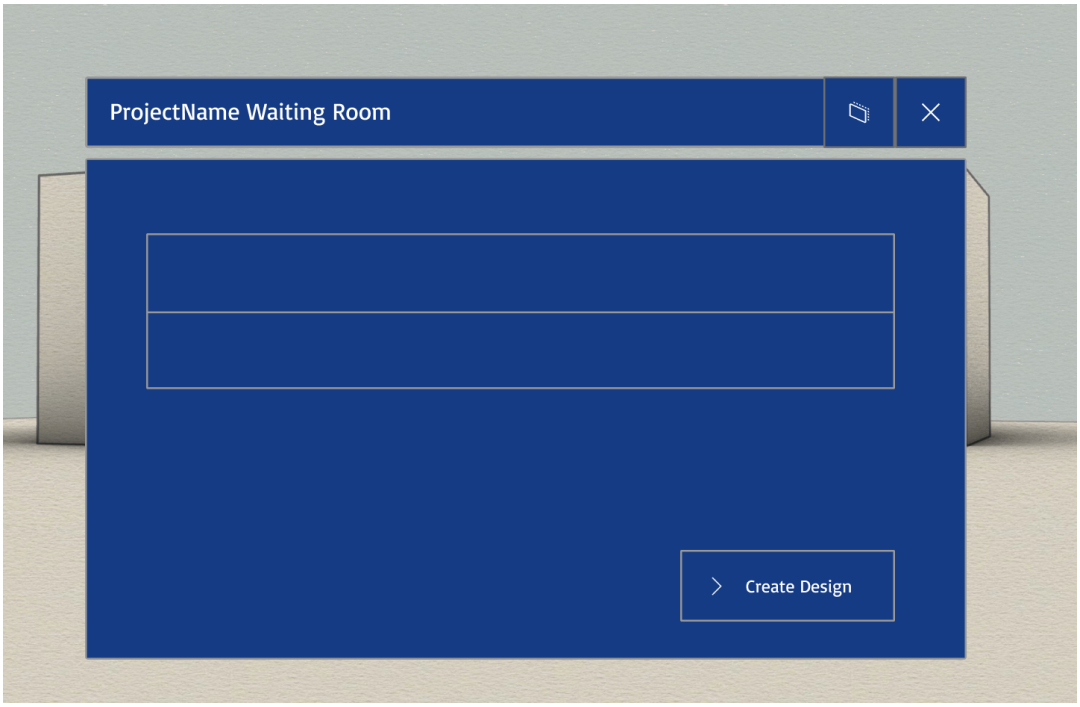


FIGURE 4.15: Waiting Room Interface (expert)

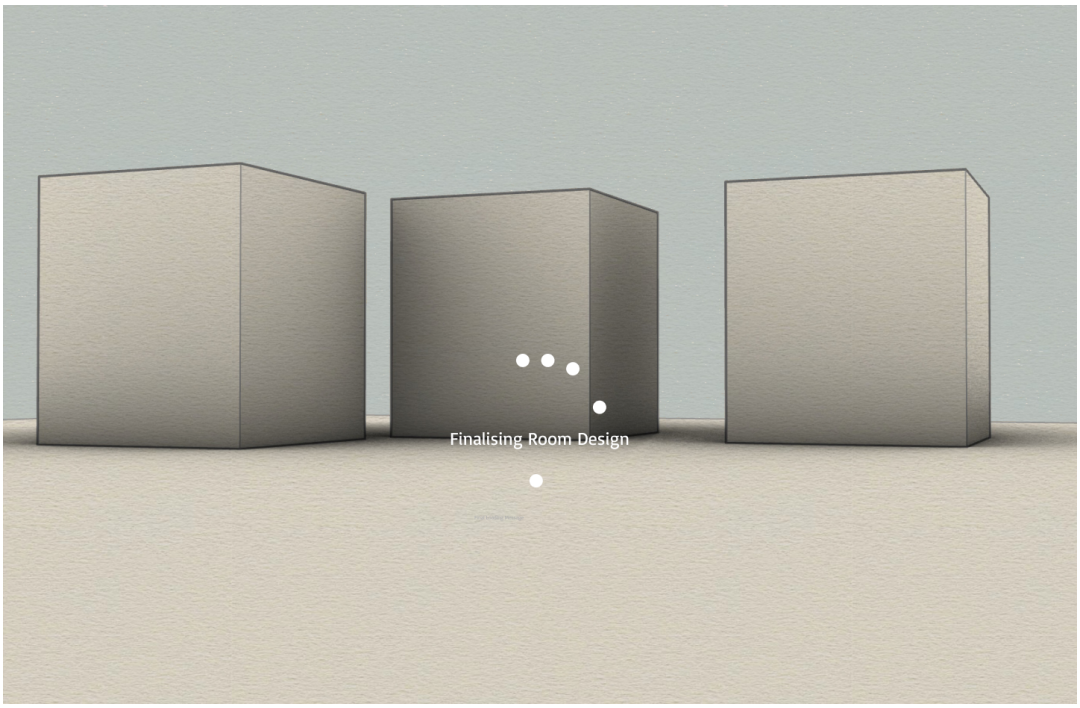


FIGURE 4.16: Finalizing Waiting Room Interface

In fig. 4.23 the saving screen of the abstract building design process is displayed.

4.4.4 Customize Dwelling Selection Prototyping

In fig. 4.24 the *Choose Design Menu* is presented. This menu is presented after the user has pressed the 'Customize Dwelling' button on the main menu. This menu contains all designed buildings ready for customization. Each building is represented by a

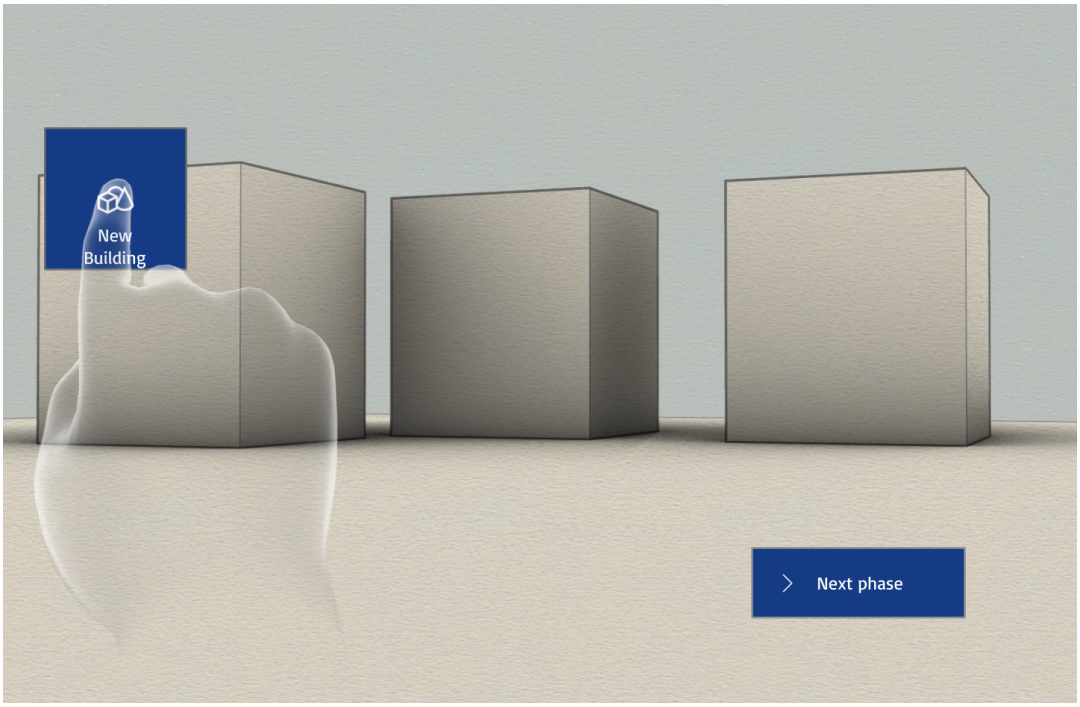


FIGURE 4.17: Create New Abstract Building Interface

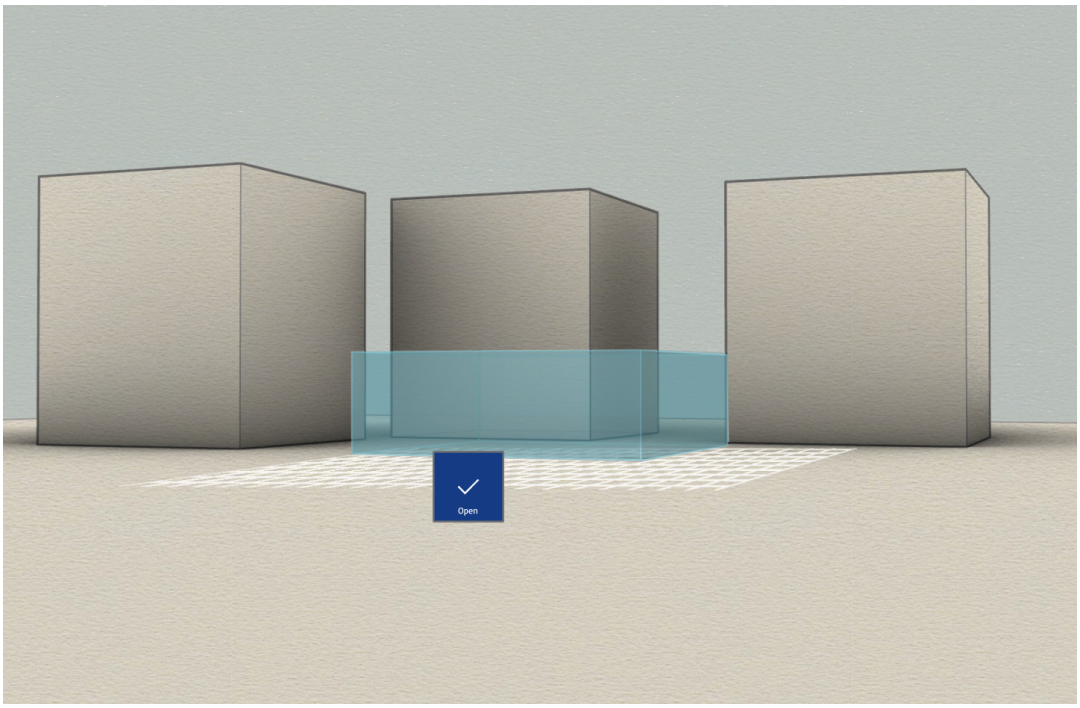


FIGURE 4.18: Place Abstract Building Interface

button on this menu. Additionally, this menu displays a text prompting the user to choose the building they would like to customize and a 'Home' button on its top right.

In fig. 4.25 the *Chosen Design Dialog* is presented. This Dialog is presented to make sure the user has chosen the correct building. It shows the selected building and has a 'Cancel' and 'Customize Building' button options on the bottom.

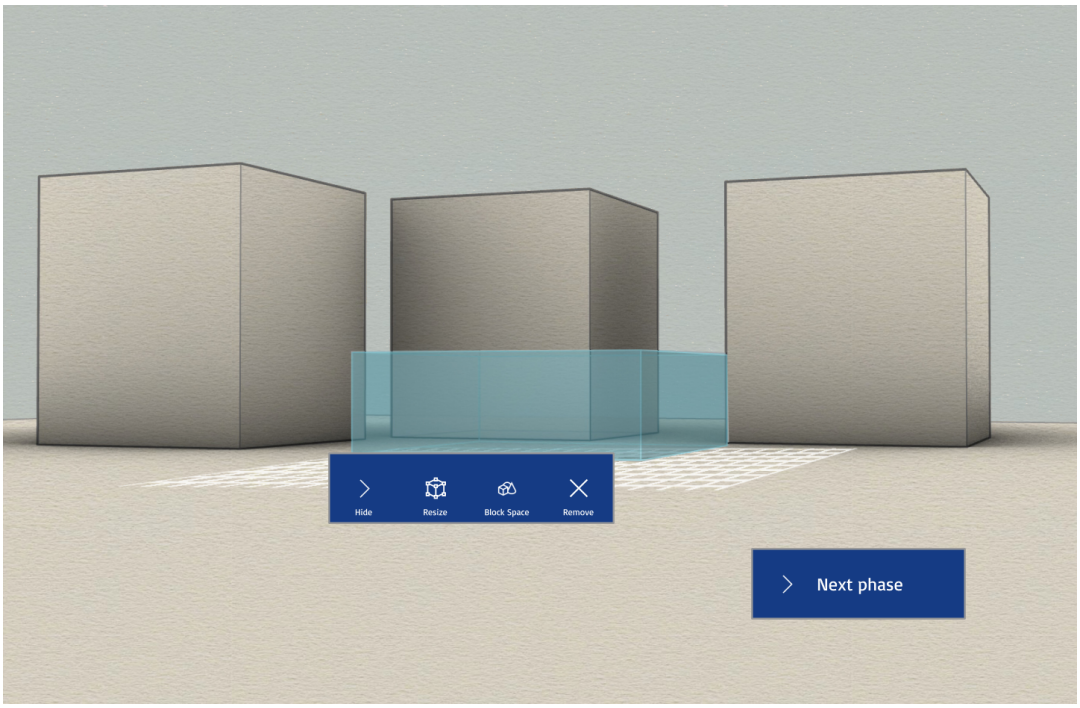


FIGURE 4.19: Building Menu Interface

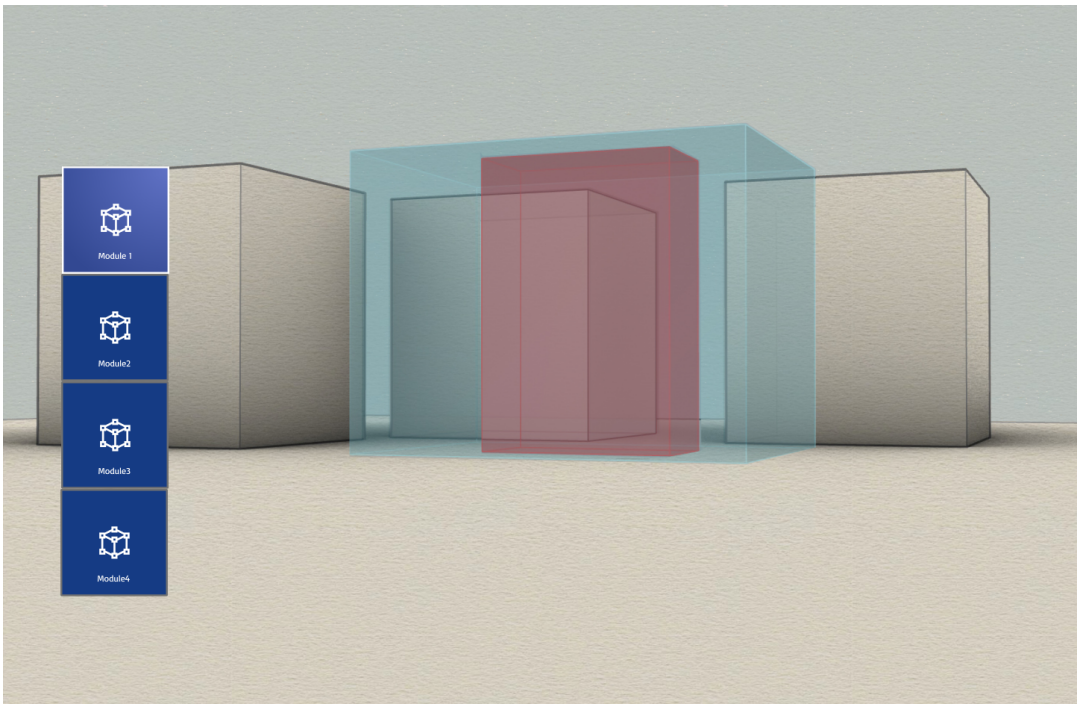


FIGURE 4.20: Module Placement Interface (Filling Non complete)

In fig. 4.26 the *Choose Design Dwelling Interface* is presented. This interface contains the selected building. Each dwelling acts as a separate button.

In fig. 4.27 the *Chosen Dwelling Dialog* is presented. This Dialog is presented to make sure the user has chosen the correct dwelling. It shows the selected Dwelling and has a 'Cancel' and 'Customize Dwelling' button options on the bottom.

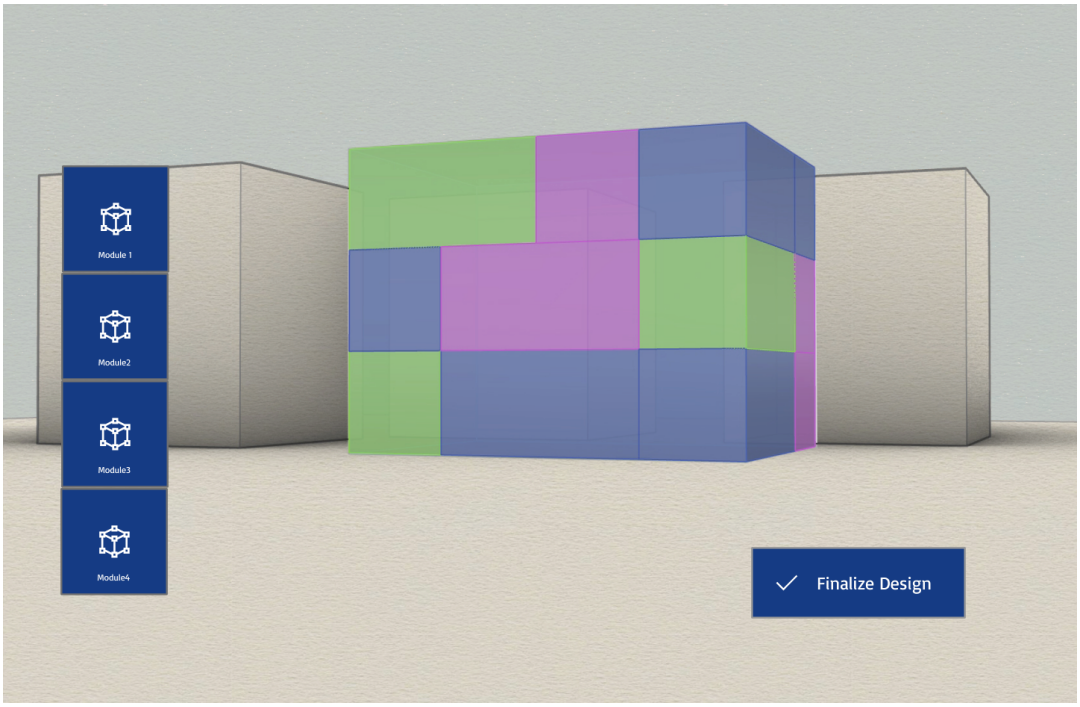


FIGURE 4.21: Module Placement Interface (Filling Complete)

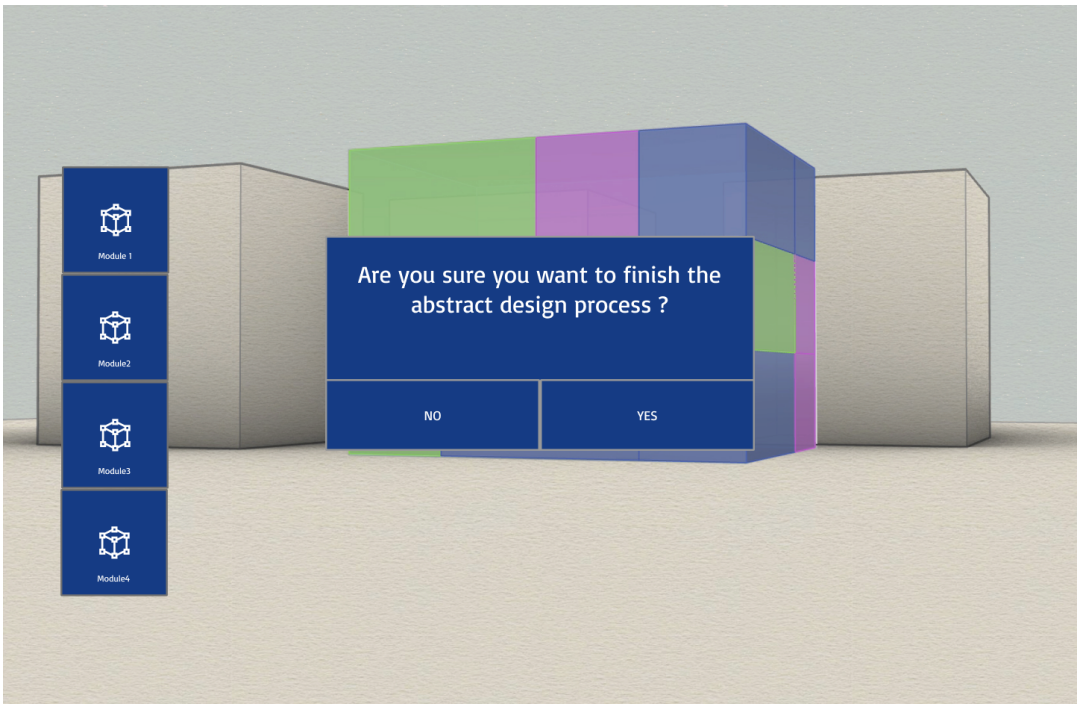


FIGURE 4.22: Building Filling Complete Dialog

In fig. 4.28 the *Customize Module Menu* is presented. This Menu has a 'Customize Facade' and 'Customize Layout' button options on its center and a 'Cancel' button on its top right. Here the user can choose which customization he wants to inflict on the dwelling chosen.

In fig. 4.29 the *Customize Dwelling Facade Waiting Room Interface* is presented. This interface Displays the room name in the top left. On the top right there is a 'Exit

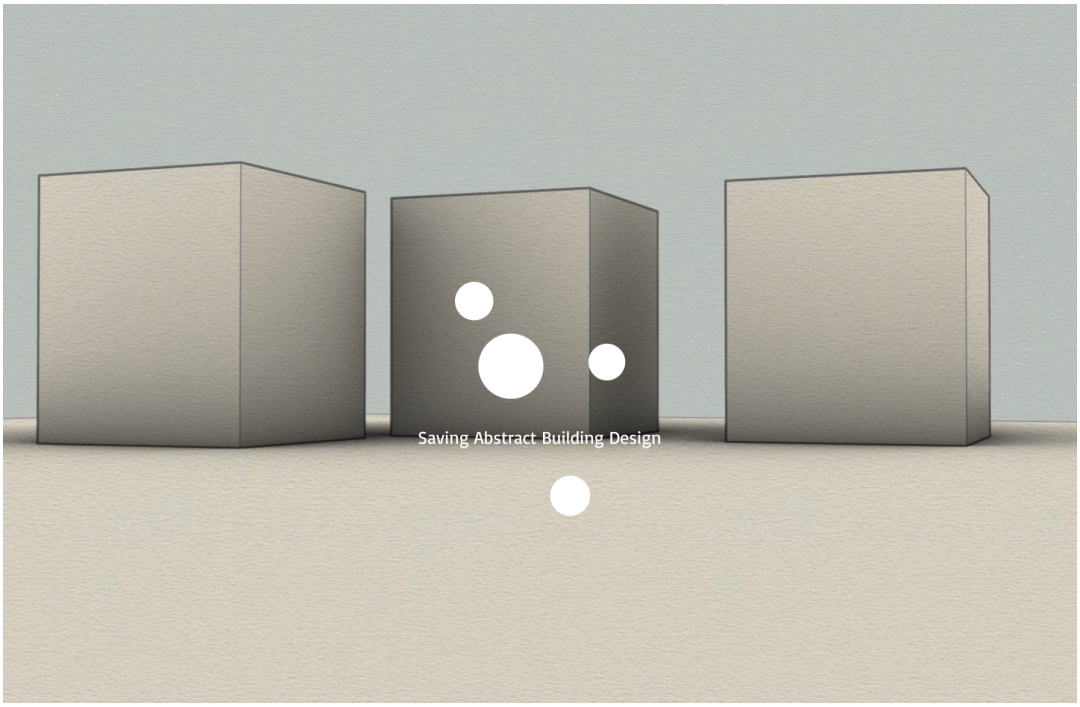


FIGURE 4.23: Saving Abstract Building Design Screen

Waiting Room' button and a 'Follow me' button. In the main fields of the interface the users in the waiting room are displayed. A 'Customize Design' button is placed, at the bottom right corner to begin the session. Additionally, the owner of the room can remove any user from the room by pressing their user button.

In fig. 4.30 *Customize Dwelling Layout Waiting Room Interface* is presented. It is similar to the waiting room interface described above.

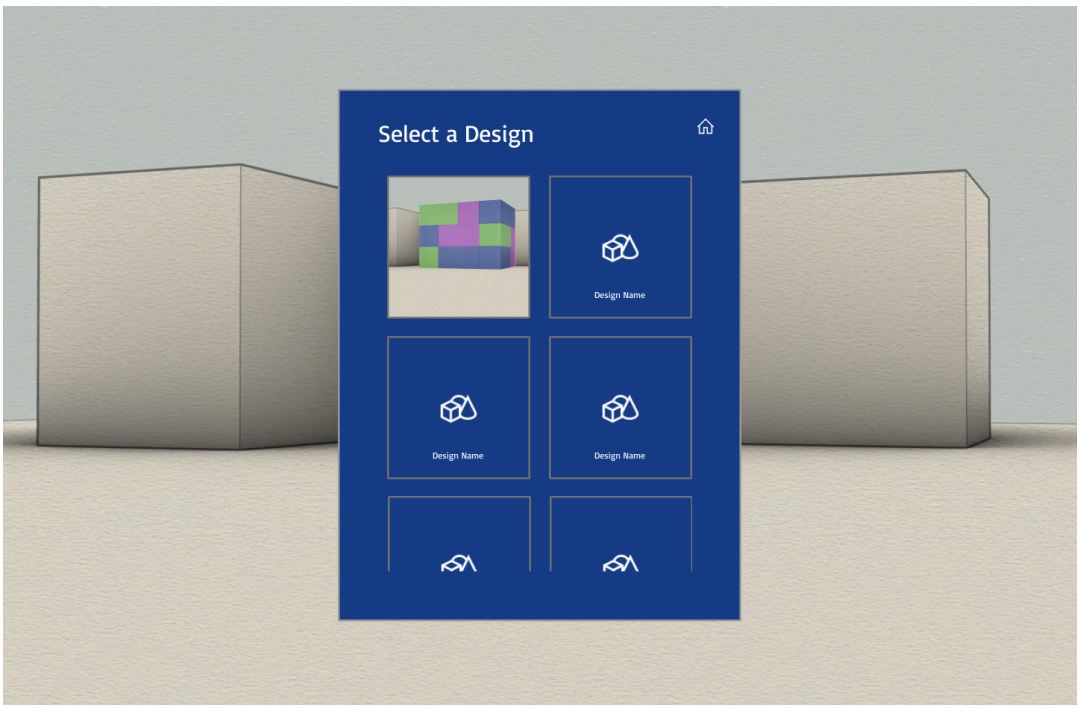


FIGURE 4.24: Choose Design Menu Interface

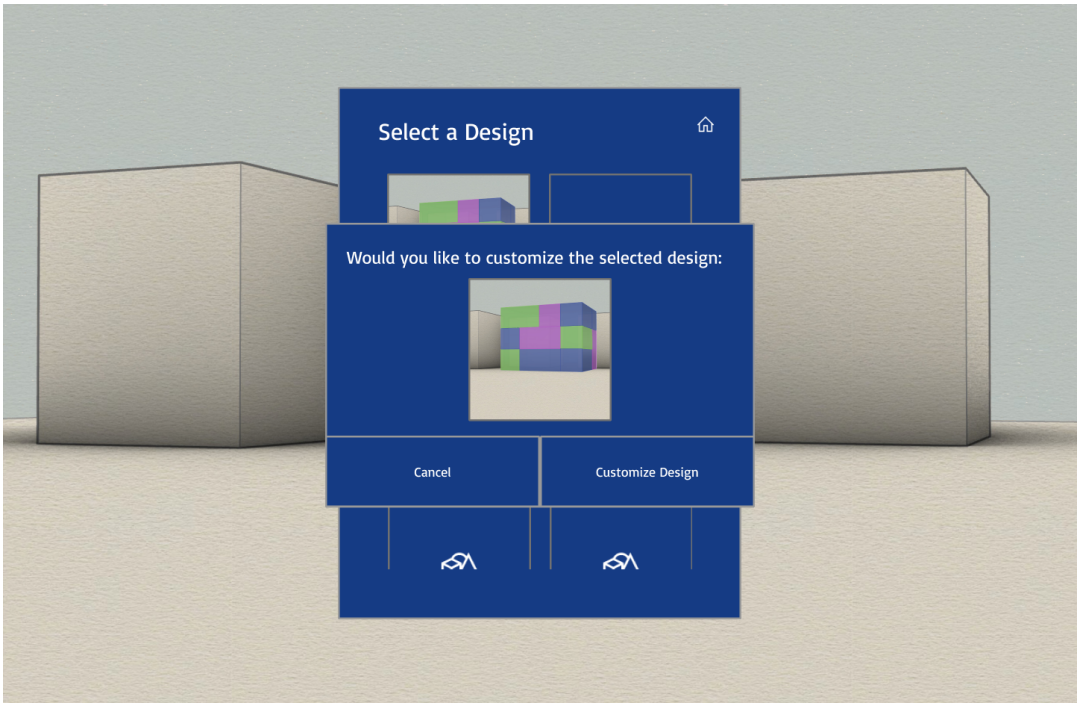


FIGURE 4.25: Chosen Design Dialog

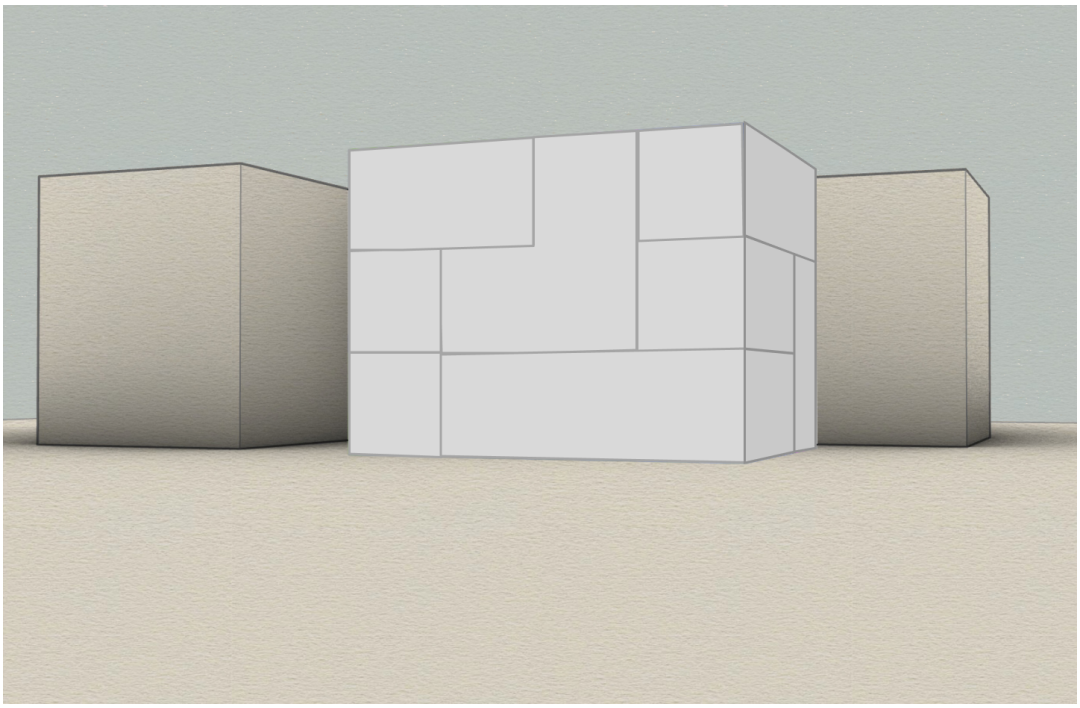


FIGURE 4.26: Choose Design Dwelling Interface

4.4.5 Customize Dwelling Layout Prototyping

In fig. 4.31, fig. 4.32, fig. 4.33 three instances of *the Dwelling Layout Customization Interface* are presented. In fig. 4.31 we see the main menu of the dwelling customization interface, that is comprised of as many buttons, as the different categories of spaces in a dwelling. For example there would be a 'Kitchen' button, a 'Bathroom' Button etc. Pressing each of these buttons results in the expansion of its respective menu.

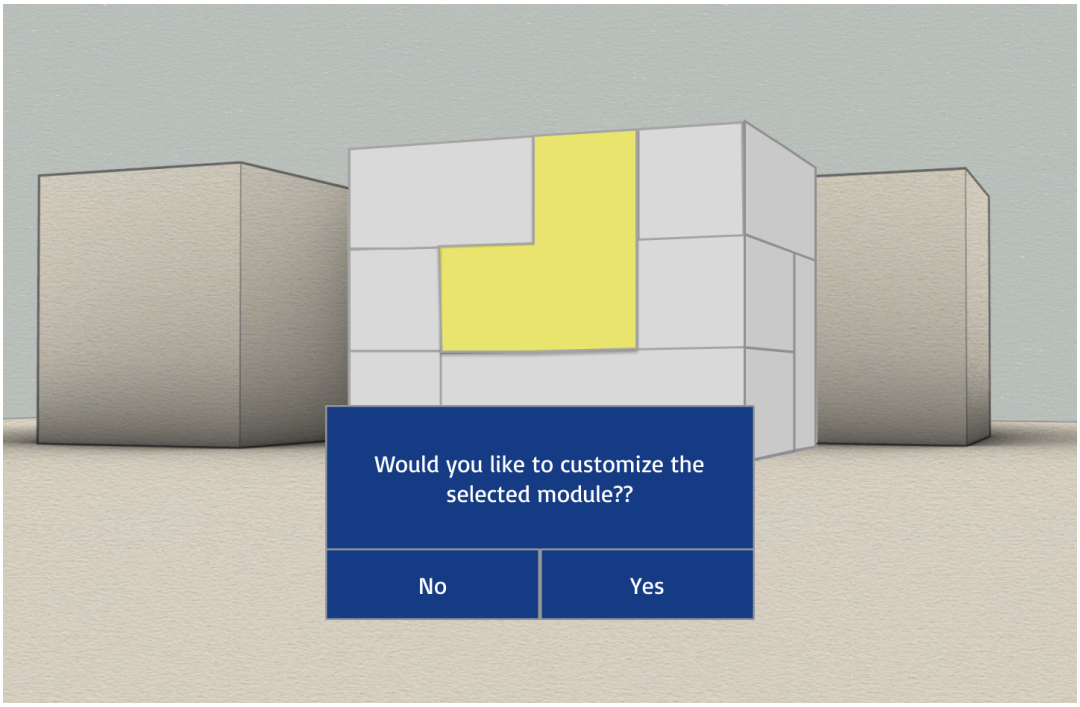


FIGURE 4.27: Chosen Design Dwelling Dialog

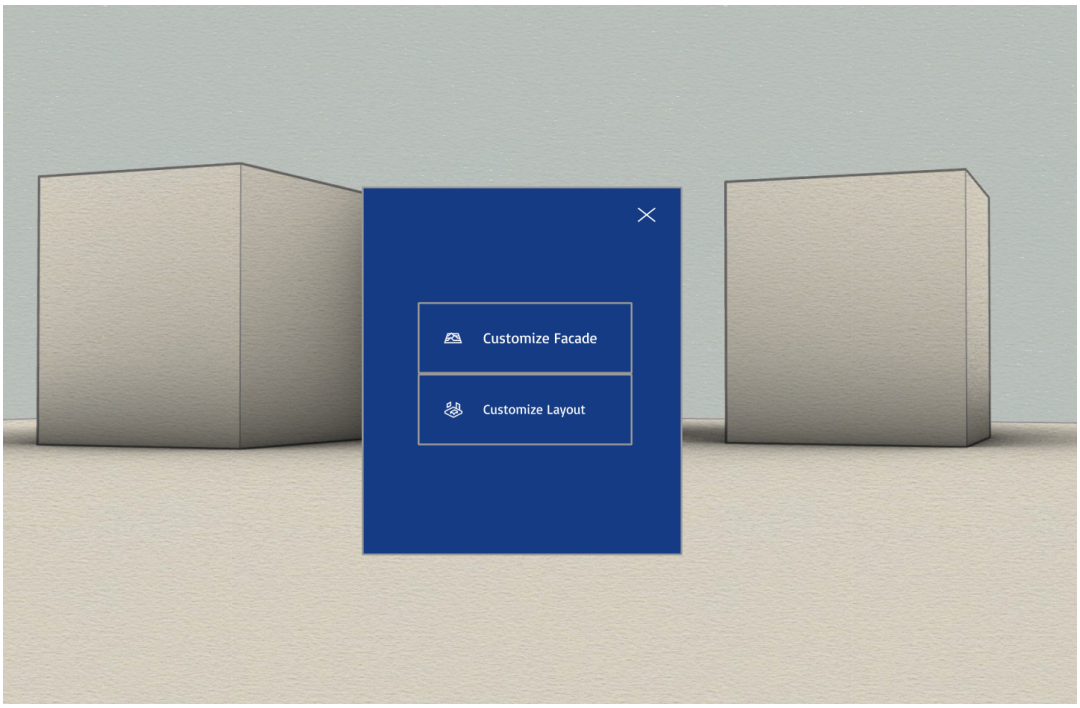


FIGURE 4.28: Customize Module Menu

In fig. 4.32, fig. 4.33, we see two different instances of the interface with the Bathroom or Bedroom sub menus expanded. These menus contain buttons for all the different variations of bathrooms or bedrooms available in the app.

In fig. 4.34 we see how the interface changes when a room is dragged to be placed in the Layout.

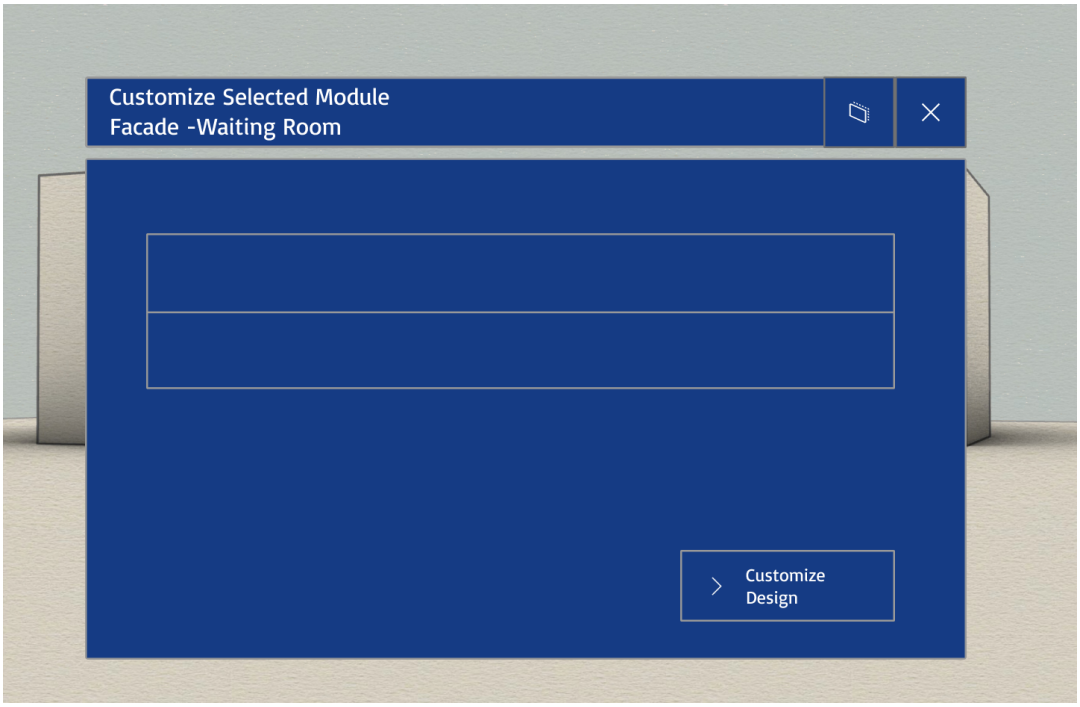


FIGURE 4.29: Customize Dwelling Facade Waiting Room Interface

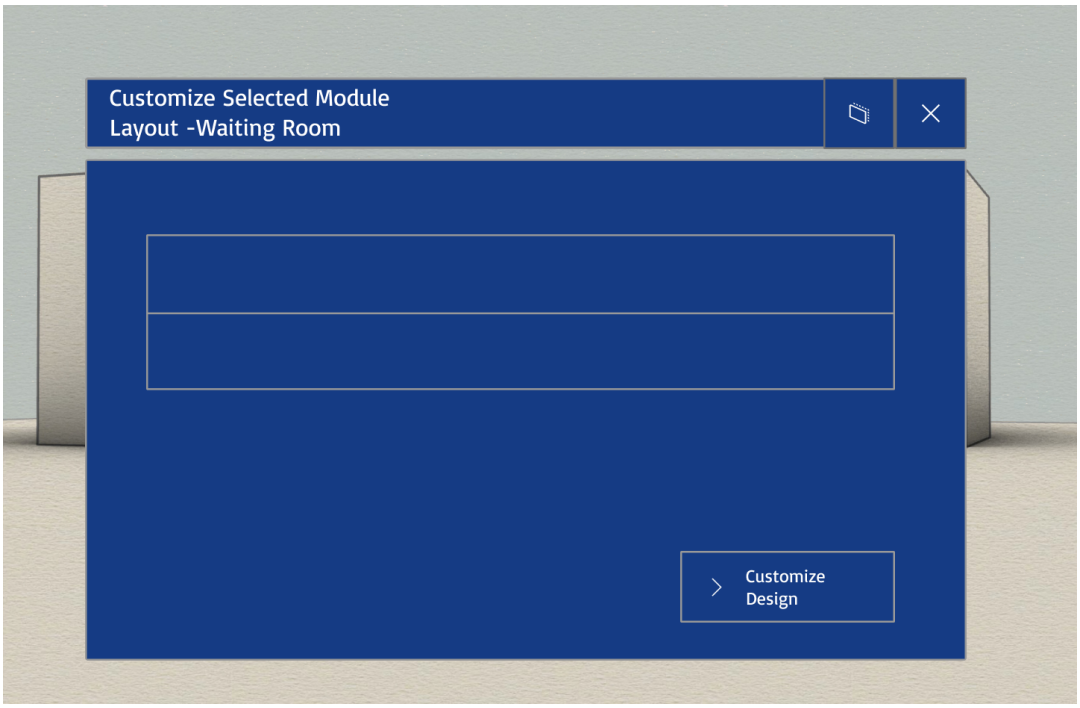


FIGURE 4.30: Customize Dwelling Layout Waiting Room Interface

In fig. 4.35 we see the *Interface Wrong Placement Dialog*. It consists of a text informing the user of the wrong placement and asking them to replace the room and a 'Yes' and a 'No' Button.

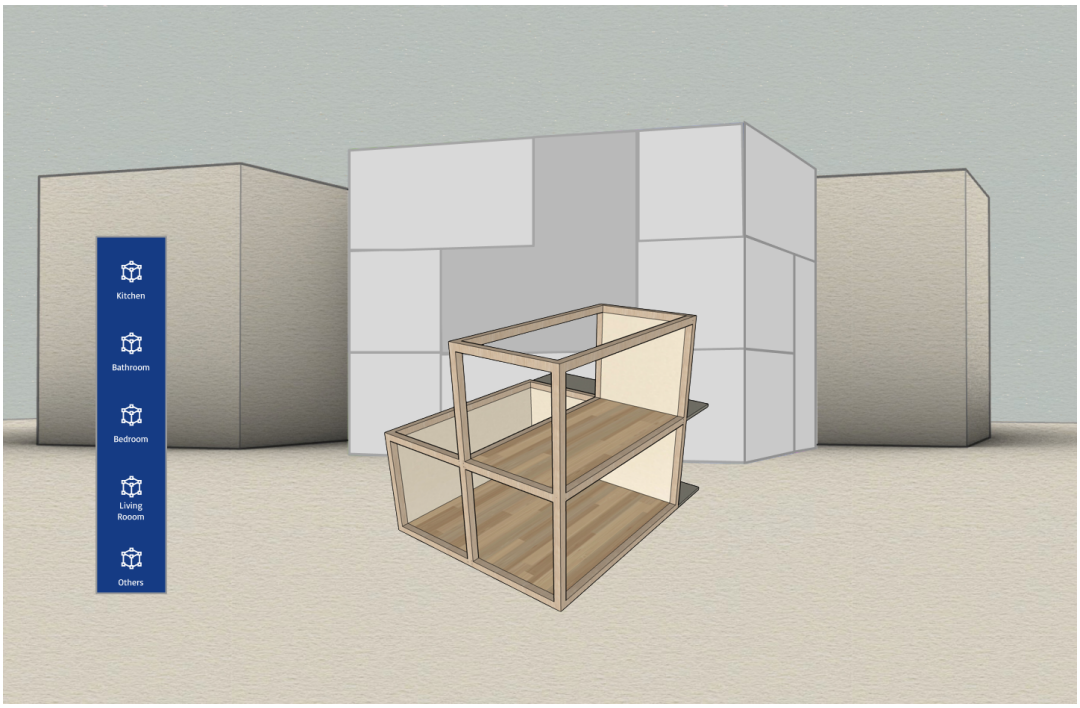


FIGURE 4.31: Layout Customization Interface - Main Menu.

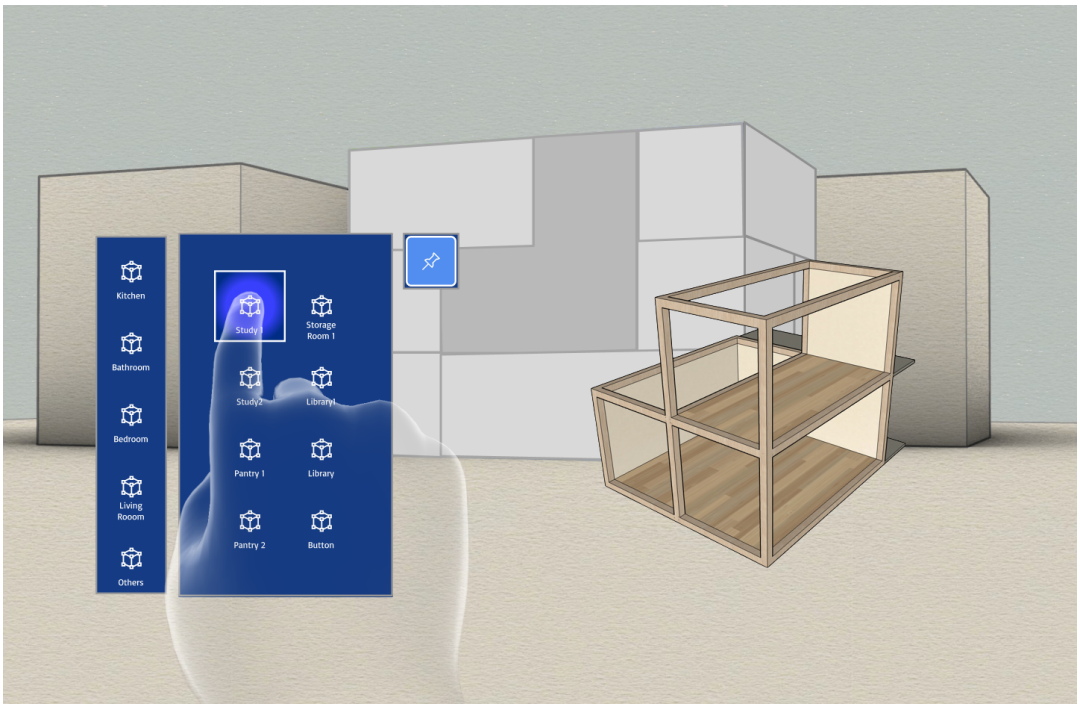


FIGURE 4.32: Layout Customization Interface - Sub Menu 1.

4.4.6 Customize Dwelling Facade Prototyping

The customize dwelling facade prototyping follows the exact same logic as the customize facade prototyping. The only difference being that the model of the interface will only include the facade of the building and the submenus will contain different variations of different facade elements (such as balconies windows etc.).

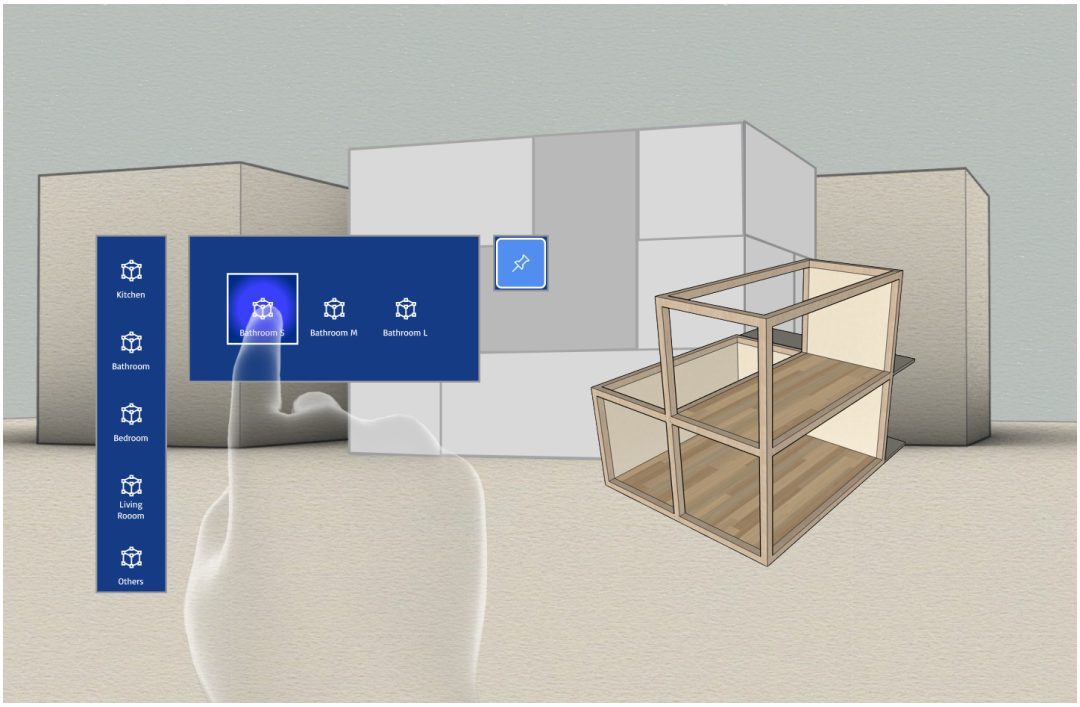


FIGURE 4.33: Layout Customization Interface - Sub Menu 2.

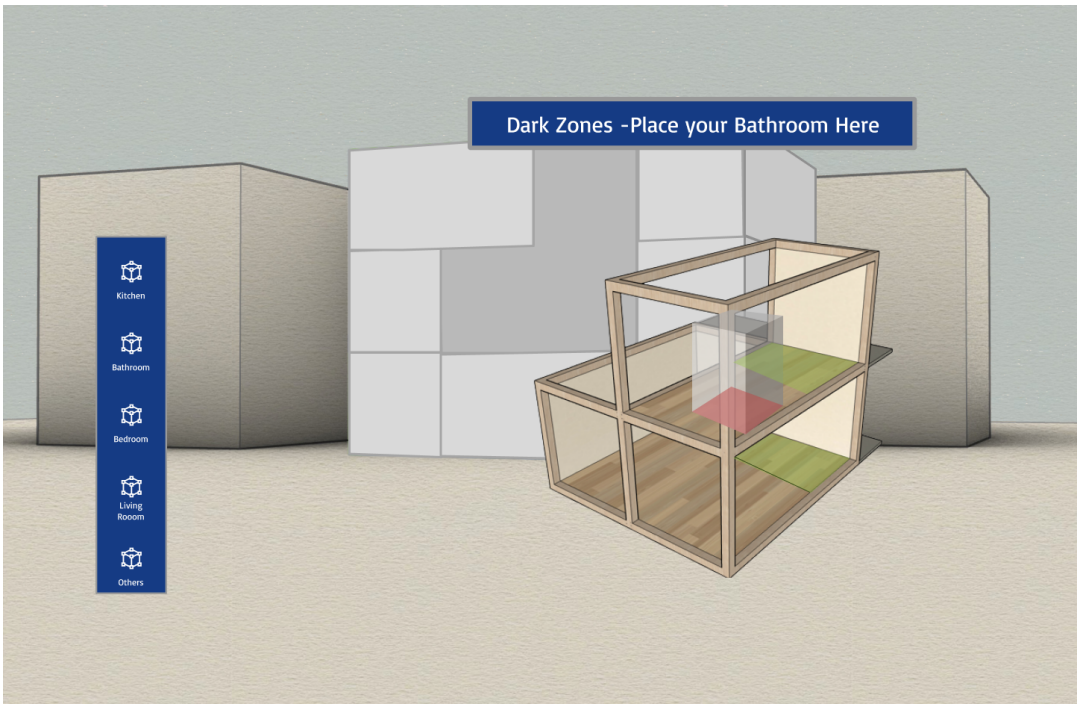


FIGURE 4.34: Layout Customization Placement Interface.

In fig. 4.36 we see the app *Quitting Dialog*. The interface has two buttons 'yes' and 'no'. This is to ensure the user has not pressed the 'Quit App Button' by mistake.

In fig. 4.37 we see the app *Quitting Screen*.

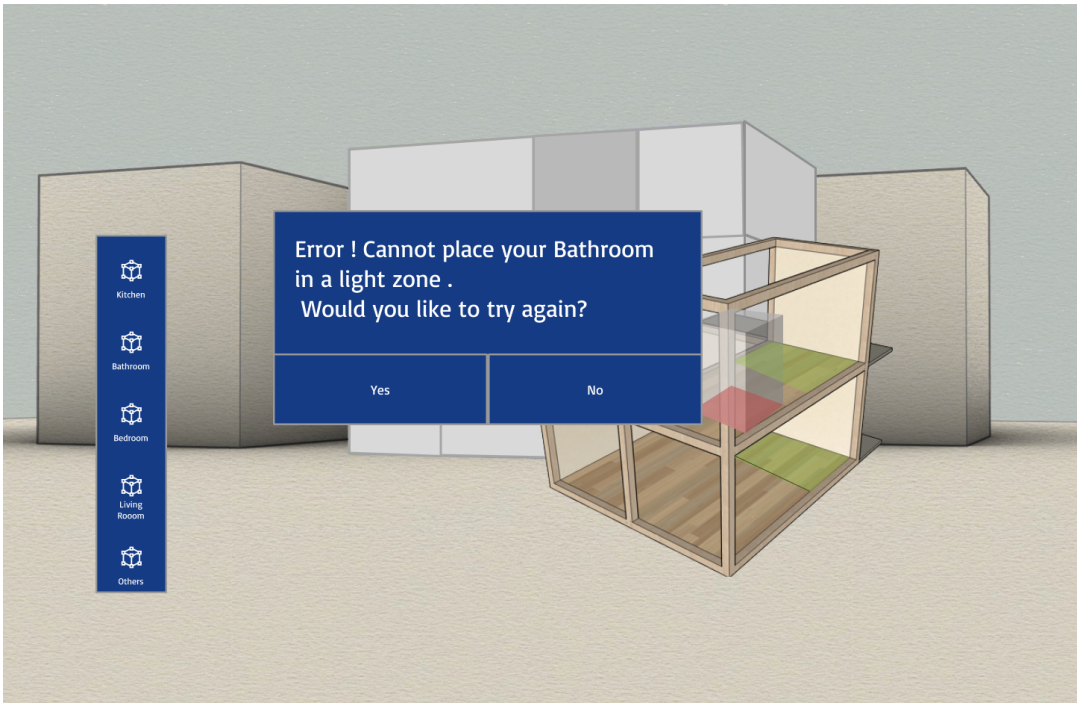


FIGURE 4.35: Layout Customization Wrong Placement Interface.

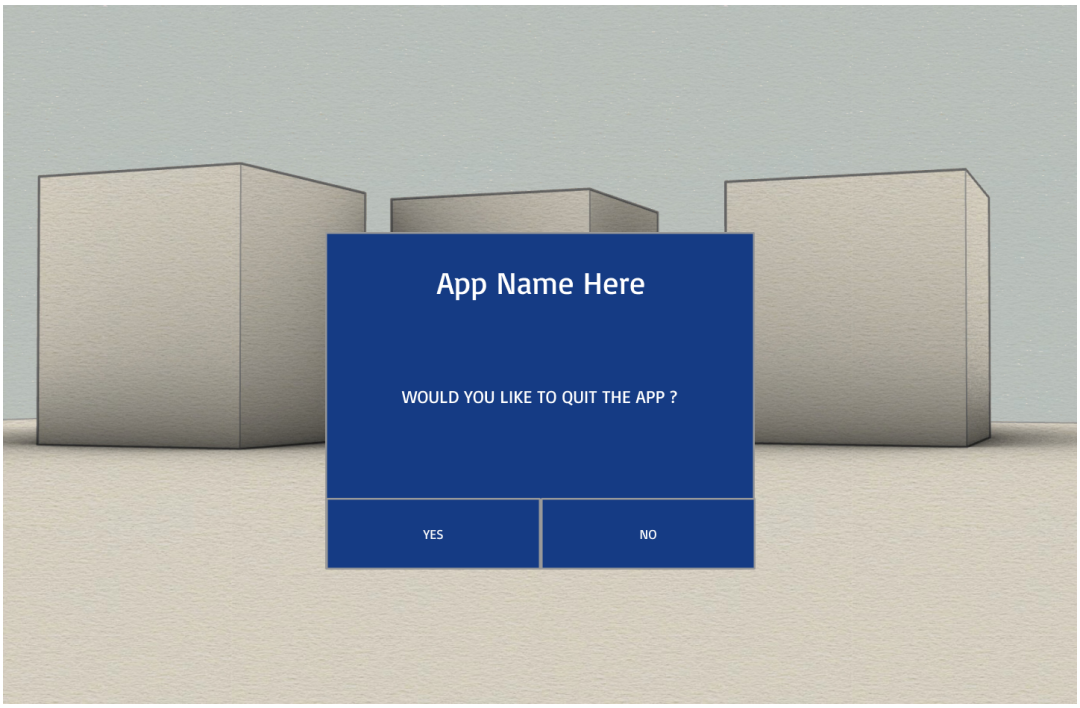


FIGURE 4.36: Quitting App Dialog.

4.5 Storyboarding

The success of any application heavily relies on its user experience (UX), as mentioned above. Storyboarding, a fundamental tool in the UX design toolbox, plays a vital role in achieving this goal. By enabling the visualization of user journeys within an app, it fosters a collaborative and effective design process, ensuring that the final product meets the needs and expectations of its users. In this subsection we

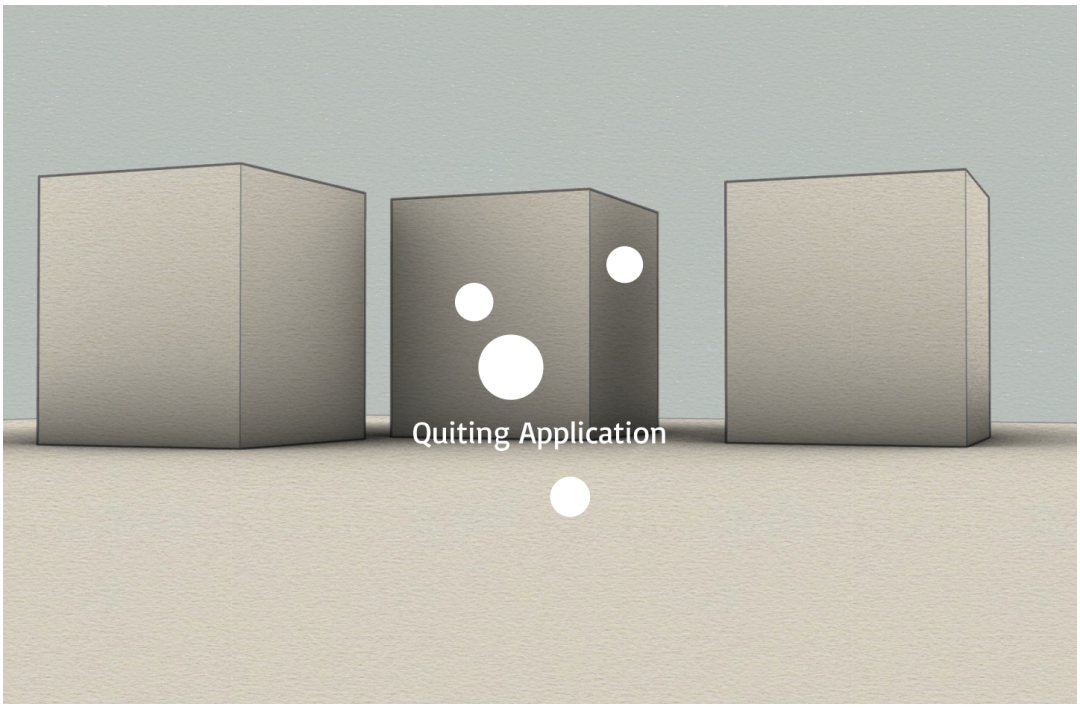


FIGURE 4.37: Quitting App Screen.

will view the storyboard of the main use cases of the AR-Apt app as created in the Figma app.

In fig. 4.38 the storyboard is displayed, of an expert user from the start of the application, until they create or join an abstract design session.

In fig. 4.39 the storyboard is displayed, of a non-expert user from the start of the application, until they create or join an abstract design session.

In fig. 4.40 the storyboard of the tutorial of the application is displayed, from its start until the user completes it.

In fig. 4.41 six flows of the hand menu application storyboard are displayed. Each flow showcases a different feature of the hand menu.

In fig. 4.42 the flow of the application is displayed, from the beginning of an abstract design session until the session is complete. This includes both abstract design use cases.

In fig. 4.43 the flow of the application from the main menu of the application is showcased, where the user chooses to customize a dwelling, until they have chosen both their desired dwelling and the type of customization they would like (facade, layout). This corresponds to the *Customize Dwelling Selection* use case.

In fig. 4.44 the flow of the application from the beginning until the end of the layout customization process, is displayed.

4.6 User Evaluation and Interface Adjustments

Evaluating the application prototype is essential for improving the user experience and aligning the design with user expectations. User testing sessions, where you

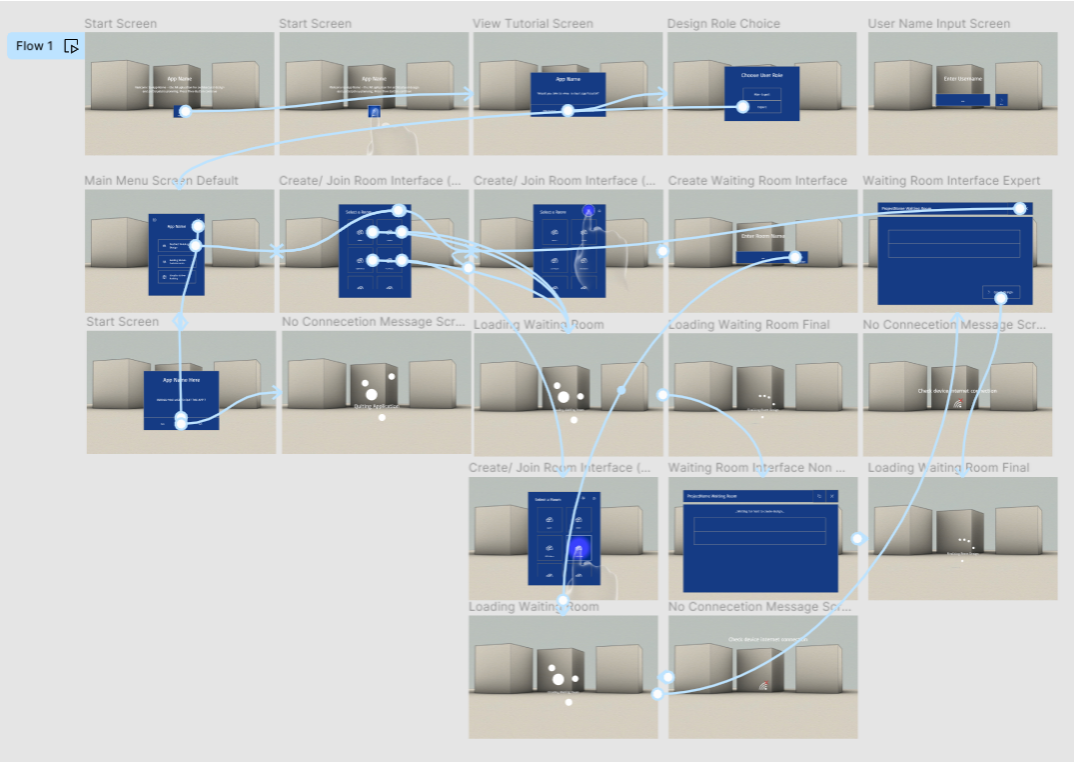


FIGURE 4.38: Main menu App Storyboard (Expert User)

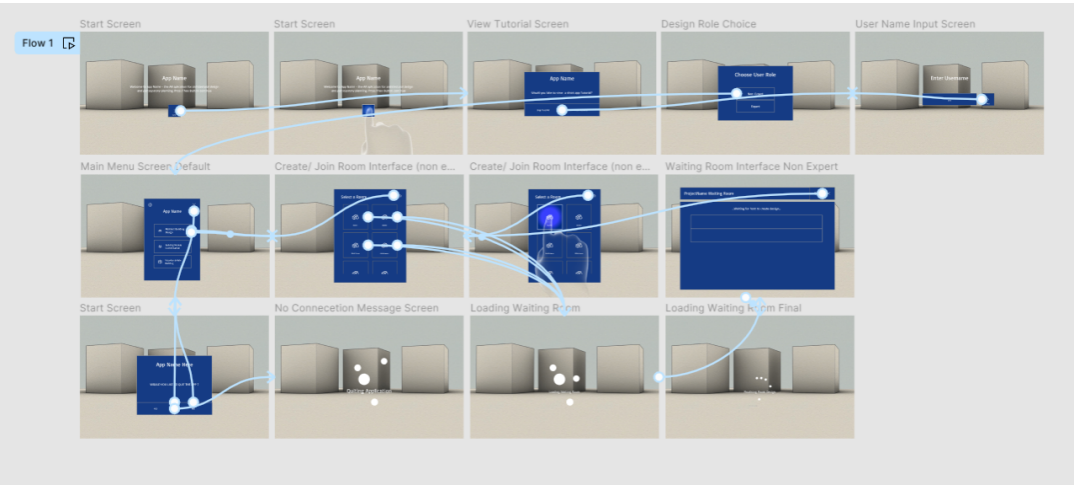


FIGURE 4.39: Main menu App Storyboard (Non-Expert User)

observe participants interacting with the prototype and gather their feedback, are a key part of this evaluation.

One powerful technique to employ during these sessions is the "think aloud" method. As users interact with the prototype, they are encouraged to verbalize their thoughts, feelings, and actions as they navigate the interface. This allows developers to gain valuable insight into potential usability issues, areas of confusion, or points of delight within the user experience.

By carefully analyzing the users' comments and reactions, you can identify specific areas for improvement, leading to more informed iterations and ultimately a better final product.

FIGURE 4.40: App Tutorial Storyboard.

In this case, three users with different areas of expertise (an ECE student, an architecture student and a simple user) participated in the evaluation of the AR-Apt. The process consisted of many iterations of evaluation, initially on the first paper prototype and later on the Figma prototyping and interfaces.

During this process most users could not figure out how to remove a participant from a waiting room. However they did not need the functionality in their single user sessions. A user suggested a '-' button be introduced, to better indicate the user removal option.

The biggest point of contention was the appearance of the customization interface

and specifically the dwelling model appearance and feedback. Most users felt unsure of the principles and rules of the room placement during the layout customization process. This resulted in a complete redesign of the model of the *Layout Customization Interface*.

The new interface will provide specific 'boxes' that the user can fill with different kind of rooms and is described bellow . This interface consists of a 'Next/Previous' Button to view all design layouts and later all room placements. It also contains a 'View Select' toggle and text, so users can navigate between their own private design and the shared design and know in which interface they currently are. It also contains a 'Compare View' toggle that places choices side by side so they can be compared, when turned on. The compare feature is available only on private interface to avoid user crowding. Finally it contains a 'Confirm Choice' button.

This way both the user is guided to create a more architecturally sound result and the overall app design process is vastly simplified.

In order to include all aspects of the application, we should note that the difficulty introduced by the fact that most users are unfamiliar with the technology on which the app will be deployed to, is not taken into consideration at this stage. However the tutorial is designed in such a way, to enable users to familiarize themselves with both the technology and the app interface simultaneously.

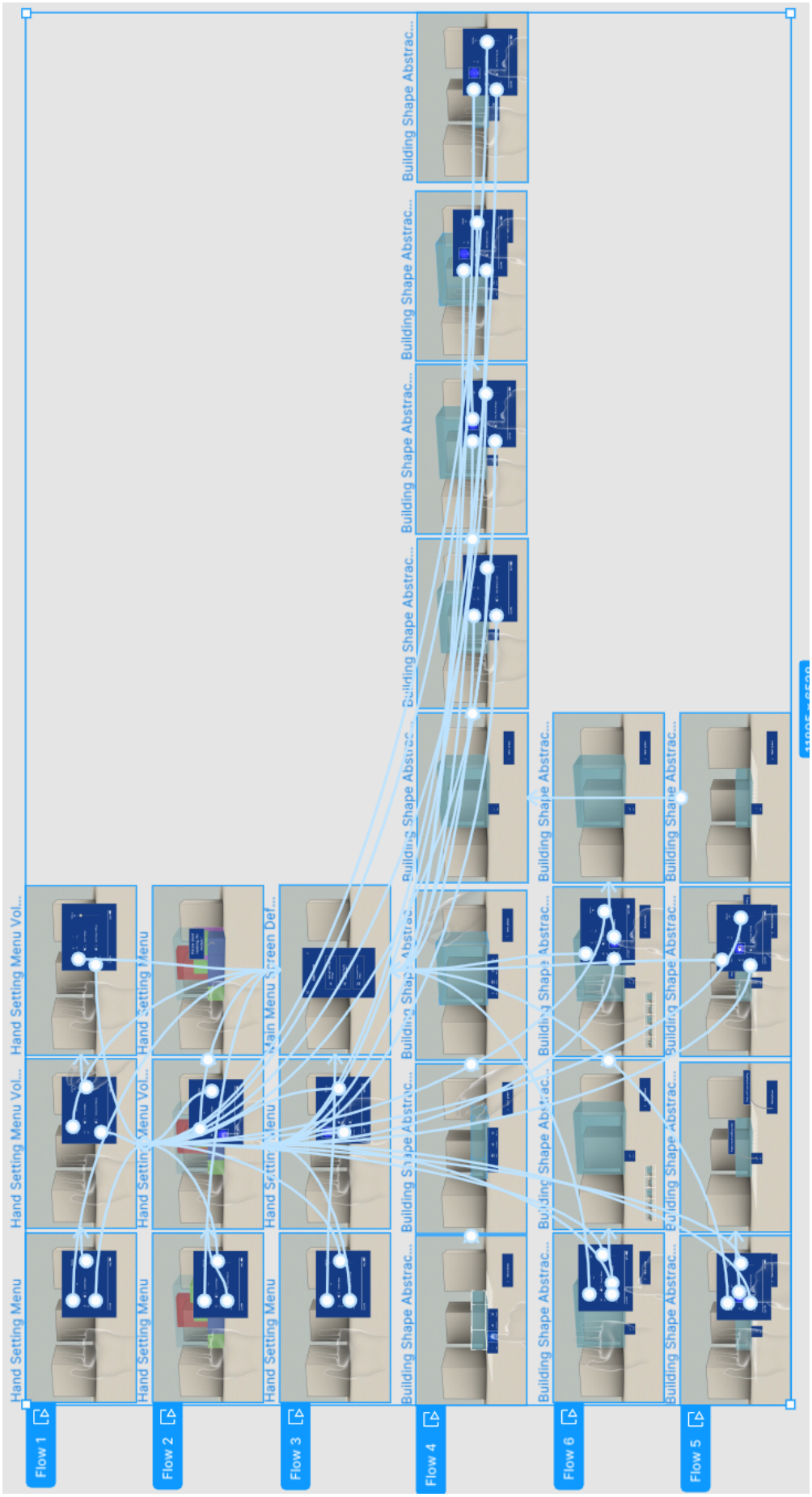


FIGURE 4.41: App Hand Menu Storyboard Flows.

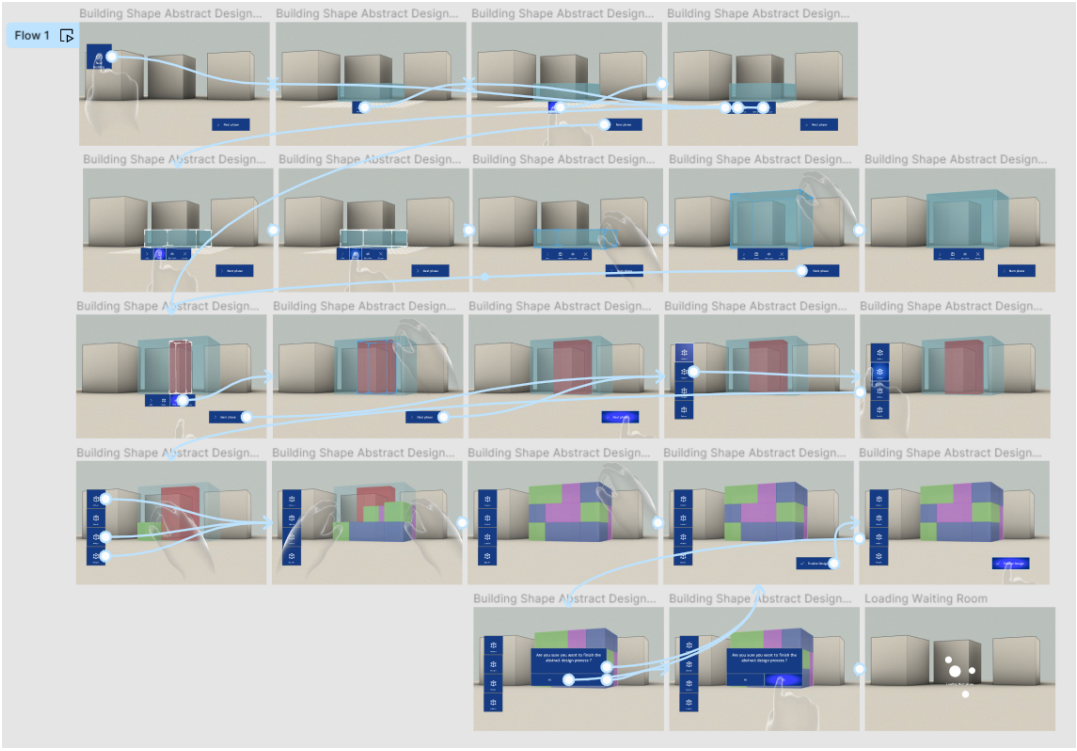


FIGURE 4.42: App Abstract Design Session.

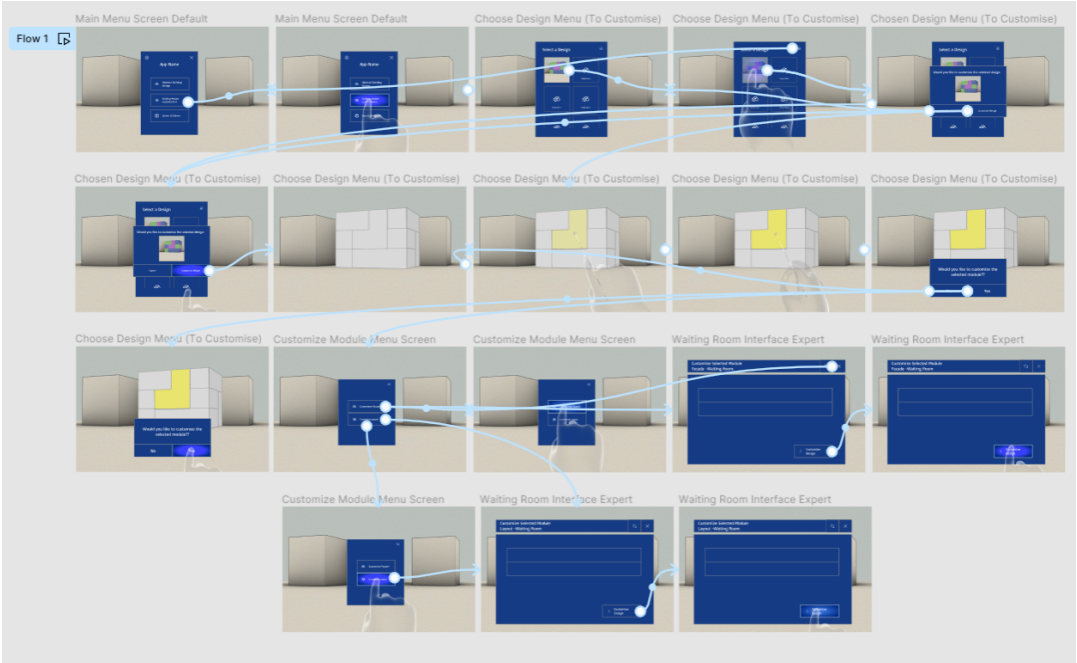


FIGURE 4.43: App Customize Session Storyboard.

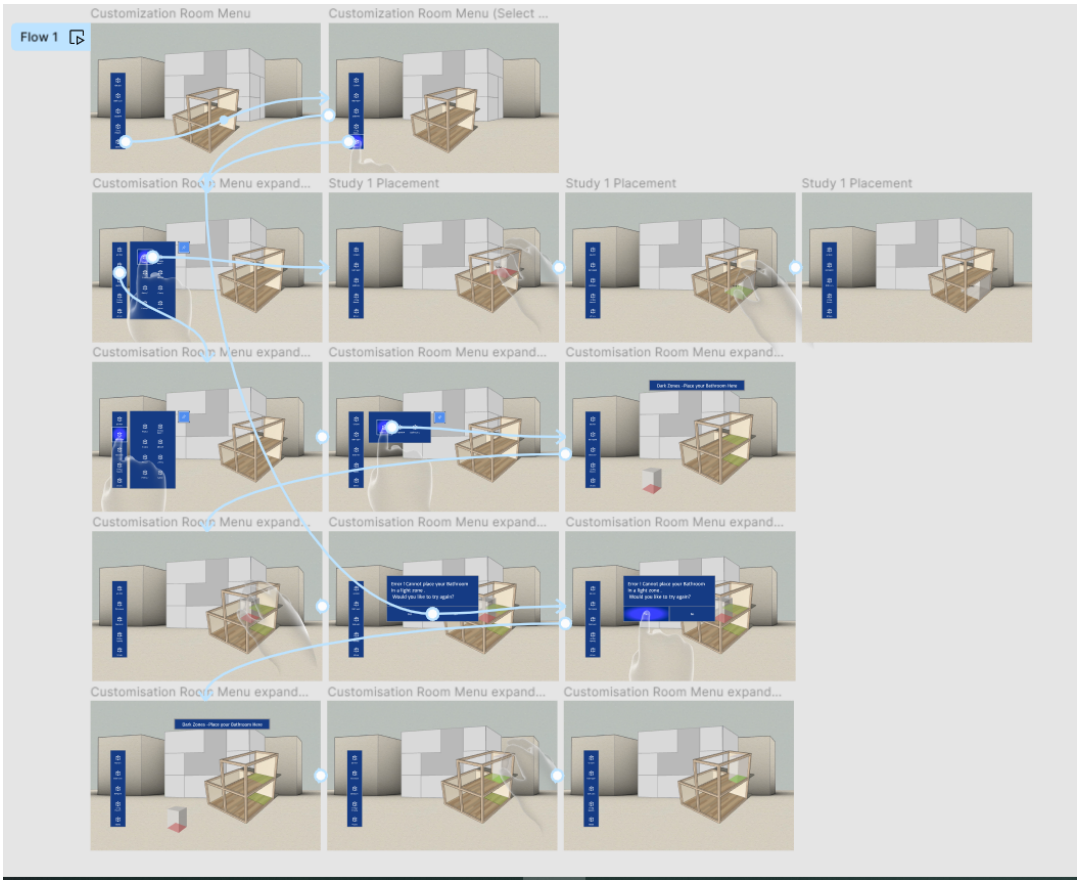


FIGURE 4.44: App Customize Dwelling Layout Storyboard.

Implementation

5.1 Introduction

In this section, a technical implementation analysis will be performed from the developer's perspective. This chapter details the steps of the development process.

This thesis is a research endeavor within the Augmented Reality (AR) technology field, implemented using the latest technologies available at the time of writing. Due to the rapidly evolving nature of AR, resolving compatibility issues between development tools and packages was a recurring challenge.

Unity Technologies' Real Time Development Platform [30], along with Microsoft Visual Studio [31] and the HoloLens 2, formed the core of our development platform, enabling us to code and compile our application for the HoloLens 2 head-mounted display (HMD).

The initial step involved a thorough study of the MRTK Dev Template project to familiarize ourselves with the Mixed Reality Toolkit 3 (MRTK3) packages and their capabilities. This streamlined our understanding of the SDK and allowed us to bypass its complex setup process.

Subsequently, a sample project, was created, to import MRTK3, OpenXR, and spatial features and to implement basic AR functionality.

A significant portion of this thesis was developed remotely, without direct access to a HoloLens 2 device. To overcome this limitation, MRTK3's input-agnostic design was leveraged and the MRTK Input Simulator was utilized. This allowed the simulation of various input modalities, such as hand tracking and gaze, enabling a degree of testing and validation of the project's core interactions.

To facilitate the multiplayer aspect of the application, enabling users to simultaneously interact with shared virtual content, Unity's comprehensive networking ecosystem was used. This involved integrating three key components: Lobbies, Relay, and Netcode for GameObjects. The combined power of Lobbies [32], Relay [33], and Netcode [34] allowed a seamless user synchronization and connection. Lobbies facilitated the initial grouping of users, Relay ensured low-latency communication for real-time interaction, and Netcode handled the synchronization of game objects and their states. This integrated approach enabled the creation of a truly collaborative AR experience where users could interact with shared virtual content in a synchronized and responsive manner.

Having established these foundational features, the development of the core functionality of the application followed: real-time collaborative building development in augmented reality. This involved creating custom user interfaces to enable users to seamlessly interact with and manipulate 3D models within a shared virtual environment to create and customize buildings and dwellings, save, view and rate them.

5.2 Core Application Structure

In the core of the Application there are some scripts working in tandem. One of the key scripts of the application is the `AppManager` Singleton script. This script helps coordinate most subscripts of the application. It contains the implementation of a state machine that transitions between the main stages (AppPhase enum) 5.1 of the application.

```
15      #region AppPhases & AppPhaseChangeEvent
        72 references
16      public enum AppPhase
17      {
18          Startup,
19          Role_Choice,
20          MainMenu,
21          Tutorial,
22          HomeDialogue,
23          Lobby_List_Design,
24          Lobby_List_Customize,
25          Lobby_Design,
26          Lobby_Customize,
27          Design_P1_Host,
28          Design_P1,
29          Design_P2,
30          Customize_P1,
31          Customize_P2,
32          Saving_Design,
33          Visualize_Menu,
34          Visualize
35
36      }
37      //Get current phase of the app
        1 reference
38      public AppPhase CurrentPhase()...
42      //Get the previous App Phase
        1 reference
43      public AppPhase PreviousPhase()...
47
48      //custom event fired when the app changes phase
        4 references
49      public class AppPhaseChangeEvent...
58
59      public delegate void OnAppPhaseChange(AppPhaseChangeEvent e);
60      public OnAppPhaseChange OnAppPhaseChanged; // Event variable
```

FIGURE 5.1: AppManager AppPhase Setup.

The `AppManager` is also managing the main application Dialog and coordinating Loading animations. It contains functions such as **RunTutorial()** and **RoleChoice()**

that are responsible for our main application subuse cases.

5.3 User Interface and Interactions

After each Appmanager transition, an event is fired that triggers the `UIManager's` response to the AppPhaseChange event. The `UIManager script` 5.2 manages the main UI transitions using the AppManager's AppPhaseChangeEvent in the function **UpdateAppPhaseEvent**. It contains methods to show and hide UI elements thus toggling the various UI components following the application flow.

```
Unity Script (1 asset reference) | 6 references
4   public class UIManager : MonoBehaviour
5   {
6       private static UIManager _instance;
7
8       [SerializeField] private List<GameObject> uiElements;  // List to hold all UI elements
9       [SerializeField] private List<GameObject> sceneHelpers;
10      private Dictionary<string, GameObject> uiElementsDict;
11
12      private string previousInterface;
13      private string currentInterface;
14      //singleton
        2 references
15      public static UIManager Instance...
35
        Unity Message | 0 references
36      public void Start()...
45
46      #region UI init & toggling
47      // Initialize all UI elements and store them in a dictionary for easy access
        2 references
48      private void InitializeUIElements()...
61
62      // Show a specific UI element by name
        11 references
63      public void Show(string uiElementName)...
74
75      // Hide a specific UI element by name
        1 reference
76      public void Hide(string uiElementName)...
87
88      // Hide all UI elements
        12 references
89      public void HideAll()...
96      #endregion
97
        2 references
98      private void UpdateAppPhaseEvent(AppManager.AppPhaseChangeEvent e)...
184
        Unity Message | 0 references
185     void OnDestroy() // Unsubscribe from event when UIManager is destroyed...
192 }
```

FIGURE 5.2: Application UI Manager Script.

5.3.1 UI Dialog Setup

Our Dialog setup follows the MRTKDevTemplate DialogExample scene. More specifically the project utilized the `DialogPool script` as is in the sample, controlled by a `DialogManager` 5.6 Singleton script.

The DialogPool 5.3 takes a dialog prefab with a certain structure as an argument. In this case the CanvasDialog prefab 5.4 provided by the MRTK was utilized with its predefined DialogAnimator. The `DialogManager` alone manages all the application dialog in a centralized approach. It contains functions with parameters designed to show Dialog 5.5 with different text and number of choices synchronously or asynchronously.

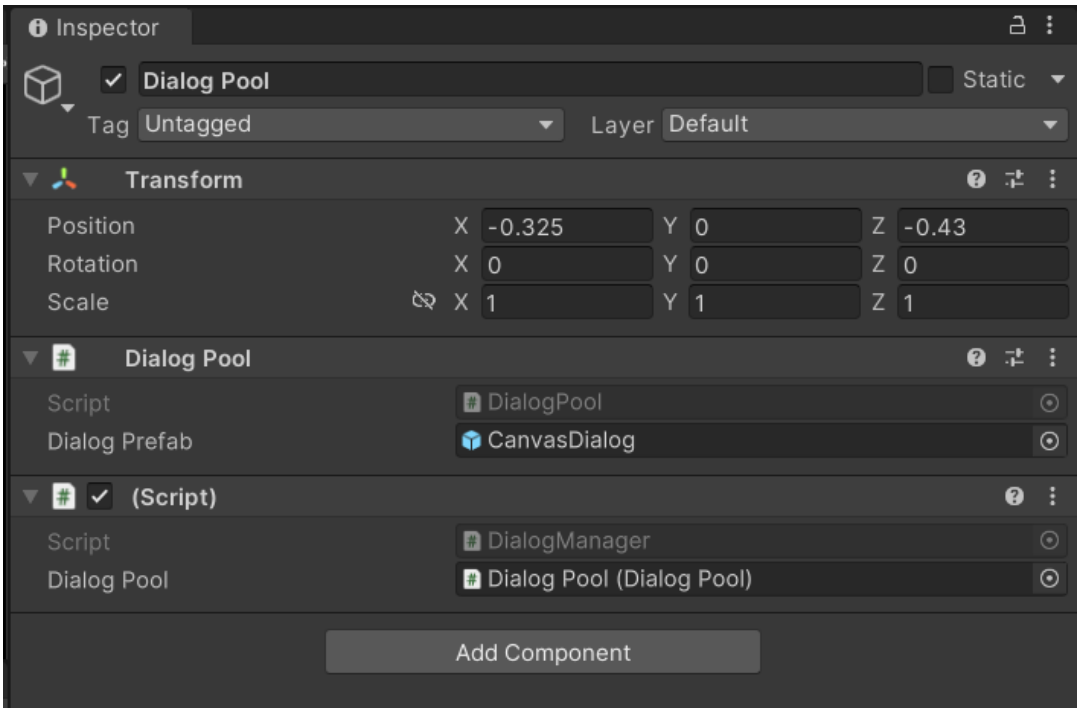


FIGURE 5.3: Canvas Dialog Pool.

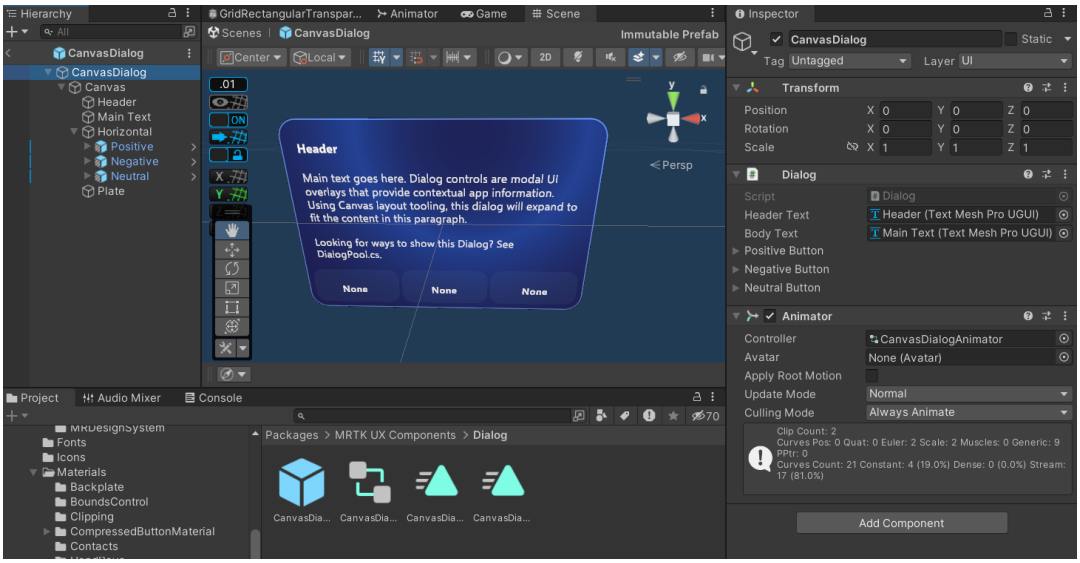


FIGURE 5.4: MRTK UX Components Canvas Dialog Prefab.

5.3.2 Loading UI

A simple customizable *Loading Screen* 5.7 was created for creating smoother scene transitions. The screen consists of a loading text and an animation of three loading dots. A `Loading Manager` script was created to control the loading screen.

5.3.3 HandMenu Setup

The *Hand Menu* **??**is a component designed to help the user at any state of the application. It containes a 'Home Button' and a 'Scene Helper' Toggle that users can use at any time to go back to the main menu and get help on the scene.

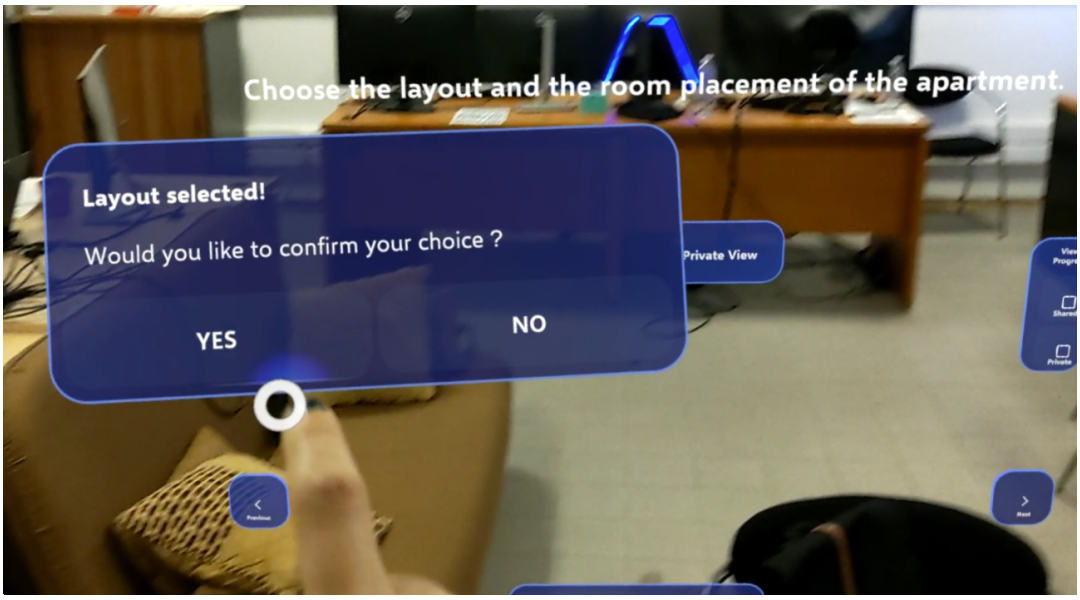


FIGURE 5.5: Application Dialog Instance

```
Unity Script (1 asset reference) | 6 references
public class DialogManager : MonoBehaviour
{
    public DialogPool DialogPool;

    private static DialogManager _instance;

    Unity Message | 0 references
    protected virtual void Awake()...
    2 references
    public static DialogManager Instance...


    0 references
    public Task<DialogButtonType> SpawnDialogWithAsync(string Header, string Body, string Neutral)
    {
        return ShowAsyncDialog(Header, Body, Neutral);
    }

    2 references
    public Task<DialogButtonType> SpawnDialogWithAsync(string Header, string Body, string Positive, string Negative)
    {
        return ShowAsyncDialog(Header, Body, Positive, Negative);
    }

    1 reference
    private async Task<DialogButtonType> ShowAsyncDialog(string Header, string Body, string Neutral)...


    1 reference
    private async Task<DialogButtonType> ShowAsyncDialog(string Header, string Body, string Positive, string Negative)...

}
```

FIGURE 5.6: Application Dialog Manager Script.

5.3.4 Scene Helper Component

The `Scene Helper` is another component that is utilized to consult the user about their existing MR surroundings. Each UI element contains a 'Scene Helper'.

This component contains tips on how each interface of the application is used and can guide the user during their use of the app. Each Scene Helper component has a number of Tooltip children.

The `Tooltip` 5.10 is a hint that describes what a specific button or interface is there for 5.11.

It is comprised of a custom prefab dialog interface that contains both the text and animator components necessary to achieve the tooltip functionality. Each 'Tooltip'

FIGURE 5.7: Loading UI.

can be toggled in two ways, either by gazing at the interface it describes for a predetermined time interval or by toggling it on, using the 'Hand Menu SceneHelper' toggle. This functionality is achieved by utilizing a custom TooltipController script coupled with the MRTK's Statefull Interactable or Pressable Button scripts.

The `TooltipController` 5.12 contains functions **OnGazeHoverEntered()** and **OnGazeHoverExited()** that determine the tooltip behaviour when the user gaze interacts with it. Setting the Tooltip Controller functions to fire when the Pressable Button's or Statefull Interactable's corresponding MRTK events happen, completes the tooltip functionality 5.13.

5.3.5 Tutorial Interface

The *Tutorial Interface* 5.15 5.14 is designed to guide users through the core interactions and functionalities of the application. It utilizes a combination of visual cues, and interactive elements to provide a comprehensive and engaging learning experience.

The **Tutorial** employs a step-by-step approach, introducing users to key concepts and gestures sequentially. Each step is accompanied by clear instructions and visual demonstrations, ensuring that users can easily follow along and practice the interactions. The interface also incorporates feedback mechanisms, providing visual and auditory cues to confirm successful completion of each step. This interactive approach encourages active participation and reinforces learning.

When the app starts the *View Tutorial Prompt* 5.16 is shown, where the user can choose to view the tutorial or not. The Tutorial can also be accessed anytime using the HandMenu.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

Unity Script (1 asset reference) | 28 references
public class LoadingManager : MonoBehaviour
{

    private static LoadingManager _instance;

    [SerializeField] private GameObject _loadingGameObject;
    [SerializeField] private TMP_Text _loadingText;

    24 references
    public static LoadingManager Instance...

    8 references
    public void SetLoadingText(string loadingText)
    {
        _loadingText.text=loadingText;
    }

    8 references
    public void EnableLoadingScreen() {
    _loadingGameObject.SetActive(true);
    }

    8 references
    public void DisableLoadingScreen()
    {
        _loadingGameObject.SetActive(false);
    }


}
```

FIGURE 5.8: Loading Dialog Manager Script.

5.3.6 Name Input User Interface

The *Name Input User Interface* 5.18 5.17 is comprised of: a 'Mixed reality TMP' input field and a non native keyboard. The setup follows the MRTKDevTemplate's NonNativeKeyboard scene. When a user presses on the input field the non native keyboard shows , so they can input their user name. When enter is pressed a dialog is spawn to confirm the user's choice.

5.3.7 User Avatar Color Choice Interface

The *User Color Choice Interface* 5.19 is comprised of: a Color Menu that shows the User Avatar in the chosen color. The user can choose a color and press the 'Confirm' button to confirm it. This color will be used to distinguish the users' avatars from each other and their animated hands during collaboration.

5.3.8 User Role Choice Interface

The *User Role Choice Interface* 5.21 is comprised of: an 'Expert' button and a 'Non Expert' button. The user can choose their role in the application.

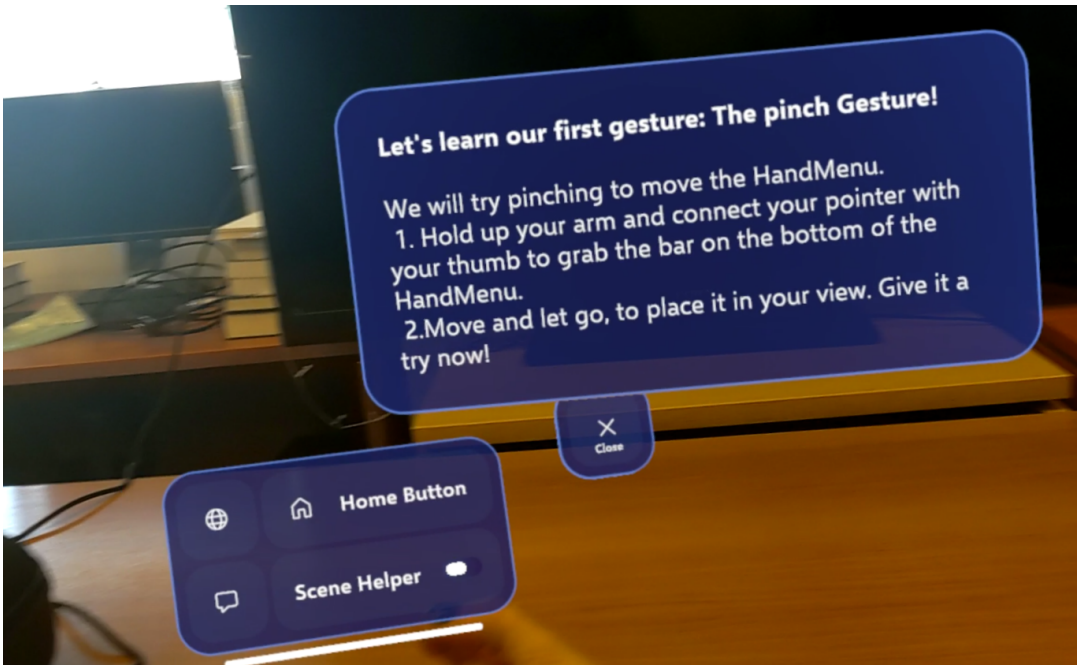


FIGURE 5.9: Hand Menu in the User Tutorial

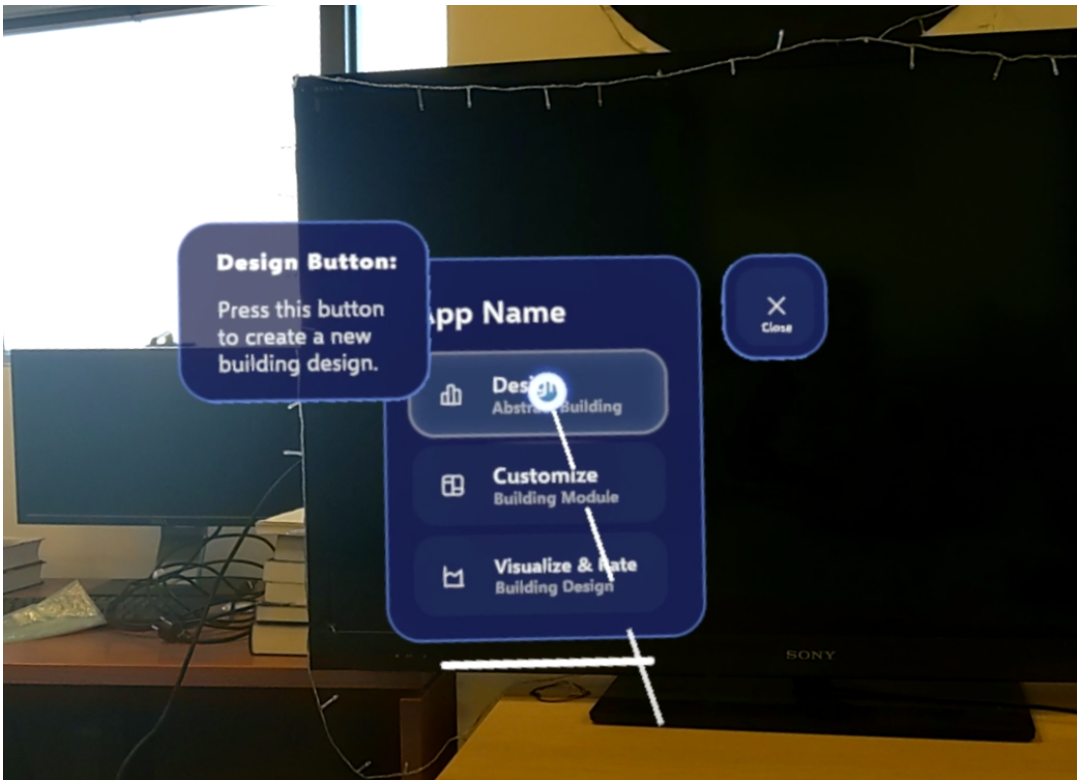


FIGURE 5.10: Main Menu design button Tooltip showing

5.3.9 Main Menu User Interface

The *Main Menu User Interafce* 5.20 is comprised of: a 'SceneHelper', a 'Main Panel' and a 'Close' button components. Unity's Horizontal and Vertical Layout Groups were utilized for the composition of the Menu. The menu has: 3 main buttons for the three main use cases of the application and finally a 'Close Application' button.

FIGURE 5.11: Main Menu Design Button Tooltip.

5.3.10 Join/Create Lobby User Interface

The *Join/Create Lobby Interface* 5.22 is comprised of the SceneHelper component and the Main Panel Components.

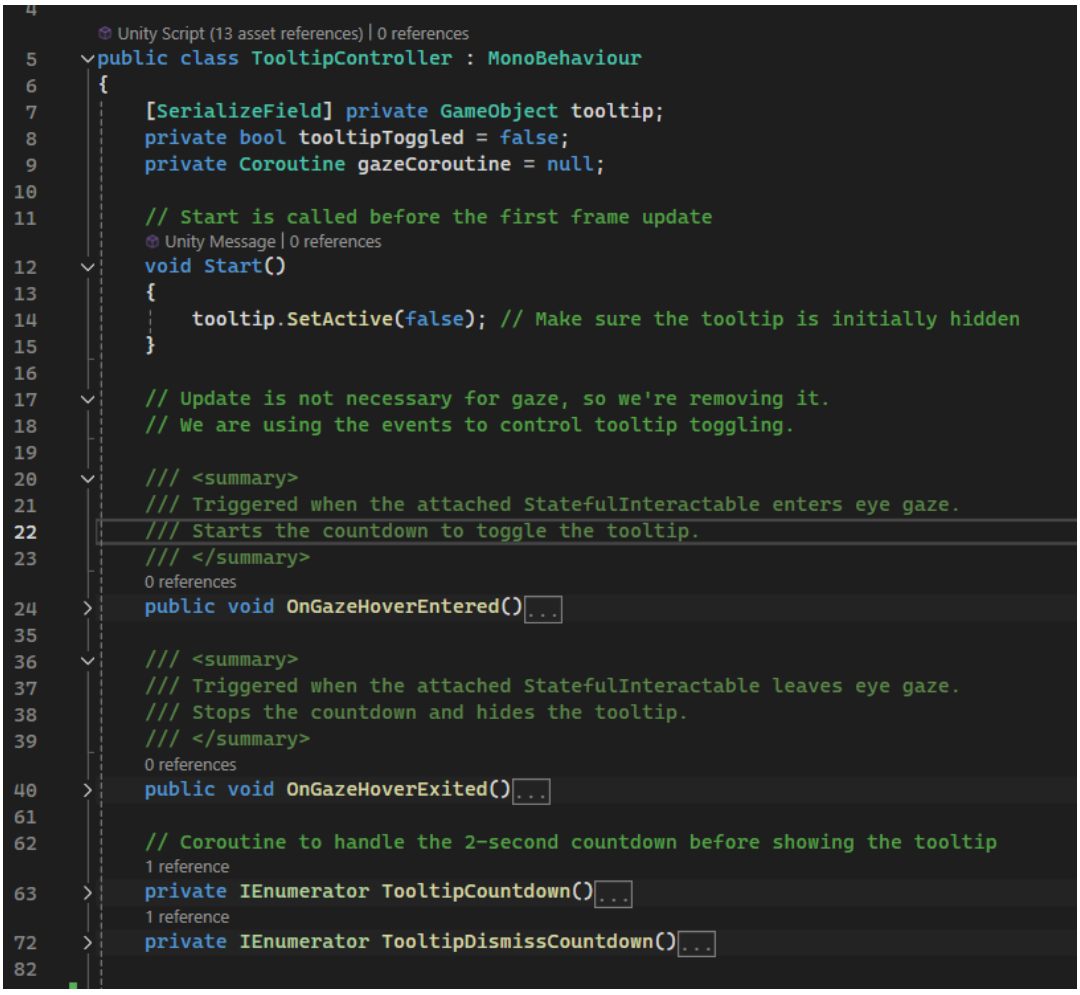
The Main Panel contains Horizontal and Vertical and Grid Layout Groups as needed in order to make the interface adjust automatically depending on the number of lobbies it contains. The Top Bar has a 'Refresh Lobbies Shown' button and a 'Home' button.The ButtonPanel contains a GridLayout Group with as many 'Lobby Button' prefabs as the existing Lobbies. Each 'Lobby Button' displays the Lobby's name and number of players joined. Finally, in the Bottom Bar we have the 'Create New Lobby' button.

The Interface contains the `LobbyListUI` script that controlls the aforementioned UI elements and other scripts that determine the positioning, sound, and animation of the Interface at runtime. More information on the `LobbyList UI` script is provided in the next section.

5.3.11 Lobby User Interface

The *Lobby UI* 5.23 5.24 is the prefab that contains all lobby related elements. It is is comprised of the 'SceneHelper' and the 'Main Panel' components.

In the Top Bar of the lobby interface, the **Lobby Name**, **Number of Players** and **Session Type** (Design or Customize) info is displayed. The top Bar also contains a 'Back' button. In the **ParticipantContainer** there is a Vertical Layout Group with as many 'Paricipant' prefabs as the joined Participants. Each 'Participant' Prefab displays the Participant's name and (only for the lobby host) their individual 'Kick Participant' button. Finally, in the Bottom Bar (solely for the lobby host) we have the 'Begin Session' button.



```
Unity Script (13 asset references) | 0 references
public class TooltipController : MonoBehaviour
{
    [SerializeField] private GameObject tooltip;
    private bool tooltipToggled = false;
    private Coroutine gazeCoroutine = null;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        tooltip.SetActive(false); // Make sure the tooltip is initially hidden
    }

    // Update is not necessary for gaze, so we're removing it.
    // We are using the events to control tooltip toggling.

    /// <summary>
    /// Triggered when the attached StatefulInteractable enters eye gaze.
    /// Starts the countdown to toggle the tooltip.
    /// </summary>
    0 references
    public void OnGazeHoverEntered()...

    /// <summary>
    /// Triggered when the attached StatefulInteractable leaves eye gaze.
    /// Stops the countdown and hides the tooltip.
    /// </summary>
    0 references
    public void OnGazeHoverExited()...

    // Coroutine to handle the 2-second countdown before showing the tooltip
    1 reference
    private IEnumerator TooltipCountdown()...
    1 reference
    private IEnumerator TooltipDismissCountdown()...
```

FIGURE 5.12: Tooltip Controller Script.

5.3.12 Design Interface

The *Design Interface* is comprised of two components. The non-networked 'Create New Building' button and its networked part, the 'Building Model' Prefab 5.26. The non networked component contains a transform where the Building Prefab is spawned.

When users begin a session the host is presented with the 'Create New Building' button 5.25 and the rest of the users are presented with a Loading animation.

When spawned, the 'Building Prefab' contains the 'Building Model' 5.27, its'Manipulation Bar', and the 'Next Phase' Button. The host can use the manipulation bar to place the building model on a vertical surface and then finalize the placement using the 'Next Phas'e Button. Then the **Design_P2_UI** 5.27 is shown for all users. This interface contains:

- **The Module Menu**: Contains a button for each dwelling. Users can press each button to spawn their desired apartment and proceed to place it on the building.
- **The Module Dock**: Is the transform where new apartments are spawned in the interface.

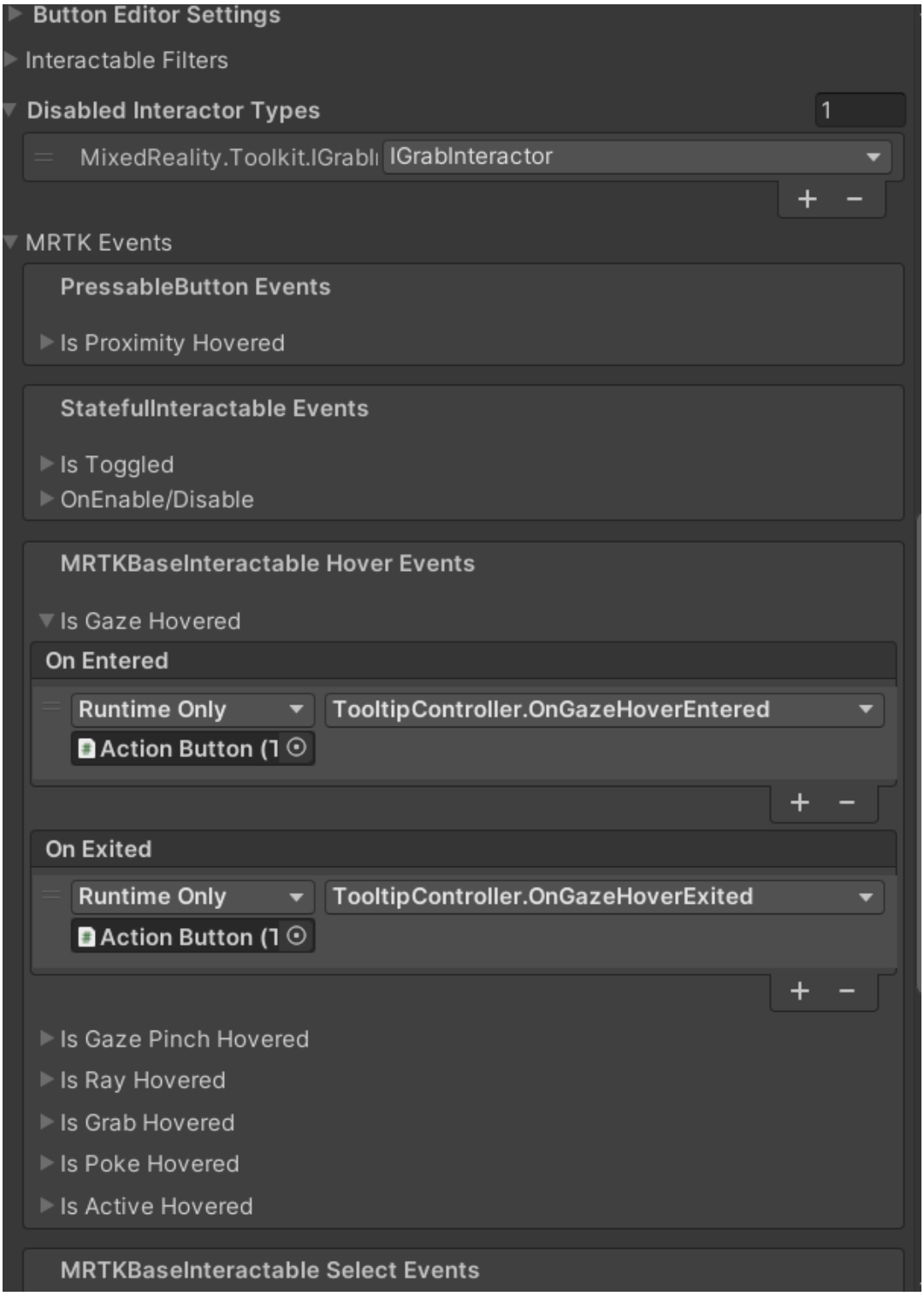


FIGURE 5.13: Pressable Button MRTK Events Tooltip Setup.

- **The Module Bin**: A bin that can be used to discard unwanted apartments spawned.
- **The Next Floor Button**: When a floor is completed with apartments, this button shows up in the host interface. The host approves the floor design by pressing this button.

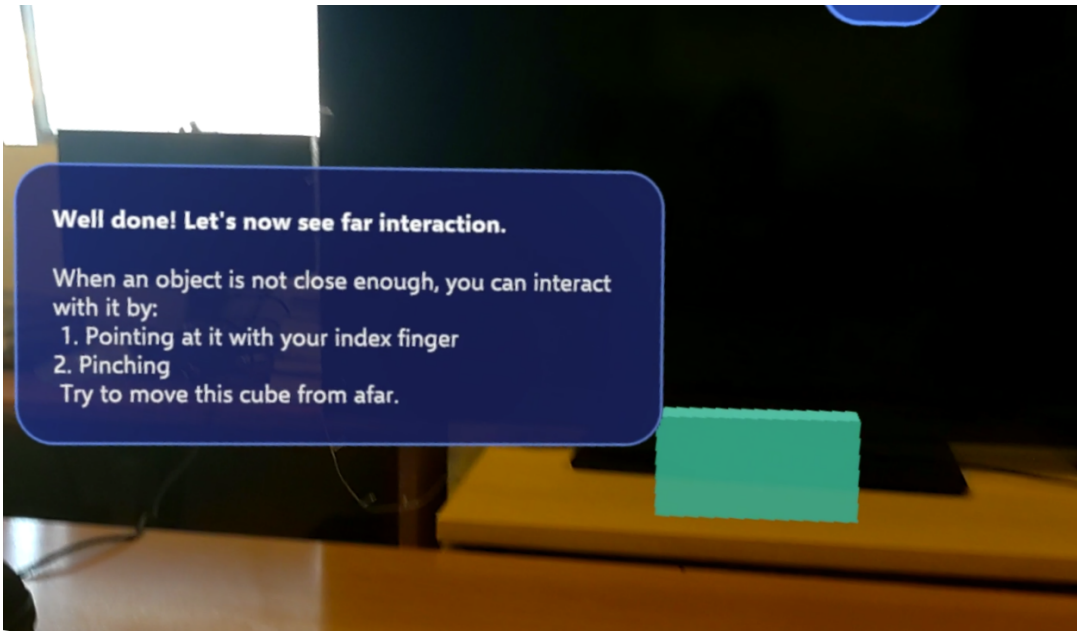


FIGURE 5.14: A snapshot of the tutorial interface showcasing how to move a faraway object.

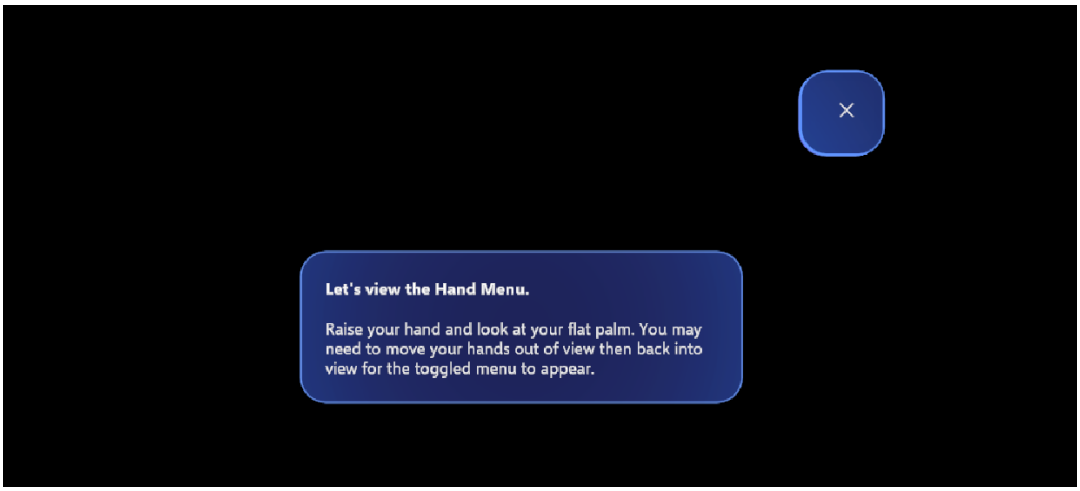


FIGURE 5.15: A snapshot of the tutorial interface showcasing how to view the Hand Menu.

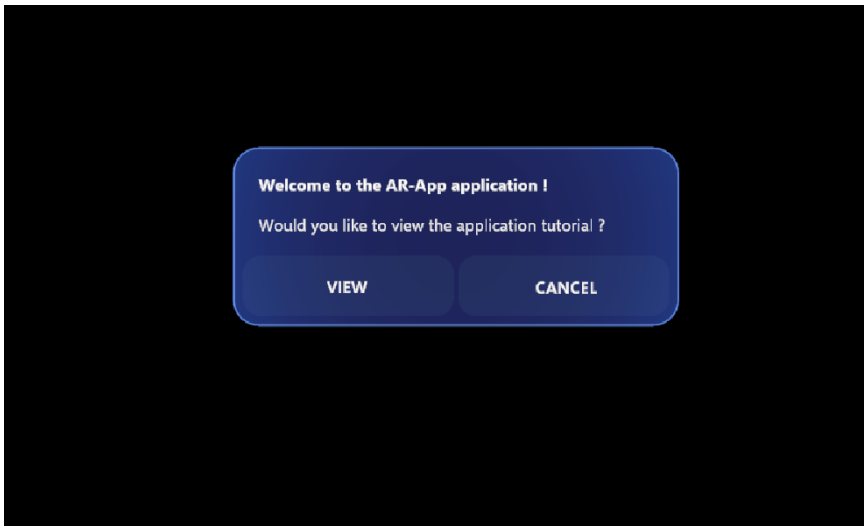


FIGURE 5.16: The tutorial Prompt.

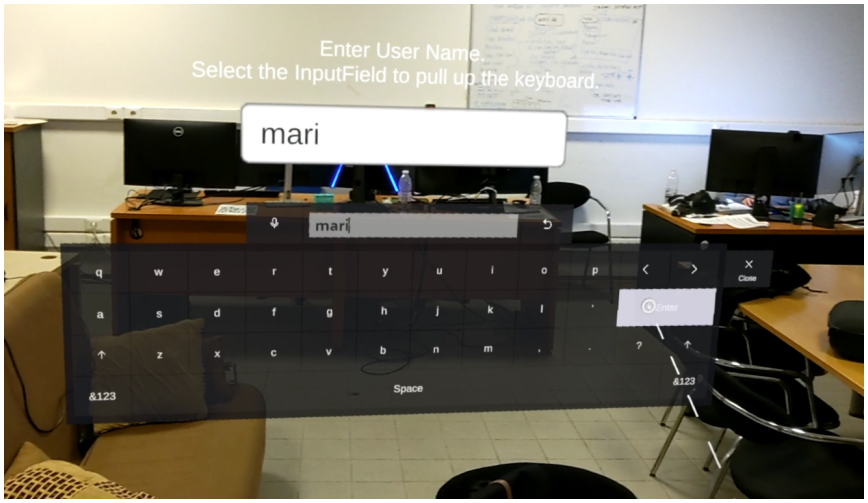


FIGURE 5.17: Name Input User Interface at runtime

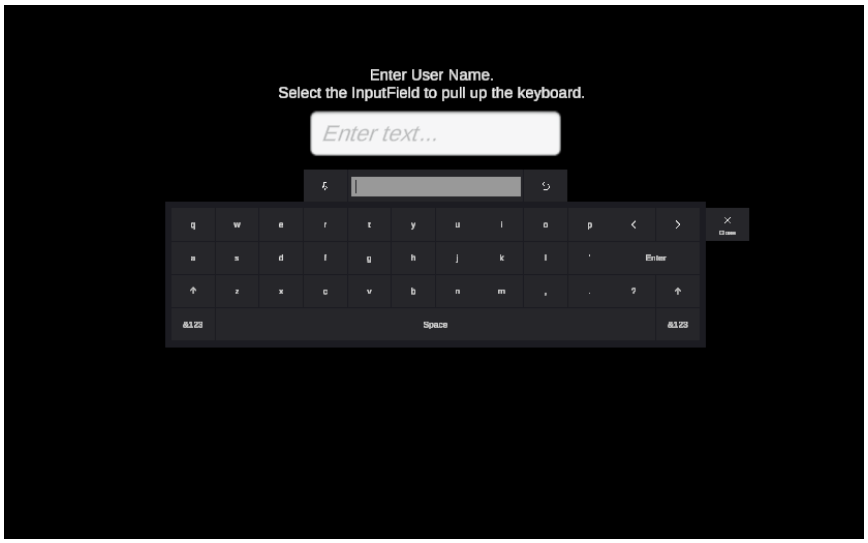


FIGURE 5.18: Name Input User Interface

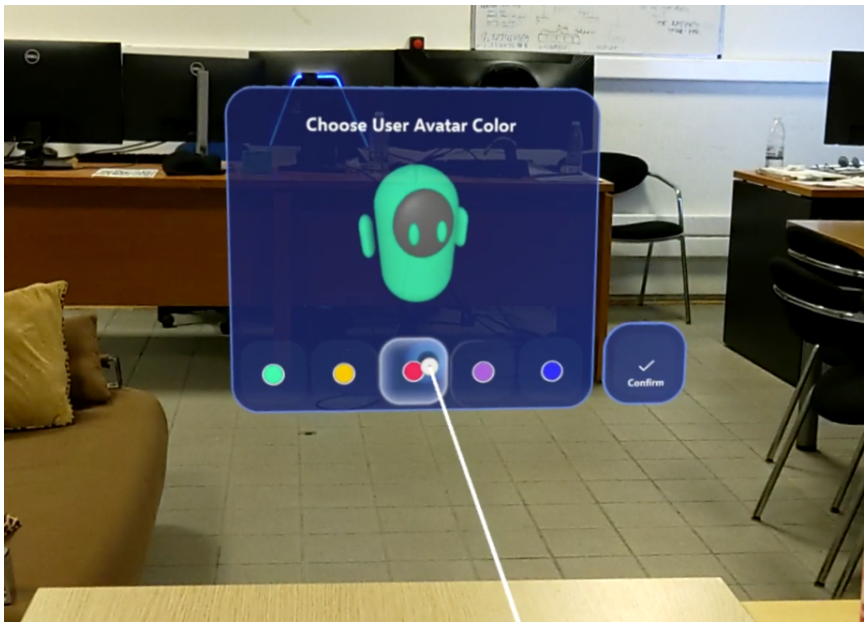


FIGURE 5.19: User Avatar Color Choice Interface

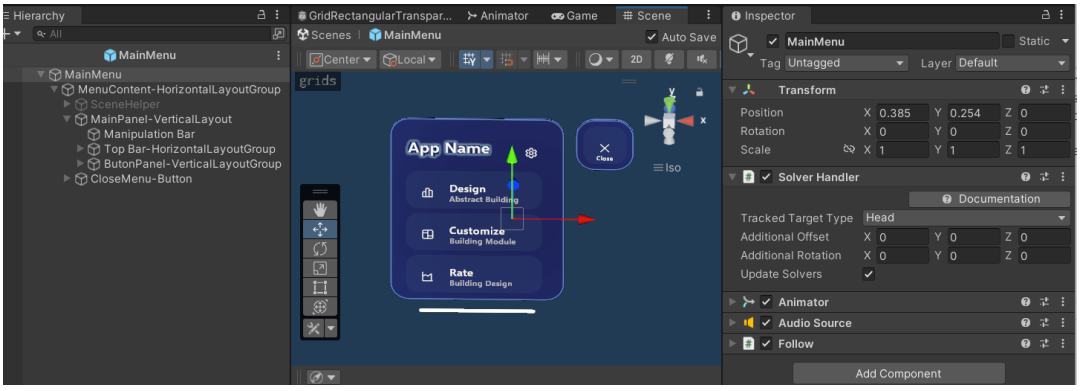


FIGURE 5.20: The Main Menu User Interface Prefab.

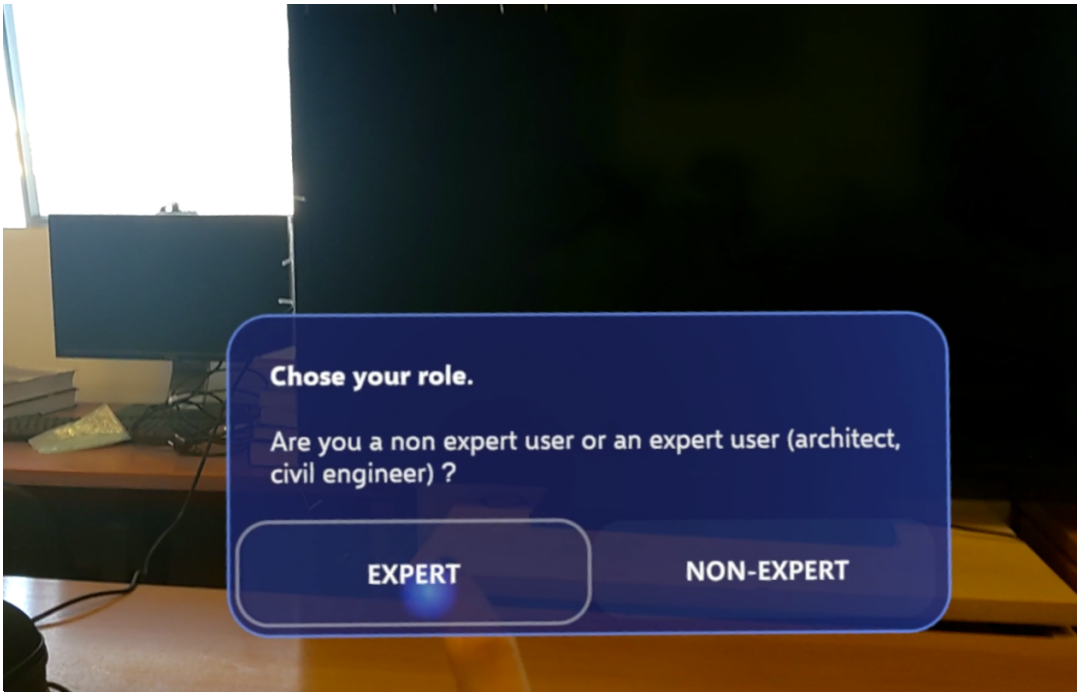


FIGURE 5.21: User Role Choice Interface.

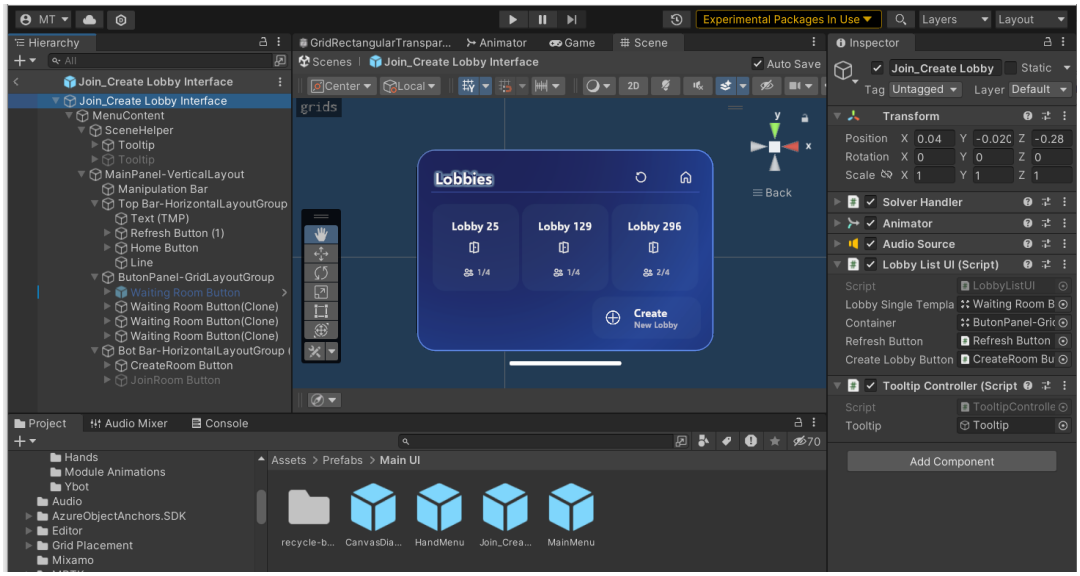


FIGURE 5.22: The Join Create Lobby User Interface Prefab.

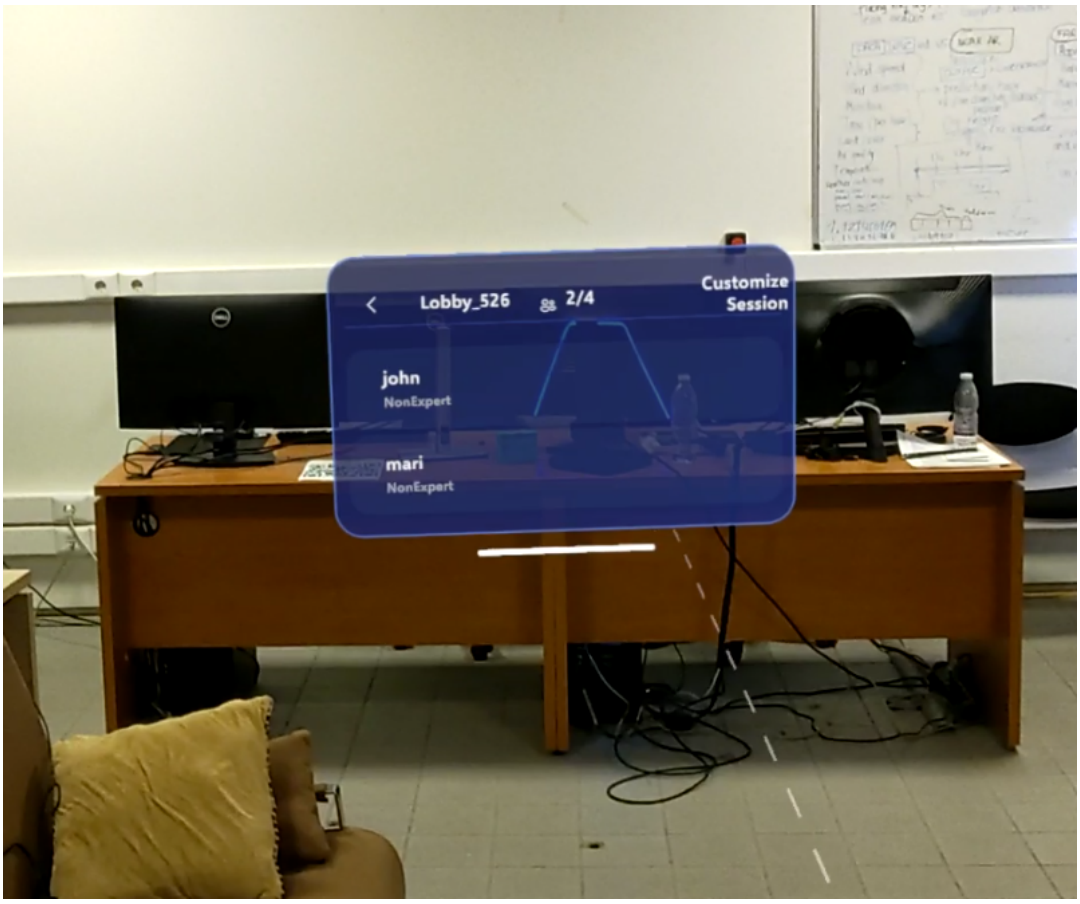


FIGURE 5.23: The Lobby User Interface.

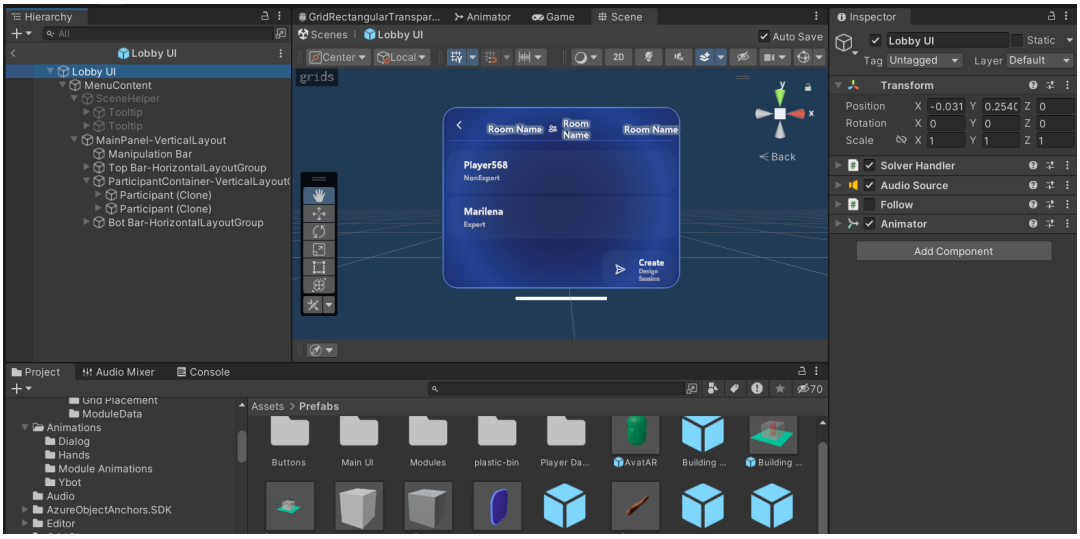


FIGURE 5.24: The Lobby User Interface Prefab.

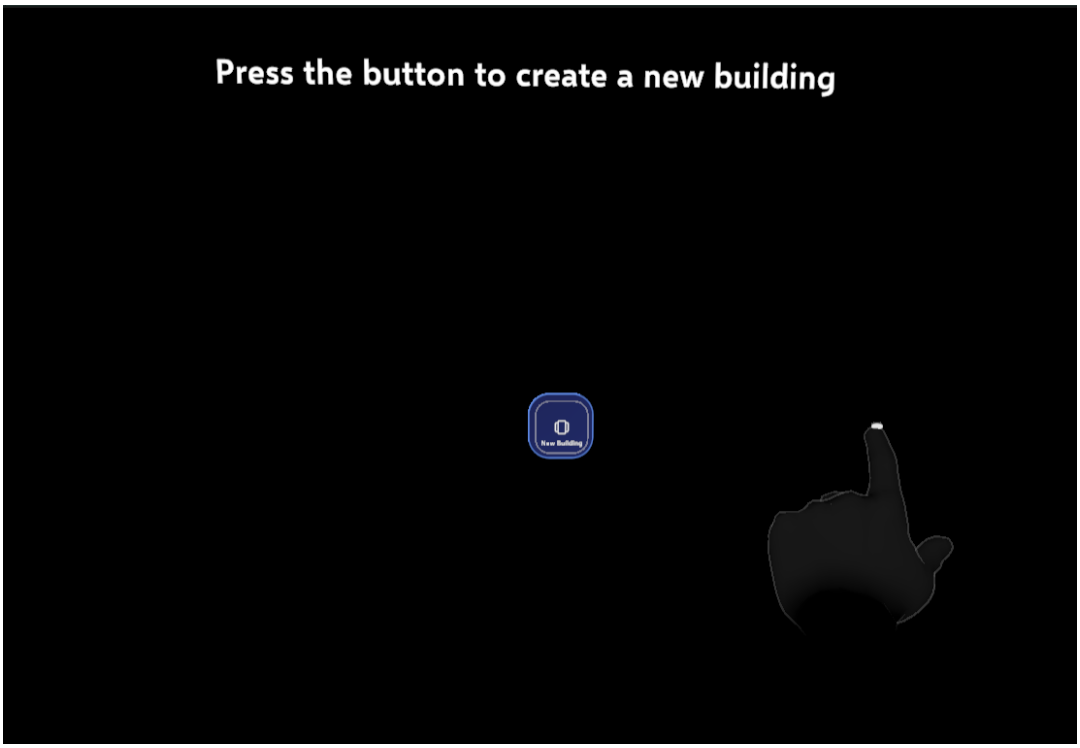


FIGURE 5.25: Create New Building Button.

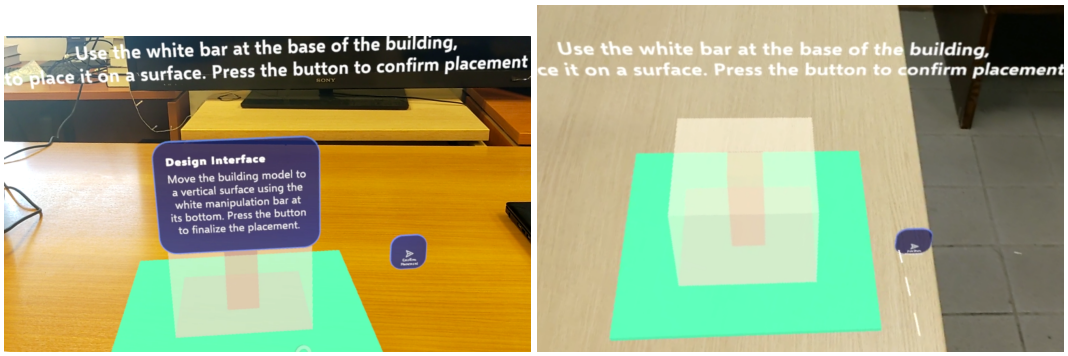


FIGURE 5.26: Design Interface at runtime.

- **Finalize Design Button**: Appears in the host's interface after all floors have been completed. Pressing this button saves the design and concludes the design phase.

In this environment all users are represented by their avatars and all their actions are represented by their hands. Users can collaborate and place apartments, filling each floor of the design until all floors are completed. Each time a floor is completed, the users confirm the placement and move on to the next one. Unwanted apartments can be placed in the bin. After the design is completed it is saved in the Unity Cloud.

5.3.13 Customize Interface

The user can select the lobby of any of the created Designs to customize an apartment of this design. After the user has chosen the design they would like to customize, they are prompted to the *Customize Interface's Dwelling Selection UI* 5.29 .

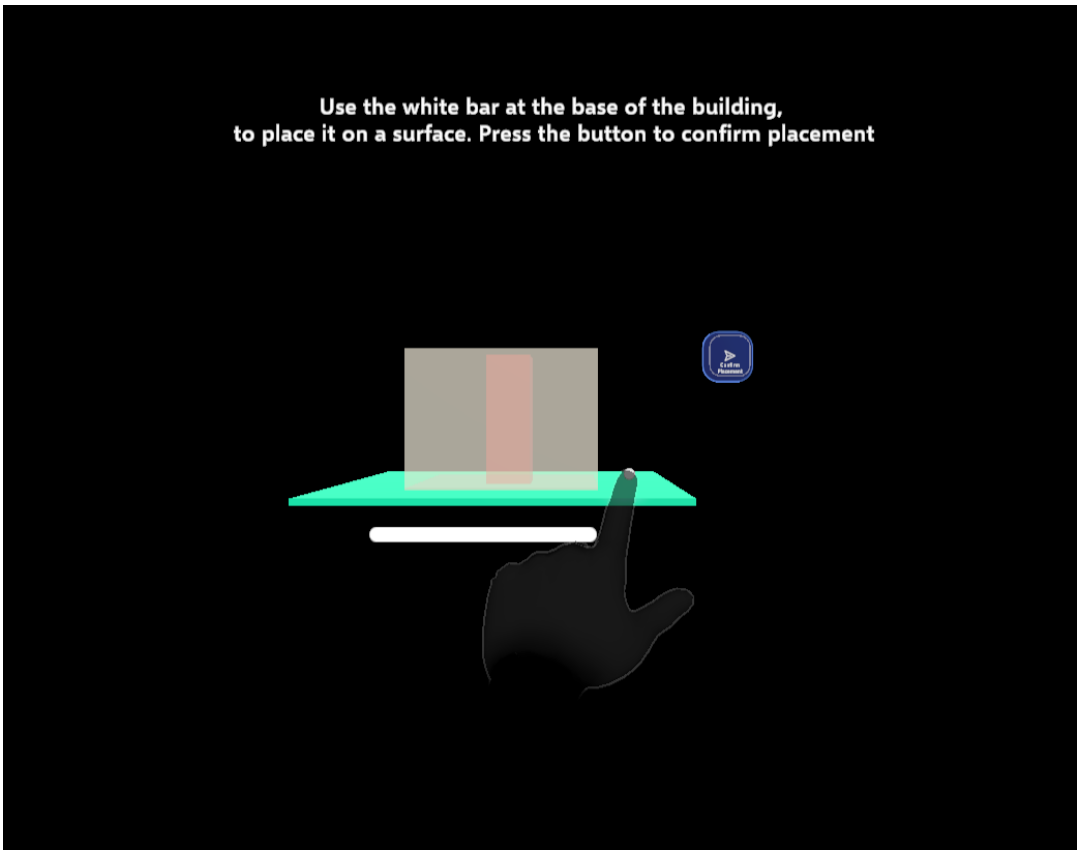


FIGURE 5.27: The Building Model Prefab.

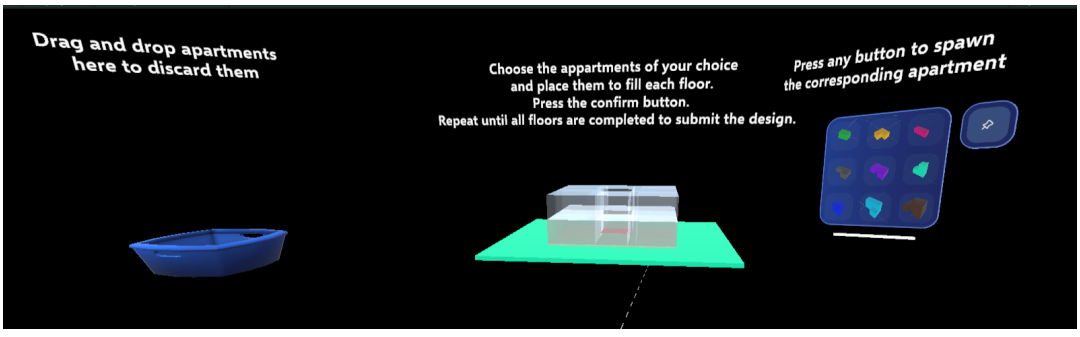


FIGURE 5.28: The Design_P2_UI.

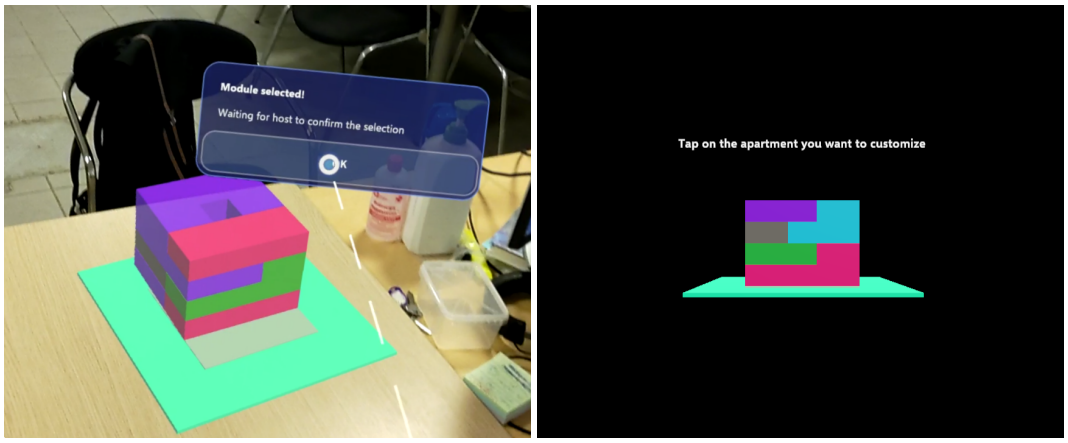


FIGURE 5.29: Dwelling Selection UI

This interface loads the design from the Unity CloudSave server and displays to the user the finished design product. From there the user can choose an apartment to customize. Depending on the user role (host, client) a dialog is prompted for the host user to confirm the apartment choice. After that, the setup of the customization interface is performed in the `CustomizeDwelling Selector` script.

The *Customize P1 UI* 5.30 5.31 is comprised of two user views, a Shared View and a Private View, where the user can work in a team or independently on a design. The view first makes the user choose the apartment layout 5.32 and then the placement of the rooms in it. The user can navigate the available designs and even view them side by side to compare them in **Compare Mode** 5.33 5.34 . The user an also go back and forth from their private design to the shared design.

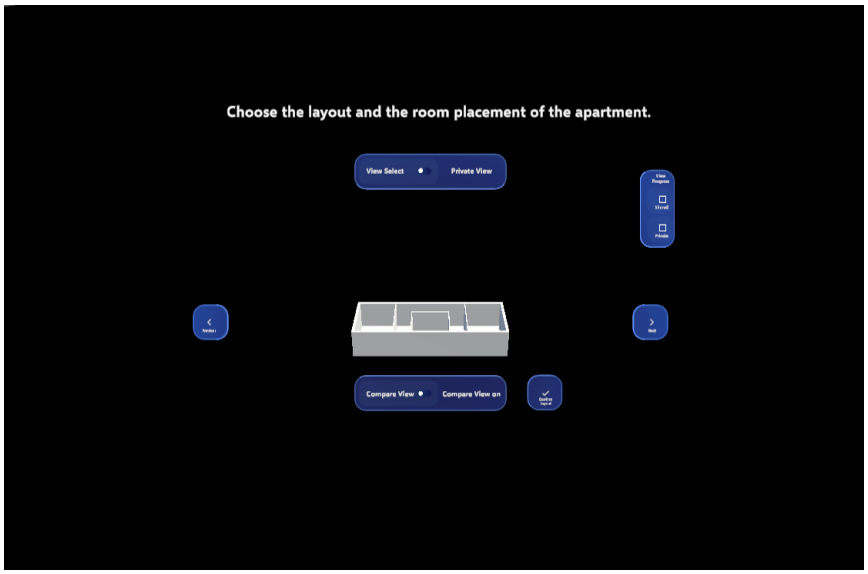


FIGURE 5.30: Customize P1 Interface

The interface involves multiple scripts that function independently while maintaining a cohesive interaction model. These scripts include `SharedView`, `PrivateView`, `ViewManager`, `ViewUIController`, `ProgressIndicator`, and `CustomizeManager`. Each script plays a unique role in managing shared and private views, user interactions, and customization workflows, ensuring a robust and synchronized AR experience.

FIGURE 5.31: Customize P1 Interface at runtime

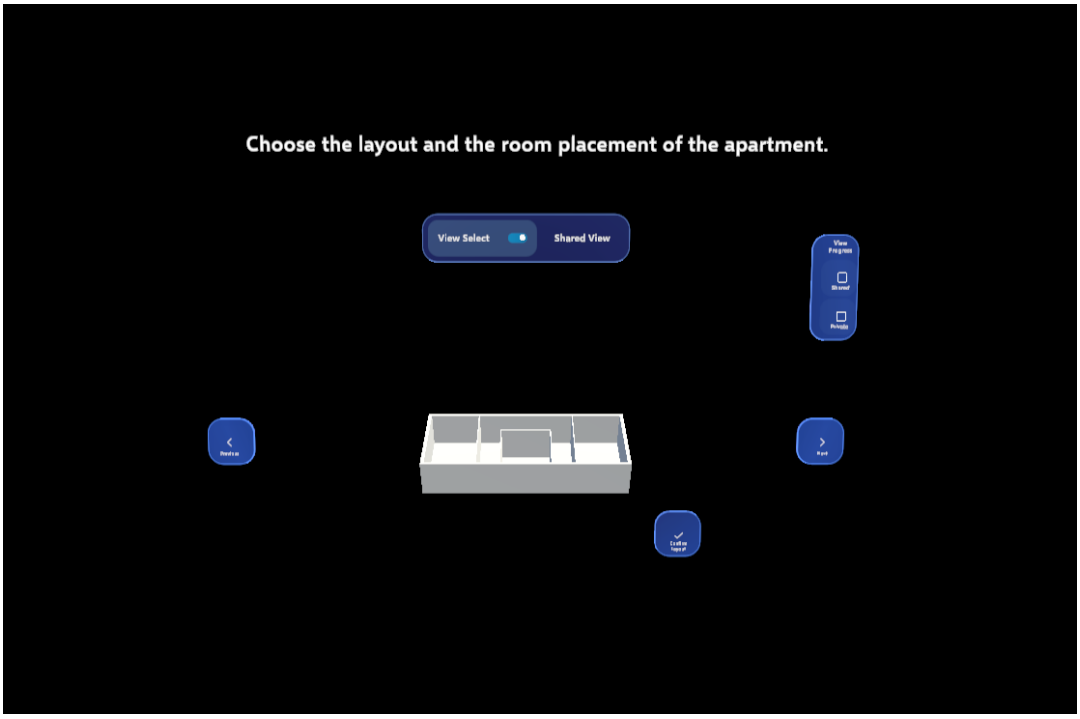


FIGURE 5.32: Customize P1 Shared View On

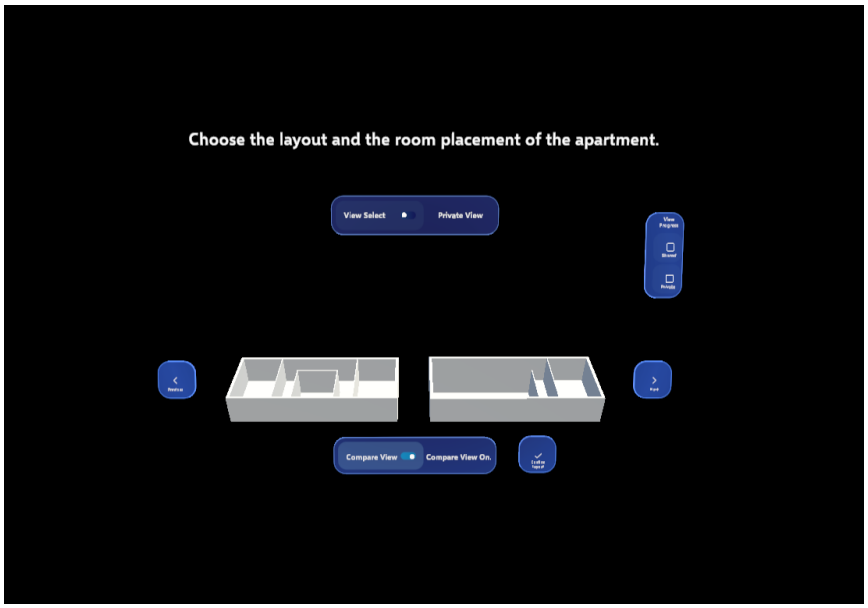


FIGURE 5.33: Customize P1 Compare Mode On

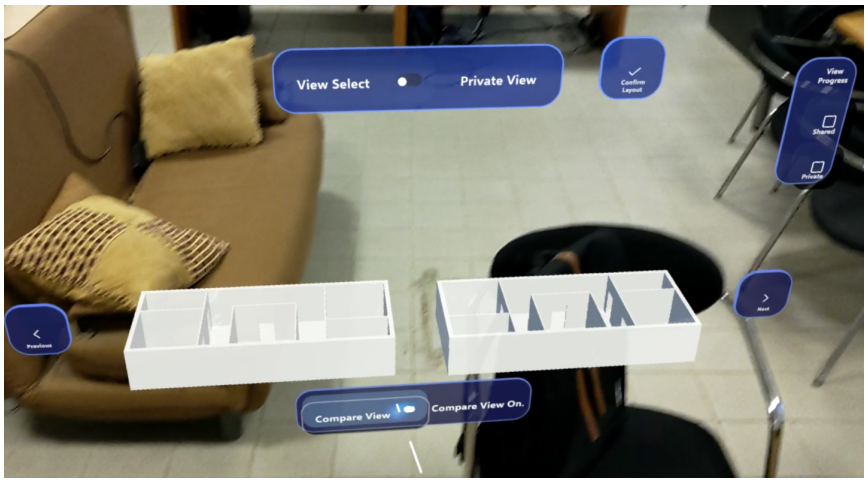


FIGURE 5.34: Customize P1 Compare Mode On at runtime

The IView Interface

The `IView` 5.35 interface establishes a common structure for managing views within the customization interface. It defines essential methods, such as **SetItems**, **NextItem**, **PreviousItem**, **ShowCurrentItem**, and **FinalizeChoice**, which enable consistent navigation and item management across all views. Additionally, properties such as **IsShared**, **IsComplete**, and **IsInCompareMode** provide state information, enabling seamless toggling between shared and private views or comparison modes.

```
public interface IView
{
    11 references
    bool IsComplete { get; set; } // Whether the view is shared
    7 references
    bool IsShared { get; } // Whether the view is shared
    14 references
    bool IsInCompareMode { get; set; } // Whether the view is in compare mode
    3 references
    Transform GetCurrentItem();// The current item being shown in the interface
    5 references
    void ResetCurrentIndex();
    10 references
    void SetItems(List<Transform> items);
    7 references
    void NextItem(); // Navigate to the next item
    5 references
    void PreviousItem(); // Navigate to the previous item
    21 references
    void ShowCurrentItem(); // Display the current item in the interface
    4 references
    void CompareViewConvert();//change view from and to compare mode

    5 references
    void ReportSharedViewState(bool state);
    9 references
    void DestroyCurrentItem();
    5 references
    Task FinalizeChoice();//choose layout
    5 references
    int SelectedIndex();

    4 references
    List<ulong> SharedClients();
    3 references
    void SetSharedItemsForClients(int index);
    4 references
    Transform GetLayoutContainer();
```

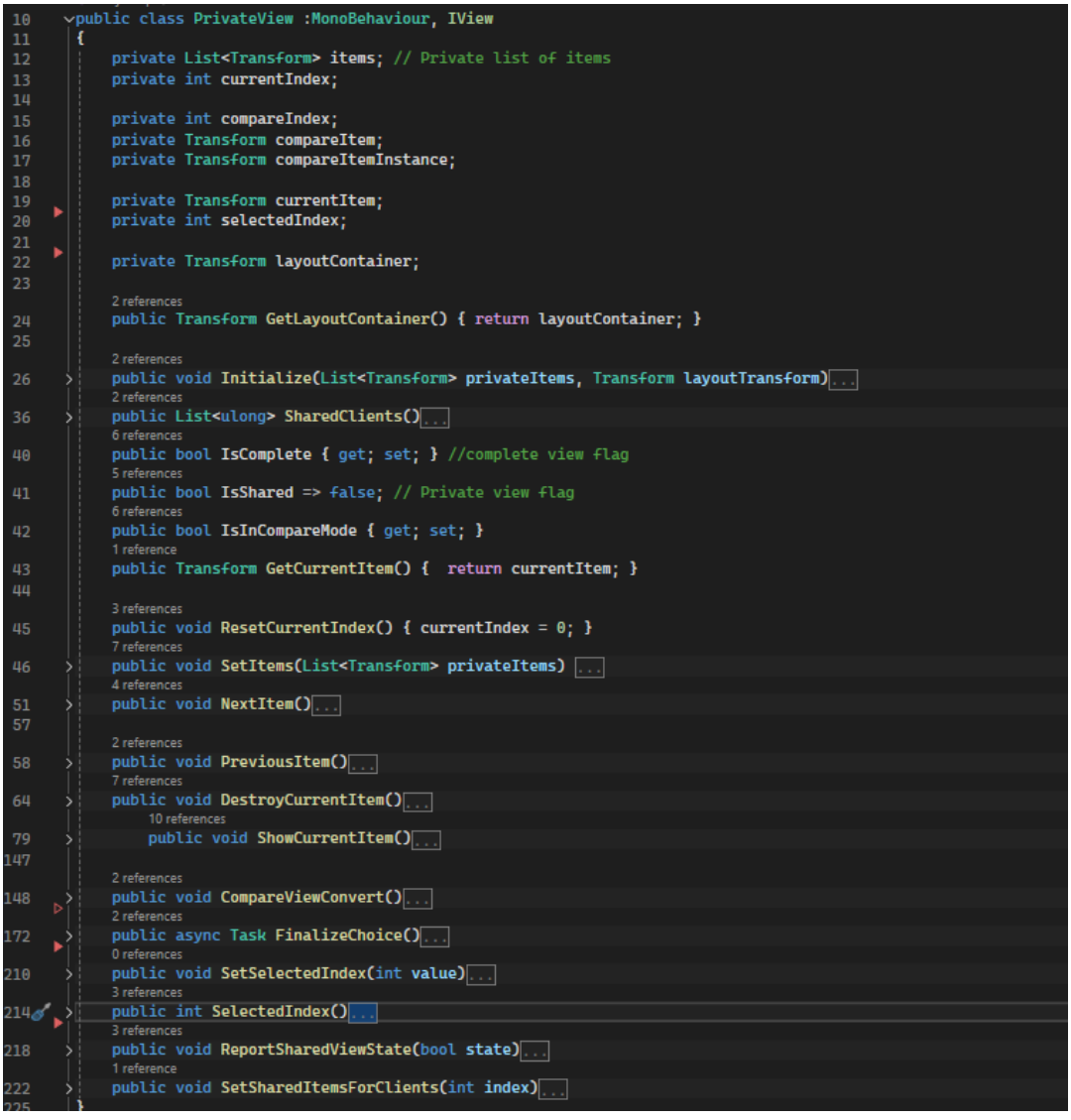
FIGURE 5.35: Iview Interface

The `SharedView` Script

The `SharedView` script implements the `IView` interface to manage the shared customization experience. By extending `NetworkBehaviour`, it enables synchronized interactions across all clients using Unity's Netcode framework. The script maintains a list of shared items and uses `NetworkVariable` and Remote Procedure Calls (RPCs) to synchronize key properties and propagate changes.

The `PrivateView` Script

The `PrivateView` 5.35 script also implements the `IView` interface but focuses on individual customization. Unlike `SharedView`, it operates locally without network synchronization. The script provides navigation, item display, and compare mode



```
10  public class PrivateView :MonoBehaviour, IView
11  {
12      private List<Transform> items; // Private list of items
13      private int currentIndex;
14
15      private int compareIndex;
16      private Transform compareItem;
17      private Transform compareItemInstance;
18
19      private Transform currentItem;
20      private int selectedIndex;
21
22      private Transform layoutContainer;
23
                2 references
24      public Transform GetLayoutContainer() { return layoutContainer; }
25
                2 references
26      public void Initialize(List<Transform> privateItems, Transform layoutTransform)...
                2 references
36      public List<ulong> SharedClients()...
                6 references
40      public bool IsComplete { get; set; } //complete view flag
                5 references
41      public bool IsShared => false; // Private view flag
                6 references
42      public bool IsInCompareMode { get; set; }
                1 reference
43      public Transform GetCurrentItem() {  return currentItem; }
44
                3 references
45      public void ResetCurrentIndex() { currentIndex = 0; }
                7 references
46      public void SetItems(List<Transform> privateItems) ...
                4 references
51      public void NextItem()...
57
                2 references
58      public void PreviousItem()...
                7 references
64      public void DestroyCurrentItem()...
                    10 references
79          public void ShowCurrentItem()...
147
                2 references
148     public void CompareViewConvert()...
                2 references
172     public async Task FinalizeChoice()...
                0 references
210     public void SetSelectedIndex(int value)...
                3 references
214     public int SelectedIndex()...
                3 references
218     public void ReportSharedViewState(bool state)...
                1 reference
222     public void SetSharedItemsForClients(int index)...
225 }
```

FIGURE 5.36: Private View Script

functionalities for a single user, allowing independent customization. The implementation ensures that private views operate consistently with shared views while maintaining isolation from collaborative workflows.

The `ViewManager` Script

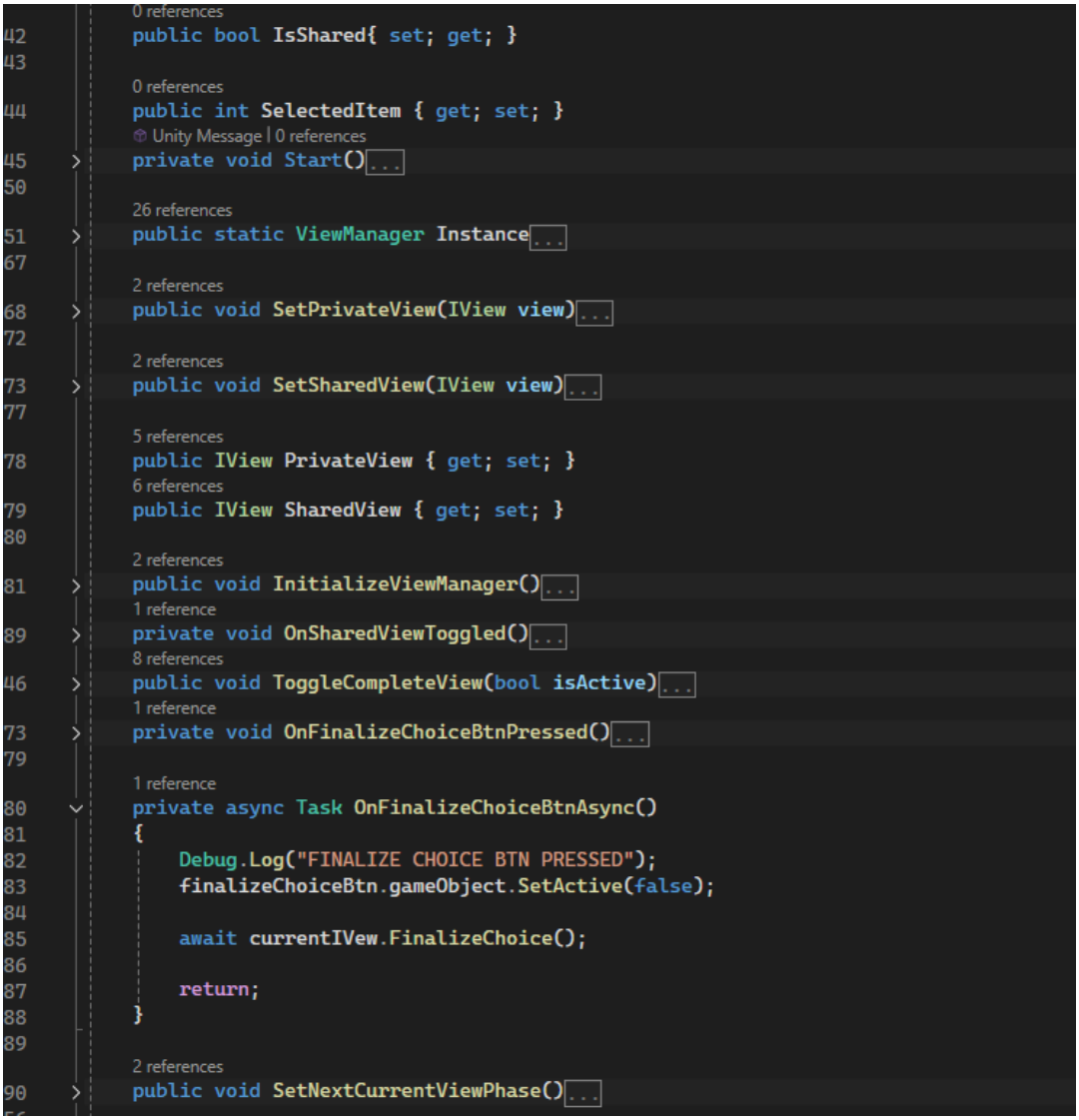
The `ViewManager` 5.37 5.38 script acts as a central controller for managing shared and private views. It maintains references to both `SharedView` and `PrivateView`, facilitating seamless toggling between the two modes. The script coordinates view state transitions, such as entering or exiting compare mode. It also integrates with the `ViewUIController` to update user interface elements dynamically based on the current view.

```
Unity Script (2 asset references) | 30 references
public class ViewManager : MonoBehaviour
{

    private static ViewManager _instance;

    [SerializeField] public IView sharedView;
    [SerializeField] public IView privateView;

    [SerializeField] public ViewUIController uiController;

    [SerializeField] public ProgressIndicator progressIndicator;

    [SerializeField] public PressableButton finalizeChoiceBtn;

    [SerializeField] private PressableButton sharedViewToggle;
    [SerializeField] private TMP_Text sharedViewText;

    [SerializeField] private TMP_Text P1_text;

    [SerializeField] private TMP_Text P2_text;

    [SerializeField] private GameObject privateSceneHelper;
    [SerializeField] private GameObject sharedSceneHelper;
    [SerializeField] private GameObject simpleSceneHelper;

    public IView currentIVew;

    private int selectedItem;
    private bool isShared = false;

    private bool isPrivateComplete = false;
    private bool isSharedComplete = false;

```

FIGURE 5.37: ViewManager Script

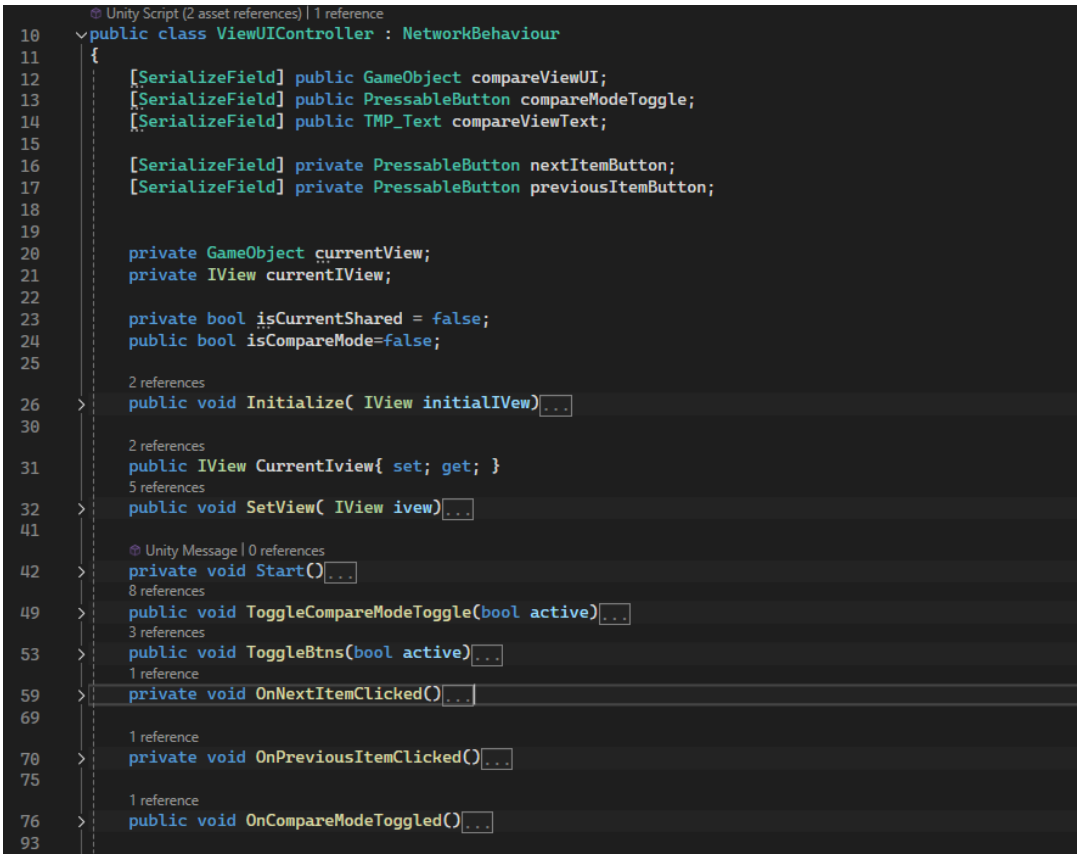
The `ViewUIController` Script

The `ViewUIController` 5.39 script manages user interactions with the customization interface. It handles button clicks for navigating items (`NextItem`, `PreviousItem`) and toggling compare mode (`CompareViewConvert`). By dynamically enabling or disabling interface elements, the script ensures a responsive user experience tailored to the active view and customization phase. It communicates with the `ViewManager` to synchronize UI states with view transitions.



```
0 references
public bool IsShared{ set; get; }

0 references
public int SelectedItem { get; set; }
Unity Message | 0 references
private void Start()...

26 references
public static ViewManager Instance...

2 references
public void SetPrivateView(IView view)...

2 references
public void SetSharedView(IView view)...

5 references
public IView PrivateView { get; set; }
6 references
public IView SharedView { get; set; }

2 references
public void InitializeViewManager()...
1 reference
private void OnSharedViewToggled()...
8 references
public void ToggleCompleteView(bool isActive)...
1 reference
private void OnFinalizeChoiceBtnPressed()...

1 reference
private async Task OnFinalizeChoiceBtnAsync()
{
    Debug.Log("FINALIZE CHOICE BTN PRESSED");
    finalizeChoiceBtn.gameObject.SetActive(false);

    await currentIVew.FinalizeChoice();

    return;
}

2 references
public void SetNextCurrentViewPhase()...
```

FIGURE 5.38: ViewManager continuation Script



```
Unity Script (2 asset references) | 1 reference
public class ViewUIController : NetworkBehaviour
{
    [SerializeField] public GameObject compareViewUI;
    [SerializeField] public PressableButton compareModeToggle;
    [SerializeField] public TMP_Text compareViewText;

    [SerializeField] private PressableButton nextItemButton;
    [SerializeField] private PressableButton previousItemButton;


    private GameObject currentView;
    private IView currentIView;

    private bool isCurrentShared = false;
    public bool isCompareMode=false;

    2 references
    public void Initialize( IView initialIVew)...

    2 references
    public IView CurrentIview{ set; get; }
    5 references
    public void SetView( IView ivew)...

    Unity Message | 0 references
    private void Start()...
    8 references
    public void ToggleCompareModeToggle(bool active)...
    3 references
    public void ToggleBtns(bool active)...
    1 reference
    private void OnNextItemClicked()...

    1 reference
    private void OnPreviousItemClicked()...

    1 reference
    public void OnCompareModeToggled()...
```

FIGURE 5.39: View UI Controller Script

The `ProgressIndicator` Script

The `ProgressIndicator` script provides visual feedback on the completion status of shared and private views. By toggling progress indicators for each mode, it informs users about their progress in the customization process. The script interacts with the `ViewManager` and `CustomizeManager` to update completion states based on user actions.

The `CustomizeManager` Script

The `CustomizeManager` script orchestrates the entire customization workflow, managing both shared and private layouts. It tracks user choices for layouts and configurations, handling transitions between customization. When both apartment layouts are finalized the user moves on to the next phase of the design.

In this phase of the design, users are tasked with placing rooms of their choice within designated room containers and submitting their finished designs 5.42, 5.43, 5.44. The implementation leverages several classes, including `LayoutManager` and `RoomContainer`, which work in tandem to facilitate room placement, layout management, and collaborative synchronization. These components ensure that the design phase is intuitive, flexible, and capable of supporting both independent and collaborative workflows.

The `LayoutManager` Class

The `LayoutManager` class is a core component responsible for managing the room placement and layout customization process. It provides functionalities for spawning, transforming, and managing rooms within a given layout. Additionally, the class handles the following key responsibilities:

- **Room Placement:** Users can select and place rooms using the **SpawnRoom** method. The method ensures that rooms are positioned without overlaps by checking for collisions or proximity to existing rooms. Overlapping rooms are automatically removed to maintain layout integrity.
- **Menu Management:** The **OpenMenu** method dynamically activates the appropriate menu for single or double rooms based on user input. The menu's position and rotation are adjusted relative to the selected room container.
- **Layout Visualization:** The class supports displaying the overall layout through the **DisplayLayoutModelServerRPC** method, which instantiates and scales layout objects.
- **Room Synchronization:** For collaborative scenarios, the class employs RPCs to propagate room placement and updates across clients. Shared clients receive real-time updates on room additions or removals.
- **State Management:** Rooms are tracked using lists of `RoomData` and `GameObject` instances, allowing efficient respawning, despawning, and saving of room configurations.
- **Finalization:** The **FinalizeCurrentLayout** method enables users to submit their completed designs. Room data is saved asynchronously, supporting both shared and private customization workflows.

The `RoomContainer` Class

The `RoomContainer` class facilitates user interaction with specific containers within the layout. It provides a mechanism for selecting spawn positions and opening the appropriate room placement menu. Key functionalities include:

- **Menu Activation:** When a user interacts with a room container, the **onRoomContainerBtnPressed** method triggers the **OpenMenu** function of the `LayoutManager`. Parameters such as the spawn position, rotation, and whether the room is a single or double are passed to configure the menu.
- **Spawn Position Management:** The class maintains a reference to the spawn position (`spawnPos`) and rotation, ensuring accurate placement of rooms within the container.

The collaboration between `LayoutManager` and `RoomContainer` creates a streamlined and user-friendly experience for room placement and layout customization. During the design phase, users interact with `RoomContainer` instances to initiate the placement workflow, while the `LayoutManager` handles spawning and managing rooms 5.40 . For shared layouts, room placement is synchronized across clients in real-time using Unity Netcode, ensuring consistency and collaboration. The system automatically resolves conflicts by removing overlapping rooms, maintaining a clean and organized layout 5.41 . Additionally, the `LayoutManager` enables users to visualize

their designs and finalize the layout with ease, ensuring an efficient and intuitive design process.

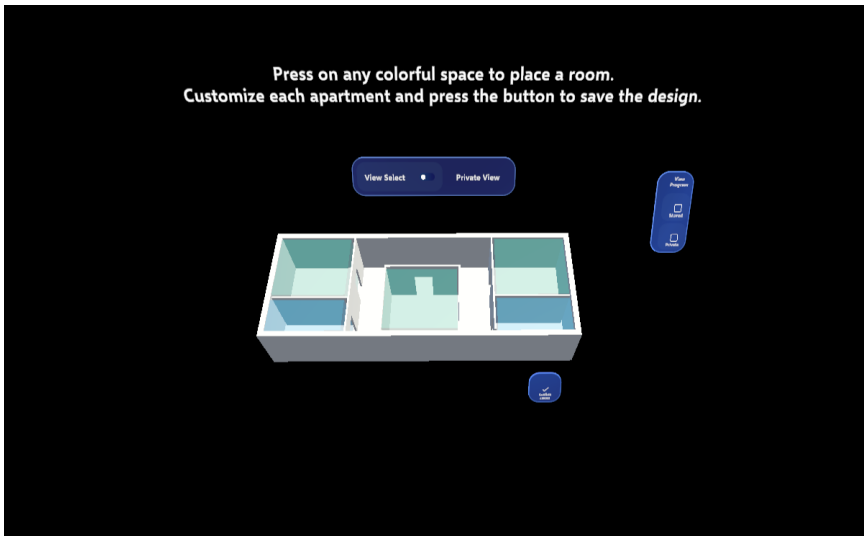


FIGURE 5.40: Customize P2 UI

FIGURE 5.41: Customize P2 UI++

5.3.14 Visualize Interface

The *Vizualize Interface* 5.45, 5.46, 5.47 is used to show and rate finished designs. The user can select the lobby of any of the created Designs to visualize the completed apartments of this design. They then can rate all available apartment designs and submit their ratings.

The `Visualize Dwelling Selection` Class

This class works like the `Customize Dwelling Selector` class. It provides an interface where the user can choose which design the would like to rate. After the user makes their choice they are prompted to the Rating User Interface.

FIGURE 5.42: Customize Phase 2 at runtime

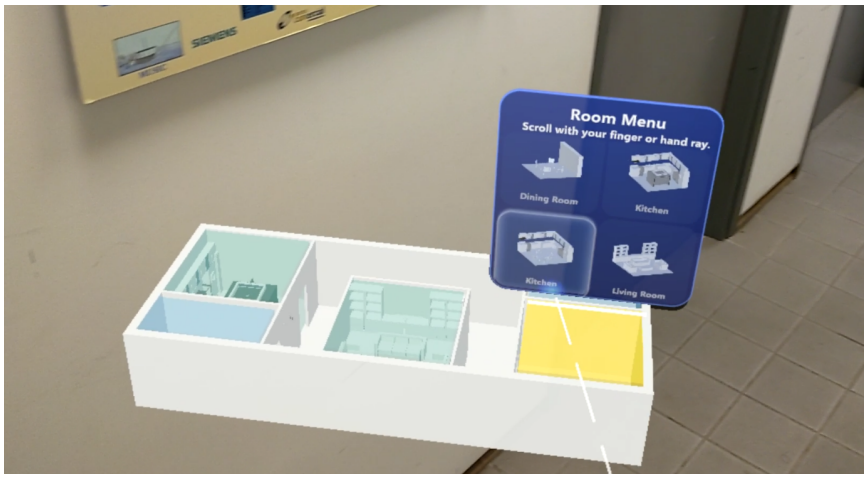


FIGURE 5.43: Customize Phase 2 at runtime++

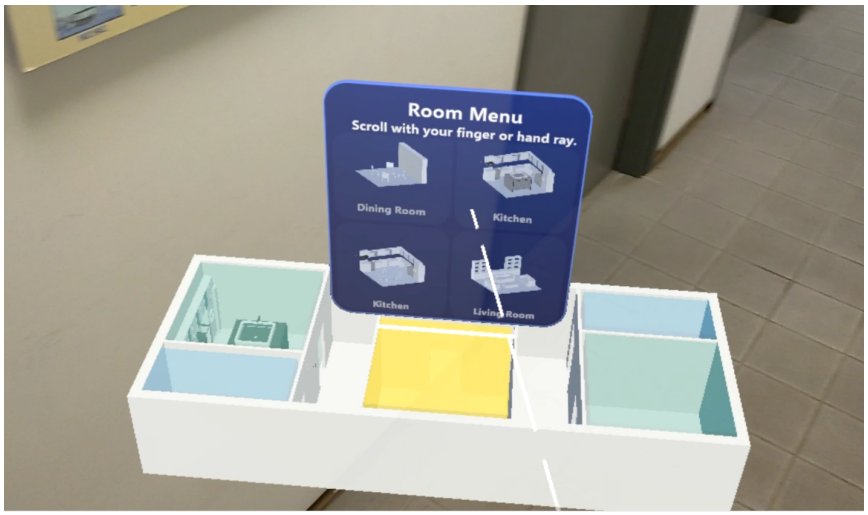


FIGURE 5.44: Customize Phase 2 Private View

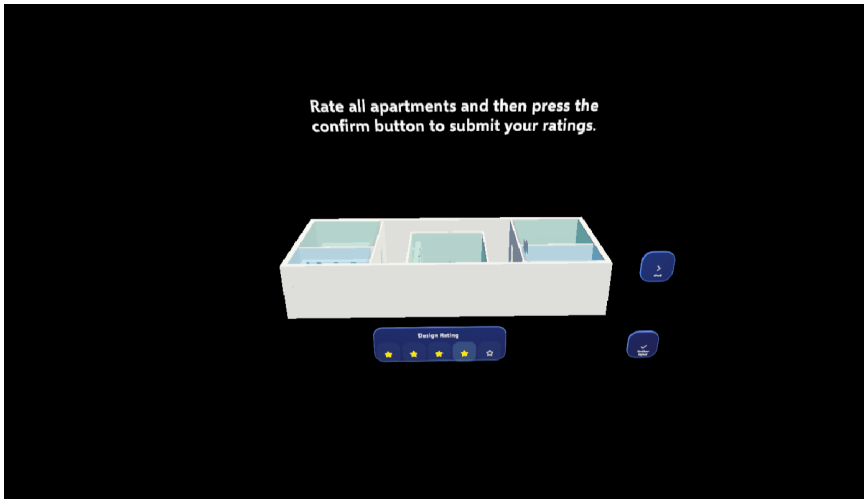


FIGURE 5.45: Rating UI

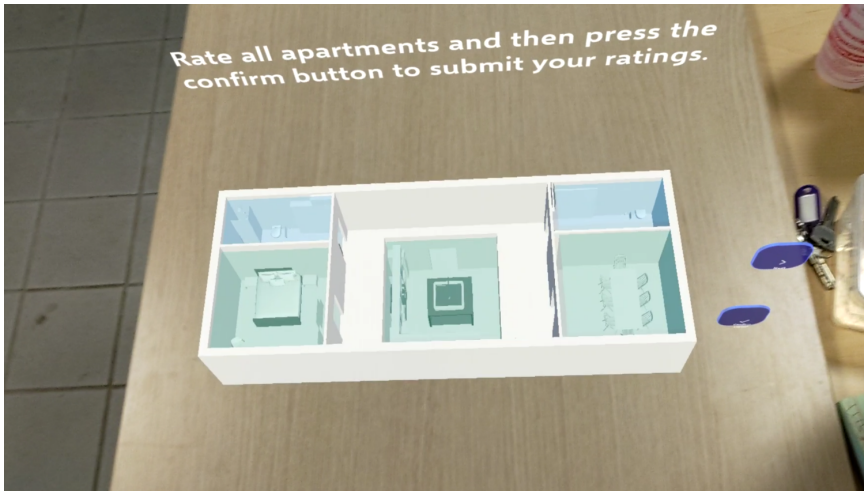


FIGURE 5.46: Rating UI at runtime

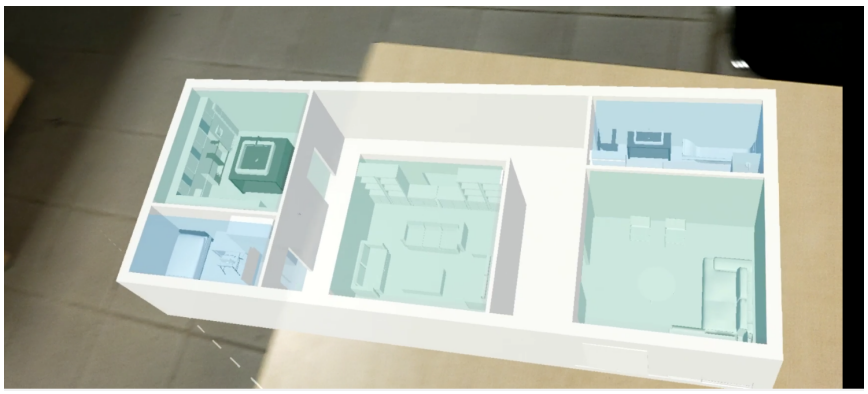


FIGURE 5.47: Rating Finished Apartment

The `RatingIndicator` Class

The `RatingIndicator` class is responsible for managing the user interface element that allows users to rate designs. It controls the visual representation of the rating system and ensures that user selections are properly registered and communicated. The class primarily handles the following functionalities:

- **Displaying Rating Selection:** The class manages an array of button images (`rateBtnImages`) representing the rating levels. When a user selects a rating, the appropriate number of rating icons are activated while the remaining ones are deactivated.
- **Handling User Input:** The **Rate(int i)** function is used to update the displayed rating based on the user's selection.
- **Storing Current Rating:** The class maintains an internal variable (`currrRating`) to keep track of the most recent rating value.
- **Communicating with the Visualization Manager:** Once a rating is selected, it is forwarded to the `VisualizeManager` via the **RateCurrentDesign()** method, ensuring that the selected rating is stored and processed within the system.

The `View` Class

The `View` class is responsible for managing the visualization of different designs. It provides functionalities for navigating through different design options, displaying the currently selected design, handling user ratings, and finalizing choices. The class primarily manages the following features:

- **Managing the Design List:** The class stores a list of available designs and keeps track of the currently selected index (`currentIndex`).
- **Navigation through Designs:** The **NextItem()** and **PreviousItem()** methods allow the user to navigate through available designs, updating the display accordingly.
- **Displaying the Current Design:** The **ShowCurrentItem()** method instantiates the currently selected design within the designated layout container, ensuring that only one design is displayed at a time.
- **Managing UI Controls:** The class interacts with the `VisualizeManager` to toggle navigation buttons appropriately using the **ToggleNextPrevBtns()** function.
- **Rating System:** The **RateCurrentDesign(int rating)** method allows users to rate the current design, storing the rating within the `VisualizeManager` for later submission.
- **Finalizing User Selection:** The **FinalizeChoice()** method prompts the user with a dialog box to confirm the submission of ratings. If accepted, the ratings are submitted via the `VisualizeManager`.

The `VisualizeViewUIController` Class

The `VisualizeViewUIController` class manages the user interface for design visualization in augmented reality. It provides controls for navigating between design items, rating them, and confirming user choices.

Key Responsibilities:

- **Navigation:** Enables users to move between design options using 'Next' and 'Previous' buttons.
- **Rating System:** Integrates with the `RatingIndicator` to allow user feedback.
- **Design Selection:** Confirms the final design choice and updates the UI accordingly.

Core Functionalities:

- **SetView()**: Displays the current design.
- **OnNextItemClicked()**, `OnPreviousItemClicked()`: Handles navigation.
- **OnConfirmButtonClicked()**: Finalizes the design selection.
- **ToggleNextBtn()**, **TogglePreviousBtn()**, **ConfirmBtnToggle()**: Manage button visibility.

This class ensures smooth user interaction, enabling effective design evaluation and selection in the AR environment.

The `VisualizeManager` Class

The `VisualizeManager` class coordinates the visualization and evaluation of architectural layouts in an augmented reality environment. It manages room spawning, rating, and user interaction.

Key Responsibilities:

- **Room Management:** Loads, spawns, and despawns room layouts from saved data.
- **User Interaction:** Interfaces with `View` and `VisualizeViewUIController` for navigation and rating.
- **Session Handling:** Initializes layouts based on selected modules and stores user feedback.

Core Functionalities:

- **SetupLayoutInterfaces()**: Loads room data and initializes the visualization UI.
- **RespawnAllRooms()**, **DespawnAllRooms()**: Manages room instantiation and removal.
- **OffsetRooms()**: Adjusts room positioning for comparison.
- **SubmitRatings()**: Saves and submits user ratings.

This class ensures smooth visualization and evaluation of design choices, supporting user-driven decision-making in AR.

5.4 User Authentication

5.4.1 Unity Authentication Service

Unity Authentication provides a secure and streamlined way to manage user accounts and authentication. In the application, Unity Authentication is leveraged to

authenticate users anonymously. This allows users to participate in collaborative sessions without the need for complex login procedures, enhancing the user experience and ease of access.

```
1 reference
public async Task Authenticate(string playerName)
{
    Debug.Log("Setting profile name: " + playerName);

    InitializationOptions initializationOptions = new InitializationOptions();
    initializationOptions.SetProfile(playerName);

    await UnityServices.InitializeAsync(initializationOptions);

    AuthenticationService.Instance.SignedIn += () =>
    {
        Debug.Log("Signed in! Player ID: " + AuthenticationService.Instance.PlayerId);
        RefreshLobbyList(sessionMode);
    };

    try
    {
        await AuthenticationService.Instance.SignInAnonymouslyAsync();
    }
    catch (AuthenticationException e)
    {
        Debug.LogError("Authentication failed: " + e.Message);
    }
}
```

FIGURE 5.48: The Authenticate method in the LobbyManager.

The authentication process is handled by the `LobbyManager` script 5.48. Upon application startup, the `LobbyManager` script interacts with the Unity Authentication service to sign in users anonymously. This generates a unique player identifier for each user, which is then used to represent them within the lobby system and the collaborative AR environment.

5.5 Networking and Collaboration

5.5.1 The Lobby System

The `LobbyManager` script acts as the orchestrator of the lobby system. It interacts with the **Unity Lobby service**, a cloud-based backend service that provides functionalities for creating, managing, and joining multiplayer lobbies. The `LobbyManager` handles the following key tasks:

```
public async void CreateLobby(string lobbyName, int maxPlayers, bool isPrivate, SessionMode sessionMode) {
    lobbyCreated=false ;
    Player player = GetPlayer();
    //Debug.Log("Lobby created with sessionm0de:" + sessionMode);
    CreateLobbyOptions options = new CreateLobbyOptions {
        Player = player,
        IsPrivate = isPrivate,
        Data = new Dictionary<string, DataObject> {
            { KEY_SESSION_MODE, new DataObject(DataObject.VisibilityOptions.Public, sessionMode.ToString(), DataObject.IndexOptions.S1) },
            { KEY_START_SESSION, new DataObject(DataObject.VisibilityOptions.Member, "0" ) }

        }
    };

    Lobby lobby = await LobbyService.Instance.CreateLobbyAsync(lobbyName, maxPlayers, options);

    joinedLobby = lobby;

    OnJoinedLobby?.Invoke(this, new LobbyEventArgs { lobby = lobby });

    Debug.Log("Created Lobby " + lobby.Name);

    lobbyCreated = true;
}
```

FIGURE 5.49: The Create Lobby method in the LobbyManager.

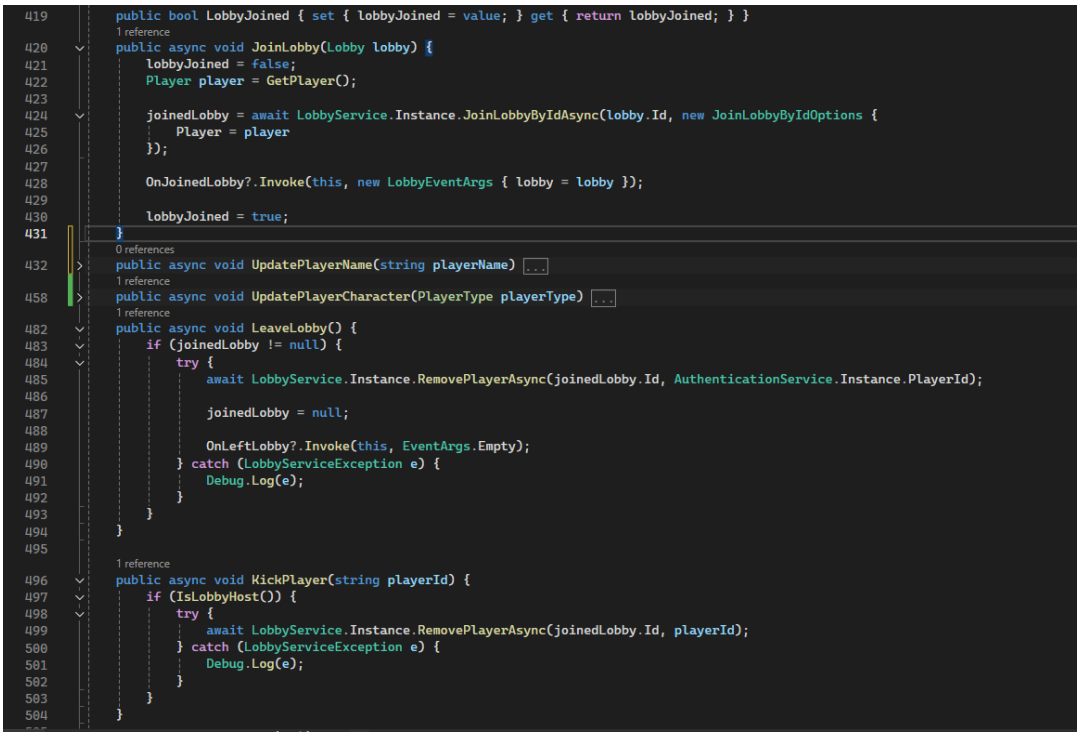
- **Lobby Creation:** 5.49 Allows users to create new lobbies, defining parameters such as lobby name, maximum player count, and the chosen session mode

```
//function that keeps lobby alive by sending a heartbeat every 15s
1 reference
private async void HandleLobbyHeartbeat() {
    if (IsLobbyHost()) {
        heartbeatTimer -= Time.deltaTime;
        if (heartbeatTimer < 0f) {
            float heartbeatTimerMax = 15f;
            heartbeatTimer = heartbeatTimerMax;

            Debug.Log("Heartbeat");
            await LobbyService.Instance.SendHeartbeatPingAsync(joinedLobby.Id);
        }
    }
}
```

FIGURE 5.50: The handlelobbyHeartbeat method in the LobbyManager.

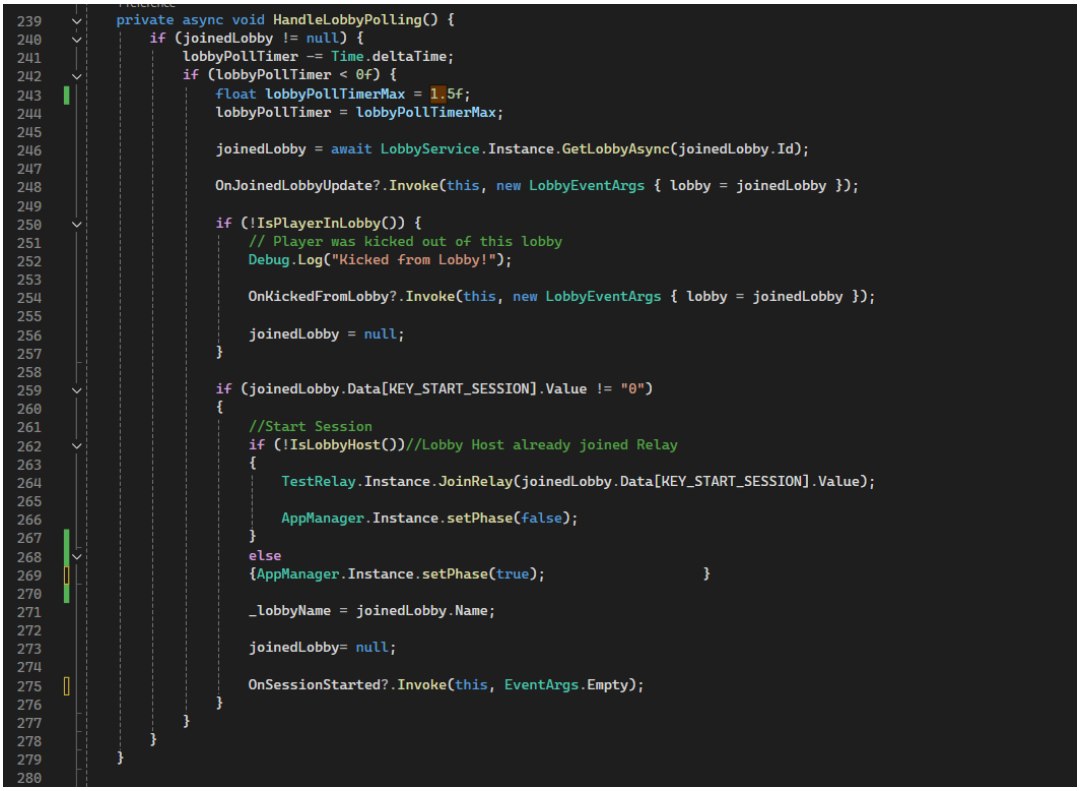
```
372     public async void RefreshLobbyList(SessionMode? targetSessionMode = null) {
373
374         //Debug.Log("Session mode on Refresh Lobby List"+targetSessionMode);
375         try {
376             QueryLobbiesOptions options = new QueryLobbiesOptions();
377             options.Count = 25;
378
379             // Filter for open lobbies only
380             List<QueryFilter> filters = new List<QueryFilter>
381             {
382                 new QueryFilter(
383                     field: QueryFilter.FieldOptions.AvailableSlots,
384                     op: QueryFilter.OpOptions.GT,
385                     value: "0"),
386             };
387
388             if (targetSessionMode != null)
389             {
390                 //Debug.Log("target Session mode is: "+targetSessionMode);
391                 filters.Add(new QueryFilter(
392                     field: QueryFilter.FieldOptions.S1,
393                     op: QueryFilter.OpOptions.EQ,
394                     value:targetSessionMode.Value.ToString()
395                 ));
396             }
397         options.Filters = filters;
398
399             // Order by newest lobbies first
400             options.Order = new List<QueryOrder> {
401                 new QueryOrder(
402                     asc: false,
403                     field: QueryOrder.FieldOptions.Created)
404             };
405
406             QueryResponse lobbyListQueryResponse = await Lobbies.Instance.QueryLobbiesAsync(options);
407
408
409             OnLobbyListChanged?.Invoke(this, new OnLobbyListChangedEventArgs { lobbyList = lobbyListQueryResponse.Results });
410
411         } catch (LobbyServiceException e) {
412             Debug.Log(e);
413         }
```

FIGURE 5.51: The Refresh Lobby List method in the LobbyManager.



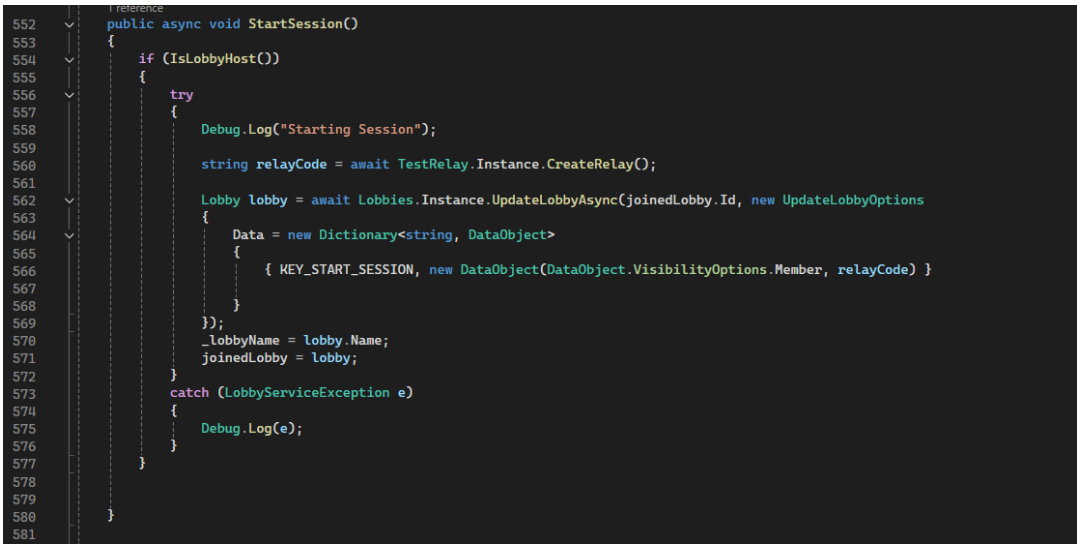
```
419     public bool LobbyJoined { set { lobbyJoined = value; } get { return lobbyJoined; } }
        1 reference
420     public async void JoinLobby(Lobby lobby) {
421         lobbyJoined = false;
422         Player player = GetPlayer();
423
424         joinedLobby = await LobbyService.Instance.JoinLobbyByIdAsync(lobby.Id, new JoinLobbyByIdOptions {
425             Player = player
426         });
427
428         OnJoinedLobby?.Invoke(this, new LobbyEventArgs { lobby = lobby });
429
430         lobbyJoined = true;
431     }
        0 references
432     public async void UpdatePlayerName(string playerName) ...
        1 reference
458     public async void UpdatePlayerCharacter(PlayerType playerType) ...
        1 reference
482     public async void LeaveLobby() {
483         if (joinedLobby != null) {
484             try {
485                 await LobbyService.Instance.RemovePlayerAsync(joinedLobby.Id, AuthenticationService.Instance.PlayerId);
486
487                 joinedLobby = null;
488
489                 OnLeftLobby?.Invoke(this, EventArgs.Empty);
490             } catch (LobbyServiceException e) {
491                 Debug.Log(e);
492             }
493         }
494     }
495
        1 reference
496     public async void KickPlayer(string playerId) {
497         if (IsLobbyHost()) {
498             try {
499                 await LobbyService.Instance.RemovePlayerAsync(joinedLobby.Id, playerId);
500             } catch (LobbyServiceException e) {
501                 Debug.Log(e);
502             }
503         }
504     }
```

FIGURE 5.52: The Join/Leave/Kick functions in LobbyManager.



```
239     private async void HandleLobbyPolling() {
240         if (joinedLobby != null) {
241             lobbyPollTimer -= Time.deltaTime;
242             if (lobbyPollTimer < 0f) {
243                 float lobbyPollTimerMax = 1.5f;
244                 lobbyPollTimer = lobbyPollTimerMax;
245
246                 joinedLobby = await LobbyService.Instance.GetLobbyAsync(joinedLobby.Id);
247
248                 OnJoinedLobbyUpdate?.Invoke(this, new LobbyEventArgs { lobby = joinedLobby });
249
250                 if (!IsPlayerInLobby()) {
251                     // Player was kicked out of this lobby
252                     Debug.Log("Kicked from Lobby!");
253
254                     OnKickedFromLobby?.Invoke(this, new LobbyEventArgs { lobby = joinedLobby });
255
256                     joinedLobby = null;
257                 }
258
259                 if (joinedLobby.Data[KEY_START_SESSION].Value != "0")
260                 {
261                     //Start Session
262                     if (!IsLobbyHost())//Lobby Host already joined Relay
263                     {
264                         TestRelay.Instance.JoinRelay(joinedLobby.Data[KEY_START_SESSION].Value);
265
266                         AppManager.Instance.setPhase(false);
267                     }
268                     else
269                     {AppManager.Instance.setPhase(true);                    }
270
271                     _lobbyName = joinedLobby.Name;
272
273                     joinedLobby= null;
274
275                     OnSessionStarted?.Invoke(this, EventArgs.Empty);
276                 }
277             }
278         }
279     }
280
```

FIGURE 5.53: The handleLobbyPolling function in LobbyManager.



```
1 reference
public async void StartSession()
{
    if (IsLobbyHost())
    {
        try
        {
            Debug.Log("Starting Session");

            string relayCode = await TestRelay.Instance.CreateRelay();

            Lobby lobby = await Lobbies.Instance.UpdateLobbyAsync(joinedLobby.Id, new UpdateLobbyOptions
            {
                Data = new Dictionary<string, DataObject>
                {
                    { KEY_START_SESSION, new DataObject(DataObject.VisibilityOptions.Member, relayCode) }

                }
            });
            _lobbyName = lobby.Name;
            joinedLobby = lobby;
        }
        catch (LobbyServiceException e)
        {
            Debug.Log(e);
        }
    }


}
```

FIGURE 5.54: The startSession function in LobbyManager.

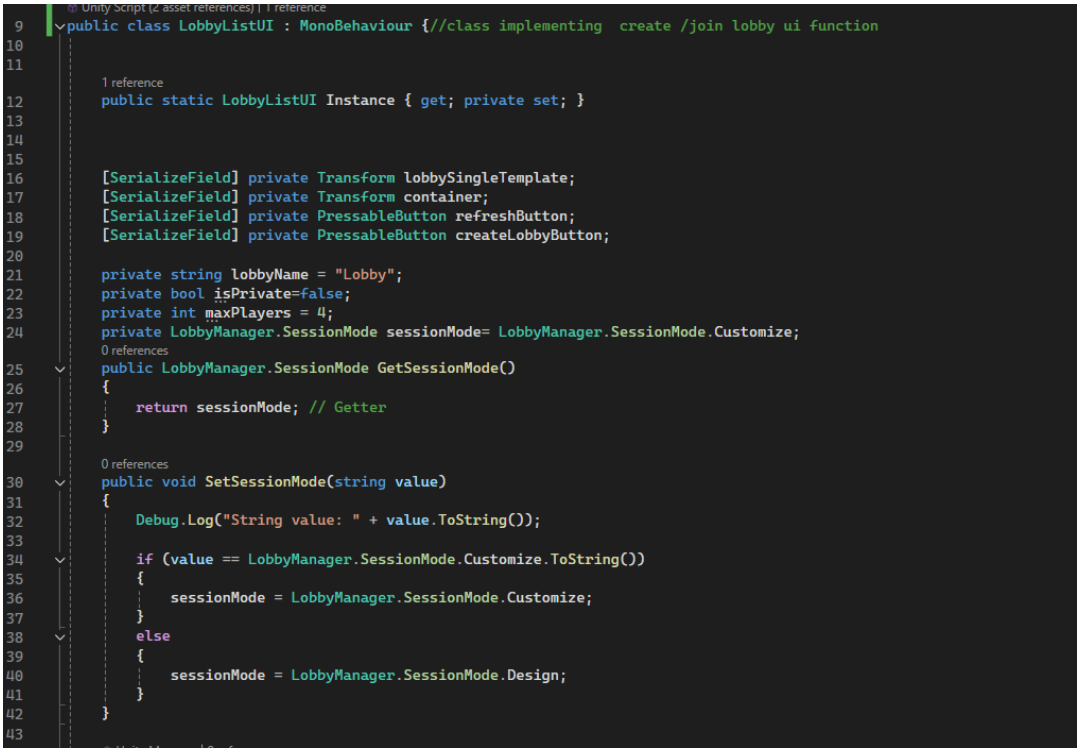
(e.g., Design or Customize). It also sends a heartbeat to the unity lobby services to keep the created Lobby alive 5.50.

- **Lobby Listing:** 5.51 Retrieves and displays a list of available lobbies from the Unity Lobby service, allowing users to browse and select a lobby to join.
- **Player Management:** 5.53 Manages player data within a lobby, including player names, unique identifiers, and player types (e.g., Expert or Non-Expert).
- **Lobby Updates:** 5.52 Handles real-time updates to the lobby state, such as players joining or leaving, changes in lobby data, and notifications when the host starts a game session.
- **Session Start:** 5.54 Initiates the start of a game session when the host triggers the action, transitioning all connected players into the shared AR environment using the **Unity Relay service**.

The **LobbyListUI** script 5.55 5.56 5.57 is responsible for the visual presentation of the lobby list interface. It displays the list of available lobbies retrieved by the 'LobbyManager', providing users with an overview of each lobby and allowing them to choose one to join. The key elements of the 'LobbyListUI' include:

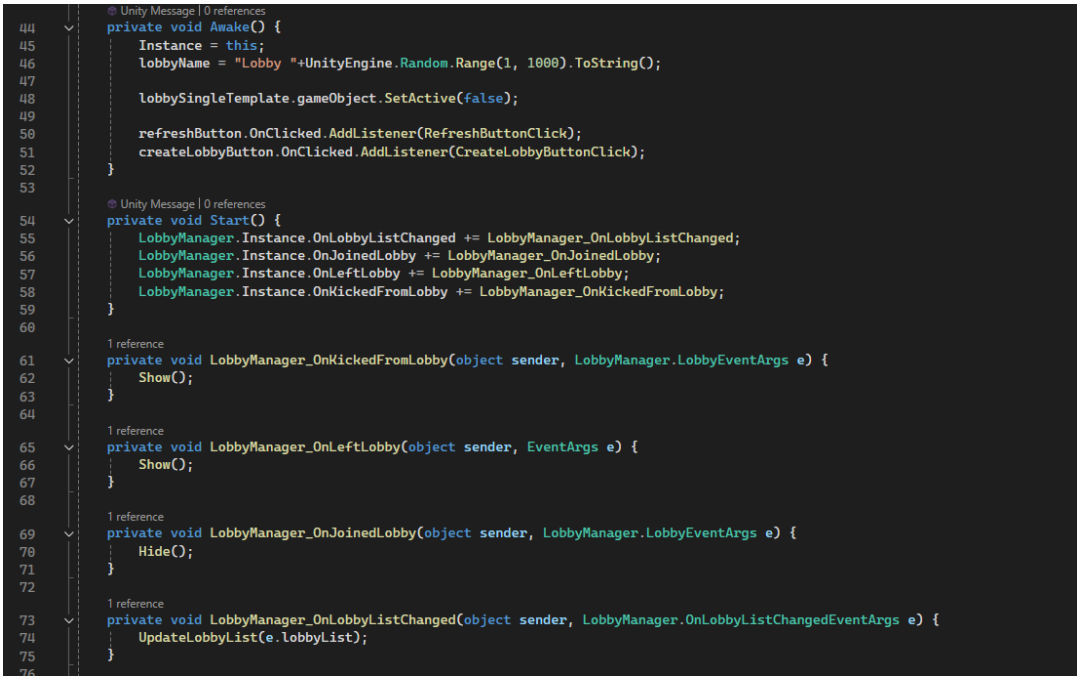
- **Lobby Information:** Displays essential information about each lobby, such as its name, current player count, and the selected session mode.
- **Refresh Functionality:** Provides a button to refresh the lobby list, ensuring users see the most up-to-date information about available lobbies.
- **Lobby Creation:** Includes a button to initiate the creation of a new lobby, prompting the user for necessary configuration details.
- **Join Lobby:** Allows users to select a lobby from the list and join it, triggering the LobbyManager to handle the connection process.

Once a user joins a lobby, the `LobbyUI` 5.58 5.59 script takes over to display the details of the joined lobby. It provides a more detailed view of the lobby and its participants, including:



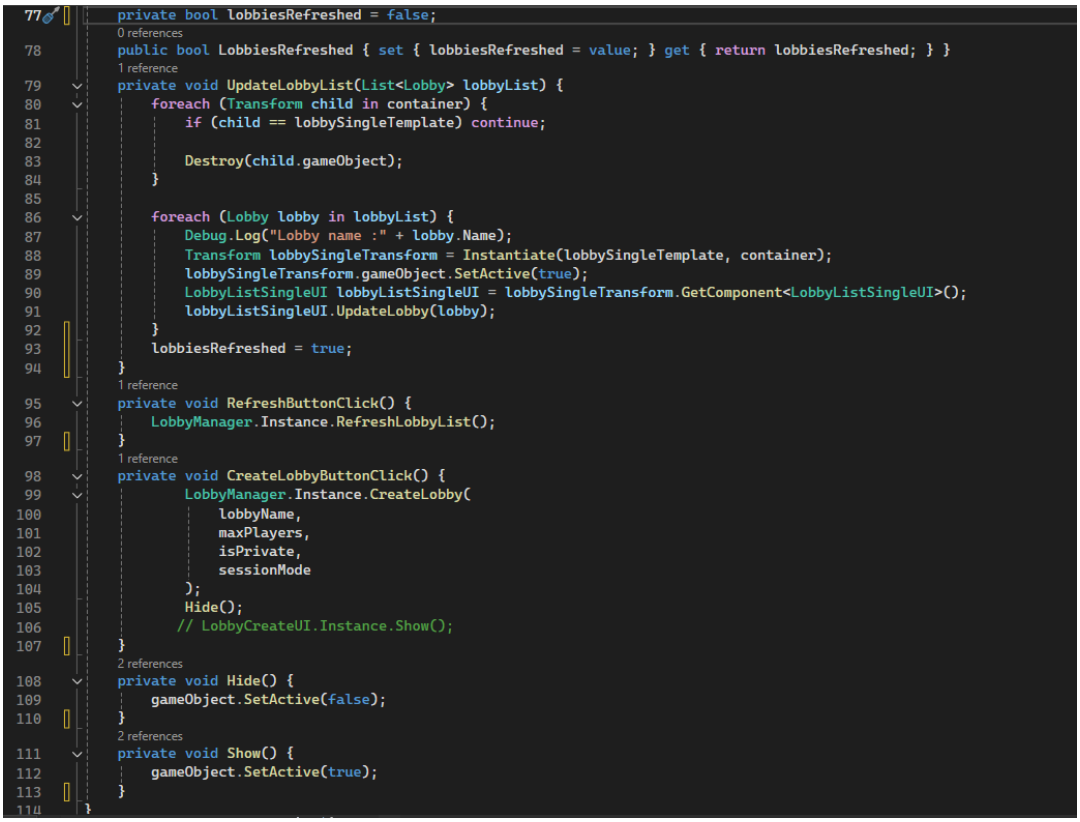
```
Unity Script (2 asset references) | 1 reference
public class LobbyListUI : MonoBehaviour {//class implementing  create /join lobby ui function


    1 reference
    public static LobbyListUI Instance { get; private set; }



    [SerializeField] private Transform lobbySingleTemplate;
    [SerializeField] private Transform container;
    [SerializeField] private PressableButton refreshButton;
    [SerializeField] private PressableButton createLobbyButton;

    private string lobbyName = "Lobby";
    private bool isPrivate=false;
    private int maxPlayers = 4;
    private LobbyManager.SessionMode sessionMode= LobbyManager.SessionMode.Customize;
    0 references
    public LobbyManager.SessionMode GetSessionMode()
    {
        return sessionMode; // Getter
    }

    0 references
    public void SetSessionMode(string value)
    {
        Debug.Log("String value: " + value.ToString());

        if (value == LobbyManager.SessionMode.Customize.ToString())
        {
            sessionMode = LobbyManager.SessionMode.Customize;
        }
        else
        {
            sessionMode = LobbyManager.SessionMode.Design;
        }
    }

```

FIGURE 5.55: The LobbyListUI script .



```
Unity Message | 0 references
private void Awake() {
    Instance = this;
    lobbyName = "Lobby "+UnityEngine.Random.Range(1, 1000).ToString();

    lobbySingleTemplate.gameObject.SetActive(false);

    refreshButton.OnClicked.AddListener(RefreshButtonClick);
    createLobbyButton.OnClicked.AddListener(CreateLobbyButtonClick);
}

Unity Message | 0 references
private void Start() {
    LobbyManager.Instance.OnLobbyListChanged += LobbyManager_OnLobbyListChanged;
    LobbyManager.Instance.OnJoinedLobby += LobbyManager_OnJoinedLobby;
    LobbyManager.Instance.OnLeftLobby += LobbyManager_OnLeftLobby;
    LobbyManager.Instance.OnKickedFromLobby += LobbyManager_OnKickedFromLobby;
}

1 reference
private void LobbyManager_OnKickedFromLobby(object sender, LobbyManager.LobbyEventArgs e) {
    Show();
}

1 reference
private void LobbyManager_OnLeftLobby(object sender, EventArgs e) {
    Show();
}

1 reference
private void LobbyManager_OnJoinedLobby(object sender, LobbyManager.LobbyEventArgs e) {
    Hide();
}

1 reference
private void LobbyManager_OnLobbyListChanged(object sender, LobbyManager.OnLobbyListChangedEventArgs e) {
    UpdateLobbyList(e.lobbyList);
}
```

FIGURE 5.56: The LobbyListUI script +.



```
77    private bool lobbiesRefreshed = false;
      0 references
78    public bool LobbiesRefreshed { set { lobbiesRefreshed = value; } get { return lobbiesRefreshed; } }
      1 reference
79    private void UpdateLobbyList(List<Lobby> lobbyList) {
80        foreach (Transform child in container) {
81            if (child == lobbySingleTemplate) continue;
82
83            Destroy(child.gameObject);
84        }
85
86        foreach (Lobby lobby in lobbyList) {
87            Debug.Log("Lobby name :" + lobby.Name);
88            Transform lobbySingleTransform = Instantiate(lobbySingleTemplate, container);
89            lobbySingleTransform.gameObject.SetActive(true);
90            LobbyListSingleUI lobbyListSingleUI = lobbySingleTransform.GetComponent<LobbyListSingleUI>();
91            lobbyListSingleUI.UpdateLobby(lobby);
92        }
93        lobbiesRefreshed = true;
94    }
      1 reference
95    private void RefreshButtonClick() {
96        LobbyManager.Instance.RefreshLobbyList();
97    }
      1 reference
98    private void CreateLobbyButtonClick() {
99            LobbyManager.Instance.CreateLobby(
100               lobbyName,
101               maxPlayers,
102               isPrivate,
103               sessionMode
104           );
105           Hide();
106          // LobbyCreateUI.Instance.Show();
107   }
      2 references
108   private void Hide() {
109       gameObject.SetActive(false);
110   }
      2 references
111   private void Show() {
112       gameObject.SetActive(true);
113   }
114
```

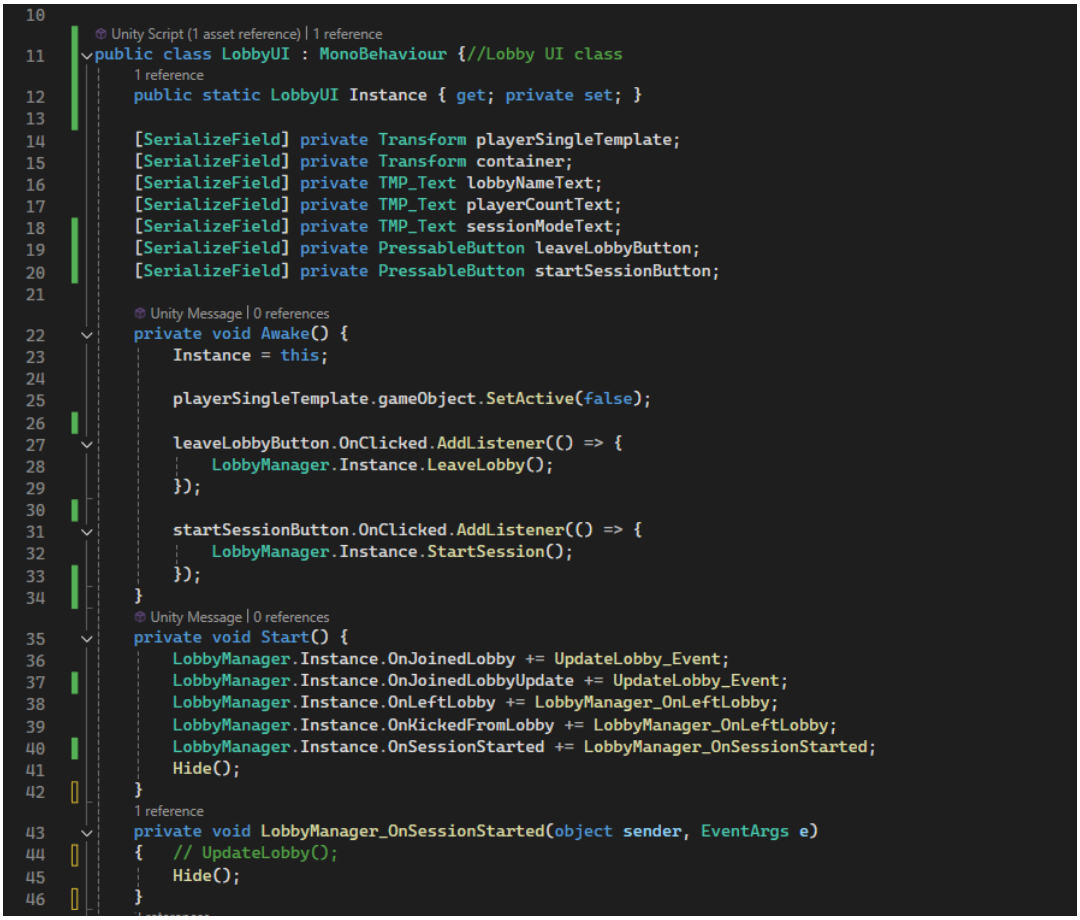
FIGURE 5.57: The LobbyListUI script ++.

- **Lobby Details:** Shows the name of the joined lobby, the current number of players, and the selected session mode for the upcoming AR experience.
- **Player List:** Displays a list of all connected players, showing their names and player types.
- **Host Controls:** If the local user is the host of the lobby, it provides controls for starting the game session using the **Unity Relay Service** and potentially managing players (e.g., kicking a player from the lobby).
- **Leave Lobby:** Includes a button for the user to leave the current lobby and return to the lobby list.

The `LobbyPlayerSingleUI` script 5.60 is a supplementary script that works in conjunction with the `LobbyUI` script. It is responsible for displaying the information of a single player within the lobby. It shows:

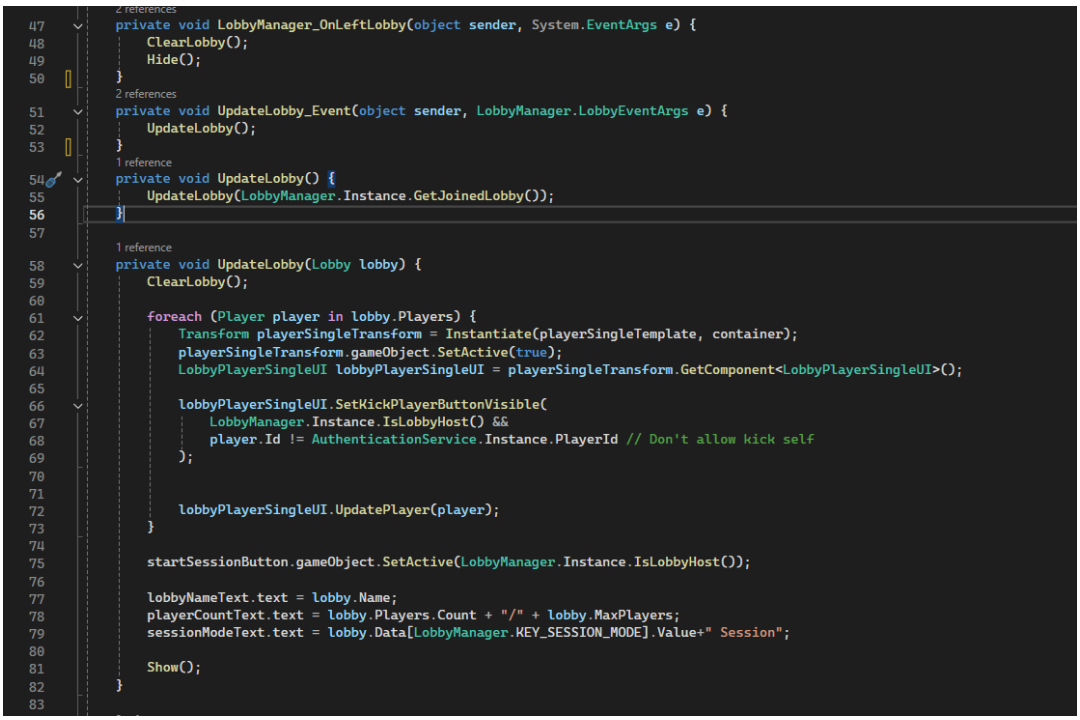
- **Player Name:** The display name of the player in the lobby.
- **Player Type:** The type of player (e.g., Expert or Non-Expert), providing context for their role in the collaborative AR session.
- **Player Type:** The type of player (e.g., Expert or Non-Expert), providing context for their role in the collaborative AR session.
- **Kick Player (Optional):** If the local user is the host, this script may include a button to kick the corresponding player from the lobby.

These four scripts work together to create a seamless and user-friendly lobby experience. The `LobbyManager` acts as the central coordinator, interacting with the Unity



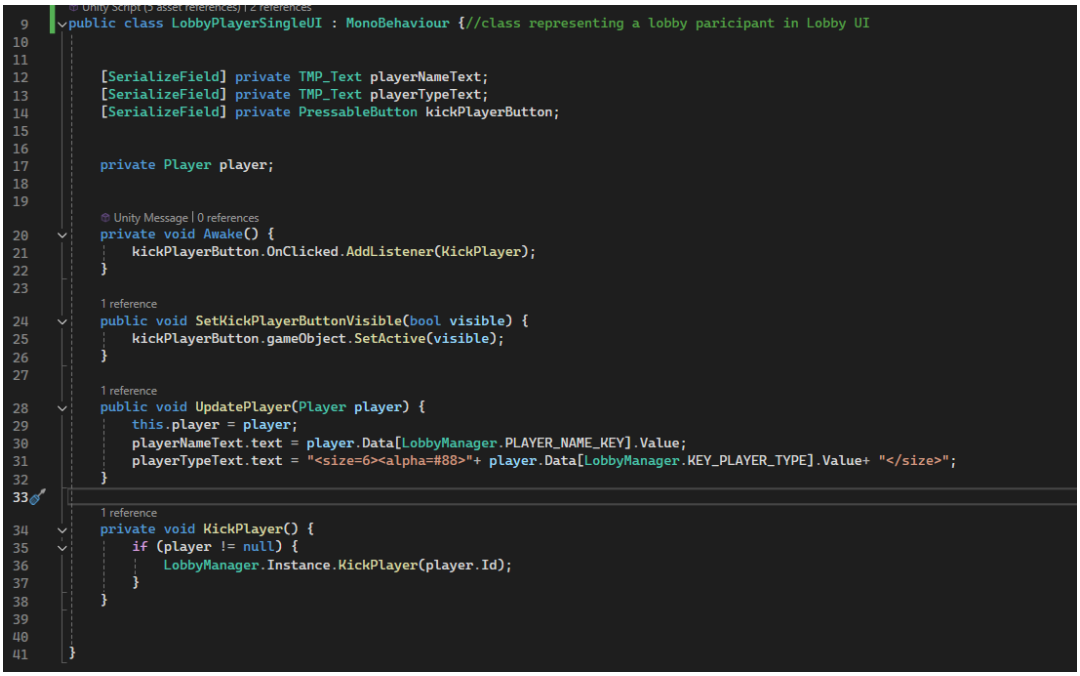
```
Unity Script (1 asset reference) | 1 reference
public class LobbyUI : MonoBehaviour {//Lobby UI class
    1 reference
    public static LobbyUI Instance { get; private set; }

    [SerializeField] private Transform playerSingleTemplate;
    [SerializeField] private Transform container;
    [SerializeField] private TMP_Text lobbyNameText;
    [SerializeField] private TMP_Text playerCountText;
    [SerializeField] private TMP_Text sessionModeText;
    [SerializeField] private PressableButton leaveLobbyButton;
    [SerializeField] private PressableButton startSessionButton;

    Unity Message | 0 references
    private void Awake() {
        Instance = this;

        playerSingleTemplate.gameObject.SetActive(false);

        leaveLobbyButton.OnClicked.AddListener(() => {
            LobbyManager.Instance.LeaveLobby();
        });

        startSessionButton.OnClicked.AddListener(() => {
            LobbyManager.Instance.StartSession();
        });
    }
    Unity Message | 0 references
    private void Start() {
        LobbyManager.Instance.OnJoinedLobby += UpdateLobby_Event;
        LobbyManager.Instance.OnJoinedLobbyUpdate += UpdateLobby_Event;
        LobbyManager.Instance.OnLeftLobby += LobbyManager_OnLeftLobby;
        LobbyManager.Instance.OnKickedFromLobby += LobbyManager_OnLeftLobby;
        LobbyManager.Instance.OnSessionStarted += LobbyManager_OnSessionStarted;
        Hide();
    }
    1 reference
    private void LobbyManager_OnSessionStarted(object sender, EventArgs e)
    {   // UpdateLobby();
        Hide();
    }
```

FIGURE 5.58: The Lobby UI script.



```
    2 references
    private void LobbyManager_OnLeftLobby(object sender, System.EventArgs e) {
        ClearLobby();
        Hide();
    }
    2 references
    private void UpdateLobby_Event(object sender, LobbyManager.LobbyEventArgs e) {
        UpdateLobby();
    }
    1 reference
    private void UpdateLobby() {
        UpdateLobby(LobbyManager.Instance.GetJoinedLobby());
    }

    1 reference
    private void UpdateLobby(Lobby lobby) {
        ClearLobby();

        foreach (Player player in lobby.Players) {
            Transform playerSingleTransform = Instantiate(playerSingleTemplate, container);
            playerSingleTransform.gameObject.SetActive(true);
            LobbyPlayerSingleUI lobbyPlayerSingleUI = playerSingleTransform.GetComponent<LobbyPlayerSingleUI>();

            lobbyPlayerSingleUI.SetKickPlayerButtonVisible(
                LobbyManager.Instance.IsLobbyHost() &&
                player.Id != AuthenticationService.Instance.PlayerId // Don't allow kick self
            );


            lobbyPlayerSingleUI.UpdatePlayer(player);
        }

        startSessionButton.gameObject.SetActive(LobbyManager.Instance.IsLobbyHost());

        lobbyNameText.text = lobby.Name;
        playerCountText.text = lobby.Players.Count + "/" + lobby.MaxPlayers;
        sessionModeText.text = lobby.Data[LobbyManager.KEY_SESSION_MODE].Value+" Session";

        Show();
    }
```

FIGURE 5.59: The Lobby UI script continuation.



```
public class LobbyPlayerSingleUI : MonoBehaviour {//class representing a lobby paricipant in Lobby UI


    [SerializeField] private TMP_Text playerNameText;
    [SerializeField] private TMP_Text playerTypeText;
    [SerializeField] private PressableButton kickPlayerButton;


    private Player player;


    Unity Message | 0 references
    private void Awake() {
        kickPlayerButton.OnClicked.AddListener(KickPlayer);
    }

    1 reference
    public void SetKickPlayerButtonVisible(bool visible) {
        kickPlayerButton.gameObject.SetActive(visible);
    }

    1 reference
    public void UpdatePlayer(Player player) {
        this.player = player;
        playerNameText.text = player.Data[LobbyManager.PLAYER_NAME_KEY].Value;
        playerTypeText.text = "<size=6><alpha=#88>"+ player.Data[LobbyManager.KEY_PLAYER_TYPE].Value+ "</size>";
    }

    1 reference
    private void KickPlayer() {
        if (player != null) {
            LobbyManager.Instance.KickPlayer(player.Id);
        }
    }


}
```

FIGURE 5.60: The LobbyPlayerSingleUI script.

Lobby service and managing the underlying logic and data. The `LobbyListUI` and `LobbyUI` scripts provide the visual interfaces for users to interact with the lobby system, while the `LobbyPlayerSingleUI` script helps present player information within a lobby.

This implementation enables users to connect, coordinate and configure their shared AR experience before entering the virtual environment, fostering collaboration and communication within the application.

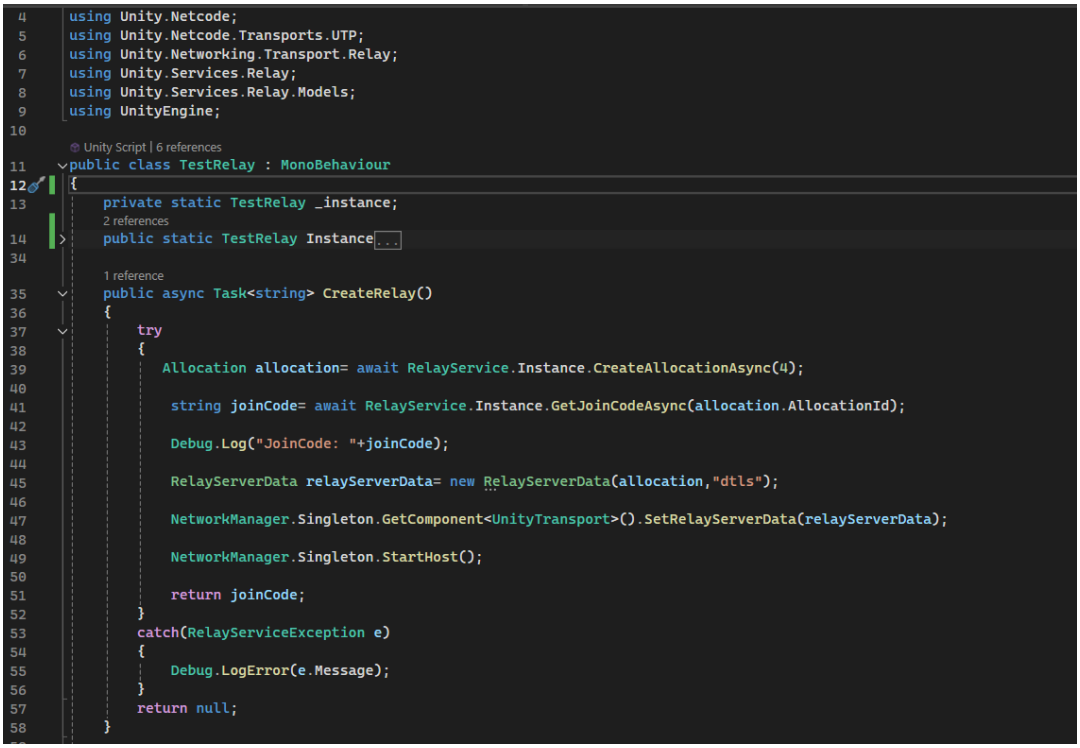
5.5.2 Unity Relay

Unity Relay is a service that enables communication between AR devices even if they are behind different network configurations (e.g., firewalls or NAT). It allows for direct peer-to-peer connections, ensuring low-latency and reliable communication crucial for a smooth collaborative AR experience.

In the application, Unity Relay is integrated through the `TestRelay` script 5.61 5.62. This script handles the creation and joining of relay connections, abstracting away the complexities of network setup and management. When a user hosts a new session, the `TestRelay` script generates a unique relay code that is used by other users to join the session. This relay code ensures that all participants connect to the same relay server and can communicate directly with each other. The relay code is automatically shared with other players in the lobby through the Unity Lobby service.

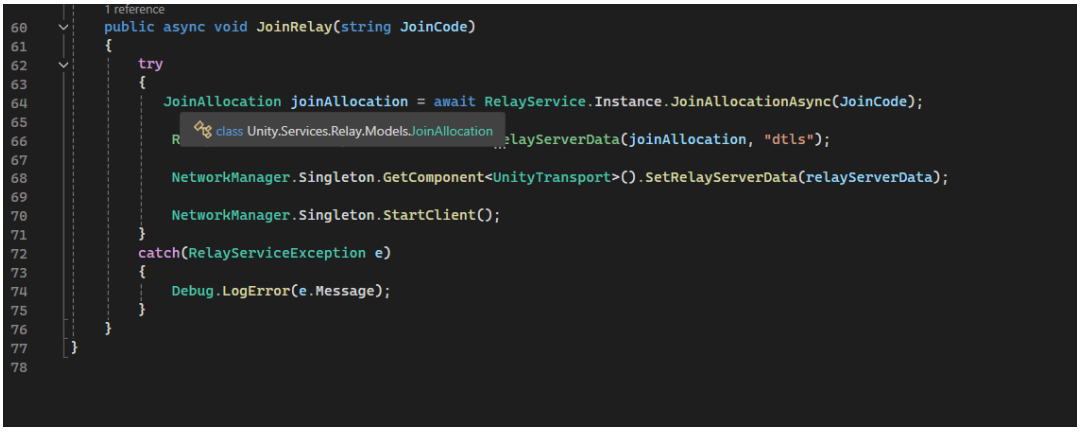
5.5.3 Networking with Netcode for GameObjects

This section delves into the implementation of networking in the collaborative AR application using Unity's Netcode for GameObjects. We begin with an analysis of Netcode's underlying architecture and mechanisms, followed by a detailed explanation of how we utilized its components and features to create a shared AR experience on HoloLens 2 devices.



```
using Unity.Netcode;
using Unity.Netcode.Transports.UTP;
using Unity.Networking.Transport.Relay;
using Unity.Services.Relay;
using Unity.Services.Relay.Models;
using UnityEngine;

Unity Script | 6 references
public class TestRelay : MonoBehaviour
{
    private static TestRelay _instance;
    2 references
    public static TestRelay Instance...

    1 reference
    public async Task<string> CreateRelay()
    {
        try
        {
            Allocation allocation= await RelayService.Instance.CreateAllocationAsync(4);

            string joinCode= await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);

            Debug.Log("JoinCode: "+joinCode);

            RelayServerData relayServerData= new RelayServerData(allocation,"dtls");

            NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

            NetworkManager.Singleton.StartHost();

            return joinCode;
        }
        catch(RelayServiceException e)
        {
            Debug.LogError(e.Message);
        }
        return null;
    }
```

FIGURE 5.61: The Test Relay script.



```
    1 reference
    public async void JoinRelay(string JoinCode)
    {
        try
        {
            JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(JoinCode);

            class Unity.Services.Relay.Models.JoinAllocation
            R                                              elayServerData(joinAllocation, "dtls");

            NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

            NetworkManager.Singleton.StartClient();
        }
        catch(RelayServiceException e)
        {
            Debug.LogError(e.Message);
        }
    }
}
```

FIGURE 5.62: The Test Relay script+.

Netcode for GameObjects (Netcode) is a high-level networking Application Programming Interface (API) within Unity that simplifies the development of multiplayer games and applications. It provides a framework for creating client-server architectures where multiple clients can connect to a central server and interact with a shared environment. Netcode handles the complexities of network communication, data synchronization, and state management, allowing developers to focus on implementing the gameplay logic and interactions of their collaborative AR experiences.

Netcode revolves around several key concepts that are fundamental to understanding its functionality:

NetworkManager: This component acts as the central control point for the network, managing connections, and overall network state. It is responsible for starting and stopping the network and handling client connections. In this component we can specify the player prefab that is instantiated as a network object and assigned to each connected client of the network. It also contains a reference to all network prefabs allowed on the network.

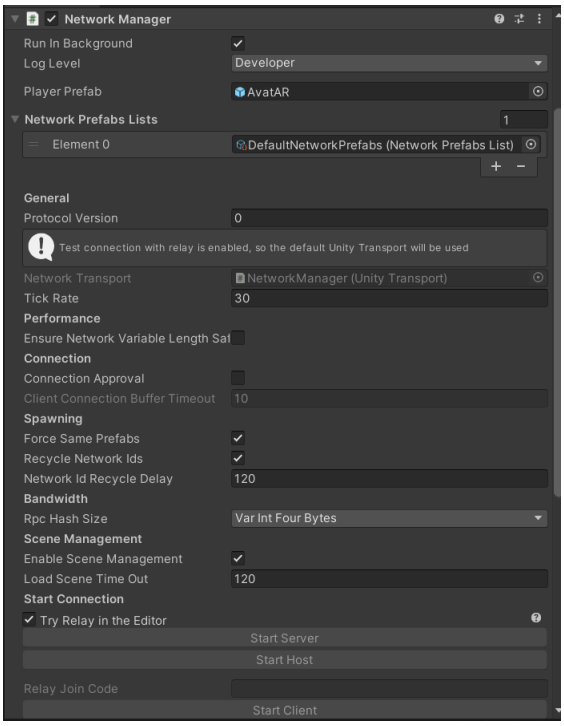


FIGURE 5.63: Application Network Manager

NetworkObjects: Any game object that needs to be synchronized across the network is designated as a NetworkObject. This allows Netcode to track its state and replicate changes to all connected clients. Furthermore, Netcode provides built-in functionality for spawning and despawning NetworkObjects across the network. This ensures that objects are created and destroyed consistently on all clients, maintaining a synchronized and shared experience. NetworkObjects can have various components attached to them that define their behavior and data.

NetworkTransform: Any game object that needs to have a synchronized position across the network is designated as a NetworkTransform. This allows Netcode to

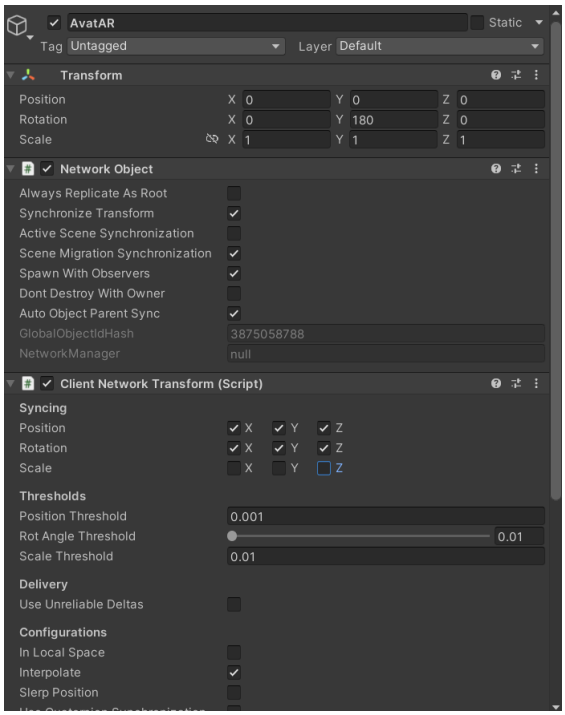


FIGURE 5.64: Player Network Object.

track its position and replicate changes to all connected clients. Objects with a NetworkTransform attached can only be moved by the server. So in the case a client wants to move such an object they have to request the move from the server. In case an object needs to be directly moved by the client, the ClientNetworkTransform can be utilized instead. In a collaborative setting we can trust users to move some objects, i.e. their player character, so both NetworkTransform and ClientNetworkTransform were used as needed in the project.

Remote Procedure Calls (RPCs): RPCs are functions that can be invoked on a remote client or the server. They provide a mechanism for triggering actions on other clients, enabling events like interacting with shared objects, playing sounds, or updating the state of the AR environment.

ServerRPCs Server RPCs are called from a client. The network gathers information on this call ,such as the object, component and method calling the RPC as well as any parameters and further information on its sender or receiever. The server receives said information and makes the call as specified.

ClientRPCs The server can invoke ClientRPCs on a network object. The call is placed on a queue and sent to a selection of clients. When received the client executes the client RPC in its corresponding network object.

NetworkVariables: NetworkVariables are variables that are automatically synchronized across the network. This ensures that all clients have a consistent view of important data, such as player positions, object states, and game progress.

Network Topologies and the Client-Server Model

Netcode for GameObjects supports various network topologies, which determine how clients and servers interact. Common topologies include client-server, peer-to-peer (P2P), and hybrid approaches.

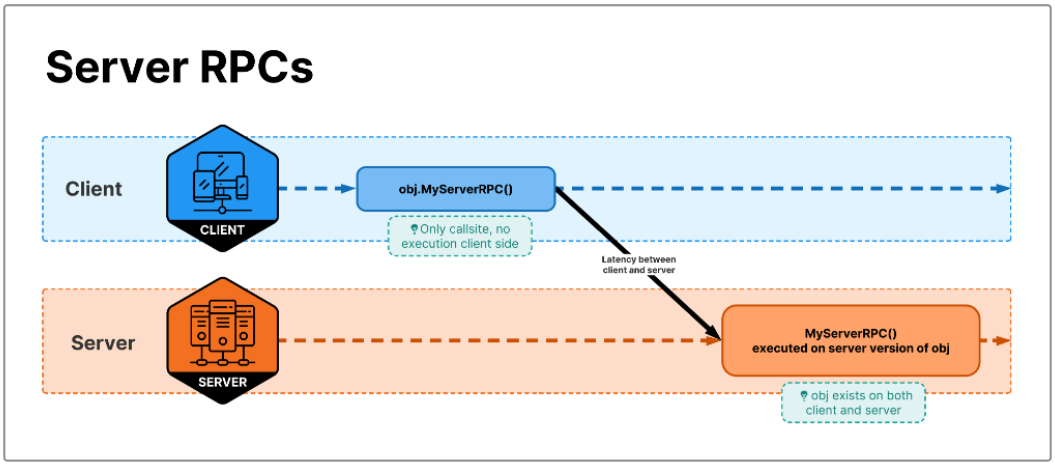


FIGURE 5.65: Server RPCs [35]

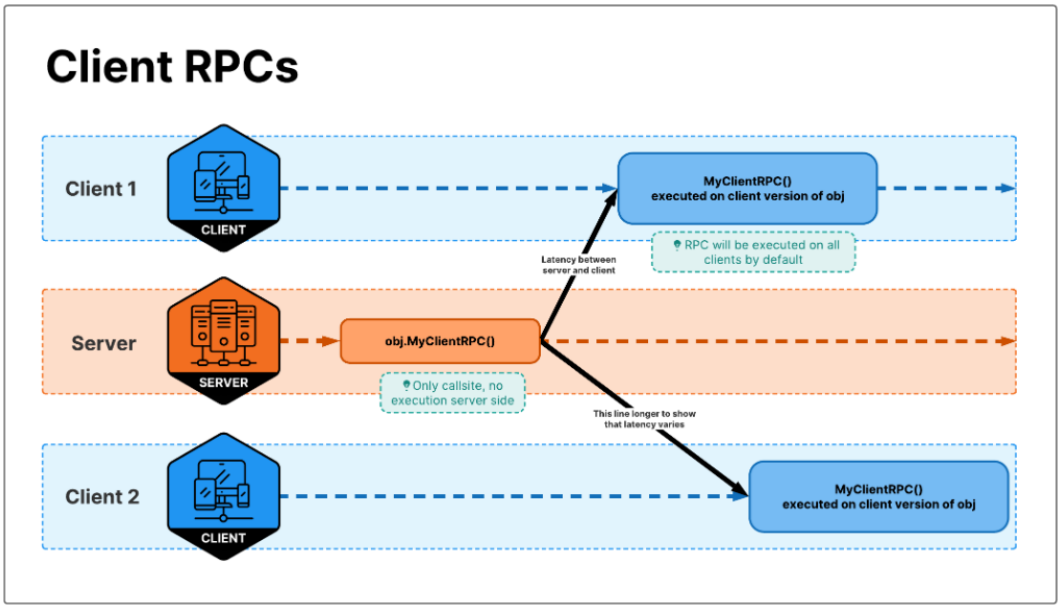


FIGURE 5.66: Client RPCs [35]

In the client-server model, a dedicated server hosts the game and manages the authoritative state, while clients connect to it and receive updates. This model offers advantages like centralized authority, scalability, and security, making it suitable for collaborative AR applications with complex interactions and a larger number of users.

Netcode primarily utilizes the client-server model. The server is responsible for hosting the game, managing connections, synchronizing state, and validating client actions. Clients connect to the server, send input data, receive updates, and render the game scene accordingly.

For the implemented collaborative AR application, the client-server model was chosen due to its suitability for handling complex interactions between multiple HoloLens 2 users and ensuring a consistent and secure shared experience.

Implementation of Netcode Components

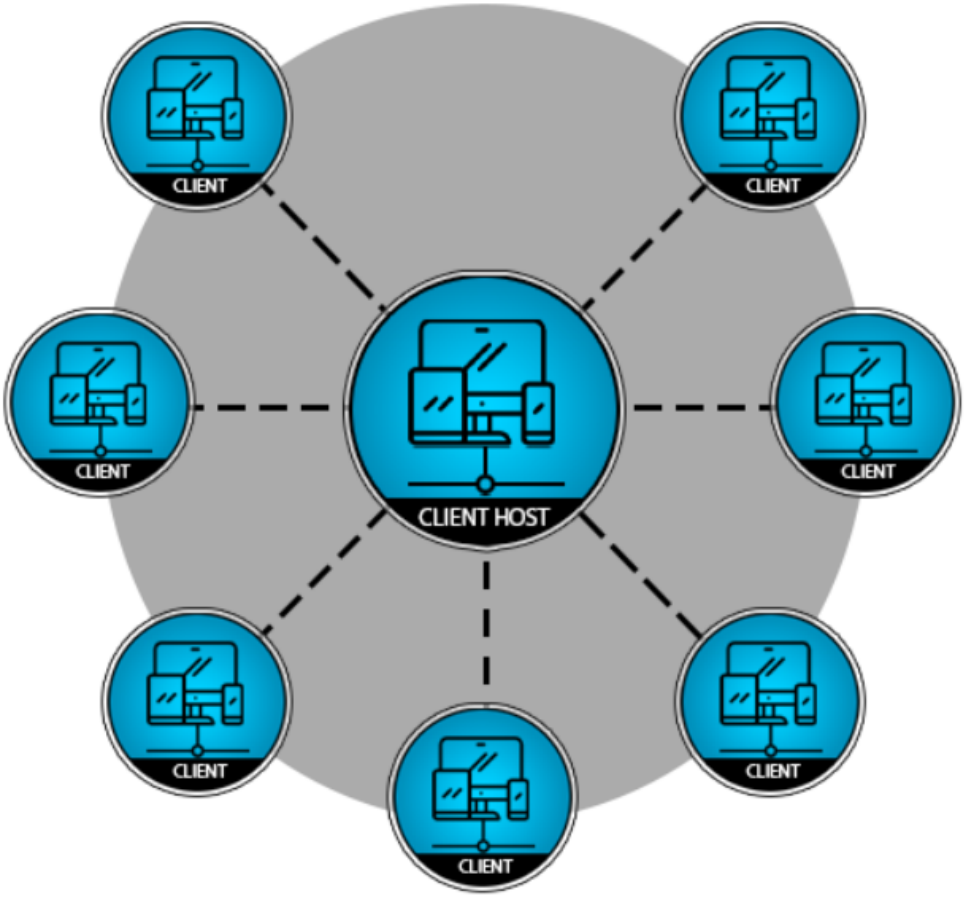


FIGURE 5.67: App Network topology [36].

In the collaborative AR application, Netcode for GameObjects was utilized to create a shared experience where multiple HoloLens 2 users can interact with the same virtual objects and environment. Here's how the key components of Netcode were implemented:

A NetworkManager 5.63 was configured in the scene to manage the network connections. We specified the relay network transport to be used and set up the appropriate network prefabs for the player and other objects that need to be spawned across the network.

The relevant game objects in the scene were designated as NetworkObjects 5.64. This included the virtual dwellings that users can interact with, as well as any UI elements that need to be synchronized across clients. NetworkTransform or ClientNetworkTransform were used to synchronize object position.

RPCs were used to handle various interactions and synchronization in the collaborative AR experience. RPCs were also used to trigger animations, and update the state of shared objects. NetworkVariables were used to synchronize critical data in

the AR scene. This included any other data that needs to be consistent across all clients, such as player avatar color etc.

The person that creates a session is deemed the host and all other users are asigned clients. With each client that spawnes, a player avatar for each player is also spawned. Each avatar moves as their user moves and has their own user color.

Any scripts that utilized networked components such as RPCs and network variables need to inherit from NetworkBehaviour. Some important networked scripts and their functionality are presented below.

`DesignNetworkSyncSript`: This script manages the synchronized state of the scene during the design stage of the application. It :

- Synchronizes key design aspects like the current floor level and button states.
- Tracks the covered area on each floor and triggers events when a floor is full.
- Manages the activation and logic of UI buttons (Next Floor, Close Lobby).
- Propagates application phase changes across the network.
- Saves and loads module placement data.

`ModuleSpawner`: This script spawns modules on the server when requested by clients and manages the spawned modules' ownership. Finally when the design is saved, it despawns and destroys all spawned modules on the server, cleaning up the scene.

`Module`: This script controls the behavior of a module (dwelling). It handles placement logic, ensuring the module snaps to designated areas with valid rotation. It provides visual feedback, changing the module's material to indicate the fit of the module in the last placed area.The script also propagates the module's state and design changes across the network. While the module is moving the script is responsible to occupy and vacate building areas.

`BuildingArea`: This script defines a BuildingArea and its behaviour. It tracks and synchronizes whether the area is occupied by a module. It highlights the area when a module can be placed there and uses NetworkVariable to synchronize the occupancy state across clients.

```
using UnityEngine;
using Unity.Netcode;
using System.Collections.Generic;
using Unity.Netcode.Components;

Unity Script (2 asset references) | 0 references
public class PlayerController : NetworkBehaviour
{
    public NetworkTransform networkTransform;
    private Animator animator;

    // Network variable to synchronize color across the network
    public NetworkVariable<Color> playerColor = new NetworkVariable<Color>(Color.white); // Default color
```

FIGURE 5.68: Player Character Color Network Variable.

5.5.4 Unity CloudSave for Application Data Management

Unity CloudSave is a core component of this application, providing a scalable and reliable solution for storing and managing data in the cloud. The system ensures persistence across sessions and devices, enabling a seamless user experience. By leveraging Unity CloudSave, the application can store, retrieve, and manage data

for lobbies, designs, customizations, and ratings, facilitating collaborative and independent workflows.

Data Organization and Storage

The data stored using Unity CloudSave is categorized into key segments:

- **Lobby Data:** Includes details such as lobby names, session modes, and associated metadata. This data is dynamically retrieved to display active lobbies.
- **Design Data:** Captures user-generated designs, including room layouts, building placements, and apartment configurations.
- **Customization Data:** Stores user-selected options during customization phases, such as apartment layouts, furniture arrangements, and room placements.
- **Rating Data:** Stores user feedback and ratings for completed designs, enabling evaluation and iterative improvements.

Workflow and Implementation

The integration of Unity CloudSave involves several steps to ensure efficient data handling:

1. **Data Serialization:** Before uploading data to the cloud, it is serialized into JSON format. This ensures compatibility with Unity's storage APIs and facilitates easy deserialization when data is retrieved.
2. **Key Management:** Each data entry is assigned a unique key to ensure organization and prevent conflicts. Keys are constructed using information such as lobby names, user identifiers, and data categories (e.g., `LobbyName_ApartmentName_UserName`).
3. **Data Upload and Retrieval:** CloudSave's asynchronous methods are utilized to upload and retrieve data, ensuring non-blocking operations. This is critical for maintaining application performance and responsiveness.
4. **File Listing and Indexing:** Lists of saved files are maintained in the cloud to support dynamic loading and display of data, such as available lobbies or saved designs.

Usage Across Application Phases

Unity CloudSave is used extensively across the application's lifecycle:

- **Lobby Management:** Lobbies created by users are stored in the cloud and dynamically retrieved for listing. This ensures all users have access to the latest lobby information in real time.
- **Design and Customization:** User-generated designs and customizations are stored securely in the cloud, allowing users to revisit their progress across devices or sessions.
- **Ratings and Feedback:** User feedback and ratings for completed designs are saved and utilized for evaluation and iterative improvements.
- **Modules and Room Data:** CloudSave also manages detailed module data and room configurations, enabling precise customization and design continuity.

Advantages of Unity CloudSave Integration

The implementation of Unity CloudSave offers several benefits:

- **Data Persistence:** Ensures that user data remains available across sessions and devices, providing a seamless experience.
- **Scalability:** Supports the growing complexity of the application by managing diverse data types and increasing storage needs.
- **Reliability:** Unity's backend infrastructure guarantees consistent data availability and integrity.
- **Collaboration Support:** Enables synchronized data sharing between users, fostering collaboration in shared AR environments.

By integrating Unity CloudSave, the application achieves robust data management capabilities, ensuring a scalable and user-centric experience.

5.5.5 Save System and Supporting Scripts Overview

The Save System in Unity manages the persistence of game data using cloud storage. It enables saving and loading of various elements such as room layouts, customizations, notes, and ratings.

The `SaveSystem` Class

The SaveSystem class is responsible for managing cloud-based save operations. It interacts with CloudStorage to store and retrieve structured data.

It supports saving room configurations and associated metadata. It maintains structured file names for organized retrieval.

The `CloudStorage` Class

CloudStorage handles data transfer between the game and Unity's Cloud Save Service. It provides asynchronous methods for saving and loading data while ensuring error handling and logging.

The `RoomInfo and RoomData` Class

RoomInfo stores the room's ID, layout index, and customization details. It enables network serialization and comparison.

RoomData records transformation data including position, rotation, and scale. It allows checking if a room's position matches a given coordinate.

The `ModuleData` Class

ModuleData is similar to RoomData but applies to modular components. It tracks ownership through an ownerID and supports cloud synchronization.

The `RatingInfo` Class

RatingInfo stores user ratings for designs. It enables serialization for cloud storage and multiplayer sharing.

System Workflow

A player customizes a room layout and the system stores RoomData and RoomInfo. The customization is saved through SaveSystem and stored in the cloud. The player retrieves saved designs by loading RoomData from the cloud. Ratings and notes are saved and retrieved similarly.

The system ensures efficient data persistence using structured keys and cloud integration. It provides a seamless experience for storing and retrieving player data in multiplayer environments.

5.5.6 Player Awareness in a Networked Environment

To help users be more aware of their collaborators' position and actions, some simple but effective awareness cues were implemented. A user avatar 5.69 was used for the embodiment of all collaborators in the MR scene. Virtual hands 5.70 were spawned to mark each user's two previous interactions with the AR environment. Each user hands are colored with a unique user color to help better diferentiate user actions. Depending on the user action a specific hand animation corresponding to each user action is played so the collaborators have a better understanding of their collaborators actions.

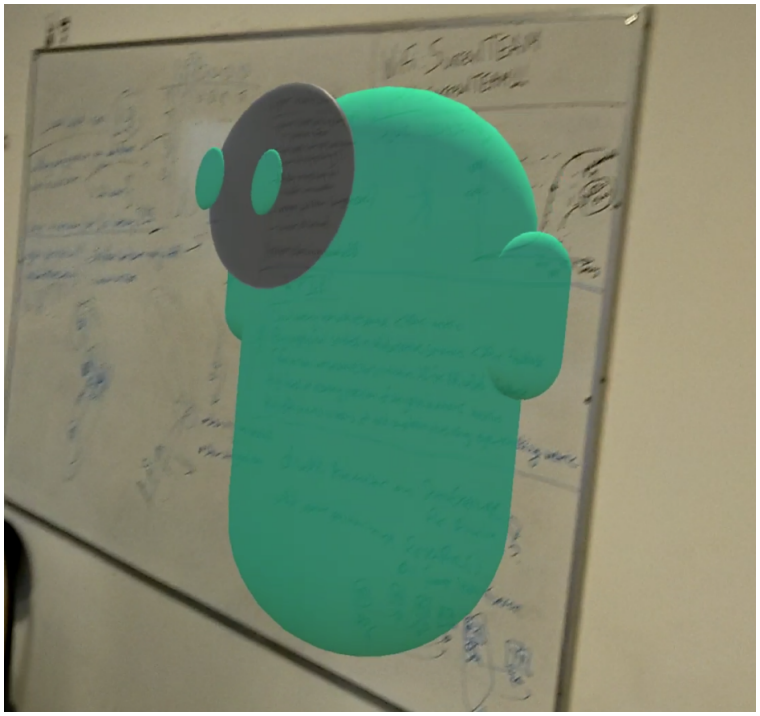


FIGURE 5.69: Player avatAR at runtime.

For the implementation of the player avatar the playerAvatAR prebaf was created 5.71 . Each prefab has a **PlayerController** script attached to it: This script controls the player's behavior and appearance in a networked AR environment. It:

- Synchronizes the player's position and rotation across the network.
- Detects when the player is walking and triggers visual effects.
- Manages the player's color, glow, and iridescent effects.

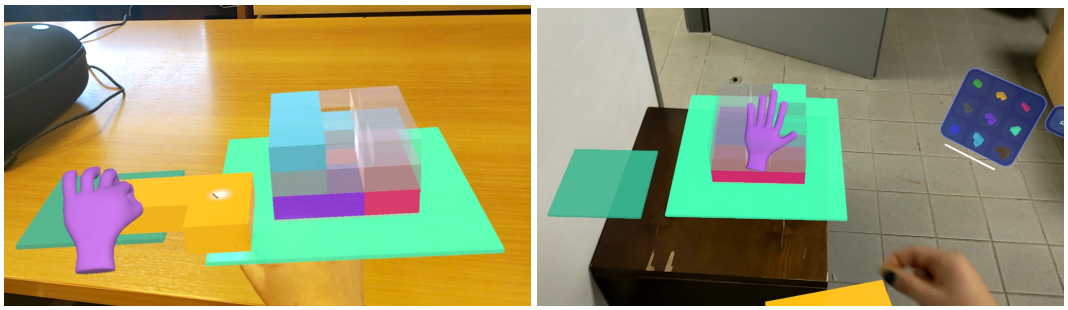


FIGURE 5.70: Player Hands at runtime.

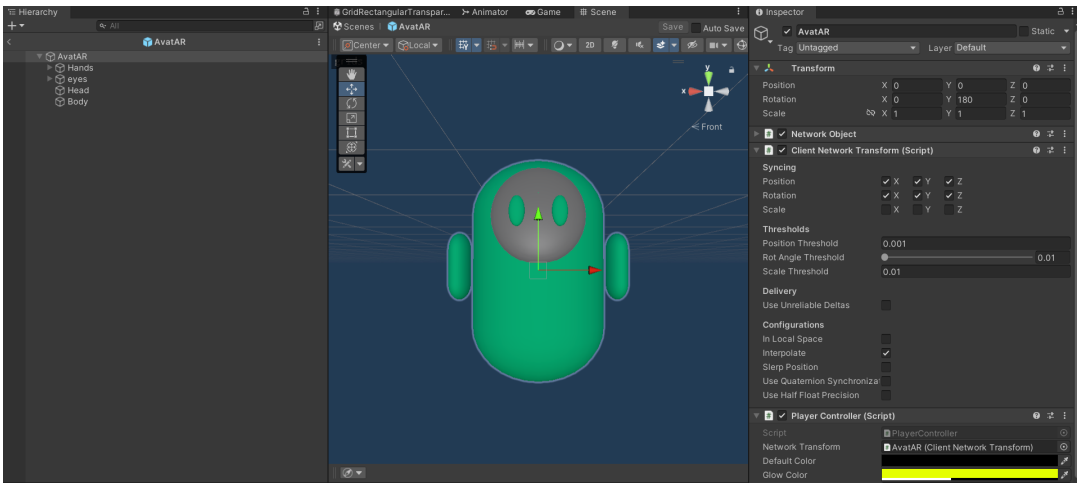


FIGURE 5.71: Player avatAR prefab.

- Uses NetworkVariables to synchronize color and iridescent state across clients.

For the hand functionality the following scripts were created: **HandManager, HandController and InteractableModule** and a hand prefab and animator. The 'HandController' manages each individual hands' prefab color and animations. The 'HandManager' manages all spawning and keeping track of all player hands in scene. The Interactable Module is a class assigned to each object of the user interface for which we need to keep track the of the user interactions with said object.

5.5.7 Challenges and Solutions

During the implementation of Netcode, we encountered some challenges related to network latency and object synchronization. To address these issues, only objects that absolutely needed to be in the network were networked, making some objects and interfaces local. Also the information sent through NetworkTransforms and ClientNetworkTransforms, was minimized to include only the absolutely necessary information. For Example the player avatar scale doesnt change throughout the experience so we opted out of sending this information across the network 5.64.

User Evaluation

Evaluating the usability and effectiveness of **AR-Apt**, the *Architectural Participatory Design* application, was an essential step in validating its capabilities in a collaborative **Augmented Reality (AR)** environment. The system was tested in **pairs of users**, simulating real-world collaborative design scenarios.

Following each session, participants completed a **System Usability Scale (SUS) questionnaire**, supplemented by additional **custom questions** tailored to assess the collaborative nature of the experience. This chapter presents the **evaluation methodology**, including the **study setup**, **questionnaire structure**, and **key findings** derived from participant feedback.

6.1 Observations and First Feedback

Beyond the structured questionnaire, qualitative observations during the sessions provided valuable insights into user experience and interaction challenges. Several key adjustments were made based on immediate feedback:

- **Avatar Positioning Adjustments**: Initially, user avatars were positioned in a way that could partially obstruct the user's field of view. To enhance visibility and comfort, **avatar offsets were adjusted** to ensure they did not interfere with the user's perspective while still maintaining spatial awareness of collaborators. After more considerations, self-avatars were turned off to avoid obstructing the user's view. This meant that only other users viewed their collaborators' avatars.
- **Hand Presence Reduction**: Users noted that keeping their virtual hands in the scene for extended periods caused confusion, as they sometimes mistook static hands for interactive elements. In response, the system was modified to make hands disappear when idle, reducing unnecessary distractions.
- **Model Positioning Adjustments**: Some architectural models were positioned too high within the AR scene, making them difficult to view comfortably. To address this, we **lowered certain models** within the environment, ensuring that users could easily examine and interact with them without requiring excessive head movement.

These iterative refinements significantly improved **user comfort**, **spatial awareness**, and **interaction clarity** within the collaborative AR environment.

6.2 Evaluation Methodology

The evaluation aimed to measure both **system usability** and the effectiveness of **collaborative interactions in AR**. To achieve this, **two users** participated in each testing session, collaboratively designing a **virtual apartment layout** using AR headsets. Each session followed a structured workflow:

1. **Task Briefing**: Participants received an introduction to the system and followed the application tutorial.
2. **Collaborative Design Session**: Users interacted with **private** and **shared workspaces** to test the distinction between **individual modifications** and **group collaboration**.
3. **Post-Session Questionnaire**: After completing the task, participants provided feedback through a **SUS questionnaire** and additional **collaboration-focused** questions.

6.2.1 Measurement Tools

The evaluation relied on two primary measurement tools:

- **System Usability Scale (SUS)**:A standardized **10-item** questionnaire designed to assess perceived usability [37], efficiency and learnability of the system.
- **Collaboration-Specific Questions**: To assess how effectively the system supported multi-user interactions, participants answered additional **Likert-scale questions**, including:
  - *The system facilitated seamless collaboration between users.*
  - *My collaborator's actions were accurately represented to me.*
  - *I was aware of my collaborator's presence in the scene.*
  - *The distinction between private and shared workspaces was clear.*

These questions aimed to evaluate **spatial awareness**, **real-time interaction fidelity**, and **workspace organization**, all of which are crucial for a **multi-user AR environment**.

6.3 Results and Discussion

6.3.1 System Usability Results

The results of the **SUS questionnaire** provided a quantitative measure of usability 6.1, 6.2, 6.3 and 6.4. The responses indicated that the system was **intuitive and easy to use**, with an average rating of **4 out of 5** in terms of frequency of usage likelihood. Additionally, **80% of participants** agreed that most people would learn to use the system quickly. However, some participants (**16.7%**) expressed minor challenges in distinguishing between **private and shared workspaces**.

6.3.2 Collaboration Insights

The responses to the **collaboration-specific** 6.5, 6.6 questions highlighted several key observations:

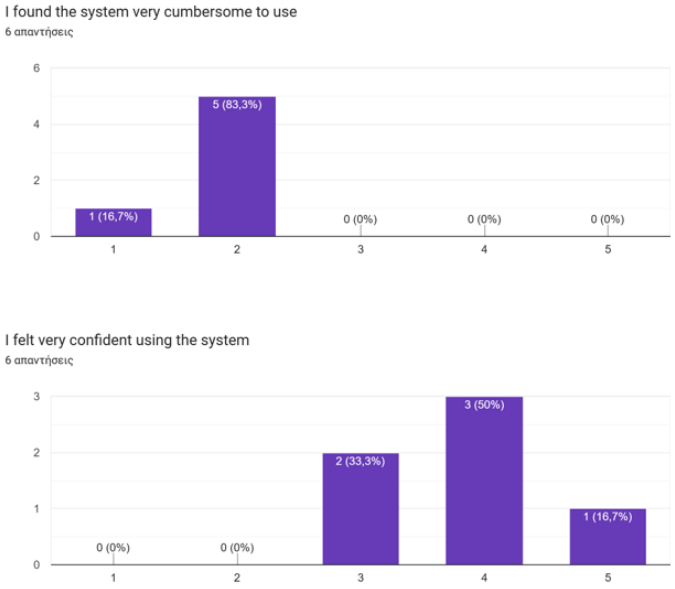


FIGURE 6.1: SUS questions.

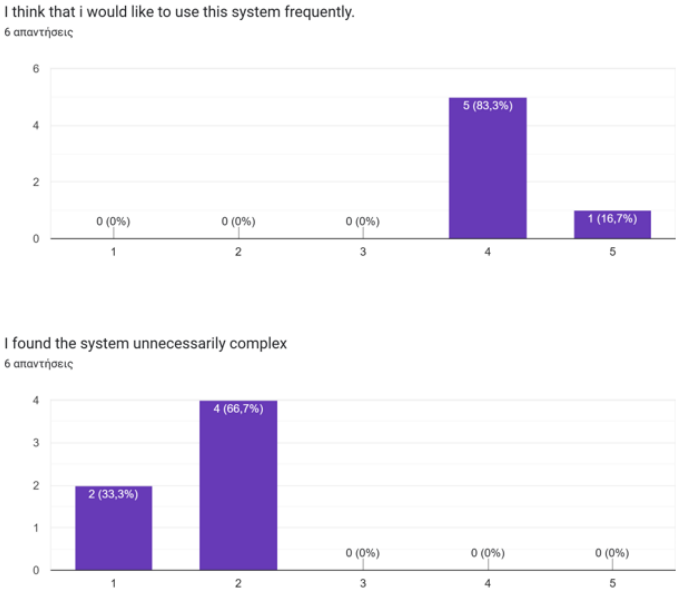


FIGURE 6.2: SUS questions 1.

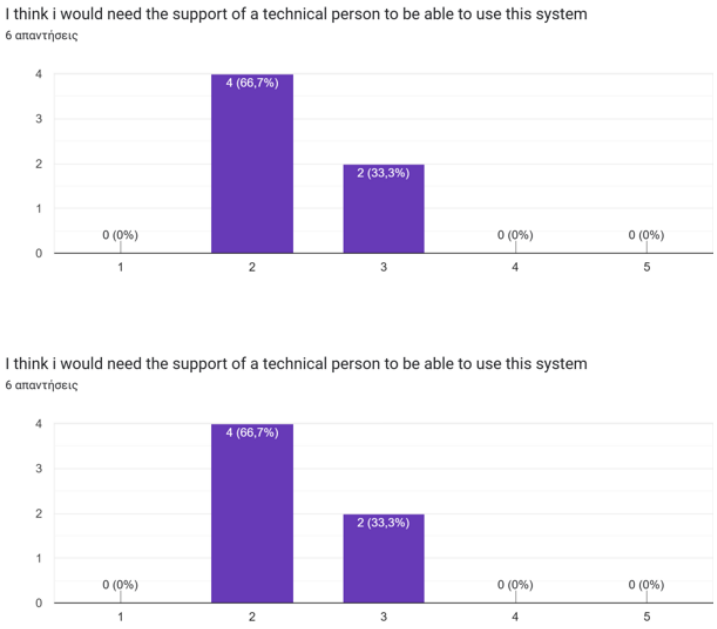


FIGURE 6.3: SUS questions 2.

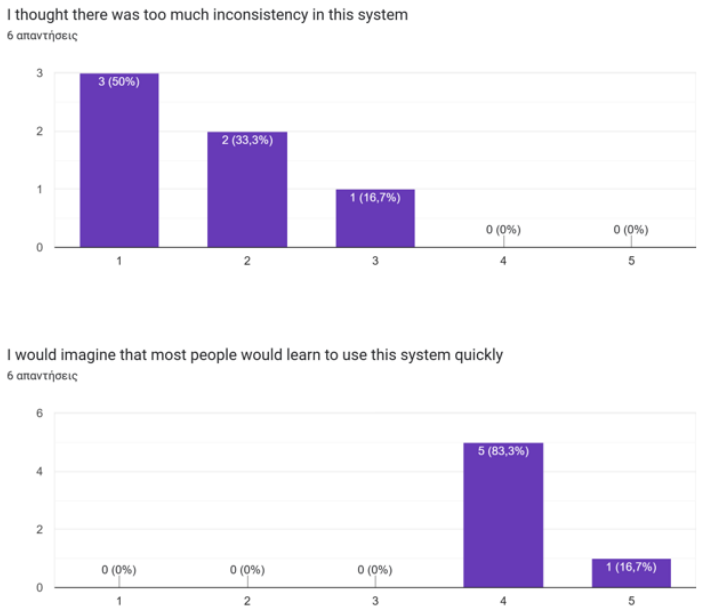


FIGURE 6.4: SUS questions 3.

- **66.7% of users** reported **high awareness** of their collaborators' actions, and **33.3% of users** reported **some awareness** of their collaborators' actions indicating that the system effectively represented real-time interactions.
- **83.3% of participants** found the distinction between private and shared workspaces to be clear, though some suggested that additional **visual feedback** could enhance workspace differentiation.

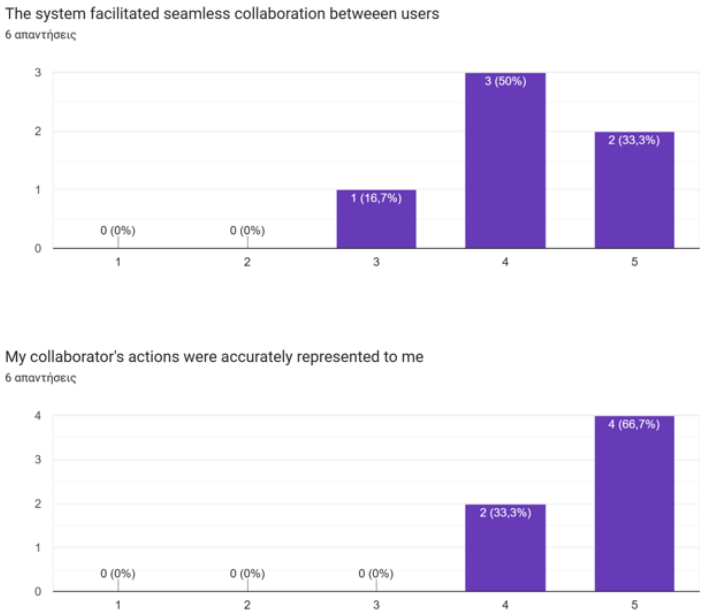


FIGURE 6.5: Collaboration Specific Questions.

6.4 Conclusion

The user evaluation provided valuable insights into the usability and collaborative functionality of the system. The SUS scores reflected an overall positive user experience, while the collaboration-focused questions highlighted areas for improvement in workspace differentiation and real-time interaction feedback.

Future iterations of the system will aim to address these minor usability concerns by refining interface design, improving real-time updates, and further optimizing the balance between private and shared interactions in collaborative AR environments.

During the implementation of this thesis, several challenges arose. Most notably, a significant portion of the development was conducted without access to a HoloLens 2 device. While this was not the intended development approach, it serves as a valuable lesson: direct interaction with the target hardware is crucial for effective AR application design. While the tools provided by Unity and Microsoft are comprehensive, they cannot fully replicate the real-world user experience.

A more effective approach would involve early and consistent interaction with the HoloLens 2. This could include access to the device during the research phase, allowing for iterative design and testing based on direct user feedback. Additionally,

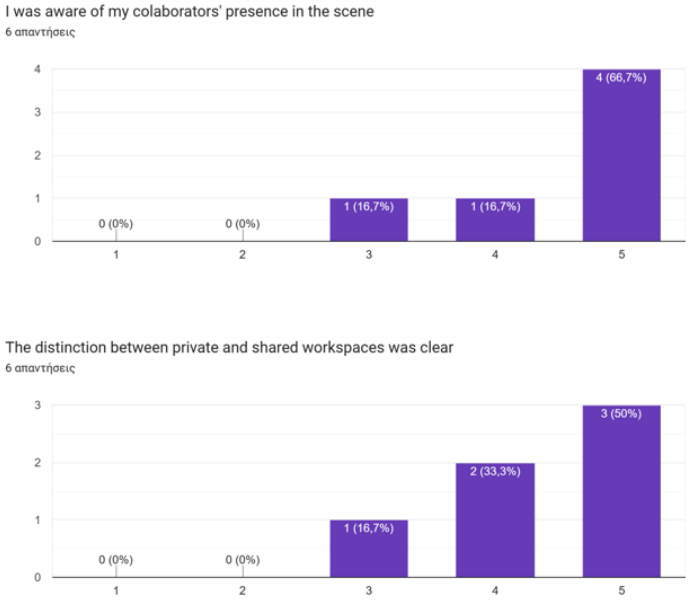


FIGURE 6.6: Collaboration Specific Questions 1.

engaging with other AR developers and leveraging their expertise and insights, such as those from the SURREAL team, would have significantly benefited the development process.

Another significant challenge encountered was the complexity of network synchronization in a multiplayer application. This highlights the importance of developing a small, functional multiplayer prototype early in the development cycle to understand the intricacies of network communication and potential challenges.

The thesis employed a host-listen server topology, which placed a substantial processing load on the host HoloLens 2 device, leading to noticeable lag compared to client devices. A more robust and scalable solution would utilize a dedicated server architecture to offload processing from the HoloLens 2 and ensure smoother performance for all users.

While the HoloLens 2 is a sophisticated device, it's crucial to acknowledge its limitations. The device's performance can be impacted by environmental factors such as lighting conditions, which can hinder outdoor use in many locations. This should be carefully considered when defining application requirements and use cases.

Finally, this thesis underscores the critical importance of a thorough and iterative user-centered design process. The emphasis on user requirements and prototyping proved invaluable in saving significant time and effort during later stages of development. A robust design process, with early user feedback and iterative refinements, is essential for successful AR application development.

Future Work

While the current implementation of **AR-Apt** successfully supports collaborative architectural design in Augmented Reality (AR), enhancements can further improve the application's capabilities and user experience. This chapter outlines potential future developments, focusing on extended customization features, real-scale walkthrough functionality, and exterior modifications that were not completed due to time constraints.

7.1 Further Interior Customization

One of the primary areas for improvement is the expansion of interior customization options. Currently, users can select and arrange room layouts, but further enhancements can make the design process more detailed.

Material and Texture Customization

Future iterations of **AR-Apt** will incorporate the ability to customize textures and materials for interior elements such as walls, floors, ceilings, and furniture. Users will be able to:

- Choose from a library of materials (e.g., wood, marble, tiles, carpets) to apply to surfaces.
- Adjust colors and reflectivity to match desired aesthetics.

This addition will improve realism and personalization, allowing users to better visualize their living spaces.

Furniture Placement and Customization

While the current system enables users to arrange dwelling modules, adding interactive furniture placement will further enhance user engagement. Features to be included are:

- Adjustable furniture sizes and orientations.
- Customizable furniture materials and colors.

These functionalities will provide users with greater flexibility in designing their interior spaces.

7.2 Real-Scale Walkthrough Implementation

Another significant future development is the implementation of a real-scale walkthrough feature, allowing users to navigate their designs in life-size AR environments. This feature will provide:

- First-person perspective navigation using AR tracking.
- Dynamic lighting adjustments to simulate real-world conditions.

A real-scale walkthrough will enhance spatial understanding and improve the decision-making process by offering an immersive experience of the designed apartments.

7.3 Exterior Customization

Due to time constraints, exterior design options were not implemented in the current version of **AR-Apts**. In future iterations, users will be able to:

- Modify the facade of buildings with different materials, window styles, and shading elements.
- Customize outdoor spaces such as balconies, and terraces.

These additions will ensure that the participatory design process includes not just interior spaces but also exterior aesthetics and functionality. This will also be able to leverage AR characteristics enabling the on-site visualization of the finished result.

7.4 Conclusion

The future development of **AR-Apts** will focus on improving customization capabilities, implementing real-scale navigation, and expanding exterior modification options. These enhancements will significantly increase user engagement and the overall realism of the AR design experience. By incorporating these features, the system will provide a more comprehensive tool for architectural participatory design, allowing users to fully realize and interact with their envisioned spaces before construction begins.

Bibliography

[1] R. Azuma et al. "Recent advances in augmented reality". In: *IEEE Computer Graphics and Applications* 21.6 (2001), pp. 34–47. DOI: 10.1109/38.963459.

[2] Ronald T. Azuma. "A Survey of Augmented Reality". In: *Presence: Teleoperators and Virtual Environments* 6.4 (Aug. 1997), pp. 355–385. DOI: 10.1162/pres.1997.6.4.355. eprint: https://direct.mit.edu/pvar/article-pdf/6/4/355/1623026/pres.1997.6.4.355.pdf. URL: https://doi.org/10.1162/pres.1997.6.4.355.

[3] Mickael Sereno et al. "Collaborative Work in Augmented Reality: A Survey". In: *IEEE Transactions on Visualization and Computer Graphics* 28.6 (2022), pp. 2530–2549. DOI: 10.1109/TVCG.2020.3032761.

[4] Anders Henrysson, Mark Billinghurst, and Mark Ollila. "Face to Face Collaborative AR on Mobile Phones". In: *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*. ISMAR '05. USA: IEEE Computer Society, 2005, pp. 80–89. ISBN: 0769524591. DOI: 10.1109/ISMAR.2005.32. URL: https://doi.org/10.1109/ISMAR.2005.32.

[5] Julia Hertel et al. "A Taxonomy of Interaction Techniques for Immersive Augmented Reality based on an Iterative Literature Review". In: *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2021, pp. 431–440. DOI: 10.1109/ISMAR52148.2021.00060.

[6] Ken Pfeuffer et al. "ARtention: A design space for gaze-adaptive user interfaces in augmented reality". In: *Computers & Graphics* 95 (2021), pp. 1–12. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0097849321000017.

[7] Shakiba Davari, Feiyu Lu, and Doug A. Bowman. "Occlusion Management Techniques for Everyday Glanceable AR Interfaces". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, pp. 324–330. DOI: 10.1109/VRW50115.2020.00072.

[8] Ann McNamara, Chethna Kabeerdoss, and Conrad Egan. "Mobile User Interfaces Based on User Attention". In: *Proceedings of the 2015 Workshop on Future Mobile User Interfaces*. FutureMobileUI '15. Florence, Italy: Association for Computing Machinery, 2015, pp. 1–3. ISBN: 9781450335041. DOI: 10.1145/2754633.2754634. URL: https://doi.org/10.1145/2754633.2754634.

[9] *HoloLens (1st gen) hardware*. https://learn.microsoft.com/en-us/hololens/hololens1-hardware(2024/07/22).

[10] *Microsoft HoloLens 2*. https://www.microsoft.com/el-gr/hololens/(2024/07/22).

[11] *Hands on: Magic Leap One review*. https://www.techradar.com/reviews/magic-leap-one(2024/07/22).

[12] *Meet Meta Quest 3, Our Mixed Reality Headset Starting at $499.99*. https://about.fb.com/news/2023/09/meet-meta-quest-3-mixed-reality-headset/(2024/07/22).

[13] *Introducing Apple Vision Pro: Apple's first spatial computer*. https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/(2024/07/22).

[14] Jeremy Kerr and Gillian Lawson. "Augmented reality in design education: landscape architecture studies as AR experience". In: *International Journal of Art & Design Education* 39.1 (2020), pp. 6–21.

[15] Mario Wolf, Heinrich Söbke, and Florian Wehking. "Mixed reality media-enabled public participation in urban planning". In: *Augmented Reality and Virtual Reality* (2020), pp. 125–138.

[16] Yang Song, Richard Koeck, and Shan Luo. "Review and analysis of augmented reality (AR) literature for digital fabrication in architecture". In: *Automation in Construction* 128 (2021), p. 103762.

[17] Gwyllim Jahn et al. "Holographic construction". In: *Design Modelling Symposium Berlin*. Springer. 2019, pp. 314–324.

[18] Gwyllim Jahn, Cameron Newnham, and Matthew Beanland. "Making in Mixed Reality. Holographic design, fabrication, assembly and analysis of woven steel structures". In: (2018).

[19] Giovanni Betti, Saqib Aziz, and Gili Ron. "Pop Up Factory: Collaborative Design in Mixed Rality". In: *Simulation-VIRTUAL AND AUGMENTED REALITY 2* 3 (2019), pp. 115–125.

[20] Maria Velaora, Richard van Roy, and François Guéna. "ARtect, an augmented reality educational prototype for architectural design". In: *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE. 2020, pp. 110–115.

[21] Chris Panou et al. "An architecture for mobile outdoors augmented reality for cultural heritage". In: *ISPRS International Journal of Geo-Information* 7.12 (2018), p. 463.

[22] Max Allen, Holger Regenbrecht, and Mick Abbott. "Smart-phone augmented reality for public participation in urban planning". In: *Proceedings of the 23rd Australian computer-human interaction conference*. 2011, pp. 11–20.

[23] Hyekyung Imottesjo and Jaan-Henrik Kain. "The Urban CoBuilder–A mobile augmented reality tool for crowd-sourced simulation of emergent urban development patterns: Requirements, prototyping and assessment". In: *Computers, Environment and Urban Systems* 71 (2018), pp. 120–130.

[24] Krystian Kwiecinski, Jacek Markusiewicz, and Agata Pasternak. "Participatory Design Supported with Design System and Augmented Reality". In: (2017).

[25] Iulian Radu et al. "A Survey of Needs and Features for Augmented Reality Collaborations in Collocated Spaces". In: *Proc. ACM Hum.-Comput. Interact.* 5.CSCW1 (Apr. 2021). DOI: 10.1145/3449243. URL: https://doi.org/10.1145/3449243.

[26] Allison Jing et al. "Eye See What You See: Exploring How Bi-Directional Augmented Reality Gaze Visualisation Influences Co-Located Symmetric Collaboration". In: *Frontiers in Virtual Reality* 2 (June 2021), p. 697367. DOI: 10.3389/frvir.2021.697367.

[27] Tahir Mahmood et al. "Improving Information Sharing and Collaborative Analysis for Remote GeoSpatial Visualization Using Mixed Reality". In: *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2019, pp. 236–247. DOI: 10.1109/ISMAR.2019.00021.

[28] *Mixed Reality Toolkit 3*. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/(2024/07/23).

[29] *MRTKDevTemplate*. https://github.com/MixedRealityToolkit/MixedRealityToolkit-Unity/tree/main/UnityProjects/MRTKDevTemplate(2024/07/24).

[30] *Unity Real-Time Development Platform*. https://unity.com/(2024/07/25).
[31] *Microsoft Visual Studio*. https://visualstudio.microsoft.com/(2024/07/25).
[32] *Unity Lobby*. https://docs.unity.com/ugs/manual/lobby/manual/unity-lobby-service(2024/07/25).
[33] *Connect your Players with Relay*. https://unity.com/products/relay(2024/07/25).
[34] *About Netcode for GameObjects*. https://docs-multiplayer.unity3d.com/netcode/current/about/(2024/07/25).
[35] *Sending events with RPCs*. https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/(2024/07/25).
[36] *Network topologies*. https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/network-topologies/(2024/07/25).
[37] Aycan Kaya, Reha Ozturk, and Cigdem Altin Gumussoy. "Usability measurement of mobile applications with system usability scale (SUS)". In: *Industrial Engineering in the Big Data Era: Selected Papers from the Global Joint Conference on Industrial Engineering and Its Application Areas, GJCIE 2018, June 21–22, 2018, Nevsehir, Turkey*. Springer. 2019, pp. 389–400.