FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Alternative display technologies for Smart Homes

Lúcio Alexandre Mota Coito Almeida

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Luis Miguel Pinho de Almeida

June 30, 2023

Abstract

This dissertation explores disruptive display technologies, namely transparent and holographic, for use in Smart Home devices. It includes a study of these technologies consisting of understanding how these technologies work and determining their application feasibility on Smart Home devices, taking into account their characteristics such as screen size and type of information shown on the display. Then, this dissertation explores the development of two prototypes that resemble smart thermostats, each utilizing a different display technology: one incorporating a holographic display and the other employing a transparent display. For each prototype, a User Interface was created from scratch using the Unity interface development platform. Finally, the dissertation includes a study on User Interface and User Experience tests to get user feedback after the development of the prototypes. Users expressed a positive reception towards the prototypes, finding them both appealing and interesting. However, concerns about the prototypes refinement and possible cost were raised when compared with more traditional devices using Liquid Crystal Display technologies.

Agradecimentos

A elaboração do presente trabalho não seria possível sem o apoio de alguns intervenientes. Assim sendo, pretendo agradecer a todos os que sempre me apoiaram e contribuíram para a realização e concretização desta etapa final do meu mestrado em Engenharia Eletrotécnica e de Computadores. Deste modo, agradeço:

À minha familia e amigos, especialmente a minha mãe Conceição e os meus avós António e Olívia, pois tudo isto foi possível graças ao esforço e dedicação que sempre tiveram. Por estarem sempre lá para me amparar e por todo o apoio incondicional.

Ao meu orientador, o Prof. Luís Almeida, e coorientador, o Eng.º Rodrigo Almeida, pela oportunidade de realizar este trabalho e pelo acompanhamento ao longo do mesmo. Obrigado por me incentivarem a fazer sempre mais e melhor, e pelo enorme contributo que tiveram para o meu crescimento pessoal e educacional.

À minha equipa da Bosch, nomeadamente Fábio, Margarida, André e Milton, por me terem acompanhado ao longo de todo este trabalho, por toda a atenção, compreensão e conhecimentos transmitidos. Obrigado pela paciência, pelas palavras amigas e pela ajuda incondicional.

À minha noiva, Eduarda, que esteve sempre presente ao longo de todo este percurso e que sempre me apoiou de forma compreensiva e motivadora. Obrigado por todo o carinho, motivação e paciência.

Ao meu grupo de colegas de curso que terminou este percurso académico comigo, nomeadamente Henrique, Marco, Daniel e Martim. Obrigado pela motivação e ajuda que me proporcionaram para conseguir concluir esta etapa importante da minha vida.

Lúcio Almeida

*“If you don’t take risks,
you can’t create a future!”*

Monkey D. Luffy

Contents

List of Figures

List of Tables

Abbreviations and Symbols

2D	Two-Dimensional
3D	Three-Dimensional
API	Application Programming Interface
AVC	Accommodation-Vergence Conflict
GPS	Global Positioning System
GUI	Graphical User Interface
HUD	Heads-Up Display
HVAC	Heating, Ventilation and Air Conditioning
IBR	Image-Based Rendering
LCD	Liquid-Crystal Display
OLED	Organic Light-Emitting Diodes
PDLC	Polymer-Dispersed Liquid Crystal
SBC	Single-Board Computer
SotA	State of the Art
UI	User Interface
UX	User Experience

Chapter 1

Introduction

Bosch Thermotechnology is responsible for developing and researching space heating, air conditioning and hot water solutions. It fabricates Heating, Ventilation and Air Conditioning (HVAC) systems, and one of its fields of action are the Smart Homes. Bosch describes the term Smart Home as an intelligent connection of individual components within a property which are controlled and monitored centrally via end devices [1]. Among many advantages like easy installation/operation and data protection, a fundamental one is that Smart Homes can automatically control regular processes while ensuring the safety of the home. Smart Homes usually provide services based on comfort and monitoring, such as automation of daily routines (e.g. dishwashers and washing machines), remote home management (e.g. garden devices, windows and doors) and intelligent environmental monitoring (e.g. electricity and temperature) [2].

Within the context of Smart Homes, the User Interface (UI) must be considered when discussing the interaction between the user and the system, mainly because a wide variety of users can interact with the product, each in their own way. Therefore, it needs to cause a good impression when the user looks at it: be clean, approachable, appealing and accessible.

According to its characteristics, a product may offer different types of input or output modalities (like voice, touch, and gesture, among others). In the HVAC scope, one method of interaction with the system is through a human-machine interface, generally composed of a graphical or segmented display and some input buttons [3]. Thus, this dissertation aims to explore and evaluate disruptive display technologies for Smart Home devices, particularly for the HVAC systems context.

1.1 Goals

The main goal of this project under development at Bosch Thermotechnology is to innovate user interaction and experience, particularly the way that the information is presented to the user. This can be done by exploring new display technologies, such as holographic and transparent, instead of using a traditional Liquid-Crystal Display (LCD) like most products currently available. The holographic display presents differentiating graphics and interactive 3D (Three-dimensional) content, replacing the usual content based on static images with holographic videos/images. The transparent displays encourage the design of inclusive products inside homes, implementing the *Less is More* concept. This concept is based on "camouflaging" the device as much as possible inside the home environment, creating a smooth integration between the product and the home design while focusing on simplicity and functionality. This minimalist and feature-oriented product design goes in hand with the Smart Home paradigm and the tendency in households to increase their sensorization through the collection of technological devices throughout the house [4].

This document is a Master Dissertation that will study and evaluate the application of disruptive display technologies on Smart Home devices. The specific goals of this dissertation are the following:

- Study the UI and UX (User Experience) concepts to understand the new possibilities that these disruptive technologies of visualization and user interaction may bring along. This study also discusses how these concepts can be evaluated and tested.
- Survey the State of the Art (SotA) specifically focuses on holographic and transparent display technologies to learn how they usually work and how they are used in other products.
- Explore suitable hardware and software options to develop two prototypes: one holographic and one transparent. These prototypes shall resemble a Smart Home device, namely a smart thermostat, and their UIs should be built according to it.
- Evaluate the prototypes during their development and after it using a UI/UX study. This evaluation shall focus on relevant topics such as the effectiveness of information, quality of the graphics, organization of the information, ease of application and integration, and cost impact.

1.2 Dissertation structure

The remainder of this dissertation is organized as follows. Chapter 2 introduces the background concepts used in the project, namely, those related to UI and UX. Chapter 3 introduces the SotA technologies that will be considered in the project, namely the holographic and transparent displays. Chapter 4 presents the evaluation of the chosen hardware, as well as the software considered for developing the UI of the prototypes. Chapter 5 details the implementation made in this project. Chapter 6 presents the results of the UI/UX tests that were done with the users once the prototypes were finished. Finally, the dissertation presents its conclusions in Chapter 7.

Chapter 2

User Interface and User Experience

This chapter presents the concepts of UI and UX and the many ways they can affect the device and the user.

2.1 User Interface

UI serves as a bridge between the user and the machine, allowing the user to carry out actions on the device, visualize content, and receive feedback from previous actions. A good UI provides an enjoyable UX and allows the user to interact with software and hardware in a natural and intuitive way [5]. If user preferences and technology acceptance are not considered during the early development and research stages, that might imply a new design in later stages, leading to extra costs [6].

Graphical User Interface (GUI), augmented reality and voice control are some examples of interfaces for Smart Home devices [7]. Since most of the interactions with our daily devices happen through GUIs or voice interaction, it is expected that the same interfaces are the most common among Smart Home devices. However, in a Smart Home with so many interconnected devices, it may turn out to be inefficient to keep relying on GUIs and voice interfaces due to excess of information, which can lead to a steep learning curve and high cognitive load [8].

2.1.1 User profiling

To have a personalized system, it is essential to identify who the users are, as well as their interests and hobbies. User profiling relies on categorizing the users based on their behaviours and daily habits, which helps the system adapt to the end-user, leading to a UX improvement. Although user profiling has benefits like helping the system adapt to a user and their expectations, one could say that it comes with a significant disadvantage regarding the user's privacy [9].

The article in [10] divides Smart Home users into three major groups mainly based on their age, but also on their financial status: young adults (18 to 25 years old), active population (26 to 65 years old) and elderly (more than 65 years old). Due to their financial limitations, young adults usually do not have the chance to pay a large amount of money to own a sophisticated device, so they seek a low-budget one that can fulfil their basic needs. The active population is usually not that financially limited but has little to no time left after their daily work routines. The information presented must be simple and concise to avoid overflowing them with more information than required. Similar to the active population, the elderly are usually not financially limited, but they are not that familiar interacting with technology. Hence, the information presented must be easily readable, simple and concise.

2.1.2 Accessibility

Thermostat interfaces must be accessible to everyone, especially the elderly and people with physical and/or mental disabilities [11]. Accessible interfaces usually lead to easier user acceptance and satisfaction. The elderly typically need large fonts and buttons to interact with the devices [10]. Another important issue is the impact of animated objects (such as videos and animated GIFs) in UIs, which can distract the users and negatively affect the performance. This is true for the common users, but it is an even bigger issue for people with certain health conditions such as autism. Since about 1% of the population is thought to be on the autism spectrum, this is something that needs to be considered during the UI development [12]. A study was made [12] regarding the duration, completion rate and the number of errors made during a set of tasks with and without animation. The results of this study can be seen in Figure 2.1.

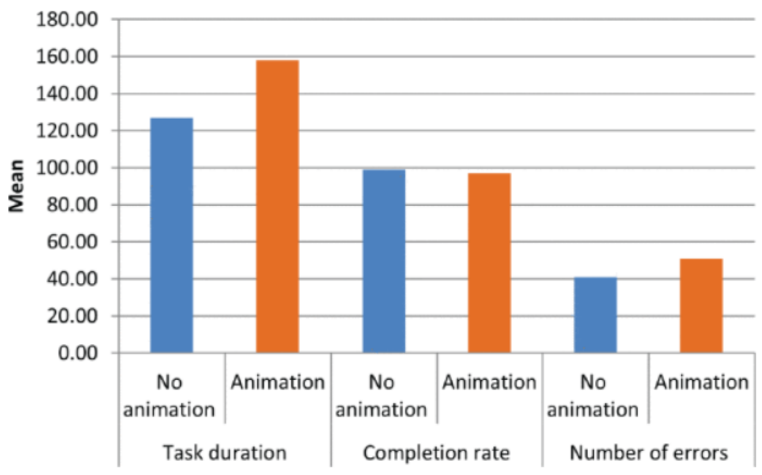


Figure 2.1: Tasks performance: animated vs. not animated [12]

2.1.3 Usability

According to the International Organization for Standardization, product usability refers to how effectively, efficiently and satisfyingly a user can achieve a specific goal in a particular context while using the product in question. Learnability is an important aspect of usability that asks the question: "Does this device provide an easy-to-learn operation?". Efficiency is another important aspect of usability that measures the number of steps needed to conclude a certain operation. Those same steps also need to be easy to remember. Some studies suggest that programmable thermostats have poor usability since people find it hard to understand them and are not motivated to overcome this difficulty. Although aesthetics are not a significant barrier while using the devices, if the interface looks complicated or overwhelming, it can limit some users when interacting with it [13].

As seen in Figure 2.2, these usability aspects, in combination with other factors such as practical acceptability (cost, compatibility and reliability of the device), will lead to the acceptability of the device by the users [13].

For the HVAC scope, the placement of the thermostat, its UI and the ability for the users to finish tasks such as scheduling are some of the factors that contribute to the usability of a thermostat. The placement of the device can be tricky since it is usually made by the user or the installer, adding too much diversity to the product usability tests. The same can happen for the user interaction with the interface since they are not required to actually interact with it. They can choose not to interact with it, making it harder for the usability tests to determine if the users do not know how to use the device or are simply choosing not to interact with it [11].

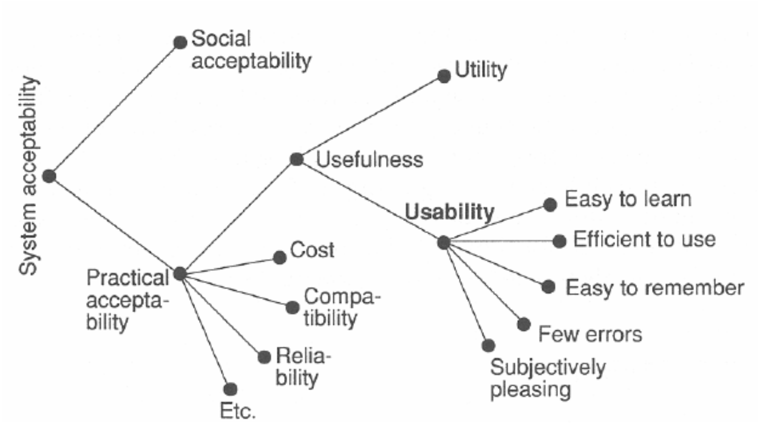


Figure 2.2: Nielsens' factors in system acceptability [13].

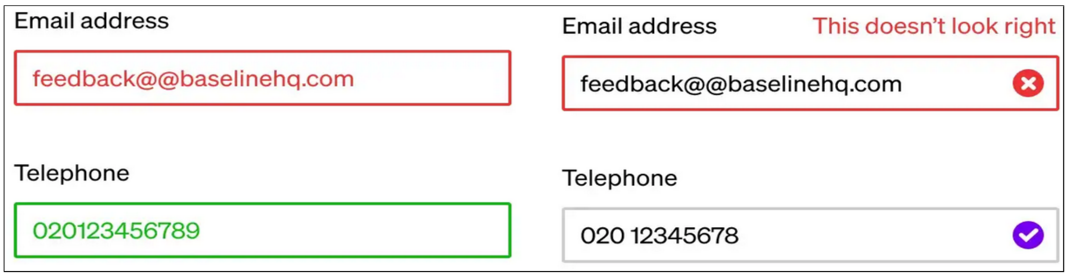
2.1.4 Interaction techniques

In the study reported in [14], a group of users was presented with several common interaction techniques such as voice control, gesture, GUI and even the lack of interaction in the form of automation. Although automation caught people's interest, they were uncomfortable with a fully automated home since they feared the system might not understand some specific house needs. Regarding voice control, although people find it very useful in certain scenarios where they have their hands full, they also find it risky since they state that the system could interpret single words inside a complete sentence as a command. As for gestures, the group felt that it was not a very natural way to interact with a system. The GUI happens to be the most trusted interaction technique among the group as they consider it to be efficient, simple and easy to interact with.

2.1.5 Color scheme

Matching colors increases the impact and appeal caused by the product by creating visual stimulation. In addition to enhancing the scientific nature of product design, color-matching schemes can also enhance the UX. In the field of art and design, color matching is a core research topic, and the color matching effect not only affects the efficiency of the design but also improves the aesthetics of the products, influencing the purchase intention and UX [15].

Some colors are associated with certain scenarios and feelings, such as green and red, which tend to be associated with good and bad scenarios, respectively. Another problem regarding those two colors is that they look similar to colorblind people. Yet, multiple UIs rely only on these colors to express those good/bad scenarios, making it difficult for those users to understand the information that the interface is passing, as seen in Figure 2.3(left). To avoid these situations, one can use visual indicators like text and/or icons to illustrate the issue in question, like seen in Figure 2.3(right), or use colorblind-friendly hues [16].


Email address
feedback@@baselinehq.com

Telephone
020123456789

Email address This doesn't look right
feedback@@baselinehq.com ✕

Telephone
020 12345678 ✓

Figure 2.3: Validation and error being represented by simply green and red colors respectively (left) Same scenario but also using indicators (right) [16].

Dark-themed interfaces are gaining more and more popularity in recent years. These themes are characterized by a light font (usually white) displayed on a dark background (typically black), leading to better legibility, aesthetics, energy savings and faster user response [17].

2.2 User Experience

Even though UI is a crucial part of the design, it is important to distinguish it from the UX since an excellent UI can have a UX that is far from optimal. UX is not just about giving customers what they want. Instead, it is based on the user's internal state, the system characteristics and the context of the interaction [18].

Interactive products form an important part of UX, which is a sub-category of the overall concept of experience. Unlike these products, the UX is intangible and volatile, and it is about the experience of acting through a product at the moment of use. Apple's iPhone, which features an aesthetic of interactivity that is unique but fulfils the same tasks as any other smartphone, remarkably popularized this notion of experience. Many new technological innovation devices do not feature new algorithms, materials or fancy interfaces. Instead, they excel because they understand how objects can be used to create and shape experiences, feelings and situations [19].

UX is evaluated through tests with the users. There is a variety of ways to conduct a UX evaluation phase. UX tests can be undertaken to evaluate essential product factors, while UX questionnaires serve as a way to evaluate the interactive products, measuring aspects like attractiveness, stimulation and novelty. Visual aesthetics of websites inventory tests evaluate the visual aesthetics of the product [20], focusing on aspects such as colorfulness, simplicity and diversity. UX can be based upon the user's interactions, sensations, thoughts, feelings, and meaning given to a situation. Still, it also focuses on the user's emotions and affective states while using the product [21]. As seen in Figure 2.4, UX can be decomposed into many categories, such as efficiency (e.g. time to complete the task), effectiveness (e.g. number of errors during the task), emotion (e.g. how the user feels while doing a task), information load (e.g. the amount of information presented to the user must not be overwhelming) and intuition (e.g. the interactions with the product must feel natural). Data regarding UX tests can be collected before, during and after the tests through questionnaires [22].

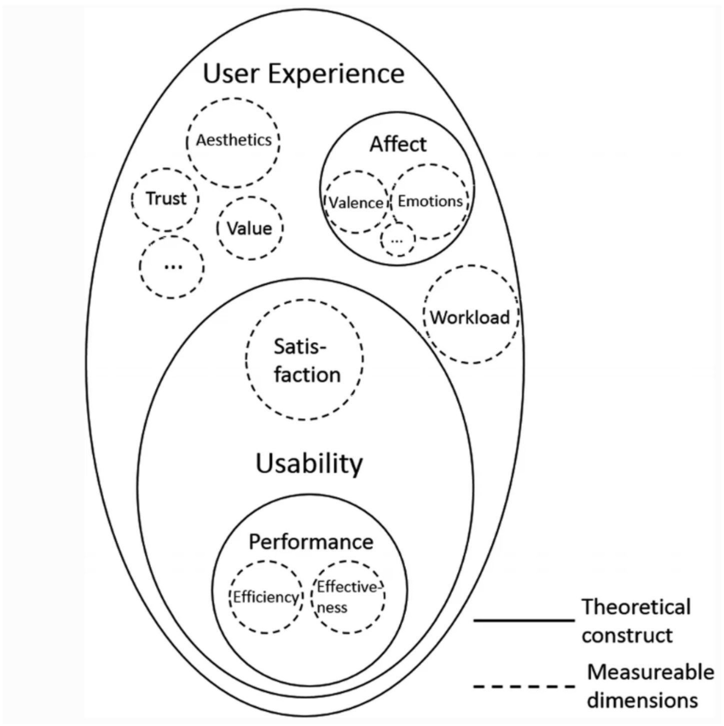


Figure 2.4: UX decomposition [21]

2.3 Summary

This chapter provided some background information on what are UI and UX concepts and their main aspects: user profiling and how it can help the product adapt to the user; the accessibility and usability, and how both are essential factors in the acceptability of a product; the interaction techniques and color schemes and how they influence the UX. This chapter also provided solutions for correctly evaluating a product's UI and UX. These solutions will be considered when evaluating the prototypes, namely the UX tests and questionnaires in Chapter 6. The UI/UX concepts will also be taken into account from the beginning of the prototypes development to promote their acceptability and improve the UX.

Chapter 3

Display technologies for HVAC

The main objective of this project is to build UIs for HVAC systems using disruptive and innovative technologies. This chapter starts by approaching a historical perspective on the evolution of the UI of thermostats. Afterwards, it presents related state-of-the-art technologies, namely holographic and transparent displays, as the main alternatives to be considered during the project.

3.1 Thermostat UI evolution

Thermostats are common parts of HVAC systems and will be the focus of this chapter. Even though home heating traces back to the Roman Empire, it was not until the 17th century that the dutch Cornelis Drebbel created the first mercury thermostat. Following him, more thermostats were developed, but it was not until 1906 that HoneyWell developed the first commercial thermostat Jewell: an analog dial thermostat that can be seen in Figure 3.1 [23]. This thermostat only displayed the desired temperature.

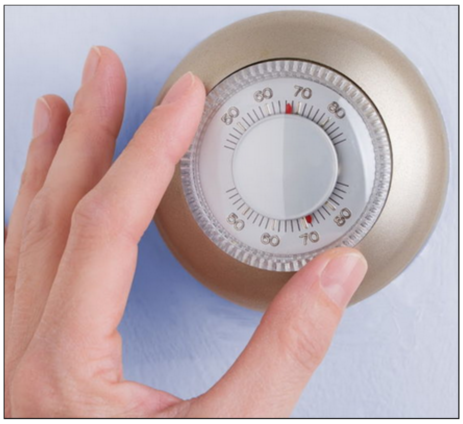


Figure 3.1: Jewell: the first commercial thermostat [23]

In the 80s, new digital thermostats were launched into the market. These digital thermostats can be seen in Figure 3.2 and not only display the current and desired temperature but also the date and time, allowing users to schedule the temperature for the entire week in advance. Although the idea was good, most people could not program these thermostats as intended due to their complicated UI [23].

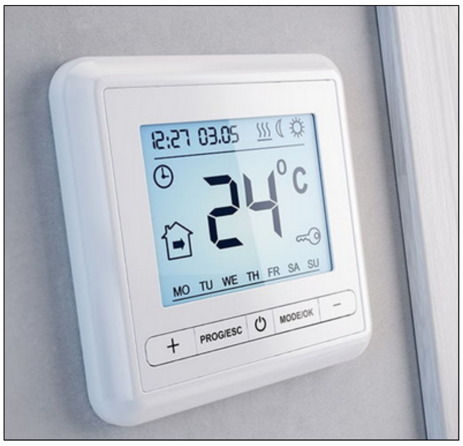


Figure 3.2: Digital thermostat [23]

Smart thermostats emerged as a solution to the eventual downfall of programmable thermostats, allowing users to interact with the device through familiar interfaces (e.g. smartphone applications). This new type of interaction provided more useful information in the display, allowing the user to better comprehend the features of the device. Web-based interfaces also solved the issue users had with the scheduling part of the thermostat, making it easier to understand and complete the task. To avoid making the same issues that the programmable thermostats had, the smart thermostats must prioritize passing valuable information to the user and have an effective but easy-to-understand layout that does not overwhelm the user. An example of a smart thermostat interface can be seen in Figure 3.3 [11].

Ecobee launched the first smart thermostat in 2007, combining WiFi with the already programmable thermostats, allowing users to control the heating and cooling of their homes without needing to be close to the controller [23]. Nowadays, smart thermostats can display a variety of information: date and time, day of the week, current and target temperature (inside), schedule mode, outside temperature, weather forecast, humidity, and system status, among others [13].

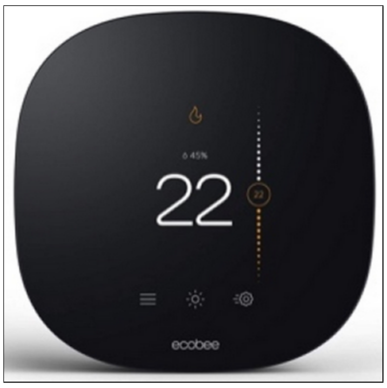


Figure 3.3: Ecobee smart thermostat [11]

3.2 Smart thermostats UI evaluation

To better understand the different types of UI found in commercially available smart thermostats, a comparative study focusing on the displays used by these devices was carried out. A summary of this evaluation is presented in Tables 3.1 and 3.2.

Being an HVAC system producer, Bosch Thermotechnology has different smart thermostats in the market. One of those devices is the EasyControl CT200[1]. This device has a 2.5 inches glass touchscreen, which is nothing more than a glass layer on top of an LCD, and it can also be controlled through a mobile application. As for the UI, CT200 has menus displaying: weather forecast, outside temperature, room temperature, humidity, scheduling, and connection to the app, among others.

Another Bosch product worth mentioning is the digital thermostat Room Thermostat II[2]. This device has a 3.17 inches matrix display, similar to a segmented display but does not have a pre-defined design, meaning it is more flexible regarding what it can show. Interacting with it through voice control and a mobile application is also possible. As for the UI, this device has menus displaying: room temperature, humidity, and operation mode, among others.

Google Nest developed the Nest Thermostat[3]. This device has a 3.3 inches touch LCD, and it is also possible to interact with it through voice control and mobile application. As for the UI, this device has menus displaying: time, date, outside temperature, room temperature, humidity, scheduling, operation mode, and energy consumption, among others.

Johnson Controls launched the smart thermostat GLAS Thermostat[4] in 2018, yet this product was later discontinued. This device has a 5.91 inches transparent Organic Light-Emitting Diode (OLED) with an integrated touch controller. It is also possible to interact with it through voice control and mobile application. As for the UI, this device has menus displaying: time, outside temperature, room temperature, maximum and minimum temperature, humidity, wind speed, air quality, and scheduling, among others.

Ecobee has one of the most known smart thermostats in the market: Smart Thermostat Premium[5]. This device has a 3.5 inches touch LCD, and it is also possible to interact with it through voice control and a mobile application. As for the UI, this device has menus displaying: time, weather forecast, humidity, room temperature, outside temperature, air quality, and scheduling, among others.

[1] https://www.bosch-easycontrol.com/gb/en/easycontrol/overview/
[2] https://www.bosch-smarthome.com/uk/en/products/devices/room-thermostat/
[3] https://store.google.com/us/product/nest_thermostat?hl=en-US
[4] https://glas.johnsoncontrols.com/
[5] https://www.ecobee.com/en-us/smart-thermostats/smart-thermostat-premium/

Table 3.1: Control characteristics of existing devices

Model	Type of Screen	Screen Size	Touchscreen	Mobile App	Voice Control
EasyControl CT200	LCD	2.5”	✓	✓	✗
Room Thermostat II	Matrix Display	3.17”	✗	✓	✓
Nest Thermostat	LCD	3.3”	✓	✓	✓
GLAS	Transparent OLED	5.91”	✓	✓	✓
Smart Thermostat Premium	LCD	3.5”	✓	✓	✓

Table 3.2: User Interface characteristics of existing devices

Model / Features	EasyControl CT200	Room Thermostat II	Nest Thermostat	GLAS	Smart Thermostat Premium
Time	✗	✗	✓	✓	✓
Date	✗	✗	✓	✗	✗
Room Temperature	✓	✓	✓	✓	✓
Outside Temperature	✓	✓	✓	✓	✓
Humidity	✓	✓	✓	✓	✓
Air Quality	✗	✗	✓	✓	✓
Weather Forecast	✗	✗	✓	✓	✓
Scheduling	✗	✗	✓	✓	✓
Connectivity	✓	✗	✓	✓	✓
Operation Mode	✗	✗	✓	✓	✓
Energy Consumption	✗	✗	✓	✓	✗

One should note that Table 3.2 refers to the features present in the displays of the devices. Some features, like energy consumption, are only available in the mobile app on some of the devices.

This analysis provided valuable insight into what to look for in the search for the hardware for the new prototypes to be developed in the project. We were able to estimate the size of the display for the prototype as well as other attributes, such as touchscreen and connectivity (mobile application), which influence the hardware choice. The analysis also provided crucial information for designing the UI of the prototypes since we now have information on multiple features that can be present in it.

3.3 Holographic display

A Holographic display is a unique screen capable of projecting colored 3D images through various techniques [24]. These displays can provide all depth cues perceptible to the human eyes [25]. They can be used in many innovative applications from different fields that require the use of images. In particular, we show next several use cases that have been reported in the literature in the domains of retail, automotive systems, smart homes and video conferencing.

3.3.1 Use cases

3.3.1.1 Retail

As in [26], a holographic display can provide a high-tech shopping experience through a tangible interface, allowing customers to interact with virtual products using a combination of an auto-stereoscopic 3D display, mid-air haptics and finger tracker as seen in Figure 3.4.

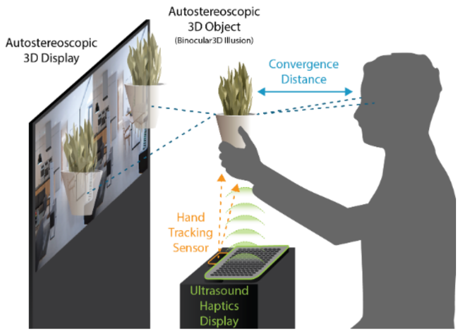

Autostereoscopic 3D Display
Autostereoscopic 3D Object
(Binocular3D Illusion)
Convergence Distance
Hand Tracking Sensor
Ultrasound Haptics Display

Figure 3.4: Concept diagram of user-product interaction using a holographic display [26]

3.3.1.2 Automotive systems

The automotive industry can use holographic displays for a different type of application: Heads-Up Display (HUD) for cars [24]. Originally, a HUD was intended to be a system used in aircraft to aid the pilots visually by allowing them not to change their visual sight to collect information [27]. It is known that 94% of car crashes were caused by human error. This is expected since drivers are exposed to an immense amount of visual and audio information, either from the car usage or devices like navigation aids [28].

Other options like auto-stereoscopic displays and light-field displays are capable of improving the UX. However, since the recommended depth range for the auto-stereoscopic display is limited due to factors like focal depth and accommodation-vergence conflict (AVC), the auto-stereoscopic display is not ideal. Light-field displays are also not the best solution because, although they are not affected by AVC, which can cause eyestrain, fatigue, headaches and motion-sickness, their image resolution is very poor when seen at a large distance from the screen. On the other hand, the holographic displays do not have any of these limitations. [29]

Holographic displays can avoid losing image resolution while also providing the full range of natural depth cues by reconstructing the amplitude and phase of light of a 3D scene [29]. These displays can allow the drivers to become aware of hidden road obstacles in full depth within a 360º view, helping them react accordingly. These displays can have algorithms that make it easier to replay the field in full color without the need for glasses and avoid visual fatigue by simply adding infinite layers. [28]

3.3.1.3 Smart Homes

Since Smart Homes are becoming a popular topic, it is reasonable to think that the popularization of holographic displays in household devices may occur in order to provide people with a high-tech lifestyle [30]. There is a 2018 project called HoloSensor [31] that improves the visualization of data from sensors related to Smart Home through holography. Multiple users can simultaneously interact with the data in real time using the augmented reality headset Microsoft HoloLens to view the holograms.

3.3.1.4 Video conferencing

The recent COVID-19 pandemic caused a massive rise in video conferencing solutions (such as Zoom and FaceTime) due to the transition to remote work, education and socialization. 3D telepresence systems have not yet found any widespread adoption [32].

A 2022 project called HoloKinect [32] is an end-to-end 3D video conferencing solution. This project uses a Microsoft Azure Kinect for red, green, blue and depth capture and a Looking Glass Portrait display for 3D visualization at each end of the two-way video conference, enabling a hologram-like effect as seen in Figure 3.5.

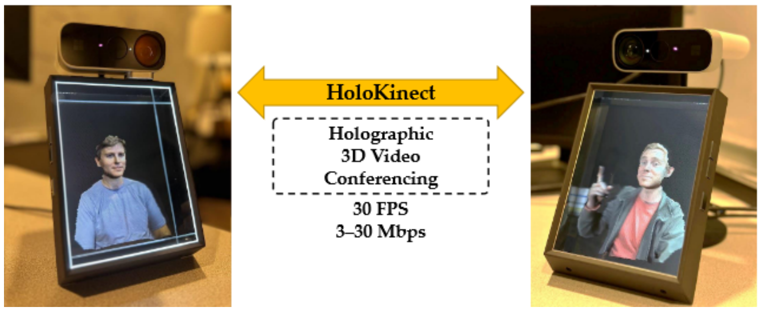


Figure 3.5: Live two-way HoloKinect 3D video call on a pair of Looking Glass Portraits [32]

3.3.2 Holography with light-field displays

The term light-field refers to a function that uses points in space to quantify the amount of light travelling in any direction [33]. A light ray is characterized by five parameters: x, y, z, (θ) and (ϕ), being the last two the elevation and azimuth angles, while the remaining ones the cartesian coordinates, as shown in Figure 3.6. Given that any image device can capture the plenoptic function l(x, y, z, θ, ϕ), it is possible to obtain a projected image at any location by selecting the light rays from (x, y, z, θ, ϕ). A regular digital camera captures the light rays at the viewing locations, but since only the outside of the object matters, it can also capture the plenoptic function. The function can be reduced from 5D to 4D by the fact that radiance does not change along light rays. There is a variety of methods to parameterize light rays in 4D static light fields, being the two plane parameterization one of the most common approaches. In this approach, a light ray l(u,v,s,t) is parameterized by its intersections with two parallel planes (u,v) and (s,t) as shown in Figure 3.7.

Every single light ray that goes through a point in the (u,v) plane will have to correspond to an image taken by a camera at that location, parameterized by the image coordinate (s,t) [34].

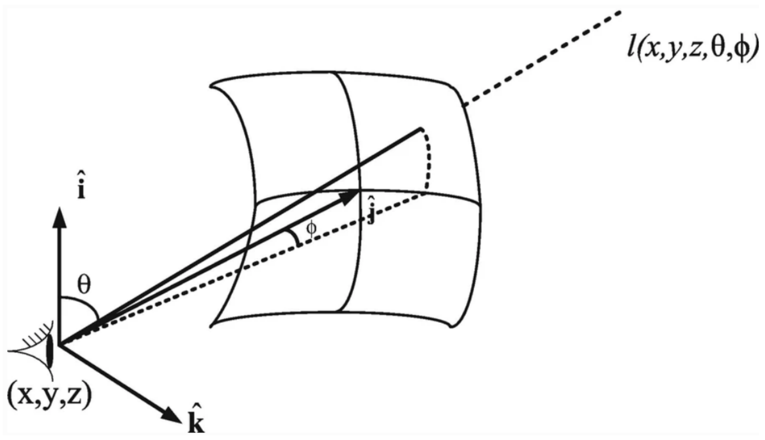


Figure 3.6: Light ray [34]

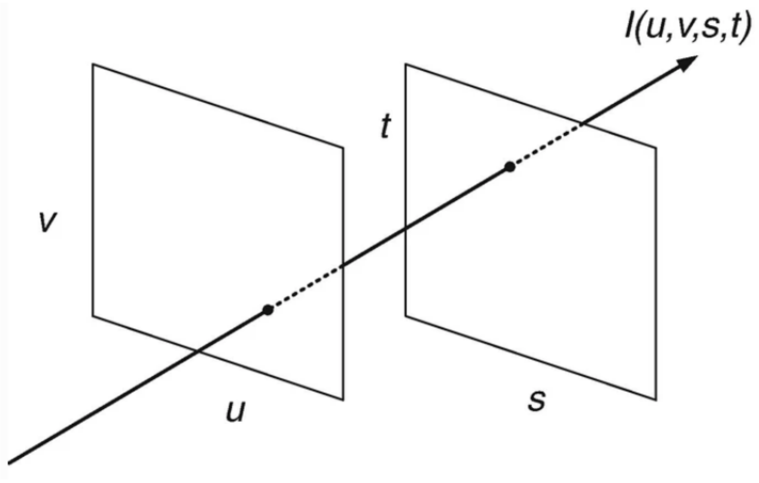


Figure 3.7: Two plane parameterization [34]

There are realistic ways to visualize and manipulate 3D objects/scenes without relying on reconstructing its 3D model, being Image-Based Rendering (IBR) one of them. This method was developed to avoid the difficulties around achieving photo realism using 3D and model-based rendering. A large quantity of samples is required for light-fields to render scenes/objects without its 3D models, but there is barely any need for geometry information. Compared with the 3D model rendering alternative, this method has better image quality for complex real-world scenes/objects and a significantly reduced computational effort [35]. Using Fourier Transform can be an added value while designing IBR systems since the amount of samples and geometry information needed can be obtained through this process [34].

3.4 Transparent display

Transparent displays can present images/videos on it while simultaneously allowing the users to see the view behind the display [36]. Transparency is the perception of the human eye to the brain, which comes from the brightness of the object behind the display [37]. These displays can be used for advertising purposes, such as shop windows that show realistic images/videos, or to make the illusion of an object floating in thin air, in the Automotive industry, such as HUDs, among others [36]. We briefly discuss the next two use case domains reported in the literature, namely for Smart Home and automotive systems.

3.4.1 Use cases

3.4.1.1 Smart Homes

Johnson Controls developed a smart thermostat called GLAS that uses a transparent OLED touchscreen. The users can interact with it via the touch display or voice recognition since it works with Amazon Alexa and Google Assistant. The main quality of this product is the ease with which it integrates into the house without disturbing its design.

There is also an exciting way [38] to make use of the home windows as transparent displays since it is an unexplored technology. The home window provides a connection between the home and the external world, having the means not only to work as a transparent screen but also as a mixed-reality display.

3.4.1.2 Automotive systems

The transparent display is also a good option in the automotive industry, like the holographic display mentioned before (basically for the same reasons). In this 2016 article [39], a low-cost HUD prototype was developed to help drivers stay focused on the road while using navigation systems like GPS (Global Positioning System). Usually, a person takes 1.2 to 1.5 seconds to observe the scene displayed on the device and react to it. This HUD allows the user to keep his eyes focused on the road while obtaining the GPS scene information.

3.4.2 Operation of a transparent LCD

Transparent LCDs are one of the most popular displays worldwide because of their process compatibility with existing liquid crystal displays and their low cost. The power consumption of transparent LCDs is relatively high since they are equipped with high-powered illumination devices behind the display to provide a strong backlight for the user to see through the display. The power consumption is inversely proportional to the transmittance of the LCD, meaning that if the LCD has a high transmittance, the power of the rear light source can be lowered. The light passing through the liquid crystal panel will pass through the lower polarizer, array substrate, liquid crystal, color film substrate and upper polarizer, leading to a large amount of light loss in each layer (color film subtract has about 70% of light loss) [37].

There are transparent LCDs using Polymer-Dispersed Liquid Crystal (PDLC) technology that have qualities such as high visibility, transparency, and low consumption. A PDLC cell consists of many micro PDLC droplets, each with different orientations and configurations. These droplets consist of nematic liquid crystal and polymer matrix that can be altered by an external electric field and control the scattering and transmittance. When the voltage is off, meaning that no external electric field is being applied, the refractive index of PDLC droplets quickly varies from droplet to droplet, resulting in an opaque state (high scattering). This state is represented in Figure 3.8(left), and as mentioned before, since the voltage is off, the backlight can not go through the pixel electrode, reflecting only scattering light. When the voltage is on, meaning that an external electric field is being applied, the refractive index is close to the refractive index of the PDLC droplets because the droplet symmetry axis aligns with the electric field. This allows the light to go through each pixel electrode without reflecting the scattering light, leading to a transparent state as seen in Figure 3.8(right) [40].

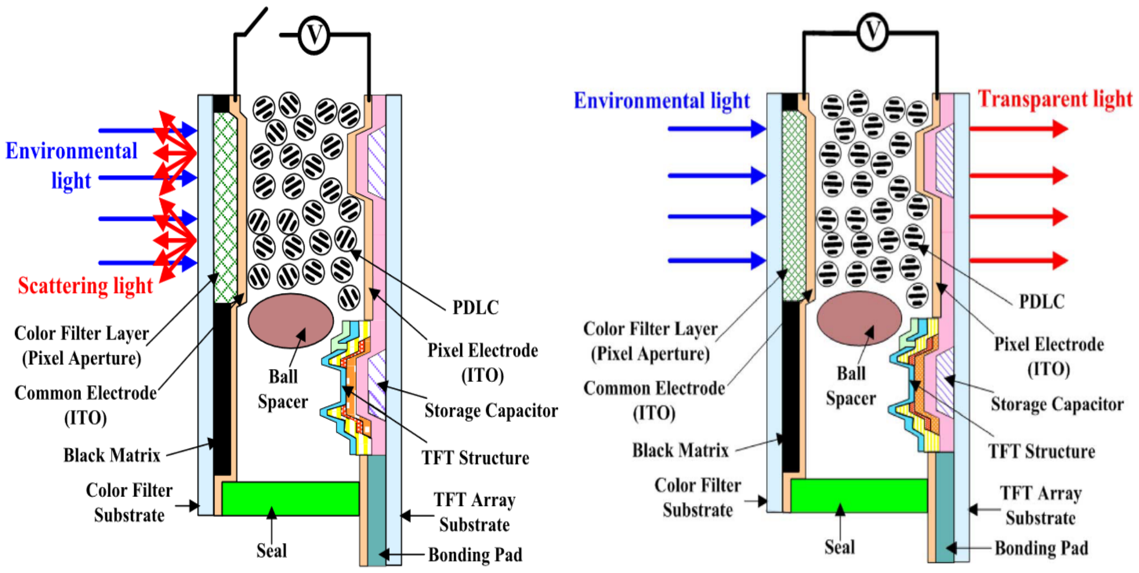


Figure 3.8: LCD states: Opaque - voltage off (left) Transparent - voltage on (right) [40]

3.4.3 Operation of a transparent OLED

OLEDs usually have emissive organic layers between an anode and cathode deposited on a substance. OLEDs can emit light from the pixels through a thin organic film layer, unlike LCDs which depend on a backlight to create light. A layer of organic material is placed between the conductors (anode and cathode), and then this whole block is placed between a seal and substrate, which are the top and bottom glass plates. When an electric field is applied to the conductors, a bright light is produced from the organic material in any color and is emitted between 0 to 100%. The color is obtained using three sub-pixels in blue, green and red. The transparency of these displays comes from their components' transparency. When the displays are turned off, they become up to 85% as transparent as the substrate used. When turned on, they allow the light to pass in both directions [41].

3.4.4 Operation of a transparent segmented display

Segmented displays[6] have individual segments that can be lit up individually, thus forming the desired content on the display. These displays excel in brightness and transparency, making them thrive in both indoor and outdoor environments. The display content needs to be designed in advance by the customer before being manufactured since the content can not be changed: it can only display what was pre-designed.

3.5 Summary

This chapter presented a brief historical analysis of UI technologies for Smart Homes, showing that current interfaces do not use disruptive technologies. Instead, they are mostly still using regular LCDs, thus reinforcing the importance of this project. However, some projects are already experimenting with disruptive technologies in the Smart Home field. Therefore, this chapter also included a study of two disruptive display technologies, holographic and transparent displays, providing a perspective of how they work and their various use cases. This study of these technologies provided valuable insight, which was helpful when choosing the hardware for this project in Chapter 4.

[6] https://www.lumineq.com/blog/matrix-vs-segmented-displays

Chapter 4

Choosing the development platform

This chapter focuses on choosing the hardware that meets all the criteria for the proposed goals, as well as the software to interact with the selected device and create a proper UI.

4.1 Hardware evaluation

The main objective of this hardware research was to find a transparent and a holographic display that best suit this use case. Therefore, some requirements must be taken into account during the hardware selection. The size is an important aspect of our Smart Home use case since it needs to be adequate for the content that needs to be shown to the user. It must be large enough to allow the user to see and read the displayed information clearly, but not so large that the information is dispersed across the screen. The price is also an influential factor in the hardware choice. The display availability and supplier are also decisive factors in ensuring that the hardware can be purchased and received in time for the development of this work. Touchscreen is desired but not imperative since buttons or voice activation are viable options for navigating between menus. It would be interesting if the display interaction could be done using a Single-Board Computer (SBC) such as a Raspberry Pi[1], in order to ease the integration and prototyping. So it would be good for the displays to have generic connectors like USB and HDMI largely supported by devices like the ones mentioned before. The main displays found and analyzed are in Tables 4.1 and 4.2.

Table 4.1: Holographic Displays

Supplier	**Model**	**Price (€)**	**Size**	**Touchscreen**	**Connectors**
Looking Glass Factory[2]	Looking Glass Portrait	378,97	7.9”	✗	USB-C, HDMI
Hypervsn[3]	SOLOM	3699	22”	N/A	N/A

[1] https://www.raspberrypi.com/products/

[2] https://lookingglassfactory.com/looking-glass-portrait

[3] https://hypervsn.com/hypervsn-solo-combo

Table 4.2: Transparent Displays

Supplier	**Model**	**Type**	**Price (€)**	**Size**	**Touchscreen**	**Connectors**
LG[4]	LG-55EW5F-A	OLED	19999	55"	✓	N/A
Lumineq[5]	ELT15S-1500	Segmented	195	3"	✓	Mini-USB
Crystal Display[6]	LCD-121-TRNV2	LCD	1132,16	12.1"	✓	VGA,HDMI,USB
Crystalfontz[7]	128x56 TOLED	OLED	24,77	1.51"	✓	Micro-USB
Shenzhen Pie[8]	Customized model	LCD	233	7"	✓	HDMI,USB

The holographic display from Hypervsn was the SOLOM model, which is a fan-like display. It is unsuitable for the use case we are exploring since it could be dangerous to have a fan-like display as a Smart Home interface (e.g. could get the attention of children, and create cold air flows around the house, among other issues). The Looking Glass Portrait developed by Looking Glass Factory is far less expensive than the other option, and it fits the use case: adequate size, has the required connectors and has an incorporated Raspberry Pi 4, which could help in the development of the prototype.

Concerning the transparent displays, getting a transparent OLED would be ideal since these do not demand a strong backlight. LG has some good options that suit this use case, but they are expensive, and the size is just too big. Crystalfontz has a much more affordable option, but they do not make bigger sizes. Lumineq has a model ELT15S-1500 which is a bit small but could be doable, although the problem is that it is a segmented display, which does not fit this use case because it is not ideal to be constrained to a pre-defined design given by a set of segments. That left us with the LCD options, and Shenzen Pie has a very customizable set of products, and it is far less expensive than Crystal Display. Thus, the chosen option was the Shenzen Pie transparent LCD.

4.1.1 Looking Glass Portrait

The holographic display of choice for this project is the Looking Glass Portrait which can be seen in Figure 4.1. It produces holograms by providing 45 to 100 discrete views of a 3D scene and presenting them across a view cone 58° wide. This arrangement of views tricks the user's sight into seeing 3D objects through the use of parallax and stereo vision. These methods allow each eye of a user to simultaneously see multiple views, providing a smooth transition between views instead of a rough one when the user moves their head around the display [42].

[4] https://www.lg.com/us/business/oled-displays/lg-55EW5F-A
[5] https://www.lumineq.com/products/transparent-displays/elt15s-1500
[6] https://crystal-display.com/products/transparent-lcd/
[7] https://www.crystalfontz.com/product/cfal12856a00151b-128x56-transparent-oled-screen
[8] https://en.piedisplays.com/

Figure 4.1: Looking Glass Portrait[9]

Looking Glass software is responsible for converting various existing 3D environments into quilt images. Quilts are an image standard that Looking Glass uses to produce 3D experiences, applicable to pictures and videos. Quilts store frame-by-frame data in standard media formats like mp4, gif, png, jpg and mov in an efficient and compact way. Each tile in the quilt refers to a Two-Dimensional (2D) image of a particular scene. The bottom-left tile of the quilt is the leftmost view of the scene, and the top-right tile is the rightmost [43]. In Figure 4.2, it is possible to see an example of a quilt with 45 views.

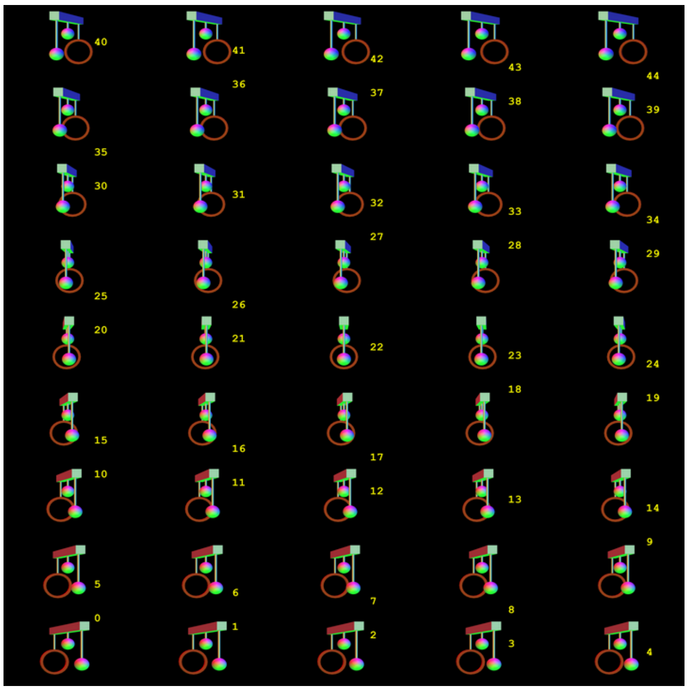


Figure 4.2: Example of a Quilt [43]

[9] https://lookingglassfactory.com/buy-looking-glass-portrait#additional-info

The holographic display is delivered in a package containing a Looking Glass Portrait, a USB-C cable, an HDMI cable and an adapter with three different conversion heads so that it can be used in various combinations of frequency and voltage (depending on the country). The Portrait has a total of five buttons on its right side: one to turn the device ON and OFF, another to turn the edge lighting ON and OFF and three capacitive buttons to interact with the device (*Back*, *Forward* and *Play/Pause*) that can be seen on Figure 4.3.

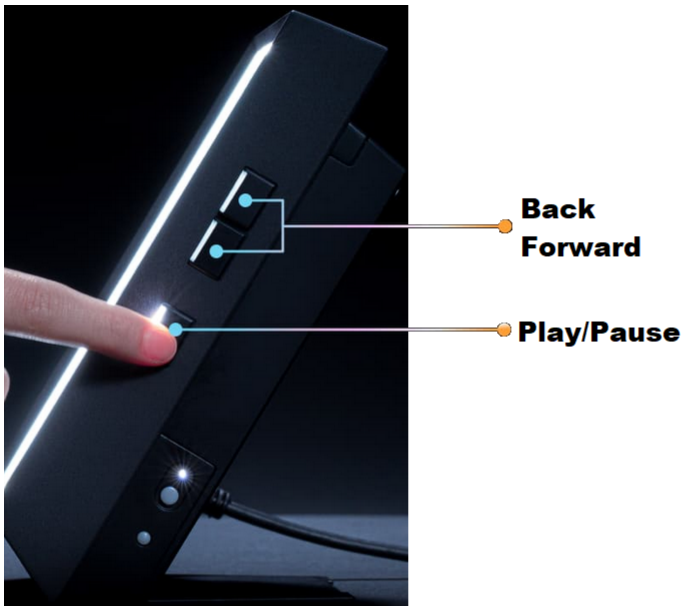
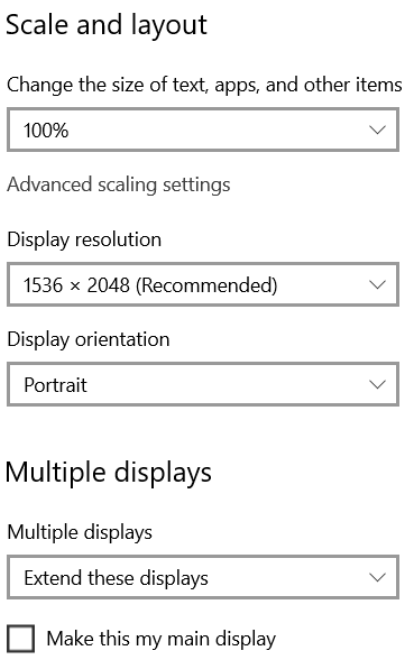


Figure 4.3: Looking Glass Portrait buttons[10]

4.1.1.1 Operation modes

The device has two operation modes: Standalone and Desktop. For the device to be in Standalone mode, it must be plugged into the socket using the USB-C cable and the proper adapter. This mode allows the device to loop through a pre-configured "playlist" of images and videos without being connected to a computer. The previously mentioned capacitive buttons allow the user to go back and forth in the playlist or even pause it. This playlist has to be configured while in Desktop mode before using the Standalone mode, otherwise the playlist will be empty. For the device to be in Desktop mode, it must be plugged into the computer using the USB-C and HDMI cables. It is essential to check the display settings on the computer after connecting this holographic display since it might come with some default configurations that need to be changed, such as getting the display to extend the desktop, making sure that the resolution is set to 1536x2048 with display scaling set to 100% and using a Portrait orientation as seen in Figure 4.4.

[10] https://lookingglassfactory.com/looking-glass-portrait


Scale and layout

Change the size of text, apps, and other items

100%

Advanced scaling settings

Display resolution

1536 × 2048 (Recommended)

Display orientation

Portrait

Multiple displays

Multiple displays

Extend these displays

☐ Make this my main display

Figure 4.4: Configurations of the holographic display

When in Desktop mode, it is possible to edit the content shown on the device through various 3D software and see the changes made in real-time. Although the display is pretty much a "plug & play" equipment, meaning that it is pretty simple to start using it after being plugged in, there is one demand: it needs to have the software Looking Glass Bridge installed. Looking Glass Factory provides this software that facilitates communication between the computer and the display as described in Figure 4.5. Using this software is as simple as opening the executable file and letting it run in the background.

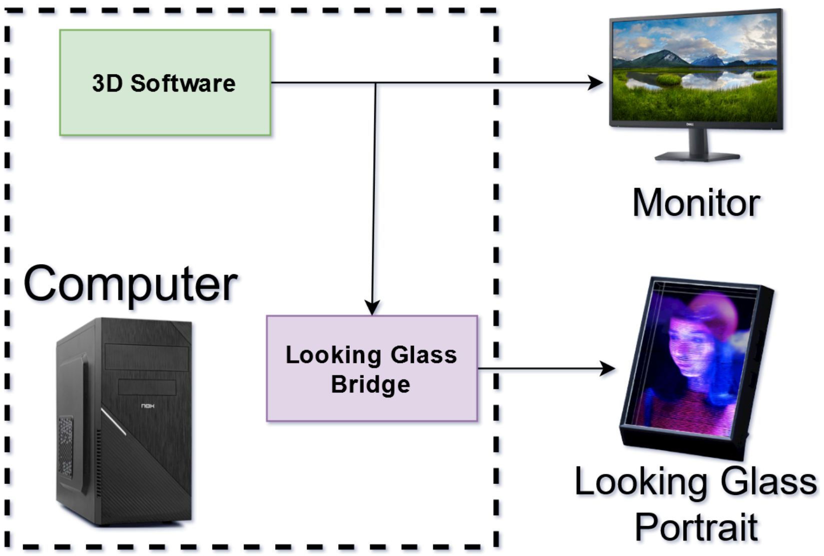


Figure 4.5: Looking Glass Bridge diagram

4.1.2 Box-shaped transparent display

The transparent display of choice for this project is the customized box-shaped transparent LCD from Shenzhen PIE that can be seen in Figure 4.6. This sample has a 7 inches touchscreen as well as HDMI and USB ports. Its working method is similar to the one described in Subsection 3.4.2: it has a strong backlight inside the box for the user to see through the display.

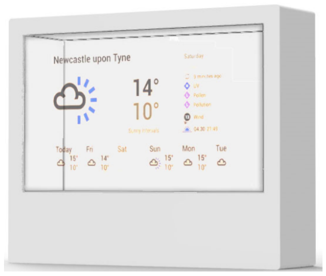
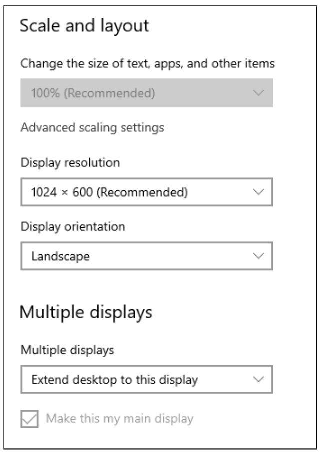


Figure 4.6: Box-shaped transparent LCD sample

This display comes in a package containing a box-shaped transparent display with an HDMI and USB port, and a USB cable attached to control the touchscreen. It also comes with an HDMI cable and a power supply cable.

4.1.2.1 Operation mode

This device must be plugged in using the power supply cable and connected to a computer or an SBC via HDMI and USB (for the touch controller). Similarly to the holographic display, it is essential to check the display settings on the computer after connecting the display since it might come with some default configurations. Make sure to set the display to extend the desktop. The resolution needs to be set to 1024x600 with display scaling set to 100% and using a Landscape orientation as seen in Figure 4.7. These are the only steps needed to use the display. After that, the display will be connected as an external monitor. Note that the display is set as the main display. This configuration is mandatory for the prototype to work correctly.



Scale and layout

Change the size of text, apps, and other items

100% (Recommended)

Advanced scaling settings

Display resolution

1024 × 600 (Recommended)

Display orientation

Landscape

Multiple displays

Multiple displays

Extend desktop to this display

☑ Make this my main display

Figure 4.7: Configurations of the transparent display.

4.2 Software platform

The software reusability between the holographic and transparent displays was taken into account while looking for a software platform suitable to develop the UI for the prototypes. For a software to be compatible with Looking Glass Portrait, it must generate a borderless fullscreen window at a specific position, distort the projection matrix of the camera, render multiple views to a texture and apply a shader to the texture. With that in mind, several software was analyzed, and those options can be individually seen below and summarized in Subsection 4.2.7.

4.2.1 Qt

Qt ("cute") is a powerful software for cross-platform design and development of GUI applications, meaning that these applications can run on multiple platforms such as Windows, Linux, macOS, iOS and Android. The framework and its applications can be compiled in standard C++ compilers. Qt has various tools that help build GUIs: widgets, layouts and graphic libraries. This software also has its own free integrated development environment (IDE) that provides intelligent code completion, syntax highlighting and a debugger. Qt has an active community that maintains a vast and well-detailed documentation, making it easier to learn and use.

Qt has direct 3D support (QtQuick3D, QtDesign 3D), and it looks able to support the Looking Glass Portrait at first glance. The only issue is that it does not have a connection between Qt and the Looking Glass Portrait, meaning that Qt does not recognize the display nor has all the configurations necessary to properly interact with it. Without this connection, it was not possible to look further into this approach. The connection would need to be made from scratch, which would be too time-consuming for this project. Therefore a new approach had to be found, preferably a software that already has this connection.

4.2.2 GTK+

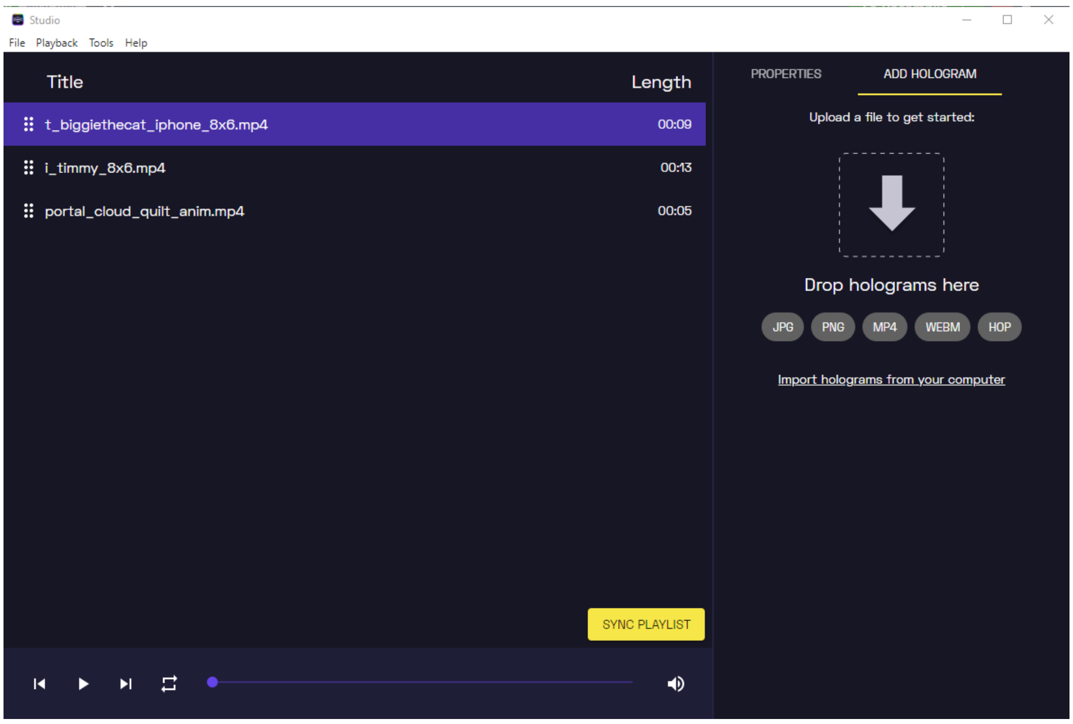
GTK+ is a cross-platform software used to create GUIs that can be written in commonly known object-oriented languages such as Python, JavaScript, and C++, to name a few. Similarly to Qt in many other aspects, GTK+ has the same tools to help build GUIs as well as detailed documentation. Being an object-oriented software makes it easier to reuse code and create custom widgets. GTK+ is also able to render 3D objects, but it does not have direct support from the software like Qt has. Similarly to Qt, it has the same issue of not having a direct connection between its software and the Looking Glass Portrait.

4.2.3 Electron

Electron is another software focused on building desktop apps. It uses web technologies such as HTML, CSS, and JavaScript. Like Qt and GTK+, Electron has vast documentation to ensure users can learn how to use the software. Although Electron can render 3D objects, it lacks direct support from the software in the same way GTK+ does. It also shares the same issue of not having a direct connection between the software and the Looking Glass Portrait.

4.2.4 Looking Glass Studio

Looking Glass Factory provides a 3D software to edit the playlist used in Standalone mode. This software, Looking Glass Studio, is seen in Figure 4.8.

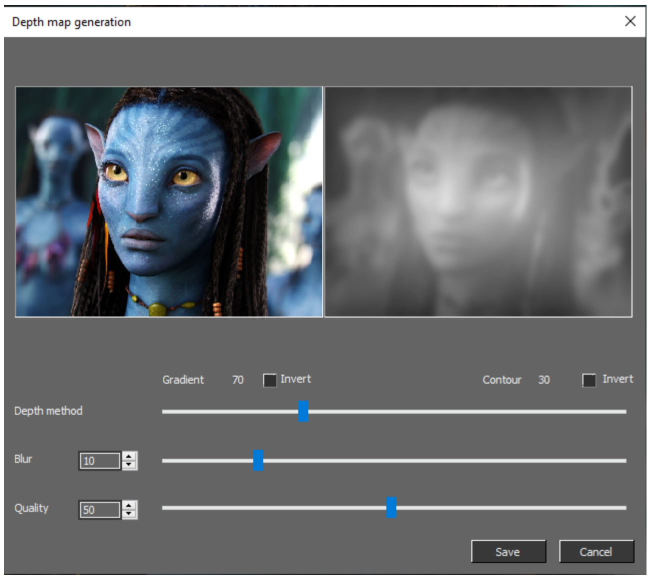


Title	Length
t_biggiethecat_iphone_8x6.mp4	00:09
i_timmy_8x6.mp4	00:13
portal_cloud_quilt_anim.mp4	00:05

Figure 4.8: Looking Glass Studio

Note that although Looking Glass Studio works with JPG, PNG, MP4, WEBM and HOP files, not all files will have the desired holographic effect when imported into the Looking Glass Portrait. This is due to the fact that the files need to have a depth map. Looking Glass Factory provides a way to generate and add a depth map to a file through a 2D to 3D converter of their own, but this is a paid service[11]. A free alternative uses the external software 3DMasterKit[12]. Figure 4.9 shows that this software can generate a customised depth map. It can also generate the customised multiviews needed for Looking Glass Portrait. Afterwards, it can be exported as a Looking Glass Portrait (8x6) JPG file.

[11] https://2dto3d.lookingglassfactory.com/
[12] https://triaxes.com/3dmasterkit/



Gradient 70 Invert
Contour 30 Invert
Depth method
Blur 10
Quality 50

Figure 4.9: 3DMasterKit - Generating depth map

Even though the playlist of videos and images is an exciting way of using this device, it does not fit the use case of this project since it only loops through the same files over and over again with no real user interaction besides pausing the playlist and going back/forward.

4.2.5 Unity

Looking Glass Factory provides plug-ins and add-ons for various 3D creator tools[13], one of them being Unity. Unity is a game engine mainly used to create 2D and 3D games that have a direct connection with Looking Glass Portrait. This means that, after installing the plug-in, it already gets all the configurations necessary to detect the Looking Glass Portrait and display high-quality content in it. This plug-in also provides Demos to explore a bit further into the interaction between the software and the display. The scenes in Unity are managed not only through their own configurations but also through scripts that can be written in C# (c-sharp).

4.2.6 Blender and Unreal Engine 5

Looking Glass Factory also provides plug-ins and add-ons for 3D creator tools such as Blender and Unreal Engine 5. Blender is a 3D creation tool, while Unreal Engine 5 is a game engine similar to Unity. Both have a similar connection between the software and the display, provided by a plug-in similar to Unity and can also be programmed using scripts written in Python and C++, respectively.

[13]https://docs.lookingglassfactory.com/developer-tools/devtools

4.2.7 Software tool comparison

To help make a decision, all the gathered information from the previous subsections was summarized in Table 4.3. This table highlights all the key factors the desired software framework must have to develop the prototypes with maximum efficiency.

Table 4.3: Software tool comparison

Software	LGP Connectivity	Direct 3D Support	Language	Platforms
Qt[14]	✗	✓	C++	Desktop, Mobile
GTK+[15]	✗	✗	Python, JavaScript, C++	Desktop
Electron[16]	✗	✗	JavaScript, HTML, CSS	Desktop
Looking Glass Studio[17]	✓	✓	-	Desktop
Unity[18]	✓	✓	C#	Desktop, Mobile
Blender[19]	✓	✓	Python	Desktop
Unreal Engine 5[20]	✓	✓	C++	Desktop, Mobile

Since having connectivity to Looking Glass Portrait is a crucial point for this decision, Qt, GTK+ and Electron were excluded. Looking Glass Studio is eliminated for the reason mentioned before. This leaves only three remaining software options. Unity was the software of choice since it covers all aspects needed from a software to develop this project and also because there was some previous knowledge of this tool from past experiences.

To summarize, all software tools and their respective versions used to support the prototypes are showcased in Table 4.4. Note that Blender was used not as a main development software tool but as an auxiliary one to handle some 3D models and import them into Unity.

Table 4.4: Versions of software tools used

Software	Version
Looking Glass Bridge[21]	2.0.6
Unity[22]	2021.3.16f1
Unity Plugin[23]	2.0 Alpha
Blender[24]	3.6

[14] https://www.qt.io/
[15] https://www.gtk.org/
[16] https://www.electronjs.org/pt/
[17] https://lookingglassfactory.com/software/looking-glass-studio
[18] https://unity.com/
[19] https://www.blender.org/
[20] https://www.unrealengine.com/en-US/unreal-engine-5
[21] https://lookingglassfactory.com/software/looking-glass-bridge
[22] https://unity.com/releases/editor/whats-new/2021.3.16
[23] https://docs.lookingglassfactory.com/developer-tools/unity/2.0-alpha
[24] https://builder.blender.org/download/daily/

4.3 Summary

In the previous Chapter 3, a study was made on the current Smart Home products, focused on the technologies used and their UIs. This proved helpful in this chapter when searching for the hardware to be used, as well as the software platform to design the UI of those prototypes. This chapter presented a comprehensive study of the available software platforms to develop UIs, and gathered all the relevant information about each one to support the final decision. It was decided to use the Unity development framework because it covers all the needs of this project and also due to the familiarity with this software. The hardware and software chosen in this chapter are used in the next Chapter 5, where the implementation and design of the prototypes are addressed.

Chapter 5

Implementing the display prototypes

The holographic display prototype was the first to be developed since it was the first to arrive after purchase. The goal was to develop a Unity project as common as possible so that it could be reused between prototypes. As previously mentioned, Looking Glass Factory provides a Unity plugin that comes with a variety of demo files that can run and show *Scenes*[1] directly on the Looking Glass Portrait. These demos served as a draft to test and implement various features to develop the holographic project.

5.1 Unity project general configurations

Each menu of the prototypes corresponds to a *Scene*. The Unity project for the holographic display was based on the previously mentioned demo project. This demo is depicted in Figure 5.1. It consists of three text files named *Moving Text*, *Instructions* and *Label* that are controlled by a script named *Demo Move Text* that is attached to the game object named *DEMO*. This script moves the *Moving Text* file in the Z axis by pressing the up and down keys on the keyboard. It also shows the distance travelled in relation to the plane of convergence. The demo comes with a *Camera*[2], named *Hologram Camera*, with some scripts (created by Looking Glass Factory) attached to it. These scripts not only visually identify the areas in the *Scene* covered by the *Hologram Camera*, but also give it some different properties to display Unity *Scenes* in a holographic way. One of these properties is that files and images seen by the *Hologram Camera* might become blurrier depending on its distance to the plane of convergence. This allows the prototype to create a 3D visual effect for text files, mainly because they can be placed on different values of the Z axis as long as they are not set too deep to avoid blurry text.

[1] https://docs.unity3d.com/560/Documentation/Manual/CreatingScenes.html
[2] https://docs.unity3d.com/ScriptReference/Camera.html

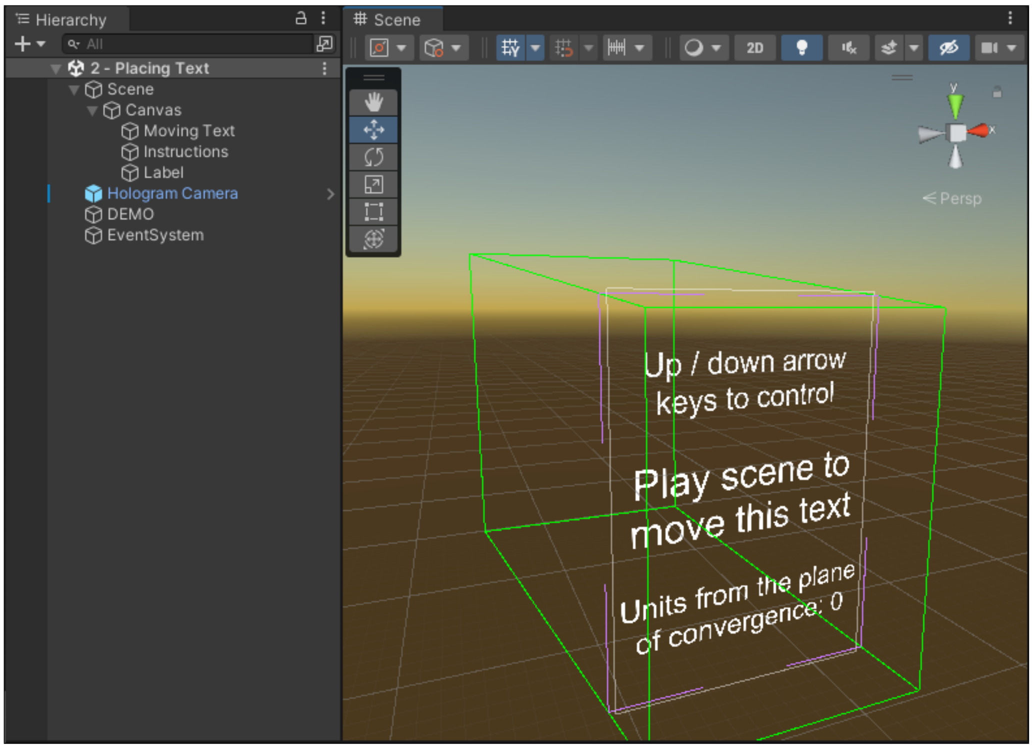
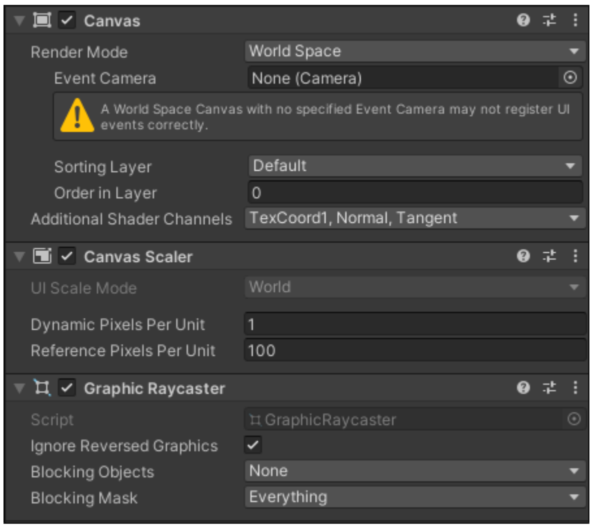


Figure 5.1: Unity Demo provided by Looking Glass Factory

The text files and images required for the prototypes can be added to the *Canvas*[3] that already came with the demo since it is synchronized with the *Hologram Camera* as seen in Figure 5.2. To start with the holographic project, the three text files and the game object containing the *Demo Move Text* script were deleted, resulting in an empty *Canvas*.



Canvas

Render Mode	World Space
Event Camera	None (Camera)

A World Space Canvas with no specified Event Camera may not register UI events correctly.

Sorting Layer	Default
Order in Layer	0
Additional Shader Channels	TexCoord1, Normal, Tangent

Canvas Scaler

UI Scale Mode	World
Dynamic Pixels Per Unit	1
Reference Pixels Per Unit	100

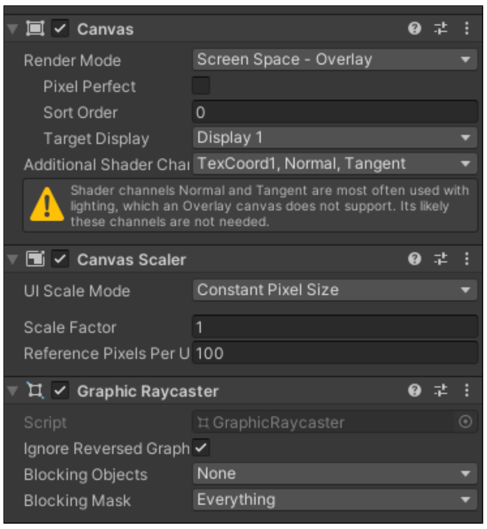
Graphic Raycaster

Script	GraphicRaycaster
Ignore Reversed Graphics	✓
Blocking Objects	None
Blocking Mask	Everything

Figure 5.2: *Canvas* configuration - holographic prototype

[3]https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

Most of the Unity configurations and settings are shared between the two projects (one for the holographic and the other for the transparent display). However, some differences should be noted. Namely, the *Hologram Camera* used on the holographic display project had to be replaced by a regular *Camera* on the transparent display project. Since this regular *Camera* does not detect depths (Z axis), the *Canvas* configuration had to be updated. More specifically, the Render Mode had to be changed from *World Space* to *Screen Space - Overlay*, as seen in Figure 5.3.



Canvas

Render Mode	Screen Space - Overlay
Pixel Perfect	
Sort Order	0
Target Display	Display 1
Additional Shader Chai	TexCoord1, Normal, Tangent

Shader channels Normal and Tangent are most often used with lighting, which an Overlay canvas does not support. Its likely these channels are not needed.

Canvas Scaler

UI Scale Mode	Constant Pixel Size
Scale Factor	1
Reference Pixels Per U	100

Graphic Raycaster

Script	GraphicRaycaster
Ignore Reversed Graph	✓
Blocking Objects	None
Blocking Mask	Everything

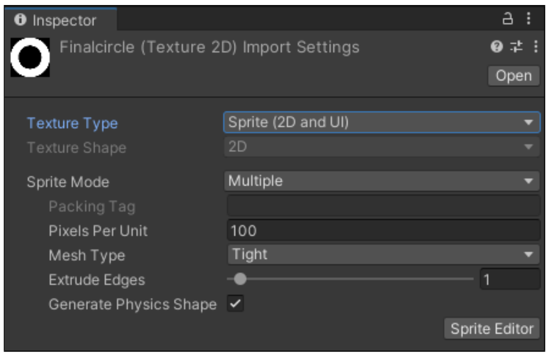
Figure 5.3: *Canvas* configuration - transparent prototype

Both projects have *Scenes* containing models that fit the theme of the menu. Most of these models were found on Sketchfab[4] and Unity Asset Store[5] and all of them were free to download. Sketchfab provides the source file of the model to import directly into Unity, while the Unity Asset Store adds the model to your assets upon login with the Unity account on their website.

5.1.1 Graphical representation

The conversion to 2D *Sprite* is a mandatory process that needs to be applied to any image, icon and emoji before they are inserted into any of the *Scenes*. A *Sprite*[6] is responsible for identifying the section of an image that should be used as a 2D object. The corresponding file needs to be configured into a 2D *Sprite* as seen in Figure 5.4 (a), and after that, the *Sprite Editor*[7] seen on the bottom right corner of that exact figure, can be opened to slice the figure and create the *Sprite* as seen in Figure 5.4 (b). Note that the 2D *Sprite* package does not come with the default Unity installation.

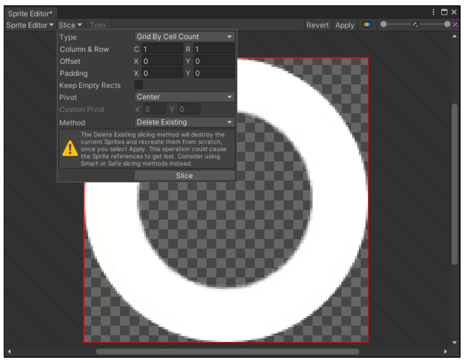
[4] https://sketchfab.com/feed
[5] https://assetstore.unity.com/3d
[6] https://docs.unity3d.com/ScriptReference/Sprite.html
[7] https://docs.unity3d.com/Manual/SpriteEditor.html



Finalcircle (Texture 2D) Import Settings

Texture Type	Sprite (2D and UI)
Texture Shape	2D
Sprite Mode	Multiple
Packing Tag	
Pixels Per Unit	100
Mesh Type	Tight
Extrude Edges	1
Generate Physics Shape	✓

(a) 2D *Sprite* configurations.



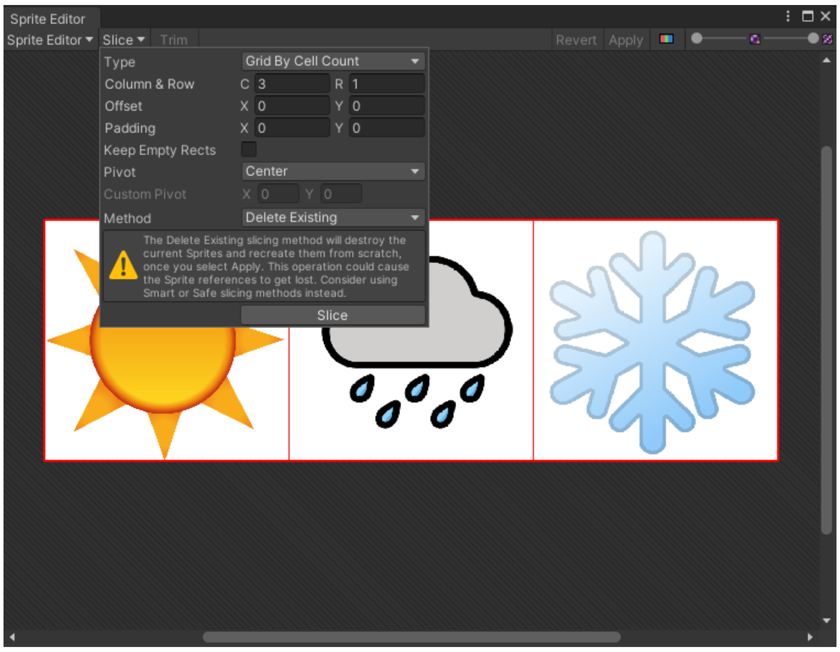
Type	Grid By Cell Count
Column & Row	C 1 R 1
Offset	X 0 Y 0
Padding	X 0 Y 0
Keep Empty Rects	
Pivot	Center
Custom Pivot	X 0 Y 0
Method	Delete Existing

The Delete Existing slicing method will destroy the current Sprites and recreate them from scratch, once you select Apply. This operation could cause the Sprite references to get lost. Consider using Smart or Safe slicing methods instead.

(b) *Sprite Editor*.

Figure 5.4: Steps to convert any image into a 2D *Sprite*

It is also possible to create multiple *Sprites* from the same image. On this project, this was especially useful to get the weather emoji (sun, rain and snow) to be displayed on the Main menu and the respective Weather menus. Since all the emojis were approximately equally spaced, it was possible to slice the *Sprite* into three columns and one row, as seen in Figure 5.5 instead of the one column and one row as in the previous example.

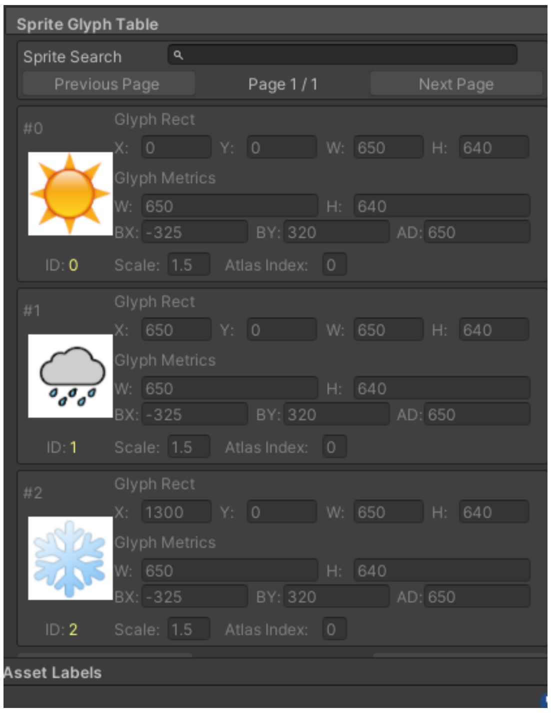


Type	Grid By Cell Count
Column & Row	C 3 R 1
Offset	X 0 Y 0
Padding	X 0 Y 0
Keep Empty Rects	
Pivot	Center
Custom Pivot	X 0 Y 0
Method	Delete Existing

The Delete Existing slicing method will destroy the current Sprites and recreate them from scratch, once you select Apply. This operation could cause the Sprite references to get lost. Consider using Smart or Safe slicing methods instead.

Figure 5.5: Multiple *Sprite* creation

Then, a *Sprite Asset* needs to be created, generating a new file containing all three emoji separated into *Sprites* with different IDs that can be edited individually, as seen in Figure 5.6.



Sprite Glyph Table

Sprite Search

Previous Page   Page 1 / 1   Next Page

#0
Glyph Rect
X: 0   Y: 0   W: 650   H: 640
Glyph Metrics
W: 650   H: 640
BX: -325   BY: 320   AD: 650
ID: 0   Scale: 1.5   Atlas Index: 0

#1
Glyph Rect
X: 650   Y: 0   W: 650   H: 640
Glyph Metrics
W: 650   H: 640
BX: -325   BY: 320   AD: 650
ID: 1   Scale: 1.5   Atlas Index: 0

#2
Glyph Rect
X: 1300   Y: 0   W: 650   H: 640
Glyph Metrics
W: 650   H: 640
BX: -325   BY: 320   AD: 650
ID: 2   Scale: 1.5   Atlas Index: 0

Asset Labels

Figure 5.6: *Sprite Asset* originated from the multiple *Sprite* creation

5.1.2 Scripts

Every Unity script used in these projects uses *MonoBehaviour*[8] as its main class. Therefore every script has an *Update*[9] method that is called every frame, as well as a *Start*[10] method that is the first method to be called on the exact frame when the script gets enabled. Other methods used in these projects were the *OnEnable*[11] and *OnDisable*[12] methods that are called when the object to which a script is attached first gets enabled or disabled, respectively. The *OnMouseDown*[13] method was also used since it is called when the user presses the mouse button (or, in this case: touches the touchscreen). All of these methods were used in combination with some other custom-made methods on the scripts of the project.

5.2 Solution architecture

An overview of the architecture designed for the holographic and the transparent prototypes can be seen in Figures 5.7 and 5.8, respectively. The diagrams show the developed scripts and the association between them. The objective of each script is described in the legend at the bottom of each figure. Both solutions are based on the menu diagram represented in Figure 5.9, which

[8] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
[9] https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html
[10] https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html
[11] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnEnable.html
[12] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDisable.html
[13] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseDown.html

shows the six menus that the prototypes will loop through. The Main menu will be the default one and contains the information that is usually most searched by the users: temperature, weather conditions, time and date. Following the Main menu, one of three Weather menus will be displayed according to the current weather: Sunny, Rain and Snow. The Setpoint menu can be accessed after the Weather menu. Its primary purpose is to let the user change the room temperature and check other information like outside temperature, humidity and wind speed. Lastly, the Energy menu provides the user with detailed information regarding the monthly household energy consumption.

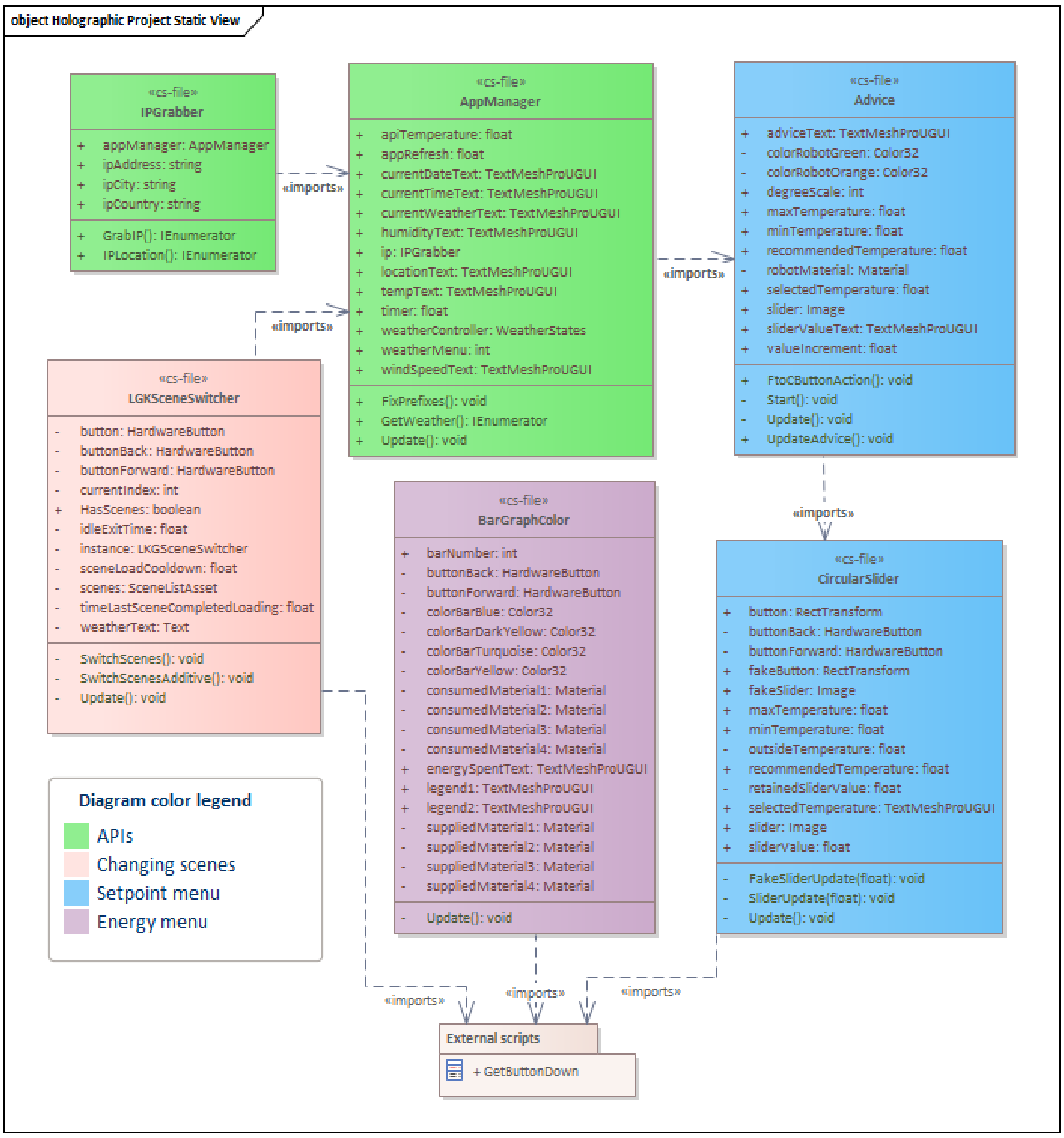

object Holographic Project Static View
«cs-file»
IPGrabber
+ appManager: AppManager
+ ipAddress: string
+ ipCity: string
+ ipCountry: string
+ GrabIP(): IEnumerator
+ IPLocation(): IEnumerator
«imports»
«cs-file»
AppManager
+ apiTemperature: float
+ appRefresh: float
+ currentDateText: TextMeshProUGUI
+ currentTimeText: TextMeshProUGUI
+ currentWeatherText: TextMeshProUGUI
+ humidityText: TextMeshProUGUI
+ ip: IPGrabber
+ locationText: TextMeshProUGUI
+ tempText: TextMeshProUGUI
+ timer: float
+ weatherController: WeatherStates
+ weatherMenu: int
+ windSpeedText: TextMeshProUGUI
+ FixPrefixes(): void
+ GetWeather(): IEnumerator
+ Update(): void
«imports»
«cs-file»
Advice
+ adviceText: TextMeshProUGUI
- colorRobotGreen: Color32
- colorRobotOrange: Color32
+ degreeScale: int
+ maxTemperature: float
+ minTemperature: float
+ recommendedTemperature: float
- robotMaterial: Material
- selectedTemperature: float
+ slider: Image
+ sliderValueText: TextMeshProUGUI
+ valueIncrement: float
+ FtoCButtonAction(): void
- Start(): void
- Update(): void
+ UpdateAdvice(): void
«imports»
«cs-file»
LGKSceneSwitcher
- button: HardwareButton
- buttonBack: HardwareButton
- buttonForward: HardwareButton
- currentIndex: int
+ HasScenes: boolean
- idleExitTime: float
- instance: LKGSceneSwitcher
- sceneLoadCooldown: float
- scenes: SceneListAsset
- timeLastSceneCompletedLoading: float
- weatherText: Text
- SwitchScenes(): void
- SwitchScenesAdditive(): void
- Update(): void
«imports»
«cs-file»
BarGraphColor
+ barNumber: int
- buttonBack: HardwareButton
- buttonForward: HardwareButton
- colorBarBlue: Color32
- colorBarDarkYellow: Color32
- colorBarTurquoise: Color32
- colorBarYellow: Color32
- consumedMaterial1: Material
- consumedMaterial2: Material
- consumedMaterial3: Material
- consumedMaterial4: Material
+ energySpentText: TextMeshProUGUI
+ legend1: TextMeshProUGUI
+ legend2: TextMeshProUGUI
- suppliedMaterial1: Material
- suppliedMaterial2: Material
- suppliedMaterial3: Material
- suppliedMaterial4: Material
- Update(): void
«cs-file»
CircularSlider
+ button: RectTransform
- buttonBack: HardwareButton
- buttonForward: HardwareButton
+ fakeButton: RectTransform
+ fakeSlider: Image
+ maxTemperature: float
+ minTemperature: float
- outsideTemperature: float
+ recommendedTemperature: float
- retainedSliderValue: float
+ selectedTemperature: TextMeshProUGUI
+ slider: Image
+ sliderValue: float
- FakeSliderUpdate(float): void
- SliderUpdate(float): void
- Update(): void
Diagram color legend
🟩 APIs
🟥 Changing scenes
🟦 Setpoint menu
🟪 Energy menu
«imports»
«imports»
«imports»
External scripts
+ GetButtonDown

Figure 5.7: Holographic prototype architecture

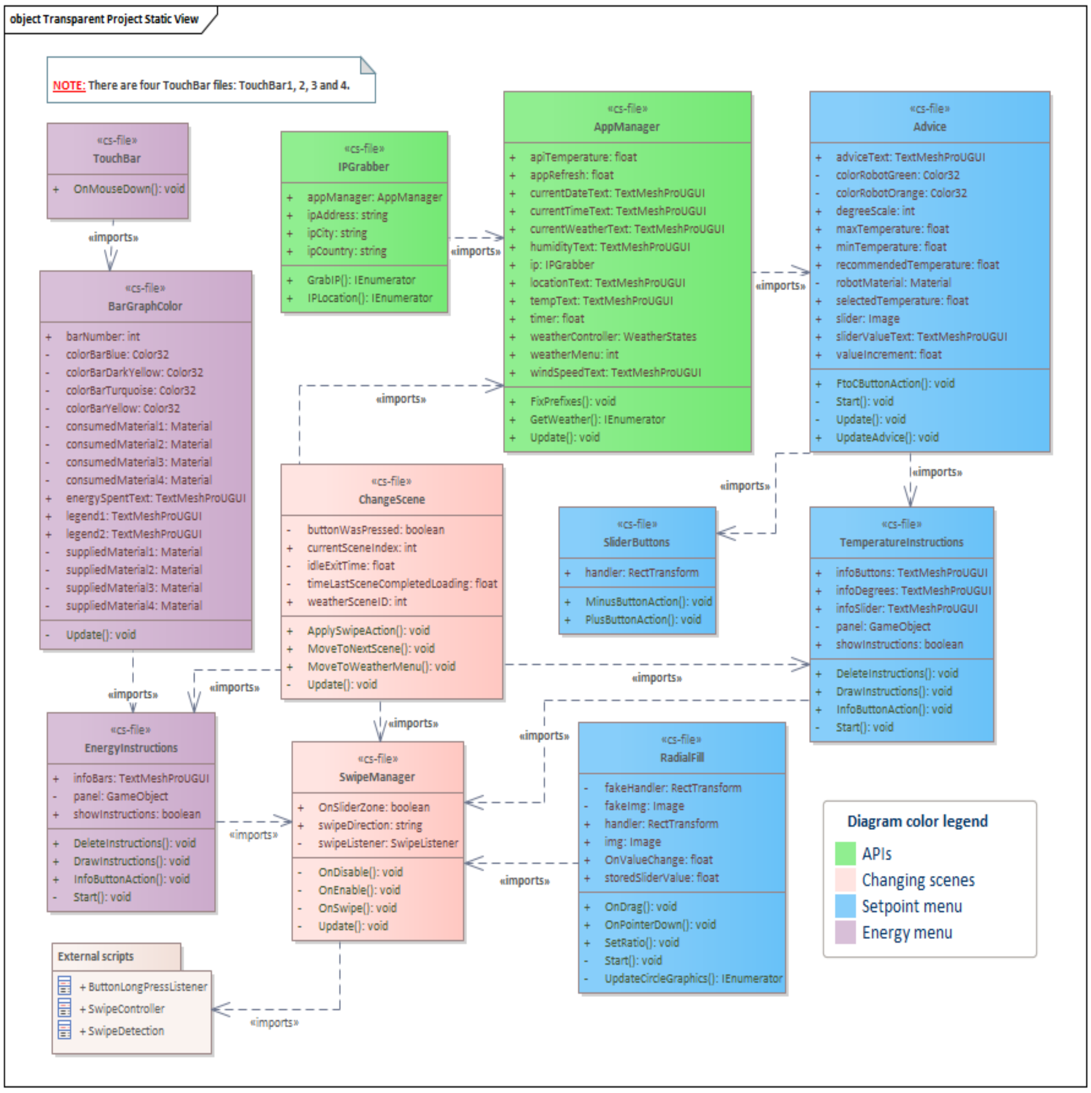

object Transparent Project Static View
NOTE: There are four TouchBar files: TouchBar1, 2, 3 and 4.
«cs-file» TouchBar
+ OnMouseDown(): void
«cs-file» IPGrabber
+ appManager: AppManager
+ ipAddress: string
+ ipCity: string
+ ipCountry: string
+ GrabIP(): IEnumerator
+ IPLocation(): IEnumerator
«cs-file» AppManager
+ apiTemperature: float
+ appRefresh: float
+ currentDateText: TextMeshProUGUI
+ currentTimeText: TextMeshProUGUI
+ currentWeatherText: TextMeshProUGUI
+ humidityText: TextMeshProUGUI
+ ip: IPGrabber
+ locationText: TextMeshProUGUI
+ tempText: TextMeshProUGUI
+ timer: float
+ weatherController: WeatherStates
+ weatherMenu: int
+ windSpeedText: TextMeshProUGUI
+ FixPrefixes(): void
+ GetWeather(): IEnumerator
+ Update(): void
«cs-file» Advice
+ adviceText: TextMeshProUGUI
- colorRobotGreen: Color32
- colorRobotOrange: Color32
+ degreeScale: int
+ maxTemperature: float
+ minTemperature: float
+ recommendedTemperature: float
- robotMaterial: Material
+ selectedTemperature: float
+ slider: Image
+ sliderValueText: TextMeshProUGUI
+ valueIncrement: float
+ FtoCButtonAction(): void
- Start(): void
- Update(): void
+ UpdateAdvice(): void
«cs-file» BarGraphColor
+ barNumber: int
- colorBarBlue: Color32
- colorBarDarkYellow: Color32
- colorBarTurquoise: Color32
- colorBarYellow: Color32
- consumedMaterial1: Material
- consumedMaterial2: Material
- consumedMaterial3: Material
- consumedMaterial4: Material
+ energySpentText: TextMeshProUGUI
+ legend1: TextMeshProUGUI
+ legend2: TextMeshProUGUI
- suppliedMaterial1: Material
- suppliedMaterial2: Material
- suppliedMaterial3: Material
- suppliedMaterial4: Material
- Update(): void
«cs-file» ChangeScene
- buttonWasPressed: boolean
+ currentSceneIndex: int
- idleExitTime: float
- timeLastSceneCompletedLoading: float
+ weatherSceneID: int
+ ApplySwipeAction(): void
+ MoveToNextScene(): void
+ MoveToWeatherMenu(): void
- Update(): void
«cs-file» SliderButtons
+ handler: RectTransform
+ MinusButtonAction(): void
+ PlusButtonAction(): void
«cs-file» TemperatureInstructions
+ infoButtons: TextMeshProUGUI
+ infoDegrees: TextMeshProUGUI
+ infoSlider: TextMeshProUGUI
- panel: GameObject
+ showInstructions: boolean
+ DeleteInstructions(): void
+ DrawInstructions(): void
+ InfoButtonAction(): void
- Start(): void
«cs-file» EnergyInstructions
+ infoBars: TextMeshProUGUI
- panel: GameObject
+ showInstructions: boolean
+ DeleteInstructions(): void
+ DrawInstructions(): void
+ InfoButtonAction(): void
- Start(): void
«cs-file» SwipeManager
+ OnSliderZone: boolean
+ swipeDirection: string
- swipeListener: SwipeListener
- OnDisable(): void
- OnEnable(): void
- OnSwipe(): void
- Update(): void
«cs-file» RadialFill
- fakeHandler: RectTransform
- fakeImg: Image
+ handler: RectTransform
+ img: Image
+ OnValueChange: float
+ storedSliderValue: float
+ OnDrag(): void
+ OnPointerDown(): void
+ SetRatio(): void
- Start(): void
- UpdateCircleGraphics(): IEnumerator
External scripts
+ ButtonLongPressListener
+ SwipeController
+ SwipeDetection
«imports»
Diagram color legend
APIs
Changing scenes
Setpoint menu
Energy menu

Figure 5.8: Transparent prototype architecture

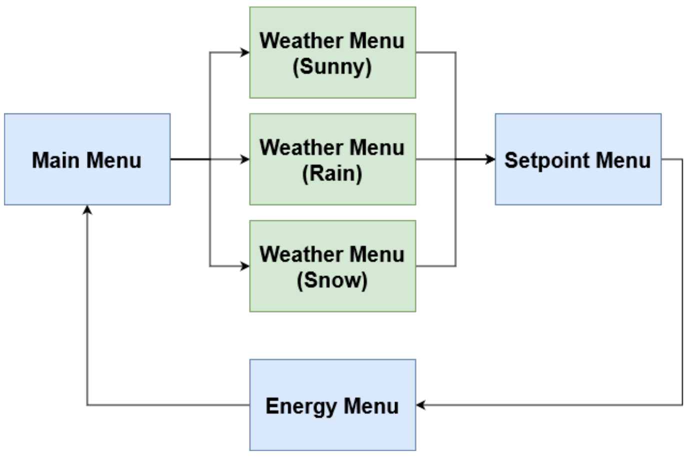


Figure 5.9: Menu diagram used on both prototypes

5.2.1 Used APIs

The prototype menus, in particular concerning the weather-related data displayed, are populated with information provided by an open-source weather channel available online through an Application Programming Interface (API). *OpenWeatherMap*[14] was the weather API used to get the weather-related data, and it was used in combination with the public IP address and geolocation API *IPIFY*[15] [16]. A key needs to be generated in order to use all of the mentioned APIs, and this generation is free on each APIs website. These keys are then inserted into a request URL that the APIs will receive and then respond to in JSON format (except for the *IPIFY* IP address response which comes in a string format). The interaction between both APIs is the same on both prototypes, and is explained by the following sequence diagrams (Figures 5.10 to 5.13) accompanied by a proper description. This interaction was based on an open-source Unity project that can be found on the following GitHub[17].

The interaction starts on the power-up, where the *GrabIP* method seen in Figure 5.10 uses *IPIFY* API to return the user's IP address. This address is then used by the *IPLocation* method seen in Figure 5.11, which provides this IP address to the *Geo IPIFY* to get the location (country and city) of the device. The end of *IPLocation* method marks the end of the initialization phase, and the *GetWeather* method seen in Figure 5.12 is called. This method provides the location to the *OpenWeatherMap* API in order to return accurate data for the correct location of the device. The returned data consists of current weather condition, wind speed, humidity and outside temperature, among many others that were not used on these prototypes.

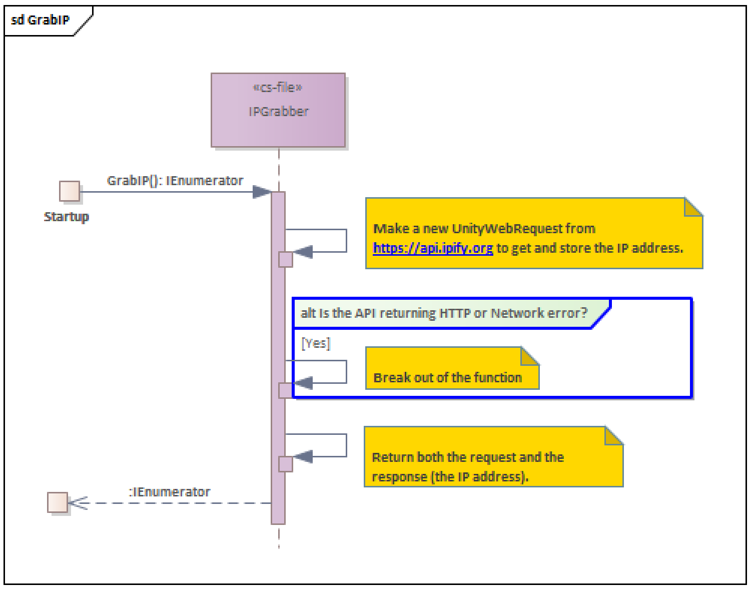

sd GrabIP
«cs-file»
IPGrabber
GrabIP(): IEnumerator
Startup
Make a new UnityWebRequest from https://api.ipify.org to get and store the IP address.
alt Is the API returning HTTP or Network error?
[Yes]
Break out of the function
Return both the request and the response (the IP address).
:IEnumerator

Figure 5.10: *GrabIP* - sequence diagram

[14] https://openweathermap.org/
[15] https://www.ipify.org/
[16] https://geo.ipify.org/
[17] https://github.com/JamesSheppardd/CurrentWeather

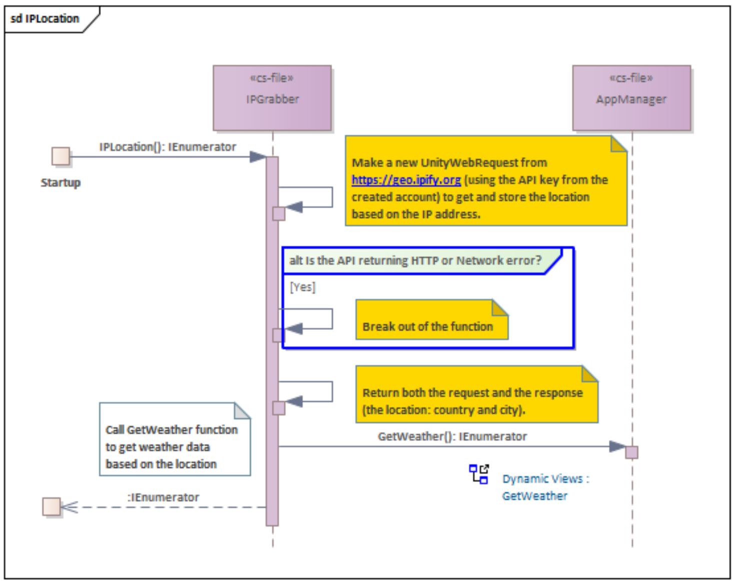

sd IPLocation
«cs-file» IPGrabber
«cs-file» AppManager
IPLocation(): IEnumerator
Startup
Make a new UnityWebRequest from https://geo.ipify.org (using the API key from the created account) to get and store the location based on the IP address.
alt Is the API returning HTTP or Network error?
[Yes]
Break out of the function
Return both the request and the response (the location: country and city).
GetWeather(): IEnumerator
Call GetWeather function to get weather data based on the location
Dynamic Views : GetWeather
:IEnumerator

Figure 5.11: *IPLocation* - sequence diagram

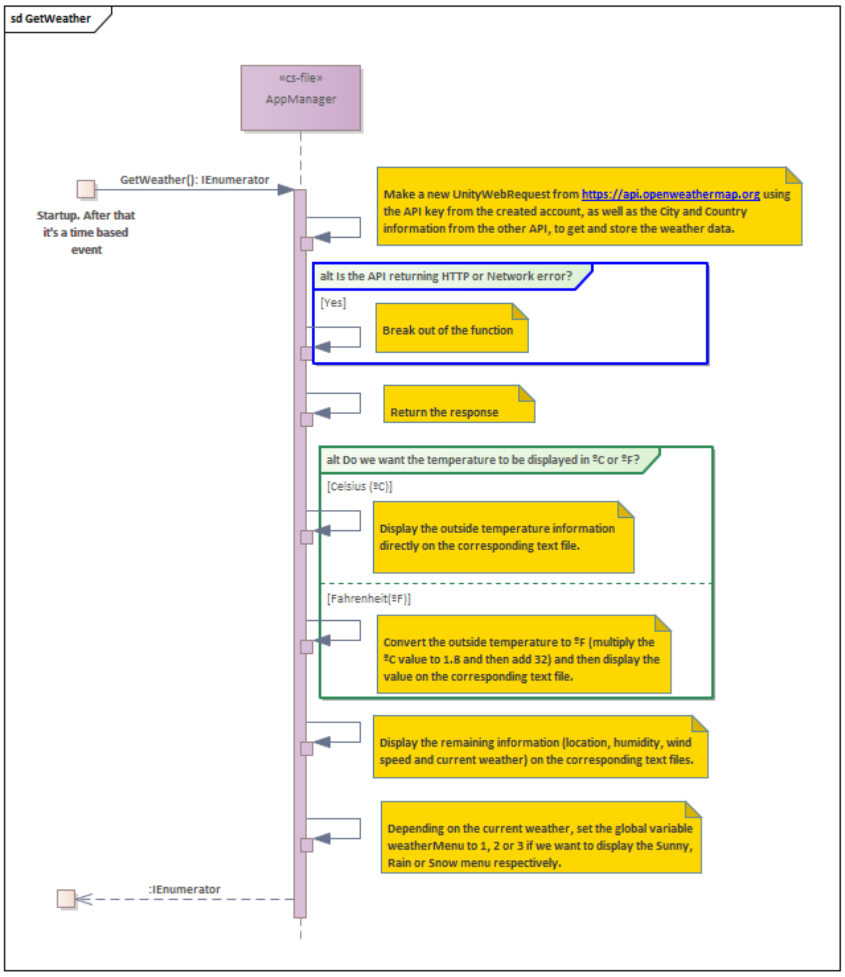

sd GetWeather
«cs-file» AppManager
GetWeather(): IEnumerator
Startup. After that it's a time based event
Make a new UnityWebRequest from https://api.openweathermap.org using the API key from the created account, as well as the City and Country information from the other API, to get and store the weather data.
alt Is the API returning HTTP or Network error?
[Yes]
Break out of the function
Return the response
alt Do we want the temperature to be displayed in ºC or ºF?
[Celsius (ºC)]
Display the outside temperature information directly on the corresponding text file.
[Fahrenheit(ºF)]
Convert the outside temperature to ºF (multiply the ºC value to 1.8 and then add 32) and then display the value on the corresponding text file.
Display the remaining information (location, humidity, wind speed and current weather) on the corresponding text files.
Depending on the current weather, set the global variable weatherMenu to 1, 2 or 3 if we want to display the Sunny, Rain or Snow menu respectively.
:IEnumerator

Figure 5.12: *GetWeather* - sequence diagram

The *AppManager* script, responsible for dealing with the different APIs interactions, has an *Update* method, seen in Figure 5.13, that deals with the time and date format and, most importantly, it grants that the data fetched from the weather API is refreshed periodically by calling the method *GetWeather*.

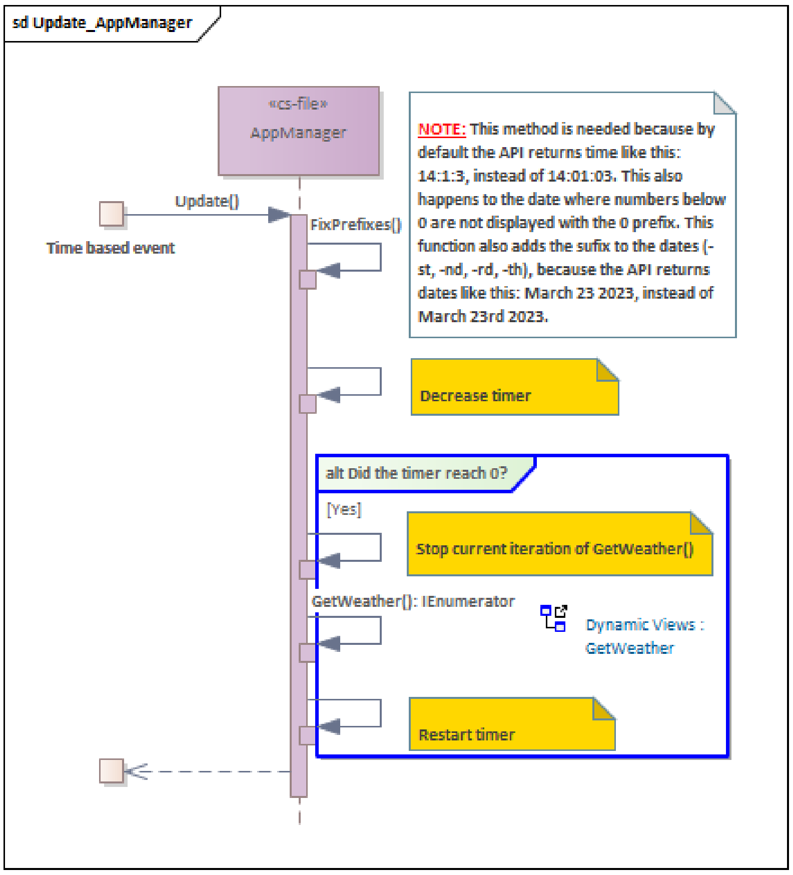

sd Update_AppManager
«cs-file»
AppManager
Update()
Time based event
FixPrefixes()
NOTE: This method is needed because by default the API returns time like this: 14:1:3, instead of 14:01:03. This also happens to the date where numbers below 0 are not displayed with the 0 prefix. This function also adds the sufix to the dates (-st, -nd, -rd, -th), because the API returns dates like this: March 23 2023, instead of March 23rd 2023.
Decrease timer
alt Did the timer reach 0?
[Yes]
Stop current iteration of GetWeather()
GetWeather(): IEnumerator
Dynamic Views : GetWeather
Restart timer

Figure 5.13: *Update* (AppManager) - sequence Diagram

5.2.2 Weather conditions

Even though *OpenWeatherMap* API returns nine types of weather conditions with dozens of variations, the conditions were grouped into the nine main ones to simplify the prototypes. In order to avoid the creation of nine different weather menus, these nine main conditions were grouped into one of three possible classes, as shown in Table 5.1.

Table 5.1: Weather conditions returned by *OpenWeatherMap* API and respective grouping

API weather condition	**Main group**	**Shown on menu**
thunderstorm with light rain	Thunderstorm	Rain
thunderstorm with rain		
thunderstorm with heavy rain		
light thunderstorm		
thunderstorm		
heavy thunderstorm		
ragged thunderstorm		
thunderstorm with light drizzle		
thunderstorm with drizzle		
thunderstorm with heavy drizzle		
light intensity drizzle	Shower rain	
drizzle		
heavy intensity drizzle		
light intensity drizzle rain		
drizzle rain		
heavy intensity drizzle rain		
shower rain and drizzle		
heavy shower rain and drizzle		
shower drizzle		
light intensity shower rain		
shower rain		
heavy intensity shower rain		
ragged shower rain		
light rain	Rain	
moderate rain		
heavy intensity rain		
very heavy rain		
extreme rain		
freezing rain		
broken clouds	Broken clouds	
overcast clouds		
light snow	Snow	Snow
snow		
heavy snow		
sleet		
light shower sleet		
shower sleet		
light rain and snow		
rain and snow		
light shower snow		
shower snow		
heavy shower snow		
mist	Mist	
smoke		
haze		
sand/dust whirls		
fog		
sand		
dust		
volcanic ash		
squalls		
tornado		
clear sky	Clear sky	Sunny
few clouds	Few clouds	
scattered clouds	Scattered Clouds	

5.3 Menus

Each menu is coded in a different Unity *Scene*, and each one has its own objects and scripts associated with it. Since both prototypes share most of the Unity project configurations and settings, they follow very similar approaches.

5.3.1 Switching menus

Interacting with the holographic display was the first challenge of this project since Looking Glass Portrait does not have a touchscreen. Initially, the idea was to use external buttons or voice commands to change between menus. However, since external buttons would probably harm both the UX and aesthetics of the display, and voice commands would be too time-consuming, another approach had to be found. As mentioned before, Looking Glass Portrait has three capacitive buttons that could interact with the display, as long as they were programmed for it on Unity's scripts. The originally named *Play/Pause* button was used as the trigger to move to the next menu. In contrast, the other two buttons, *Back* and *Forward*, interacted with specific features of the Setpoint and Energy menus that will be addressed further in this document.

Interacting with the transparent display was easier than with the holographic since it has a touchscreen. Therefore, the display is able to detect input touches on the touchscreen, so changing between *Scenes* using a digital button is a valid approach for this prototype. To make use of the large touchscreen display, a swiping alternative was also implemented to change between menus.

5.3.1.1 Switching menus in the holographic display

The *LGKSceneSwitcher* script has an *Update* method responsible for the menu switching. This method can be seen in Figure 5.14, and it starts by calculating how long the current *Scene* has been active. This value resets on every user interaction, and if the value exceeds a predefined idle timer, the prototype will automatically go to the Main menu (unless it is already in it).

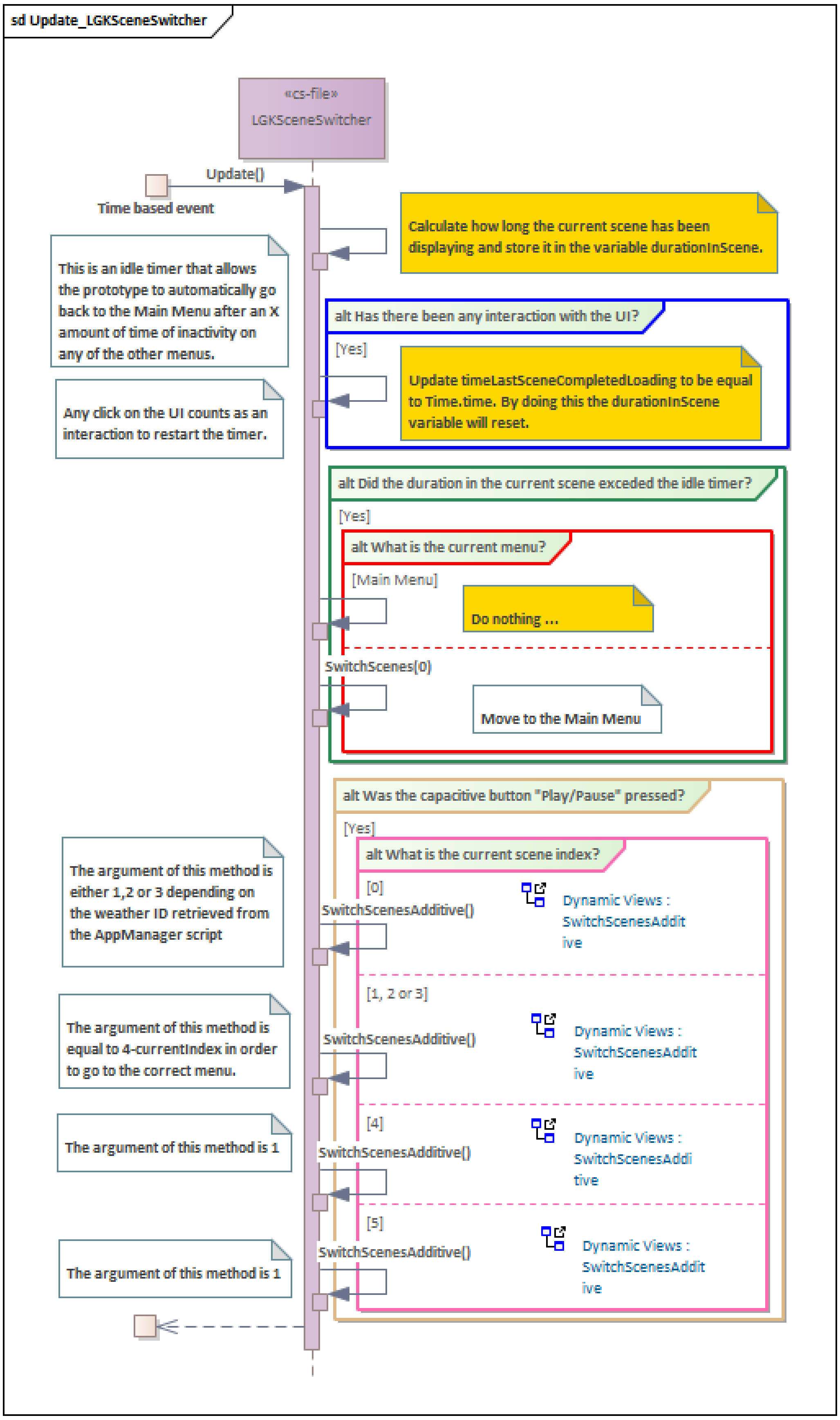
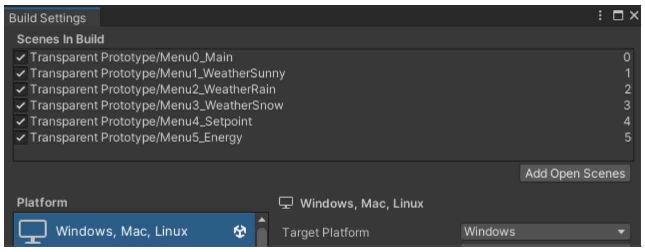

sd Update_LGKSceneSwitcher
«cs-file»
LGKSceneSwitcher
Update()
Time based event
This is an idle timer that allows the prototype to automatically go back to the Main Menu after an X amount of time of inactivity on any of the other menus.
Any click on the UI counts as an interaction to restart the timer.
Calculate how long the current scene has been displaying and store it in the variable durationInScene.
alt Has there been any interaction with the UI?
[Yes]
Update timeLastSceneCompletedLoading to be equal to Time.time. By doing this the durationInScene variable will reset.
alt Did the duration in the current scene exceded the idle timer?
[Yes]
alt What is the current menu?
[Main Menu]
Do nothing ...
SwitchScenes(0)
Move to the Main Menu
alt Was the capacitive button "Play/Pause" pressed?
[Yes]
alt What is the current scene index?
[0]
SwitchScenesAdditive()
Dynamic Views : SwitchScenesAdditive
The argument of this method is either 1,2 or 3 depending on the weather ID retrieved from the AppManager script
[1, 2 or 3]
SwitchScenesAdditive()
Dynamic Views : SwitchScenesAdditive
The argument of this method is equal to 4-currentIndex in order to go to the correct menu.
[4]
SwitchScenesAdditive()
Dynamic Views : SwitchScenesAdditive
The argument of this method is 1
[5]
SwitchScenesAdditive()
Dynamic Views : SwitchScenesAdditive
The argument of this method is 1

Figure 5.14: *Update* (LGKSceneSwitcher) - sequence diagram

Before exploring this method in more depth, the concept of *Scene* index needs to be addressed. The *Scene* index needs to be integrated into the build settings of a Unity project. Each *Scene* gets a unique index value associated (an integer number starting at zero), as seen in Figure 5.15.



Build Settings

Scenes In Build

✓ Transparent Prototype/Menu0_Main	0
✓ Transparent Prototype/Menu1_WeatherSunny	1
✓ Transparent Prototype/Menu2_WeatherRain	2
✓ Transparent Prototype/Menu3_WeatherSnow	3
✓ Transparent Prototype/Menu4_Setpoint	4
✓ Transparent Prototype/Menu5_Energy	5

Add Open Scenes

Platform

Windows, Mac, Linux

Windows, Mac, Linux

Target Platform: Windows

Figure 5.15: Unity build settings

GetButtonDown is a method developed by Looking Glass Factory that comes with the demo, and it is responsible for returning if a certain capacitive button was pressed or not. This method is called next on *LGKSceneSwitcher*'s *Update* method, and if a button is pressed, it switches to the next *Scene* by calling the method *SwitchSceneAdditive*, seen in Figure 5.16.

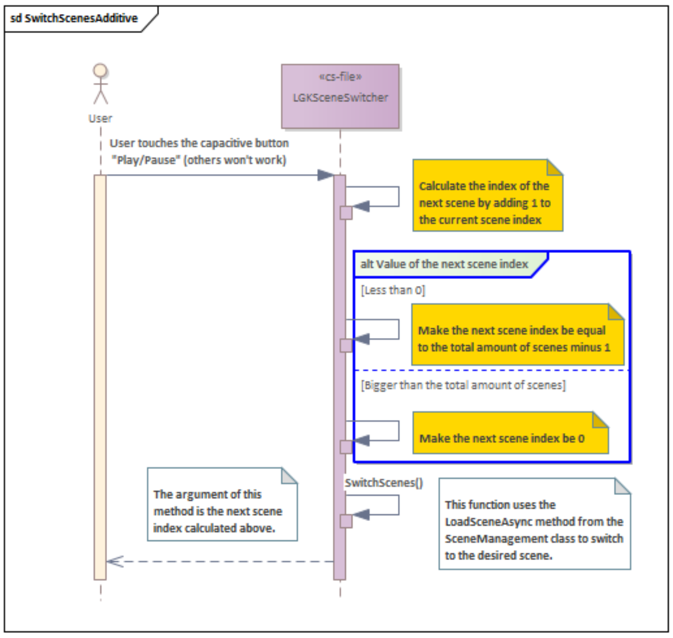
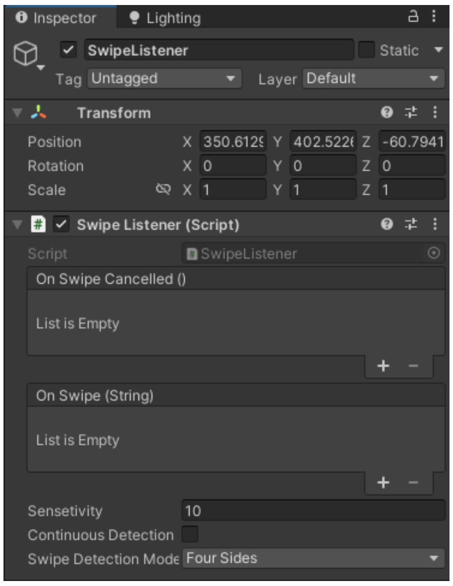

sd SwitchScenesAdditive
User
«cs-file»
LGKSceneSwitcher
User touches the capacitive button "Play/Pause" (others won't work)
Calculate the index of the next scene by adding 1 to the current scene index
alt Value of the next scene index
[Less than 0]
Make the next scene index be equal to the total amount of scenes minus 1
[Bigger than the total amount of scenes]
Make the next scene index be 0
SwitchScenes()
The argument of this method is the next scene index calculated above.
This function uses the LoadSceneAsync method from the SceneManagement class to switch to the desired scene.

Figure 5.16: *SwitchScenesAdditive* - sequence diagram

Note that it is only possible to access the capacitive buttons when using the 2.0 alpha version of the Unity plug-in provided by Looking Glass Factory, since the classes and scripts used will not be available on previous versions of this plug-in.

5.3.1.2 Switching menus in the transparent display

As referred, swipes were implemented in the transparent display to switch between menus. To properly detect a swipe, an external package[18] was downloaded into Unity. This package contains various scripts that work together to detect a swipe. The *SwipeListener* is the one that will be directly used on the *Scenes*. Every *Scene* needs to have an empty object with this script attached to it, and configured according to Figure 5.17, making sure that the detection mode is set to four sides so that it detects up, down, right and left swipes.



```
Inspector   Lighting
SwipeListener                      Static
Tag Untagged            Layer Default
Transform
Position      X 350.612  Y 402.522  Z -60.7941
Rotation      X 0        Y 0        Z 0
Scale         X 1        Y 1        Z 1
Swipe Listener (Script)
Script                  SwipeListener
On Swipe Cancelled ()
List is Empty
On Swipe (String)
List is Empty
Sensetivity             10
Continuous Detection
Swipe Detection Mode    Four Sides
```

Figure 5.17: *SwipeListener* - object

Another script named *SwipeManager*, which was based on an existing project[19], uses the previously mentioned empty object in its methods. The first method responsible for activating the *Swipe Listener* is named *OnEnable*, and can be seen in Figure 5.18. This method is called every time the previously mentioned empty object gets enabled/activated. This happens when the *Scene* is first loaded, and it only gets disabled/deactivated upon changing to another *Scene*, which will trigger the calling of the *OnDisable* method seen in Figure 5.19. This method deactivates the *Swipe Listener*. While the *Swipe Listener* is active, the method named *OnSwipe*, seen in Figure 5.20, is called every time the user makes a swipe movement, and it stores the direction of the swipe (left, right, up or down). The swipe movement can be made in any region of the display, except on the slider zone when on the Setpoint menu. In that specific case, the swipe will be ignored to avoid accidental menu or setpoint changes. The *Update* method seen in Figure 5.21 ensures the swipe movement is allowed by checking that the swipe coordinates are outside the limited zones.

[18]https://github.com/khadzhynov/SwipeController
[19]https://github.com/herbou/Unity_SwipeDetection

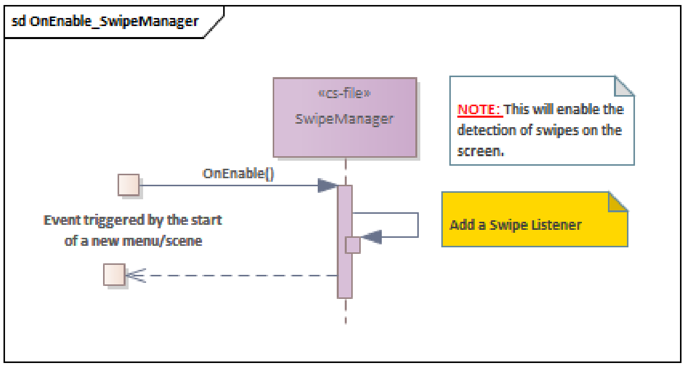

sd OnEnable_SwipeManager
«cs-file»
SwipeManager
NOTE: This will enable the detection of swipes on the screen.
OnEnable()
Event triggered by the start of a new menu/scene
Add a Swipe Listener

Figure 5.18: *OnEnable* (SwipeManager) - sequence diagram

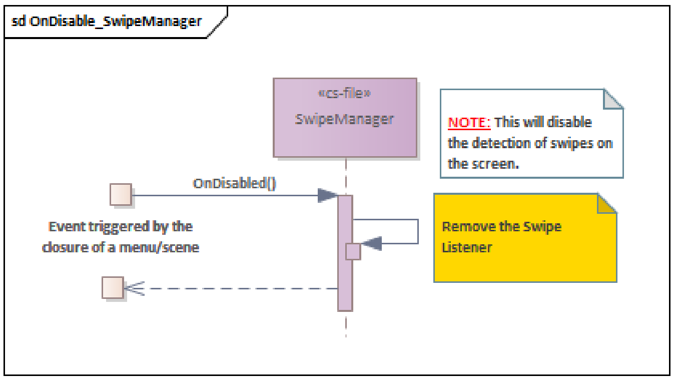

sd OnDisable_SwipeManager
«cs-file»
SwipeManager
NOTE: This will disable the detection of swipes on the screen.
OnDisabled()
Event triggered by the closure of a menu/scene
Remove the Swipe Listener

Figure 5.19: *OnDisable* (SwipeManager) - sequence diagram

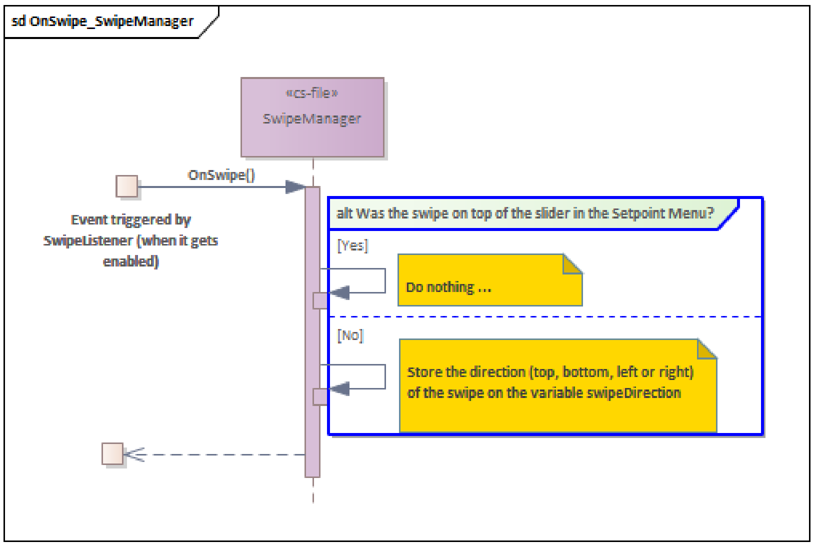

sd OnSwipe_SwipeManager
«cs-file»
SwipeManager
OnSwipe()
Event triggered by SwipeListener (when it gets enabled)
alt Was the swipe on top of the slider in the Setpoint Menu?
[Yes]
Do nothing ...
[No]
Store the direction (top, bottom, left or right) of the swipe on the variable swipeDirection

Figure 5.20: *OnSwipe* (SwipeManager) - sequence diagram

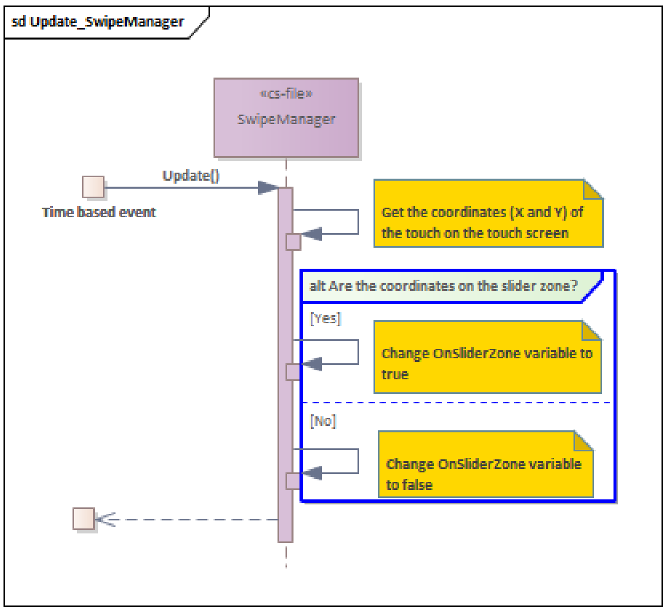

sd Update_SwipeManager
«cs-file»
SwipeManager
Update()
Time based event
Get the coordinates (X and Y) of the touch on the touch screen
alt Are the coordinates on the slider zone?
[Yes]
Change OnSliderZone variable to true
[No]
Change OnSliderZone variable to false

Figure 5.21: *Update* (SwipeManager) - sequence diagram

This transparent prototype also includes buttons to change between *Scenes* for people who do not want to use the swipe functionality. These buttons, named *NextSceneButton*, were created using the *Button*[20] class. They were added to every *Scene* and positioned in the top right corner.

To actually switch menus using the swipes detected by *SwipeManager*, a new script called *ChangeScene* was created. This script is similar to the *LGKSceneSwitcher* used on the holographic prototype, although it has some significant changes. The *Update* method of this script is shown in Figure 5.22 and it starts by retrieving the information of which Weather menu is supposed to be shown according to the *OpenWeatherMap* API. After that, it calculates how long the current *Scene* has been active in the same way as the holographic prototype.

Some menus have user instructions. If these instructions are being shown, the swipe will be ignored and the prototype will stay on the same menu. If the instructions are not being shown and the user makes a swipe movement, then the *ApplySwipeAction* seen in Figure 5.23 method will be called. The first thing this method does is to check the current *Scene* index. After that, it checks in which direction the user swiped to know if it should go to the next menu or the previous one. If the user swipes top or bottom, then the swipe is ignored. The method *MoveToNextMenu* is used to move to any menu except the Weather menu because, in that case, the method *MoveToWeatherMenu* is used instead. Both methods, represented by Figures 5.24 and 5.25, are very similar, the only difference being that *MoveToWeatherMenu* checks which should be the correct Weather menu according to the API before loading a new *Scene*.

[20]https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Button.html

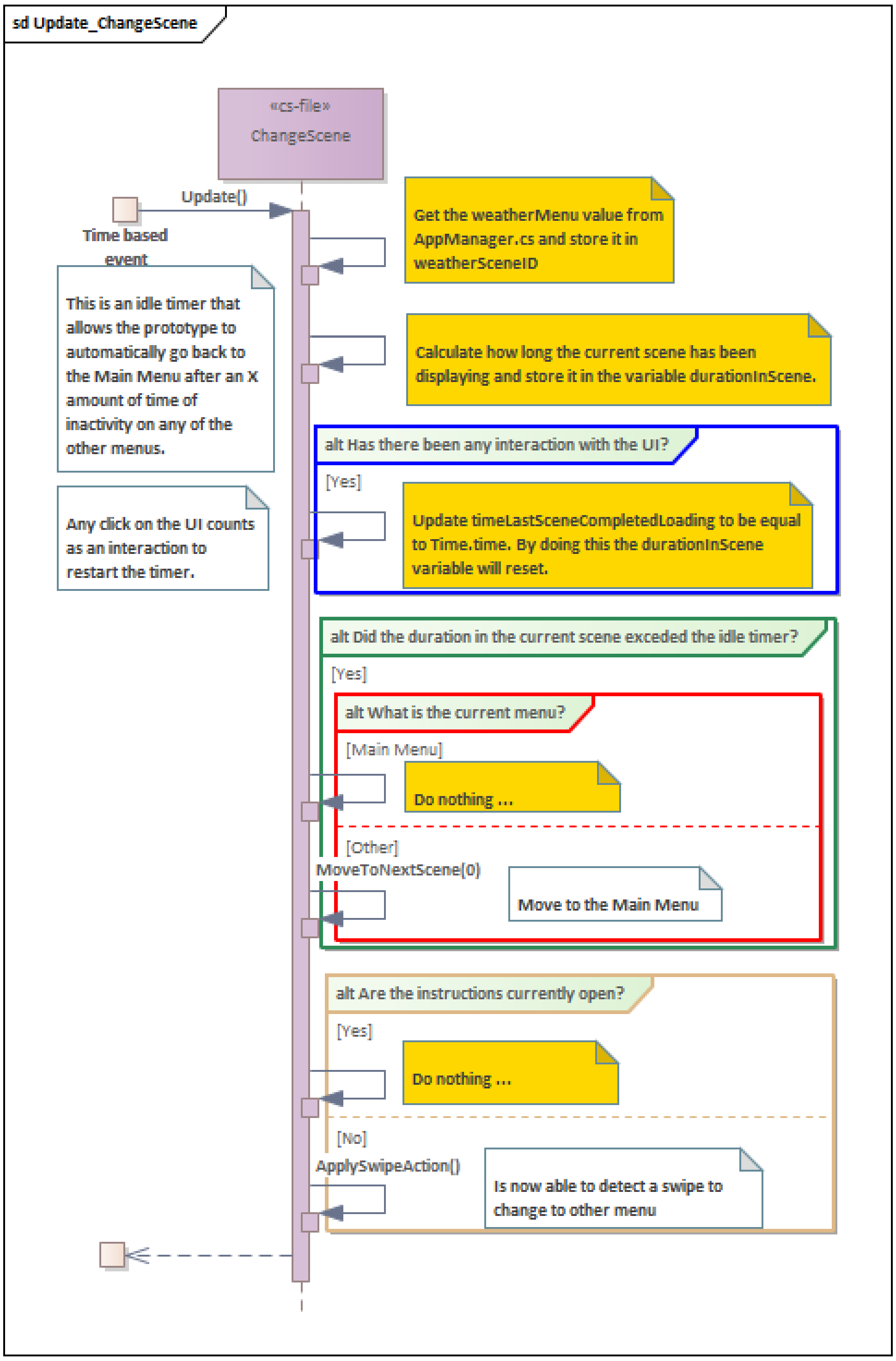

sd Update_ChangeScene
«cs-file»
ChangeScene
Update()
Time based event
This is an idle timer that allows the prototype to automatically go back to the Main Menu after an X amount of time of inactivity on any of the other menus.
Any click on the UI counts as an interaction to restart the timer.
Get the weatherMenu value from AppManager.cs and store it in weatherSceneID
Calculate how long the current scene has been displaying and store it in the variable durationInScene.
alt Has there been any interaction with the UI?
[Yes]
Update timeLastSceneCompletedLoading to be equal to Time.time. By doing this the durationInScene variable will reset.
alt Did the duration in the current scene exceded the idle timer?
[Yes]
alt What is the current menu?
[Main Menu]
Do nothing ...
[Other]
MoveToNextScene(0)
Move to the Main Menu
alt Are the instructions currently open?
[Yes]
Do nothing ...
[No]
ApplySwipeAction()
Is now able to detect a swipe to change to other menu

Figure 5.22: *Update* (ChangeScene) - sequence diagram

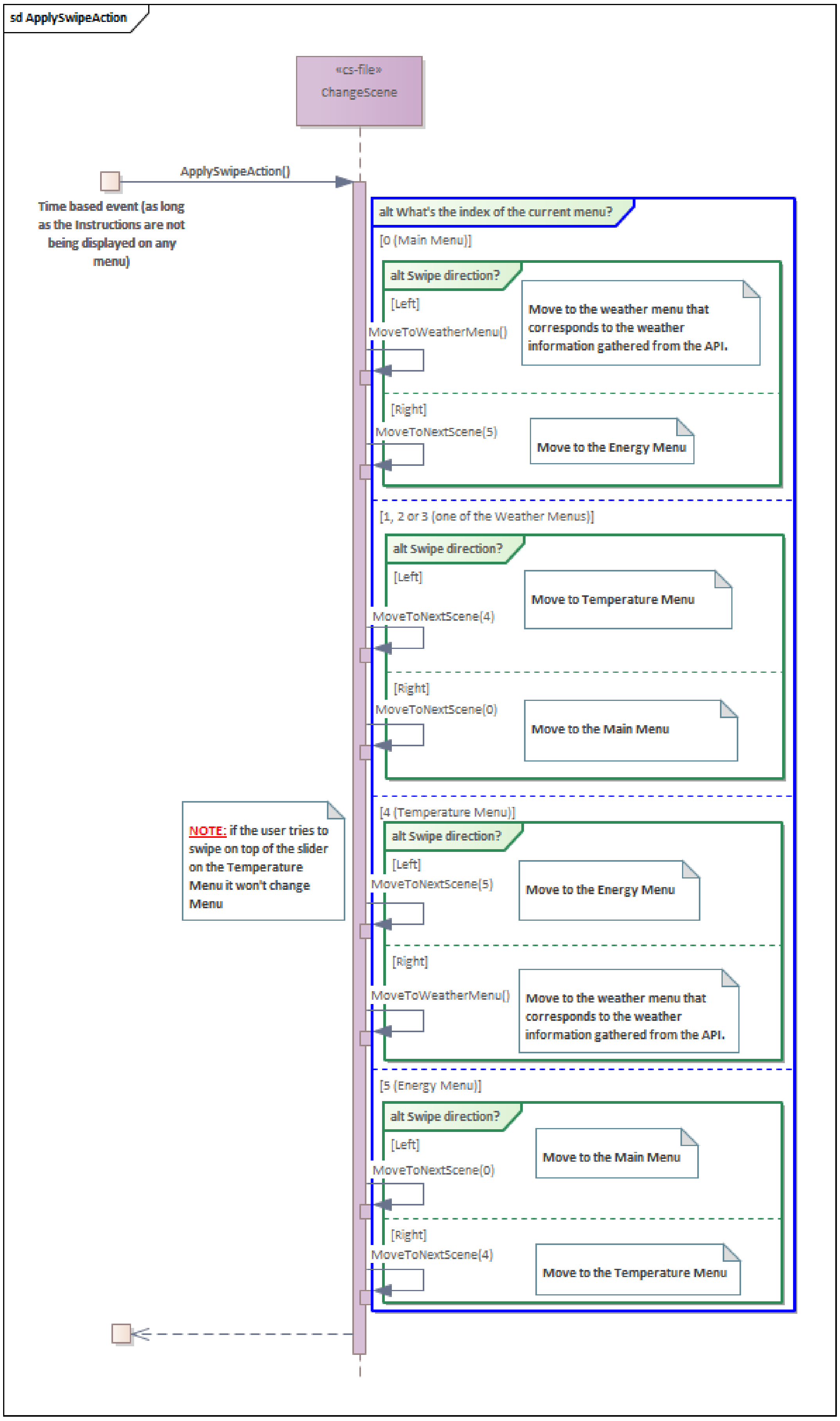


Figure 5.23: *ApplySwipeAction* - sequence diagram

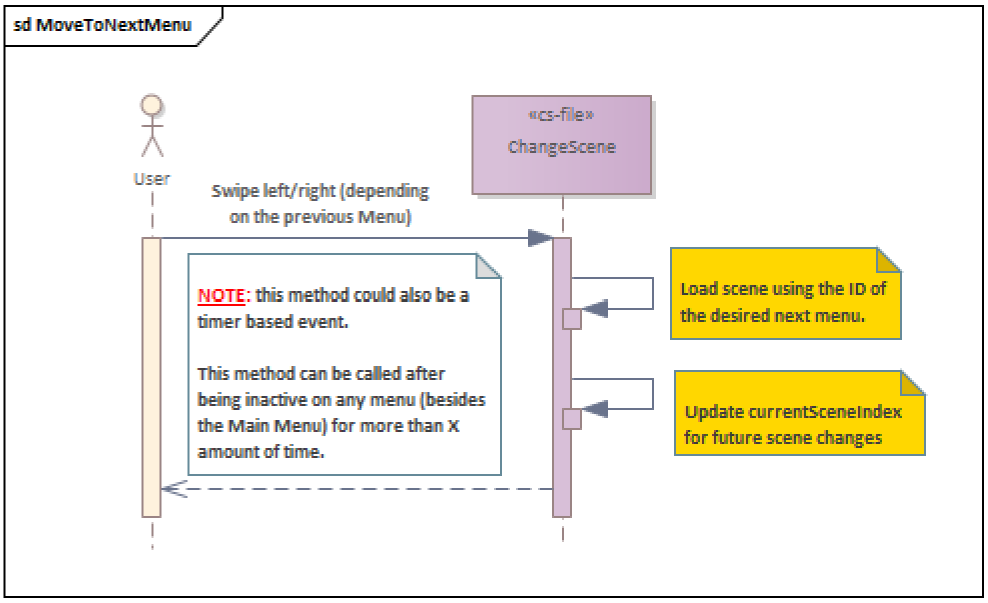

sd MoveToNextMenu
User
«cs-file»
ChangeScene
Swipe left/right (depending on the previous Menu)
NOTE: this method could also be a timer based event.
This method can be called after being inactive on any menu (besides the Main Menu) for more than X amount of time.
Load scene using the ID of the desired next menu.
Update currentSceneIndex for future scene changes

Figure 5.24: *MoveToNextMenu* - sequence diagram

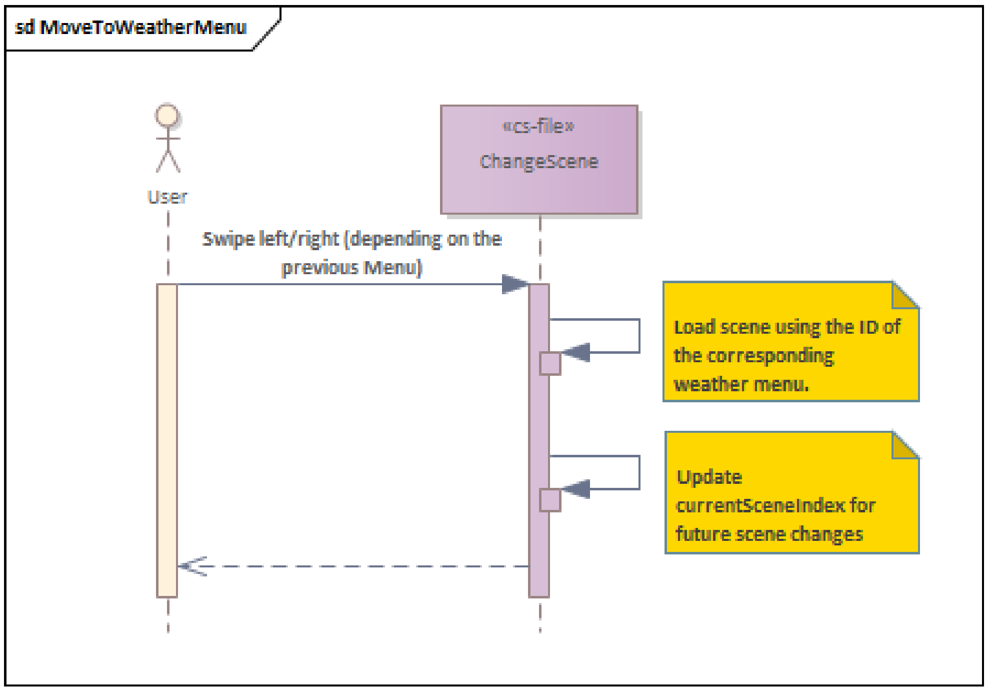
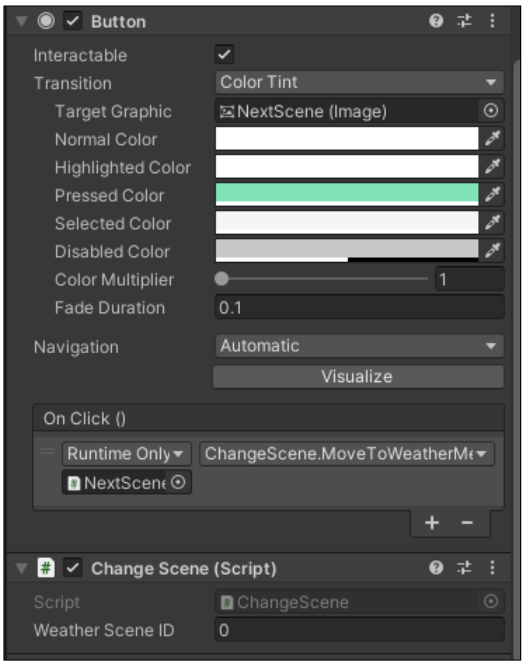

sd MoveToWeatherMenu
User
«cs-file»
ChangeScene
Swipe left/right (depending on the previous Menu)
Load scene using the ID of the corresponding weather menu.
Update currentSceneIndex for future scene changes

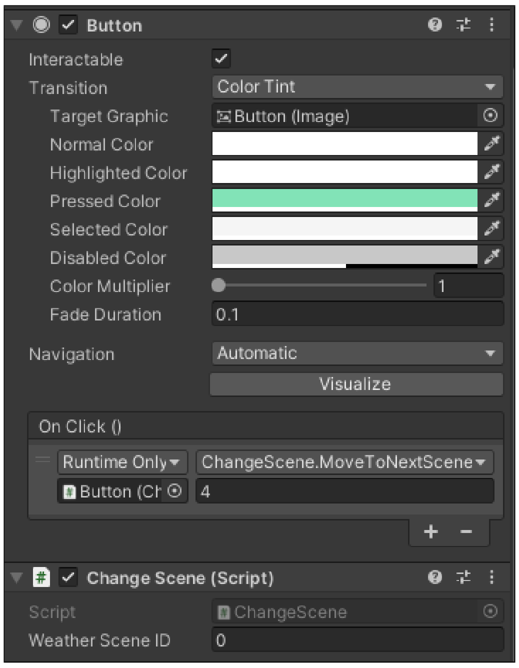
Figure 5.25: *MoveToWeatherMenu* - sequence diagram

Finally, to use the *NextSceneButton* to switch menus, the *ChangeScene* script needs to be attached to the buttons in every *Scene* so that it can change between *Scenes* upon click. The button allows the user to move to the next menu but not to the previous one. The button configurations have a setting on Unity where a method can be configured to be called upon clicking the button. When on the Main menu, *NextSceneButton* only allows the prototype to change to the correct Weather menu according to the API, therefore the *MoveToWeatherMenu* needs to be selected as seen in Figure 5.26 (a). On the remaining menus, the *MoveToNextMenu* can be called, and the correct *Scene* index must be provided as a parameter to the method. The transition between the Weather menu and the Setpoint menu, whose scene index is 4, is illustrated by Figure 5.26 (b).



```
Button
Interactable        ✓
Transition          Color Tint
  Target Graphic    NextScene (Image)
  Normal Color
  Highlighted Color
  Pressed Color
  Selected Color
  Disabled Color
  Color Multiplier  1
  Fade Duration     0.1
Navigation          Automatic
                    Visualize
On Click ()
Runtime Only   ChangeScene.MoveToWeatherMe
NextScene
Change Scene (Script)
Script              ChangeScene
Weather Scene ID    0
```

(a) Main menu.



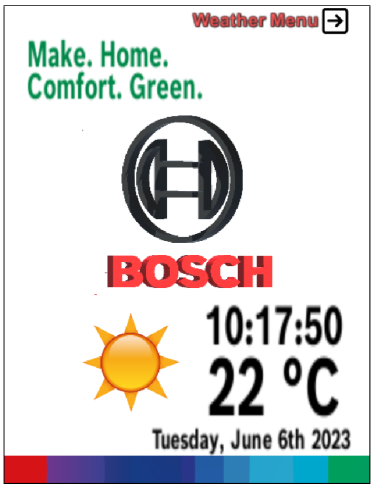
```
Button
Interactable        ✓
Transition          Color Tint
  Target Graphic    Button (Image)
  Normal Color
  Highlighted Color
  Pressed Color
  Selected Color
  Disabled Color
  Color Multiplier  1
  Fade Duration     0.1
Navigation          Automatic
                    Visualize
On Click ()
Runtime Only   ChangeScene.MoveToNextScene
Button (Ch     4
Change Scene (Script)
Script              ChangeScene
Weather Scene ID    0
```

(b) Weather menu (sunny).

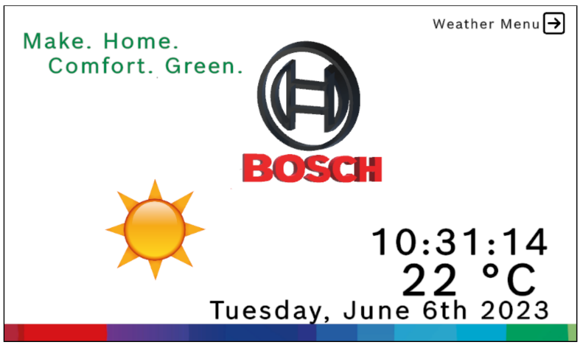
Figure 5.26: *NextSceneButton* - configurations.

5.3.2 Main menu

This is the first menu that appears after starting up the prototypes. This menu focuses on the 3D Bosch logo rotating in the middle of the screen to emphasize the brand and its 2D slogan, "Make. Home. Comfort. Green.". At the bottom of the screen, one can see information obtained from the APIs used: date, time, weather condition and outside temperature. In the top right corner, it is possible to see what will be the next menu. These features can be seen in Figure 5.27 (a) and (b). Apart from the fact that the top right corner information works as a digital touch button on the transparent prototype, the Main menus of both prototypes are coded in the same way besides the fact that the elements were re-positioned to fit the different resolutions of the displays.



(a) Holographic prototype.

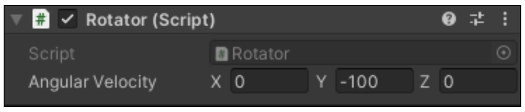


(b) Transparent prototype.

Figure 5.27: Main menus on both prototypes.

The API data is simply displayed on text files created under the *Canvas* of this Unity *Scene*. The emoji referring to the weather is also displayed through a text file. At the beginning of this chapter, a *Sprite Asset* was created using three emoji referring to sun, rain and snow weather. A different ID from 0 to 2 was assigned to each *Sprite* as previously seen in Figure 5.6, therefore by writing *"<sprite=X>* (where X is either 0, 1 or 2) on a text file the proper emoji is shown.

The Bosch logo is the only 3D object not imported from Sketchfab or Unity Asset Store, and was instead made from scratch on Blender. It was necessary to convert a Bosch logo image into .svg format, insert it into Blender and use the Extrude function to create depth. Then, the file was exported and subsequently imported into Unity without any errors. To make it rotate, a script file named *Rotator* that came with the demo project was used. This script simply sets the angular velocity in each of the axis and applies a rotation using that same velocity. In this case, the logo only rotates in the Y axis, so the X and Z axis have no velocity, as seen in Figure 5.28.


Rotator (Script)

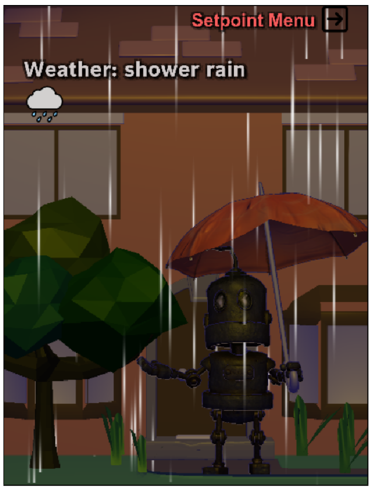
Script: Rotator

Angular Velocity: X 0 Y -100 Z 0

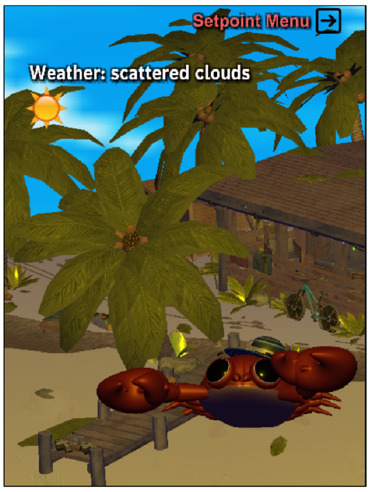
Figure 5.28: *Rotator* script - configurations

5.3.3 Weather menus

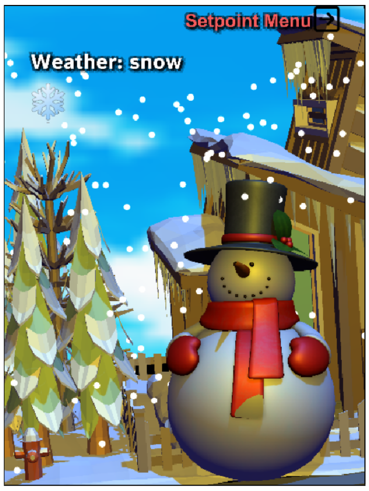
The Weather menu follows the previous Main menu and displays a small amount of information regarding the weather conditions, emphasizing the creation of an environment relatable to the current weather. Depending on the weather condition, one of the three menus shown in Figures 5.29 (a), (b) and (c), and 5.30 (a), (b) and (c), will be displayed according to the information in the previously mentioned Table 5.1.



(a) Weather menu: rain

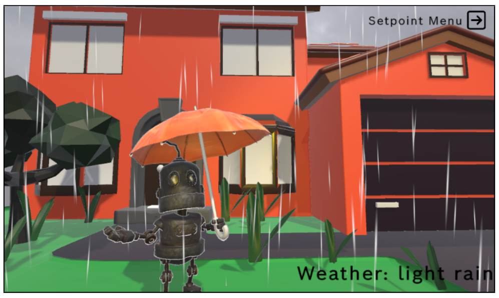


(b) Weather menu: sunny

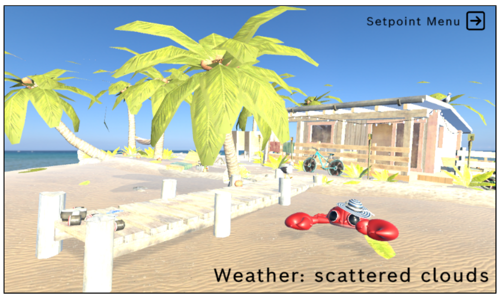


(c) Weather menu: snow

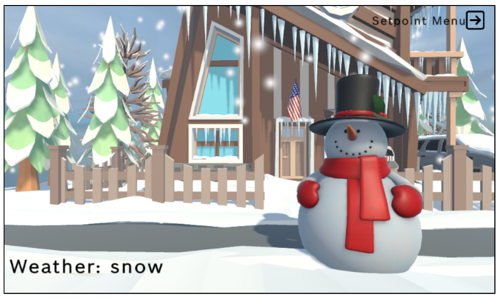
Figure 5.29: All Weather menus of the holographic prototype.



(a) Weather menu: rain



(b) Weather menu: sunny



(c) Weather menu: snow

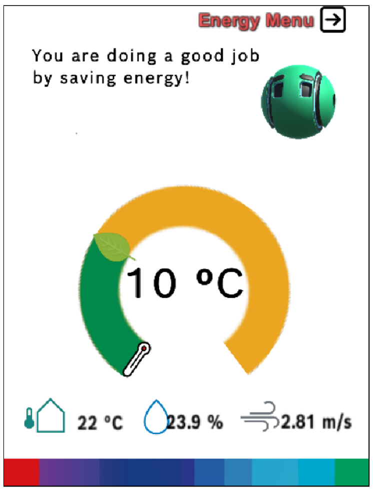
Figure 5.30: All Weather menus of the transparent prototype.

Similarly to the Main menu, the Weather menu of both prototypes were coded in the same way. The current weather condition is displayed through a text file like on the previous menu. The scenarios and 3D models all came from Sketchfab.com and were directly imported into Unity. This website contains some realistic and detailed models, but low-poly and cartoon-like 3D assets were chosen for both prototypes. These elements are built using a small number of polygons, therefore the name low-poly. Elements that use a high amount of polygons were not a valid option due to their high computational requirements, making the *Scenes* very slow and decreasing the performance of the prototype.

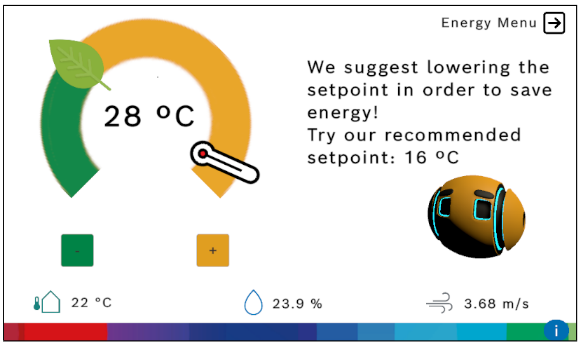
On the Rain and Snow menus, the falling rain and snow were created using a *Particle System*[21] to emphasize the weather condition. On the Sunny menu, the light source was increased to resemble a sunny day.

5.3.4 Setpoint menu

Following the Weather menu, both prototypes present the Setpoint menu seen in Figure 5.31 (a) and (b). This is one of the two menus that the user can actually interact with (besides switching between menus), while also being one of the menus with the highest amount of information. Here the user can get more details regarding the weather conditions like: outside temperature, humidity and wind speed. Similarly to the Main menu, this information is simply put into a text file and displayed directly on the prototype, accompanied by icons that resemble the transmitted information. In the case of the holographic display, the Z-axis coordinates of these icons are different from the other objects to create a sense of depth.



(a) Holographic prototype


Energy Menu
We suggest lowering the setpoint in order to save energy!
Try our recommended setpoint: 16 °C
28 °C
-
+
22 °C
23.9 %
3.68 m/s

(b) Transparent prototype

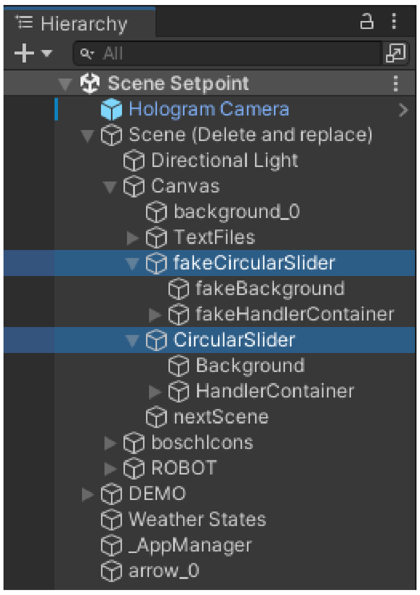
Figure 5.31: Setpoint menus on both prototypes.

[21] https://docs.unity3d.com/ScriptReference/ParticleSystem.html

A circular slider is placed in the center of the prototype, and it is the main focus of this menu, allowing the user to change the setpoint at will (from 10ºC up to 30ºC). The transparent prototype also has two buttons that can be used to interact with the slider. The value of this setpoint is displayed in the center of the slider and is updated whenever the user interacts with it. The slider has two main areas divided by the recommended setpoint and distinguishable by their green and orange colors, meaning that the selected setpoint is either below, above or on the recommended temperature. This menu also includes a virtual robot imported from Unity Asset Store that works as an assistant. The robot changes its color and provides a piece of advice to the user depending on what area of the slider the selected setpoint is.

5.3.4.1 Slider design

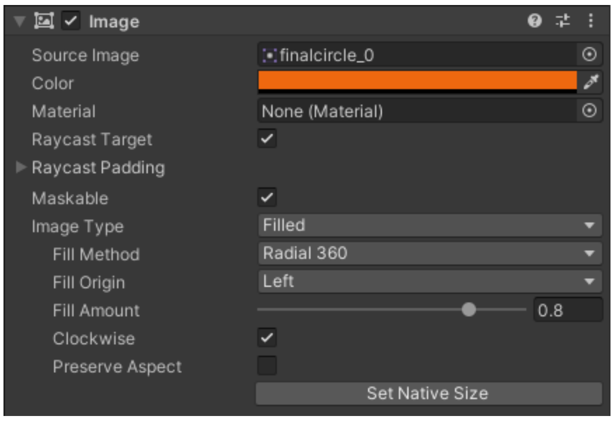
An overview of the slider creation process can be seen in Figure 5.32. To create the slider, an *Image*[22] object named *CircularSlider* was created inside the *Canvas*. The source image for this object is a thick white ring with no background that was converted into a 2D *Sprite*. As shown by Figure 5.33, the image type of this object was updated to *Filled*, the fill method to *Radial 360* and the fill origin to *Left*. This makes it so that the image appears from left to right if the fill amount increases and disappears from right to left if the fill amount decreases. The fill amount is the parameter that the user will interact with when using the slider or the buttons.



```
Scene Setpoint
  Hologram Camera
  Scene (Delete and replace)
    Directional Light
    Canvas
      background_0
      TextFiles
      fakeCircularSlider
        fakeBackground
        fakeHandlerContainer
      CircularSlider
        Background
        HandlerContainer
      nextScene
    boschIcons
    ROBOT
  DEMO
  Weather States
  _AppManager
  arrow_0
```

Figure 5.32: Setpoint menu hierarchy - holographic prototype

[22] https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Image.html



Image

Source Image	finalcircle_0
Color	🟧
Material	None (Material)
Raycast Target	✓
Raycast Padding	
Maskable	✓
Image Type	Filled
Fill Method	Radial 360
Fill Origin	Left
Fill Amount	0.8
Clockwise	✓
Preserve Aspect	

Figure 5.33: *CircularSlider* - configuration

The fact that the slider starts to disappear according to its fill amount is a problem since it completely disappears when this value reaches zero, and the user might have difficulties clicking on it again. To counter this, a new object named *Background* was created, and as the name suggests, it acts as the background color for the actual slider. It has the exact configurations of its parent object, *CircularSlider*, but with a different color so that the change is perceptible. The fill amount of *Background* will be static, meaning that the user will not interact with it.

Lastly, an object named *HandlerContainer* was created, and it contains another object named *Thermometer*, which will act as the main handler of the slider and move around it depending on the increment or decrement of the fill amount caused by the user interaction. As the name suggests, this handler is designed as a thermometer.

This approach would be enough for the slider to work alongside a script file, but since it needs to identify the recommended temperature on the slider, it needs a few changes. To deal with this, two sliders were created: a main one that is invisible and a "fake" one that is visible. The main slider is the one that the user will interact with. Every object of this slider is invisible except for the main handler (thermometer). The fake slider is totally visible and is set at startup so that the fake handler (a leaf) stays in the correct position for the recommended temperature, dividing the fake slider into two regions: one green below the recommended setpoint and the other orange above the recommended setpoint. This means that when the user moves the main handler, it will be visible that the selected setpoint is either below, above or on the recommended temperature.

Unity has a *Slider*[23] class that could be used to develop the sliders but, unfortunately, during this project, it was not possible to have all the desired features and aesthetics working efficiently while using this class. Therefore, another method had to be found, leading to the solution that was just described. Both prototypes use this solution and differ only on the script file that is attached to *CircularSlider*. These files control the way of interacting with the slider on each prototype.

[23] https://docs.unity3d.com/2018.2/Documentation/ScriptReference/UI.Slider.html

5.3.4.2 Slider in the holographic display

In the holographic prototype, the user presses the capacitive buttons *Back* or *Forward* to interact with the slider. The first button will decrease the setpoint value while the other increases it. The *Update* method of the *CircularSlider* script (that is attached to the object of the same name) handles all the processes necessary to update the slider using the capacitive buttons and can be seen in Figure 5.34. It starts by updating the fill amount of the fake slider to a custom value that allows the fake handler to be placed on top of the recommended temperature. Next, the method increases or decreases the slider value depending on which button was pressed (if it was pressed). This press is recognised through the *GetButtonDown* method that was previously mentioned during the *LGKSceneSwitcher* script. The slider value is then printed on the correct text file, and the main handler (thermometer) is updated to stay on top of the current temperature. This is done by simply updating the fill amount of the slider and rotating the main handler with the help of a directional vector. Finally, the slider value is stored so this information is not lost upon switching menus.

At the same time, the robot color and advice are constantly being updated through the *Update* method on the *Advice* script. This method, seen in Figure 5.35, gets all the setpoint values (minimum, maximum, recommended and selected) and starts by checking if the selected setpoint is below the recommended setpoint (visually set by the leaf handler). If so, the robot assistant color changes to green (using the *SetColor*[24] method) and displays a message saying "You are doing a good job by saving energy!". If the handler is right on top of the recommended temperature, a message saying "Thank you for selecting our recommended temperature!" will be displayed, and the robot will stay green. On the other hand, if the temperature is above the recommended setpoint, the robot turns orange, and a different message will appear: "We suggest lowering the setpoint in order to save energy! Try our recommended setpoint: x", where x is the recommended setpoint value.

[24] https://docs.unity3d.com/ScriptReference/Material.SetColor.html

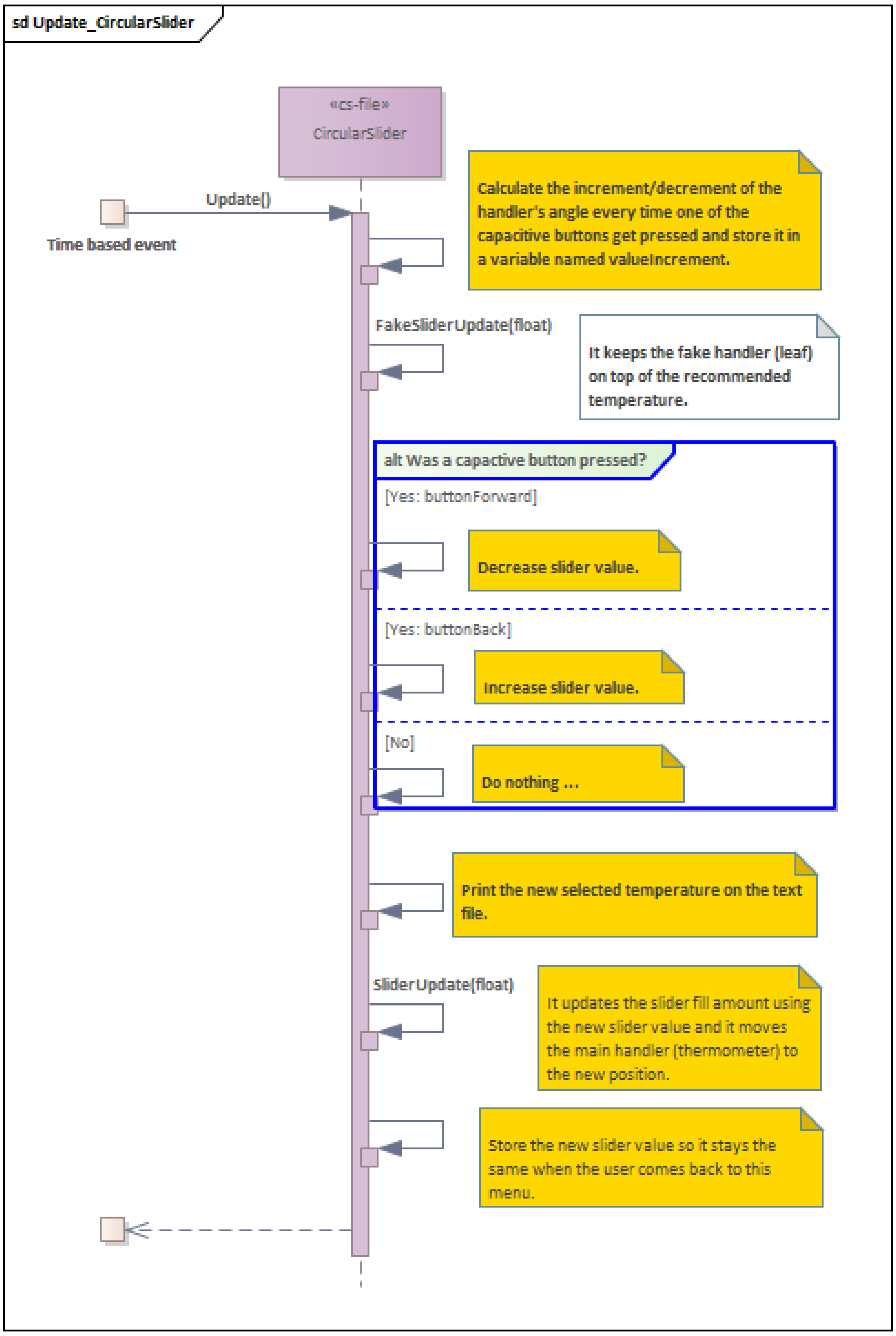

sd Update_CircularSlider
«cs-file»
CircularSlider
Update()
Time based event
Calculate the increment/decrement of the handler's angle every time one of the capacitive buttons get pressed and store it in a variable named valueIncrement.
FakeSliderUpdate(float)
It keeps the fake handler (leaf) on top of the recommended temperature.
alt Was a capactive button pressed?
[Yes: buttonForward]
Decrease slider value.
[Yes: buttonBack]
Increase slider value.
[No]
Do nothing ...
Print the new selected temperature on the text file.
SliderUpdate(float)
It updates the slider fill amount using the new slider value and it moves the main handler (thermometer) to the new position.
Store the new slider value so it stays the same when the user comes back to this menu.

Figure 5.34: *Update* (CircularSlider) - sequence diagram

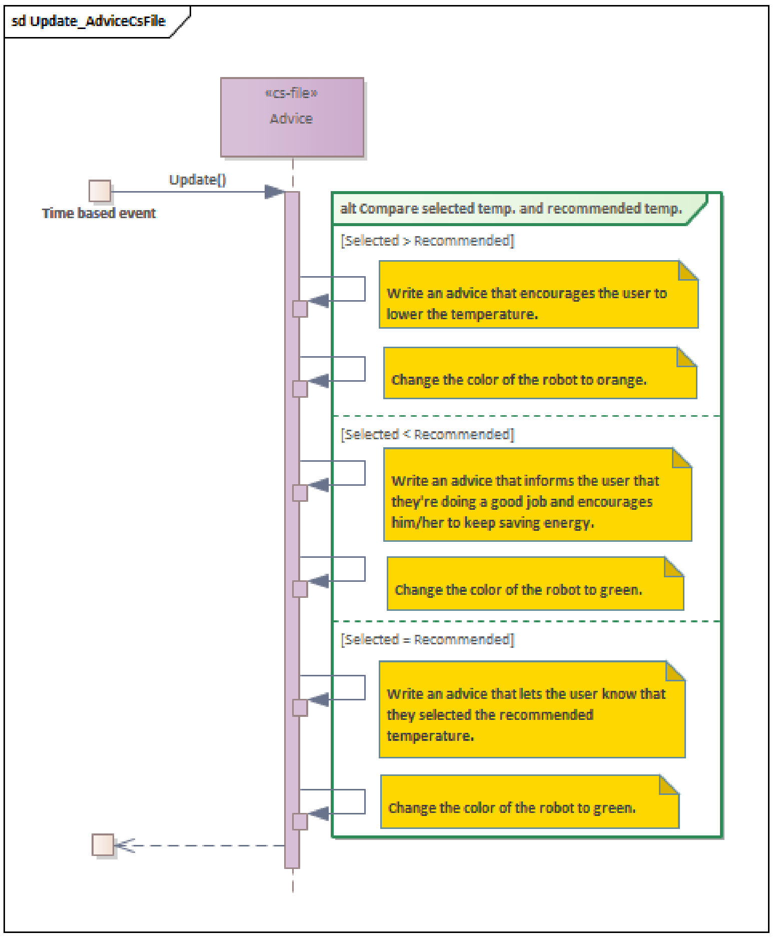

sd Update_AdviceCsFile
«cs-file»
Advice
Update()
Time based event
alt Compare selected temp. and recommended temp.
[Selected > Recommended]
Write an advice that encourages the user to lower the temperature.
Change the color of the robot to orange.
[Selected < Recommended]
Write an advice that informs the user that they're doing a good job and encourages him/her to keep saving energy.
Change the color of the robot to green.
[Selected = Recommended]
Write an advice that lets the user know that they selected the recommended temperature.
Change the color of the robot to green.

Figure 5.35: *Update* (Advice) - sequence diagram

5.3.4.3 Slider in the transparent display

On the transparent prototype, the user can interact with the slider in two different ways: either directly using the touchscreen to drag the slider handler or pressing the buttons on the bottom of the slider to increase or decrease the setpoint value. The buttons serve as a way to help people with difficulty selecting a specific value through the slider drag, as well as to simply provide an alternative if the user does not enjoy dragging the slider handler.

The methods responsible for enabling the slider touchscreen drag are inserted on the *RadialFill* script that is attached to the *CircularSlider* object. This script has a *Start* method that runs on the startup of this menu, and it is responsible for updating the fill amount of the slider and then updating the main handler to match this value. A similar process is made to keep the fake handler on top of the recommended temperature, and it is done in the same way as in the holographic prototype: setting a custom fill amount for the fake slider. This method can be seen in Figure 5.36.

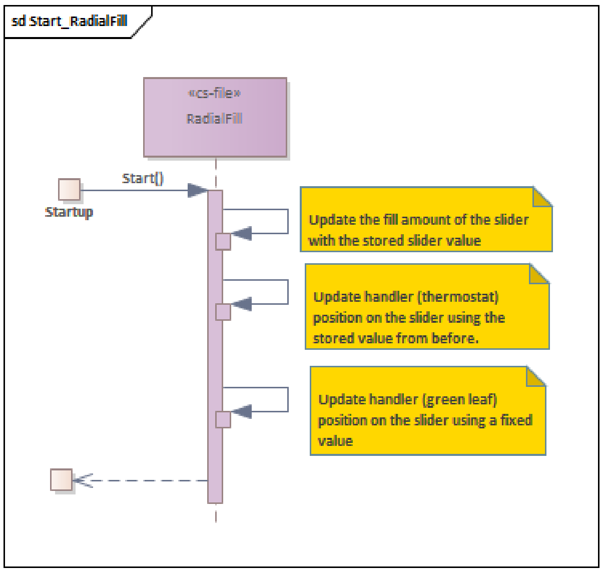

sd Start_RadialFill
«cs-file»
RadialFill
Start()
Startup
Update the fill amount of the slider with the stored slider value
Update handler (thermostat) position on the slider using the stored value from before.
Update handler (green leaf) position on the slider using a fixed value

Figure 5.36: *Start* (RadialFill) - sequence diagram

Upon touching the touchscreen, the *OnPointerDown* method is called. This simple method can be seen in Figure 5.37, and it simply checks if the touch was made on top of the slider zone or not. If the touch was made out of that zone, it simply does nothing. Otherwise, it calls the *OnDrag* method.

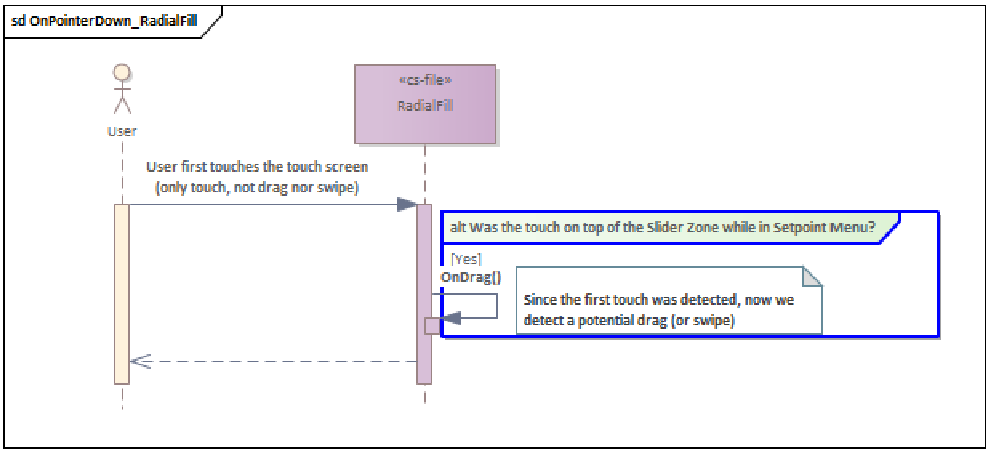

sd OnPointerDown_RadialFill
User
«cs-file»
RadialFill
User first touches the touch screen (only touch, not drag nor swipe)
alt Was the touch on top of the Slider Zone while in Setpoint Menu?
[Yes]
OnDrag()
Since the first touch was detected, now we detect a potential drag (or swipe)

Figure 5.37: *OnPointerDown* - sequence diagram

The *OnDrag* method, which can be seen in Figure 5.38, pinpoints the touch to its exact location to then create a directional vector that will allow the method to determine the slider angle where the main handler should be placed. The fill amount of the slider is then calculated through this angle. Lastly, the slider fill amount is updated, and the main handler is re-positioned to match the new value.

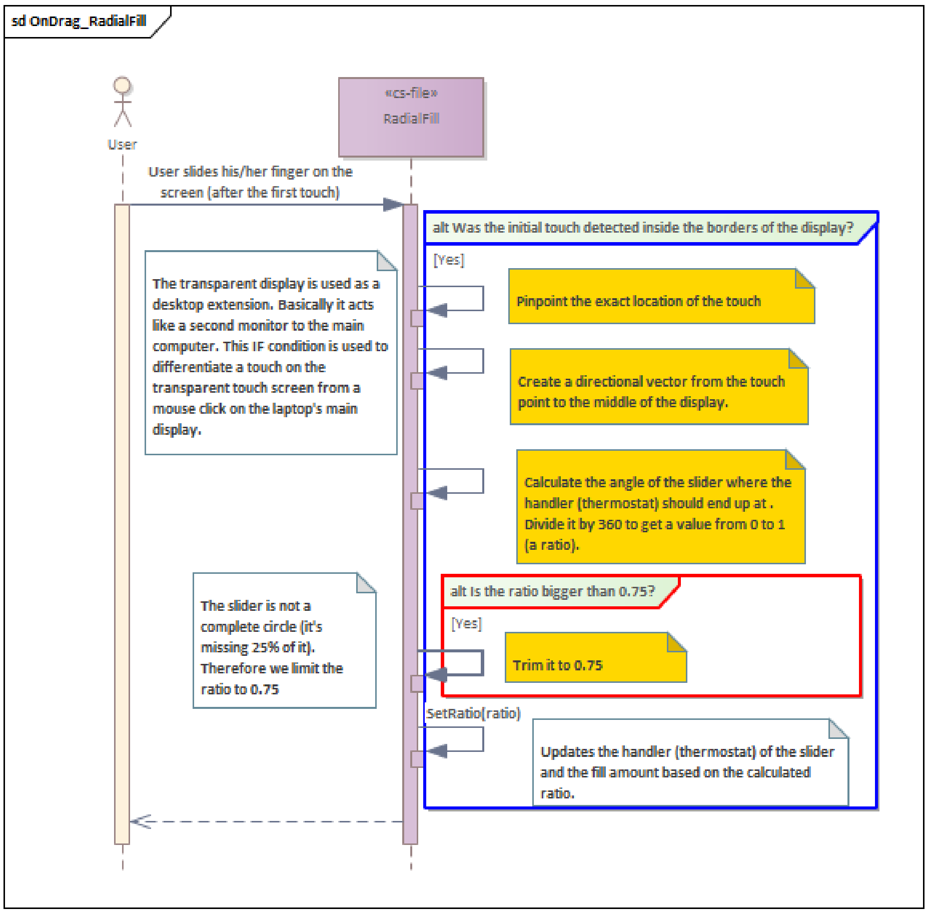

sd OnDrag_RadialFill
User
«cs-file»
RadialFill
User slides his/her finger on the screen (after the first touch)
The transparent display is used as a desktop extension. Basically it acts like a second monitor to the main computer. This IF condition is used to differentiate a touch on the transparent touch screen from a mouse click on the laptop's main display.
alt Was the initial touch detected inside the borders of the display?
[Yes]
Pinpoint the exact location of the touch
Create a directional vector from the touch point to the middle of the display.
Calculate the angle of the slider where the handler (thermostat) should end up at . Divide it by 360 to get a value from 0 to 1 (a ratio).
The slider is not a complete circle (it's missing 25% of it). Therefore we limit the ratio to 0.75
alt Is the ratio bigger than 0.75?
[Yes]
Trim it to 0.75
SetRatio(ratio)
Updates the handler (thermostat) of the slider and the fill amount based on the calculated ratio.

Figure 5.38: *OnDrag* - sequence diagram

The *Plus* (+) and *Minus* (-) buttons are alternatives to the slider drag in order to change the setpoint value. These buttons were created using the *Button* class and were correctly positioned at the bottom of the slider. Both buttons have a setting on Unity where a method can be configured to be called upon clicking the button. The methods used are the *MinusButtonAction* and the *PlusButtonAction*. These methods can be seen in Figures 5.39 and 5.40, and they are very similar to each other. The *MinusButtonAction* starts by checking if the selected temperature is already minimum because if so, it does nothing since it can not go any lower. If it is not minimum, then it decreases the fill amount of the slider in a way that the setpoint value is reduced by one degree, and the main handler is updated accordingly. This new value is then stored so that it is saved even if the prototype changes to other *Scenes*. The *PlusButtonAction* is very similar, it starts by verifying if the selected temperature is already maximum, and if it is not, then it increases the fill amount of the slider in a way that the setpoint value is increased by one degree.

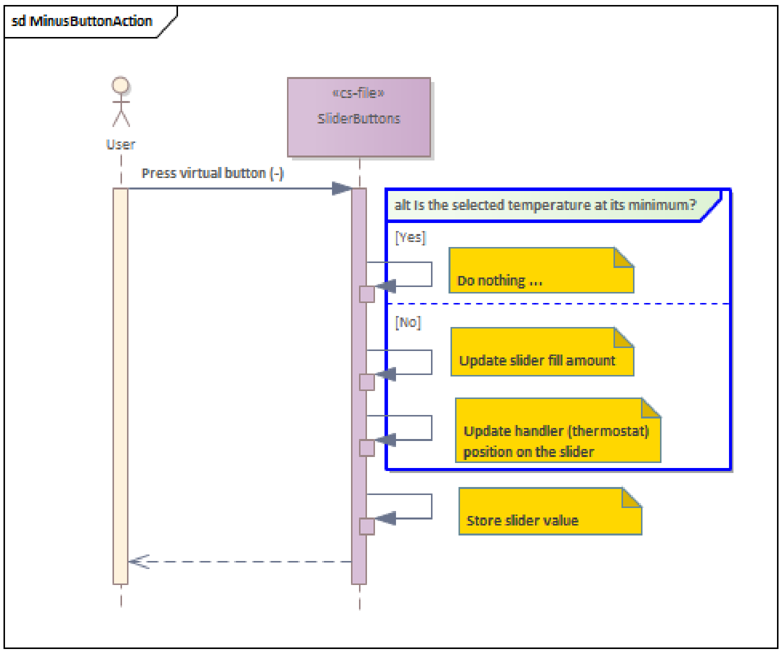

sd MinusButtonAction
User
«cs-file»
SliderButtons
Press virtual button (-)
alt Is the selected temperature at its minimum?
[Yes]
Do nothing ...
[No]
Update slider fill amount
Update handler (thermostat) position on the slider
Store slider value

Figure 5.39: *MinusButtonAction* - sequence diagram

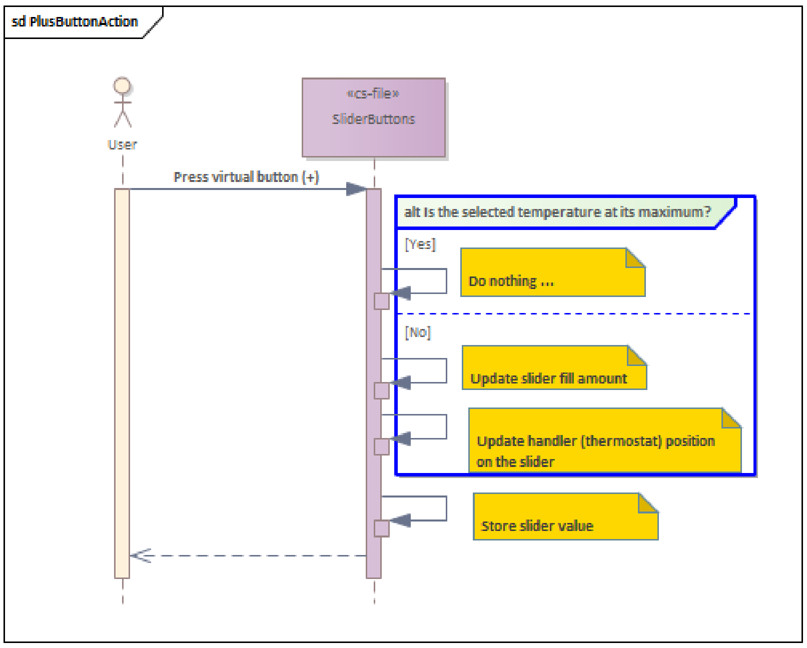

sd PlusButtonAction
User
«cs-file»
SliderButtons
Press virtual button (+)
alt Is the selected temperature at its maximum?
[Yes]
Do nothing ...
[No]
Update slider fill amount
Update handler (thermostat) position on the slider
Store slider value

Figure 5.40: *PlusButtonAction* - sequence diagram

The default temperature unit for both prototypes is the Celsius degree, but the transparent prototype has a way to change between Celsius (ºC) and Fahrenheit (ºF) degrees. A button named *F_to_C_Button* was created using the *Button* class and was positioned in the middle of the slider, where the slider value is displayed. This button transparency was set to make it invisible. An external script, named *Button Long Press Listener*[25], was introduced since a long press would suit this use case better than a regular press in order to prevent accidental and undesired changes to the temperature unit. This script and another script named *Advice* are attached to the *F_to_C_Button*. Figure 5.41 illustrates the *FtoCButtonAction* method responsible for storing the information about which temperature unit has been selected.

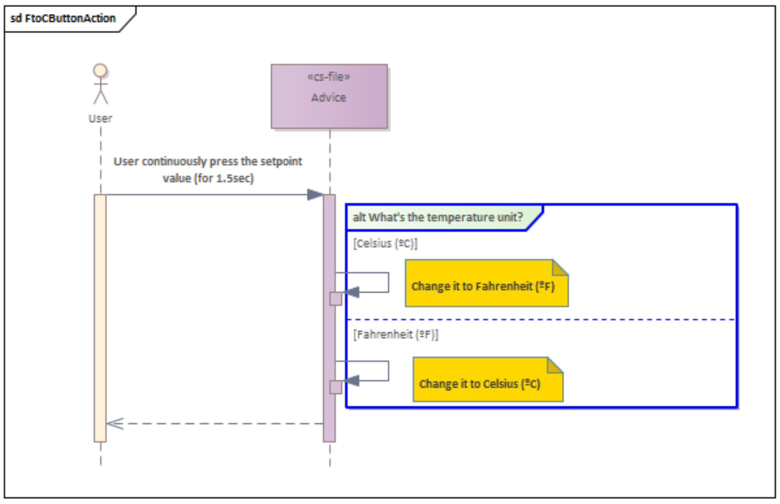

sd FtoCButtonAction
User
«cs-file»
Advice
User continuously press the setpoint value (for 1.5sec)
alt What's the temperature unit?
[Celsius (ºC)]
Change it to Fahrenheit (ºF)
[Fahrenheit (ºF)]
Change it to Celsius (ºC)

Figure 5.41: *FtoCButtonAction* - sequence diagram

The setpoint value displayed on the center of the slider will change in real-time upon the user's interaction with the slider. This change is handled by the *Update* method of the *Advice* script since it is called every second. This method is described in Figure 5.42, and it starts by checking the temperature unit to set the maximum, minimum and recommended temperature in Celsius or Fahrenheit. After that, it calculates the selected temperature based on the fill amount of the slider so that it can finally display this value on a text file located at the center of the slider.

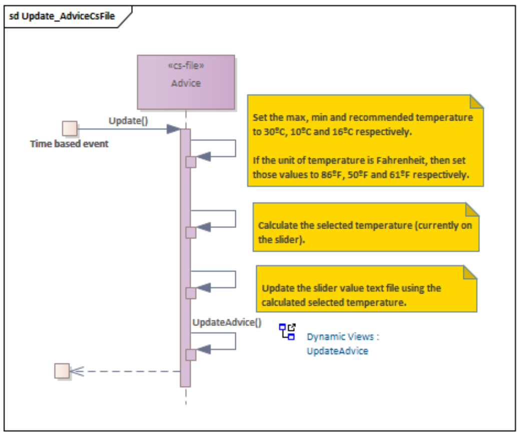

sd Update_AdviceCsFile
«cs-file»
Advice
Update()
Time based event
Set the max, min and recommended temperature to 30ºC, 10ºC and 16ºC respectively.
If the unit of temperature is Fahrenheit, then set those values to 86ºF, 50ºF and 61ºF respectively.
Calculate the selected temperature (currently on the slider).
Update the slider value text file using the calculated selected temperature.
UpdateAdvice()
Dynamic Views : UpdateAdvice

Figure 5.42: *Update* (Advice) - sequence diagram

[25] https://github.com/herbou/Unity_ButtonEvents

The last thing this method does is call another method named *UpdateAdvice*, which can be seen in Figure 5.43. This method starts by checking if the instructions of this menu are being displayed, and if so, it temporarily deletes the content on the advice text file for the user to better read the instructions. If the instructions are not being displayed, then it starts doing the same steps as the *Update* method of the *Advice* script from the holographic prototype: updates the robot color and the advice it gives based on the selected setpoint.

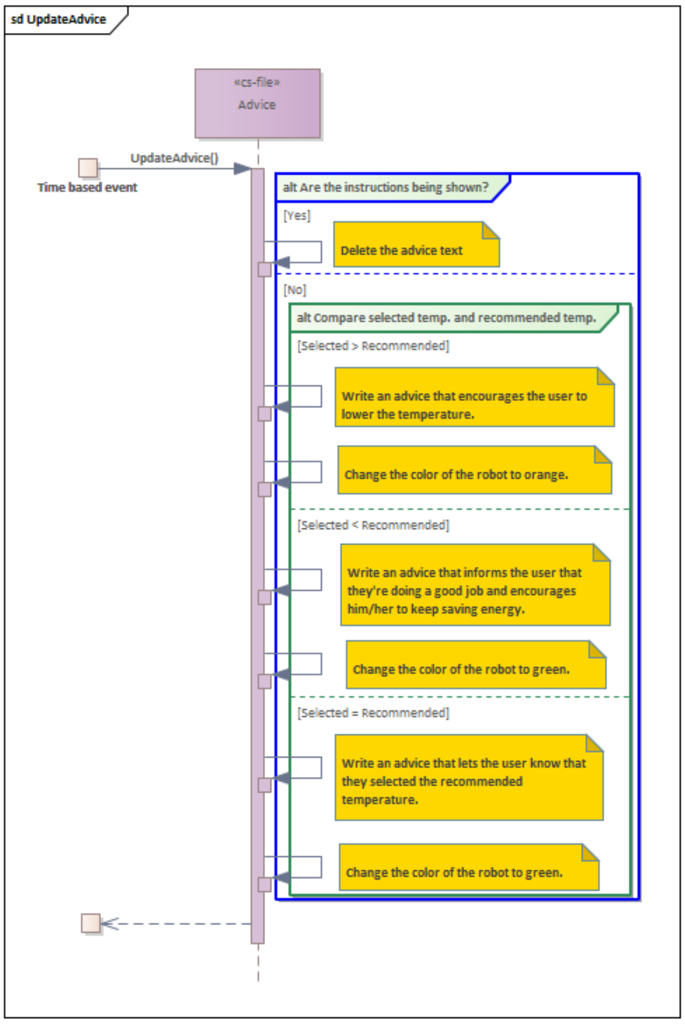

sd UpdateAdvice
«cs-file»
Advice
UpdateAdvice()
Time based event
alt Are the instructions being shown?
[Yes]
Delete the advice text
[No]
alt Compare selected temp. and recommended temp.
[Selected > Recommended]
Write an advice that encourages the user to lower the temperature.
Change the color of the robot to orange.
[Selected < Recommended]
Write an advice that informs the user that they're doing a good job and encourages him/her to keep saving energy.
Change the color of the robot to green.
[Selected = Recommended]
Write an advice that lets the user know that they selected the recommended temperature.
Change the color of the robot to green.

Figure 5.43: *UpdateAdvice* - sequence diagram

As mentioned before, this Setpoint menu has instructions, which can be seen in Figure 5.44. The instructions can be accessed by the *InfoButton* that was created using the *Button* class and was positioned in the bottom right corner of the menu. A *Panel* object was also added to the *Scene* to be displayed when reading the instructions, so that the background information is blurred and the user merely focuses on the instructions. The *InfoButton* also has a setting on Unity where a method can be configured to be called upon clicking the button. The method called upon a click is named *InfoButtonAction* and can be seen in Figure 5.45. This method starts by checking if the panel is active or not. If the panel is deactivated when the *InfoButton* is pressed, it means that the instructions are not being displayed. Therefore, the panel must be activated, and the instructions must be written on the corresponding text files that are already correctly placed on the *Scene*. If the panel is activated, it means that the *InfoButton* press was meant to exit the instructions, therefore the panel must be deactivated again, and the instructions must be temporarily deleted from the text files. The menu provides instructions that teach the user to interact with the slider by dragging it or by using the *Plus* (+) and *Minus* (-) buttons, as well as to change the setpoint value from Celsius to Fahrenheit and the other way around.

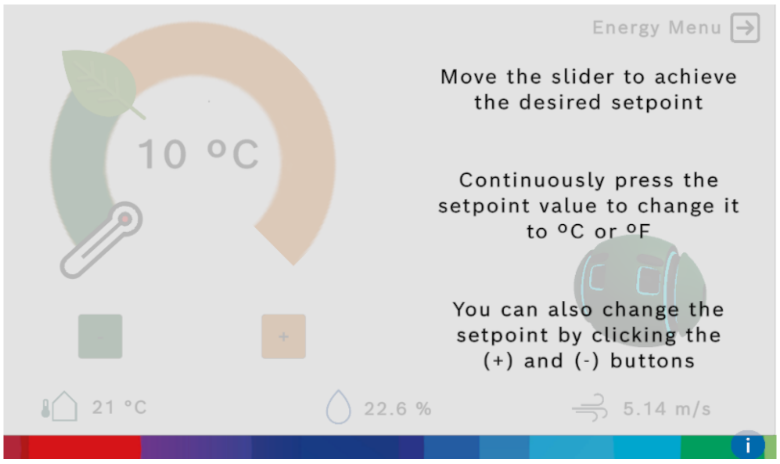

Energy Menu
10 °C
Move the slider to achieve the desired setpoint
Continuously press the setpoint value to change it to °C or °F
You can also change the setpoint by clicking the (+) and (-) buttons
21 °C
22.6 %
5.14 m/s

Figure 5.44: Instructions of Setpoint menu

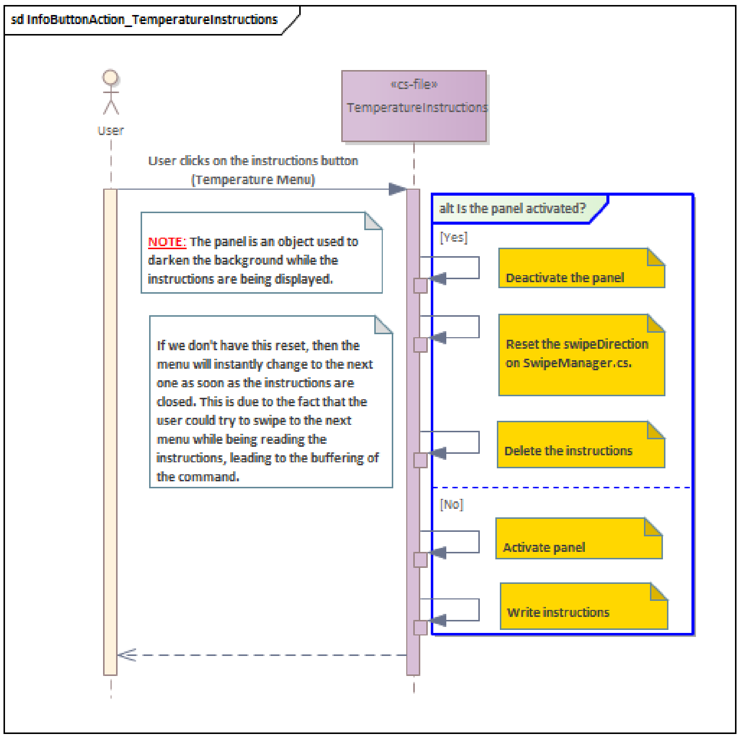
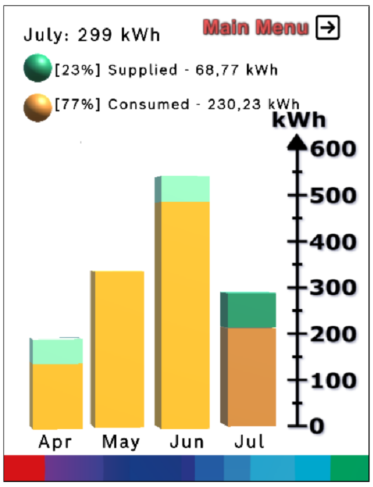

sd InfoButtonAction_TemperatureInstructions
User
«cs-file» TemperatureInstructions
User clicks on the instructions button (Temperature Menu)
NOTE: The panel is an object used to darken the background while the instructions are being displayed.
If we don't have this reset, then the menu will instantly change to the next one as soon as the instructions are closed. This is due to the fact that the user could try to swipe to the next menu while being reading the instructions, leading to the buffering of the command.
alt Is the panel activated?
[Yes]
Deactivate the panel
Reset the swipeDirection on SwipeManager.cs.
Delete the instructions
[No]
Activate panel
Write instructions

Figure 5.45: *InfoButtonAction* (TemperatureInstructions) - sequence diagram

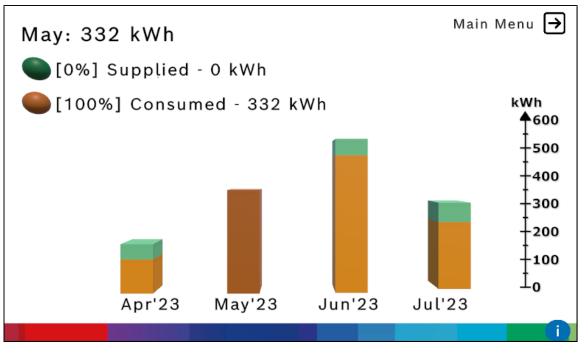
5.3.5 Energy menu

The Energy menu is the last one in the menu loop of both prototypes. This menu can be seen in Figure 5.46 (a) and (b), and it contains a bar graph that shows the household energy consumption in the last four months. It separates the amount of Kilowatt hour (kWh) into consumed and supplied energy, meaning that the latter refers to the energy produced by devices like solar panels. In contrast, the first one refers merely to the consumed energy, as the name suggests.



July: 299 kWh
[23%] Supplied - 68,77 kWh
[77%] Consumed - 230,23 kWh

(a) Holographic prototype



May: 332 kWh
[0%] Supplied - 0 kWh
[100%] Consumed - 332 kWh

(b) Transparent prototype

Figure 5.46: Energy menus on both prototypes.

The bar graphs are composed of *Cube*[26] classes, that were positioned and adjusted correctly to the *Scene*. Each bar consists of two *Cubes*: one for the supplied energy and the other for the consumed, which can be zero as shown in the month of May in Figure 5.46 (a) and (b).

This is the second and last menu that the user can interact directly with. The user can loop through the last four months (bars) to check the energy values of those months, as well as the total energy spent, by subtracting the supplied energy from the consumed energy. The selected month gets the color of its bar changed, as visible in the previous Figure 5.46 (a) and (b).

5.3.5.1 Energy menu in the holographic display

In the holographic display, the user can select a bar by simply pressing the *Back* and *Forward* capacitive buttons of Looking Glass Portrait. The detection of this button press is done through the *GetButtonDown* method that was also used in the *LGKSceneSwitcher* and *CircularSlider* scripts.

The *BarGraphColor* script processes every essential information on its *Update* method. This method can be seen in Figure 5.47 and starts by updating a variable that represents the bar (month) selected on the bar graph. Pressing the *Forward* button will decrease this value, and pressing the *Back* button will increase it (in the range of 1 to 4.). When this variable has the value 1, the selected bar is the left-most one, and when it has the value 4, the selected bar is the right-most one. This

[26]https://docs.unity3d.com/ScriptReference/PrimitiveType.Cube.html

variable is checked in order to change the color of the selected bar to a darker tone to imply that this is the selected bar. It also sets the color of the remaining bars to lighter tones. It updates the text files with specific information about the consumption of the selected month.

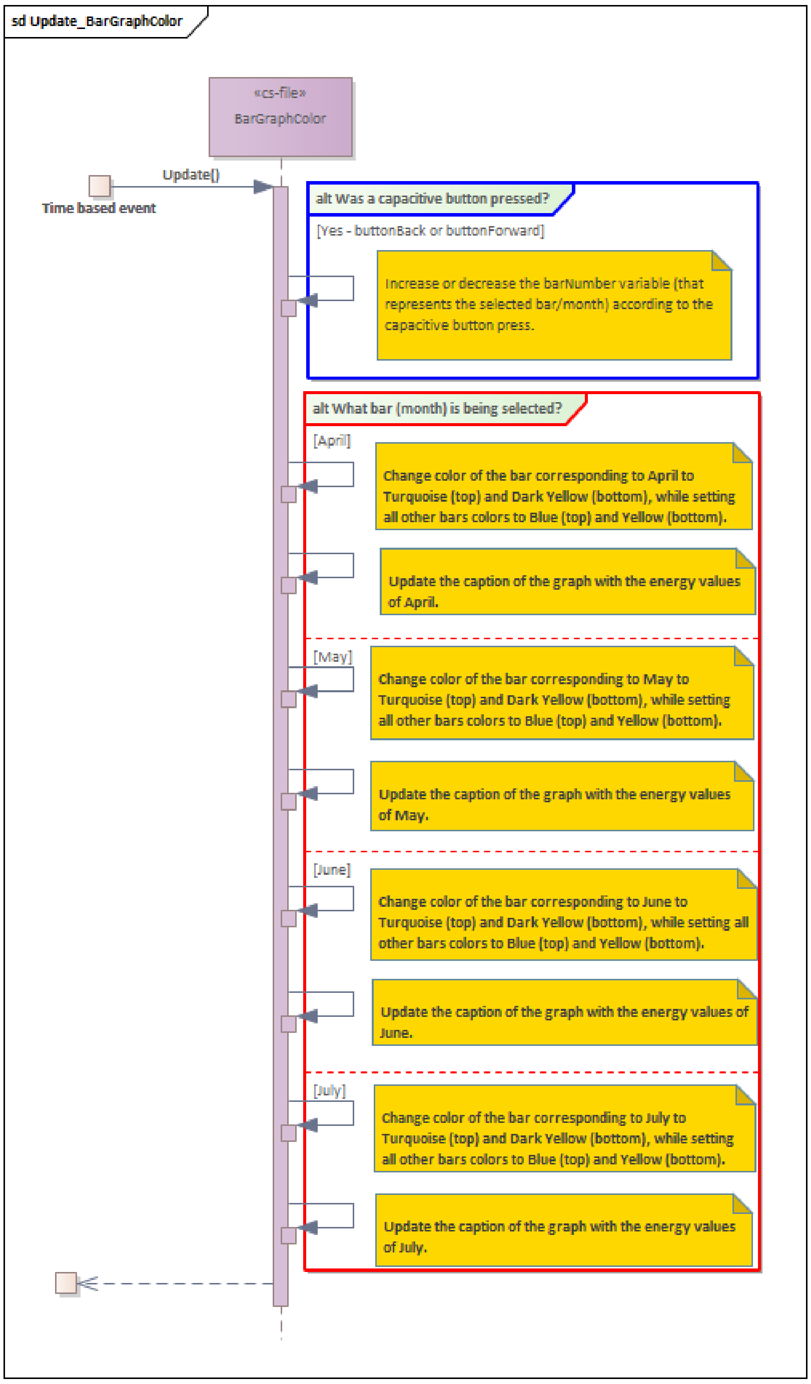

sd Update_BarGraphColor
«cs-file»
BarGraphColor
Update()
Time based event
alt Was a capacitive button pressed?
[Yes - buttonBack or buttonForward]
Increase or decrease the barNumber variable (that represents the selected bar/month) according to the capacitive button press.
alt What bar (month) is being selected?
[April]
Change color of the bar corresponding to April to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of April.
[May]
Change color of the bar corresponding to May to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of May.
[June]
Change color of the bar corresponding to June to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of June.
[July]
Change color of the bar corresponding to July to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of July.

Figure 5.47: *Update* (BarGraphColor) - sequence diagram

5.3.5.2 Energy menu in the transparent display

The difference between the Energy menu in the holographic and transparent displays is that, on the transparent one, the monthly consumption can be looped using the user touch on the bar graph. Another notable difference is that the transparent prototype has instructions similar to the Setpoint menu. These instructions can be seen in Figure 5.48.

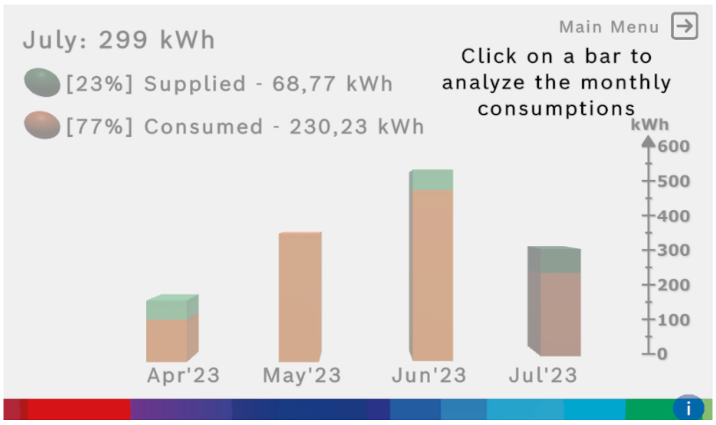

July: 299 kWh
[23%] Supplied - 68,77 kWh
[77%] Consumed - 230,23 kWh
Main Menu
Click on a bar to analyze the monthly consumptions
kWh
600
500
400
300
200
100
0
Apr'23
May'23
Jun'23
Jul'23
i

Figure 5.48: Instructions of Energy menu

For the user to select a bar on the transparent prototype, four scripts named *TouchBar1*, *TouchBar2*, *TouchBar3* and *TouchBar4* were created and attached to the pairs of bars corresponding to the same month. As seen in Figure 5.49, this script only consists of one very simple method that is called *OnMouseDown*, which updates the variable that represents the bar (month) selected on the bar graph. This variable is located on another script named *BarGraphColor*, and this script is very similar to the one used on the holographic prototype. The *Update* method of this script can be seen in Figure 5.50. One of the main differences is that the mentioned variable is not updated upon a button touch, given that this is already being handled by the *TouchBar* scripts. The other difference is that it also checks if the instructions of this menu are being shown to the user because if they are, then the user interaction with the bars is ignored. If the instructions are not being shown, it proceeds in the same way as the holographic prototype: changing the color of the bars to contrast the selected one and updating the text files.

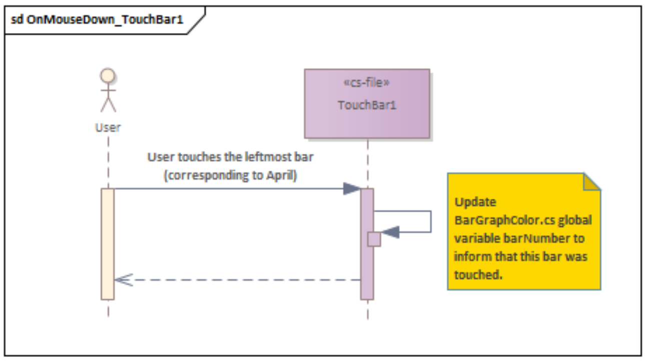

sd OnMouseDown_TouchBar1
User
«cs-file»
TouchBar1
User touches the leftmost bar (corresponding to April)
Update BarGraphColor.cs global variable barNumber to inform that this bar was touched.

Figure 5.49: OnMouseDown (TouchBar1) - sequence diagram

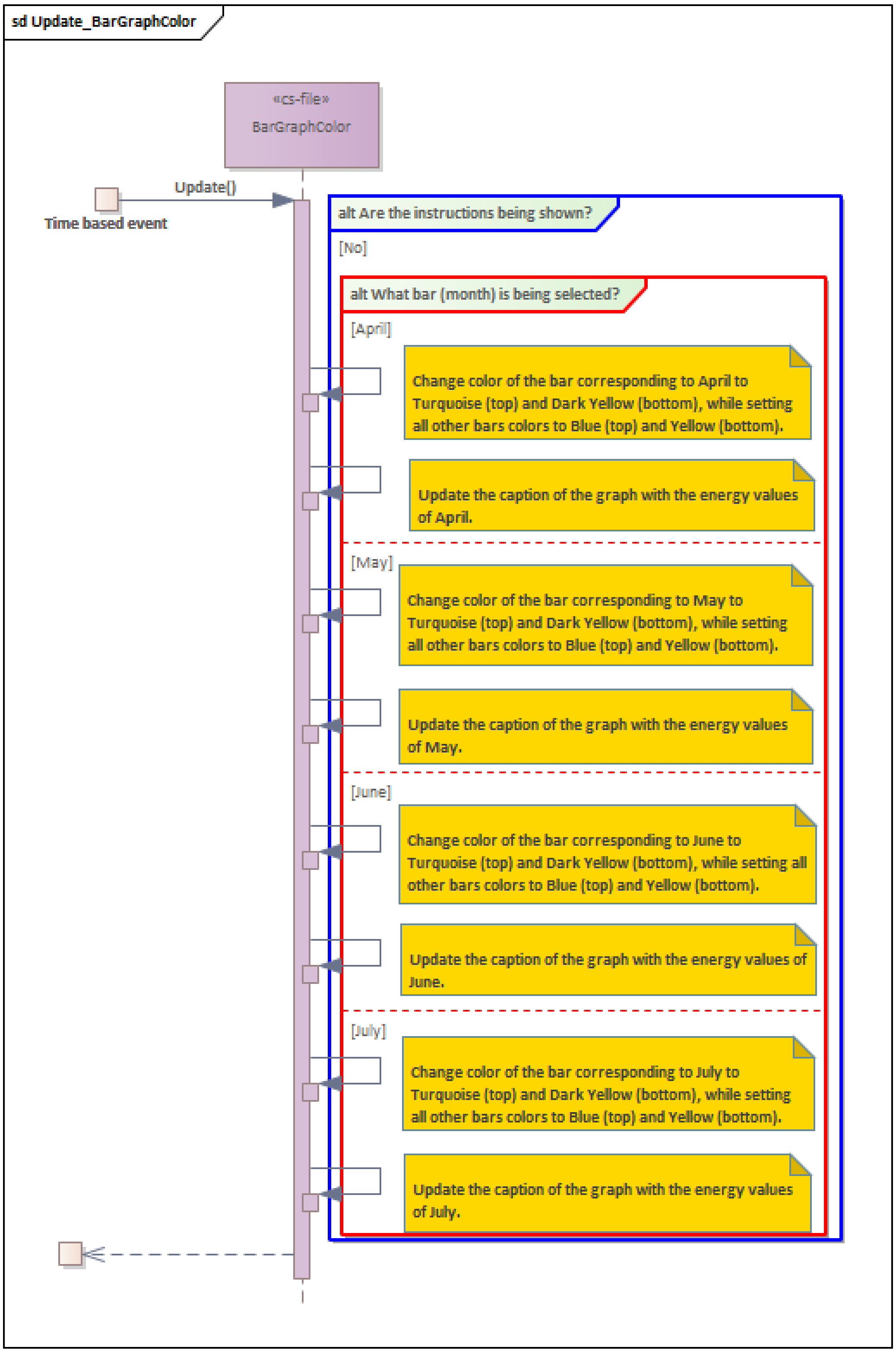

sd Update_BarGraphColor
«cs-file»
BarGraphColor
Update()
Time based event
alt Are the instructions being shown?
[No]
alt What bar (month) is being selected?
[April]
Change color of the bar corresponding to April to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of April.
[May]
Change color of the bar corresponding to May to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of May.
[June]
Change color of the bar corresponding to June to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of June.
[July]
Change color of the bar corresponding to July to Turquoise (top) and Dark Yellow (bottom), while setting all other bars colors to Blue (top) and Yellow (bottom).
Update the caption of the graph with the energy values of July.

Figure 5.50: *Update* (BarGraphColor) - sequence diagram

5.4 Final prototypes

No changes were made to the holographic display prototype hardware since it comes in a frame. The final result is visible in Figure 5.51.

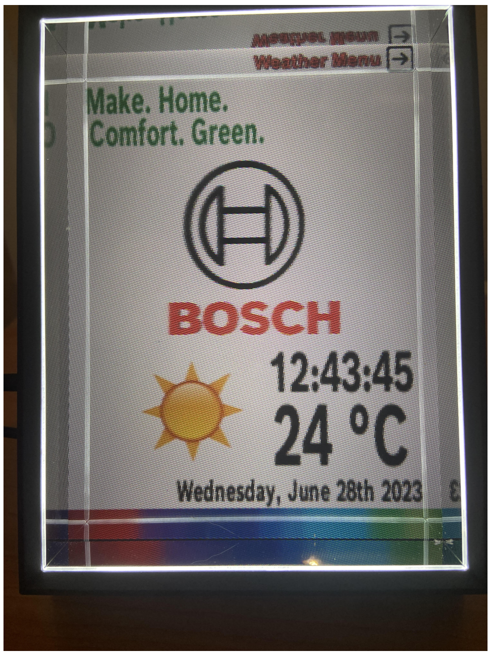


Figure 5.51: Finalized holographic prototype

Regarding the transparent display prototype, a 3D-printed Bosch logo and slogan were designed and integrated into this prototype, as seen in Figure 5.52. One of the main aspects of this display is that the user can see what is inside the box, and if there is nothing in it, then the transparency of the display would not be used.

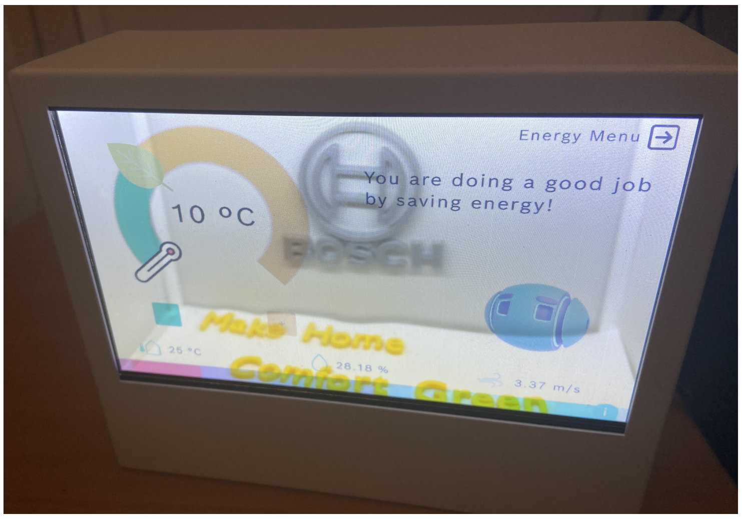


Figure 5.52: Finalized transparent prototype with 3D-printed models

5.5 Alternative computing platform

As referred, both prototypes must be connected to a computer to load the UI into the display. It would be interesting not to depend on a full-fledged computer to make the prototype work, so the next challenge was to run the prototypes on a smaller and easier-to-transport device. Using an SBC like a Raspberry Pi could be a solution to this problem. Unfortunately, the outcome of this experiment was not the expected one since the integration of the prototypes with a Raspberry Pi was not successfully achieved, as will be explained next.

5.5.1 Holographic prototype limitations

The Looking Glass Bridge software that is required to detect the Looking Glass Portrait does not support the ARM64 architecture of the Raspberry Pi. Even though Looking Glass Portrait has an incorporated Raspberry Pi 4, it is only used for Standalone Mode, meaning that it does not support any real-time graphic processing and only does video playback.

Since Looking Glass Bridge is available for x86 operating systems like Linux and Windows, running this Unity project on a smaller device could be possible. Dell Wyse 3040 Thin Client[27] is an alternative to Raspberry Pi that supports x86 operating systems.

5.5.2 Transparent prototype limitations

Even though the transparent prototype does not require additional software to detect the display, the problem relies on Unity. We tested a Raspberry Pi 3 Model B[28] with the Raspbian operating system[29], which is based on the Debian Linux distribution. Even though Unity can build the project for various platforms, including Linux, it does not support the Raspberry Pi ARM64 architecture.

The Unity project can be built using WebGL, enabling the project to be executed from a WebGL-compatible browser with the help of a local host. Both approaches worked, and the UI could be opened. Still, it had significant problems that prevented the prototype from working properly: inputs and outputs were limited (e.g. swipe only worked in one direction and, after some interactions stopped working), UI did not fit the screen size and was constantly freezing, forcing the prototype reset.

5.6 Summary

This chapter discussed all the steps of the development of both prototypes: from the basic concepts and the software platforms used, to the designed architecture, consisting of the specific scripts and methods, created and implemented to support this integration. The differences between

[27] https://www.dell.com/en-us/shop/wyse-endpoints-and-software/wyse-3040-thin-client/spd/wyse-3040-thin-client

[28] https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[29] https://www.raspberrypi.com/software/operating-systems/

both prototypes were highlighted when discussing the scripts and methods. The menus were designed taking into account the UX study conducted in Chapter 2. It also included a section on computing platform improvement that still needs some investigation in order to make proper conclusions.

Chapter 6

UI/UX tests

The UI/UX tests were introduced in Chapter 2, and their purpose is to evaluate the design and usability of a product. To evaluate the prototypes developed in this project, a series of tasks was created for the users to carry out, and each task was timed for later analysis. Following these tests, a questionnaire was created to let the users evaluate their own experience upon interacting with the prototypes and to evaluate the characteristics of the prototypes themselves.

6.1 Tasks proposed

The transparent prototype has more features than the holographic one, so it also has more tasks. These tasks were performed in a closed room with only two persons: the responsible for the test (tester) and the user. The tester must time the tasks and take additional notes, which will be explained in more detail below.

- **Task 1 - Explore**
  The users' first task is to explore the prototype as desired with no time restriction to familiarise the users with the device. When assessing the holographic prototype, the tester must take note if the users managed to switch menus by using the capacitive button with no help. When assessing the transparent prototype, the tester must take note if the users managed to switch menus by pressing the top right corner button or by swiping (it could be both). It also takes note if the users realizes that it is possible to swipe in both directions (left and right) and not only left, as it could be visually misleading by the top right button.

- **Tasks 2 and 3 - Instructions**
  The following tasks only apply to the transparent prototype since the holographic one does not have this feature. To complete tasks 2 and 3, the users must open and read the instructions on the setpoint and energy menus, respectively. The tester must take note if the users found the buttons that open the instructions on both menus without help.

  These tasks aim to prepare the users for the upcoming tasks by reading the instructions.

- **Task 4 - Change setpoint**
  This task applies to both prototypes and requests the users to change the setpoint value four times. The reason it asks for changing the setpoint four times is to increase the probability that the users input various setpoint values below and above the recommended setpoint in order to notice the change in the robot color and the advice. When assessing the holographic prototype, the tester must take note if the users managed to change the setpoint value through the capacitive buttons or not. On the transparent prototype, the tester must take note if the users managed to change the setpoint value through the buttons or the slider (could be both) or if they could not change it at all.

- **Task 5 - Change setpoint unit**
  This task only applies to the transparent prototype for the same reasons mentioned in tasks 2 and 3. This task instructs the users to change the setpoint unit from the default Celsius to Fahrenheit. The tester must take note if the users successfully accomplished the task by continuously pressing the setpoint value for 1.5 seconds to trigger the change of unit.

- **Task 6 - Energy consumption**
  This task applies to both prototypes, and it requests the users to analyze the detailed consumption of all four months presented on the energy menu. When assessing the holographic prototype, the tester must take note if the users pressed the capacitive buttons to loop through all the months and analyze the data. On the transparent prototype, the tester must take note if the users pressed the bars on the touchscreen in order to analyze a specific month in more detail or if they could not analyze the months at all.

- **Task 7 - Setpoint unit back to Celsius**
  This task only applies to the transparent prototype for the same reasons mentioned in tasks 2, 3 and 5. This task is similar to Task 5 since the users must do the same actions to change the setpoint unit back to Celsius.

6.2 Questionnaire proposed

After the tests, the users were given a questionnaire regarding the performed tasks. They must answer each question according to the scale defined in it. Each question has an open space for the users to write further comments to complement their answers if necessary. The questions for both prototypes are similar except for Questions 2 and 4, as shown below.

- **Question 1 - Information**
  The users must classify the information transmitted by the menus of the prototypes on a scale from 1 to 5, with 1 being "Not clear" and 5 "Very clear".

- **Question 2 - Instructions**
  This question only applies to the transparent prototype, as explained in tasks 2 and 3 descriptions above. The users must classify the clarity of the instructions given by the setpoint and energy menus on a scale from 1 to 5, with 1 being "Not clear" and 5 "Very clear".

- **Question 3 - Legibility**
  The users must classify the legibility of the text presented in the prototypes on a scale from 1 to 5, with 1 being "Poor legibility" and 5 "Good legibility".

- **Question 4 - 3D objects**
  On the holographic prototype, the users must classify the impact of the 3D animations (e.g. the spinning Bosch logo and the robot head) on the visibility of the menu on a scale from 1 to 5, with 1 being "Negative effect" and 5 "Positive effect". On the transparent display, the users must classify the impact of the 3D impressions placed inside it (using the same scale).

- **Question 5 - Response time**
  The users must classify the response time of each prototype on a scale from 1 to 5, with 1 being "Very slow" and 5 "Very fast".

- **Question 6 - Discomforts**
  This is the only question that the users do not need to classify according to a scale. A list of discomforts (nausea, headache, eyestrain and dizziness) is given, and they must mark the ones felt during the tests. They can also refer to other discomforts or leave it blank if no discomfort was felt.

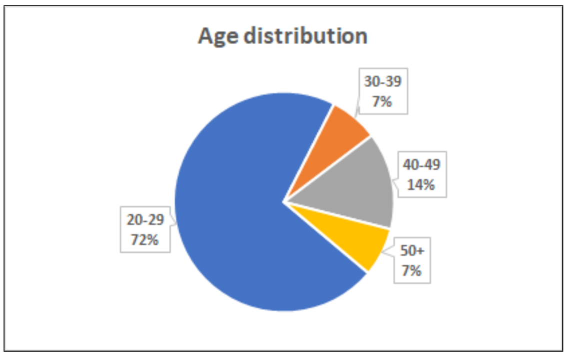
- **Question 7 - Ambient light**
  The users must classify the impact that the ambient light has on the visualisation of the menus of the prototypes on a scale from 1 to 5, with 1 being "Negative impact" and 5 "Positive impact".

- **Question 8 - Aesthetics**
  The users must classify the aesthetics of the presented hardware on a scale from 1 to 5, with 1 being "Unpleasant" and 5 "Pleasant".

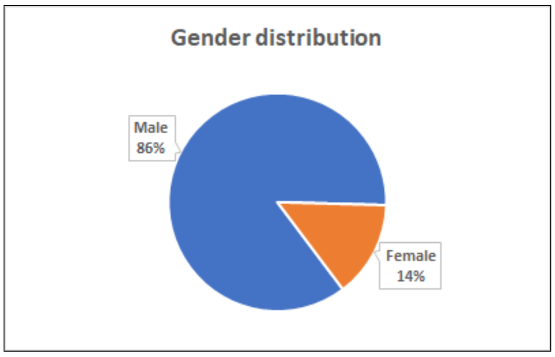
- **Question 9 - Replacement**
  The users must give their opinion about how they would feel about a possible replacement of the current LCD technology on a device like a smart thermostat with a holographic/transparent display on a scale from 1 to 5, with 1 being "Disagree" and 5 "Agree". The users must justify the opinion given.

6.3 Statistics and results

The tests were made in the span of two weeks with a total of fourteen users. The users' age averaged 30 and a half years old, ranging from the early twenties to late fifties, as seen in Figure 6.1 (a). The gender was mainly male-dominated, as seen in Figure 6.1 (b).



(a) Distribution of ages.



(b) Distribution of genders.

Figure 6.1: Users' personal information.

6.3.1 Tasks outcome

Figure 6.2 shows the average task duration for both prototypes. Every user managed to complete all tasks with no help from the tester, although some tasks turned out to be more complicated for the users than it was anticipated. Task 1 turned out to be the most time-consuming one, as expected, but especially tasks 2 and 5 turned out to be surprisingly time-consuming.

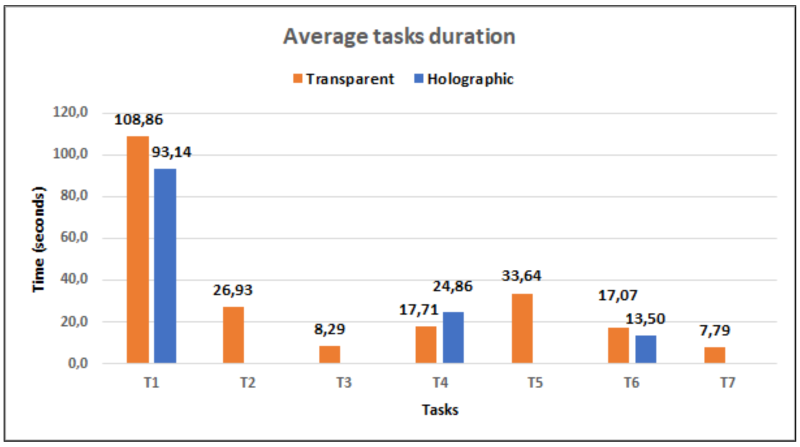
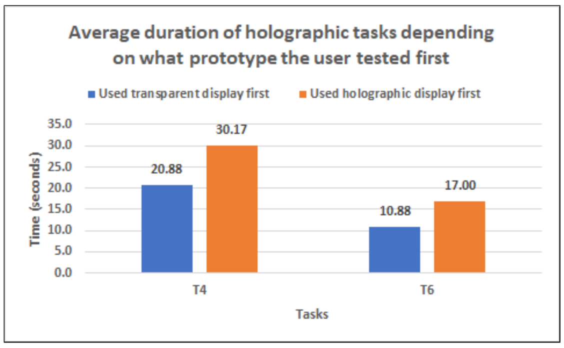
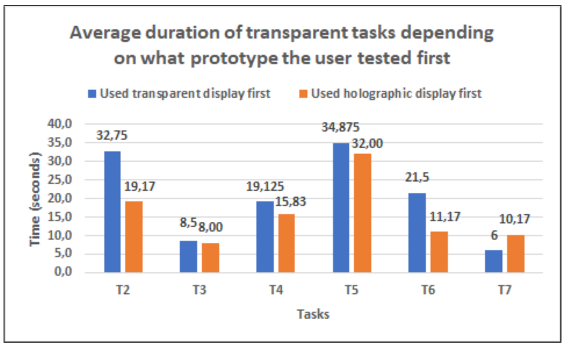


Figure 6.2: Average task duration for both prototypes

Almost half of the users started by testing the holographic prototype and then proceeded to test the transparent one. The remaining users did the opposite. This difference is reflected in Figure 6.3 (a), where it is possible to see that the users that started the tests with the transparent prototype were quicker in finishing the holographic tasks. The same can be said for those who started the tests with the holographic prototype since those users finished the transparent tasks quicker than those who started testing the transparent prototype, as seen in Figure 6.3 (b).



(a) Average duration of holographic tasks.



(b) Average duration of transparent tasks.

Figure 6.3: Average duration of tasks depending on what prototype was tested first.

6.3.1.1 Task 1

In the first task of the transparent prototype, it was possible to differentiate various users: 28.57% switched menus by pressing the top right corner button and nothing more; another 28.57% switched menus by swiping on the touchscreen and did not use the button mentioned before; the remaining 42.86% of the users managed to use both the button and the swipe to switch menus. Furthermore, 10% of the users who used swipe navigation did not notice that it was possible to swipe left and right.

All users that missed to navigate using the digital touch button actually tried to press the button, but did not manage to activate it on the first couple of tries, therefore deducing that it was just another piece of information instead of a button and searching for an alternative. This fact raises concerns about the positioning of the button (top right corner), which seems to interfere with the users' ability to interact with it. This can easily be fixed by lowering the position of the button and expanding its touch area to avoid such situations.

Some users stated that the swipe feature was not clear to them when first interacting with the display. This explains why 28.57% of users only used the buttons.

The holographic display only has one way to switch menus, and every user managed to complete the task. Although, some users shared that the fact that the prototype only allows to loop in one direction can be frustrating. Also, the top right corner button has information displaying what the next menu will be, but there is no such information about the current menu. Some users even tried to touch this top right corner information thinking the display had a touchscreen, although this can be explained by the fact that most of the users who did this tested the transparent prototype first.

6.3.1.2 Tasks 2 and 3

Task 2 was, unexpectedly, one of the most time-consuming tasks on the list. This happens because the button is positioned in the bottom right corner of the display and is not clearly visible at first glance, according to the feedback given by the users. This leads to the users wandering around the menus in search of the instructions during this task. Task 3 is similar but has a smaller average duration since the users replicate the steps used on Task 2.

6.3.1.3 Task 4

The average duration for Task 4 is within the expected time interval, and the users did not have trouble completing it. Some users shared concerns regarding the lack of confirmation upon changing the setpoint value. On the transparent prototype, most of them tried to find a confirmation button and even switched menus as a way to confirm that the changes were applied. On the holographic prototype, the users also tried to find a way to confirm the change by pressing one of the available prototype capacitive buttons.

On the transparent prototype, even though most users interacted with both the slider and the buttons below it during Task 1, only 14.29% of them used the buttons to change the setpoint value; 42.86% used the slider and ignored the buttons; and the remaining 42.86% of users used both the buttons and the slider.

6.3.1.4 Task 5

Similarly to Tasks 2 and 3, Task 5 was supposed to be a quick task but turned out to be the one users struggled with the most. This happened due to three major reasons. The first one is that the users must continuously press the setpoint value for 1.5 seconds to complete the task. However, most users would give up on the continuous press after one or even half a second and start trying other approaches. The second reason is that people did not read the instructions and spent a significant amount of time trying to press random objects in the menu until finally going back to the instructions and reading the procedure. Finally, even after reading the instructions stating that the users should continuously press the setpoint value, they would incorrectly press the handler (thermometer).

6.3.1.5 Task 6

Task 6 is very straightforward, and users had no difficulty looping through the monthly consumption. This is reflected in a reasonably low task duration on both prototypes.

6.3.1.6 Task 7

Similar to Tasks 2 and 3, the duration of Task 7 was significantly reduced due to the fact that all the users had to do was replicate what was made in Task 5 and put the setpoint value back to Celsius.

6.3.2 Questionnaire outcome

The average classification for each answer on both prototypes can be seen in Figure 6.4. Question 6 is not included in this graphic because it is the only question without a scale. Question 2 does not apply to the holographic prototype, therefore the graphic only shows one bar regarding the transparent prototype. The classifications are very similar in both prototypes with the exception of Questions 4 and 8 which have a big gap between the prototypes.

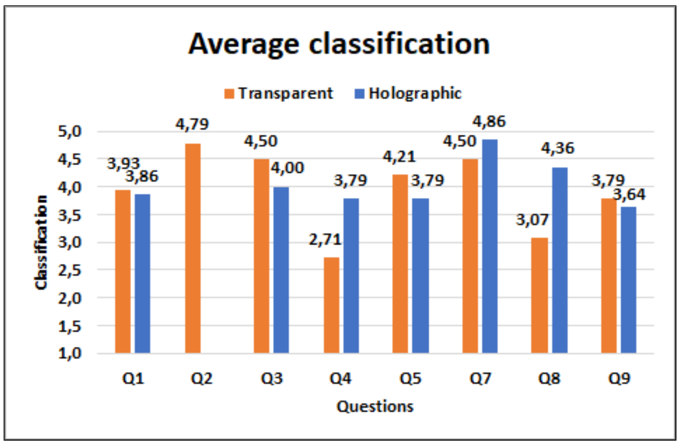


Figure 6.4: Average classification for each question

6.3.2.1 Question 1

As seen before, this question got an average classification of 3.93 and 3.86 for the transparent and holographic prototypes, respectively. Recalling that 1 means "Not clear" and 5 "Very clear", these classifications mean that the users found that the information presented was clear but with room for improvement.

Some justified the given classification with the fact that, on the transparent prototype, the instructions button was too hidden or even camouflaged on the bottom of the display. It was also stated that they would probably not notice the button if it was not for the given tasks. Some said that on both prototypes, the energy menu information was not very clear, especially regarding the relationship between consumed and supplied energy. The fact that the transparent prototype did not inform the users that swiping was an available feature on this prototype also caused some confusion among users. Lastly, some users thought that some menus on both prototypes were too overwhelming, meaning that there was too much information on the screen at the same time.

Even though the transparent prototype received more criticism, it ended up with a higher classification.

6.3.2.2 Question 2

Even though the instructions button was hard to find, as mentioned before, the instructions were very clear to the users, leading to a 4.79 average classification. Some users even added that the fact that the information was divided into bullet points helped to understand the fundamentals of the prototype.

6.3.2.3 Question 3

This question averaged a considerably high classification of 4.5 and 4.0 for the transparent and holographic prototypes, respectively, meaning that the legibility of the text was good but could still be improved. Some users mentioned that bright colors are not clearly visible from certain angles and obstruct the view of the transparent display, especially on the weather menus. A specific user also stated that the panel used to darken the background to highlight the instructions interfered with the reading of those instructions.

The lowest classification of the holographic display compared to the transparent one can be explained by the fact that most users found the display resolution to be very poor, resulting in some pixelated menus with blurry letters and images.

6.3.2.4 Question 4

This question has a considerable gap in the classification of each prototype, as the transparent one got a 2.73 average classification while the holographic one got 3.79. The transparent prototype has the lowest classification because users did not like the 3D impressions inside the display. Some stated that it is irrelevant, and others that it obstructs the view.

The holographic prototype has a higher average classification because, according to the users, the 3D models did not obstruct the view like on the transparent one. Similarly to the previous prototype, some users stated that these 3D models were irrelevant. Still, other users actually liked the models and claimed that they made the prototype more captivating and interactive. The only other downside pointed out by users was the fact that some models had a bad resolution, according to what was previously said in Question 3.

6.3.2.5 Question 5

Even though Question 5 had a relatively high average classification of 4.21 for the transparent prototype, the same can not be said for the holographic one, which got 3.79. According to the users, the response time of the transparent was clearly superior in comparison with the holographic display. Some users justified the classification on the transparent prototype due to the fact that the top right corner button used to switch menus did not work all the time, even though they considered the response time of the rest of the prototype very satisfactory.

On the holographic display, the users considered it a bit too slow to process the button inputs for switching menus and even for other operations like increasing the setpoint value or looping through the month consumption. Some even pointed out that after pressing the button in quick successions, the prototype would ignore those successions.

6.3.2.6 Question 6

The average duration for each test was around 220 seconds and 131 seconds for the whole transparent prototype test and holographic test, respectively. Even though the tests were not intensive, some users claimed to feel some sort of discomfort. No discomforts were pointed out by any user during the transparent prototype test, but 28.57% of users did feel eyestrain during the holographic prototype test, while 7.14% felt dizzy. The remaining 64.28% felt no discomfort, as seen in Figure 6.5.

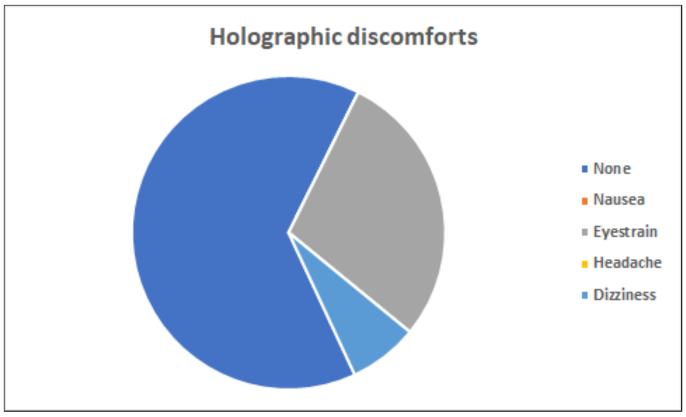


Figure 6.5: Discomforts felt by users during holographic prototype test

6.3.2.7 Question 7

Question 7 has the highest average classifications out of all questions since people classified that the ambient light positively impacted both prototypes. Is it worth mentioning that the test should also be made in a room with a darker ambient light to see if there was any significant difference. However, the tests were made in a room where we had no control over the lights.

6.3.2.8 Question 8

Question 8 is the question with the most significant classification gap between prototypes. Although the users tended to like the functionality of the transparent prototype, they did not enjoy the aesthetics of it. On the other hand, the holographic prototype was slower and did not have so many features, but the users enjoyed its aesthetics, even considering it modern and elegant. People also commented about the fact that too many cables were shown on both prototypes, which had a negative impact on their aesthetics. Some users stated that the dimensions of the prototype were a bit too large for what they were meant to.

6.3.2.9 Question 9

Even though the classification gap is minimal, question 9 shows that users are more willing to accept a transparent display on a device like a smart thermostat than a holographic one. These average classifications are not that high, meaning users were sceptical about adopting these disruptive technologies as a possible replacement for the traditional LCDs. Users justified their answers by stating that it would be a good idea if introducing these displays did not affect the price or the way the device works. Others pointed out that there is no reason for such a complex interface in a daily device and that they prefer a more practical and simple interface. On the other hand, users mentioned that these out-of-the-box technologies made the device modern, attractive, appealing and dynamic while serving the same purpose as a traditional LCD.

Specifically on the transparent prototype, some users stated that the overheating of the device was a major concern. On the holographic prototype, the users claimed that the technology was interesting, but the poor resolution of the prototype was an issue. Also, the view angle for this prototype was too small, according to the users, adding that they could not see the content of the display if they moved their heads too much. The holography was not amusing for some users since they added that there was no real benefit in having it.

6.3.2.10 Users Suggestions

At the end of the questionnaire, an open space was left for the users to suggest changes and talk about topics that were not addressed during the questionnaire. Some of the suggestions were the following:

- **Sleep mode**

  Both displays could have a sleep mode after 10 or 15 minutes. There is no reason for the display to be turned on 24 hours a day, consuming so much power, especially the transparent one with the strong backlight.

- **Menu identification**

  The next menu is being identified on every menu of both prototypes, but the current and previous ones could also be identified.

- **Bosch symbol**

  There is no need for the Bosch symbol to have so much focus on the main menu. Customers would not be interested in having that symbol highlighted in their household. The logo could be placed in the corner of the Main menu and in a smaller size.

- **Menu navigation**

  On the transparent prototype, the navigation could be made through a static sidebar where the users could click on it to open the list of menus and choose which menu to travel to.

- **Default menu**

  Currently, the Main menu is the default menu to which the prototype will go back after a certain amount of time, but it makes more sense for the Weather menu to be the default one since it is the more appealing menu.

- **Instructions button**

  The instructions button could use a gear icon instead of the letter *i* (for information) and be placed in the bottom left corner to be more eye-catching.

- **Setpoint confirmation**

  There should be a way to confirm if the setpoint was changed or not. Some users think that when they change the value, they are in "edit mode" and that there should be a way to confirm the new input. Also, the setpoint value that the users can change is the desired temperature for the room; for that reason, it is vital to inform the users about the current room temperature. For example, if the room is at 16ºC and the users input 18ºC, then the room temperature will not immediately increase by 2ºC, so they should see the two values.

- **Weather menu**

  Some users thought the Weather menu was pointless, and others thought that more information could be shown on this menu, like the weather forecast for the next 2 or 3 days, instead of just the current day.

- **Go back to the previous menu**

  There should be a way to return to the previous menu in the holographic prototype so that the users do not need to loop through all the menus. The transparent prototype could also have a virtual button to go back to the last menu since it has one to go to the next menu.

- **Button position**

  On the holographic prototype, the capacitive button used to switch menus is lower in comparison with the information regarding the next menu, which can be misleading because people looked for a capacitive button on the top right corner of the display.

6.3.3 Conclusions and thoughts

All users classified these prototypes as if they would purchase them for their households. Since these disruptive technologies are far more expensive than the more traditional ones, that would result in a more expensive final product which did not please the users. Apart from the fact that the majority of users enjoyed these technologies, they had no real preference for having them in their households over the more traditional LCDs. Primarily because they provided the same information, just in a different way, which most users did not value that much.

These disruptive technologies would be more suited for high-end product segments, considering that the target customers would give priority to the aesthetics and novelty of the product in favor of its price. For example, a hypothetical futuristic store would greatly benefit from these disruptive technologies. It would fit its futuristic theme and could be placed in plain view of the customers to gain their attention in the same way that some stores use these same technologies on their shop windows for the same purpose.

The displays used for the prototypes were not designed for this use case but were beginner-friendly in all the ways. Interacting with them is fairly simple, and designing the UI was straightforward due to their support for the Unity framework.

6.4 Summary

This chapter presented the UI/UX tests used to assess the developed prototypes using a target group of people. It reviewed all the tasks and questions proposed to the users. It presented and discussed the results through a statistical study necessary for a deeper analysis of both prototypes. The prototypes were well received by all users, but the cost impact was a major concern among them since a traditional LCD is cheaper and transmits the same information, just in a different way.

Chapter 7

Conclusion

This dissertation aimed at the development of innovative UI designs considering disruptive display technologies for Smart Home devices. Their evaluation and UX assessment were done by conducting tests with a diverse group of participants focusing on the effectiveness of information, graphical quality, organization of information, ease of application and cost impact.

After the initial assessments and selection of the software platform to be used to construct the UI, the development of the prototypes were fairly intuitive due to the immense online information, such as templates and tutorials to help get started with those platforms. The transparent display hardware did not pose major concerns to the project. On the other hand, the holographic display hardware had issues being detected by most software platforms tested, restricting the options. Unity is one of a few software platforms that already had this connection configured, which influenced its selection in favor of others platforms without such a connection prepared. Another major downside of the prototypes hardware was that both displays were considerably expensive compared to the traditional LCDs, especially the holographic one, considering that it does not even work as a capacitive touchscreen. The commercial holographic displays available today on the market are not well suited to the Smart Home use cases, given their prices and operation characteristics. The prototypes worked great when connected to a laptop, but deploying the UIs from Unity to be used on an SBC was more challenging than expected. This meant that the prototypes could not be configured to run in an SBC, negatively impacting the application of these technologies.

Through the testing phase, valuable insights were obtained. Participants were encouraged to interact with the prototypes, express their opinions, and provide feedback regarding their usability, aesthetics, and overall impression. The results of the tests indicated that the disruptive technologies were mostly perceived as engaging and interesting by the participants. However, despite their potential benefits, the prototypes did not garner a strong preference over traditional smart thermostats. This was primarily attributed to two key factors: increased cost and the need for further refinement in the UIs of the prototypes.

Given these findings, it is clear that additional research and development efforts are necessary before transparent and holographic display-based smart thermostats can compete with and surpass traditional models in terms of user preference.

7.1 Future work

Future studies should focus on refining the UIs, addressing the usability concerns, and exploring cost-effective manufacturing processes. Some future work could revolve around adding the following features/improvements to the prototypes:

- Adding instructions to the holographic prototype so that it becomes similar to the transparent.
- Adding the option to change between Celsius and Fahrenheit to the holographic prototype for the same reason mentioned above.
- Adding a virtual button on the transparent display to go to the previous menu and adding a way to do the same on the holographic display. This will save the user some time.
- Add a confirmation button upon changing the setpoint value, as well as differentiate the setpoint value into two separate values: desired and current setpoint. This will make the menu more clear for the user.
- Make a deeper study about running the prototypes on an SBC to increase the application of these technologies.
- Increase the accessibility of both prototypes by adding features such as colorblind mode, large fonts for elderly people and an animation stop button for people with certain health conditions that could be triggered by the UIs movements.

Some other user suggestions mentioned at the end of Chapter 6 could also be taken into account.

References

[1] Bosch. Smart home explained. https://www.bosch-smarthome.com/uk/en/smart-home-explained/. Accessed: 2022-12-2.

[2] Davit Marikyan, Savvas Papagiannidis, and Eleftherios Alamanos. A systematic review of the smart home literature: A user perspective. *Technological Forecasting and Social Change*, 138:139–154, 2019.

[3] Chen Peng, Changhao Shi, and Alex Liu. Hmi design using hvac platform. *Freescale Semiconductor*, Application Note 4617 (AN4617).

[4] Alex S Taylor, Laurel Swan, Rachel Eardley, Abigail Sellen, Steve Hodges, and Ken Wood. Augmenting refrigerator magnets: why less is sometimes more. In *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles*, pages 115–124, 2006.

[5] Nuno Filipe Pinto Ferreira. *iPortalDoc Mobile*. Master thesis, University of Porto, July 2017. Available at https://repositorio-aberto.up.pt/handle/10216/106856.

[6] Birte Schiffhauer, Jasmin Bernotat, Friederike Eyssel, Rebecca Bröhl, and Jule Adriaans. Let the user decide! user preferences regarding functions, apps, and interfaces of a smart home and a service robot. In Arvin Agah, John-John Cabibihan, Ayanna M. Howard, Miguel A. Salichs, and Hongsheng He, editors, *Social Robotics*, pages 971–981, Cham, 2016. Springer International Publishing.

[7] Michal Luria, Guy Hoffman, and Oren Zuckerman. Comparing social robot, screen and voice interfaces for smart-home control. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 580–628, 2017.

[8] Samantha Reig, Terrence Fong, Jodi Forlizzi, and Aaron Steinfeld. Theory and design considerations for the user experience of smart environments. *IEEE Transactions on Human-Machine Systems*, 2022.

[9] Igor Ðuric, Dusan Barac, Zorica Bogdanovic, Aleksandra Labus, and Bozidar Radenkovic. Model of an intelligent smart home system based on ambient intelligence and user profiling. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2021.

[10] Theo Franquet, Yumeng Sun, Alexandre Abou Chahine, Ilaria Albanese, Guillaume Ramé, Xingzi Zhu, Jeremy Pitt, Kristel Fobelets, and Mrs Esther Perea. The smart thermostat–group 13. 2016.

[11] Ruth Tamas, William O'Brien, and Mario Santana Quintero. Residential thermostat usability: Comparing manual, programmable, and smart devices. *Building and Environment*, 203:108104, 2021.

[12] Mona Alzahrani, Alexandra L Uitdenbogerd, and Maria Spichkova. Impact of animated objects on autistic and non-autistic users. In *Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society*, pages 102–112, 2022.

[13] Therese Peffer, Marco Pritoni, Alan Meier, Cecilia Aragon, and Daniel Perry. How people use thermostats in homes: A review. *Building and Environment*, 46(12):2529–2541, 2011.

[14] Tiiu Koskela and Kaisa Väänänen-Vainio-Mattila. Evolution towards smart home environments: empirical evaluation of three user interfaces. *Personal and Ubiquitous Computing*, 8(3):234–240, 2004.

[15] Wu Fan and Wang Jiaoqing. Research on interactive genetic color matching design based on user cognitive noise. In *2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, pages 1247–1250. IEEE, 2022.

[16] Andrew Wilshere. From a colourblind designer to the world: please stop using red and green together. https://uxdesign.cc/from-a-colourblind-designer-to-the-world-please-stop-using-red-and-green-together-b311f321832c. Accessed: 2022-12-21.

[17] Austin Erickson, Kangsoo Kim, Alexis Lambert, Gerd Bruder, Michael P Browne, and Gregory F Welch. An extended analysis on the benefits of dark mode user interfaces in optical see-through head-mounted displays. *ACM Transactions on Applied Perception (TAP)*, 18(3):1–22, 2021.

[18] Marc Hassenzahl and Noam Tractinsky. User experience-a research agenda. *Behaviour & information technology*, 25(2):91–97, 2006.

[19] Marc Hassenzahl. User experience and experience design. *The encyclopedia of human-computer interaction*, 2, 2013.

[20] Ji Hyun Yi and Hae Sun Kim. User experience research, experience design, and evaluation methods for museum mixed reality experience. *Journal on Computing and Cultural Heritage (JOCCH)*, 14(4):1–28, 2021.

[21] Andreas Sonderegger, Andreas Uebelbacher, and Jürgen Sauer. The ux construct–does the usage context influence the outcome of user experience evaluations? In *IFIP Conference on Human-Computer Interaction*, pages 140–157. Springer, 2019.

[22] Zackarias Alenljung and Jessica Lindblom. User experience in augmented reality: A holistic evaluation of a prototype for assembly instructions. In *International Conference on Human-Computer Interaction*, pages 139–157. Springer, 2021.

[23] Laura Bourland. The history of thermostats and their evolution to smart. https://getmysa.com/blogs/home-automation/the-history-of-thermostats-and-their-evolution-to-smart. Accessed: 2022-12-23.

[24] Jung-Young Son, Vadim V Smirnov, Joohwan Chun, Vladimir I Bobrinev, Victor G Komar, and You Seek Chun. Holographic screens and their applications. In *Sixth International Symposium on Display Holography*, volume 3358, pages 337–346. SPIE, 1998.

[25] Lidan He, Kexuan Liu, Zehao He, and Liangcai Cao. Three-dimensional holographic communication system for the metaverse. *Optics Communications*, 526:128894, 2023.

[26] Jin Ryong Kim, Stephanie Chan, Xiangchao Huang, Kenneth Ng, Limin Paul Fu, and Chen Zhao. Demonstration of refinity: an interactive holographic signage for new retail shopping experience. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–4, 2019.

[27] Vadim V Smirnov, Jung-Young Son, Joo-Hwan Chunb, Hyuk-Soo Lee, and Jieun Bahn. Holographic automotive display with full-color properties. In *Practical Holography XIV and Holographic Materials VI*, volume 3956, pages 199–204. SPIE, 2000.

[28] Jana Skirnewskaja and Timothy D Wilkinson. Automotive holographic head-up displays. *Advanced materials*, 34(19):2110463, 2022.

[29] Maryia Kazhura. Exploring new depths: How could passengers interact with future in-car holographic 3d displays? In *International Conference on Human-Computer Interaction*, pages 35–61. Springer, 2022.

[30] Haikuo Zhou. The development, special traits and potential of holographic display technology. 2015.

[31] Jisun Jang and Tomasz Bednarz. Holosensor for smart home, health, entertainment. In *ACM SIGGRAPH 2018 Appy Hour*, pages 1–2. 2018.

[32] Stephen Siemonsma and Tyler Bell. Holokinect: Holographic 3d video conferencing. *Sensors*, 22(21):8118, 2022.

[33] Matthew Hamilton, Thomas Butyn, and Russ Baker. Holographic displays: Emerging technologies and use cases in defence applications. In *NATO MSG-159 2018 Annual M and S Conference*, 2018.

[34] Shing Chow Chan. *Light Field*, pages 748–755. Springer International Publishing, Cham, 2021.

[35] Katsushi Ikeuchi, editor. *ICP*, pages 599–599. Springer International Publishing, Cham, 2021.

[36] Yiyang Ye, Zhen Liu, Tupei Chen, et al. Toward transparent projection display: recent progress in frequency-selective scattering of rgb light based on metallic nanoparticle's localized surface plasmon resonance. *Opto-Electronic Advances*, 2(12):12190020, 2019.

[37] WANG Haihong, JIAO Feng, and MA Qungang. Research and development of high transmittance transparent liquid crystal display. *Chinese Journal of Liquid Crystals and Displays*, 6:901–905, 2014.

[38] Leena Ventä-Olkkonen, Jonna Häkkilä, and Kaisa Väänänen-Vainio-Mattila. Exploring the augmented home window: user perceptions of the concept. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia*, pages 190–198, 2014.

[39] Siddhant Chouksey and Sumedha Sirsikar. A prototype of low cost heads up display for automobiles navigation system. In *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, pages 205–210. IEEE, 2016.

[40] Chun-Wei Su, Chia-Cheng Liao, and Mei-Yung Chen. Color transparent display using polymer-dispersed liquid crystal. *Journal of Display Technology*, 12(1):31–34, 2015.

[41] Srećko Kunić and Zoran Šego. Oled technology and displays. In *Proceedings ELMAR-2012*, pages 31–35. IEEE, 2012.

[42] Looking Glass Factory. Key concepts overview. https://docs.lookingglassfactory.com/keyconcepts/how-it-works. Accessed: 2022-06-12.

[43] Looking Glass Factory. Key concepts: quilts. https://docs.lookingglassfactory.com/keyconcepts/quilts. Accessed: 2022-06-12.