Multi-casting: A Mirroring Display Application for viewing Multiple Oculus Quest Headsets

---

A Capstone
Presented to
the Graduate School of
The University of North Carolina Wilmington

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in Computer Science and Information Systems
Computer Science

---

by
Austin Whittaker
December 2023

Proposed to:
Dr. Toni Pence, Committee Chair
Dr. Elham Ebrahimi
Dr. Kevin Matthews

Abstract

This paper presents a novel application designed for monitoring and managing virtual reality (VR) headsets in educational settings. The application addresses the challenges of efficiently managing multiple VR headsets, streamlining processes such as device naming, monitoring, and user management. Utilizing technologies like Python, Tkinter, CustomTkinter Scrcpy, and Android Debug Bridge, the project successfully integrates a user-friendly desktop application that introduces the ability to enhance the educational learning experience through the use of VR within classrooms. The application allows instructors to monitor and manage multiple headsets simultaneously, offering features like batch renaming and real-time viewing of student activity. Testing in simulated educational environments shows promise of the application's effectiveness, highlighting its potential to transform VR usage in educational settings.

Executive Summary

During my time at the University of North Carolina Wilmington, I honed my skills in various computer science disciplines, including virtual reality (VR) and computer science, culminating in a unique collaboration with UNCW's mixed reality team. We focused on developing a VR application for elementary education, specifically a Sea Turtle Simulation project aimed at promoting a deeper understanding of sea turtles and their habitats. However, a significant challenge persisted: the need for instructors to wirelessly monitor multiple VR headsets simultaneously. This unresolved issue became the cornerstone of my capstone project.

Through witnessing first hand in regards to this problem, I sought to address the lack of a comprehensive solution for simultaneously monitoring multiple VR headsets within a single application. Existing strategies, such as Oculus Casting, only facilitated the viewing of one headset's display at a time, and prior approaches didn't support concurrent, wireless monitoring of multiple headsets. My capstone project aimed to fill this gap by developing a solution that met the needs of a future-ready educational environment.

I delivered a GUI application designed for integration into educational settings. The solution involved creating a Python GUI desktop application that allow instructors to connect multiple VR headsets to a computer using USB-C or USB ports. This setup enabled the display of multiple headsets on the instructor's computer screen. The application allowed for establishing a wireless connection via TCP with the headsets though WiFi. Once connected, instructors could distribute the headsets to students, facilitating an immersive VR-enhanced lesson. This innovative approach aimed to empower educators to deliver engaging and immersive learning experiences, providing real-time assistance and guidance to students within a virtual reality-enhanced learning environment

Table of Contents

List of Figures

Chapter 1

Introduction

Throughout my academic journey at the University of North Carolina Wilmington (UNCW), I have diligently focused on honing my skills in computer science, with particular emphasis on object-oriented programming, database management, data structures, virtual reality, and machine learning. My comprehensive understanding of software engineering principles, design patterns, and game development has equipped me with a solid foundation to propel my career forward in this dynamic field. In the last eighteen months, I have had the privilege of collaborating with UNCW's mixed reality team, comprised of highly skilled and motivated individuals, on the innovative Virtual Access to STEM Careers Sea Turtle Simulation project. This unique opportunity allowed me to gain valuable insights into the rigorous testing processes and real-world applications of mixed reality technologies, through our partnerships with educational institutions like DC Virgo and invitations to the Karen Beasley Sea Turtle Rescue and Rehabilitation Center. Our collective efforts have led to the development of a virtual reality application designed to simulate educational experiences for elementary school students, fostering a deeper understanding of sea turtles and the importance of protecting their habitats. However, throughout our demonstrations, a recurring challenge has emerged, which remains unresolved. The need to enable instructors or teachers to monitor the progress of multiple students using virtual reality headsets simultaneously in a wireless way. The primary objective of my capstone project is to address this pressing issue: **How to develop a desktop application that enables teachers to monitor all headset displays within the classroom and provide real-time support to their students utilizing virtual reality-based learning.** By developing this innovative solution, I aim to enhance the effectiveness of

virtual reality-based educational applications within the classroom and promote a more engaging, responsive, and immersive learning experience for teachers to use for their students. The experience requirements for this project have been outlined and guided by past experiences that I have worked with over the course of my academic journey.

1.1 Virtual Reality Development

During the spring of 2022, I embarked on an intensive learning experience to master the fundamentals of Unity, a powerful platform for creating virtual environments, and honed my skills in C-Sharp programming. This comprehensive training proved valuable when I took on the role of lead developer and project manager for a team of three, working on an immersive virtual reality application known as "The Diamond of Florentine."

Set in the captivating backdrop of the 17th century, "The Diamond of Florentine" invites players to step into the life of a pirate whose ship has been tragically sunk in battle. The player's mission is to navigate a mysterious island teeming with enigmatic riddles, offering assistance to local inhabitants in order to unearth vital clues about the location of the sunken vessel. As players successfully unravel these puzzles and complete various tasks, they must eventually procure a boat to embark on a thrilling quest to locate and retrieve the legendary diamond before it is lost to the ocean's depths for eternity. Our team was committed to crafting an engaging, interactive experience that would closely emulate a genuine, real-world adventure. Utilizing the robust capabilities of the Unity game engine, we endeavored to develop the most visually stunning and high-definition experience possible within the limitations of contemporary graphics processing technology.

Among the many immersive features of this application, the realistic swimming activity emerged as a standout favorite among participants who had the opportunity to demo the game. Positioned at the heart of the ocean, at specific coordinates obtained from the island, players are able to leap into the water and swim to the seafloor in search of further clues leading to the coveted diamond. In order to create a convincing swimming simulation in the virtual environment, our team carefully studied real-world swimming techniques and prioritized user-friendliness to facilitate seamless immersion for players. To accomplish this, we harnessed the advanced capabilities of the Oculus Quest 2 controllers, capturing the precise tracking and velocity of the player's hand movements with due consideration given to the speed of each motion. Utilizing a straightforward

grab-and-release mechanism, users can fully immerse themselves in the virtual world, enjoying a remarkably authentic simulation of swimming along the ocean floor. This attention to detail and commitment to user experience contributed to the overall success and appeal of "The Diamond of Florentine."

This project ignited my passion for virtual reality and its potential applications, ultimately paving the way for my involvement with UNCW's Mixed Reality Lab team. As a member of this esteemed group, I was granted the opportunity to contribute to the cutting-edge Virtual Access to STEM Careers project. Building on the skills and insights acquired from my work on "The Diamond of Florentine," I became eager to further explore the possibilities of virtual reality in the realm of education and beyond.

1.2 Virtual Access To STEM Careers

During the summer of 2022, I was employed to work with the Virtual Access to STEM Careers (VASC) Sea Turtle Simulation Project. This project is a National Science Foundation-funded initiative developed at the University of North Carolina at Wilmington, aimed at supporting STEM education for under-served elementary school students. Through VASC, students acquire core and interdisciplinary skill-sets and strategies aligned to standards, practice relevant skill-sets and strategies to a level of proficiency, and engage in collaborative, interdisciplinary, problem-based activities known as Virtual Grand Challenges that simulate authentic tasks associated with related STEM occupations. Leveraging the benefits of interactive learning experiences, the project employs the Unity cross-platform game engine and development tool to create an immersive, engaging virtual reality (VR) game. The VASC Sea Turtle Simulation project provides students with a unique, hands-on learning experience to foster engagement and knowledge retention, as they explore the world of sea turtles, their habitat, and conservation efforts.

As part of the science curriculum, students typically begin their learning journey with hands-on experiences in a classroom setting, accompanied by instructional materials such as booklets. Our virtual reality project seeks to replicate and enhance this learning experience by immersing students in realistic environments where they can actively engage with tools and practices. These virtual settings can range from classrooms to beach landscapes, providing a dynamic and interactive learning experience.

However, during the development of this project, we have encountered several challenges that must be addressed to ensure its successful implementation within the educational curriculum. One such difficulty is the inability for instructors to view what students are seeing in their virtual reality headsets, which can hinder the provision of timely assistance when students require help with an activity. Furthermore, maintaining student engagement and focus within the virtual environment can be challenging, as the stimulating nature of the experience may inadvertently encourage distraction from the intended learning objectives.

While Oculus provides tools that enable the streaming of headset content, their strict terms and conditions dictate that only one Facebook account can be linked to a device, and this account must be used for streaming purposes. This limitation raises the question of how an instructor can simultaneously monitor the progress of multiple students using virtual reality headsets in a wireless fashion.

Based on our experiences in demonstrating the project to students, I have gained valuable insights into the potential obstacles that may arise when integrating this virtual reality project into a curriculum. The primary goal of this capstone project is to develop a solution that addresses these issues, ensuring that teachers can effectively monitor students' progress within the virtual environment, provide assistance as needed, and maintain engagement with the learning objectives. By tackling these challenges, I aim to optimize the virtual reality experience for teacher/instructors, ultimately enhancing educational outcomes.

1.3 Objective

The VASC project served as a catalyst for my motivation to create an application that enhances teacher monitoring towards the students that are interacting within the virtual reality simulation, in order to monitor progress and provide assistance when assistance is needed. Having witnessed first-hand the challenges that arise during the implementation of such a project, I was able to identify key components that can help ensure a smooth and effective experience for all participants.

In the following sections, I will outline the crucial aspects of this capstone project, including the problem statement, the proposed solution, and the envisioned final product. By addressing these components systematically, I aim to develop a comprehensive and well-rounded approach to

improving the overall effectiveness and usability of the VASC simulation for teacher end users wishing to implement virtual reality lesson plans within their curriculum.

The Problem: Although there are several existing strategies that facilitate the viewing of content from a single virtual reality headset, a comprehensive solution that addresses the ability to simultaneously monitor multiple headsets within a single application remains elusive. As previously mentioned, Oculus Casting offers a means to view a user's display; however, its functionality is constrained to one headset at a time. While other approaches such as Seth Angell's previous implementation of "Web Based Session Monitoring" [Angell, 2022] discussed in this paper, laid the groundwork for such a solution, it ultimately precluded the ability to view content from multiple headsets concurrently in a wireless fashion. In the context of this capstone project, my aim is to overcome these limitations discussed within this paper, and develop a final optimal solution application that caters to the requirements of a future-ready finished final product based off of open ended interviews, and testing with potential end users.

The Proposal: The proposed solution is a sophisticated, user-friendly application meticulously designed for seamless integration into educational environments, complete with all essential software needed to meet the needs of potential end users such as teachers based off of open ended interviews and questionnaires. This innovative application empowers instructors to effortlessly connect multiple virtual reality headsets to a computer using USB-C or USB ports. By executing a Python GUI desktop application, python scripting can be used to utilize command-line functionality binded to button clink inputs. By navigating the app through the click of the start button command, the content displays from 15 to 30 headsets will be displayed on the instructor's computer screen. Subsequently, instructors will establish a wireless connection with the headsets via WIFI. Once the connection is secure, the instructor can disconnect the headsets from the computer, originally tethered via USB or USBC, and distribute them to students, irrespective of the Oculus account associated with each device. With the headsets in hand, students can embark on an immersive, virtual reality-enhanced lesson. This process can enable educators to concentrate on delivering engaging and immersive learning experiences, and assistance for their students, thus allowing educational content to be guided by instructors within the virtual reality-enhanced learning environment.

The Final Product: The finished solution is an intuitive application that conceals the complexity of its underlying programs, ensuring ease of use for teacher to monitor student progression. The application will meet the needs of instructors through the use of questionnaires, testing

and focus groups where the application will apply. The application will leverage the Screen Copy (SCRCPY) library to display the video feed from the Oculus Quest 2 device, while utilizing Python to execute command-line scripts for presenting the headsets' views. Additionally, the application will incorporate the ability to label the screens being viewed, enabling instructors to efficiently track and monitor individual students. To further enhance the instructor's mobility and facilitate their ability to assist students as needed, the application will also integrate Duet Display technology. This technology will allow a teacher or instructor to cast a second monitor via a tablet to drag the application to in order to assist a student providing mobility. By incorporating these features, I plan to test this final product in an educational setting to ensure the application provides a seamless and efficient user experience, allowing educators to focus on delivering a high-quality, immersive learning experiences within a virtual reality-enhanced environment to their students.

1.4 Organization Of The Paper

The organization of this paper is as follows: We started by presenting the introductory material and discussing the motivation behind this project. Subsequently, we will delve into background information on the virtual reality development landscape, the underlying technology, and some existing solutions to the problem at hand. Following this, we explore the proposal for the project submitted in May 2023, along with my predictions regarding the development process. Next, we provide a development section that details my experiences during the project, as well as the accompanying documentation and diagrams of the completed product. In conclusion, we revisit the predictions made earlier in the project and offer insights for future developers who may build upon or utilize this work in their own endeavors.

Chapter 2

Background

Multicasting, in the context of Oculus Quest headsets, refers to the simultaneous streaming and sharing of content across multiple devices in a coordinated manner. This process enables instructors, teachers, or group facilitators to efficiently manage and monitor the activities and progress of users in a virtual reality (VR) environment, ensuring a consistent and synchronized learning or collaborative experience. Implementing multicasting for Oculus Quest headsets involves establishing a robust network infrastructure that can handle the real-time transmission of high-quality VR content to multiple devices. The underlying technology leverages a combination of hardware and software components, including wireless communication protocols, data compression algorithms, and optimized rendering techniques, to ensure minimal latency and high fidelity for all participants. In a professional or educational setting, multicasting Oculus Quest headsets can significantly enhance the efficacy of VR applications by allowing instructors to provide real-time guidance, feedback, and support to users as they navigate the virtual environment.

The primary objective of this project is to develop a user-friendly application that enables multicasting multiple Oculus Quest headsets within a controlled environment. When considering this, there are a few important sections to cover. In this section, we will examine the foundational aspects of virtual reality, including its definition, historical development, technological advancements, and ongoing efforts to enhance its overall performance and adoption in society. We will then address the importance of multicasting in the context of virtual reality applications. Lastly, we will explore relevant work that has been proposed or completed in the past, present, or is anticipated for the future, delving into the ways in which these projects contribute to the broader field of this project.

2.1 Virtual Reality

The historical development of virtual reality can be traced back to the 1960s with the advent of early head-mounted displays, such as the Sensorama by Morton Heilig figure 2.1 [Heilig, 1962] and the Sword of Damocles by Ivan Sutherland figure 2.2 [Sutherland, 1968]. These pioneering devices laid the groundwork for the immersive experiences that define VR today. Over the years, VR technology has experienced significant advancements, as described in Rheingold's book [Rheingold, 1991], propelled by innovations in computer graphics, display technology, and sensor systems.

Burdea and Coiffet (2003) provide an overview of various applications of VR, including gaming, entertainment, education, training, and healthcare [Burdea and Coiffet, 2003]. As VR technology has evolved, there has been a focus on enhancing its performance, usability, and adoption across various sectors [LaValle, 2016]. Some key innovations include miniaturization and increased processing power of VR hardware, improvements in motion tracking and input systems, and the implementation of advanced rendering techniques. Jerald emphasizes the importance of human-centered design for virtual reality, ensuring that the technology is comfortable and intuitive for users [Jerald, 2016]. One of the major challenges in the VR experience has been addressing issues such as latency, motion sickness, and user interaction. LaViola discusses the problem of cybersickness in virtual environments and the need for solutions to mitigate its impact [LaViola, 2000].

In recent years, researchers and developers have explored novel approaches to tackle these issues. Fernandes and Feiner present a technique called subtle dynamic field-of-view modification, which helps combat VR sickness [Fernandes and Feiner, 2016]. Other approaches include optimizing hardware and software components to reduce latency and integrating artificial intelligence, machine learning, and haptic feedback systems to enhance user interaction. As virtual reality technology continues to evolve, it is poised to play an increasingly prominent role in shaping the future of human-computer interaction and transforming how we interact with digital content and each other in our rapidly evolving digital society.

2.2 Importance of Multicasting

As virtual reality (VR) technology continues to advance and find its way into various sectors, from gaming and entertainment to education and training, the need for efficient and effective communication and data distribution becomes paramount [Bouras et al., 2014]. One critical aspect

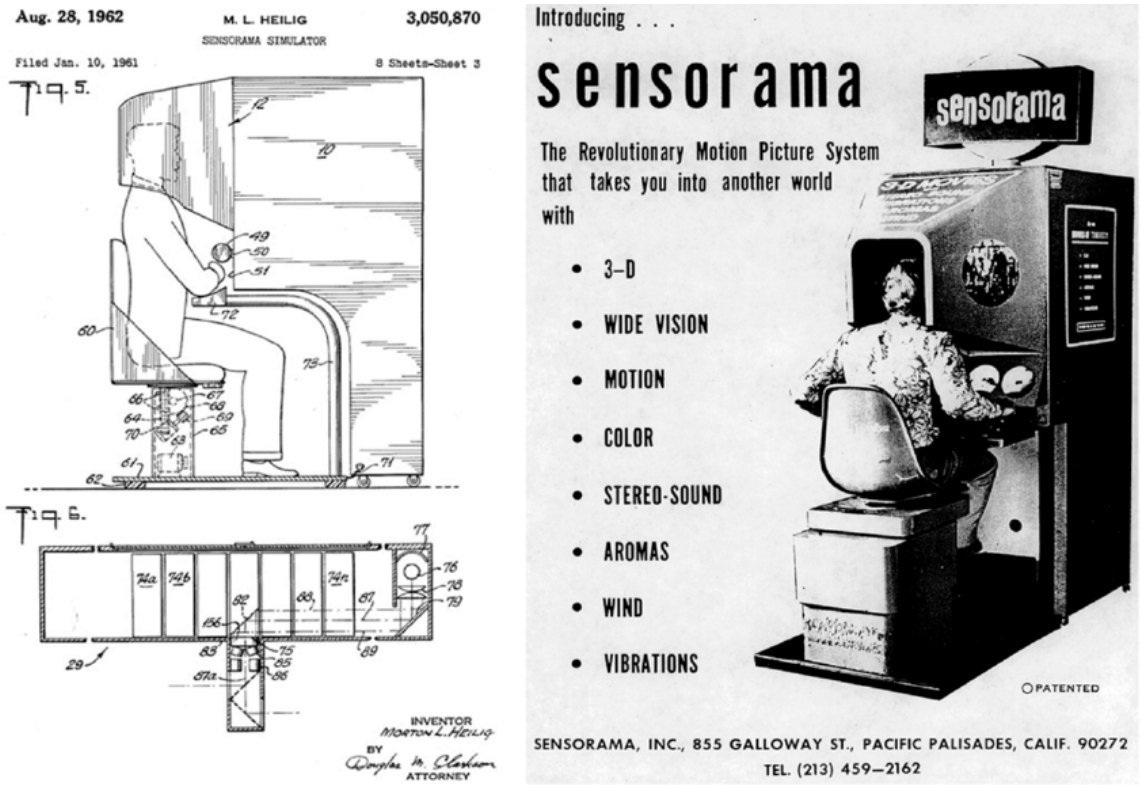

Figure 2.1: Sensoroma, the first virtual immersion system.

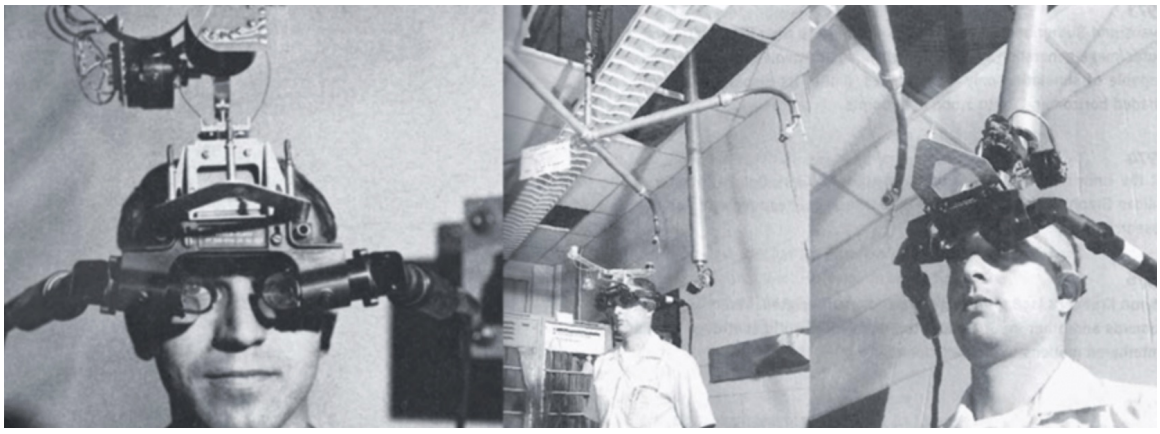

Figure 2.2: Sword of Damocles, the first virtual reality headset.

of this is multicasting, a technique that allows a single source to simultaneously receive data from multiple devices [Venkatraman et al., 2014]. In the context of VR applications, multicasting offers several significant advantages that enhance user experience.

Multicasting enables the simultaneous sharing of individual virtual environments, allowing multiple users to engage in collaborative or individual experiences within the same digital space [Singh and Prabhakaran, 2017]. This fosters a sense of presence and social connection among participants, which is essential in applications such as virtual classrooms, remote training sessions, and multiplayer gaming [Schroeder, 2011]. By providing real-time interaction among multiple users or individual activities by allowing a single application to view multiple users, multicasting can create more engaging and immersive experiences that can be tailored to the participants' needs [Greenhalgh, 2001]. By directly relaying a participant's feed to the real world via instructor or administrator provides a sense of understanding to questions or an idea of a participant's plans or thought process. This ultimately can drive user satisfaction and adoption of VR technology. Multicasting can facilitate flexibility in this ideology as well with the ability to maneuver to different participants within the same space [Venkatraman et al., 2014]. This ensures a consistent and high-quality experience for all participants, regardless of the number of users involved [Bouras et al., 2014]. Multicasting eventually can simplify content management by centralizing the distribution process [Venkatraman et al., 2014]. This enables administrators or instructors to update or run virtual environments more easily and ensures that all users receive the same content simultaneously. This centralized approach can also simplify troubleshooting and maintenance, as any issues can be addressed at the source level, streamlining the overall management of VR applications [Singh and Prabhakaran, 2017].

Multicasting plays a crucial role in the context of virtual reality applications by enhancing teacher experiences, improving participant performance, and enabling large-scale deployments within the classroom. By addressing the challenges of data distribution and resource optimization, multicasting contributes to the continued growth and adoption of VR technology across various industries and sectors [Schroeder, 2011].

2.3 Existing Solutions

Undertaking a comprehensive evaluation of existing solutions in the market or libraries is a critical first step when embarking on this project. It is important to determine whether there are any similar solutions already available, as this would enable myself to avoid redundancy and focus efforts on novel solutions. While several existing solutions may provide some level of overlap with the project's objectives, none of them meet all the requirements necessary to satisfy the project's specific needs. To this end, a sample of solutions that have been considered is provided below.

2.3.1 Oculus Casting

The native screen-casting feature within Oculus OS is an immediate and previously utilized solution used by the VASC project as well as Oculus consumers [Quest, 2023]. As a first-party streaming solution, it enables users to stream their current screen to the Oculus app via mobile phone or a compatible web browser. This feature leverages the streaming role of the popular Google Cast protocol, commonly known as Chromecast, to achieve this functionality. While this solution offers the convenience of simple setup and ease-of-use, there are some limitations to consider.

The utilization of an established protocol simplifies the steps required to set up a device to receive the stream, making it effortless for users. Additionally, a user can stream the headset directly to any HDMI enabled display using a Chromecast HDMI receiver, which costs around $35. However, the screen-casting option in Oculus OS operates at a device level rather than an in-game feature. Therefore, any system built on this technology must utilize an Oculus Quest or other Google Cast compliant device [Quest, 2023]. Moreover, this solution presents a barrier for my improvement, such as a multi-stream environment where multiple users stream to a single display [Malik et al., 2018].

2.3.2 ClassVR

The ClassVR system comprises several components, including a set of VR headsets and a tablet-based control system. The system allows teachers to create and manage content, control student experiences, and assess student progress [ClassVR, 2023]. The VR headsets provide students with an immersive learning experience, enabling them to interact with a range of simulations, models, and environments [ClassVR, 2023]. ClassVR's extensive range of curriculum-aligned content covers various subjects, including science, history, geography, and language [ClassVR, 2023]. The content

is designed to enhance students' understanding of complex concepts and engage them in interactive learning experiences. Teachers can easily select and deliver content to students based on their individual needs, preferences, and learning styles [ClassVR, 2023].

In a study conducted by Mansor, Mokhtar and Sharef [Mansor et al., 2022], the educational potential of the ClassVR system was explored. The research involved primary school students who participated in a series of lessons using the ClassVR platform. The study found that the ClassVR system positively impacted the students' engagement, motivation, and learning outcomes. Another study by Kluge, Maltby, Kuhne, Evans and Walker [Kluge et al., 2023] investigated the adoption of virtual reality technology, including the ClassVR platform, in the elementary school education context. The authors conducted a case study analysis of various schools using VR in their classrooms and reported positive results in terms of student engagement and learning outcomes.

The platform also includes a range of educational tools and features, such as quizzes, interactive whiteboards, and customizable lesson plans. These tools enable teachers to design engaging and interactive lessons that promote critical thinking, collaboration, and problem-solving skills [ClassVR, 2023]. Overall, ClassVR is a comprehensive and user-friendly platform that enhances the traditional classroom experience by leveraging the power of VR and AR technology. It provides an immersive, interactive, and personalized learning experience that motivates students and improves learning outcomes [ClassVR, 2023].

It is worth noting that ClassVR has developed both the headsets and the accompanying technology, making it an integral aspect of research. Notably, this aspect is crucial as it does not involve the utilization of Oculus Quest headsets. Furthermore, while the platform does offer multicasting functionality, it is not directly relevant to the specific objectives and goals of the project. The reasoning behind this limitation is the price points behind ClassVR. ClassVR is a business, and the service they provide comes in the form of a monthly subscription plan paid by the school implementing this technology [ClassVR, 2023]. In order to obtain the price point offered, one can contact the sales department, which will offer a quote depending on the size of the class.

The Oculus Quest 2, on the other hand, has the highest estimated number of units sold of all time, making it an extremely popular and accessible option for both consumers and institutions [VRSpace, 2021]. The affordability of the Oculus Quest 2 outweighs the cost of paying the ClassVR subscription over time, as it provides a cost-effective solution for integrating VR technology into various settings.

Moreover, the Oculus platform allows for greater freedom in development, enabling third-parties to create and tailor lesson plans specifically for curriculum, such as the Virtual Access to STEM Careers project. By choosing to work with the Oculus Quest 2, educators and developers can benefit from the vast user base, affordability, other than a subscription-based service like ClassVR.

2.3.3 Tethering and Desktop Streaming

Tethering and desktop streaming in VR can be useful for a variety of purposes, such as improving performance and capturing the VR experience for remote viewers [Wikipedia contributors, 2023b]. Tethering refers to the process of connecting a VR headset to a computer and utilizing it as a monitor. This allows the computer to take on some of the computational load, which can be particularly helpful when streaming VR content [Wikipedia contributors, 2023b]. Desktop streaming, on the other hand, involves capturing the screen output from the computer and transmitting it to a remote client, typically over the internet.

While these methods can be effective for improving performance and sharing the VR experience with others, they can be problematic when it comes to multicasting wirelessly. Wireless multicasting can be challenging to achieve when using tethering and desktop streaming [Wikipedia contributors, 2023a]. This is because the data being transmitted wirelessly must be processed by the computer, which can add latency and reduce overall performance [Wikipedia contributors, 2023a]. While tethering and desktop streaming in VR can be useful for improving performance and sharing the VR experience with others, it can be problematic for multicasting wirelessly especially in a classroom setting. It is important for the user to be active and engaged within a simulation without having to worry about wires that a user could accidentally unplug or trip over [Wikipedia contributors, 2023c].

2.3.4 3rd Party Solutions

Third-party solutions can significantly improve the user experience of multicasting on Oculus Quest 2 devices, particularly in professional settings where multiple users need to collaborate or present their work simultaneously. This section outlines the most prominent third-party applications and tools that facilitate multicasting by enabling multiple Oculus Quest 2 devices to stream their display to a single computer.

To identify these solutions, a thorough search and review of online forums, VR technology

blogs, and expert opinions through scholarly publications in the field of virtual reality was conducted. Popular platforms such as Reddit were browsed, where virtual reality enthusiasts share their experiences and knowledge. Reputable technology websites and publications that specialize in virtual reality hardware and software were also consulted. By analyzing user experiences, expert recommendations, and the features of various software solutions, the most effective and widely-used third-party applications for multicasting on Oculus Quest 2 devices were determined. These include Virtual Desktop, Open Broadcaster Software (OBS) Studio, Seths Web Based Session Monitoring, and Screen Copy (scrcpy).

In the context of multicasting on Oculus Quest 2 devices, Virtual Desktop is a widely-used application that enables users to wirelessly stream their PC VR games and applications to their Oculus Quest 2 device [VRDesktop, 2022]. Virtual Desktop works by wirelessly transmitting the display output and user input between a PC and an Oculus Quest 2 headset, effectively turning the VR headset into an extension of the computer's display. In a professional context, it can be used to mirror the displays of a single or multiple computers. By using a multi-cast networking setup, Virtual Desktop allows users to share their VR experiences with a larger audience in real-time [VRDesktop, 2022].

However, it's important to note that Virtual Desktop does not have the capability to view more than one headset simultaneously on a single computer [VRDesktop, 2022]. This means that it is not be an ideal solution for professional settings where multiple users can display their screens simultaneously.

Open Broadcaster Software (OBS) Studio is a powerful, open-source software for video recording and live streaming, widely used by content creators, gamers, and professionals alike [Software, 2022]. With its versatile plugin system, OBS Studio can be extended to accommodate various sources, including Oculus Quest 2 devices, enabling the capture and display of their output on a single computer [Software, 2022]. This capability makes it easier for professionals to manage, monitor, and present VR content.

OBS Studio works by allowing users to create multiple scenes that consist of various sources, such as video capture devices, display captures, and more. Users can switch between these scenes during a live stream or recording, offering flexibility and control over the content displayed [Software, 2022].

Despite its strengths, OBS Studio does not provide a built-in solution for multicasting or simultaneous viewing of multiple Oculus Quest 2 headsets. While it is possible to capture and

display the output of a single Oculus Quest 2 device, integrating multiple devices into a single OBS Studio session would require the use of additional plugins or software. These added components can increase the complexity and technical requirements of the setup, making it less user-friendly and accessible.

Given the project's goal of finding a straightforward and user-friendly approach to managing multiple VR headsets, OBS Studio may not be the most optimal solution. Its lack of native multi-casting support, combined with the added complexity of integrating additional plugins or software, might present challenges for users seeking an efficient and accessible way to manage and monitor multiple Oculus Quest 2 devices simultaneously.

It is important to mention an earlier project that attempted to address the mirroring issue when Oculus casting did not yet exist. Seth Angell, a student at the University of North Carolina Wilmington, who also worked on the Virtual Access to STEM Careers (VASC) project [Angell, 2022]. The project aimed to develop a solution for mirroring the display of an Oculus headset, specifically targeting an educational setting.

Seth created a project called "Web-Based Session Monitoring," which utilized Unity, WebRTC, Docker, and a Node.js server to broadcast a live view of the display of the Oculus headset [Angell, 2022]. This innovative project allowed instructors or administrators to monitor the VR experiences of users in real-time through a web interface. Additionally, it included an alert system where a teacher could view a headset's display and type messages that would be sent and displayed within the Oculus headset for the wearer to see, enhancing communication and guidance within the VR environment [Angell, 2022].

While Seth's project made significant strides in addressing the mirroring challenge, it did not fully solve the multicasting needs of this project. The Web-Based Session Monitoring project focused primarily on enabling live monitoring of an individual Oculus headset and provide a communication channel between the instructor and the user. However, it did not provide a native solution for multicasting, where multiple headsets could be simultaneously displayed and managed on a single computer [Angell, 2022].

Lastly, screen Copy(scrcpy) is an open-source screen mirroring application that allows users to display and control Android devices, including the Oculus Quest 2, via a USB connection or wirelessly through TCP/IP [Vimont, 2022a]. In a professional setting, scrcpy can be utilized as a viable solution for multicasting multiple Oculus Quest 2 headsets in a controlled environment,

enabling instructors or administrators to effectively manage and monitor the VR experiences of multiple users [Vimont, 2022a].

Key advantages of using scrcpy in a professional context for multicasting Oculus Quest 2 devices include high performance [Vimont, 2022a]. Scrcpy offers low-latency screen mirroring with minimal resource usage, providing a smooth and efficient experience for both the end-users and the administrators managing the headsets [Vimont, 2022b]. Another advantage is cross-platform compatibility. Scrcpy is available for Windows, macOS, and Linux, making it a versatile solution for organizations with diverse hardware and software ecosystems [Vimont, 2022a]. Scrcpy can be easily installed and configured, with minimal dependencies and straightforward command-line options. This simplifies the process of setting up and managing multiple Oculus Quest 2 devices for multicasting purposes [Vimont, 2022a]. Scrcpy also allows administrators to adjust various settings, such as screen resolution, bit rate, and frame rate, to optimize the performance of the screen mirroring based on the specific requirements of the environment [Vimont, 2022c]. Lastly, scrcpy is cost-effective. As an open-source solution, scrcpy offers a budget-friendly alternative to commercial screen mirroring applications, without compromising on quality or functionality [Vimont, 2022a].

Based on the comparison Table 2.1, Scrcpy emerges as the best solution for multicasting with Oculus Quest 2 devices for several reasons. Firstly, it supports all major platforms, including Windows, macOS, and Linux, which ensures broad compatibility. Secondly, Scrcpy is free, making it a cost-effective option for users, unlike some other solutions, such as Virtual Desktop and ClassVR.

A key advantage of Scrcpy over other alternatives is its native multicast capability, which allows for simultaneous display and management of multiple headsets on a single computer. This functionality is not offered by Oculus Casting, Virtual Desktop, OBS Studio, or Seth's Web-Based Session Monitoring Project. Additionally, Scrcpy provides wireless capability and is rated as highly user-friendly, making it easy for for this project to implement and manage.

In summary, Scrcpy stands out as the most suitable solution for multicasting with Oculus Quest 2 devices due to its platform compatibility, cost-effectiveness, native multicast capability, wireless support, and ease of use..

Solution	Platform	Cost	Multicast Capability	Wireless Capability	Ease of Use
Oculus Casting	Windows, macOS and Linux	Free	No	Yes	High
ClassVR	Windows and macOS	Paid Subscription	Yes	Yes	Moderate
Virtual Desktop	Windows and macOS	$19.99	No	Yes	Moderate
OBS Studio	Windows, macOS and Linux	Free	No	Yes	Low
Web Based Session Monitoring	Windows, macOS and Linux	Free	No	Yes	Moderate
Scrcpy	Windows, macOS and Linux	Free	Yes	Yes	High

Table 2.1: Comparison table for third-party multicasting solutions for Oculus Quest 2 devices

Chapter 3

Proposal

3.1 The Problem

Presently, casting for a single Oculus headset poses no difficulties, as this issue has been adequately addressed. However, as Oculus headsets become increasingly prevalent in homes, institutions, and businesses, the capability to monitor multiple devices within a unified space remains elusive. With the growing number of devices being sold, the need for a multicasting solutions became prevalent through my endeavors working towards a final product of the VASC project. Through forums on the Meta Quest website I discovered no documentation on this issue or potential implementation to resolve this use case.

The value of an inside view lies in fostering a sense of belonging or understanding through different scenarios. For instance, this could relate to parents observing their children playing different games simultaneously. Predominately within an educational context, a teacher could monitor students' progress as they complete activities, while in a workplace, new employees could undergo training with a supervisor available to offer assistance or ensure they are on the right track.

Observing firsthand experience of the VASC sea turtle simulation, this project recognizes the importance of providing a real time display within a controlled space to teachers whose students may have questions about completing tasks. The objective of this project is to develop a comprehensive solution that encompasses a range of functions and features to address the multicasting challenge. Ultimately, the goal is to provide a user-friendly system that enables seamless monitoring of multiple headsets with just a few clicks of the mouse. With that being said, below we will dive into the specifics

of the proposed solution.

3.2 Pitch

In order to address the challenges associated with multicasting multiple Oculus headsets, this proposal outlines the development of a user-friendly, executable desktop application that requires no in-depth understanding of its underlying mechanics. Designed with the same ease of use as a standard Microsoft Word file, the program aims to satisfy a teacher, or instructors needs to view all Oculus Headsets to be used within an educational setting.

The general workflow for enabling wireless multicasting is as follows. The administrator connects the desired number of Oculus headsets to a computer, within reasonable limits, such as 15-30. The administrator launches the application by double-clicking its icon. Upon loading, the administrator inputs the number of devices to be displayed, allowing the program to organize the display windows efficiently. The administrator initiates the connection of headsets via Wi-Fi by clicking "connect". Once connected, the instructor clicks "start" and proceeds to distribute the headsets as needed.

While this overview outlines the fundamental process, it is essential to consider factors such as frames per second, processing power, and GPU constraints. These considerations will be explored further in subsequent sections of this proposal. As proposed deliverables, I have noted a few features from personal experience, which is subject to be changed based off of data collected from potential end users, to be included into this project.

Features:

- An executable desktop icon that once pressed can boot up an application for the user.
- A simplistic UI that provides ease of use and abstraction from the inner workings behind the application.
- Screen Copy, an open source screen mirroring application that can obtain the display of the Android device.
- Python, with the use of libraries sush as tkinter, sys, subprocessing, and win32gui, to create a GUI interface for input and grid view organization of displays.

- Display labeling, attach a customizable name to the screen to keep track of the wearer.
- Wireless casting directly to the device via TCP/IP.
- Clickable display to allow the administrator to expand and diagnose a problem the wearer my have.

3.3 Technology Stack Justification

3.3.1 Desktop Application vs. Web Application

Stepping into this untouched territory I'm confronted with the decision of creating a web application or an executable desktop application. Based off of the information that I have collected, there are several reasons I'm choosing to create an desktop application rather than a website to stream content. Below are a few reasons that I have decided to go down this route.

First is performance, native applications can take advantage of the device's hardware more effectively than web applications [Davidson, 2022], which may lead to improved performance for resource-intensive tasks such as streaming high-quality video or processing real-time data. Another is offline capabilities, Internet access is not necessary while working with a desktop application, as it is installed in the system and you are leveraged to access it any time [Positiwise, 2021]. If the user were unable to connect to the internet then they would still be able to tether the stream to their computer. They can store content locally and still offer functionality even when offline, whereas websites require a continuous connection [Nepal, 2021]. Security and control, by developing a native application, the creator can have more control over the environment in which the content is streamed, [Davidson, 2022] potentially providing a more secure experience for the user. This can also help protect proprietary content and prevent unauthorized access or distribution [CE-IT, 2021]. Better user experience, "native applications can provide a more consistent, smooth, and customizable user interface, which can lead to increased user satisfaction and engagement [Davidson, 2022].

3.3.2 Why Python and not Java or C++?

Choosing Python to create a GUI over Java or C++ can have several advantages, depending on the specific requirements and goals of the project. Some reasons to choose Python for creating a GUI are as follows.

Need	Alternatives	Pros	Cons	Choice	Justification
Stream Content	Desktop	Better performance, install all that is needed	Many have less accessibility compared to web apps	Desktop	Improved performance, offline capabilities, better user experience, and more control over the streaming environment
	Web Application	More accessible, potentially easier to update	Performance limitations, need continuous internet		

Table 3.1: Desktop Application vs. Web Application

Python is known for its simple syntax and readability, making it easier to use [JavaTpoint, 2021]. With previous experience with this language allows for quicker development and debugging compared to languages like Java or C++. Python has a wide variety of libraries and frameworks available for creating GUIs, such as Tkinter, PyQt, PyGTK, Kivy, and PySide [Chaetognathan, 2023]. Though Java has frameworks such as JavaFX and C++'s QT framework, these libraries described provide python a range of options to suit the needs, making it easier to create functional, attractive, and customizable interfaces [NetworkInterview, 2021]. Python, along with its GUI libraries, typically offers cross-platform compatibility, meaning the same code can run on multiple operating systems (e.g., Windows, macOS, Linux) with minimal modifications [Chaetognathan, 2023]. Python's ease of use, extensive libraries, and fast development cycle make it an excellent choice for rapidly prototyping and testing GUI applications designed toward teachers to gather opinions and requests for future development. Though Python as well as Java and C++ have a large and active community, regardless of the choice I could obtain support through forums, tutorials, and code examples [Chaetognathan, 2023] to help through out the development process. This can be beneficial when encountering challenges during development, as chances are, someone else has already solved a similar problem through syntax before in the past. Using Python for GUI development enables seamless integration with other Python libraries, "such as data processing, machine learning, or web development libraries," [Waraich, 2023] making it easier to develop complex applications.

Need	Alternatives	Pros	Cons	Choice	Justification
Create a GUI	Python	Simple syntax, extensive libraries, cross-platform compatibility, large community, personal experience	Slower compared to Java or C++	Python	Ease of Use, extensive libraries, faster development time with experience, seamless integration with other python libraries
	Java	JavaFX framework, potentially faster than python	More complex syntax, fewer GUI libraries compared to Python		
	C++	Qt framework, faster than Python	More complex syntax, fewer GUI libraries compared to Python, steeper learning curve		

Table 3.2: Programming Language for GUI

3.4 Wireless Casting

3.4.1 Android Debug Bridge

Android Debug Bridge (ADB) is a versatile command-line tool that facilitates communication between a computer and an Android device. [Android, 2023] As part of the Android SDK (Software Development Kit), ADB allows developers and IT professionals to perform various tasks, including installing and debugging applications, managing files, and accessing device logs. [Android, 2023] In this overview, we will discuss the key features of ADB, its applications in professional settings, and its impact on the development and management of Android devices.

One of the key features of the Android Debug Bridge (ADB) is its support for wireless connections, which enables users to create wireless casting to TCP/IP. [Android, 2023] With ADB, users can connect to Android devices over Wi-Fi and perform tasks such as installing and debugging applications, managing device settings, and accessing device information.

To set up wireless casting with ADB, users first need to connect their Android device to their computer via USB and enable USB debugging. Then, they can use the ADB command-line interface to establish a wireless connection between the device and the computer. This can be done by running the command "adb tcpip port" where "port" is a specified port number. [Android, 2023]

Once the wireless connection is established, users can disconnect the USB cable and use ADB commands over Wi-Fi to interact with the Android device. For example, they can use the "adb connect" command to connect to the device, and then use commands such as "adb shell" to access the device's shell, "adb push" to transfer files to the device, and "adb pull" to retrieve files

from the device. [Android, 2023]

Wireless casting with ADB can be particularly useful for developers and IT professionals who need to manage multiple Android devices at once, as it allows them to connect to and control devices without the need for physical cables. It can also be beneficial for automated testing, as it enables developers to run automated tests on devices without the need for manual intervention. Overall, ADB's support for wireless connections offers users increased flexibility and efficiency in managing Android devices, making it a valuable tool in the professional sphere.

3.4.2 Realtime Casting

For demo purposes, I aim to have a working application similar to figure 3.1 below that will allow a user to see what the VR headset is displaying and chnage the names of the devices. This is simply to act as visual proof that the system is working, as without it it would be hard to demonstrate.

3.5 Screen Copy (Scrcpy)

3.5.1 Screen Copy

Scrcpy, which stands for "screen copy," is an open-source application that enables users to display and control Android devices via a USB connection or wirelessly from a computer running Windows, macOS, or Linux. [Vimont, 2023] Developed by Genymobile, scrcpy has garnered widespread recognition for its speed, efficiency, and versatility in managing and controlling Android devices. In this brief overview, we will discuss the key features of scrcpy, its potential applications in professional settings, and the advantages it brings to various industries.

From Scrcpy main website scrcpy is designed to be lightweight and efficient, ensuring low-latency screen mirroring and responsive control of Android devices. Scrcpy works seamlessly with Windows, macOS, and Linux operating systems, providing a uniform experience for users across different platforms.Though primarily USB-based, scrcpy also supports wireless connections through ADB (Android Debug Bridge), allowing for greater flexibility in device management. Scrcpy does not require root access or the installation of additional software on the Android device, preserving its security and stability. Scrcpy also allows users to modify various settings, such as screen resolution

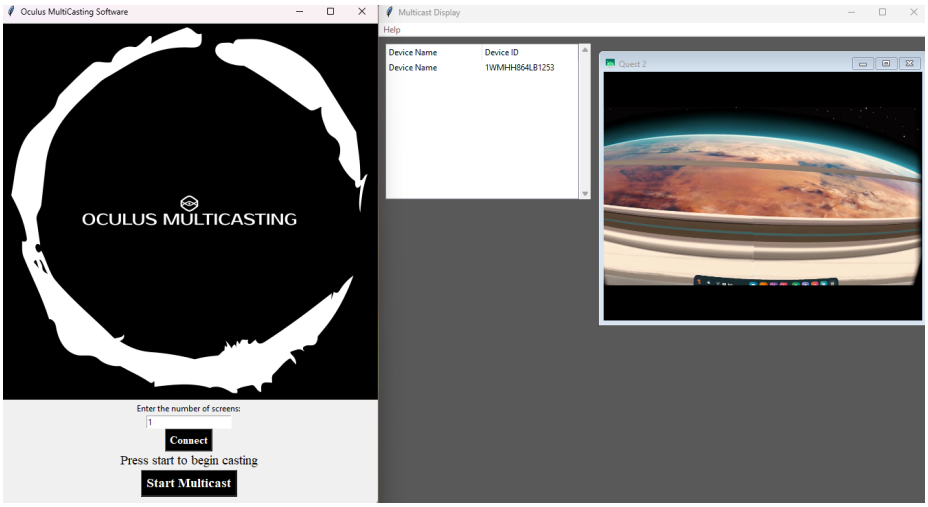

Figure 3.1: Example of potential Application UI

and bit rate, to optimize performance and tailor the experience to their needs.[Vimont, 2023]

3.6 Docking Device

For the successful implementation of this project, it is essential to ensure that all Oculus Quest devices are properly connected to a computer to establish a seamless connection. A typical computer can support connections to 3 or 4 devices simultaneously via USB-C or USB ports; however, this capacity may be insufficient to meet the demands of the project.

A viable solution to address this limitation is to invest in a docking device that can expand the computer's connectivity options by adding additional USB ports as required. These docking devices are commercially available and are designed to accommodate a diverse array of devices, including the Oculus Quest. The price range for such hardware varies, with options available between $20 and $100.

By selecting an appropriate docking device that aligns with the project's requirements and budget, it is possible to enhance the computer's connectivity capabilities, thereby ensuring the smooth operation of the application and facilitating a more efficient workflow.

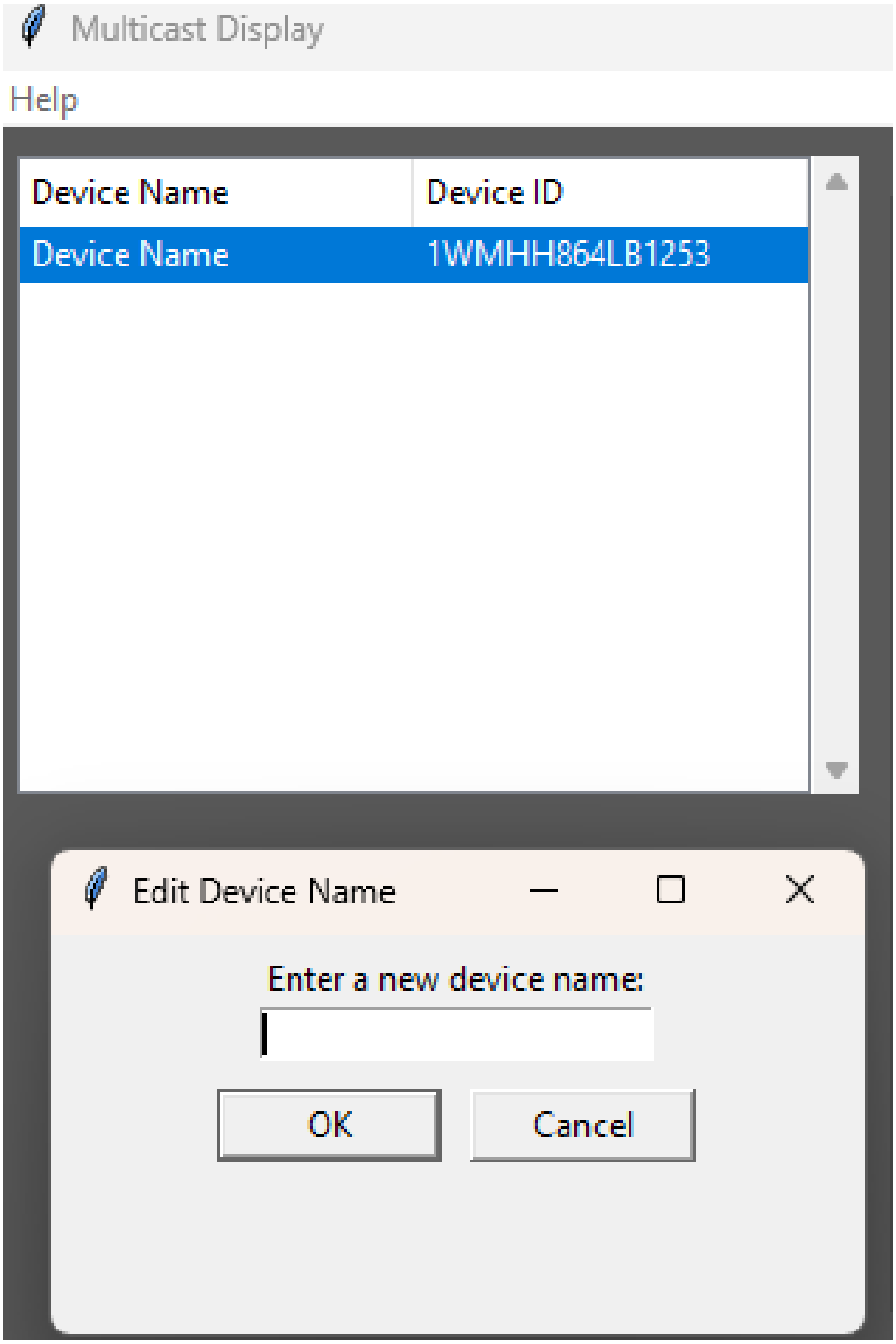

Figure 3.2: Example of editable device name

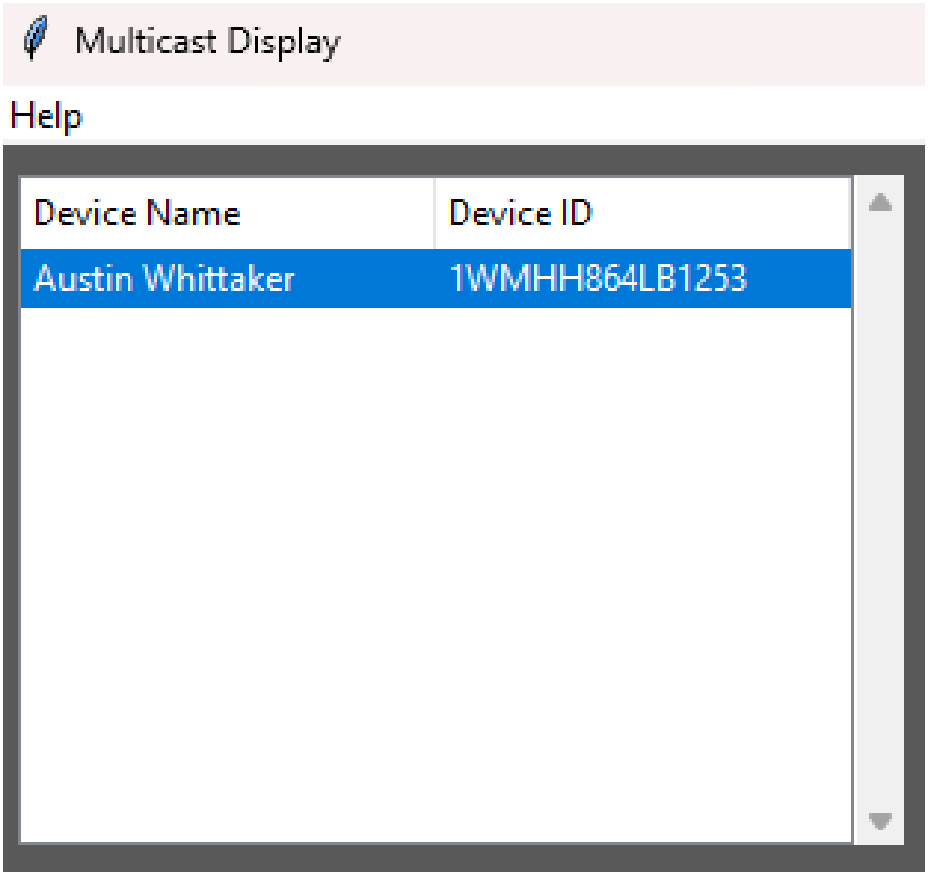

Figure 3.3: Example of post editing of device name

3.7 Deliverables

The product I am proposing to deliver at the end of my capstone can be defined by the following deliverables:

- **Application:** A user interface that provides ease of use for the user unaware of the inner working on the back end. Figure 3.1
- **Connection:** Connecting 15-30 Oculus Quest devices using the application allowing for wireless casting.
- **Streaming:** A service which enables the receiving and relaying of streamed video to clients for one or more devices in a grid view design. Figure 3.1
- **Device naming:** Allow the user to name the connected devices to keep track of the wearer. Figure 3.2 and figure 3.3
- **Packaged Application:** A downloadable installer to allow the user to download and obtain everything they may need in order to run the application.

I will revisit these predictions after the project is completed and examine where they were accurate, in what ways they fell apart, and discuss the challenges I faced with these solutions and my lessons learned throughout this Capstone project.

Chapter 4

Planning

This Capstone project aims to develop a comprehensive solution that enables teachers to monitor multiple Oculus headsets within a unified space. As with any new project, there are a few high-level assumptions that must be made up front to ensure that this project is feasible. This chapter will outline the assumptions made and the steps to be taken to create a successful solution including an estimated timeline of milestones.

4.1 High Level Assumptions

In this section, we discuss the high-level assumptions made during the development of a prototype application designed to enable multicasting of multiple Oculus Quest 2 displays in a single interface. This prototype aims to validate the feasibility of implementing such an application in schools for curriculum-based learning. The assumptions made were based on the experiences obtained from testing the Virtual Access to STEM Careers project at D.C. Virgo, an elementary school in Wilmington, North Carolina. One assumption made is that the application should be a desktop application that can run on a teacher's computer. This would ensure ease of use and integration within existing school infrastructure. To create an intuitive and user-friendly interface, we utilized the Python GUI framework Tkinter. The interface consists of an opening window welcoming the user, along with a series of buttons designed to enable straightforward user interaction. I implemented Scrcpy and a series of Python scripts running in the background to facilitate the connection process. These scripts execute command-line operations to connect the Oculus Quest devices to the

appropriate Wi-Fi port numbers, starting at 5555 and incrementing upwards based on the number of headsets connected. Once the connection process is complete, a secondary window opens, displaying all connected Oculus headsets within a contained space for the user to view. This layout allows for easy monitoring and management of the devices. During the prototype design process, I included a list column on the left-hand side of the secondary window to display all connected headsets. This feature allows the teacher to keep track of and rename headsets, facilitating the identification of which student is using each headset.

The prototype-first approach was employed to ensure that the creation of such an application was indeed possible. Once the prototype was built, it could be tested by users who might potentially implement this system in their classrooms. This approach allows for feedback and improvements to be made before deploying the application for curriculum-based learning in schools.

4.2 Testing and Data Collection

The desktop application GUI will be tested in a controlled environment, specifically in classrooms with teachers who have agreed to participate in the testing. The application will be tested on both Mac and Windows operating systems to ensure compatibility with the most commonly used systems.

To ensure the effectiveness of the desktop application GUI, it is necessary to gather data from teachers who will be using it in their classrooms. This data will include information on teacher needs, preferred features, and areas where they would like to see improvement. The data will be collected through a combination of questionnaires and interviews, and the results will be analyzed to inform the development of the application.

4.3 Component Level Assumptions

For this project, there are multiple components and there are several different assumptions that have been made, guiding the design and development of each component. These assumptions encompass various aspects, such as technology selection, customization versus pre-existing solutions, and defining the necessary features to meet users' needs. To provide a clear understanding of the project's scope and objectives, I present these assumptions and the rationale behind the ideology of

each decision throughout this section.

4.3.1 Determining Optimal View Display Style

The application's design takes into account the varying preferences of teachers in terms of display styles for monitoring their students' devices. Recognizing that different teachers may have different preferences, I aim to determine the most suitable display style based on user feedback gathered from interviews, questionnaires, and testing. Two primary display style options are being considered: a grid view-based display and a zoom-style view. The grid view-based display organizes connected headsets in a grid-like arrangement, offering an overview of the entire classroom. On the other hand, the zoom-style view focuses on a single student's headset display at a time, allowing for individualized attention. To evaluate and decide on the optimal display style for the application, I will conduct user testing with interviews and questionnaires. By engaging potential users in the decision-making process, I aim to identify the most suitable display style that meets the diverse needs of educators.

Face-to-face or virtual interviews will provide qualitative insights into the users' experiences and preferences regarding the display styles. Teachers will have the opportunity to express their opinions and discuss the advantages and disadvantages of each display option, contributing valuable insights to the decision-making process. Questionnaires will be distributed among potential users to gather quantitative data on their preferences for display styles. This method allows for a broader scope of feedback and ensures that the opinions of a wider range of teachers are taken into account.Once the interviews and questionnaires have been conducted, the gathered feedback will be analyzed to determine which display style best meets the needs and preferences of educators. The chosen display style will be implemented in the application to provide a clear and accessible monitoring experience that caters to the diverse requirements of teachers in classroom settings.

4.3.2 Secondary Monitor Display for Mobility

In this section, I explore the idea of creating a secondary monitor display to enhance the mobility of the application, allowing teachers to more effectively assist students while walking throughout the classroom. This feature was inspired by the experiences gained while testing the Virtual Access to STEM Careers project at D.C. Virgo in Wilmington, North Carolina. By enabling the

teacher to use a tablet as a secondary display, they can maintain oversight of the connected headsets and offer guidance to students even when not situated at their desk. One possible approach to achieving this secondary display functionality is by leveraging Duet Display technology. Duet Display is a software solution that enables users to extend their computer screen to an iOS or Android tablet, effectively transforming it into a secondary monitor. This technology allows for seamless and wireless screen extension, providing enhanced flexibility and mobility. Due to the project's time constraints, the consideration must be made whether to replicate Duet Display technology or directly utilize it in the application. Replicating the technology would potentially save costs for the end users, but may consume valuable time and resources. Alternatively, directly integrating Duet Display could expedite the development process while ensuring compatibility with an established and well-tested technology.

The decision to replicate or utilize Duet Display technology, as well as the overall usefulness of the secondary monitor display feature, will be determined through user testing, questionnaires, and interviews. This approach will ensure that the application's design addresses the needs and preferences of its users. User testing will involve having teachers interact with the application in a classroom setting, both with and without the secondary monitor display feature. This hands-on experience will provide invaluable feedback regarding the usability and effectiveness of the feature in real-world scenarios. After conducting user testing, questionnaires, and interviews, the feedback gathered will be analyzed to determine the most effective and desirable approach to incorporating a secondary monitor display feature in the application. By prioritizing user preferences and needs, the application's design will cater to the diverse requirements of teachers in classroom settings.

4.3.3 Desktop Installer Program

An essential aspect of ensuring seamless user experience is providing an easy-to-use desktop installer program that automates the installation process for the end-users. This section discusses the creation of such an installer, which would download all necessary components for the application, including Python, required libraries, Scrcpy, and ADB, across different operating systems such as Windows and MacOS.

4.3.3.1 Unified Installer for Essential Components

A well-designed installer program simplifies the setup process for users by automatically downloading and configuring all required components, such as Python and its necessary libraries, Scrcpy for screen mirroring, and ADB for device communication. The installer would detect the user's operating system and tailor the installation process accordingly, ensuring compatibility with both Windows and MacOS environments.

4.3.3.2 Online Distribution through GitHub

One approach for distributing the installer program is through an online platform such as GitHub. GitHub provides a centralized location for hosting the installer and any associated files, ensuring easy access for end-users. Users can download the installer directly from the GitHub repository, and the installation process would automatically download and configure all the required components from the internet.

4.3.3.3 Advantages of a Comprehensive Installer Program

A well-designed desktop installer program provides numerous benefits, including:

- **Simplified Setup:** By automating the download and configuration of necessary components, the installer reduces the time and effort required from the user, facilitating a smooth and efficient setup process.
- **Consistent Installation:** An installer program ensures that all users receive the same version of the application and its dependencies, minimizing potential issues due to variations in components or configurations.
- **Ease of Access:** Distributing the installer through an online platform like GitHub ensures that users can easily access and download the necessary files, regardless of their geographical location or time constraints.
- **Cross-Platform Compatibility:** By tailoring the installation process to the user's operating system, the installer program ensures compatibility with various platforms, such as Windows and MacOS, broadening the application's reach and potential user base.

In conclusion, a comprehensive desktop installer program is a vital component of the application, simplifying the installation process for end-users while ensuring compatibility across different operating systems. By hosting the installer on a platform like GitHub, users can easily access, download, and configure the application, making it a valuable addition to the project.

4.3.4 Docking System for Initial Tethered Connection

The process of connecting multiple Oculus Quest 2 headsets in a classroom setting requires an initial tethered connection to establish their IP addresses, which subsequently allows for wireless connectivity. However, the average number of USB and USB-C ports available on a computer typically ranges from 4 to 8, which may be insufficient for classrooms with 15 to 30 students. To address this challenge, I propose the use of a docking system that can accommodate the required number of headsets. A USB or USB-C docking extender offers a cost-effective and practical solution for connecting multiple headsets simultaneously. By providing additional USB or USB-C ports, a docking extender enables a single computer to accommodate the necessary number of tethered connections for larger classrooms. Docking extenders are available in various price ranges, typically falling between 10 and 50 dollars, depending on their features and capabilities. The primary reason for establishing a tethered connection is to obtain the IP addresses of the headsets automatically, rather than manually inputting each headset's IP address. This approach streamlines the setup process, reducing the time and effort required for teachers to establish wireless connectivity between the headsets and the application. By employing a docking system, the project addresses the limitations of available USB and USB-C ports on a computer, ensuring that a sufficient number of headsets can be connected in larger classroom settings.

4.4 Timeline Predictions and Milestones

The following timeline outlines the key milestones in the development and testing of the desktop application GUI:

Project Component	Estimated Difficulty	Confidence in Estimate	Confidence In Ability To Build To Spec
GUI Application	Easy - 1 week	90%	100%
Display Style	Moderate - 3 Weeks	85%	90%
Secondary display	Hard- 5 weeks	75%	80%
Docking Hardware	Easy - 2 weeks (Shipment and implementation)	90%	95%
Installer package	Hard - 4 Weeks	80%	85%

Table 4.1: Estimation Of Component Development Timelines and Difficulty

Date	Milestone
01/29/23	Research and gather relevant and related works
02/12/23	Start writing chapter 1 and continue research
02/26/23	Start writing chapter 2
03/12/23	Begin implementing a prototype to ensure feasibility
03/26/23	Finish writing chapters 3
04/09/23	Test implementation using two devices for feasibility
04/23/23	Finish writing chapter 4
05/07/23	Complete Capstone proposal for Committee
05/21/23	Interview potential users and gather feedback for requested features
06/04/23	Git repository created and order Docking Hardware
06/25/23	Build GUI application based off of gathered feedback
07/09/23	Test new implementation to potential users
07/23/23	Build and merge display style based on potential users needs
08/06/23	Test new display system to ensure functionality
08/27/23	Build and implement secondary display for mobility
09/10/23	Test secondary display
09/24/23	Create installation package for shipment
10/08/23	Test installation package and secondary display
10/22/23	Complete installation package and chapter 5 begin writing chapter 6
11/05/23	Edited Chapter 5 for clarity
11/19/23	Complete Chapter 6
12/03/23	Edit Final Paper
12/10/23	Complete Final Capstone Defense for Committee

Table 4.2: Ideal Timeline With Milestones

Chapter 5

Development

The development phase marks a pivotal transition from conceptualization to the tangible realization of the virtual reality (VR) monitoring application. This chapter will narrate the journey of developing this project, shedding light on the technical decisions, challenges overcome, testing, and the progressive steps taken to bring the application to life.

5.1 Infrastructure

The foundation of any application lies in its infrastructure. For the VR mirroring application, the infrastructure was designed to ensure scalability, performance, and overall user satisfaction. The design choices made were not only to facilitate current needs but also to anticipate future expansion. The infrastructure serves as the bedrock upon which all functionality is built, ensuring that every component works together to deliver an optimal user experience. Further detailing will dive into the specifics of the hardware and software selection, libraries and the implementation related to the decision implementation.

5.1.1 Python GUI

Selecting Python for the graphical user interface (GUI) was a decision I made that was driven by Python's extensive libraries and community support. Utilizing The Tkinter library allowed me to construct essential GUI elements such as buttons, tables, windows, editable fields and an image to welcome the user. The aim was to create an interface that was not only functional but also inviting

and straightforward for the end-user.

However, it became evident that while the application was functionally sound, the aesthetics did not match the modernity the application deserved. This realization led me to seek out an upgrade in the visual department. My exploration led me to CustomTkinter by Tom Schimansky, which is a python UI-library based on Tkinter [Schimansky,]. This enabled me to provide new, modern and fully customizable widgets. It enabled a visual transformation, providing the application with a contemporary look featuring a sleek dark mode, more engaging buttons, and framed modules that display a more organized look.

The enhancements in the application's graphical user interface are vividly illustrated in figures 5.1 and 5.2, presenting a stark contrast to the earlier rendition seen in figure 3.2. In the revised layout, I adopted a more intuitive arrangement by placing the welcoming image to the left, thereby naturally guiding the user's gaze to the actionable buttons on the right. The buttons themselves have been altered with a rounded design, softening the visual aesthetic and inviting interaction. Surrounding these buttons, frames now delineate interactive areas, adding to the application's structure and navigability. Moreover, the entire application basks in the sleekness of a dark mode theme, offering a visually soothing experience that is easier on the eyes and underscores the modernity of the interface.

5.1.2 Screen Copy (Scrcpy)

Scrcpy played a central role in mirroring the VR headset displays to the user's computer. The Scrcpy application allows the ability to execute customizable commands to better fit the application, enhancing its performance and display output. Out of the box when running scrcpy the initial display shown on the screen, is a view through both the wearer's eyes. An example of this is shown in figure 5.3. This dual perspective, while effective, seemed to offer more detail than necessary for monitoring purposes. A decision arose as to strictly viewing either the left or the right eye. When doing some research on this topic I came across an article over eye dominance. Kate Green states that while everyone has a hand dominance the same goes for eye dominance. 67 percent of the population have a right eye dominance [Green, 2020]. Seizing on this insight, I refined the display output to feature the view from the right eye only, aligning with the majority's dominant perspective. This strategic crop, shown in figure 5.4, not only conserves screen real estate, as evidenced in figure 5.2, but also clarifies the user's focal point within the VR environment. Ideally, customizing the display to match

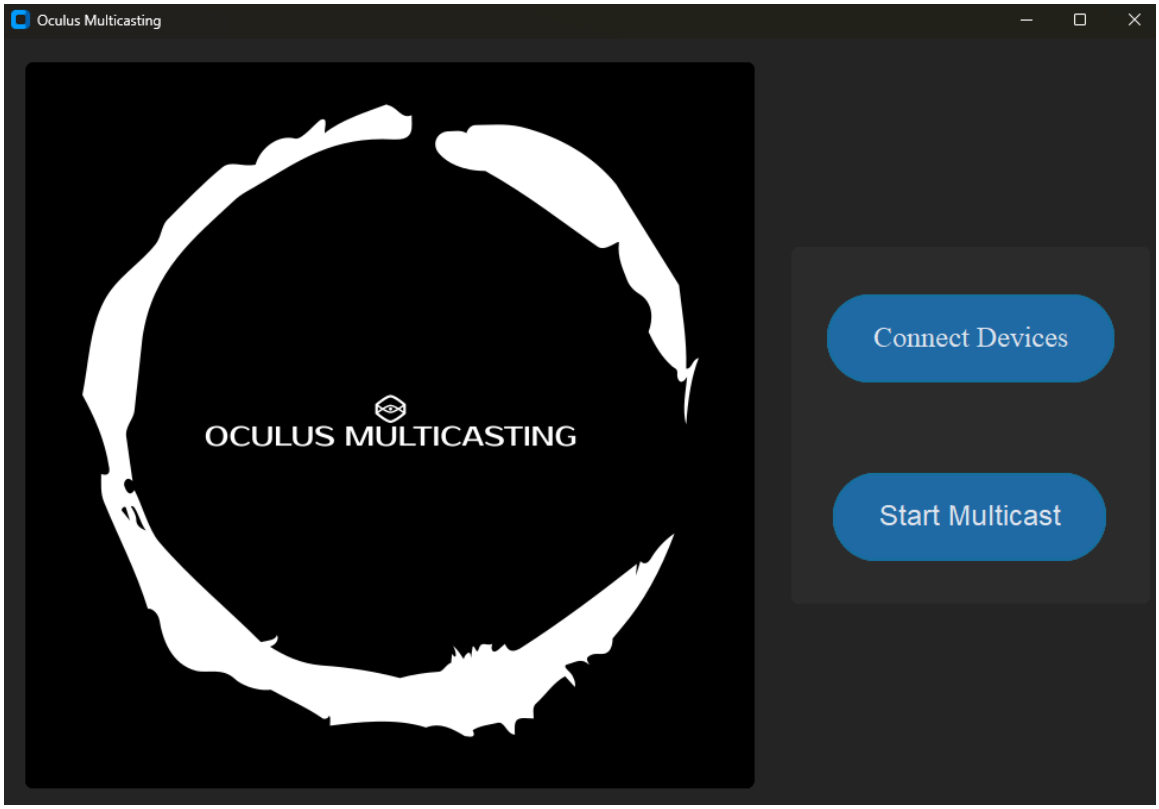

Figure 5.1: GUI design for the intital screen using customTkinter

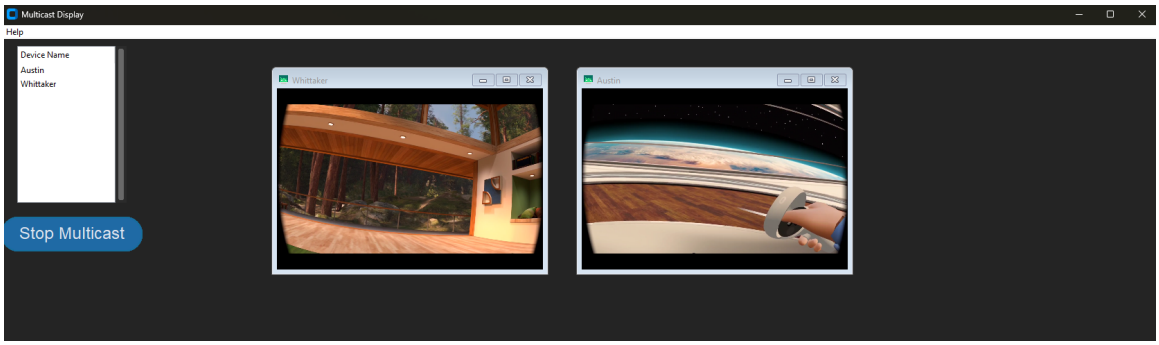

Figure 5.2: GUI design for viewing displays using customTkinter

an individual's eye dominance would enhance the application's personalization. However, given the project's time constraints, this feature remains on the horizon for future updates. Despite this, the current implementation marks a significant step forward in creating a more streamlined and focused user experience.

Managing multiple live streams from VR headsets within the classroom setting presented a considerable technical challenge. Through dedicated testing and iterative optimization, I was able to enhance the application to reliably facilitate concurrent monitoring of these streams. In this segment, I aim to discuss the specific aspect of frames per second (FPS) and its impact on the clarity and fluidity of the displayed content. Scrcpy's functionality to adjust the FPS proved to be pivotal in achieving a balance between display clarity and system performance. Through extensive trials, I evaluated the effectiveness of various FPS settings. Starting at 30 FPS, I noticed that rapid movements or quick turns would result in a lagging display. This prompted me to increase the FPS to 60, which significantly improved the smoothness of the visual output. However, I observed that as more headsets were connected, this high FPS setting began to strain the system. Ultimately, I found that setting the FPS to 45 struck the ideal balance—offering a crisp display and fluid motion without overburdening the system, even as the number of connected headsets increased. This optimized FPS setting ensured that each headset's display maintained clarity and the overall system operated without discernible lag. The capability to fine-tune the FPS is a feature that holds potential for inclusion in future updates of the application, allowing users to customize performance based on their specific hardware capabilities and requirements.

5.1.3 Android Debug Bridge

The android debug bridge (ADB) was the backbone for device communication and connection. In the development of this VR monitoring application, ADB's Daemon emerged as an indispensable tool, integral to bridging the gap between my development environment and the Android operating system ran on the Oculus headsets. Functioning as a background service on the devices, the ADB Daemon was necessary in establishing a robust communication channel between my development machine and the VR headsets.

This bridge facilitated a variety of crucial tasks such as executing shell commands such as recognizing connected headsets, and insuring a strong connection to the server. The ability to directly interact with the headsets via ADB was invaluable, as it worked seamlessly when paired

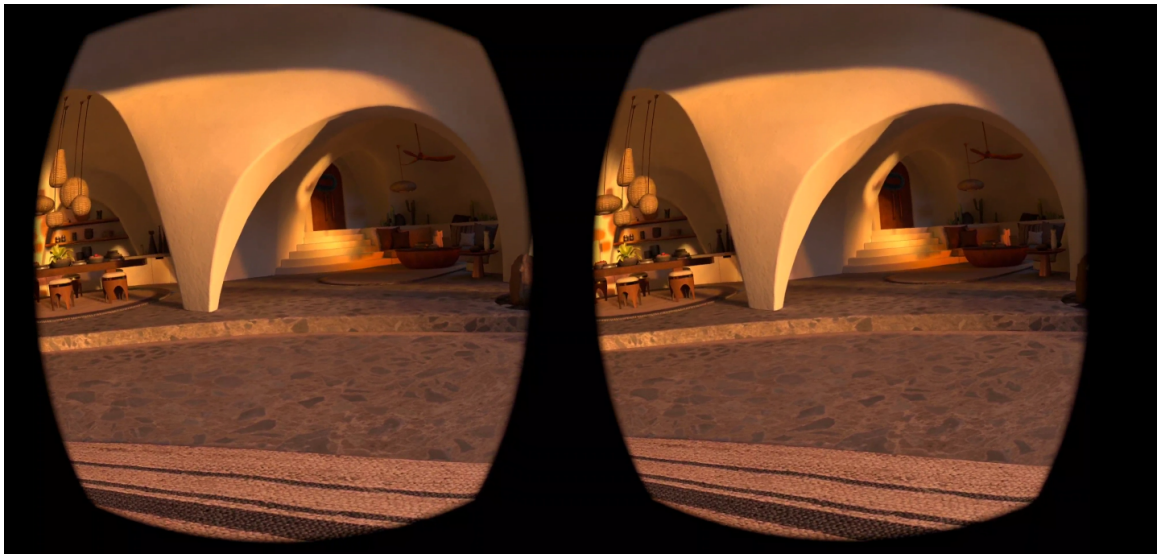

Figure 5.3: Rendering for default view when using scrcpy

Figure 5.4: Window display showcasing the crop around the right eye.

with scrcpy.

ADB's ability to work across various Android devices and versions further underscored its utility in the project. This universality meant that no matter the type or version of the Android-powered VR headset, I could rely on ADB to provide the necessary functionality.

In summary, the ADB Daemon was more than just a development tool; it was an integral part of the application's interaction with the VR headsets, enabling a seamless and secure connection process. Its comprehensive range of capabilities made it an irreplaceable part of the project's technical backbone.

5.1.4 Secondary Display

The concept of integrating a secondary display emerged as a vital feature, particularly for situations where users, seated at a desktop computer, need to maintain connectivity with VR users who have physically moved away from the primary workstation. My initial vision was to enable the computer's screen to be mirrored effortlessly onto a more portable device like a tablet or an iPad, thus bridging the spatial gap between the desktop user and the VR participant.

However, the development of such a feature, while beneficial, posed a significant challenge in terms of time and resource allocation. It became clear that dedicating resources to this would detract from refining the core functionality of the application, which was paramount given the project's time constraints. Consequently, I opted for a more pragmatic approach that leveraged existing solutions.

Rather than reinventing the wheel, I decided to integrate established remote access tools like AnyViewer or TeamViewer. Specifically AnyVeiwer which is a free and secure remote control software developed by AOMEI Technology. It provides quick and stable access to remote computers, with high-quality images that make it feel like you're using a local computer. AnyViewer supports secure remote access to your business or personal computer from any device and anywhere, with end-to-end encryption via a strong 256-Bit Elliptic Curve Cryptography (ECC) algorithm [any,].These platforms offer robust and free mirroring support, enabling users to remotely control their desktop computers via a tablet or iPad. This solution not only circumvents the need for extensive development of a new mirroring feature but also ensures reliability and ease of use, as these tools are well-tested and widely used. By integrating these tools, I was able to provide the desired functionality without compromising the development focus on the application's core features.

Figure 5.5: Vangree docking station used to connect all headsets to the computer

5.1.5 Docking Station

Recognizing the constraints posed by the limited availability of USB/USB-C ports on most computers, it became essential to devise a solution that could accommodate the connection of multiple devices in a streamlined and organized manner. The quest for an efficient and practical answer led me to the acquisition of a docking station, a choice that proved to be remarkably beneficial for the project.

I selected a highly capable docking station manufactured by Vangree for $69.00, as illustrated in figure 5.5. This device stood out due to its capacity to simultaneously connect up to seven different headsets, either through USB-C or USB connections and great reviews from other customers. This versatility was particularly advantageous, when used during development and testing.

The implementation of this docking station into my workflow was a game-changer. It significantly simplified the process of connecting multiple VR headsets, allowing for a more efficient setup and quicker transition between different testing phases. The time savings were substantial, enabling me to focus more on the development and testing of the application, rather than individually disconnecting and reconnecting all headsets to test when dealing with the hardware management.

In essence, the Vangree docking station not only brought about a much-needed organization to the device setup but also accelerated the development process by simplifying the connectivity challenges inherent in managing multiple VR headsets.

5.2 Core Functionality

As we delve into the core functionality of our virtual reality (VR) monitoring application, we explore the sophisticated mechanisms that enable its seamless operation and user-centric features. Central to this functionality is the innovative use of TCP/IP for wireless connectivity, the adept handling of window displays through pywin32, and the comprehensive Python implementation that brings it all together.

Leveraging TCP/IP protocols, the application establishes a robust and reliable wireless connection with the VR headsets. This wireless approach not only simplifies the setup by eliminating the need for cumbersome cables but also enhances the flexibility and range of the monitoring system. This connectivity is crucial for real-time interaction and monitoring, ensuring that instructors can maintain a consistent connection with each student's VR experience, regardless of their physical

location in the classroom.

In parallel, the application utilizes pywin32, a powerful tool for interacting with Windows APIs. This integration allows for sophisticated manipulation and control of window displays, a vital aspect when handling multiple live feeds from various headsets. Through pywin32, the application adeptly manages these windows, ensuring that each feed is displayed optimally in an organized fashion.

The Python implementation ties all these functionalities together, forming the backbone of the application. Python's versatility and robust library support enable seamless integration of different modules, from network communication to UI management. This integration is pivotal in delivering a cohesive and user-friendly experience.

A standout feature of the application is its device management capability, particularly the ability to rename VR headsets. This functionality, which can be applied to individual headsets or to all headsets en masse, is more than a mere convenience. It allows for efficient organization and identification of devices, which is crucial in a classroom setting where multiple headsets are in use simultaneously. This feature not only aids in the logistical aspect of managing the devices but also enhances the overall functionality of the monitoring system, making it a powerful tool in the educational technology landscape.

5.2.1 TCPIP

The implementation of TCP/IP in the context of our VR monitoring application is a critical aspect, especially regarding the wireless connection with the VR headsets. This process primarily revolves around the use of the TCP/IP protocol for network communication, with a specific focus on utilizing port 5555, a commonly used port for network communications in Android devices.

5.2.1.1 Implementation

For the Initial Setup and Configuration, the first step involves ensuring that all headsets are developer mode enabled and each VR headset is connect to the same local network as the monitoring computer. This is essential for establishing a common platform for communication. Once the headsets and the computer are on the same network they are plugged into the docking station shown in figure 5.5. The most important step to this process is allowing the connected computer to obtain access the Oculus quest headset. This step is mandatory due to Oculus's securty

and framework of the headset. Once the above is completed I initiate the process of setting up a TCP/IP connection.

1. Establishing a Connection on Port 5555: Port 5555 plays a pivotal role in this setup. This port is known for its use in network debugging and communication with Android devices, making it an ideal choice for this application. I configure all the VR headsets to listen for incoming connections on this specific port. This is done by executing ADB (Android Debug Bridge) commands that prepare the device for TCP/IP communication over this port.

2. Wireless Communication: With the headsets listening on port 5555, the monitoring application on the computer establishes a wireless connection to each headset. This is accomplished by sending requests to the designated port via scrcpy via the headsets unique IP address. For the use of TCP over UDP, the TCP/IP protocol ensures reliable and ordered delivery of packets, which is crucial for maintaining a stable and consistent connection.

3. Data Transmission and Monitoring: Once a stable connection is established, the application begins transmitting data between the VR headsets and the monitoring computer. This includes streaming the VR display from the headsets in a controlled space to the application via a python library pywin32. The user is then instructed with a pop-up window to then disconnect all headsets. The use of TCP/IP ensures that these data transmissions are fast, reliable, and wireless which is vital for real-time monitoring and interaction.

4. Handling Multiple Connections: The application is designed to handle multiple connections simultaneously, connecting to each headset on the same port but differentiating each device based on its unique IP address. This allows the application to monitor multiple headsets concurrently, providing a comprehensive overview of the entire classroom's VR interactions.

5.2.2 SCRCPY and PyWin32

The integration of Scrcpy and the PyWin32 library was instrumental in the creation of this VR monitoring application, tailored specifically for the Windows operating system. While the initial vision was to craft a cross-platform solution, certain technical decisions—such as employing PyWin32—naturally limited the application to Windows compatibility.

My aspiration was to deliver a universally accessible application, one that could seamlessly

operate across various operating systems including Mac and Linux. However, the project's scope coupled with the hardware and resources at hand guided the decision to focus on a Windows-centric GUI. The reality of development constraints led to this strategic choice, ensuring that we could provide a robust and reliable user experience for Windows users.

In this section, we'll explore the fusion of Scrcpy's mirroring capabilities with PyWin32's window management features, and how this synergy has resulted in a cohesive and efficient interface for those utilizing the Windows platform in their VR educational endeavors.

5.2.2.1 Integrating Scrcpy with Python

The core functionality of this application relies on Scrcpy to mirror the VR headset displays onto the monitoring computer. To achieve this, I implemented a Python script that sends commands to Scrcpy via command line. This integration required careful planning and testing to ensure that commands were executed efficiently and responses were handled correctly. By leveraging Python's versatility, I was able to create a dynamic script that not only initiates the Scrcpy session but also manages various parameters like display quality, size, window title and cropping to optimize the mirroring process.

5.2.2.2 Display Windows Management with PyWin32

So now that I can obtain the displays of each headset it is important that each headset is shown in a organized fashion. That is where the PyWin32 library comes into play. PyWin32's role in our application was pivotal in managing the display windows. This comprehensive library allowed me to access a wide array of Windows APIs directly from Python, enabling intricate control over the GUI elements and ensuring that they remain within the application. Using PyWin32, I automated tasks like opening and positioning the window displays for each VR headset stream in a grid like fashion such as 3 by 3 due to screen space. This automation was crucial in maintaining an organized and user-friendly interface, especially when dealing with multiple streams simultaneously.

5.2.2.3 Implementation of Scrcpy Command Execution and Window Embedding

The practical application of Scrcpy commands via Python, paired with the strategic use of the PyWin32 library for window management, forms the crux of our application's display capabilities.

```python
def start_screen_mirror_Name(device_id, x, y, width, height, entered_name):
    subprocess.Popen(["scrcpy", "-s", device_id, "--always-on-top",
                      "--window-x", str(x), "--window-y", str(y),
                      "--window-width", str(width), "--window-height", str(height),
                      "--crop", "1730:974:1934:450", "--max-fps", "45", f"--window-title={entered_name}", "--tcpip",
                      "--no-audio"])
    time.sleep(1)
    window = win32gui.FindWindow(None, "scrcpy - " + device_id)
    win32gui.ShowWindow(window, win32con.SW_MINIMIZE)
```

Figure 5.6: The execution of the Scrcpy command via Python and the subsequent management of the display windows using PyWin32, illustrating the seamless integration and automation achieved in the application.

This intricate implementation is crucial for the real-time mirroring of VR headset displays and their subsequent organization on the monitoring computer's screen.

To initiate the mirroring process, in figure 5.6 a Python function, start_screen_mirror_name, is employed. This function leverages the subprocess. Popen method to execute Scrcpy with a series of command-line arguments. These arguments are meticulously tailored to include the device ID for specificity, coordinates for window placement, dimensions for scaling, and the desired framerate to maintain performance consistency across devices. An example of this command execution is depicted in the provided code snippet, illustrating the precision and adaptability of our implementation.

Following the launch of the Scrcpy stream, the win32gui module of PyWin32 comes into play. The FindWindow method locates the newly created Scrcpy window, ensuring that each VR headset's display is individually identified and managed. The subsequent ShowWindow method then minimizes the Scrcpy window to maintain an uncluttered desktop environment.

In addition to initiating the display stream, the script includes a robust window embedding mechanism. The find_window function serves as a locator, identifying the target window by its class name, in this case, 'SDL_app'. Upon locating the window, the try_embed function is invoked, which employs a recursive embedding attempt. This function uses SetParent to nest the Scrcpy window within the target window, followed by MoveWindow to finalize its position and size. This recursive approach, illustrated in the second code snippet, ensures persistent attempts at embedding, retrying every second until successful or until the maximum number of attempts is reached.

Should the attempts surpass the defined threshold without success, a messagebox is prompted via showerror, alerting the user to the embedding failure and suggesting a retry. This proactive error handling is indicative of the application's user-centric design, prioritizing continuous operation and clear communication with the user.

```
def embed_window(proc, target_window, x, y, width, height):

    Austin Whittaker *
    def find_window():

        class_name = "SDL_app"
        hwnd = win32gui.FindWindow(class_name, None)
        if hwnd:
            return hwnd
        return None

    max_attempts = 10
    attempts = 0

    Austin Whittaker
    def try_embed():
        """Attempt to embed the window inside the target window...."""
        nonlocal attempts
        attempts += 1
        if attempts <= max_attempts:
            hwnd = find_window()
            if hwnd is not None:
                win32gui.SetParent(hwnd, target_window.winfo_id())
                win32gui.MoveWindow(hwnd, x, y, width, height, True)
            else:
                app.after(1000, try_embed)
        else:
            messagebox.showerror("Error", "Failed to embed the window. Please try again.")

    app.after(1000, try_embed)
```

Figure 5.7: Implementation for embedding a window within the application.

The implementation culminates in a structured and user-friendly interface, where each VR headset's display is neatly embedded within the monitoring application, as visualized in figure 5.7. This figure captures the essence of our window management strategy, showcasing the efficiency and organization achieved through the Python script. Although this implementation worked the maximum number of displays that I were able to display were 6 due to the resources provided. In the future I would like to test out more headsets to find a maximum number of headsets.

5.2.3 Headset Naming

To facilitate easy identification and monitoring within the VR classroom environment, I engineered a user-friendly device naming protocol as a core feature of this application. Upon the initial connection and startup of the application, each VR headset is automatically assigned a default identifier, typically the unique identification number associated with the Oculus headset. This unique identifier ensures that each device is distinguishable at the outset.

However, recognizing that mere numerical identifiers could lead to confusion during active monitoring—especially when trying to correlate headsets to their specific users—I introduced the capability for custom device naming. This feature requires instructors to assign meaningful names to each headset prior to their distribution among students shown in figure 5.8. As a result, the monitoring process becomes significantly more intuitive, with each VR headset bearing a name that directly corresponds to the individual user. This not only streamlines the monitoring process but also adds a layer of personalization that enhances the overall management of the VR classroom experience.

5.2.3.1 Individual headset Renaming

Within the device management suite of our application, the flexibility to rename individual headsets on-the-fly emerged as a key functionality. Although the initial naming convention serves to clearly identify each participant, I anticipated scenarios where such identifiers might need to be updated. For instance, if a single student opts out of the session, and their headset is reassigned to another participant, it would be impractical to recall and reassign names for all headsets.

To address this, I implemented a feature allowing the instructor to seamlessly update the name of any single headset within the application. This targeted renaming capability negates the need for a complete system reset, thus saving time and avoiding disruption. The feature enhances

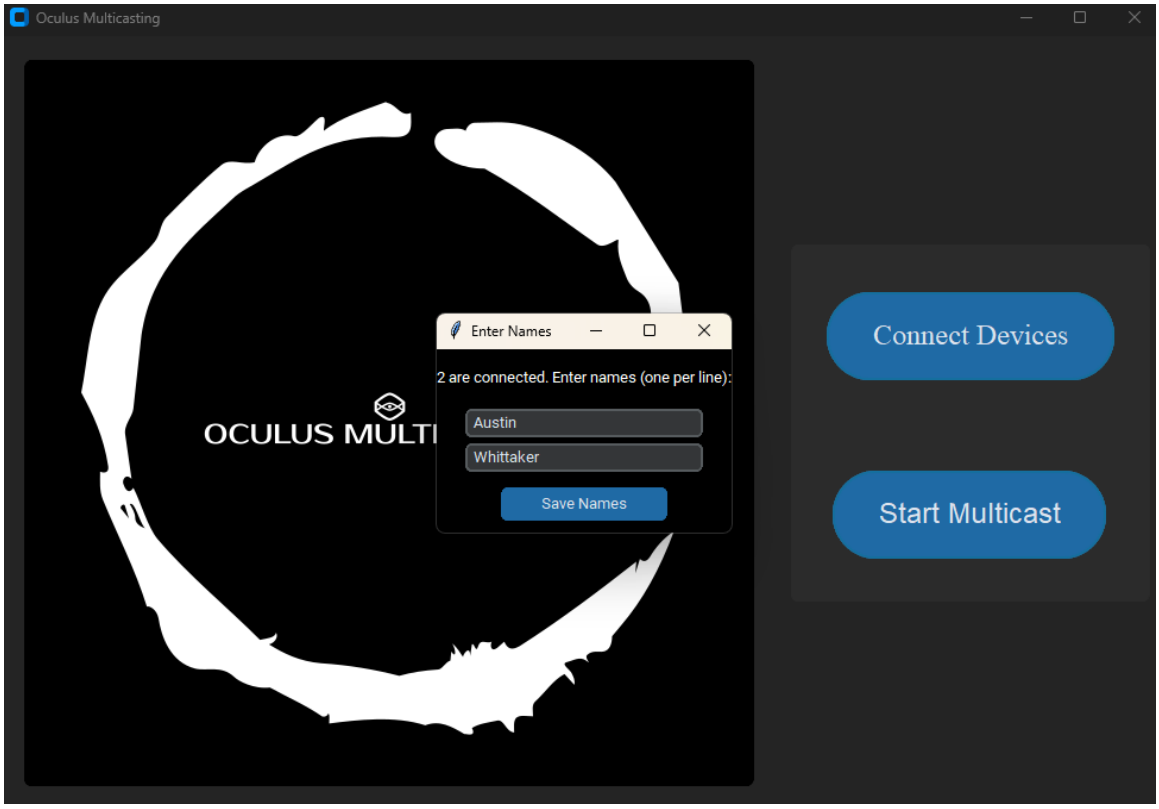

Figure 5.8: Naming the device headsets before application start up.

the application's adaptability to dynamic classroom environments, where changes often occur with little notice, ensuring that the transition from one user to another is as smooth as possible.

5.2.3.2 Implementation of Individual Headset Renaming Feature

The implementation of the device naming feature represents a blend of user-centric design and practical functionality within our VR monitoring application. As users initiate the connection of the VR devices, they are greeted with the opportunity to assign distinctive names to each discovered headset. This initial step is critical as it not only aids in identifying users but also personalizes the monitoring experience. Upon entering all the names, these identifiers are cataloged in a list that correlates each name to its respective device. This list becomes the backbone of our application's user interface, mapping each headset to a user-friendly monitor displayed in the application's tree view, as seen on the left side of the GUI shown in figure 5.2.

The ability to rename a headset individually is an extension of this functionality, offering users the flexibility to update the name of any headset in real time. Should the need arise—perhaps due to a change in the user of a headset or simply to correct a typo, the user can easily select the desired headset from the tree view and invoke the rename option through a right-click context menu. This intuitive interface for renaming ensures that the user can maintain an organized and accurate representation of all participants.

Once the renaming process is initiated, the application promptly updates its interface, reflecting the new name in both the tree view and the corresponding display window. This dynamic update is crucial for ensuring that the monitoring process remains uninterrupted and that the display windows accurately represent the current users.

Figure 5.6 illustrates the practical application of Scrcpy commands integrated within the Python script, enabling the display of the newly entered names as window titles. This seamless integration ensures that each window not only streams the content from the correct headset but also bears the appropriate label for quick and easy identification by the monitoring user.

In practice, the renaming of devices is implemented through a series of methodical steps within the script. Initially, the save_names function captures and stores the entered names shown in figure 5.9, adding them to an 'entered_names' list only if the input field is not left empty. Subsequently, the logic checks if the length of 'entered_names' matches the number of connected devices, ensuring that each device is paired with a name. If a renaming is requested, the rename_device

```python
def save_names():
    """
            Save the entered names to the 'entered_names' list.
    """
    global saveName
    saveName = True
    for entry in entries:
        name = entry.get()
        if name:  # Only if the entry is not empty
            entered_names.append(name)
            print(name)
    name_entry_window.destroy()
```

Figure 5.9: Python script that saves the initial names.

function is called shown if figure 5.10, which first verifies the selected item's existence within the treeview table before prompting the user for a new name. The application then updates both the visual component in the tree view and the backend list to reflect the change, as depicted in the respective code snippets.

This renaming feature embodies the application's commitment to providing a user-friendly and efficient tool for VR classroom management, streamlining the process of headset monitoring and enhancing the overall user experience.

5.2.4 Batch Renaming of Headset Devices

Due to evolving needs of dynamic educational environments, a pivotal enhancement was integrated into this application—the capability for batch renaming of headset devices. This feature was conceived out of necessity to streamline the transition between different groups of students engaging with the VR headsets based off of previous experience with my time within the VASC project.

Recognizing the impracticality and time consumption involved in manually renaming each headset for new sessions, I introduced a function that allows all connected headsets to be renamed in a single operation. This bulk renaming tool is particularly useful when a fresh cohort of students arrives and necessitates a quick setup, allowing for a seamless handover of headsets without the logistical bottleneck of one-by-one renaming.

```python
def rename_device(event, tree):
    item = tree.focus()  # Get the selected item
    values = tree.item(item, "values")

    # Check if values are not empty and that the item is indeed a device to be renamed
    if values:
        current_name = values[0]
        new_name = simpledialog.askstring("Edit Device Name", "Enter a new device name:", initialvalue=current_name)

        if new_name and new_name != current_name:
            # Update the Treeview with the new name
            tree.item(item, values=(new_name,) + tuple(values[1:]))

            # Update the entered_names list
            try:
                index = entered_names.index(current_name)
                entered_names[index] = new_name
            except ValueError:
                print(f"The current device name {current_name} was not found in the entered_names list.")
```

Figure 5.10: Python script that showcases to rename the individual headset name.

With this feature, instructors can now efficiently reset the device names to match the new users, thereby maintaining an organized and up-to-date monitoring interface. This mass renaming capability significantly reduces downtime between classes and enhances the overall functionality of the VR educational experience.

5.2.5 Batch Renaming of Headset Devices Implementation

In the pursuit of efficiency, particularly when transitioning between different user groups, the implementation of a batch renaming feature within this VR monitoring application represents a significant stride forward. The functionality, as depicted in the accompanying code snippet, enables the renaming of all connected headsets in one cohesive action.

This process shown in figure 5.11 begins with the initiation of the renaming window, rename_entry_window, which presents a sleek interface against a dark background, as configured in the code. The window dimensions are set to accommodate the list of entry fields corresponding to the number of connected headsets, dynamically determined at runtime. The calculated center positioning ensures the window appears centered on the screen, providing a user-friendly experience.

With the new_entries list initialized to store the newly assigned names, the save_names_again() function stands ready to capture and commit these names to the entered_names list. Each entry widget, corresponding to a headset, is generated programmatically based on the number of connected devices, and users can input new names swiftly and efficiently.

```python
def renameing_entry_window(window, devices):
    rename_entry_window = Toplevel(app, bg="black")
    rename_entry_window.title("Rename all")
    rename_entry_window.geometry("500x600")  # Set to desired width x height

    numberOfConnectedDevices = len(devices)
    screen_width = rename_entry_window.winfo_screenwidth()
    screen_height = rename_entry_window.winfo_screenheight()

    # Calculate the center position
    x_position = (screen_width / 2) - (500 / 2)  # Adjust 500 to your window's width
    y_position = (screen_height / 2) - (600 / 2)  # Adjust 600 to your window's height

    rename_entry_window.geometry(f"{500}x{600}+{int(x_position)}+{int(y_position)}")

    global new_entries
    new_entries=[]

    Austin Whittaker
    def save_names_again():
        """
                Save the entered names to the 'entered_names' list.
        """
        count = 0

        for new_entry in new_entries:
            name = new_entry.get()
            if name:  # Only if the entry is not empty
                entered_names[count] = name
                print(name)
            count+=1
        rename_entry_window.destroy()
        start_multicast()

    label = customtkinter.CTkLabel(rename_entry_window,
                                   text=f"{numberOfConnectedDevices} are connected. Enter names (one per line):",
                                   text_color="white")
    label.pack(pady=10)

    for i in range(numberOfConnectedDevices):
        new_entry = customtkinter.CTkEntry(rename_entry_window, height=25, width=200)
        new_entry.pack(pady=2)
        new_entries.append(new_entry)  # Store the entry widget

    save_button = customtkinter.CTkButton(rename_entry_window, text="Save Names", command=lambda: [save_names_again(),window.destroy()])
    save_button.pack(pady=10)
```

Figure 5.11: Python script that showcases to rename the individual headset name.

Upon clicking the "Save Names" button, the save_names_again() function iterates over the entry widgets, updating the entered_names list with the new names if the entry fields are not left blank. This updated list is then reflected across the application's interface, simultaneously updating the display names for all connected headsets. The window is then closed, and the multicasting application restarts, showcasing the new names in the tree view, and thereby streamlining the renaming process for an entire class or group of VR participants.

This batch renaming feature not only saves considerable time but also reduces the complexity involved in device management, further enhancing the application's practicality and user experience in educational VR settings.

5.3 Packaging

When it came to packaging the VR monitoring application, my primary objective was to streamline the deployment process for end-users by offering a simple, downloadable installer. This installer would ideally encapsulate all the requisite components, thereby simplifying the setup experience. The envisioned outcome was an all-in-one solution that would facilitate immediate use upon download, eliminating any complex installation procedures.

However, as the project progressed, I encountered constraints related to time, which in turn impacted the breadth of deliverables I could reasonably provide. The reality of these limitations necessitated a shift in my packaging approach. After evaluating various options, I settled on utilizing PyInstaller—which analyzes all of the code to discover and every module within the library this script needs in order to execute. Then it collects copies of all those files – including the active Python interpreter – and puts them with the script in a single folder, or optionally in a single executable file [pyi,].

PyInstaller emerged as the optimal choice for a number of reasons. It offered the capability to bundle Python scripts, libraries, and dependent binaries into a single executable file, which was in alignment with my initial packaging goals. This approach meant that instead of a downloadable installer it would then be able to be distributed through a repository such as github. This decision not only resolved the issue of time constraints but also ensured that users would have the able to download this package manually though the use of cloning and following the ReadMe guide stated within the repository.

While this setup may not align with educators due to limited knowledge in this subject, it can lead to future works which will be discussed in the later sections. Through the use of PyInstaller, it also meant that the application could be distributed with confidence via Github, knowing that the end product would be consistent across different Windows systems. Users would benefit from a simplified installation guide, receiving a self-contained executable that would be both reliable and ready to use in their educational settings.

5.4 Testing

Testing the VR monitoring application presented a unique set of challenges. Initially, I had planned to conduct thorough testing at D.C. Virgo during the summer, aiming to collect data in an environment closely aligned with the target user base. However, as summer progressed, my circumstances changed with a job offer from CGI Federal, where my role as a software engineer significantly constrained my availability. Despite my eagerness to field-test the application, my commitment to extensive training and the desire to excel in my new position meant that finding an opportunity to leave for data collection became increasingly difficult.

As the application was still in the refinement stages and no suitable testing window materialized, I sought alternative methods to gather the insights I needed. Fortuitously, an opportunity arose to collaborate with former members of the VASC team—individuals who were well-acquainted with the environment for which the application was designed. This testing also included users with little to no prior knowledge to the Oculus quest and it's functionality. Their insights promised to be as valuable as those from the intended setting.

I endeavored to set up a test on the UNCW campus, only to encounter another hurdle: network security protocols. The application's need for access to specific ports, such as port 5555, posed a security risk according to campus IT policies, thus barring me from proceeding with the tests in that context. Navigating around this roadblock, I turned to using a mobile hotspot, which provided a limited yet secure and controlled environment for testing. This workaround allowed me to finally engage with end-users in a limited fashion and collect the crucial data needed to evaluate and improve the application's performance in real-world scenarios.

5.4.1 Results

The user testing for this VR monitoring application was conducted at UNCW, where I orchestrated a comprehensive session to collect the essential data needed for evaluation. I successfully recruited seven individuals to participate in the testing with either virtual reality experience, classroom experience working with students, or both, participants without both of these experiences were excluded for testing. The way this testing was set up was by assigning a participant as the role of an instructor while the other participants acted as students within the simulated VR environment.

To guide the participants through the testing process, I provided a detailed handout that outlined the application's functionality. This served as a reference point to facilitate their interaction with the software. During the session, I employed unobtrusive observation techniques, allowing me to gather data without influencing the users' natural interactions with the application. As the participants engaged with the application, I took meticulous notes on their initial reactions and interactions. This included their responses to the UI elements like button clicks, and their ability to discern ongoing processes within the application. I paid particular attention to how intuitively they followed on-screen instructions and whether the layout and design effectively mitigated any potential confusion.

A critical aspect of the testing was assessing the ease with which the participants could monitor multiple headsets. I observed their understanding of the display arrangement and whether they could identify which student's view they were observing. The functionality for window resizing was also scrutinized; I noted whether participants opted to double-click or use the maximize button to enlarge the display window. The renaming feature was another focal point—did the participants naturally gravitate towards right-clicking, double-clicking, or typing to rename an individual headset? This insight was vital for evaluating the intuitiveness of the user interface. Additionally, I evaluated the discoverability of key functionalities like the refresh feature, which is essential for maintaining the application's currency with the headsets' statuses. Finally, the process of renaming all headsets simultaneously was tested to ensure that this bulk action was intuitive and efficient.

The comprehensive observations and data gathered during testing proved to be extremely valuable for evaluating the usability of this VR monitoring application. This process not only illuminated the application's strengths and areas for improvement but also charted a path for targeted refinements aimed at elevating the user experience. Following the hands-on walkthrough, partic-

ipants were encouraged to share their immediate impressions of the program's visual appeal and functionality through the use of a survey. They were asked to rate the ease of use, offer feedback on specific features, and suggest any additional functionalities they believed would enhance the application. This feedback is integral to the ongoing development process, ensuring that the application evolves in alignment with the real-world needs and preferences of its users.

5.4.2 Data Collection and User Feedback Analysis

The detailed survey questions that participants were invited to complete after the testing phase can be found in the chapter seven titled Appendix. During the testing phase, participants were queried about their familiarity with virtual reality and classroom environments, yielding a tripartite split in experience levels. Approximately 43% had experience in both virtual reality and classroom settings, 28% were versed in classroom experience but not VR, and the remaining 28% had VR experience with no classroom background. The gender distribution of feedback was 57% from male participants and 42% from female participants. This diverse input is crucial for the iterative improvement of the application, and I will dive deeper into these insights in the conclusion section, where I'll discuss user feedback and potential enhancements.

Referencing figure 5.12, Feedback regarding the welcome screen's initial presentation and usability was predominantly positive. A solid 71% of users awarded the screen's design and layout a perfect score, praising its clarity and user-friendliness. Meanwhile, 14% rated it a 4 out of 5, with suggestions pointing towards a preference for a 'light mode' interface for those less acquainted with technology. Another 14% gave it a 3 out of 5, recommending the addition of a help section for basic application guidance.

The connectivity process via hotspot revealed limitations due to the number of headsets that could simultaneously maintain a connection. Challenges included signal strength issues over time, necessitating reconnection efforts. 57% of participants felt neutral about the connection process, with half of these individuals lacking prior virtual reality experience, which may have influenced their perceptions. Some 14% rated the process highly, whereas 28% deemed it exceptional. Suggestions for improvement included eliminating the need to click 'allow' on each headset and introducing individual connection confirmations, which would provide immediate feedback as each headset is connected—a feature that aligns well with user expectations, despite current restrictions due to Oculus's framework and debugging requirements.

Regarding the initial naming functionality, a resounding 85% found it highly intuitive, citing its straightforward and clear execution. Functionality feedback pointed towards the potential for a 'preset class' feature, which would allow for the automatic loading of pre-defined names, underscoring the desire for even more streamlined setup processes in educational contexts.

For the visual layout of the secondary screen, which aggregates and exhibits the feeds from all connected headsets, the reception was majorly positive, with 57% of participants expressing strong approval of the content visibility. A further 28% were content with the display method, and 14% remained neutral. Despite the limitations inherent to hotspot connections, the majority were content with the display clarity of the headsets. Given a choice between the existing grid view, a "Zoom-style" gallery, or the option to toggle between views. "Zoom-style" relates to a single window maximized while the remaining headsets are displayed at the top of the window in a ordered line. The consensus leaned towards maintaining the current grid arrangement, although a minority expressed interest in having flexible viewing options.

The renaming feature elicited mixed reactions, converging on a common suggestion for improvement. In its present form, the renaming process requires a right-click to access the option from a context menu. This method was highly rated by 43% for its ease of use, while 42% rated it slightly lower but still positive, and 15% rated it as less intuitive. The predominant piece of feedback from users was the desire for a more direct renaming method, such as an adjacent button next to the headset name for a quicker, more streamlined experience.

Casting content to a tablet for testing proved to be inconclusive, as the data collection was hampered by the hotspot's limitations and insufficient signal strength to reliably conduct this part of the experiment.

Regarding the functionality to rename all headsets simultaneously, the feature was met with high approval, with 85% of participants finding it easy to use, clear, and straightforward. The feedback also highlighted the need for a clear explanation of the feature's purpose and function within the application's interface.

When participants were asked about the potential integration of this technology into a classroom setting, the response was unanimously affirmative, indicating a strong belief in the application's value for educational purposes. As for recommendations, 71% indicated they would be 'very likely' to suggest the app to colleagues or friends, while 28% were 'likely' to do so. When considering personal use at home with friends or family, the responses varied more, with 14% indicating 'very

unlikely', 14% neutral, 14% 'likely', and a majority of 57% 'very likely' to utilize the application in a home setting. These insights underscore the application's perceived utility across both educational and personal domains.

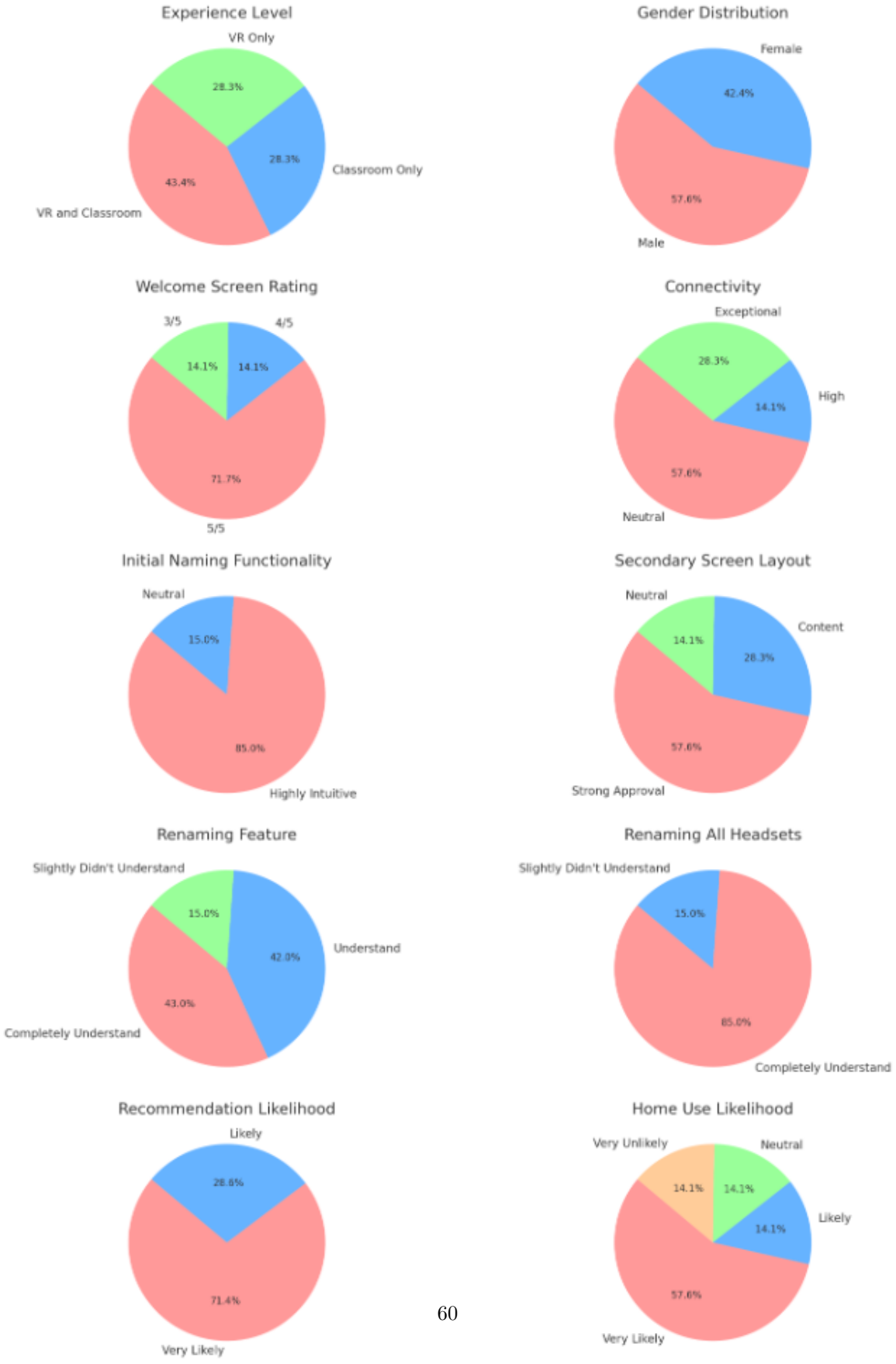

Figure 5.12: Data representation of information collected during testing.

Chapter 6

Conclusion

6.1 Revisiting Previous Predictions

In this section, the document revisits the pivotal development decisions made during the project. A significant focus is on the choice to develop a standalone desktop application rather than a web-based solution for multicasting Oculus headsets. This decision was underpinned by several factors, including the project's nature, required performance metrics, and the desired user experience. The desktop application was favored for its superior performance and reliability, especially for tasks demanding high computational resources, which were essential for managing the data-intensive nature of multicasting VR content. Another crucial aspect was the integration of Scrcpy, a screen mirroring application, which allowed seamless mirroring of VR content from the Oculus headsets to the desktop. The flexibility provided by Python scripting, with its vast libraries and easy syntax, enabled efficient handling of complex tasks. The use of Tkinter and customTkinter for GUI development facilitated a high level of customization and creation of a user-friendly interface. Additionally, the ability of desktop applications to directly access system hardware was beneficial for handling specific requirements of VR content streaming and processing. In retrospect, developing a desktop application was a strategic and effective decision, offering the necessary control and customization options required for the project's specific needs.

6.1.1 Was a Desktop Application the Right Decision?

The critical decision to develop a standalone desktop application instead of a web-based solution for multicasting Oculus headsets was driven by several key factors. The nature of the project, the required performance metrics, and the target user experience all pointed towards a desktop application as the optimal choice. Advantages of Desktop Application

- Performance and Reliability: Desktop applications generally offer better performance and reliability compared to web-based applications, especially for tasks demanding high computational resources. This was crucial for managing the data-intensive nature of multicasting VR content.
- Scrcpy Integration: The use of Scrcpy, a screen mirroring application, was pivotal. It allowed seamless mirroring of VR content from Oculus headsets to the desktop application. This integration was more straightforward and robust in a desktop environment than it would have been in a web-based setting.
- Python Scripting Flexibility: Utilizing Python for scripting provided immense flexibility. Python's vast libraries and easy syntax made it possible to handle complex tasks efficiently. The integration of Python scripts via command line arguments further enhanced the application's functionality.
- Customization with Tkinter and customTkinter: The choice of Tkinter and customTkinter for the GUI development enabled a high level of customization. This approach allowed the creation of a user-friendly interface, which was intuitive and guided the users through the application with ease.
- Direct Hardware Access: Desktop applications can directly access system hardware, which was beneficial for handling the specific requirements of VR content streaming and processing.

In retrospect, the choice to develop a desktop application proved to be a strategic and effective decision. The desktop environment offered the necessary control and customization options required for the specific needs of the project. It facilitated a more tailored user experience and allowed for the integration of specific technologies that were crucial for the project's success.

6.1.2 Was Python the correct language?

In reflecting on the choice of Python as the programming language for this project, as mentioned in Section 3.3.2, it's important to consider the alternative options that were available, such as Java or C++. Each of these languages has its own set of advantages and would have brought different aspects to the project. However, Python was ultimately chosen for several key reasons:

Advantages of Python:

- Development: Python's simplicity and readability significantly sped up the development process. Its concise syntax and high-level data structures enabled quicker implementation of complex functionalities.

- Extensive Libraries: Python offers a vast array of libraries, which facilitated various aspects of the project, from GUI development to network communication. This abundance of resources was instrumental in overcoming technical challenges efficiently.

- Ease of Learning and Use: Python's straightforward and readable syntax made it an ideal choice, especially given my familiarity. This aspect was crucial in reducing the learning curve and focusing more on development rather than language intricacies.

Considering Java and C++:

- Java: Known for its portability and robustness, Java could have offered a high level of cross-platform compatibility and performance optimization. Its strong memory management and extensive set of libraries would have been beneficial for developing a complex, networked application.

- C++: This language is renowned for its speed and control over system resources. C++ could have offered enhanced performance, particularly in handling real-time data processing and rendering, which are critical in VR applications. However, its complexity in terms of syntax and memory management might have posed a steeper learning curve and longer development times.

While Java and C++ each have their merits and could have been effective for the project, Python was the right decision in this context. Its ease of use, rapid development capabilities, and extensive libraries aligned well with the project's needs, enabling a more efficient and effective

development process. This choice reflects the importance of selecting a programming language not just based on its technical capabilities, but also on how well it fits with the developer's skills, the project's requirements, and the overall development timeline. The success of the project, in this case, validates the decision to use Python, illustrating how the right tools can greatly influence the outcome of a technological endeavor.

6.2 Component predictions revisited

In this section I would like to reevaluate the various technological components and tools chosen during the project's development phase. This introspective section is aimed at evaluating whether the initial predictions about these components held true in the practical implementation and execution of the project.

6.2.1 Android Debug Bridge

In reflecting on the use of the Android Debug Bridge (ADB) for my project, I can confidently say that choosing ADB was the right decision, and I am thoroughly satisfied with the outcome. The versatility of ADB as a command-line tool was central to the project's success, as it facilitated seamless communication between the computer and Android devices. This feature was crucial, especially considering the complex nature of the tasks involved in my project.

ADB's ability to support wireless connections was a game-changer. It enabled a range of tasks crucial for the development and management of my application. The effective implementation of ADB in establishing wireless casting to TCP/IP and connecting Android devices over Wi-Fi significantly enhanced the core functionality of my project. Integrating ADB into the technology stack of my project was a strategic move that aligned perfectly with the project's requirements. Overall, the incorporation of ADB proved to be a wise decision, underlining its value as an indispensable tool in my project's development.

6.2.2 Screen Copy

From my initial prediction for casting the Oculus displays, the focus was on the utilization of Screen Copy, commonly known as Scrcpy. Reflecting on this choice, I am not only satisfied with the decision to use Scrcpy, but also deeply impressed by its impact on the project. Scrcpy's role

was pivotal in making the viewing of all Oculus headsets possible, which was a cornerstone for the success of my project.

Scrcpy, an open-source application, enabled me to display and manipulate the display of the Android devices connected to a computer. Its compatibility with the operating system, such as Windows, proved more than efficient for the requirements of this project. The low-latency screen mirroring and responsive control offered by Scrcpy were essential for real-time monitoring and management of the Oculus headsets.

One of the most remarkable aspects of Scrcpy was its efficient and lightweight design, ensuring minimal impact on the system's performance while delivering high-quality mirroring capabilities. This feature was especially crucial in managing multiple VR headsets simultaneously, where system efficiency and stability were paramount. Additionally, Scrcpy's support for wireless connections through ADB further enhanced its flexibility, allowing me to manage the devices more effectively and efficiently.

The successful integration of Scrcpy into my project greatly facilitated the monitoring and management of multiple Oculus headsets, making it possible to achieve the project's primary goal. The reliability and efficiency it brought to the table were invaluable, and its impact was evident in the smooth operation and user-friendly experience of the application. In conclusion, the decision to incorporate Scrcpy into my project was not just a right choice but a transformative one, significantly contributing to the project's overall success.

6.2.3 Docking Device

For docking all of the headsets for an input connection, I focused on the integration and utilization of the Vangree docking device. Reflecting on this inclusion, I am extremely satisfied with its performance and the pivotal role it played in the project. The Vangree docking device was a crucial element in streamlining the connection process, providing an organized and efficient solution for connecting multiple Oculus headsets to the computer.

One of the most significant advantages of using the Vangree docking device was its ability to save on connection space. This feature was vital in a setting where multiple VR headsets needed to be connected simultaneously. By allowing more headsets to be connected in a single, consolidated space, the docking device dramatically enhanced the usability and organization of the setup. This improvement was not just a matter of convenience but also contributed to the overall effectiveness

of the monitoring and management system I was developing.

The docking device's design facilitated a more orderly arrangement of the headsets, reducing clutter and the potential for connection errors. This organization was particularly beneficial when connecting all of the headsets, where ease of use and reliability are paramount.

Moreover, the integration of the Vangree docking device into the project's technology suite allowed for a more efficient use of physical space. This aspect can be crucial in classrooms where space can be limited. It meant that more headsets could be connected and managed without requiring extensive physical infrastructure, thus potentially making the VR experience more accessible within a real educational setting.

6.2.4 Mirroring Display to tablet

The ability for mobility when using this application was the concept of extending the display monitor to a tablet, encompassing both Android and iOS operating systems. Upon reflection, I realized that I may have overestimated the feasibility of completing this feature within the given timeframe and underestimated the complexity involved in cross-platform development. This realization was a critical learning point in the project's journey.

Initially, the idea of creating a feature to display the monitor on a tablet was highly appealing. It promised to add an extra layer of flexibility and accessibility to the project, allowing educators to monitor the VR headsets more conveniently. However, the technical challenges in implementing this feature across different operating systems became evident as the project progressed. The nuances of developing for both Android and iOS platforms, each with its unique requirements and limitations, presented a significant hurdle.

In the face of these challenges, I turned to AnyViewer as a secondary option. AnyViewer, a robust and reliable application, proved to be a fitting alternative. Its compatibility with the program and adherence to the project requirements were notable. It seamlessly integrated with the existing system, offering a practical solution to the initial objective of tablet display functionality.

The successful incorporation of AnyViewer into the project not only provided a viable solution but also opened up avenues for future enhancements. The experience underscored the importance of adaptability and finding effective alternatives when faced with unforeseen technical challenges.

Going forward, the idea of a direct feature for displaying the monitor on tablets, specifically

tailored to work across Android and iOS, remains an exciting prospect for future work. This potential enhancement could further elevate the usability and reach of the project, especially in diverse educational environments where such flexibility would be highly beneficial. The groundwork laid by the current project, along with the lessons learned, will undoubtedly serve as a valuable foundation for these future developments.

6.2.5 Desktop Installer

In my prediction of my capstone project, I addressed the deployment strategy for the application, particularly focusing on the challenge of integrating a downloader application for all required libraries and dependencies. Initially, my goal was to have a fully functional downloader that would streamline the installation process by automatically handling these dependencies. However, as the project progressed, I recognized the need to pivot due to time constraints and the complexity of implementing such a solution.

In lieu of the original plan, I turned to PyInstaller as a practical and efficient alternative. PyInstaller is a well-established tool that simplifies the distribution of Python programs by bundling them into stand-alone executables. This shift in strategy proved to be not only a feasible plan but also a successful approach to deployment.

The process involved downloading the source code from GitHub, where I am hosting the project. To assist users in setting up the application, I provided detailed instructions in the ReadMe section of the repository. This approach ensured that users could successfully deploy the application by following a clear, step-by-step guide. The decision to use PyInstaller and rely on GitHub for distribution was driven by the need to prioritize core functionality and features of the application within the available time frame.

While this approach worked well, the idea of a dedicated downloader application remains a compelling aspect for future enhancements of the project. Such an application would further streamline the deployment process, making it more user-friendly and efficient, especially for users who may not be as familiar with manual installation procedures. Incorporating this feature in future work would add significant value, enhancing the overall user experience and accessibility of the application.

6.3 For future developers

This section is dedicated to guiding future developers who may work on this project or similar ones. It is divided into two subsections: "What I Would Do Differently" and "What Can Still Be Done." These insights are gleaned from the experiences and lessons learned during the course of the project.

6.3.1 What I'd do differently

Reflecting on the journey of this project, there are several areas where I would approach things differently if given another opportunity. Firstly, a broader initial exploration of technologies could have yielded significant benefits. Delving deeper into alternatives like Java or C++, or investigating various frameworks and libraries at the outset, might have uncovered more efficient pathways or innovative solutions that were not immediately apparent. This expanded research phase could have provided a more holistic view of the technological landscape and its potential applications to the project.

Another key area for improvement would be the implementation of more frequent and thorough testing phases throughout the development cycle. Incremental testing and validation could have played a crucial role in identifying and addressing issues at an earlier stage, potentially saving considerable time and resources. Regular testing phases would ensure that the project aligns closely with its objectives and functions as intended at every step.

Enhanced documentation and more detailed code commenting is another aspect I would emphasize more in future projects. While the project's documentation met basic requirements, a more comprehensive approach would greatly benefit future developers. Detailed comments within the code, along with extensive architecture diagrams, could facilitate a deeper understanding of the system's workings, making it easier for others to modify or build upon the existing foundation.

Finally, a more systematic integration of user feedback during the development process would be highly beneficial. Actively seeking and incorporating user insights could have provided valuable perspectives on user needs and preferences, leading to a design that is more user-centric and intuitive. Overall it came down to responsibilities required to make ends meet. Understanding the end-user's experience and requirements is crucial for developing a solution that not only meets technical specifications but also delivers a satisfying and efficient user experience.

These reflections are intended to guide future development work, not only on this project but also in similar technological endeavors. Learning from these experiences and adjusting strategies accordingly can significantly enhance the efficiency, effectiveness, and user satisfaction in future projects.

6.3.2 What can still be done

Although that this program is fully functional there are a few things that if I had more time I would love to implement and expand on for future developers.

- Further Optimization and Refinement: There is always room for performance optimization and refinement of features. Future developers can focus on enhancing the efficiency of the existing codebase and refining the user interface for a more seamless experience.

- Enhancements Based on Data Collection and Feedback Analysis: Insights from Section 5.4.2 indicate the need for continuous improvement in usability. Future work should focus on refining the user interface and enhancing the overall user experience based on the feedback collected. This could include optimizing the renaming functionality for monitoring the VR headsets, improving navigation and interaction elements within the application, and addressing any specific user requests or challenges identified during the feedback analysis. Some specific examples would be implementing a button next to the name of the headset for individual renaming. Another example would be implementing a quick name generation button to quickly name all of the headsets to save time.

- Cross-Platform Compatibility: Expanding the application to be compatible with more platforms, such as mobile devices or different operating systems such as MacOS and Linux, could significantly increase its accessibility and usability. This would not be limited to the expansison of viewing different virtual reality headsets such as HTC Vive as well as other virtual reality competitors.

- Collaboration with Other Developers and End-Users: Engaging with a community of developers and end-users can provide fresh perspectives and innovative ideas for the project. Collaborative development efforts can lead to more robust and user-friendly solutions.

- Mirroring display of Monitor: Without the use of AnyViewer or TeamViewer this application

would benefit from being able to cast the screen to a tablet device without relying on the use of third party software. Simply at the click of a button connect an available device and step away from the computer.

- Larger Testing population: Due to security reasons, access to ports on campus is a information technology violation and limited the number of headsets I could connect through the use of a mobile hotspot. In the future when port accessibility is available on campus, I would like the ability to test this application with at minimum of fifteen students at a time to test the limits of the application. Testing would also include testing the availability of the application due to quick internet outages as well as a full classroom change. This would be new students entering the classroom and putting on the headsets without collecting them all.

6.4 Closing thoughts

A project like this has never been done before being the first of its kind that is readily available to the public and I believe that at some point virtual reality has the possibility to provide an educational value to students. Hence enhancing the way that we learn and the way that we understand the material that is presented to us. I believe that this project is a stepping stone to the future and symbolically the way that we perceive that future.

This project represents a journey into a brand new territory. Being one of the first of its kind related to headsets available for purchase such as the Oculus Quest. It has opened the door to a new realm of possibilities at the intersection of virtual reality (VR) and education where the teachers are in control compared to VR Education companies. The uniqueness of this endeavor cannot be overstated – it's not just a technological achievement for myself, but a pioneering step towards an innovative future in learning and interaction.

The potential of virtual reality to revolutionize education is a factor of time. This project has laid the foundation for how VR can enhance the learning experience for educators to monitor, providing immersive, interactive environments for students that transcend traditional classroom boundaries. It's about more than just presenting information; it's about transforming how students engage with and understand complex concepts for teachers to provide answers to. By bringing subjects to life in a virtual space, we can cater to diverse learning styles, making education more accessible, engaging, and effective.

Beyond its practical applications, this project symbolizes how we perceive and shape our future. It reflects a shift in thinking, from what is known and conventional to what is possible and innovative. Virtual reality in education is not just about using new technology; it's about re-imagining the learning experience, pushing boundaries, and exploring new opportunities to the way that we know, the way that we remember, and the way that we can explain.

Chapter 7

Appendix

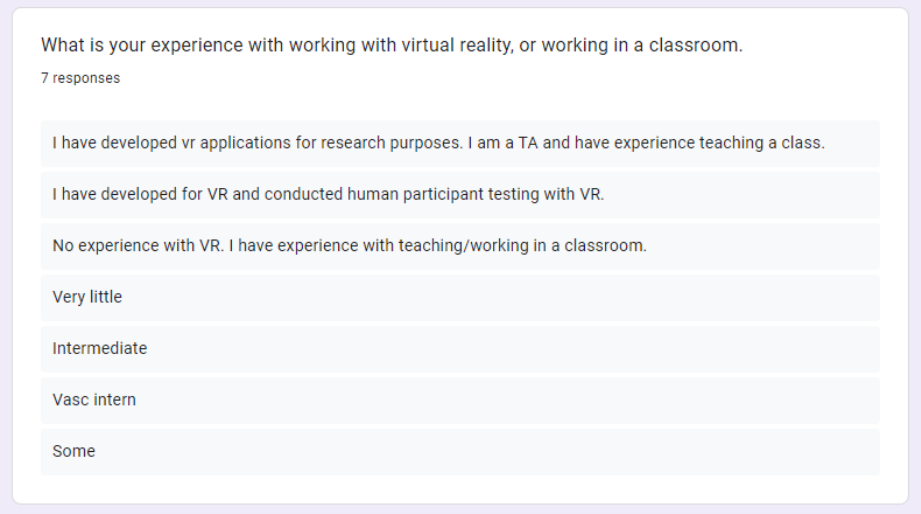

Figure 7.1: Obtaining the experience level for virtual reality experience, classroom experience or both.

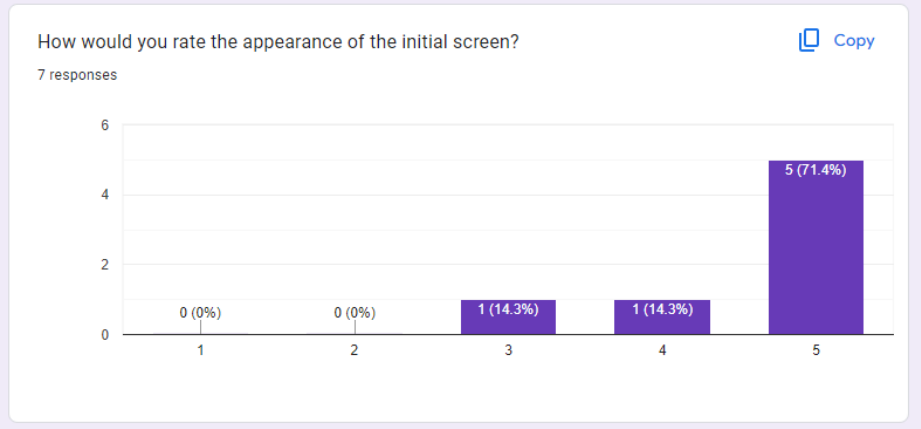

Figure 7.2: Obtaining data based on the visual appeal of the initial welcome screen.

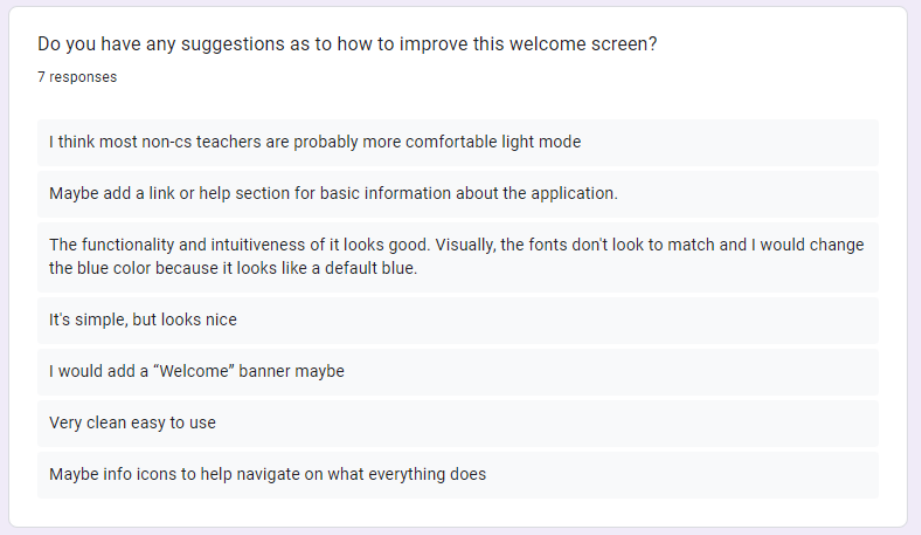

Figure 7.3: Obtaining feedback on how to improve the welcome screen.

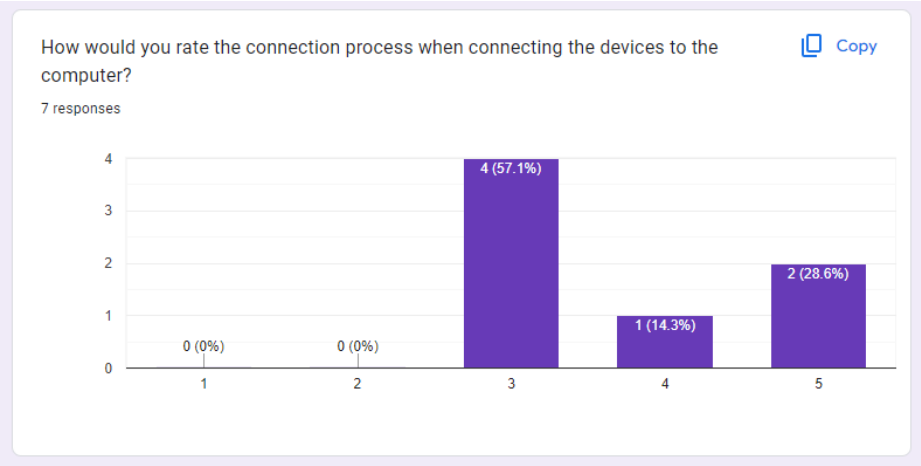

Figure 7.4: Obtaining data related to the difficulty of the device connection process.

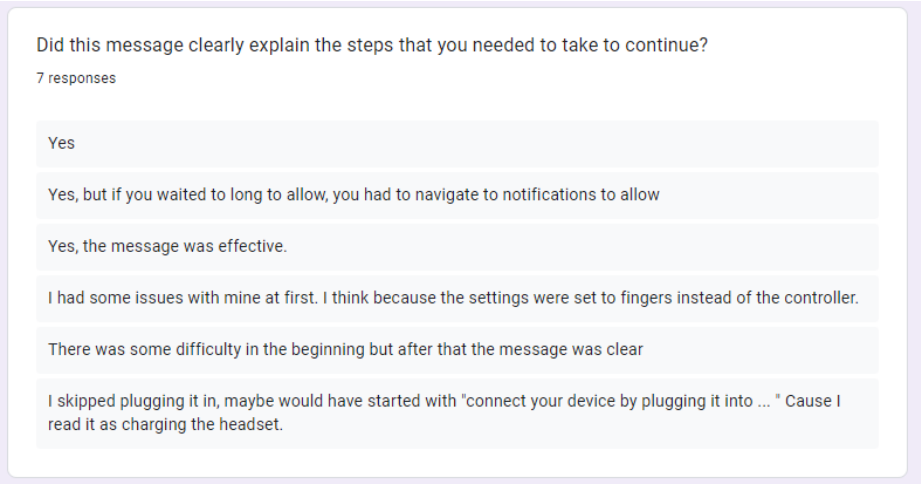

Figure 7.5: Obtaining Feedback to the level of clarity the display message presented for the connecting devices.

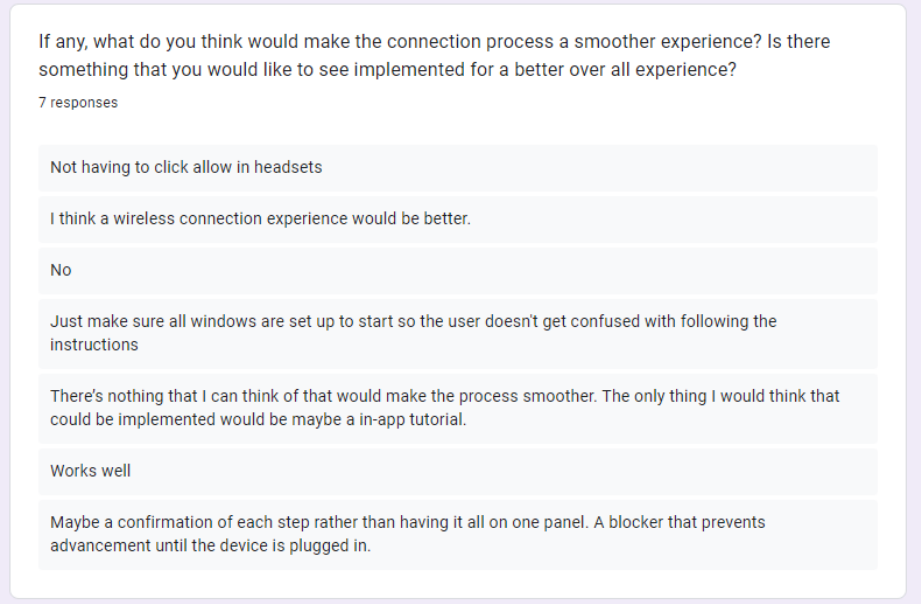

Figure 7.6: Feed back on what would make the device connection process simpler.

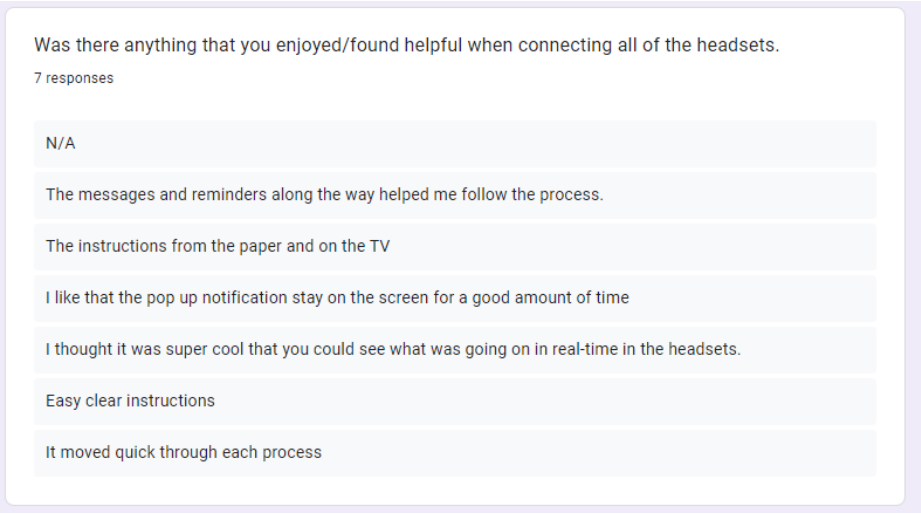

Figure 7.7: Feedback on what the user enjoyed or disliked when connecting all of the headsets.

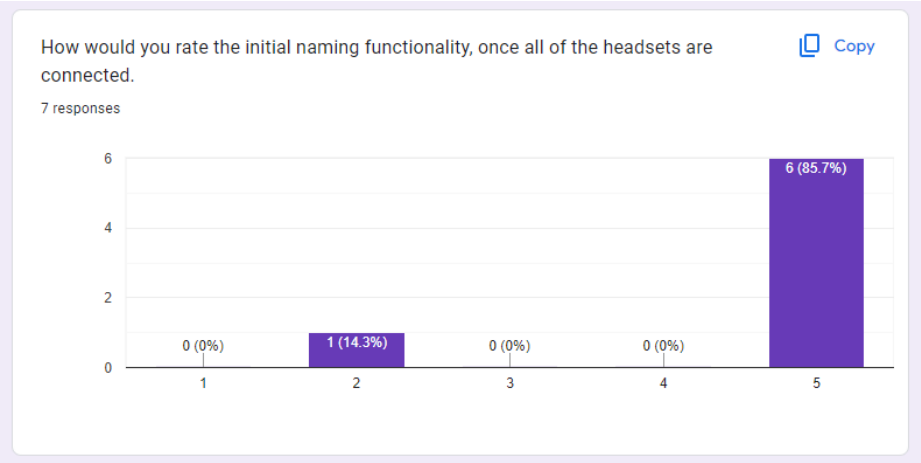

Figure 7.8: Rating on the initial naming functionality one all headsets are connected.

Figure 7.9: Feedback determining what the user liked or disliked about the naming feature for future improvement.

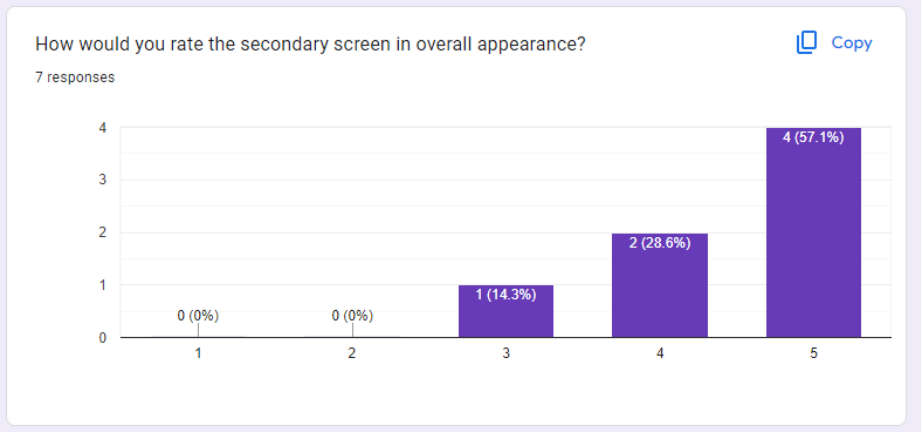

Figure 7.10: Rating on the overall appearance of the secondary screen showcasing all of the headsets.

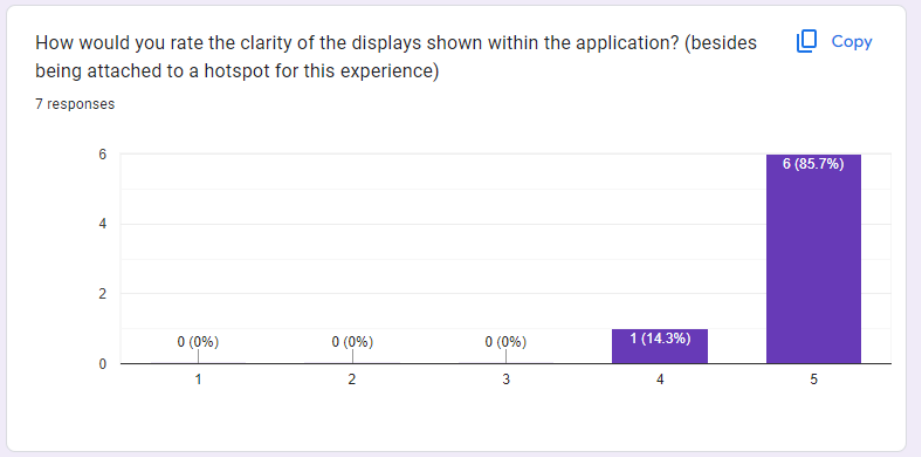

Figure 7.11: Rating the over all clarity of the headset displays visible to the user.

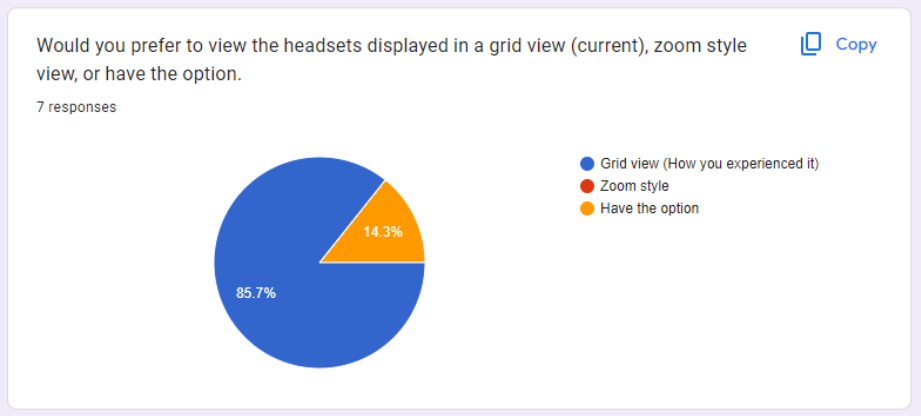

Figure 7.12: Chart asking the user the preferred view style that they would like to see when observing all headsets.

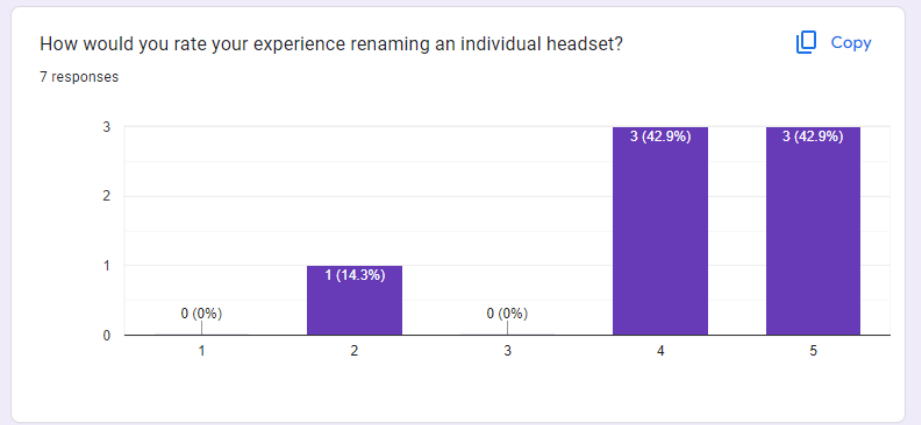

Figure 7.13: Rating the experience of renaming an individual headset.

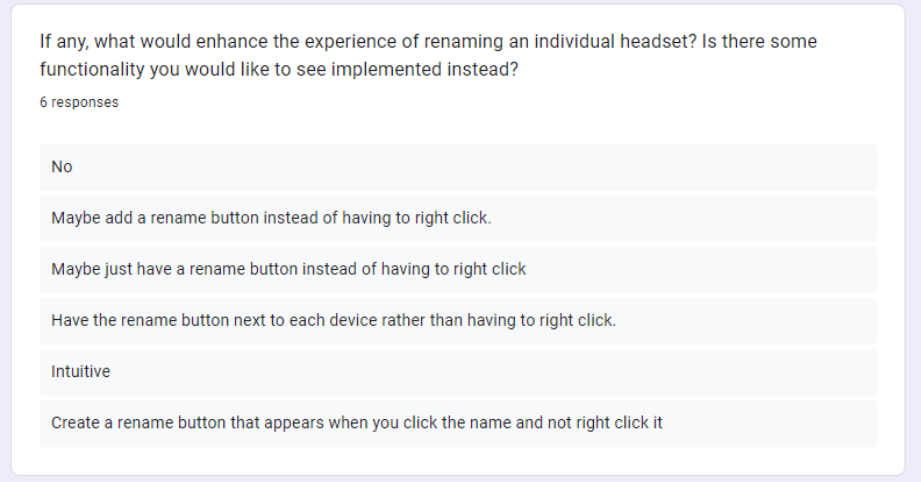

Figure 7.14: Feedback regarding the renaming of an individual headset functionality.

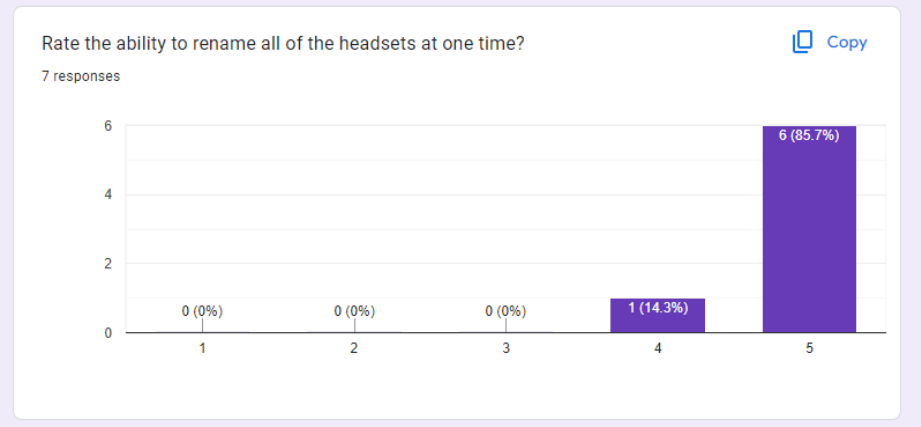

Figure 7.15: Rating the experience of renaming all of the headsets at one time.

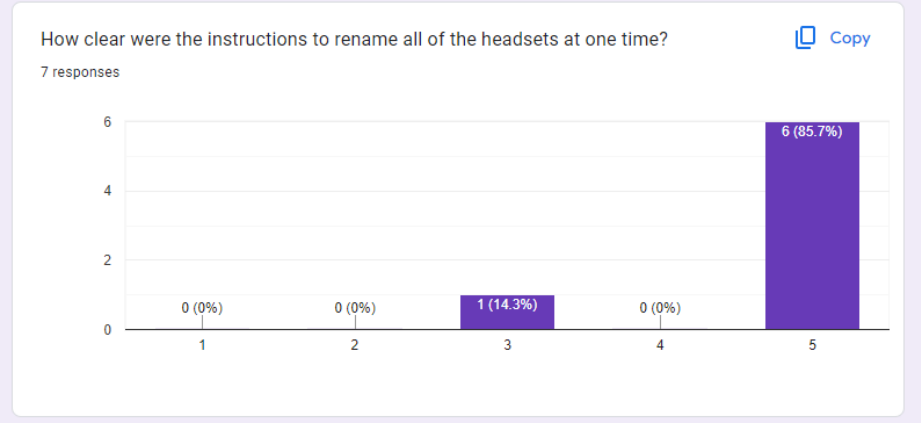

Figure 7.16: Rating how clear the instructions were to rename all of the headsets at one time.

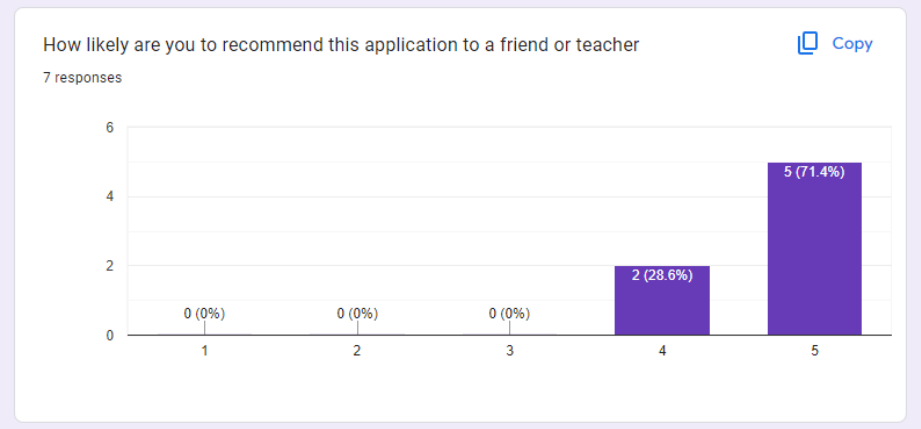

Figure 7.17: Users asked the likely hood they would recommend to a friend or teacher.

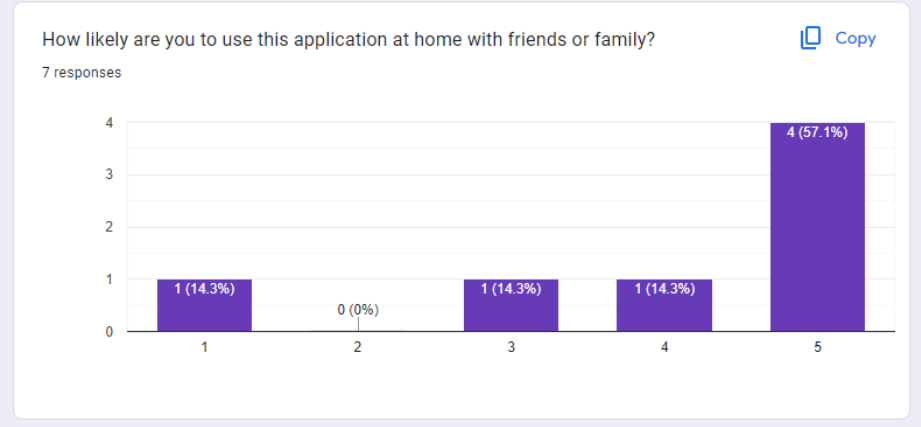

Figure 7.18: Users asked the likely hood they would use this application at home with friends or family.

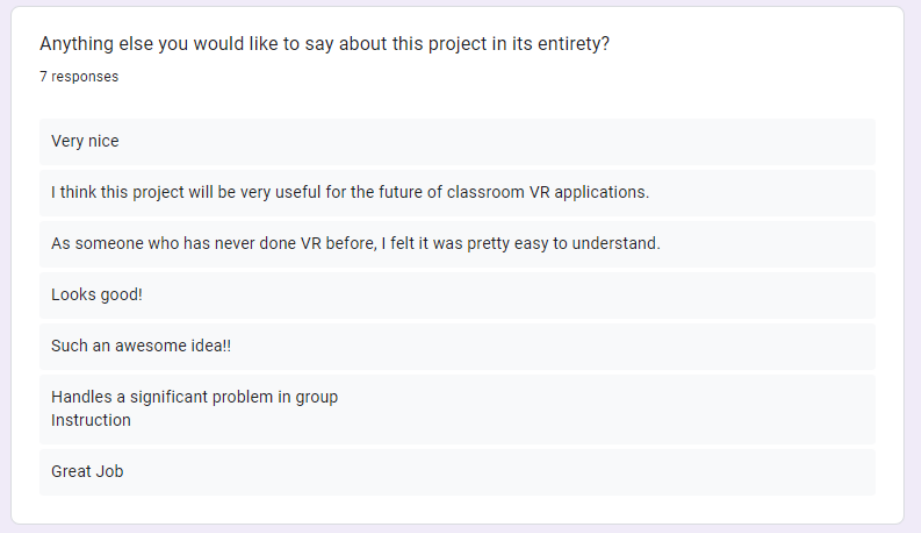

Figure 7.19: Overall Feedback of the entirety of the application

Bibliography

[any,] Anyviewer — a powerful remote desktop connection software. `https://www.anyviewer.com/company.html`. Accessed: November 18, 2023.

[pyi,] What pyinstaller does and how it does it. `https://www.pyinstaller.org/en/stable/operating-mode.html`. Accessed: November 18, 2023.

[Android, 2023] Android (2023). Android debug bridge (adb). `https://developer.android.com/tools/adb`. [Online; accessed 5-May-2023].

[Angell, 2022] Angell, S. (2022). Web based session monitoring: A unity3d framework proposal.

[Bouras et al., 2014] Bouras, C., Hornig, H., and Tsiatsos, T. (2014). Multicast group communication for multicast and unicast virtual reality applications. *Multimedia Tools and Applications*, 71(2):683–716.

[Burdea and Coiffet, 2003] Burdea, G. C. and Coiffet, P. (2003). *Virtual Reality Technology*. Wiley-IEEE Press, 2 edition.

[CE-IT, 2021] CE-IT (2021). Desktop app vs web app.

[Chaetognathan, 2023] Chaetognathan, P. (2023). Which python gui library should you use in 2023?

[ClassVR, 2023] ClassVR (2023). How classvr works in the classroom. `https://www.classvr.com/how-classvr-works/`. [Online; accessed 5-May-2023].

[Davidson, 2022] Davidson, T. (2022). Native app vs. web app: Are native apps still relevant? — clean commit.

[Fernandes and Feiner, 2016] Fernandes, A. S. and Feiner, S. K. (2016). Combating vr sickness through subtle dynamic field-of-view modification. In *2016 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 201–210. IEEE.

[Green, 2020] Green, K. (2020). Which is your dominant eye, and why do we have one? `https://www.optimax.co.uk/blog/which-is-your-dominant-eye/`.

[Greenhalgh, 2001] Greenhalgh, C. (2001). Co-presence and interaction in a collaborative virtual environment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4):339–364.

[Heilig, 1962] Heilig, M. L. (1962). Sensorama simulator. US Patent 3,050,870.

[JavaTpoint, 2021] JavaTpoint (2021). C vs c++ vs python vs java.

[Jerald, 2016] Jerald, J. (2016). *The VR book: Human-centered design for virtual reality*. Morgan & Claypool.

[Kluge et al., 2023] Kluge, M. G., Maltby, S., Kuhne, C., and et al. (2023). Comparing approaches for selection, development, and deployment of extended reality (xr) teaching applications: A case study at the university of newcastle australia. *Education and Information Technologies*, 28:4531–4562.

[LaValle, 2016] LaValle, S. M. (2016). *Virtual reality*. Cambridge University Press.

[LaViola, 2000] LaViola, J. J. (2000). A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin*, 32(1):47–56.

[Malik et al., 2018] Malik, A., Bilal, A., Khushnood, A., Cheema, W., and Hassan, Z. (2018). A comparison of networked and standalone virtual reality-based training for teaching mechanical assembly. In *2018 IEEE Technology Engineering Management Conference (TEMSCON)*, pages 1–7. IEEE.

[Mansor et al., 2022] Mansor, E., MOKHTAR, M. M., and SHAREF, N. M. (2022). Exploring the acceptance of 360 videos in virtual reality settings for teaching and learning. *Asia-Pacific Journal of Information Technology and Multimedia*, 11(1):78–89.

[Nepal, 2021] Nepal, W. H. (2021). Difference between desktop application and web application. [Online; accessed 5-May-2023].

[NetworkInterview, 2021] NetworkInterview (2021). Programming languages for gui development. `https://networkinterview.com/programming-languages-for-gui-development/`. [Online; accessed 5-May-2023].

[Positiwise, 2021] Positiwise (2021). Web application vs desktop application: Pros and cons. `https://positiwise.com/blog/web-application-vs-desktop-application-pros-and-cons`. [Online; accessed 5-May-2023].

[Quest, 2023] Quest, M. (2023). Cast to a screen with meta quest. `https://www.meta.com/help/quest/articles/in-vr-experiences/oculus-features/cast-with-quest-2/`. [Online; accessed 5-May-2023].

[Rheingold, 1991] Rheingold, H. (1991). *Virtual Reality*. Summit Books.

[Schimansky,] Schimansky, T. Customtkinter: A modern and customizable python ui-library based on tkinter.

[Schroeder, 2011] Schroeder, R. (2011). *Being there together: Social interaction in virtual environments*. Oxford University Press.

[Singh and Prabhakaran, 2017] Singh, A. and Prabhakaran, B. (2017). A survey on multicast protocols for 3d tele-immersive environments. *Multimedia Systems*, 23(6):647–681.

[Software, 2022] Software, O. B. (2022). Obs studio. `https://obsproject.com/`. [Online; accessed 5-May-2023].

[Sutherland, 1968] Sutherland, I. E. (1968). A head-mounted three-dimensional display. In *AFIPS Conference Proceedings*.

[Venkatraman et al., 2014] Venkatraman, K., Vellingiri, S., Prabhakaran, B., and Nguyen, N. (2014). Mpeg media transport (mmt) for 3d tele-immersion systems.

[Vimont, 2022a] Vimont, R. (2022a). scrcpy. [Online; accessed 5-May-2023].

[Vimont, 2022b] Vimont, R. (2022b). scrcpy performance. [Online; accessed 5-May-2023].

[Vimont, 2022c] Vimont, R. (2022c). scrcpy settings. [Online; accessed 5-May-2023].

[Vimont, 2023] Vimont, R. (2023). Scrcpy — download (latest version).

[VRDesktop, 2022] VRDesktop (2022). Virtual desktop. [Online; accessed 5-May-2023].

[VRSpace, 2021] VRSpace (2021). Which popular vr headsets have generated the most revenue? [Online; accessed 5-May-2023].

[Waraich, 2023] Waraich, K. (2023). Code studio. `https://www.codingninjas.com/codestudio/library/java-vs-python-and-c#`. [Online; accessed 5-May-2023].

[Wikipedia contributors, 2023a] Wikipedia contributors (2023a). Multicast — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2023].

[Wikipedia contributors, 2023b] Wikipedia contributors (2023b). Tethering — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2023].

[Wikipedia contributors, 2023c] Wikipedia contributors (2023c). Virtual reality headset — Wikipedia, the free encyclopedia. [Online; accessed 5-May-2023].