Spring 2021

Interactive Virtual Reality Reading Experience

Nathaniel Shetler
nds39@zips.uakron.edu

Nathaniel Shetler

Williams Honors College

Abstract

The project will be an interactive virtual reality reading experience. The user will be able to read a book or story in VR. When certain achievements are reached, such as finishing a chapter, the user will be given the opportunity to transport to the environment that they are reading about. This will give the user a great opportunity to interact and learn hands-on with the material they are reading about. For example, if the user is reading about World War I, they will be given the opportunity to transport to the battlefields or trenches in Europe.

The project will be created for the Oculus Quest platform with support for full hand tracking. It will be an example of what types of virtual reality learning experiences can be created inside of the Unity3D engine. The documentation of the project will give readers insight into how to put their own stories or history sources into virtual reality in both book and full three dimensional environment form.

Table of Contents

Chapter I

Introduction

This project will examine the viability of virtual reality as an educational tool through the creation of an interactive virtual reality reading experience. It will provide users with a virtual reality environment to read books in, as well as other environments that are tied to the reading material. When reading a book, if the user reaches certain achievements, such as finishing a chapter, they will be able to transport to the environment or world that they are reading about in the book. This will allow the user to learn hands-on about the material that they are reading about. This could be beneficial in educational subjects such as history, allowing users to experience first-hand, the historical environments or events that they are studying. One group that could benefit from this project is those who struggle with reading, whether that be because they struggle to pay attention, or because they simply find it boring. It will give users goals to aim for and rewards to earn. The goals being finishing chapters, and the rewards being unlocking transportation to new environments by finishing chapters. The project will be created for the Oculus Quest platform using the Unity3D engine.

Chapter II

Background

Project Background

The idea for the project arose during Dr. Xiao's Human Computer Interaction (HCI) course. Throughout the course, we explored the different ways users can interact with computers and the different ways that programmers can use these interaction techniques in their own work. During the course, we discussed the uses and benefits of virtual reality. Virtual reality allows for an incredibly immersive experience in regards to interaction with the computer and program. In addition, we learned about hand tracking technology, specifically Leap Motion. This technology allows the user to control programs with simply their own hands. In the beginning stages of planning the honors project, Dr. Xiao suggested that I should combine both VR and hand tracking in order to make the experience as immersive as possible. This led to the final idea for the Interactive Virtual Reality Reading Experience project and its design.

Virtual Reality and Its Uses

Virtual Reality is a growing form of media that has experienced a surge in popularity over the past decade. It is a fully immersive experience that transports the user into a virtual environment. Since 2010, platforms such as Oculus, HTC VIVE, and more recently the Valve Index, have popularized the use of VR as a gaming experience. Typically these headsets were expensive and were required to be attached to a powerful computer in order to run the VR headset and games for them, which further increased the price of admission into experiencing Virtual Reality. However, in recent times the price for entry has been decreasing. Companies have been finding new ways to make Virtual Reality more widely available. Companies began creating devices to turn smartphones into partly functional VR headsets. Following this, Facebook, who currently owns Oculus, decided to create their own dedicated portable VR headset called the Oculus Go that debuted in 2018.[1] Both the smartphone devices and Oculus Go were rather limited when it came to VR experiences in comparison to their PC powered counterparts, mainly due to the lack of computing power that each type of device had. Where these devices lacked in power, they benefited from being wireless, stand alone headsets that required no powerful computer nor any wires tethering them down. Facebook recognized the benefit of being a stand alone, wireless system, but also recognized the prior lack of power that limited such devices. This inspired them to create and release the Oculus Quest.[2]

The Oculus Quest is a stand alone, wireless system that is more powerful than the Oculus Go that came before it. This allowed the device to play games and run experiences that were previously relegated to only PC powered VR systems. The Quest also introduced full hand tracking technology. This allows users to control certain applications with only their hands, negating the need for controllers. Not only is the Quest used for gaming, but also for business. Oculus aims to use the Quest, and the recently released Quest 2[3], to aid business in things such as meetings, training, and the creation of new virtual work sources.[4] Another area where Virtual Reality is becoming more common is in education. One form of education that is currently being used that is related to this project, is virtual expeditions and experiences. Google is using VR and AR (Augmented Reality) to allow students to view educational items such as globes and

[1] https://www.oculus.com/go/?locale=en_US
[2] https://www.oculus.com/quest/
[3] https://www.oculus.com/quest-2/?locale=en_US
[4] https://business.oculus.com/?locale=en_US

dinosaurs in a virtual space, and also for virtual field trips.[5] While VR is currently used in some educational settings, the possible uses for it are still being discovered.

The Unity Engine

Unity is a development platform that allows users to create applications for nearly every popular platform available. Currently, it is primarily used as an engine to create video games, but is also being used in other fields as well. Some popular games created using the Unity Engine include Hollow Knight, Escape from Tarkov, Cuphead, and many more. The other fields it is being used in include architecture, engineering, and construction to aid users in their workflow[6], film, animation, and cinematics to helps users create their work[7], and in the automotive, transportation, and manufacturing industries to design and create products.[8] The Unity Engine is a valuable source for users who want to create innovative products and applications.

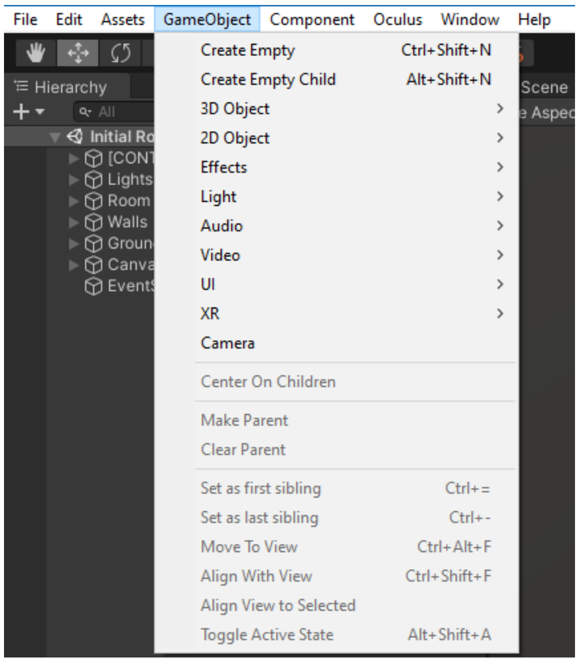
Unity is an incredibly useful tool for creating Virtual Reality applications. If using an Oculus headset as the platform, Oculus and Unity have documentation available that assist users with setting up projects, using the resources provided by Unity, and writing the C# code needed for applications. While the process for creating applications can be frustrating at times, overall, it is a streamlined experience and is why many developers, including I, choose Unity for creating VR applications for the Oculus platform.

[5] https://edu.google.com/products/vr-ar/expeditions/#about
[6] https://unity.com/solutions/architecture-engineering-construction
[7] https://unity.com/solutions/film-animation-cinematics
[8] https://unity.com/solutions/automotive-transportation-manufacturing

Chapter III

Design

The Project was designed using the Unity3D engine and the C# programming language that goes alongside it. The Unity3D editor aids tremendously in the design and development process. The Unity3D editor provides users with a variety of tools to aid in the creation process. These include physics, 2D objects, 3D objects, lighting effects, graphical effects, UI, and various others.



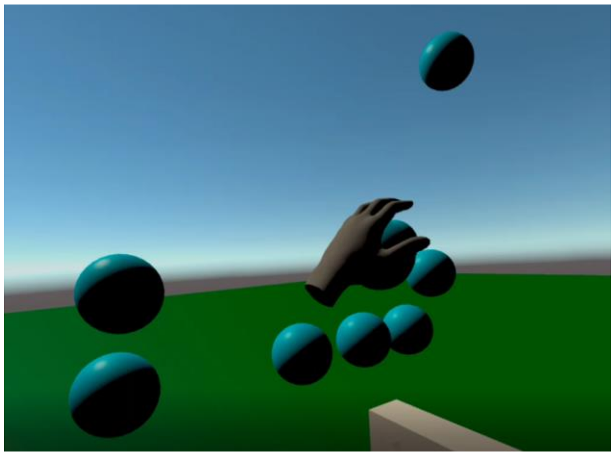
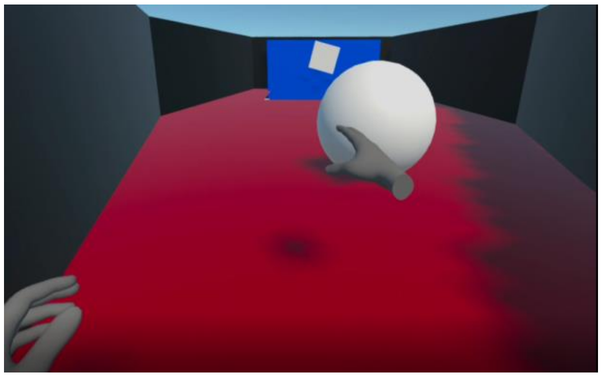
This is an image of the Unity3D menu where users can see the items provided by the Unity3D editor.

These tools can be used and built upon to create items for the users specific project and idea. For 3D objects, basic items such as cubes, spheres, cylinders, planes, and others can be used to create new items. In addition, Unity3D allows users to import their own 3D objects that they have created or choose already created assets from the Unity Asset Store. For this project, objects used were either created using Unity3D's built in objects or obtained for free from the Unity Asset Store.

The Oculus Quest and Hand Tracking Controls

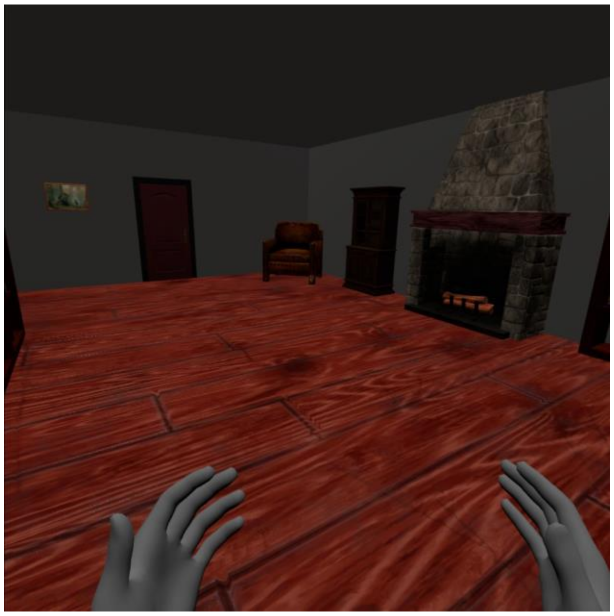
This section will describe the design details for the project, for more in depth descriptions on implementation details, please see the next chapter, Chapter IV. The Oculus Quest was chosen as the target Virtual Reality headset for the project because at the time, it was one of the most powerful all in one (meaning it requires no additional hardware, such as a PC) VR headsets available on the market. However, since the start of the project, the Oculus Quest 2 was released and is slightly more powerful. In addition, the Oculus Quest has built in hand tracking technology that allows users to control items in VR with only their hands. Users can simply put on the headset and not worry about holding anything else. They control everything with only their hands and gestures created by moving their hands and fingers. The combination of being a powerful all in one headset with hand tracking capabilities allow for ease of portability, ease of use, and less complication overall. In an educational environment, educators would not need to worry about any wires that get in the way or start any software on a separate computer, but rather they only need to have students put the headset on to start experiencing Virtual Reality.

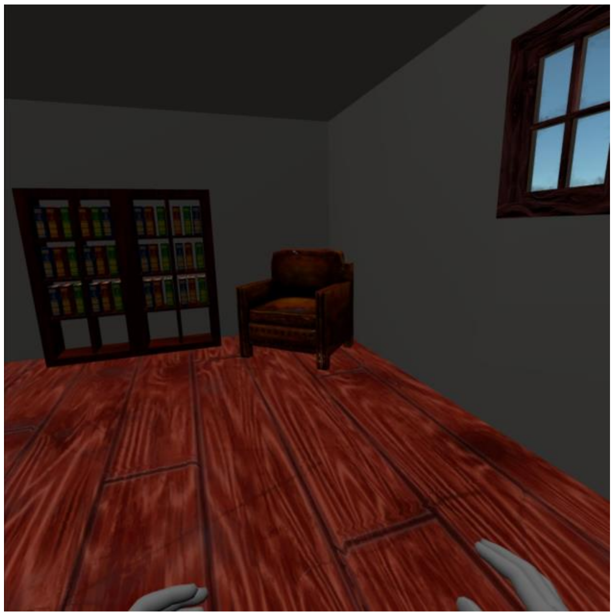
The design process started with two test projects in Unity3D. The first of which was a simple application that was used to test out the headset and hand tracking. It was simple, the only things in the project were the users hands and a number of floating balls that the user can hit around. The second project was the recreation of a program previously created in Dr. Xiao's HCI course. It was initially built to use Leap Motion hand tracking and output onto a standard computer monitor, but was now using Oculus Quest's hand tracking and outputting on the headset. It is a simple game in which users push a ball up a slope into a hole. Both of these basic projects were simply proof of functionality for the Oculus Quest headset within a Unity3D project. Following these two basic projects, design for the actual project began.



These images are from the two programs used to test the functionality of the Oculus Quest within Unity3D.

The Initial Room

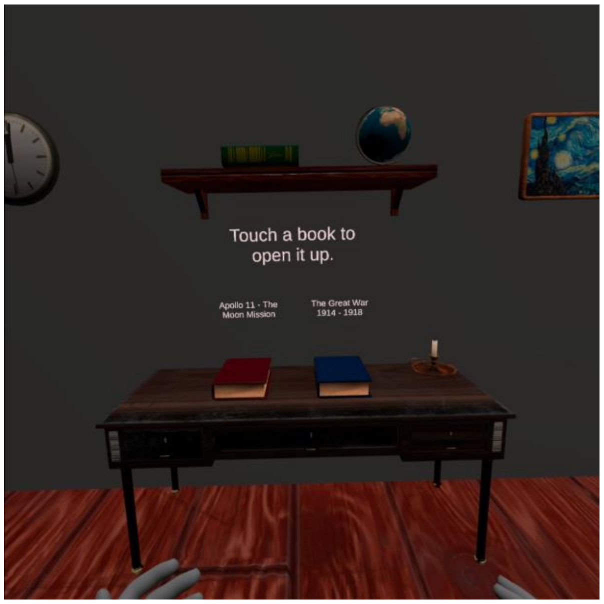
When designing the initial room that the user of the application starts in, a sense of calm and cozy was desired. In order to achieve this, a small room that is similar to a study was created. For the structure of the room, Unity3D cube objects were used and converted into wall-like shapes. There were also a window and door inserted into the room to make it look more appealing and less like a dungeon. For the floor, a hardwood texture was used. A fireplace was also added, with it being one of the centerpieces of the space. The room was furnished with bookshelves filled with books, chairs, small paintings, shelving, a table, a clock, a globe, and a few other items. These items fill the room out more to make it look less bare. Each item in the room has physics built in to it, so if the user wants to interact with the item, they are able to.





These two images are of the three walls behind where the user will actually be looking most of the time.

The main area in which the user will be interacting with the application is relatively small with the focus being on a table and the items on it. On the table are a few books. There are decorations surrounding the table to furnish the environment, but the books are the main focus. The books on the table are the ones that the user can pick up and read. The directions on how to do so, as well as the titles of the book, are present and in bright white lettering so that the user can clearly see them. These act as the main directions for the user and are simple to follow. The directions inform the user that way that they can pick up the book is by simply touching it. Initially, the user would have to physically lift the book, but this was not precise and difficult to do consistently. A more streamlined approach was necessary in order for the user to have a more enjoyable experience. This is what led to the current approach being used, meaning that the user only has to touch the book and it will open up for them.

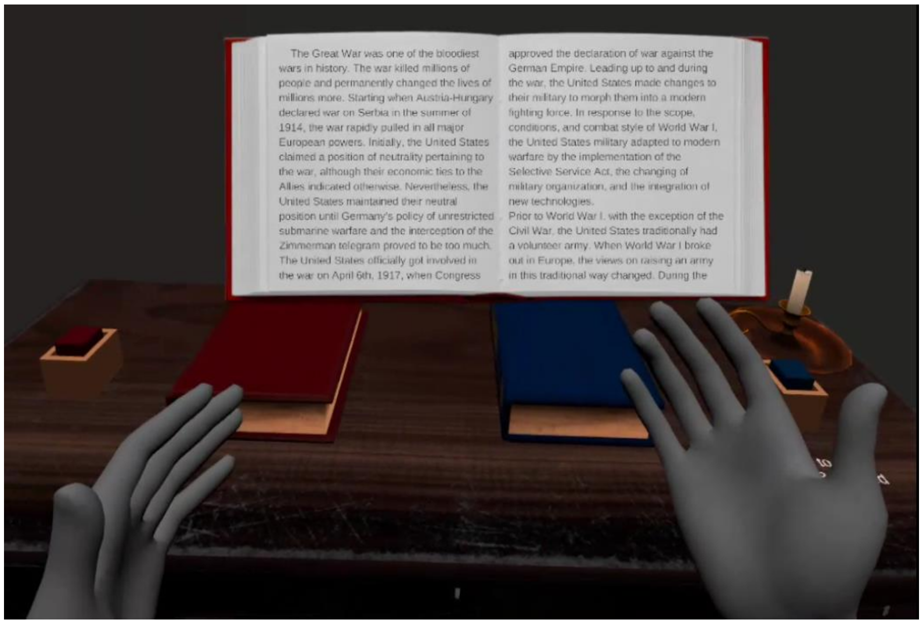

Touch a book to open it up.
Apollo 11 - The Moon Mission
The Great War 1914 - 1918

The table and interaction directions in the initial room.

The Reading Environment and Its UI

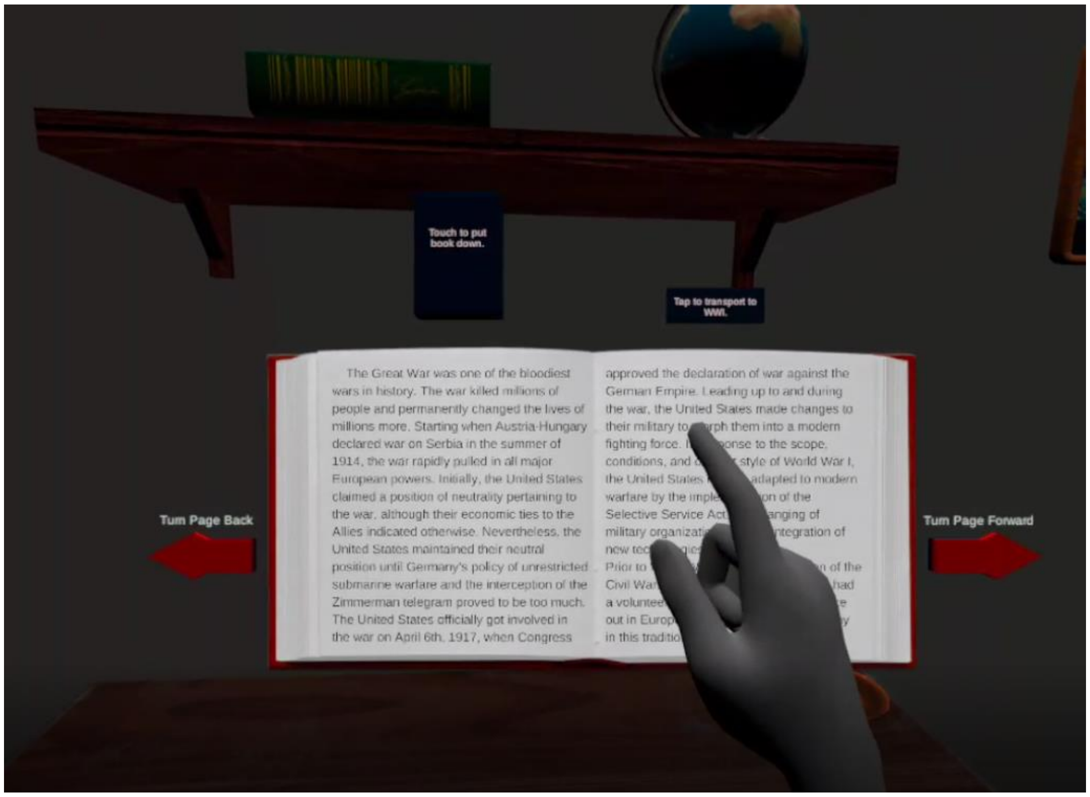
Once the user touches the book, it will open up and attach itself to the center of the user's vision. Wherever the user looks, the book will follow. In addition to the open book, there are additional pieces of UI for the user. On each side of the book, an arrow is present that if pressed, changes the page. The arrows are labeled with directions to touch them to turn the page. In addition, there is a floating “book down” button that appears above the open book to the left that will close the book when the user touches it. It is clearly labeled. The final part of the book reading UI is the button that the user presses to transports the user to the environment that the reading material is about. This button is only visible when a certain page number is reached. In the case of the project, the environment traveled to is a battlefield from World War I. When visible, the transport button is located above the open book to the right and is also clearly labeled. All of these elements of the UI travel wherever the user goes within the virtual environment. Every item that the user needs to interact with the application is present right in front of the user and at their fingertips. While reading the book, calming music is played in the background.

The UI for the reading environment did not start this way and went through various changes. Initially, to put down the book, the user would have to touch the same book on the table that they previously touched to open it up. This was not as intuitive, as the open book covered up some of the user's vision space, meaning that it was harder to find and see the book on the desk that they would have to touch to close the open book in front of them. In addition, this tied them down to a certain space in the virtual environment. With the new UI, the user is able to walk anywhere within the virtual environment to read and is able to close the book from anywhere as well. Additionally, the same issue was present for the button the user presses to transport to the environment that they are reading about. Initially, the button to transport to the environment that the user is reading about was placed on the table next to the book the user used to have to touch to close the open book. This had many of the same issues as the old close book functionality and was replaced for the button that is present in the UI that travels with the user.



The Great War was one of the bloodiest wars in history. The war killed millions of people and permanently changed the lives of millions more. Starting when Austria-Hungary declared war on Serbia in the summer of 1914, the war rapidly pulled in all major European powers. Initially, the United States claimed a position of neutrality pertaining to the war, although their economic ties to the Allies indicated otherwise. Nevertheless, the United States maintained their neutral position until Germany's policy of unrestricted submarine warfare and the interception of the Zimmerman telegram proved to be too much. The United States officially got involved in the war on April 6th, 1917, when Congress approved the declaration of war against the German Empire. Leading up to and during the war, the United States made changes to their military to morph them into a modern fighting force. In response to the scope, conditions, and combat style of World War I, the United States military adapted to modern warfare by the implementation of the Selective Service Act, the changing of military organization, and the integration of new technologies.
Prior to World War I, with the exception of the Civil War, the United States traditionally had a volunteer army. When World War I broke out in Europe, the views on raising an army in this traditional way changed. During the

Old UI with the "book down button" being on the table as well as the transport button being on the table as well.



The Great War was one of the bloodiest wars in history. The war killed millions of people and permanently changed the lives of millions more. Starting when Austria-Hungary declared war on Serbia in the summer of 1914, the war rapidly pulled in all major European powers. Initially, the United States claimed a position of neutrality pertaining to the war, although their economic ties to the Allies indicated otherwise. Nevertheless, the United States maintained their neutral position until Germany's policy of unrestricted submarine warfare and the interception of the Zimmerman telegram proved to be too much. The United States officially got involved in the war on April 6th, 1917, when Congress

approved the declaration of war against the German Empire. Leading up to and during the war, the United States made changes to their military to [illegible]rph them into a modern fighting force. I[illegible]ponse to the scope, conditions, and [illegible] style of World War I, the United States [illegible] adapted to modern warfare by the imple[illegible]on of the Selective Service Act[illegible]anging of military organizati[illegible]ntegration of new tec[illegible]
Prior to [illegible]n of the Civil War [illegible] had a voluntee[illegible] out in Europ[illegible] in this traditio[illegible]

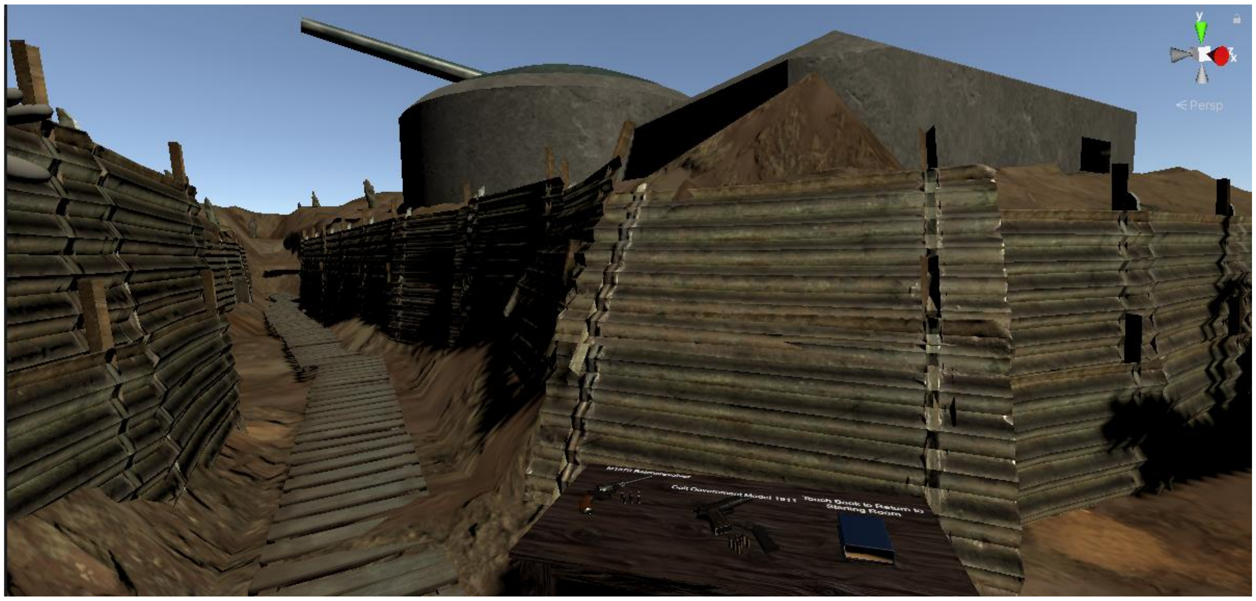
The new UI in which everything travels with the user's vision.

The reasoning for the change in UI was testing. After much testing done by myself, I became annoyed at having to go back to the table to put the book down or travel to the reading material environment. The open book obscured vision and the location of the buttons anchored one to the table area within the virtual environment. Other users also tested out the old UI and agreed that it needed to change. After completing the new UI, myself and other users were more satisfied with the experience. In addition, the new UI is, unsurprisingly, faster to access for the users.

Overall, the current reading environment and its UI allows the user to read the book anywhere they want and perform any action necessary in an easy, simple way.

The Reading Material Environments

After the user presses the button to transport them to the environment that they are reading about, the environment will appear. In the case of this project, the environment that the user can transport to is a World War I battlefield. The purpose of being able to transport to the environment that the user is reading about is to see first hand the material that is being described in a book or story. This allows users to experience history or even fictional environments (if implemented) in an immersive manner that can aid in the learning experience. In creating the World War I Battlefield, free assets from the Unity Asset store were used. The assets included a basic battlefield layout that was used as the base for the environment. Other items such as tree stumps, trench defenses and decorations, artillery, and firearms were to make the environment more immersive and appear like an actual trench. For the time being, the user can look around and explore the battlefield and trenches if they choose. In the environment, battle music, which also came from the Unity store, plays to further immerse the user. A battlefield was chosen as the environment to transport to because the reading material details how World War I changed the United States military, and the battlefields were a large contributor to these changes. Being so, the ability to see the battlefields that sparked the changes seemed desirable.


Persp
Colt Government Model 1911
Touch Book to Return to Starting Room

This is an image of the battlefield that users can transport to.

Overall, the project was designed to be an easy to follow experience in which the user is initially put into a comfortable environment where they can open up a book and read it in an intuitive manner and then can transport to an environment that correlates with the material that they are reading about when they reach a certain point in the book or story (At the current time, the ability to transport to the new environment is always present). The ways in which the user interacts with the UI went through changes sparked by much testing from myself and other users. In all, the experience is designed to allow users to read and experience history or fiction in an immersive, easy to understand manner.

Chapter IV

Implementation

When implementing the design choices for this project, previous Unity and C# knowledge was used in addition to documentation provided by Oculus and Unity. Before getting started on the actual project, the first step was to set up the Oculus Quest for development in Unity3D.

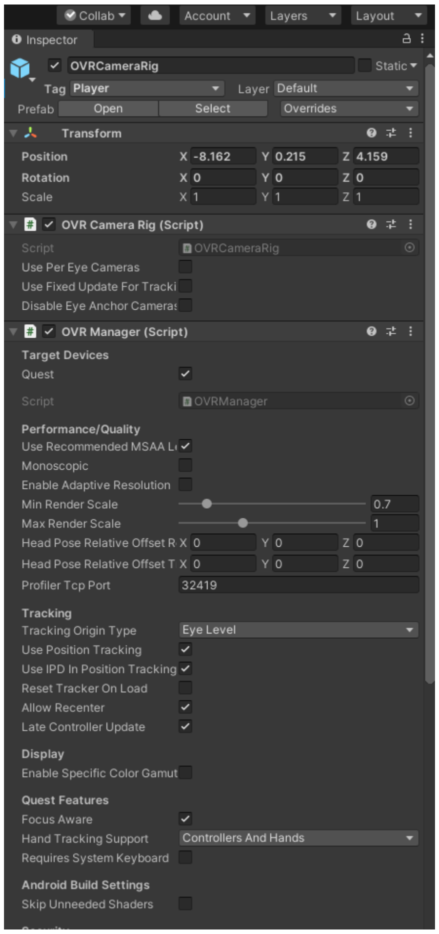
Oculus Quest and Unity3D Setup

To accomplish the setup, documentation provided by Oculus was used.[9] The following is a brief summary of the process followed using the documentation. For full instructions, visit the documentation links cited. The first step in the setup was to install the Unity3D editor on the PC where the development will be done. The version of Unity used for this project is 2019.4.9f1, which at the time was the most current stable version of Unity3D. After installing the Unity3D editor, the next step was to create a new project. Following this, enabling the Oculus Quest for development and testing was required. To do this, I had to install the Oculus App on my smartphone. Once installed, I had to connect the app to the Oculus Quest headset. On the app, a selection for "Developer mode" was then selected. Now, once the Oculus Quest is plugged into the PC in which Unity3D is installed, the headset will prompt the user to allow USB debugging.[10] From there, no further actions are needed from the headset itself.

Now, the rest of the work for setup was done through the Unity3D editor. The next important step was to import the Oculus Integration Package that is obtainable from the Unity

[9] https://developer.oculus.com/documentation/unity/unity-gs-overview/
[10] https://developer.oculus.com/documentation/unity/unity-enable-device/

Asset Store.[11] In order to do so, detailed instructions from Oculus were followed. The last and most convoluted step in the setup process was to configure a significant amount of settings. Detailed attention was paid to the documentation provided by Oculus.[12] Even with following the documentation, the process can be lengthy, but once completed, the development process for the Oculus Quest using Unity3D is straightforward and intuitive. The final result allows users to use an object in their Unity3D project called OVRCameraRig that acts as the main camera for the project and tracks the Oculus Quest headset in VR. Multiple settings are then configurable to fine tune the headset and camera if the user desires to do so.



✓ OVRCameraRig			
Tag	Player	Layer	Default
Transform			
Position	X -8.162	Y 0.215	Z 4.159
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1
✓ **OVR Camera Rig (Script)**			
Script	OVRCameraRig		
Use Per Eye Cameras	☐		
Use Fixed Update For Tracki	☐		
Disable Eye Anchor Cameras	☐		
✓ **OVR Manager (Script)**			
Target Devices			
Quest	✓		
Script	OVRManager		
Performance/Quality			
Use Recommended MSAA L	✓		
Monoscopic	☐		
Enable Adaptive Resolution	☐		
Min Render Scale	0.7		
Max Render Scale	1		
Head Pose Relative Offset R	X 0	Y 0	Z 0
Head Pose Relative Offset T	X 0	Y 0	Z 0
Profiler Tcp Port	32419		
Tracking			
Tracking Origin Type	Eye Level		
Use Position Tracking	✓		
Use IPD In Position Tracking	✓		
Reset Tracker On Load	☐		
Allow Recenter	✓		
Late Controller Update	✓		
Display			
Enable Specific Color Gamut	☐		
Quest Features			
Focus Aware	✓		
Hand Tracking Support	Controllers And Hands		
Requires System Keyboard	☐		
Android Build Settings			
Skip Unneeded Shaders	☐		

OVRCameraRig and its settings within Unity3D

[11] https://developer.oculus.com/documentation/unity/unity-import/
[12] https://developer.oculus.com/documentation/unity/unity-conf-settings/

Hand Tracking Setup

The next important step in the implementation process was to get hand tracking functional. Hand tracking is the user input method used for the project, instead of using standard controllers. Oculus also provided information on the hand tracking setup process. Detailed instructions were followed in order to get hand tracking functional.[13] In brief summary of the process, Oculus provides prefabs (3D objects) for the hands that are able to track the movement of each individual bone in the hand. Once these hand prefabs were added to the project, a number of other settings were required to be configured. This was pretty much the end of the documentation provided by Oculus. From there, a few things were still needed to get the hands functional within the project. The hand prefabs needed to have physics enabled, which was doable through some of the settings in Unity. This allowed the hands to actually interact with different 3D objects within the project. A few other settings were tuned and the end result was fully functional hand tracking in the Unity3D project.

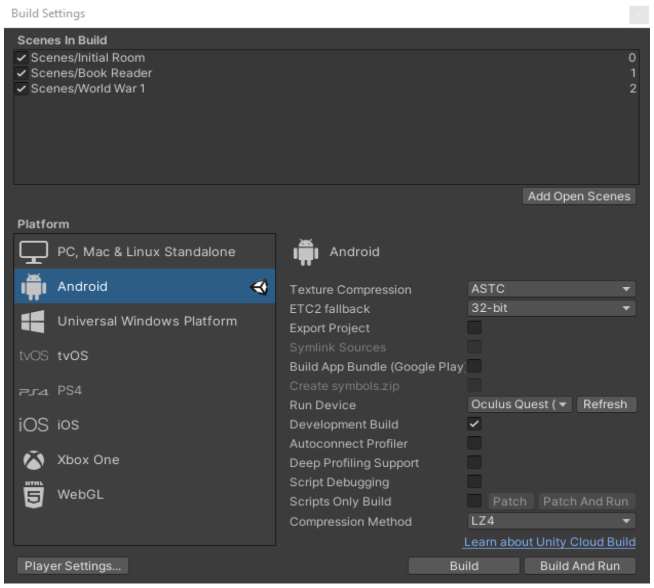
Testing and Running Unity3D Projects on Oculus Quest

Before doing anything else, the developer must first have gone through the process to enable development for the Oculus Quest. This process and cited documentation is discussed above. The process allows users to create their own project applications.

After development was enabled and the Oculus Quest headset and hand tracking were set up within Unity3D, a way to test the functionality was needed. In order to push the project application to the Oculus Quest headset for testing, a few things needed configured within Unity3D. Most of the things were already done during the Oculus Quest setup process, but a few

[13] https://developer.oculus.com/documentation/unity/unity-handtracking/?locale=en_US

additional configurations were necessary. In Unity3D, a section called “Build Settings” is where the developer can configure how the application is built and run. Within this section, one must select the “Run Device” setting, which indicates the device that the project application will run on. The Oculus Quest must be selected for this. Additionally, this section lays out the different scenes, or environments created within the project. If there are multiple scenes or environments, the developer must select which ones to include in the build. If a scene is not selected, it is not put into the build and therefore the user cannot see it within the running project application. Next, the developer must connect the Oculus Quest headset to the computer that Unity3D is running on via USB-C cable. Once connected, the developer can click an option to “Build and Run” the project application. This will build the application, creating an apk file, and then push it onto the Oculus Quest headset. As a side note, the Oculus Quest is built off of an android system, which is why an apk file is created for the project application. The project application is then located on the Oculus Quest headset and can be run just like any other application by the user of the headset.



Build Settings

Scenes In Build

✓ Scenes/Initial Room	0
✓ Scenes/Book Reader	1
✓ Scenes/World War 1	2

Platform

- PC, Mac & Linux Standalone
- Android
- Universal Windows Platform
- tvOS
- PS4
- iOS
- Xbox One
- WebGL

Android

Texture Compression	ASTC
ETC2 fallback	32-bit
Export Project	
Symlink Sources	
Build App Bundle (Google Play	
Create symbols.zip	
Run Device	Oculus Quest (
Development Build	✓
Autoconnect Profiler	
Deep Profiling Support	
Script Debugging	
Scripts Only Build	
Compression Method	LZ4

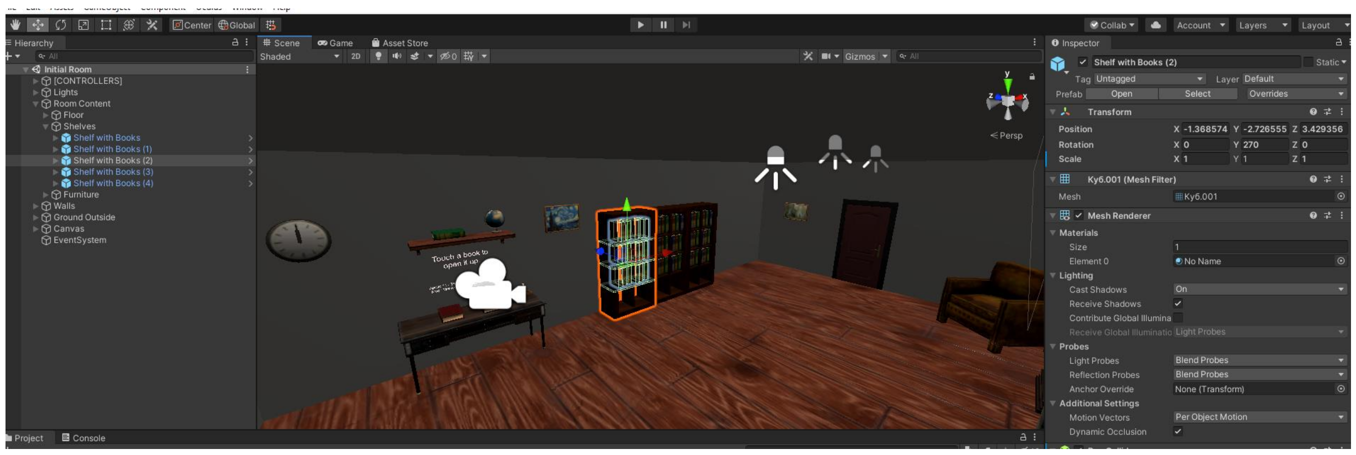
An example of the build settings for an Oculus Quest project with three scenes.

Creation of the Rooms and Environments

The rooms and environments that users can experience were created using the Unity3D editor. Within the editor, built in Unity3D objects as well as free assets from the Unity Asset Store were used. When implementing the design plan for the rooms and environments, Unity3D allows for precise positioning of objects so that the developer can place things exactly where they want. For detailed descriptions of the design of the environments and reasonings for doing so, please see Chapter III. Once all of the 3D objects that make up the rooms or environments were placed using the Unity3D editor, additional configuration had to be done to enable the objects to react to physics. Physics is an important part of the project. Without physics, items are static and allow no interaction from other objects. To implement physics, each item needed to have certain Unity3D attached to it. The items that need to be attached to it are called “Rigid Body” and “Box Collider” (or mesh collider). These items allow the object to detect other objects and react accordingly. For certain objects, colliders appear around the entire object and need no adjustment, but for others a great deal of adjustment is needed in order for the object to sense other objects that come in contact with it. This is solved via the Unity3D editor and also through testing. Physics, such as gravity, can be adjusted accordingly, but for this project are generally left at their default settings. In addition, lighting is an important part of the project.

Lighting the rooms and environments appropriately make a big difference in what is visible and how things look. Lighting and its effects are accomplished through the Unity3D editor. Unity allows for vast customization for lighting, but for the sake of this project, light is kept generally uniform so no shadows block out any of the environment objects. However, shadows may be implemented in the World War I battlefield in the future.

Overall, the implementation of the rooms and environments and the items in them was accomplished using the Unity3D editor.



Inspector
Shelf with Books (2) Static
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X -1.368574 Y -2.726555 Z 3.429356
Rotation X 0 Y 270 Z 0
Scale X 1 Y 1 Z 1
Ky6.001 (Mesh Filter)
Mesh Ky6.001
Mesh Renderer
Materials
Size 1
Element 0 No Name
Lighting
Cast Shadows On
Receive Shadows ✓
Contribute Global Illumina
Receive Global Illuminatic Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion ✓

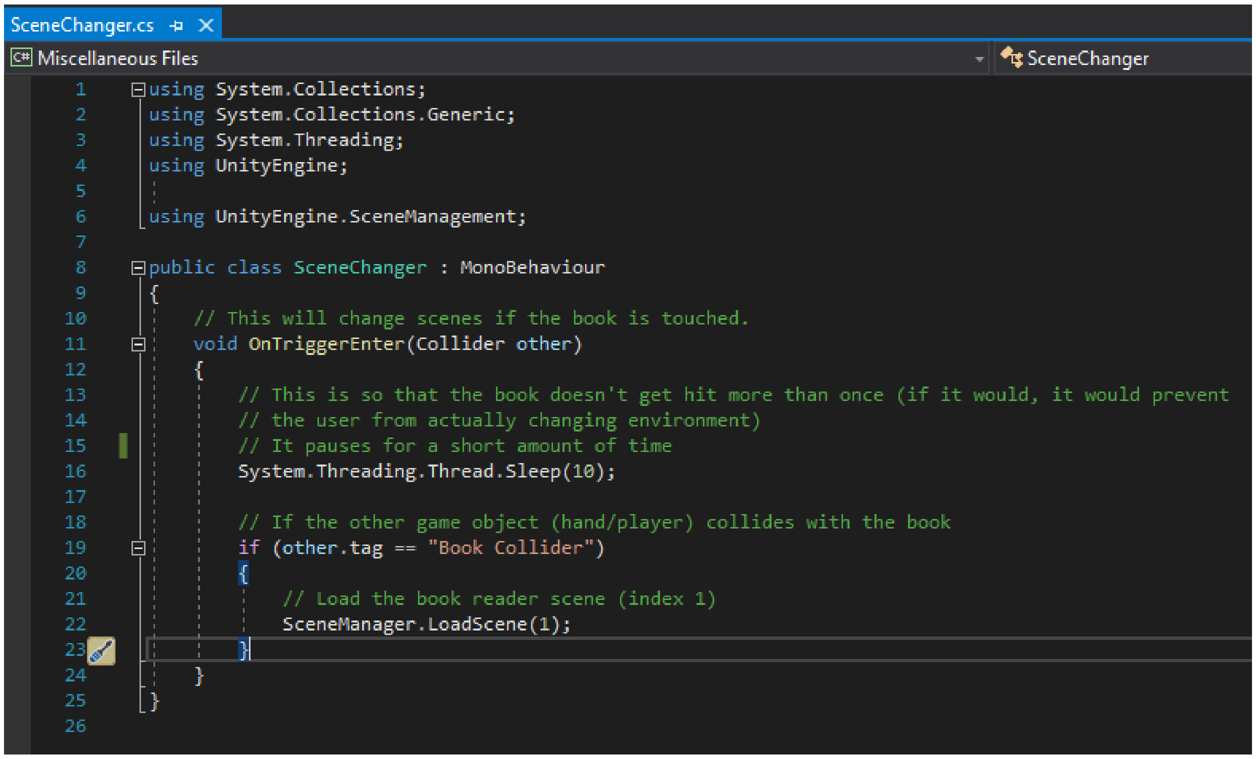
This is an image of the Unity3D editor.

Implementation of Opening The Book

In the initial room that the user starts in when first starting up the project application, they appear in front of a desk. On the desk, a couple of books are set out along with labels on what they are about and how to open them. After much consideration, as discussed in Chapter III, the way to open up the books is to simply touch them. The way this was implemented took some time and consideration. Initially, I attempted to see if there was a way to detect the collision between the hand prefabs provided by Oculus and the book object set up on the table. After much trial and testing, this did not work reliably. An alternative approach was needed.

Instead of detecting the collision between the hands and the book, I decided to detect a collision between the book on the desk and an invisible set of planes that were set up around the book. These invisible planes set up around the book are floating, but can still be moved when things contact them. When the user goes to touch the book, they impact the invisible plane which in turn impacts the book. This concept is similar to that of a button. To detect the collision, a C#

script is attached to the book that constantly checks to see if the collision is occurring, and if the script detects the collision, it opens up the book. To code this, the function called OnTriggerEnter() was used. This function is one used for Unity and documentation from Unity is provided for it.[14] Inside the function, I added code to do the actual detection and then open the book, which in reality is done by changing the scene to one in which the book is opened up. I also added a slight sleep function that will aid in preventing detecting the collision more than once. The code for the script can be seen below:



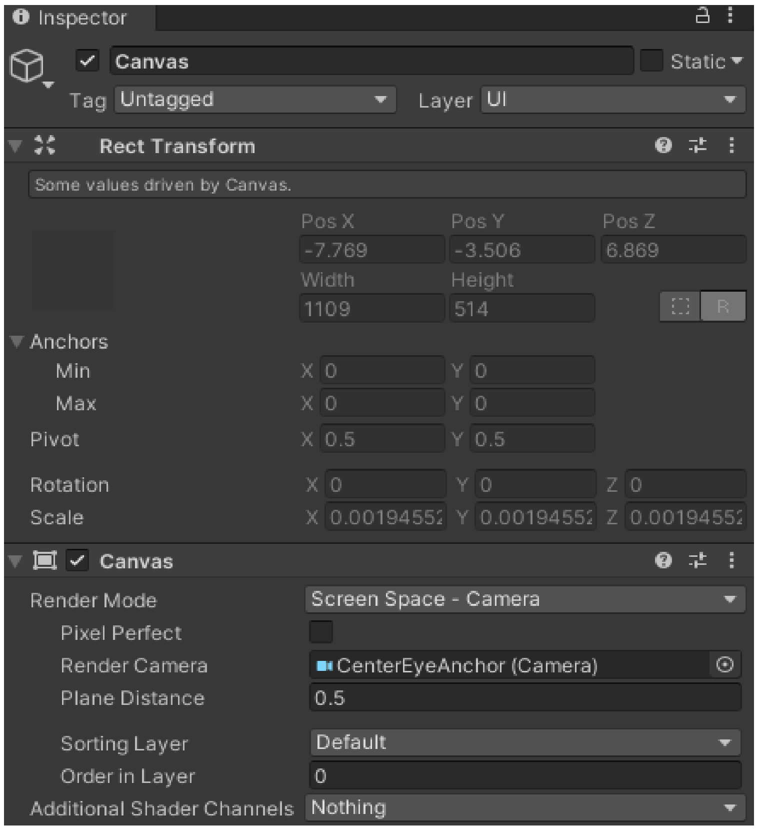
```
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;

using UnityEngine.SceneManagement;

public class SceneChanger : MonoBehaviour
{
    // This will change scenes if the book is touched.
    void OnTriggerEnter(Collider other)
    {
        // This is so that the book doesn't get hit more than once (if it would, it would prevent
        // the user from actually changing environment)
        // It pauses for a short amount of time
        System.Threading.Thread.Sleep(10);

        // If the other game object (hand/player) collides with the book
        if (other.tag == "Book Collider")
        {
            // Load the book reader scene (index 1)
            SceneManager.LoadScene(1);
        }
    }
}
```

This is the SceneChanger script used to open the book.

[14] https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

The Book Reader UI

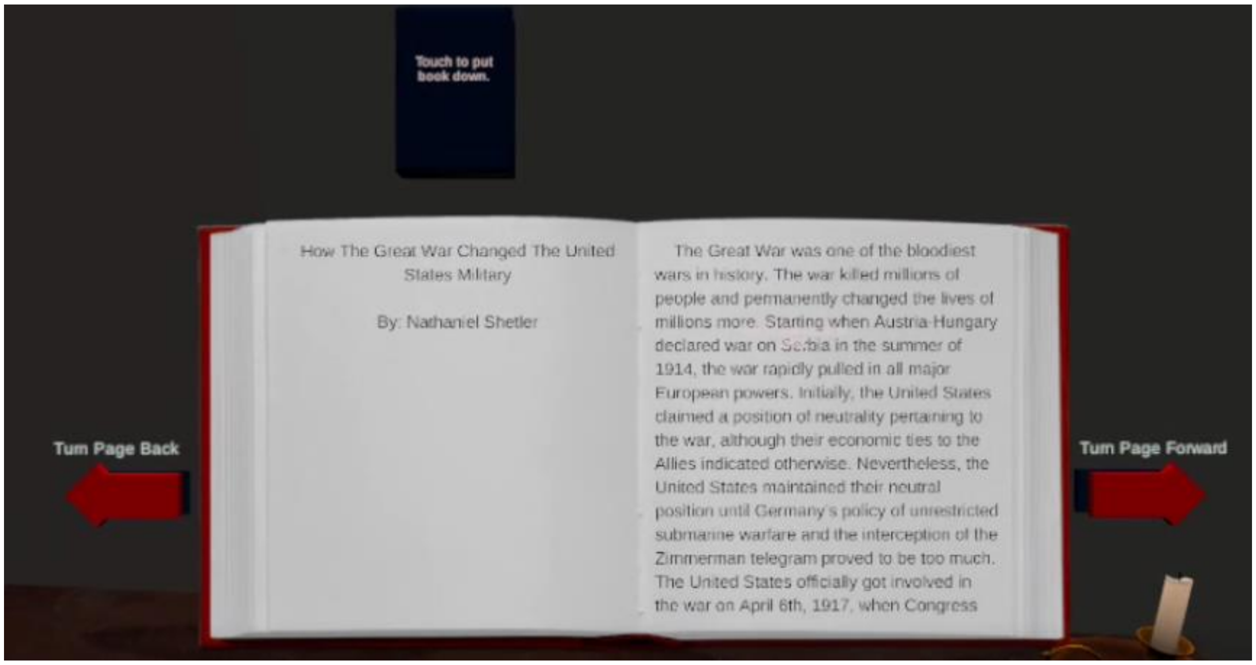
Once the user opens up the book, a new UI is present in front of them. The book is attached to the user's vision and travels with them wherever they look. In addition, buttons to turn pages, a button to close the book, and a button to transport to the environment that they are reading will all be present at some point while reading the book. To implement this, a Unity "Canvas" was used. The canvas is an element provided by Unity that outputs UI elements. To get the canvas to follow wherever the user looks, some configuration was required. I needed to set the "Render Mode" to "Screen Space - Camera" and then set the "Render Camera" to the OVRCameraRig CenterEyeAnchor that represents the Oculus Quest headset. These settings can be seen below:



```
Inspector
Canvas                                    Static
Tag Untagged                    Layer UI
Rect Transform
Some values driven by Canvas.
                      Pos X        Pos Y        Pos Z
                      -7.769       -3.506       6.869
                      Width        Height
                      1109         514                  R
Anchors
  Min                 X 0          Y 0
  Max                 X 0          Y 0
Pivot                 X 0.5        Y 0.5
Rotation              X 0          Y 0          Z 0
Scale                 X 0.00194552 Y 0.00194552 Z 0.00194552
Canvas
Render Mode                 Screen Space - Camera
  Pixel Perfect             
  Render Camera             CenterEyeAnchor (Camera)
  Plane Distance            0.5
  Sorting Layer             Default
  Order in Layer            0
Additional Shader Channels  Nothing
```

This image is of the settings for the Canvas UI object in Unity3D

After configuring the Canvas, I then had to add the objects that make up the book and other UI elements. For the book, a blank image of a book was used. For the pages within the book, a group of Unity UI Text objects were created and loaded with the page text. Only two images were made visible at once. Setting the visibility of the text objects was done through a C# script. The pages are loaded into a list that is easily changeable to the user through the Unity Editor. For the button that puts the book down, a similar solution to that of how the user picks up the book was used. In front of the button is an invisible plane and when the user goes to touch the button, the plane contacts the button which in turn runs the script that puts the book down. The same idea was also used for the button that transports the user to the environment that they are reading about, although this button is only visible when a certain page number is reached. For the buttons that turn the pages, the detection for a button press remains similar to other buttons, but in order to actually turn the page, more scripts were needed. More details on the page turning implementation can be found in the following section. The two page turn buttons are found on either side of the book, one for back and one for forward.

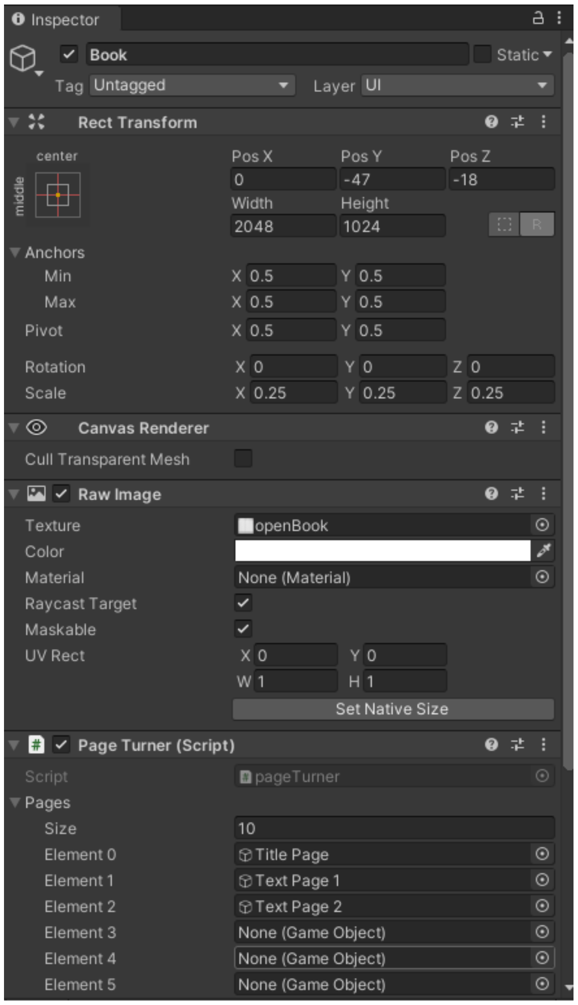


How The Great War Changed The United States Military

By: Nathaniel Shetler

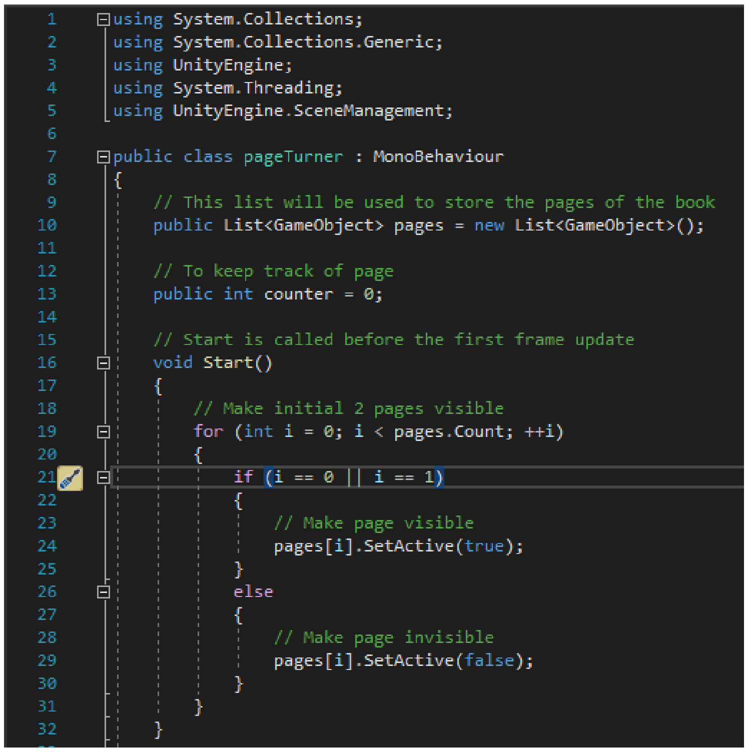
The Great War was one of the bloodiest wars in history. The war killed millions of people and permanently changed the lives of millions more. Starting when Austria-Hungary declared war on Serbia in the summer of 1914, the war rapidly pulled in all major European powers. Initially, the United States claimed a position of neutrality pertaining to the war, although their economic ties to the Allies indicated otherwise. Nevertheless, the United States maintained their neutral position until Germany's policy of unrestricted submarine warfare and the interception of the Zimmerman telegram proved to be too much. The United States officially got involved in the war on April 6th, 1917, when Congress

This image shows the current UI for the user while reading the book. **Note:** The transport to environment button only appears when a certain page number is reached.



```
Inspector
Book                                   Static
Tag Untagged                 Layer UI
Rect Transform
center          Pos X       Pos Y       Pos Z
middle          0           -47         -18
                Width       Height
                2048        1024                R
Anchors
  Min           X 0.5       Y 0.5
  Max           X 0.5       Y 0.5
Pivot           X 0.5       Y 0.5
Rotation        X 0         Y 0         Z 0
Scale           X 0.25      Y 0.25      Z 0.25
Canvas Renderer
Cull Transparent Mesh
Raw Image
Texture         openBook
Color
Material        None (Material)
Raycast Target  ✓
Maskable        ✓
UV Rect         X 0         Y 0
                W 1         H 1
                Set Native Size
Page Turner (Script)
Script          pageTurner
Pages
  Size          10
  Element 0     Title Page
  Element 1     Text Page 1
  Element 2     Text Page 2
  Element 3     None (Game Object)
  Element 4     None (Game Object)
  Element 5     None (Game Object)
```

At the bottom of this image is the list of pages created in the C# script attached to the book object.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading;
using UnityEngine.SceneManagement;

public class pageTurner : MonoBehaviour
{
    // This list will be used to store the pages of the book
    public List<GameObject> pages = new List<GameObject>();

    // To keep track of page
    public int counter = 0;

    // Start is called before the first frame update
    void Start()
    {
        // Make initial 2 pages visible
        for (int i = 0; i < pages.Count; ++i)
        {
            if (i == 0 || i == 1)
            {
                // Make page visible
                pages[i].SetActive(true);
            }
            else
            {
                // Make page invisible
                pages[i].SetActive(false);
            }
        }
    }
```

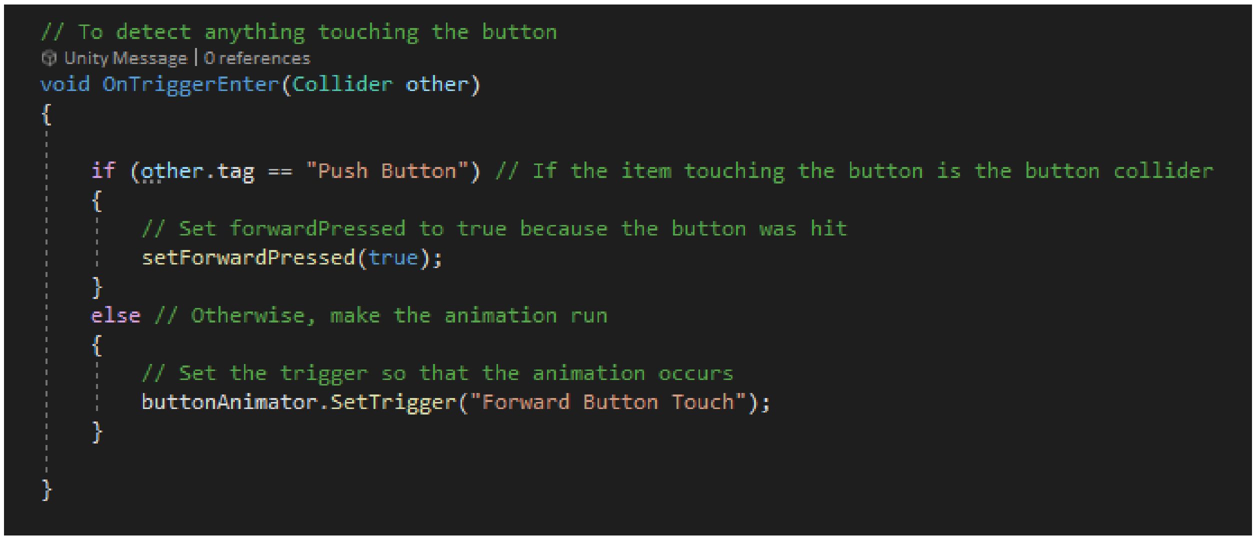
This is the portion of the script that is stored in the book. This part of the script sets the first two pages of the book as visible and the rest as invisible at the start of the application.

Page Turning Implementation

To implement the page turning, multiple different GameObjects and scripts were needed to create a smooth page turning simulation. The basic components used for the implementation include: a forward and back button, animation components for the buttons, colliders for the buttons to hit, a book object containing text GameObjects that represents the pages of the book, and various scripts that are attached to each GameObject. To start, when the user comes in contact with the forward or back button, a script (forwardButtonAnimation.cs or backButtonAnimation.cs) detects the collision. In the script, an animation trigger, which starts the button animation, is activated. This makes the animation start, which moves the button backwards. The animation was created through Unity[15].

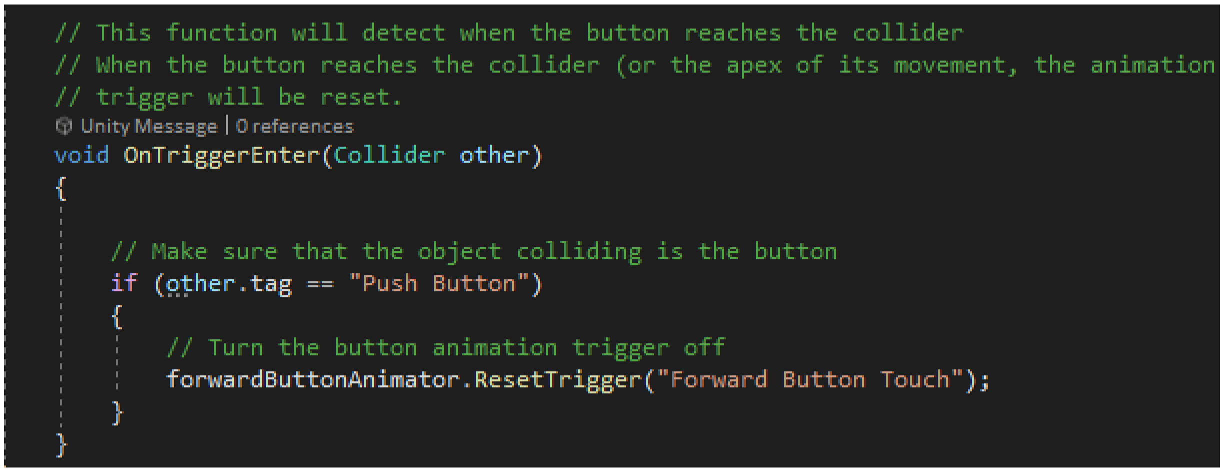
Once the button goes all the way rearward, it contacts the button collider. A few different things occur when this collision happens. First, a different script, that is attached to the button collider, that is called either forwardButtonCollider.cs or backButtonCollider.cs detects the collision and resets the animation trigger. This animation trigger reset ensures that the animation will stop when the button moves forward to its original starting position. Second, when the button contacts the button collider, it is also detected by the forwardButtonAnimation.cs or backButtonAnimation.cs scripts that are attached to the forward and back buttons. When this detection occurs, the script sets a boolean named ‘forwardPressed’ or ‘backPressed’ to true. When this happens, another script, named pageTurner.cs, which is attached to the book GameObject that stores all of the book pages, detects that the boolean has been set to true. This script checks the value of the boolean each frame. When the script detects that the boolean is set to true, the simulation for the page turning begins. If the forward button is pressed, the next two pages are made visible, and the current pages are made invisible. If the back button was pressed, the previous two pages, if not at the start of the book, are made visible and the current pages are made invisible. The page that the user is currently on is kept track of through a counter variable. Parts of the scripts can be seen below:

[15] https://docs.unity3d.com/Manual/AnimationSection.html
- https://docs.unity3d.com/ScriptReference/Animation.Play.html
- https://docs.unity3d.com/Manual/animeditor-AnimatingAGameObject.html
- https://docs.unity3d.com/ScriptReference/Animator.SetTrigger.html



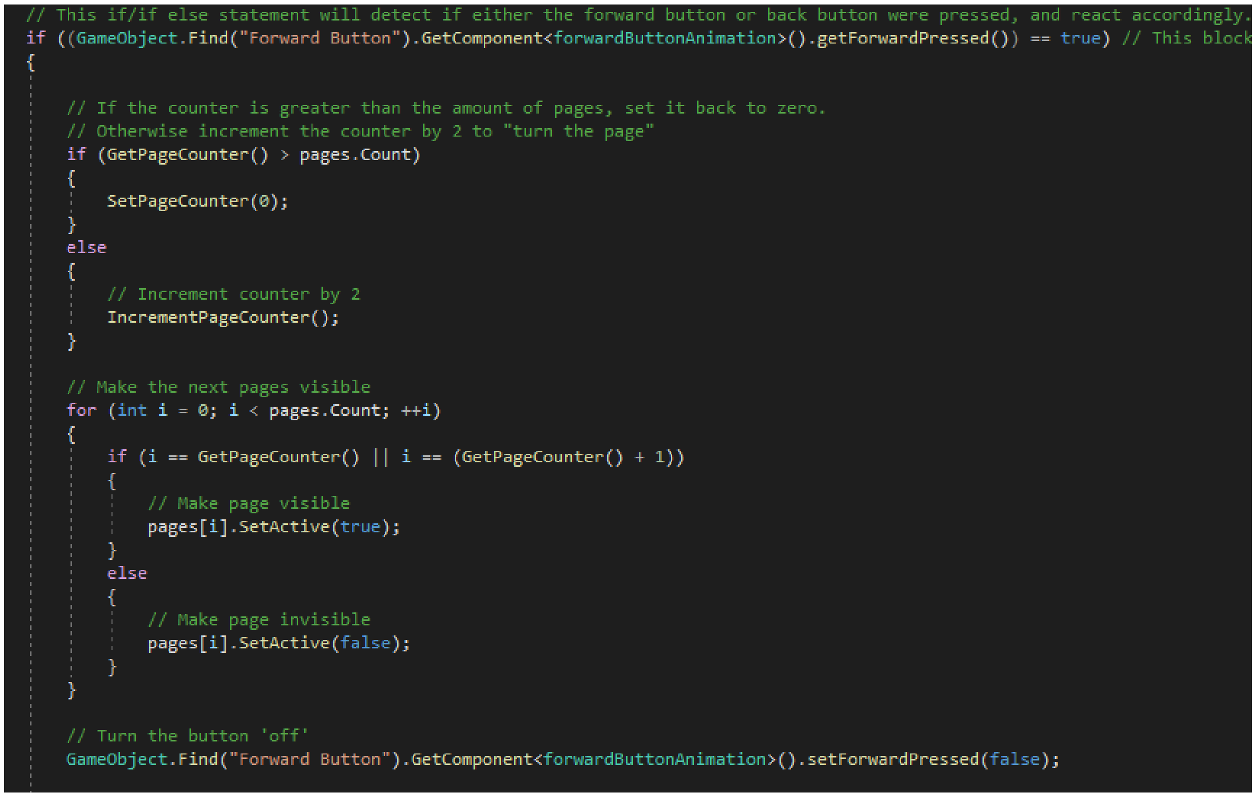
```
// To detect anything touching the button
Unity Message | 0 references
void OnTriggerEnter(Collider other)
{

    if (other.tag == "Push Button") // If the item touching the button is the button collider
    {
        // Set forwardPressed to true because the button was hit
        setForwardPressed(true);
    }
    else // Otherwise, make the animation run
    {
        // Set the trigger so that the animation occurs
        buttonAnimator.SetTrigger("Forward Button Touch");
    }

}
```

This is part of the forwardButtonAnimation.cs script that is attached to the forward turn button. This part detects a collision and depending on what the collision is, either sets a trigger to start the animation, or sets the forwardPressed boolean to true to start the page turning.



```
// This function will detect when the button reaches the collider
// When the button reaches the collider (or the apex of its movement, the animation
// trigger will be reset.
Unity Message | 0 references
void OnTriggerEnter(Collider other)
{

    // Make sure that the object colliding is the button
    if (other.tag == "Push Button")
    {
        // Turn the button animation trigger off
        forwardButtonAnimator.ResetTrigger("Forward Button Touch");
    }
}
```

This is part of the forwardButtonCollider.cs script that is attached to the button collider. When the button collider is touched by the forward turn button, the trigger is reset so that the animation stops once the button returns to its original starting position.

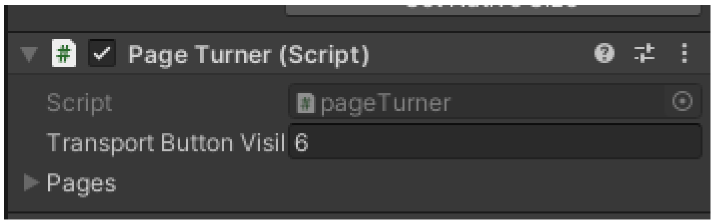


```
// This if/if else statement will detect if either the forward button or back button were pressed, and react accordingly.
if ((GameObject.Find("Forward Button").GetComponent<forwardButtonAnimation>().getForwardPressed()) == true) // This block
{

    // If the counter is greater than the amount of pages, set it back to zero.
    // Otherwise increment the counter by 2 to "turn the page"
    if (GetPageCounter() > pages.Count)
    {
        SetPageCounter(0);
    }
    else
    {
        // Increment counter by 2
        IncrementPageCounter();
    }

    // Make the next pages visible
    for (int i = 0; i < pages.Count; ++i)
    {
        if (i == GetPageCounter() || i == (GetPageCounter() + 1))
        {
            // Make page visible
            pages[i].SetActive(true);
        }
        else
        {
            // Make page invisible
            pages[i].SetActive(false);
        }
    }

    // Turn the button 'off'
    GameObject.Find("Forward Button").GetComponent<forwardButtonAnimation>().setForwardPressed(false);
```

This is part of the pageTurner.cs script that is attached to the book GameObject. In this part, when the boolean ‘forwardPressed’, which is stored in the forwardButtonAnimation.cs script attached to the forward turn button, is set to true, the page counter is increased and then the new pages become visible, while the old ones become invisible. The boolean is then set to false to stop the page turning.

Transport Button

The transport button, that transports the user to the world or environment that they are reading about, only becomes visible when a certain page is reached. This page number can be changed easily through the Unity editor to whatever the user desires. There is a variable attached to the pageTurner.cs script called transportButtonVisible that controls this logic. A picture from the Unity editor of this can be seen here:



This is an image of the pageTurner.cs script that is attached to the book. The user can easily change the Transport Button Visible variable to change when the transport button becomes visible.

Before the user reaches the page that the button becomes visible, the button is located behind the book, so it is not visible to the user. Once the page is reached, the button animates forward and then upwards until it reaches its final position above the book to the right. The animation for the button was created through Unity using keyframe animations. The button functions the same as each button described previously.

Chapter V

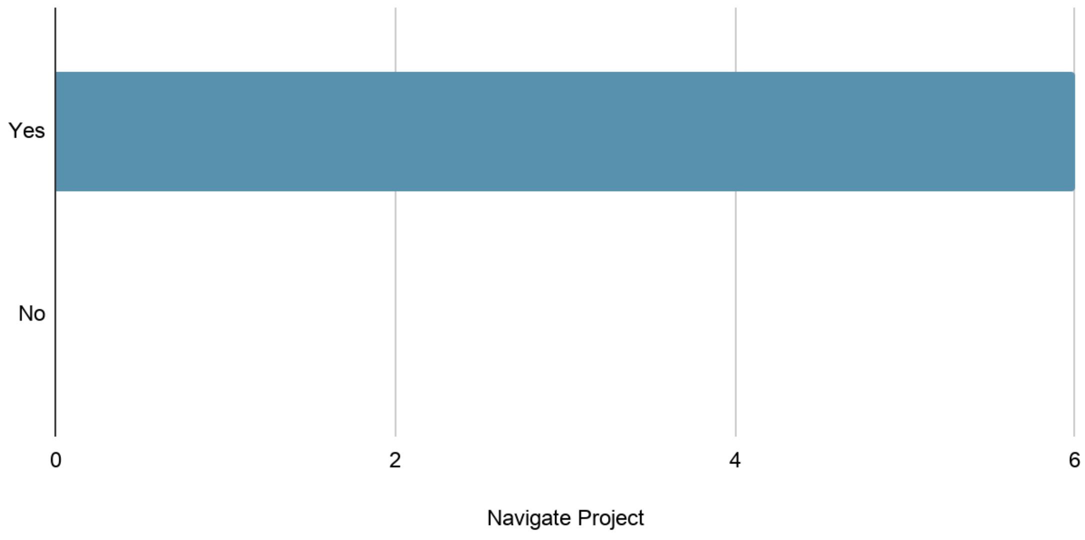
Results and Analysis

In conclusion, this project provided an excellent opportunity to blend the disciplines of computer science and education. The project provides users with a new and interactive way to experience reading and learning about history. The goal of the project was achieved, and the final result is a fully functional virtual reality application for the Oculus Quest headset. The basic outline of the project can be found at: https://github.com/NateShetler/Senior-Project. However, this location doesn't house all of the various assets and the project APK file (the application itself), due to the size of the files. If these are wanted, contact Nathaniel Shetler and the full set of project files and/or the project APK will be provided. Additionally, a demonstration of the project in action can be found on YouTube at: https://www.youtube.com/watch?v=Q7o4C9x2f6E. In all, the project provided a great opportunity to learn about the development process of a Virtual Reality application. It furthers the research into the various uses of Virtual Reality in an educational setting, a growing area of study that will have an impact in the education of students moving forward.

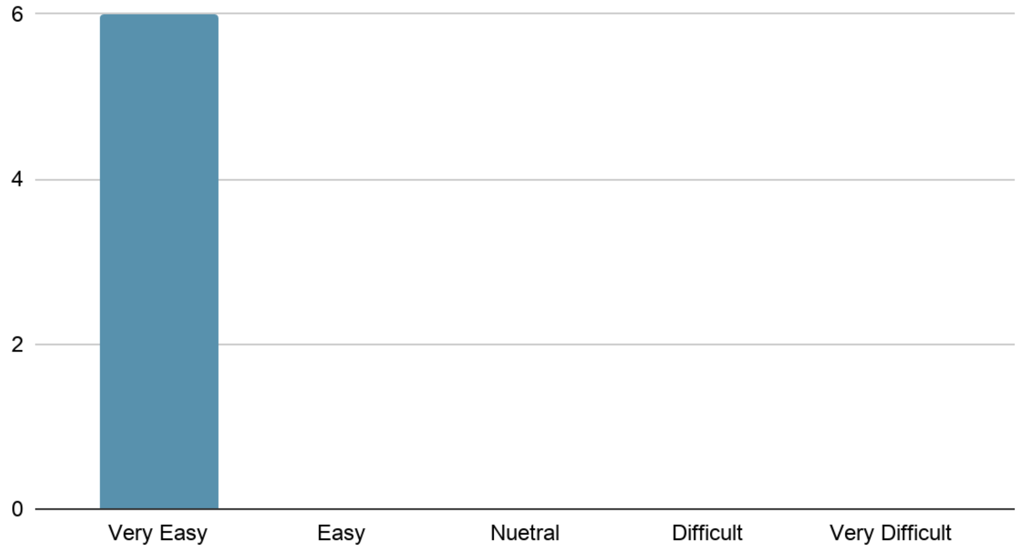
User Experience

A survey was done on a group of six users for the project. The users tested the project and were asked a series of questions to measure the user experience of the application. Due to the pandemic, additional users were unable to be asked to test out the application at this time. Once more users are able to test the application, their input will be added to the charts found in this section.

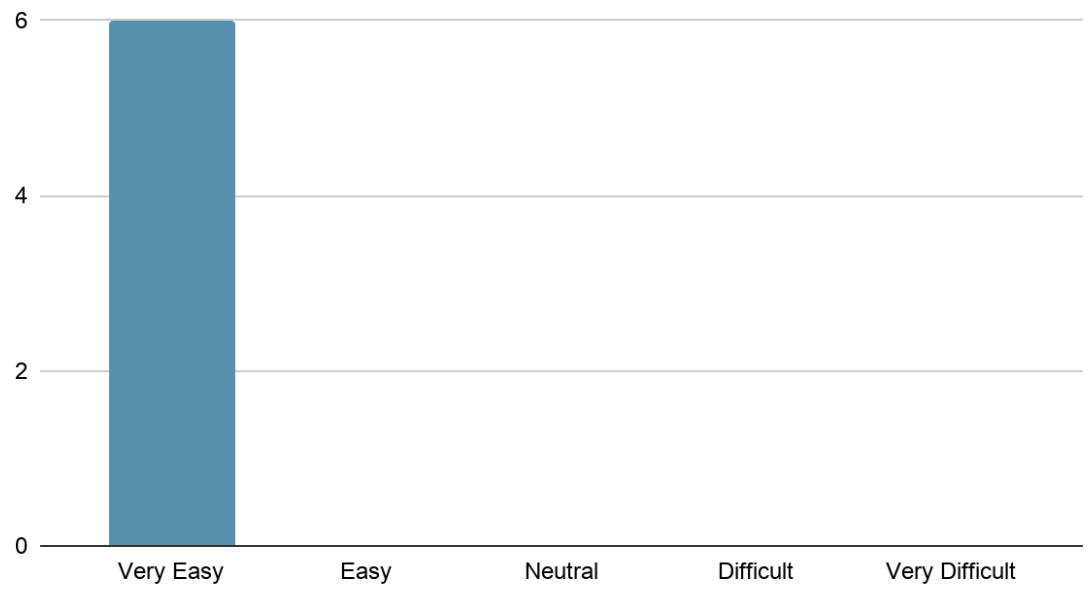
Were you able to successfully navigate the User Interface of the application?



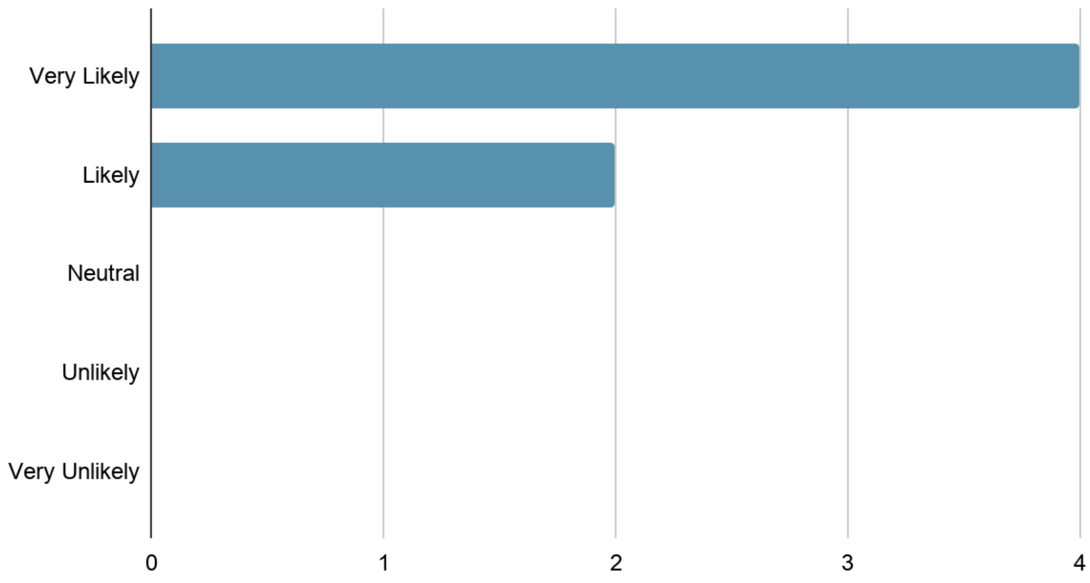
How easy was the User Interface to use?



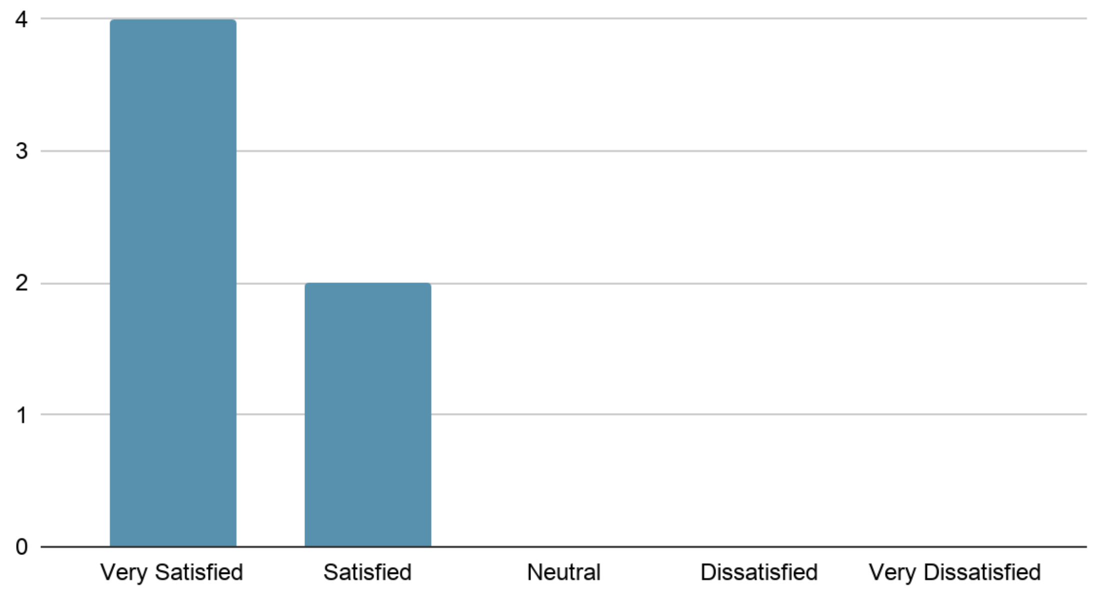
How easy were the controls to figure out and use?



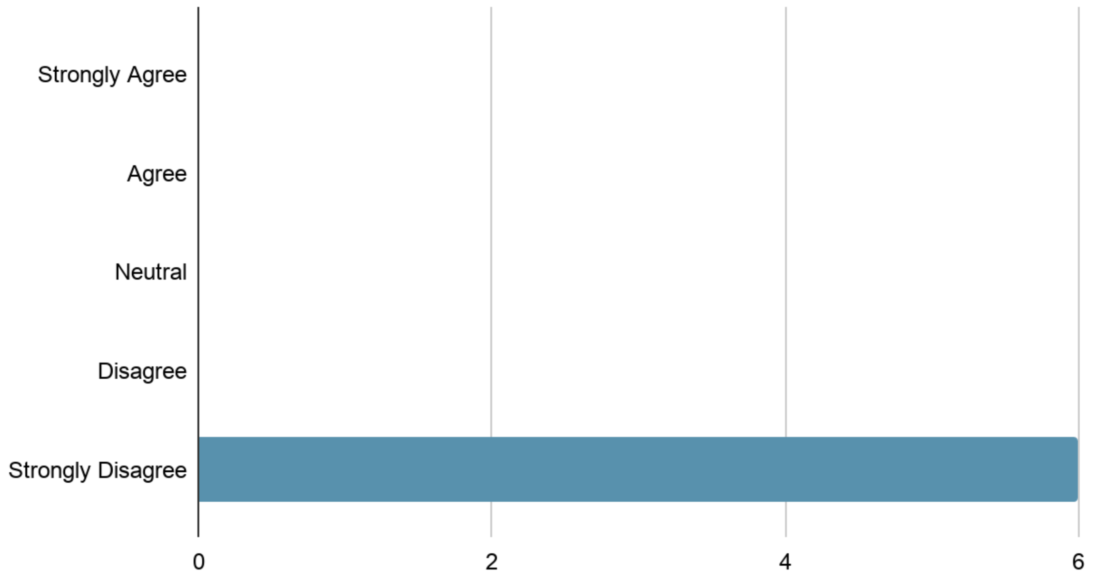
How likely are you to recommend the application to others?



How satisfied are you with the application experience?



I found the application unnecessarily complex



Chapter VI

Future Work

In the future, this project has many opportunities for growth. While the goal of the project was reached and the result is satisfactory, the idea can be expanded on by myself or others. For instance, the World War I environment could be added to, additional books and environments could be added, or alternate initial rooms could be added.

In the World War I environment, more events can be added to further the user immersion. The artillery in the trench could become functional and allow the user to fire it. Vehicles, such as tanks or planes flying above can be added. The firearms can be made to be picked up. However, picking up and using these firearms would present a number of challenges due the nature of hand tracking based controls. There are numerous possibilities on what can be added to this environment.Additionally, other books and environments can be added to the application. Providing the user with more options would expand the appeal of the application. Other history books or even fictional books can be added. It would not be difficult to add additional books, as the framework for the project functionality is already in place. The possibilities for additional books and environments are seemingly limitless. As long as there is written material or a book, it can be added and implemented. Finally, additional initial rooms or reading rooms can be added. To further customize the reading experience, the user could choose which room they would like to start in or read in. The rooms can have different styling based on the user’s preferences. This would further the appeal to more users. Overall, the project has many opportunities for future work, and many of them will be explored in the coming months and years.

Bibliography

Introduction:

Google. Bring Your Lessons to Life with Expeditions. Retrieved From: https://edu.google.com/products/vr-ar/expeditions/#about

Oculus. Oculus Go Description. Retrieved from: https://www.oculus.com/go/?locale=en_US

Oculus. Oculus Quest Description. Retrieved From: https://www.oculus.com/quest/

Oculus. Oculus Quest 2 Description. Retrieved From: https://www.oculus.com/quest-2/?locale=en_US

Oculus. Oculus For Business. Retrieved From: https://business.oculus.com/?locale=en_US

Technologies, Unity. 3D Software for Architecture and Engineering Construction. Retrieved From: https://unity.com/solutions/architecture-engineering-construction

Technologies, Unity. 3D Animation, CGI, and Cinematics. Retrieved From: https://unity.com/solutions/film-animation-cinematics

Technologies, Unity. 3D Product Visualization for automotive, transportation, and manufacturing. Retrieved From: https://unity.com/solutions/automotive-transportation-manufacturing

Implementation:

Oculus Developer. Oculus Get Started with Unity Guide. Retrieved From: https://developer.oculus.com/documentation/unity/unity-gs-overview/

Oculus Developer. Enable Device for Development. Retrieved From: https://developer.oculus.com/documentation/unity/unity-enable-device/

Oculus Developer. Import Oculus Integration. Retrieved From:

https://developer.oculus.com/documentation/unity/unity-import/

Oculus Developer. Configure Unity Settings. Retrieved From:

https://developer.oculus.com/documentation/unity/unity-conf-settings/

Oculus Developer. Hand Tracking in Unity. Retrieved From:

https://developer.oculus.com/documentation/unity/unity-handtracking/?locale=en_US

Technologies, Unity. Collider OnTriggerEnter. Retrieved From:

https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

Technologies, Unity. Animation Guide. Retrieved From:

https://docs.unity3d.com/Manual/AnimationSection.html

Technologies, Unity. Playing Animations. Retrieved From:

https://docs.unity3d.com/ScriptReference/Animation.Play.html

Technologies, Unity. Animating a Game Object. Retrieved From:

https://docs.unity3d.com/Manual/animeditor-AnimatingAGameObject.html

Technologies, Unity. Setting Animation Trigger. Retrieved From:

https://docs.unity3d.com/ScriptReference/Animator.SetTrigger.html

Technologies, Unity. Scene Management. Retrieved From:

https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.
html

Technologies, Unity. Lists and Dictionaries Guide. Retrieved From:

https://learn.unity.com/tutorial/lists-and-dictionaries#