Automated Data Collection: A UnityXR Framework

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
Blake Blackport
December 2022

Proposed to:
Dr. Toni Pence, Committee Chair

Abstract

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 UNCW Mixed Reality Lab

I've been working in the Mixed Reality department of the research lab at UNCW since January 2020. I've worked on two major projects during my career, as well as collaborated with the machine learning department of the research lab. The overall goal of the lab as a whole is to create unique experiences using Mixed Reality that can somehow provide a new benefit to users. We have both Virtual Reality headsets, and Augmented Reality headsets in the research lab. However, my work was primarily focused on Virtual Reality using the Oculus Quest headset, both the first and second generation.

1.2 ORNL Data Visualization Project

This was the project I initially joined when I began at the research lab, and the one that most alligned with my interests. Our overall goal for the project was to create new or improved methods of visualizing data with a 3 dimensional display (VR or AR headset) that was in some way an improvement over the standard 2 dimensional display (computer). My original interest going into computer science was data science, however this project made me grow to love data visualization in particular. I still prefer doing the more backend-oriented work involving data, however I also like the analysis side of things. With data visualization I feel there's more of an emphasis on analysis, as that's typically the main reason for visualizing data. It's also an added benefit when your end

result is a futuristic display in VR, and not some data in a csv file.

Clearly for a data visualization project, there is a lot of work being done on data. This was my main role on the project, to create a simple and efficient interface for the other developers to easily interact with whatever data they needed to visualize. I helped contribute to projects such as a visualization of pollution data by placing the user in a room and filling the room with the amount of particles based on the pollution levels in the area they were viewing. Another fascinating project I contributed to was visualizing Covid data on a 3D globe for analysis. I played a large role in obtaining, parsing, and analyzing this data while my coworker focused more on the visualization. Once more experienced I moved onto larger individual projects, such as creating a large "hard drive" framework built to make interfacing with any file type easy for my coworkers. We would frequently get data in a variety of formats so by implementing this they only had to worry about interfacing with my class, while I worried about how that data would be parsed into the class based on the file type. I also moved more into the front-end visualization side of things on this project, building a 3D graphing tool for placing a 3D graph anywhere in a scene based on relative location. Complete with tools for scaling, slicing, and moving the graph dynamically. However back-end still remains my favorite portion to work on.

1.3 Virtual Access to Stem Careers Project

The Virtual Access to Stem Careers Project, or VASC for short, aims to provide unique learning experiences through the use of Mixed Reality. Our current projects goal is to build an educational program to teach students about sea turtles using the Oculus Quest 2. This is a surprisingly valuable project as it achieves something previously not feasible. If a student living in the mid-west wanted to learn about sea turtles, or marine biology in general, they may have to drive upwards of 10 hours just to get to the nearest beach for hands-on experience. By creating this experience in a Virtual Reality program, it gives us the portability to take this experience anywhere. It would be quite expensive for a school to take a long field trip to the beach, however they can purchase an Oculus Quest 2 for only $300. This headset can be reused many times and a few headsets could allow the entire school to gain this experience.

While crafting this program, we made accuracy of the educational experience one of our top priorities. We would regularly visit professionals that do what we want to program for a living.

Seeing as most of our development team knows little to nothing about sea turtles, this feedback is essential for accurate information. We also like to return after creating the program to demo the project to them for feedback on things we did incorrectly or things we might be able to improve.

1.4 Motivation

While working on the VASC project we realized we wanted to track the users progress throughout the project. Using this data we could draw conclusions accurate conclusions on where points of issues may be in the project and improve them. In order to collect this data, we needed a framework for logging events from the Unity XR system in a simple way that developers not familiar with the framework can easily use it. Unfortunately, we were unable to find an existing framework for our needs. As a result, my goal is to create a custom framework to specifically meet our needs. However, as this could easily be applied for any project using Unity XR, I also want it to allow for general use and be easily implemented to any new or existing projects. This served as the inspiration for this capstone.

1.5 Mixed Reality

Mixed Reality is a term that has recently become quite popular. There is a lot of misunderstanding surrounding the word so this section will aim to define it more clearly. This section will heavily focus on head-mounted-displays (HMD) in particular. HMD's are displays that you wear on your head. Since they are displays, this means to be an HMD it must have some display aspect. This means it is safe to assume that all HMD's utilize some form of Mixed Reality. Many devices have VR or AR capabilities nowadays, almost all modern phones have AR capabilities built-in, if you add some cardboard google glasses it can do VR as well. I chose to focus on HMD's as they are the most relevant to my work. They are also arguably one of the most "true" forms of Mixed Reality on the market today as their main goal typically is some form of Mixed Reality.

1.5.1 History

The term Mixed Reality was first coined in 1994 in a paper by Paul Milgrim and Fumio Kishino (https://learn.microsoft.com/en-us/windows/mixed-reality/discover/mixed-reality). It is surprising to most, myself included, to think of terms like Mixed Reality being discussed so long ago. Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) are similar to Machine Learning (ML) and Artificial Intelligence (AI) in the sense that they have been theoretically possible for a long time, however they have only recently become realistically possible. This is of course thanks to the astounding innovations of hardware over the past few decades. ML and AI can require massive amounts of computing power and storage which would previously have cost exorbitant amounts of money if they were even possible to build with the technology at that time. AR, VR, and MR, when specifically referring to head-mounted-displays (HMD) faced a similar but slightly opposite issue. They all require decent computing power, but still nowhere near the amount that ML or AI would. The issue was actually in miniaturizing this computing power.

Modern HMD's are now standalone and can function solely with a standalone headset, many don't even require controllers now. For this to be possible you need to pack in a ton of computing power while still fitting it on someone's head. It also had to be light enough for the headset to be comfortable and not cause strain on the users neck or back. This was simply not feasible with the technology available two decades ago. Because of this, the first versions of Virtual Reality HMD's were tethered by a large cord to a mount on the ceiling. This mount would either be stationary

or allow for some movement of the user. The headset was actually only a display, and all of the computing would be done by the device connected to the headset through the cord.

Many modern headsets still use this technique today actually, though they use a tiny USB cord plugged into a personal computer. The reasoning is also the same as it was back then, to achieve more computing power. The headset we develop on, the Oculus Quest 2, is a perfect example of why one might want to do this. The Quest 2 is a fully standalone headset, meaning all you need is the headset to run any Quest applications. However, you have the option to plug the quest into a computer using a USB-C cord. It is very annoying to use a wired headset, because as you move around it is very easy to get tripped up with the cord, so plugging in is certainly a disadvantage in that regard. Not to mention requiring a computer with good enough parts to run the program. The advantage however, is now your computer can run the application your headset would have and your headset, like the old days, will just function as a display. This means you can use significantly better parts than could be feasible within a headset alone to run your applications.

1.5.2 Virtual Reality

Virtual Reality is defined as "the use of computer modeling and simulation that enables a person to interact with an artificial three-dimensional (3-D) visual or other sensory environment" (https://www.britannica.com/technology/virtual-reality). As previously mentioned I chose to focus mainly on HMD's which provide full-immersion, however there are two other types of VR (https://heizenrader.com/the-3-types-of-virtual-reality/#: :text=There%20are%203%20primary%20categories,%2C%20a
The least commonly discussed kind, is one many of us know and may use regularly. If you've ever played a 3D game on a game console or your computer, you've used VR. This form of VR is referred to as no-immersion. This is because you can clearly tell you are still in the physical world, however you are interacting and experiencing a 3D environment through the screen of your device. The third type of VR is semi-immersive and it is a lot less clearly defined, and also less common, however there are cases that don't fit neatly within the other two types. Generally speaking they are 3D experiences that represent the real world, but are digitally generated and displayed on a device that does not provide full-immersion. A great example of this would be a virtual tour you can take on your phone of an apartment or house. This is a 3D representation of an object in the real world, but it's viewed on a device that isn't fully immersive like an HMD would be.

The most pure VR experience would be a fully Digital World in which as far as you're able

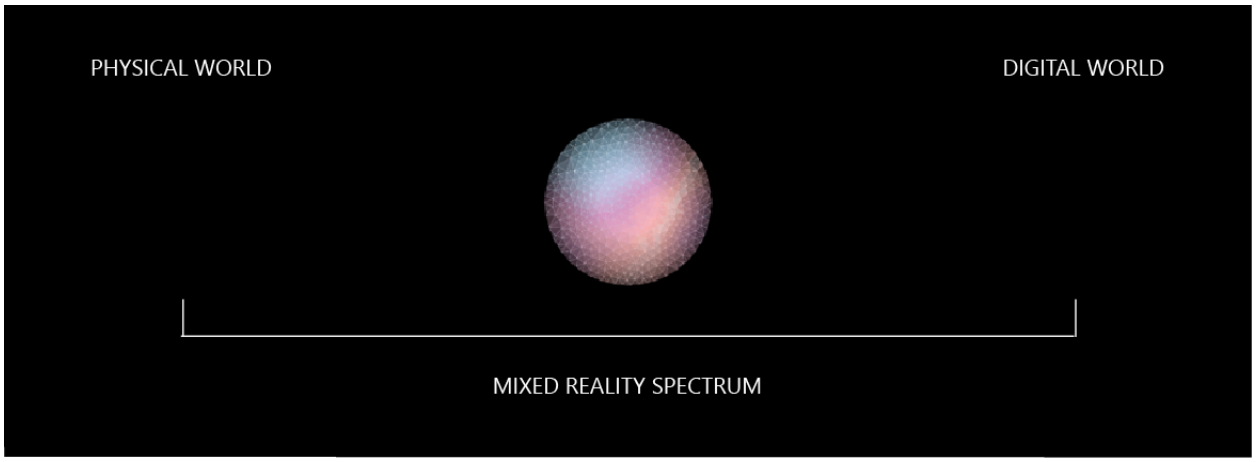

Figure 1.1: A visual representation of Mixed Reality

to tell, the digital world is real. There are no devices that can provide such an experience today, as such the term Virtual Reality refers to a spectrum of devices. Currently the devices farthest on the Virtual Reality spectrum are HMD's, often paired with other devices as well such as hand-tracking gloves, controllers, and some have even gone as far as to make haptic equipment that reacts to the environment. For example, one group designed haptic gloves with a motor on each finger that restricts your movement when you grab an object in VR (https://www.dextarobotics.com/en-us). The result of this is the sensation of truly feeling an object. If you were to pick up say a baseball in VR, you couldn't just close your fist, when your fingers began colliding with the baseball object in VR, the motors would restrict your fingers movement so it feels like you're really grabbing a baseball. It is astonishing how close to a fully digital world we can get to today combining multiple of these haptic feedback equipment. However some argue the only way to achieve a truly digital world would be through some form of brain-computer-interface. This is due to the fact that no matter how immersive the equipment becomes, you likely will still be at least somewhat aware of the fact that you are still in the physical world wearing equipment that is generating your experience. However, it is theorized that through brain-computer-interfaces one could possibly experience true full-immersion in a Virtual Reality simulation.

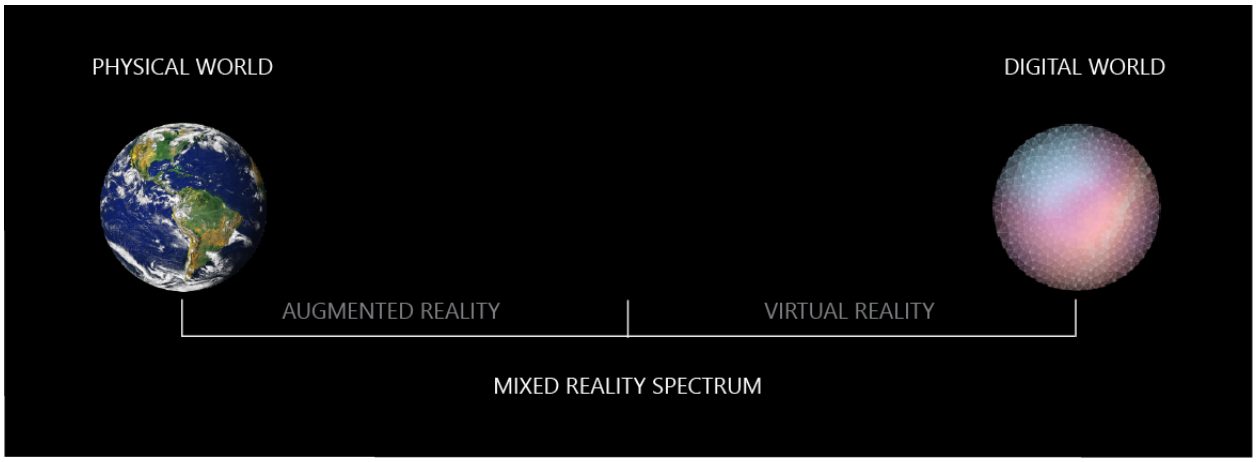

Figure 1.2: A visual representation of where Augmented Reality and Virtual Reality lie on the Mixed Reality Spectrum

1.5.3 Augmented Reality

1.5.4 Mixed Reality

As shown in the image above 1.1, Mixed Reality is generally agreed to be defined as the spectrum between the physical world and the digital world. The physical world is the one we live in every day, whereas the digital world obviously refers to the "worlds" that are generated by computers. For something to be fully a digital world would mean you are entirely immersed in the application and as far as you can tell the digital world is entirely real. There is no such application that exists today that would be considered a true digital world, so the term Mixed Reality aims to define this spectrum between the two extremes.

Extended Reality (XR) is an umbrella term that encompasses MR, VR, and AR. It is used in the name of the Unity XR framework as the framework is built to run on any headset. The Unity XR framework provides an outline for how functionality should perform and each headset is responsible for creating their own interface for connecting it with their device. This makes the framework very portable and makes developers lives significantly easier

Chapter 2

Development

In this chapter I plan on explaining the overview of how the project works. I will also explain and defend why I made the choices I did.

2.1 Overview

Figure 2.1 will be the main point of focus for this chapter as it provides a high-level insight to the project. The core purpose of this framework is to listen to the four UnityXR events we use in our projects at work and either log, or do something when those events occur. To do so, We have the four classes that subclass the UnityXR classes that handle these events. These are the DcGrabInteractable, DcSimpleInteractable, DcTeleportArea, and DcTeleportAnchor classes. When using this framework, Unity developers would use my classes in the same place they would have previously used the UnityXR events. To the developer it appears and functions exactly the same, with the only difference being the name of the class. This was important as if every developer on the team was forced to understand the framework it would likely result in significantly more bugs and issues. This way, the only thing the developers on the project need to know, aside from the one setting the framework up, is to use these classes the exact same way they would have used the UnityXR classes for any objects they want data to be logged for.

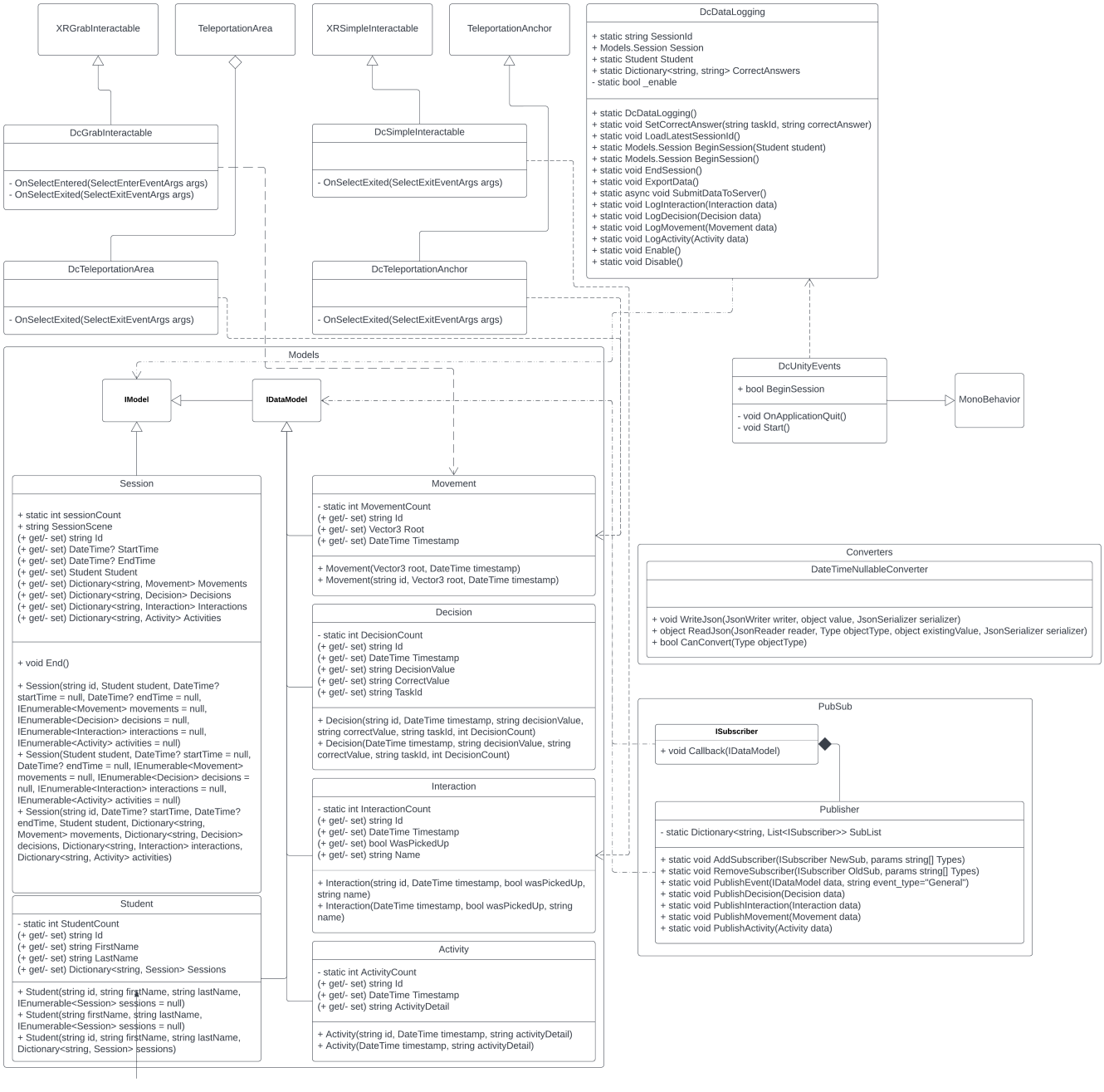

Figure 2.1: A UML diagram of the entire data collection framework

2.2 Models

To make things simpler to understand, I'd like to briefly explain some of the packages and classes and what they actually refer to in better detail. The Models package contains a variety of classes whose main purpose is solely storing data. I created 2 interfaces within this package solely for organization purposes so you could easily refer to any of the data collection models as a parameter or variable. I settled on 2 interfaces more for future-proofing than anything else. While most of the models only store data now, I added the IModel interface in case some Models require more functionality in the future. Currently the data is not processed on the headset, so until we have a need for processing headset-side they're mostly a neat way of organizing data. However, they do also provide a simple way of loading the raw exported data back into those classes if needed. The Session class currently has minor functionality and may see more implemented in the future so I decided it should directly implement the IModel interface. The other interface is IDataModel, which implements IModel, and refers to the rest of the models who have no methods besides a constructor and only store data.

2.2.1 Data Models

The 3 main components we track are the users Movements (tracked by DcTeleportArea and DcTeleportAnchor) which are any time a user moves somewhere within the scene. The next component is Decisions (tracked by DcSimpleInteractable), these are any time a user clicks a button on a UI within the scene. For example, we have a UI with quiz questions displayed on it in our project. Any time the user picks an answer, an Decision would be logged with their choice, the time, and what the correct answer was if there is one. Finally there's Interactions (tracked by DcGrabInteractable), these are any time a user picks up, or drops an object. We also use a Student object to store information on the user, each session will have one corresponding Student object.

2.2.2 Sessions and Activities

A Session refers to one instance of a user playing through a scene. A scene in Unity you can somewhat think of as a module or a level in a video game. Our project has multiple scenes, each with multiple tasks. So if a user plays 3 scenes in our project, they would have created 3 separate sessions. This helps us minimize the size of data stored in one session as if they replay a scene 20

times in one sitting, we wouldn't have 20 logs crammed into one file. The Activity class is used to refer to the multiple tasks within the scene, or anything that doesn't fit neatly within the other classes. It is designed to be more open-ended and general for storing any data I didn't account for beforehand. An example would be that one of our tasks tells students to measure a sea turtle with calipers and a tape measure. Depending on your needs you could log an activity after each fin is measured, or when the entire activity is completed, depending on the data you cared about. This is useful in analysis as we can see insights on specific tasks such as how long it takes to complete each task on average. We also plan on eventually using this data as the input to our progression system, so the program can determine what activities have been completed.

2.2.3 Converters

Since Converters is listed as a package on the diagram, it makes sense to explain it. However it is a very simple package so I will keep it brief. There is one class in the package which is a converter used when reading or writing any of the models to/from json format. This is because we use DateTime objects in our logging, which are not json standard. The reason it is its own package is more future-proofing as we may need other converters for different data types in the future.

2.3 Publisher Subscriber

Recently, the need for a progression system arose in our project. This would entail the user being limited to the tutorial when they first made an account, and each time they complete a module they would unlock a new one. In order to implement this, we needed to know when a user completed the modules somehow. This gave me the idea of implementing a publisher subscriber method to my entire data collection system so if any of the data wanted to be used at run-time it easily could. You can subscribe to any of the 4 events logged in this framework, Movements, Decisions, Interactions, and Activities. For ease of use, I also included a "general" topic which simply means you want to subscribe to all events regardless of topic.

To subscribe to the publisher you would implement the ISubscriber interface create a call-back function for the data to be sent to. Then you'd simply call the AddSubscriber method of the Publisher class and add the subscriber instance to whichever topic you wanted it to listen for. Any time any of the events are logged, any subscribers to that topic will have their callback method

called and be passed the data being logged. For the progression system, we only cared about the Activity event, as that's the only one required for progression. However, implementing the others were extremely simple. So this is another instance of attempted future-proofing for the framework. Seeing as I want it to be general use, it is important to me for the framework to account even for issues I don't experience myself.

2.4 Data Logging Methods

As mentioned in the overview 2.1, we have 4 main classes that handle data logging. There is one additional class, called DcDataLogging which also assists. DcDataLogging essentially acts as the main class for managing data collection at run-time. It contains the methods for actually logging the data which the other 4 classes call, and it also has methods for starting a new session, ending a session, enabling or disabling the entire framework, and more. I chose the approach of separating this logic into one class for ease of use. The 4 data logging classes are used as components in Unity which means they must be able to be instantiated. I wanted one easy to access control center, and dealing with instantiating and referring to the proper instance, etc. can greatly complicate the process. So DcDataLogging is a static class that can be accessed easily from anywhere.

Now comes the question of how we're actually going to store this data. I considered a variety of options, but settled on using json to store the data. Obviously when dealing with bulk data something like a database would likely be a significantly better choice. However, as this data is logged to the headset, I never want there to be bulk data because headsets have limited storage and processing power. Because of this, I plan to combine this framework with my coworker Seth Angell's Teacher Portal project. What this would look like is any time a session was completed, the program would attempt to send the data to the teacher portal to be stored in a database, and it could then be deleted from the headset. I say attempt to send the data, as not only may network conditions be unstable, but many of the locations the headsets will be going to such as schools will often have limited if any internet connection.

As the teacher portal is still a work in progress, this portion of the data collection is also. So in the current state the data must be manually pulled off the headset instead of it being automated. The functionality for sending the data currently exists, but the server receiving the database isn't complete. There also needs to be additional logic written for dealing with long periods of not having

internet while still making sure all files on the headset are uploaded.

2.4.1 Data Integrity

As with just about any program, Unity programs are prone to bugs, errors, and unexpected crashes. My original approach was to end the session when they complete the last task in the scene, and then log the data. I have since however revised this as I don't want the data to be lost if the user had to exit the scene before completion for any reason. My new approach is to actually export the data every time any of the logged events occur. This means that no matter what happens during the program, there will always be a file storing up to the last event before the program exited. While you could potentially extrapolate this information from the event logs, it also provides a simple indicator of whether or not the user completed the entire scene. Since the EndTime attribute isn't set until the session completes, if it is null then you would know the user did not complete that scene.

An obvious concern with outputting the data this frequently would be efficiency. This concern is also exaggerated by using json as you can't simply add a new element to the file, you need to rewrite all of the data to the file. Because of this, any code related to the game functioning is always ran before data logging methods are ran in the framework. This way even if the logging takes a bit longer than expected, it shouldn't interfere with the user experience. I have not done testing on the efficiency of the framework as a whole because when implemented to our project there was no noticeable difference in performance. This is certainly an area I could analyze more and improve on in the future, however it is currently plenty efficient for our needs.