Developing a Brain-Computer Interface to Enhance Storytelling in Games with the Identification of Cognitive States

A Major Qualifying Project Proposal
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In Partial Fulfillment of the requirements for the degree of Bachelor of Science.

By:
James Cao
Andrew Nguyen
Jagger Polvino

Date: March 20, 2024

WPI Faculty Advisors
Erin Solovey
Gillian Smith

PhD Student Advisor
Max Chen

Abstract

Brain-computer interfaces (BCIs) are a technology that has been incorporated into the medical field as a method of better understanding the human brain. Now, BCI is being tested in other aspects, including entertainment. Our project aims to combine BCI with augmented reality (AR) to generate a new experience for gamers. Our team created an AR application on the Microsoft HoloLens 2 using the Unity game engine. Our murder mystery game, *The Consultant's Case of Concentration*, combines traditional murder mystery elements with a fabricated focus state that alters the game experience for users based on their focus state using BCI. The outcome of our project is a foundation that other project groups can use to develop a full game.

TABLE OF CONTENTS

1. Introduction

Brain-computer interfacing (BCI) is a field that has developed significantly within the past few decades and has numerous applications, primarily in medicine and research. Another field of application for BCIs is in entertainment, specifically video games. In video games, there are multiple aspects that BCI can improve, one of the most popular being controllers. The problem with these kinds of BCI applications is heavy dependence on the neuroimaging technology correctly interpreting the active thoughts of players and are not always reliable for enhancing the player's experience with the game. However, there are other applications of BCI in video games that use brain data as an auxiliary tool for enhancing the game rather than a major feature. These applications use non-invasive neuroimaging techniques such as Functional Near-Infrared Spectroscopy (fNIRS) and electroencephalography (EEG) as non-intrusive methods of passively measuring brain activity to adapt the storytelling experience depending on the player's current cognitive state.

The integration of BCI into games is still a relatively new and developing avenue for improving storytelling in video games. Current research centered around BCI lacks the connection from BCI to video games, creating a gap in research between the design and implementation of video games, and BCI. Exploring the less studied neuroimaging technique, fNIRS, as a method to bridge this gap has created novel challenges, but also presents the opportunity to create a video game where users can experience the story differently depending on cognitive state.

The goal of this project was to take on this challenge and establish an infrastructure for the future development of AR games that adapt their narrative style based on cognitive data retrieved using fNIRS and identify avenues of development as well as challenges involved with integrating brain signals into a game. Our areas of background research included fNIRS, augmented reality, HoloLens, and game storytelling. By the end of our project, we developed a text-based murder mystery/investigation game where the given dialogue, clues, and choices are influenced by the focus state of the player, built to be used as a foundation for future BCI game development.

2. Background

2.1 Neuroimaging: EEG and fNIRS

In order to start our project, we needed to identify our options on BCI methods for obtaining cognitive data/neuroimaging. There are multiple potential methods of doing so, each with its own benefits and detriments. However, for this project specifically, there was a focus on non-invasive BCIs as this is the preferred method for neuroimaging within healthy users. This narrowed down applicable devices to either electroencephalography (EEG) or functional near-infrared spectroscopy (fNIRS), both of which are non-invasive and allow for live interactions (Solovey et al., 2021). In addition to featuring non-invasive neuroimaging techniques, both the EEG and fNIRS are portable and relatively low cost making them more feasible to use for our project.

EEG measures electrical activity within the brain caused by neurons firing electric signals as part of the nervous system. More specifically, it measures the thousands of neurons firing in sync from the cerebral cortex toward the scalp when the brain is active; it cannot detect small groups of neural activity, only large discharges all at once (Britton et al., 2016). It does this by using pairs of sensors called electrodes, with one acting as the main sensor and the other as a reference. They work in pairs to compare voltage differences and filter out interference caused by other sources of electricity, such as neuron activation for scalp muscles, eyes, tongue, and other organs (Britton et al., 2016).

fNIRS, on the other hand, works by monitoring blood-oxygen levels in a user's brain. The blood-oxygen level is the concentration of oxygenated hemoglobin (HbO) and deoxygenated hemoglobin (HbR) in red blood cells, and fNIRS measures them by emitting near-infrared wavelength light into the brain and detecting how much light was absorbed by the blood cells (Casey, 2020). Near-infrared light is specifically used because hemoglobin absorbs it, but other living tissue around it does not absorb it as well. One general pattern of HbO/HbR levels is when neural activation occurs in a subject, metabolism increases, which causes a dip in HbO concentration and a spike in HbR, before leading to a spike in the demand and concentration of HbO and a dip in HbR. As the subject calms down, both concentrations gradually shift back to their baseline values.

There are a few critical differences between EEG and fNIRS. EEG monitors electrical activity coming straight from the brain and only needs detector probes while fNIRS requires shining light into the brain and thus needs both light source probes and detector probes. Another difference is that each technique works on different timescales; EEG measures electrical signals on a magnitude of milliseconds, while fNIRS measures blood-oxygen level, which changes on a magnitude of seconds. EEG is therefore better at detecting quick or rapidly changing cognitive states. However, EEG detects when electrical signals reach the surface of the head, but not precisely where the signal originates from, while fNIRS detects HbO/HbR changes 1-3 cm past the surface of the head, across the entire detected region, meaning fNIRS has a better spatial sense of the brain for detecting more detailed brain activity (Li et al., 2022). Because of the faster detection time of EEG, it is more suitable for explicit input, where the user actively thinks/produces brain data for direct input to the system (e.g. thinking/visualizing "left" or "right" to steer a car). Meanwhile, fNIRS's spatial detection makes it more suitable for implicit input, where the system passively collects the user's brain data and makes inferences without direct user input (e.g. recording stress levels while driving a car and adapting the car's infotainment system based on the brain signal).

For our project, we used fNIRS for neuroimaging because it provided more detailed data on brain activity, as opposed to EEG which is better at detecting rapid changes in cognitive state. It's also more suitable for our purposes since our project used an implicit BCI system design; our project focuses on passively detecting and implicitly using cognitive states to enhance a game, not explicitly controlling a game with thoughts.

2.2 fNIRS Data Collection and Processing

An essential prerequisite for BCI is a means of streaming raw data and event markers from the data acquisition source and a presentation destination. This allows for real-time processing and visualization of the raw data, which is essential for feature extraction used in BCI (Brain Support, 2020). Data flows from the acquisition source to the presentation destination through a transmission control protocol (TCP) link or an internet protocol (IP) link. Through this link, the presentation destination receives raw data, triggers (time periods of interest), and timestamps from the acquisition computer in a digestible way.

Collecting and visualizing raw data requires two things; the devices to perform fNIRS, and an acquisition computer to display the data. The devices for fNIRS include a headset with sensors, such as the NIRScap from NIRx, and an fNIRS platform to collect data using the headset and sensors, such as the NIRSport2 from NIRx. For the acquisition computer, one such specialized data collection software, Aurora fNIRS from NIRx, obtains raw data, HbO, and HbR concentration changes obtained from the NIRSport2, and visualizes them in real-time in several display modes (Brain Support, 2020). To start collecting real-time data through Aurora, a montage configuration must be selected. A montage is a map that describes the layout of the probes on the fNIRS cap and highlights whether they are probes that are emitting light or receiving light. After a montage is selected, Aurora then prompts the user to complete a signal optimization, during which the NIRSport2 increases the source brightness in the probes stepwise until optimal signal amplitude for each channel is obtained (Brain Support, 2020).

The metrics reported include signal level, dark noise, and source brightness (Brain Support, 2020). The signal level is the voltage reading at a detector and reflects how well light passes through tissue from a source to a detector. If the average value is high enough, the signal level of the channel formed by this source and detector is marked as excellent, which is indicated on the montage in Aurora as green, or acceptable which is indicated by yellow. If the signal level is too low, it is marked in red as critical. In addition to this coloring scale, the actual signal level, measured in millivolts, is reported for each wavelength of light for each channel. Dark noise is just light that is not emitted by light source probes, such as ceiling lights. Source brightness is the level of light that is emitted by the light source probes (Brain Support, 2020).

After Aurora has received the data from NIRSport 2, the data is sent to the presentation destination where the data is processed with an analysis program, such as Turbo-Satori (TSI). This data is sent through a protocol called Lab-Steaming-Layer (LSL). LSL consists of various libraries and tools that allow for live data streaming. It is required that the software sending and receiving data are on the same network. In Aurora, LSL is automatically enabled (Brain Support, 2021). After Aurora and TSI are connected, TSI can begin to calculate and show HbO and HbR concentration values (Brain Innovation). The HbR data is shown with thinner drawing pens in the selected channel plotter but with the same color as the corresponding oxygenated signal. A properly functioning channel should show the HbO signal moving upward during brain activation while the HbR channel should move downward. The HbR signal, with a negative amplitude, should be about 10-20% of the amplitude of the HbO signal. To calculate in real-time for future time points, the HbO or HbR concentration values need a baseline of raw data. TSI initially shows unprocessed data before switching to the processed HbO or HbR data display as soon as the baseline is available. The default baseline is calculated from the first 200 received values, but the baseline can be altered using the “Hb/HbO baseline calculation period, Begin:” and “End” fields in the “Real-Time Analysis Setup” dialog within TSI (Brain Innovation).

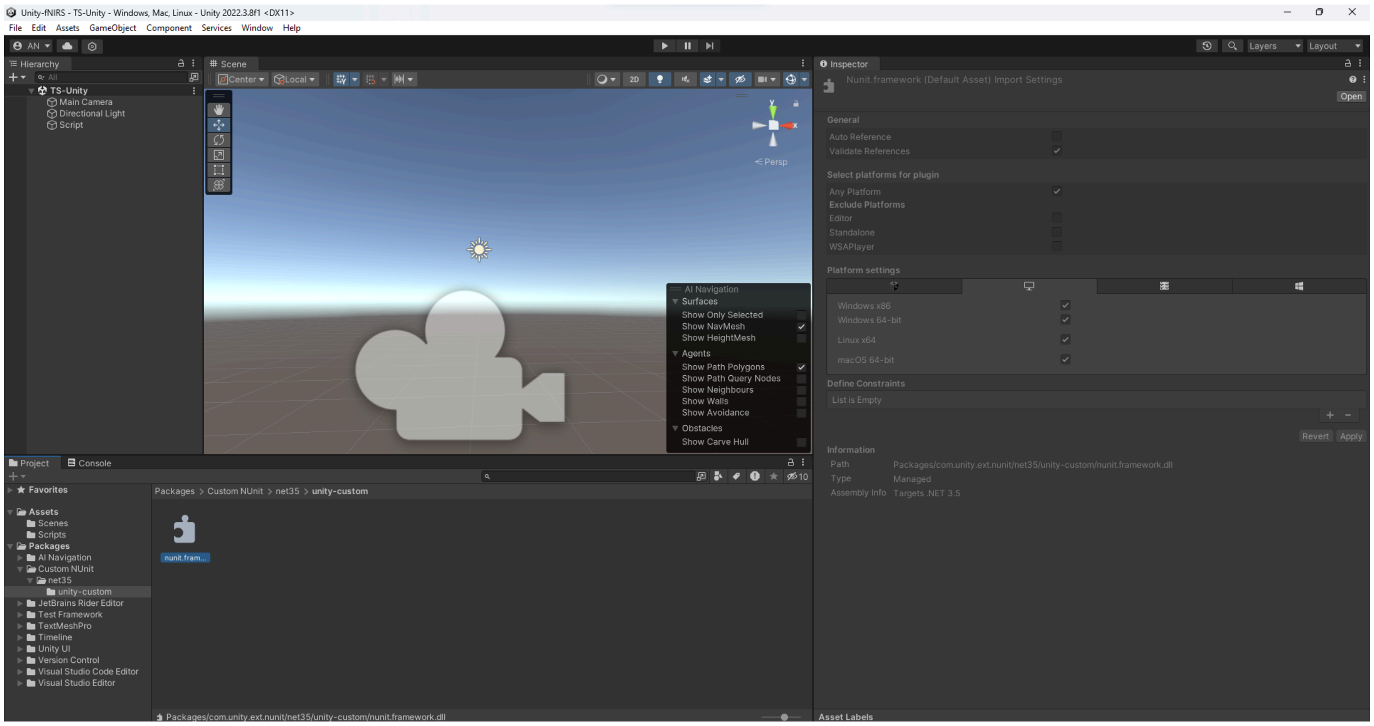
If there is too much data being streamed from the acquisition source to the presentation destination, the traffic over the network may cause a significant decrease in data processing speed and efficiency. To reduce the amount of traffic over the network, manually selecting the most important channels allows faster transmission of data. TSI allows this by extending the channel selection panel and selecting the necessary channels. The selected channels are color-coded in the plot windows so that they can be more easily identified (NIRx Medical Technologies Data).

Values for neurofeedback are calculated by subtracting the last baseline from the current scaled HbO and HbR values. The HbR values are inverted so that a negative deflection results in a positive feedback signal. HbO and HbR neurofeedback values are stored in text files by default, one for each time point, in a subfolder called “NeurofeedbackValues” which is placed within the folder containing the protocol (.prt) file. Each text file is named “Feedback-.txt” and contains 7 values: the precondition HbO-baseline value, the HbO feedback value, the calculated HbO level stored as an integer specifying the filling of the thermometer, the precondition HbR baseline value, the HbR feedback value, the calculated HbR thermometer level, and a value which specifies the current condition as an index value (NIRx Medical Technologies Data).

For our project, Aurora and TSI were integral for obtaining and processing the HbO/HbR data retrieved from the fNIRS headset. After the data was processed in TSI, it was then ready to be streamed to and interpreted by Unity.

2.3 Unity Game Engine

In order to accomplish our goal, we needed an environment suitable for developing a game, which we found in Unity. Unity is a game engine that provides game-building tools that render objects, create 2D or 3D environments, script through C#, and handle objects and inputs (Foxman, 2019). Games built in Unity consist of "scenes" that show the developer the current environments of the game. These scenes can be filled with game objects, which can be anything like lights, the camera, characters, items, etc. An example of a Unity project with a camera and light object in a scene is shown in Figure 1. However, to make a game object be something specific, a developer would have to give it properties or components that differentiate them from each other. The components would give the functionality and appearance of the object. By combining the features of objects, scenes, and scripting, a developer can create a game with a purpose. Unity also has many features that allow different assets and packages to be implemented into the project that allows different features. One of these is the Mixed Reality Toolkit which allows Unity to create a game with their scene in virtual or augmented reality (Unity Technologies, 2023).



Nunit.framework (Default Asset) Import Settings

General
Auto Reference ☐
Validate References ✓

Select platforms for plugin
Any Platform ✓
Exclude Platforms
Editor ☐
Standalone ☐
WSAPlayer ☐

Platform settings
Windows x86 ✓
Windows 64-bit ✓
Linux x64 ✓
macOS 64-bit ✓

Define Constraints
List is Empty

Information
Path Packages/com.unity.ext.nunit/net35/unity-custom/nunit.framework.dll
Type Managed
Assembly Info Targets .NET 3.5

Figure 1. Example of Unity Project with Light and Camera Object

2.4 Augmented Reality

Augmented reality (AR) allows users to use a device to view new things that are not present in everyone else's eyes while not blocking out the rest of the world around them. Often, AR is confused with virtual reality (VR) which manipulates the entirety of what the user sees rather than enhancing the current view (Ciena, 2016). A modern and popular example of AR in games is Pokémon Go. This smartphone game allows users to view pokémon in the real world through the user's phone camera (Niantic, 2023). The figure below shows how Pokémon Go displays this through a smartphone.

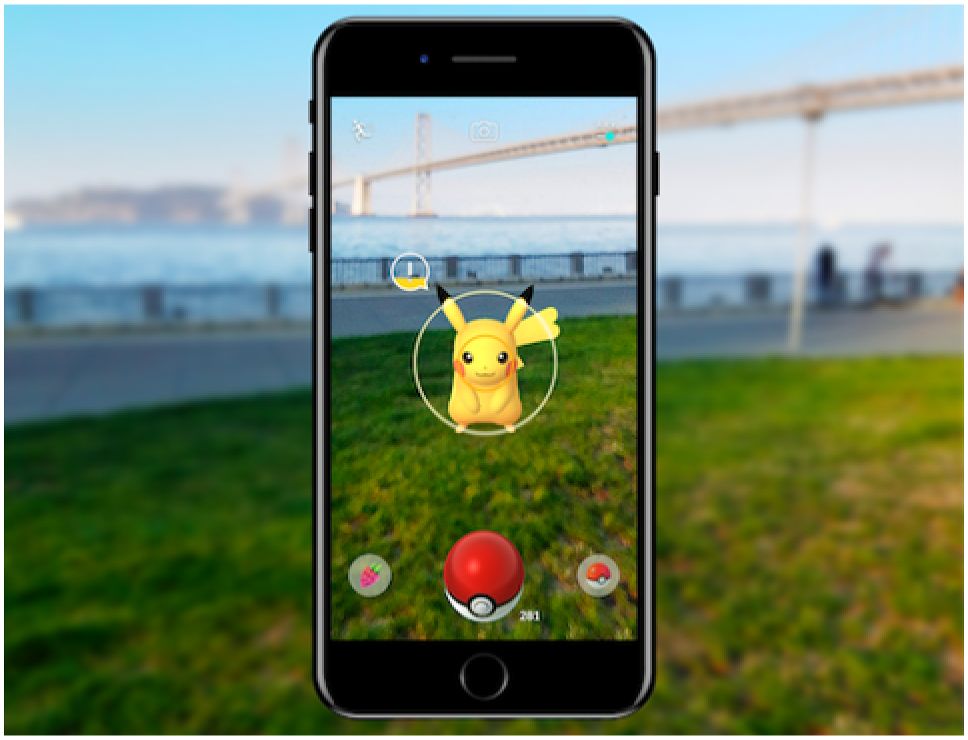


Figure 2. AR in Pokemon Go (Niantic, 2023)

AR games typically add virtual game objects into the environment of the player and allow them to interact in some way (Wright, 2023). To play an AR game, players can either open the game through their smartphone or a dedicated headset. Playing an AR game through a smartphone is much more popular and easily accessible, whereas a headset is more expensive. However, a headset would provide a much better experience as the player does not have to view through the bounds of their phone screens (Wright, 2023). To implement a game made in Unity into AR, we chose to use the Mixed Reality toolkit package.

2.5 Mixed Reality Toolkit

The Mixed Reality Toolkit (MRTK) is a package for Unity that provides the project components and features of an AR game environment. The toolkit allows developers to have a framework that gives them the ability to swap out components (Microsoft, 2020). MRTK also includes an in-editor simulation in Unity which allows developers to see changes as they are happening in real-time. The toolkit also provides templates of objects that can be used and viewed in AR through a headset (Microsoft, 2022). The figure below shows some templates with features that can be made with MRTK.

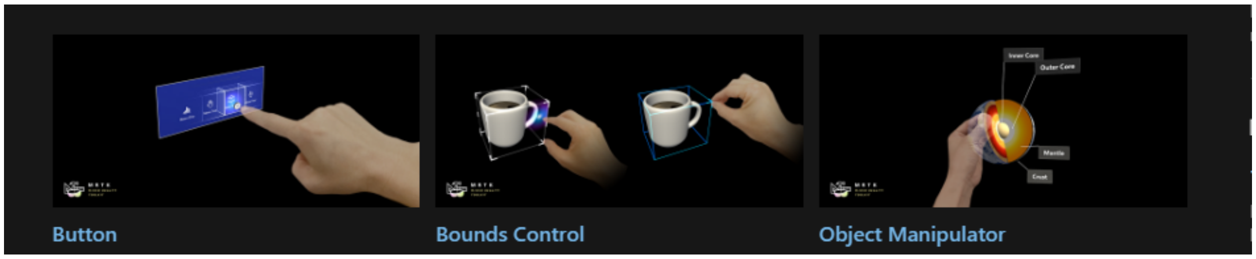


Figure 3. Examples of MRTK Features (Microsoft, 2022)

A headset provides a more immersive experience by displaying visuals in 3D (Wright, 2023). In this case, we decided for our game to use the Microsoft HoloLens 2 as it is compatible with Unity applications.

2.6 HoloLens

The Microsoft HoloLens 2 is a second-generation "untethered holographic computer," meaning that it is a standalone computer in the form of a visor attached to a headband that uses sensors and holographs to display information to its users (Microsoft, 2023). The HoloLens simulates AR by projecting 2D and 3D objects onto its visor and uses a mixture of head-tracking, hand-tracking, and eye-tracking to allow the user to manipulate the projections despite them not being physically present. Using the HoloLens with MRTK has proven to work in previous Major Qualifying Projects (MQP) such as *Sewn Into Memory: Reliving Memories Through An AR Quilt*. This project used Unity to create an application for the HoloLens using MRTK specifically features such as buttons and input text boxes onto their Unity Scene (Jones et al. 2023).

2.7 Storytelling in Games

In order to develop our game, we needed to understand how storytelling would affect our game. Storytelling in games enhances the overall experience by keeping the player engaged. Important elements of all storytelling include narrative, world-building, and character development. Video games have specific aspects that build on the story such as gameplay, visuals, environment, and level design (Porokh, 2023). Through gameplay, players can interact with the visuals, environment, and level to progress through the narrative (Arjorant, 2017). Worldbuilding can be defined as a process of creating an imaginary world and can be described through graphics, text, and design. It is an important step to begin creating a story as it sets up rules, the setting, visuals, and science (Reid, 2020). Character development can enhance the game story through how characters, including the player, affect the plot through interactions within the world (Kim et al. 2021).

3. Methodology

3.1 Goal

The goal of our project was to establish an infrastructure for the future development of AR games that adapt their narrative style based on cognitive data retrieved using fNIRS and identify avenues of development as well as challenges involved with integrating brain signals into a game. To accomplish this goal, we identified three objectives; (1) establish a connection from the fNIRS cap/sensor to a Hololens app, (2) design and program the AR game in Unity, and (3) integrate the BCI functionality of the fNIRS cap into the AR game. The first two objectives had to be completed before the third as the third involved combining the results of one and two.

3.2 Objective 1: Establish a Connection From the fNIRS Cap/Sensor to a HoloLens App

To establish a connection between fNIRS and HoloLens, there were multiple intermediary steps required (see Figure 4). The first connection was between the NIRSport2 device and the software, Aurora. To establish this connection, we needed: a computer, a NIRSport2 device, and a separate WiFi router or a private WiFi connection. The reason why a private WiFi connection was advised was that on a public WiFi connection, there was too much interference from other devices that were connected to the WiFi, and thus made connecting Aurora to the NIRSport2 device difficult. Both the NIRSport2 device and the computer running Aurora had to be on the same WiFi to connect.

Once this connection was established, the next connection was between Aurora and the software that could analyze the data from Aurora, which in our project was the software Turbo-Satori (TSI). Aurora and TSI were run on the same computer for our project and were therefore always sharing the same WiFi source alongside the NIRSport2.

When launching TSI, the software asked the user to choose between “Real-Time”, which would take real-time data from Aurora, or “Simul-Mode” which would prompt the user to input a pre-existing dataset that was saved on the computer that was running the TSI software. After connecting both Aurora and TSI, the data was then sent from TSI to a Unity test app made for obtaining data from TSI, which in our project was done through the same WiFi. We also had to connect the IP address of the computer running TSI to the computer running Unity.

After successfully streaming data from TSI to Unity, we then deployed the Unity app onto the HoloLens, where data would also be streamed successfully so long as the deployed app still used the IP address of the TSI device and the HoloLens was connected to the same WiFi source as the TSI device.

3.3 Objective 2: Design and Program the AR Game in Unity

Our next step was to design and develop the game itself. This included both writing the story and programming the game.

Our first step to complete this objective was to determine what kind of game we were making, the gameplay it would include, and an outline of the story. After that, we split tasks between drafting the story and creating the Unity/AR user interface (UI). Once we finished drafting the story and had it reviewed by our advisors, we started creating a Python story test file to simulate the cognitive states and choices a player can make when playing the game, since at that point the Unity UI was still a work in progress. After we finished creating the Unity UI, we used the Python file as a guide to create a framework for how the story would progress in the game.

3.4 Objective 3: Integrate BCI Functionality From fNIRS into the AR Game

After we set up the BCI components/connections and created the base of the AR game, our next step was to combine the two. Using the connection formed between the fNIRS cap and the HoloLens app from Objective 1, we set up a connection between the fNIRS cap and the AR game on the HoloLens. After that, we created a fixed set of HbO and HbR thresholds that would determine the focus state of the player: if the player had HbO and HbR concentration levels above that threshold, they were considered focused, and below that threshold, they were considered unfocused. Once we defined that threshold, we determined points throughout the game where the player's HbO and HbR were compared to the thresholds for the game to give certain responses. Figure 4 illustrates the data flow in the proposed methodology for integrating fNIRS into the AR game.

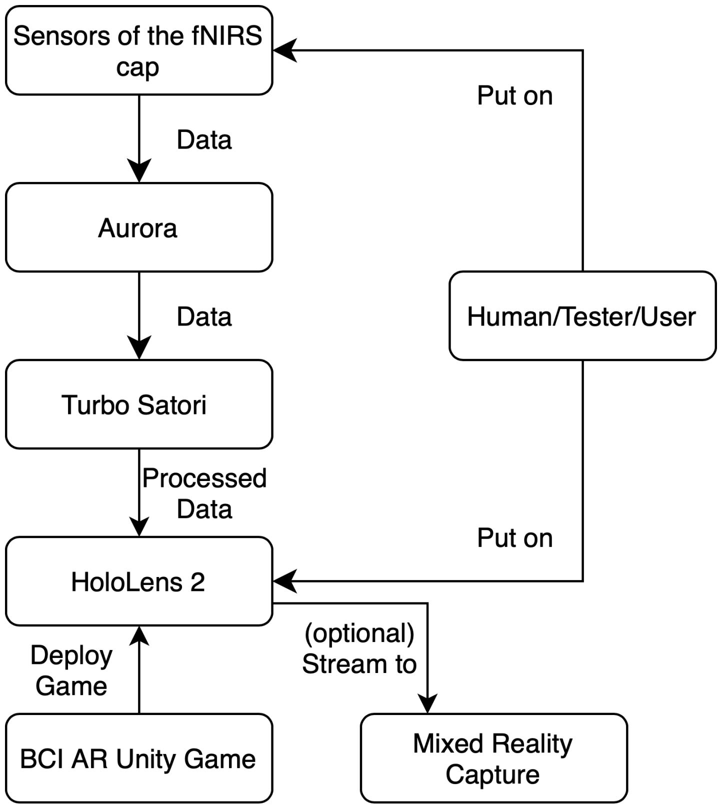


Figure 4. Flow of Data Between Components

4. Deliverables

In this section, we present three key deliverables: a guide/README on how we set up connections between devices and software, the AR game itself, and a basic datastream between the fNIRS devices and HoloLens which future projects can use. Each deliverable corresponds to an objective from our methodology.

4.1 Device and Software Connections

We have created documentation as a part of a README on GitHub (Appendix C) on how to connect TSI to Unity, read data from TSI in Unity in real-time, as well as connecting Unity to HoloLens. This documentation is incredibly important since there were many devices and software that had to be connected to begin setting up the BCI functionality of the game and learning how to connect them took a considerable amount of time.

On one computer, we had Aurora and TSI running with the sensor cap sending data and then TSI sending to Hololens. These connections allowed us to stream and view HbO and HbR data in real-time on the device that would run the game.

To avoid interference with other devices on the network and to avoid security firewalls built into public networks, all devices used are connected to a single private wireless network. The data generated from the NIRScap/NIRSport is streamed to Aurora before being sent to TSI for pre-processing. We chose Aurora for obtaining fNIRS data because its method of connecting to the NIRScap/NIRSport through a network was easy to set up and use. It also allows fNIRS sensor configuration customization, which we configured to use with a prefrontal cortex montage, as that is the area most associated with detecting levels of concentration (Causse et al., 2017). Aurora connects to TSI through the LSL protocol. The protocol is implemented on Aurora and imports the raw data to TSI to perform the real-time analysis ("Turbo-satori: real-time analysis software", n.d.).

We used a Unity project called Unity-fNIRS that connects TSI to a scene and presents the HbO and HbR data that comes from the fNIRS (Chen et al., 2023). This Unity project utilized three C# files to do this. The first file was the TSIMessage file, which contained the object class information for a message being sent, including information on the header message, the message request, its size in bytes, and the response. The second file was the TSINetworkInterface file, which allowed the streaming of data from fNIRS to Unity. It did this by setting up the IP address of the Unity scene and requesting data from TSI, before receiving the data from TSI via a TSIMessage. The third file was a Visualization file that obtained data from the TSINetworkInterface file and displayed it to the Unity scene. This script is still able to run on the HoloLens and do what is intended after the Unity project is pushed as an application. This Unity project served as the basis for implementing the story of our game so that it reacted to the live data streaming.

4.2 Creating The Consultant's Case of Concentration

To create the game story, we discussed ideas for stories that could be used in a game that incorporated BCI. Some of our ideas included a bartender story, a murder mystery, or an escape room story. Ultimately, we decided on a murder mystery story that had branching dialogue based on the focus state detected by the fNIRS device. The dialogue would branch based on the focus state of the player which affected the outcome of the murder mystery. We compiled the game story dialogue into a CSV file to be read through the game's code.

4.2.1 Game Concepts

The main concept of our BCI AR game was to use fNIRS data to enhance the player's experience with the game throughout the story, as well as incorporate real-world interactions alongside the virtual interface. In order to do this, we needed to choose a game genre where cognitive state and interactions with real-life objects could be integrated effectively. Our initial ideas were a bartending game, a murder investigation/mystery game, or an escape room game. These game types allowed for a focus on storytelling as we took an entirely text-based approach to convey information to players. Additionally, each of these stories created opportunities for us to integrate brain signals from the fNIRS into the storytelling of the game as they had discrete interactions for the user to experience.

The escape room game would have the users playing as someone trapped in a room, with clues on how to escape being given to them by a narrator. The BCI integration idea was if the user was in a focused brain state, they would receive clearer and more concise clues about how to escape. Conversely, if the player was in an unfocused brain state, they would receive more convoluted clues from the narrator, making escape more difficult.

The bartender game would have the user playing from the perspective of a bartender and would interact with various NPCs sitting at the bar. The BCI integration idea here was that if the user was more focused on their interactions with the NPC, then they would have been able to obtain more information from the conversation. On the other hand, if the user was not focused during their interaction with the NPC, then the conversations would be more bland.

The murder mystery game would have the user player aid a detective in a murder case, investigating clues/details to deduce the murderer. For BCI integration, if the user was more focused when listening to the detective's clues, they would get more relevant or useful details, and would conversely get less or misleading details if unfocused.

We decided on the murder mystery concept because we felt the bartending game and the escape room game would not tell a story as well as a murder mystery as the other two lacked direction story-wise and felt more arcade-like. In addition to having a clearer direction that the story could take, given that the chosen brain states were focused and unfocused and that being unfocused is easier than focused, we believed that a murder mystery game would garner more focus from the user, resulting in a better balance between focused and unfocused throughout the game.

4.2.2 Creating the Story

Once we had our chosen game concept of murder mystery, our next step was to write the story. In order to create a story, we needed to create both a setting and characters. We decided to use a 1930s London setting for a few notable reasons. The time between World War I and World War II was known as a golden age for detective fiction due to changed reading habits caused by WWI: previously, detective fiction had more focus on non-violent crimes such as theft, but after WWI, murder mysteries became more common and popular (Link, 2023). The main structure of detective fiction from this time, known as clue puzzle/whodunit, was also formatted in a way that allowed readers to follow along with the investigation and make their own connections between clues as if they were also investigators in the story, similar to playing a detective game (Link, 2023).

The next step was to determine what kind of murder mystery we wanted to create. Murder mystery/detective-type stories are generally composed of two sub-stories, the crime itself and the investigation. The story of the crime is what actually happened during the crime, while the story of the investigation is the following events uncovering the story of the crime. Depending on which of these two sub-stories the murder mystery focuses on, the game can be either a "whodunit," which focuses on the story of the crime, or a thriller, which focuses on the story of the investigation (Larsen et al., 2020). We decided to go with the whodunit style since it was the simpler style to write, and is better suited for a puzzle game that evokes the player's focus to solve said puzzles.

In terms of characters, there are five, not including the player: the detective, the murder victim, and three suspects. The detective's role was to act as the narrator and main speaker of the story. The victim is the murder victim whose case the detective is investigating. The three suspects are civilians believed to have murdered the victim due to motivations or specific circumstances, which are stated in clues the detective gives to the player. The player's role is of a private consultant aiding the detective in sorting out the clues to figure out who was the real murderer.

After developing our settings and character, our next step was to create a script. We first created an outline, roughly split into a beginning, middle, and end. The beginning was used to introduce the setting, characters, and the player's role. The middle was composed of details/clues that the detective would share with the player and ask for their opinion on them. The end was wrapping up the clues, accusing one of the suspects of being the true murderer, and showing one of three endings. Figure 5 illustrates the full story flow.

The most important part of the story was the clues, and they were also the parts that we decided to incorporate BCI aspects into. For each clue, it would generally follow a pattern: the fNIRS headset would record the HbO/HbR levels of the player while the detective would give some general details about the clue, the fNIRS would stop recording and average out the oxygen levels during that time, and the detective would give more details, the quality of which would depend on the focus state. A focused state would give more accurate details while an unfocused state may give misleading or vague details. Afterward, the player would be given some choices, with the choices given dependent on the focus states detected earlier. Once a choice is selected, the detective comments on the choice selected before moving on to the next clue.

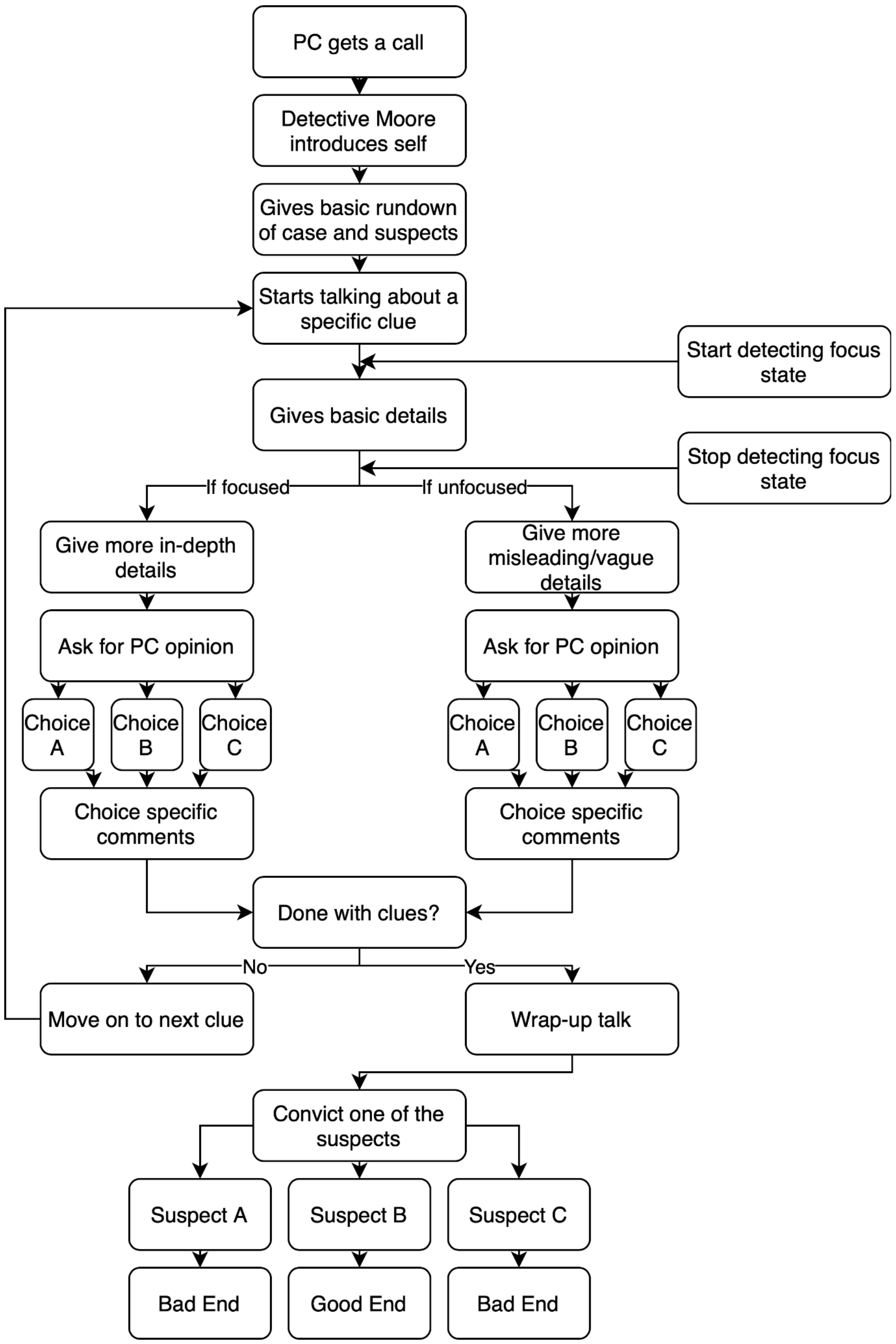


Figure 5. Flow of Story/Game

In addition to having three different clues for the player to investigate throughout the story, we also incorporated physical objects for the player to interact with as a part of the story. We decided to use these objects to take full advantage of the AR functions of the HoloLens 2 device and to add a unique gameplay element to the game. Prior to beginning the game, the player would be informed by us that the game contained physical objects for them to investigate, but not explicitly told what was in the box. The player would eventually be prompted by the detective to investigate the knives found at the scene of the crime in the game, and in reality, there would be a labeled box that contained the knives for the player to examine (shown in Figure 6). Once this event is triggered, the player would see a screen with the text, “Have you finished examining them?” and two options, one that says “Yes, I’m done”, and the other “Gimme another minute.” If the player chose the first option, then the story would continue as normal. If they needed more time, then the options would loop with the text “Alright, I’ll give you some more time” until the player was ready to move on. We also included a receipt in the box for the player to examine, which had the same dialog options as the knives. The receipt can be seen in Appendix B.

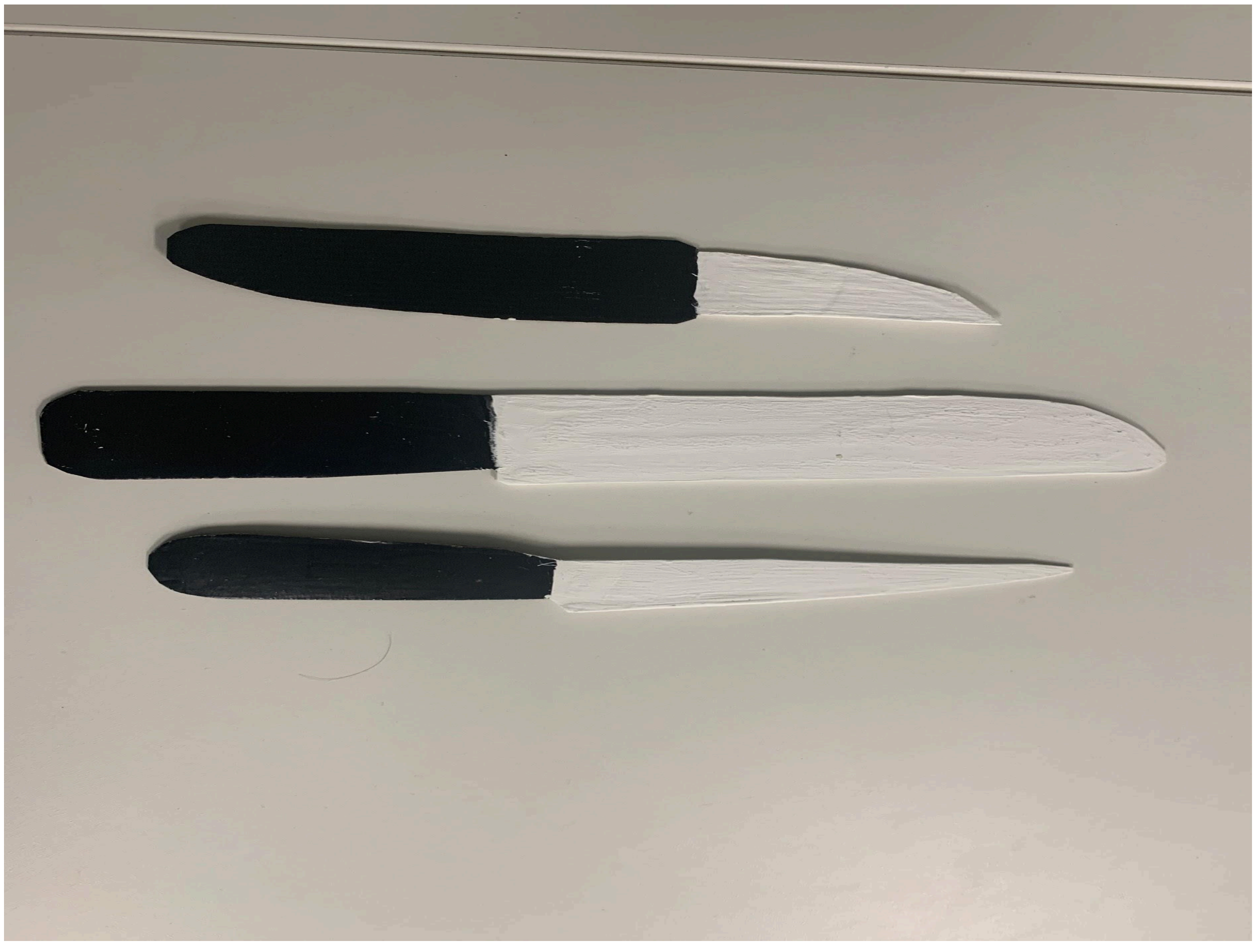


Figure 6. Cardboard Knives used for the AR Game

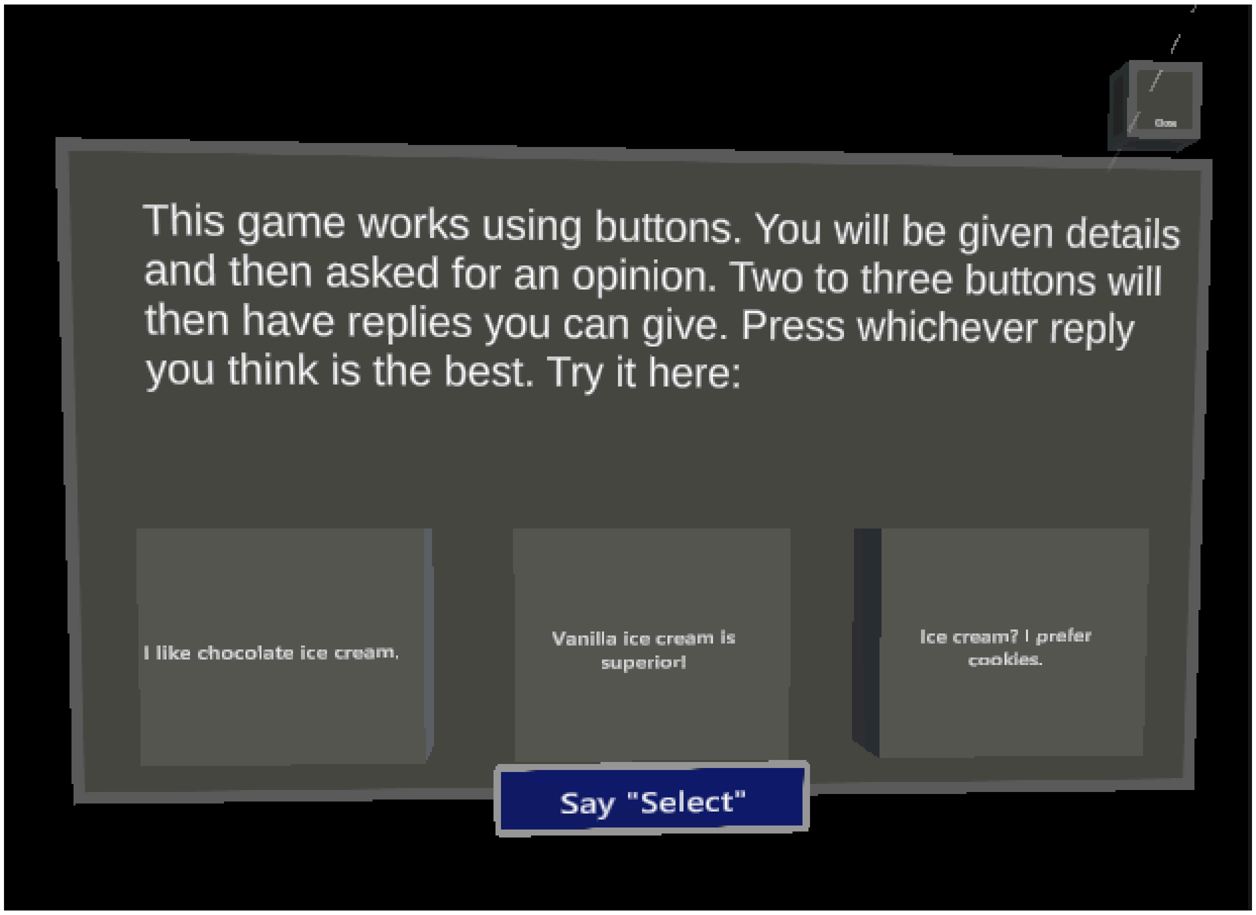
4.2.3 Programming the AR Game

Once we finished drafting the game's story script, we decided to create the game in two iterations: the first iteration in Python as an I/O text demo, and the second iteration in Unity as the final AR game. The Python iteration was made to be a prototype for what the story would look like in the final game, and the purpose was to simulate the cognitive states and choices a player can make when playing. At first, we hardcoded the story into the Python script but decided to use a modular system to make adding more to the script for future work easier and more comprehensive.

We formatted the story script into an Excel spreadsheet, where the story progressed row by row, and the first column of each row would state what purpose the row would have. The Python script would then read that Excel file row by row, using the first column (index 0) of each row to determine what action should be taken for the row. Some example row types include dialogue for simply printing dialogue, choices for players to select, comments for documentation not meant to affect the story, and state for prompting the player if they are focused or unfocused. This system allows easy editing for the story, and its modularity came into play when we decided to add a tutorial for the game directly into the spreadsheet. All we had to do was shift all the rows down and add new ones for the tutorial at the top, demonstrating all the possible types of dialogues and responses the player would read and make throughout the game.

One of the more important aspects of the spreadsheet was the point_change row type, which was used to control who the player and detective would convict as the true murderer at the end of the game. Each suspect has an integer value assigned to them which could be called a suspicion value. These values all start at zero, and each time a point_change row is read by the Python script, those values are changed based on the values in the row. Some choices are weighted heavier than others, depending on the detective's own beliefs and thoughts about the response the player chose; if the detective agrees that a certain clue is indicative of a suspect's connection to the murder, more points would be added to the suspect's suspicion value. At the end of the game, whoever has the highest suspicion value would be convicted as the true murderer, and the ending would also be determined. Convicting the correct suspect would lead to the good end while accusing either of the other two would result in a bad ending. After the Python script was completed, it was used as a guide to create a similar modular framework for the Unity game.

Both the Python and Unity code would read the story script from the Excel spreadsheet. However, Unity could not read from an Excel spreadsheet without needing to install additional Microsoft plugins/libraries, so we had to convert the story script Excel file into a CSV using semicolons as delimiters. Since the Unity code was also pulling from the same spreadsheet file (in CSV form) as the Python demo, it read the script in a similar way to the Python demo: row by row, using the first column to determine the course of action for the row.


This game works using buttons. You will be given details and then asked for an opinion. Two to three buttons will then have replies you can give. Press whichever reply you think is the best. Try it here:

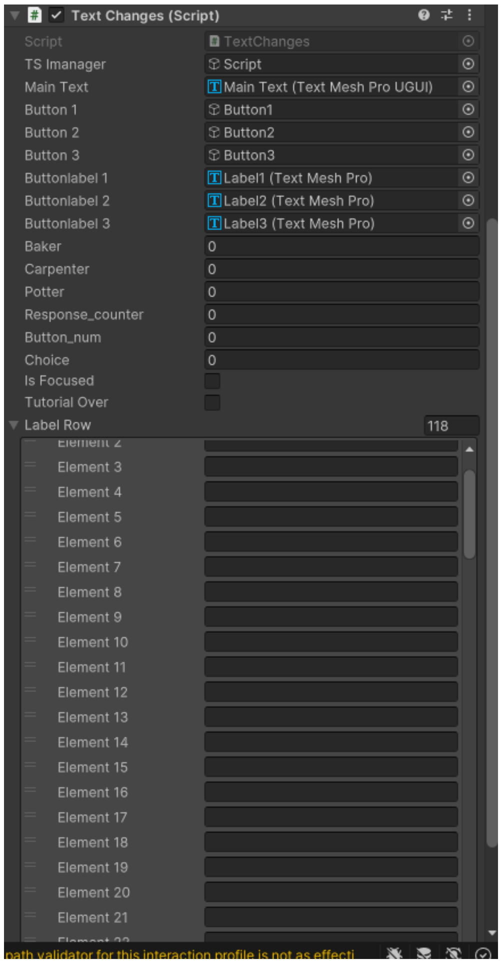
I like chocolate ice cream.
Vanilla ice cream is superior!
Ice cream? I prefer cookies.
Say "Select"

Figure 7. Game UI

In our Unity project, we created a script in the file TextChanges.cs where the functions of the game and its buttons are located. Within the TextChanges.cs file, the function start() runs immediately as the game begins. Additionally, we set a default focused state to true that runs concurrent to start() and is also where the buttons are enabled or disabled based on three different things: check whether there is text in them in setButtonStatus(), finding the CSV file in the “StreamingAssets” folder, and putting the rows and columns into a 2D array. The “StreamingAssets” folder allowed an asset to be pushed into a build and onto the HoloLens. Without this folder, the CSV would not be included in the build and thus the game would not display any dialogue. Additionally, a “TSINetworkInterface” object was used to initialize the start function which helped retrieve information from the fNIRS device that allowed the code to determine the state the game was in.

We used many global variables to help with the main functions of the game, the most used ones being arrays labelRow and csvValues. The labelRow variable was used to differentiate between rows. For example, the labelRow variable would differentiate between a row named “comment” and a row named “dialog”. The csvValues variable contained the entire CSV and was used to display the text. Additionally, we created variables for each button and their corresponding labels. One variable name for the button included “button_num” which tells the function play_turn, which is described later, which of the three buttons pressed. There were also variables for each NPC and their corresponding point values. The point values assigned to each character would increase or decrease based on the choices the player makes.

Within the CSV, we had a label called choice_3 for when the player was presented with three different dialog options. When the player is presented with branching options, each option has a different line of dialog related to it, and the purpose of the variable “choice” was to keep players within this line of related dialog. We also used a “strsplit” variable which is a string array used to handle point changes. For specifically the tutorial portion of our game (illustrated in Figure 7), choice_3, and path_3 were coded slightly differently. In the tutorial, the player is given three different choices, however, there is no path of dialog related to the choices that the player makes unlike in the actual game. This means that the path_3 label did not need to be saved from the tutorial and instead was saved throughout the game as a way to present the player with the appropriate ending at the end of the game. We incorporated a boolean called “isFocused” to tell us which state the player was in and would display different text based on the state. Also, we created a variable from the library, System.Random, called “rand” which randomized the state to be either focused or unfocused whenever the game reached a state row in the CSV. The reason why we included the random variable was that we ran out of time to use the fNIRS device on users to test the game. So we decided to emulate changes in focus state using the “rand” variable. Finally, we used an integer variable called “response_counter” to keep track of where in the CSV the game was. The setup of the variables on Unity can be seen in Figure 8.



Text Changes (Script)

Script	TextChanges
TS Imanager	Script
Main Text	Main Text (Text Mesh Pro UGUI)
Button 1	Button1
Button 2	Button2
Button 3	Button3
Buttonlabel 1	Label1 (Text Mesh Pro)
Buttonlabel 2	Label2 (Text Mesh Pro)
Buttonlabel 3	Label3 (Text Mesh Pro)
Baker	0
Carpenter	0
Potter	0
Response_counter	0
Button_num	0
Choice	0
Is Focused	
Tutorial Over	
Label Row	118
Element 2	
Element 3	
Element 4	
Element 5	
Element 6	
Element 7	
Element 8	
Element 9	
Element 10	
Element 11	
Element 12	
Element 13	
Element 14	
Element 15	
Element 16	
Element 17	
Element 18	
Element 19	
Element 20	
Element 21	
Element 22	

path validator for this interaction profile is not as effecti

Figure 8. Unity Inspector Code Setup

Our game buttons all contained two functions set up as events in the Unity inspector. Each button had a set button function, where each button was assigned “setButton1()”, “setButton2()”, or “setButton3()” with the first button on the left being assigned “setButton1()”, the middle button assigned “setButton2()” and the button on the right assigned “setButton3()”. Then, each button had a “playTurn()” function. The event setup of one of the buttons is shown in Figure 9. The “playTurn()” function goes through the CSV values stored in an array from “start()” and uses “response_counter” as an index to know where in the array to read from. The “playTurn()" function contained a switch case that looked at the name of the label and went through the switch case. For the dialogue label, the dialogue that is associated with the text in the same row within the CSV is presented on the scene. For state_dialogue, dialogue is shown based on the focus state the user is in. State_path_3 is the response dialogue to the choice. Path_2 shows a response dialogue that only occurs with two options. The “wait” label is meant to allow players more time to interact with the physical objects that we created for the game. The player can choose to move on or choose the option for more time. If the player chose more time, this led to “wait_response” which decreased “response_counter” each time the player pressed the option for more time to keep the “response_counter” index in the same spot. The decrease is necessary because, after the switch, the index increased by one no matter the case. The second integer for the 2D array can be different from the first one based on choice and case, see Appendix B for information on what column number would be used based on case. Some events may happen at the same time. The “accuse” case displayed text based on the point values from earlier responses. Path_3 also includes a “tutorialOver” boolean. Before the tutorial, it acts just like state_path_3 without the states where it shows a response due to a choice of three. After the tutorial, the boolean is set to true and path_3 shows dialogue based on the story path that is occurring. Some events need to occur at the same time as other events. For example, buttons being set and displayed at the same time as the dialogue or point changes needing to occur after the player has made a choice. If they were separate events, there would be moments where some actions required a double-click to progress or have unintentional skips in the reading of the array. So choice_3 and choice_2 appeared in the dialogue and state_path_3 case in an if statement asking if that was next in the array. To avoid the double-click and unintentional skip errors, 1 was added to the index to skip over the line in the CSV. Other rows needed to be skipped as well, specifically the comments in the CSV. To handle this, we added the response counter so that the next button press that calls on play turn is past the comment and at the right place in the array. This skipping also happened for columns in two different scenarios. The first is when the columns must be read because of the choice made by the player and the state they are in. The solution to this was making the column text “skip”. The second scenario happens when the column that is supposed to be read has the text “pass 4” which increases the response counter by 4 instead of 1 which skips the next 4 rows. Now point_change rows are skipped. However, the actual point change happens in state_path as that is the response of the dialogue with choices. In the state_path case, it checked if point changes were in the array before it and got the button that was clicked. Then based on the state, it looked at the column and used strsplit to separate the words and numbers. Lastly, the numbers were added to the point value corresponding to the name of the person that came before the number in the column.

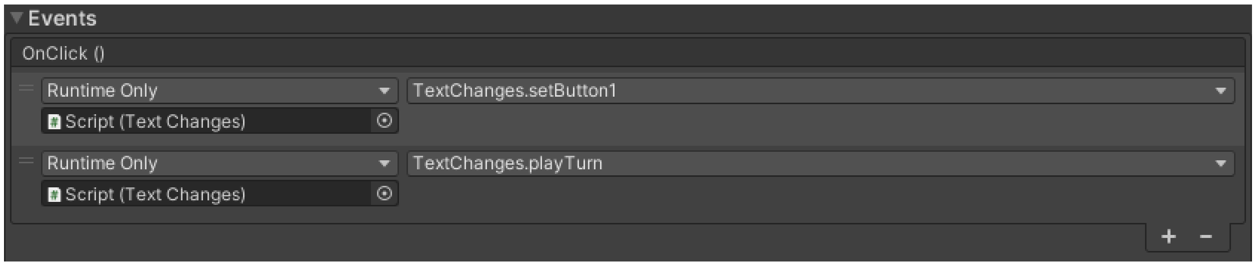


Figure 9. Unity Inspector Button Event Setup

A modular system has the advantage of being easy to build upon since the entire script of the story is in a spreadsheet that progresses through the story row by row. This made inserting new rows into the middle of the script or adding more dialogue or other details to the end of the script a straightforward task. This allows others to either continue this project or adapt the script system and code for another project.

By the end of the project, we created a fully operational text-based adventure AR game called *The Consultant's Case of Concentration* that can be played on the HoloLens 2 device. The dialogue and options would be different based on how focused the player is, similar to a real-life case of listening to someone on the phone or not paying attention to them. We created the game in Unity with a slate and three buttons for the options. The buttons used C# code to navigate through the story. In addition, we created physical objects that the player can interact with to make the game feel more like augmented reality than just a normal video game. We never implemented the fNIRS brain data into the game and as a result emulated state detection with a random number generator instead. This feature of emulating the state of the player can be used to test the functionality of the game without connecting the brain sensor or reading brain signals. Videos of playthroughs of the game through the HoloLens and Unity can be viewed in Appendix D.

4.3 BCI Dataflow

After we completed creating the game in Unity, we connected all devices and software together to form a complete flow of BCI data from the fNIRS headset to our game on a HoloLens device. As shown in Figure 4, The Unity game completed for objective 2 was deployed to the HoloLens, and the connections used in objective 1 were applied to the game. The fNIRS headset (through the NIRSport2) sends raw data in real-time to Aurora for interpretation/formatting, which is then sent to TSI for preprocessing and analysis. When started up, the game on HoloLens then connected to TSI, assuming that both the HoloLens and the device running TSI were on the same network and the deployed game had the IP address of the TSI device.

By the end of the project, we created a connection between TSI and the HoloLens game, and data was streamed between the two, but the game did not directly use the data streamed. Our intention was to look at the streamed data, process/average the HbO/HbR data obtained, and determine whether the user was focused or unfocused based on whether the processed data was above or below a threshold. This is something future groups or projects can implement if they are also detecting cognitive states, or disregard and use the streamed data for other purposes.

5. Discussion

5.1 Challenges

5.1.1 Initial Challenges

One of the major challenges that we faced as a group when starting this project was that we collectively had very little knowledge about the technology that we would be working with as our initial research was more surface-level. As a result, we spent the majority of the first few weeks of our project doing in-depth research into what exactly the fNIRS device could do and how it worked. The major challenge here was that there was a lot of dense information and it was only on the NIRx website, the company that made the fNIRS device that we were using, where we could find all the information we needed in one place. However, to access the website, each of us in the group had to individually reach out to the support staff at NIRx and request access. The issue was that one member did not receive access at the same time, and thus they had a more difficult time continuing research on the fNIRS. During this timeframe of about one month, we were limited to researching the device without actually interacting with it. This was due to the fNIRS device being held in a laboratory that we did not have access to.

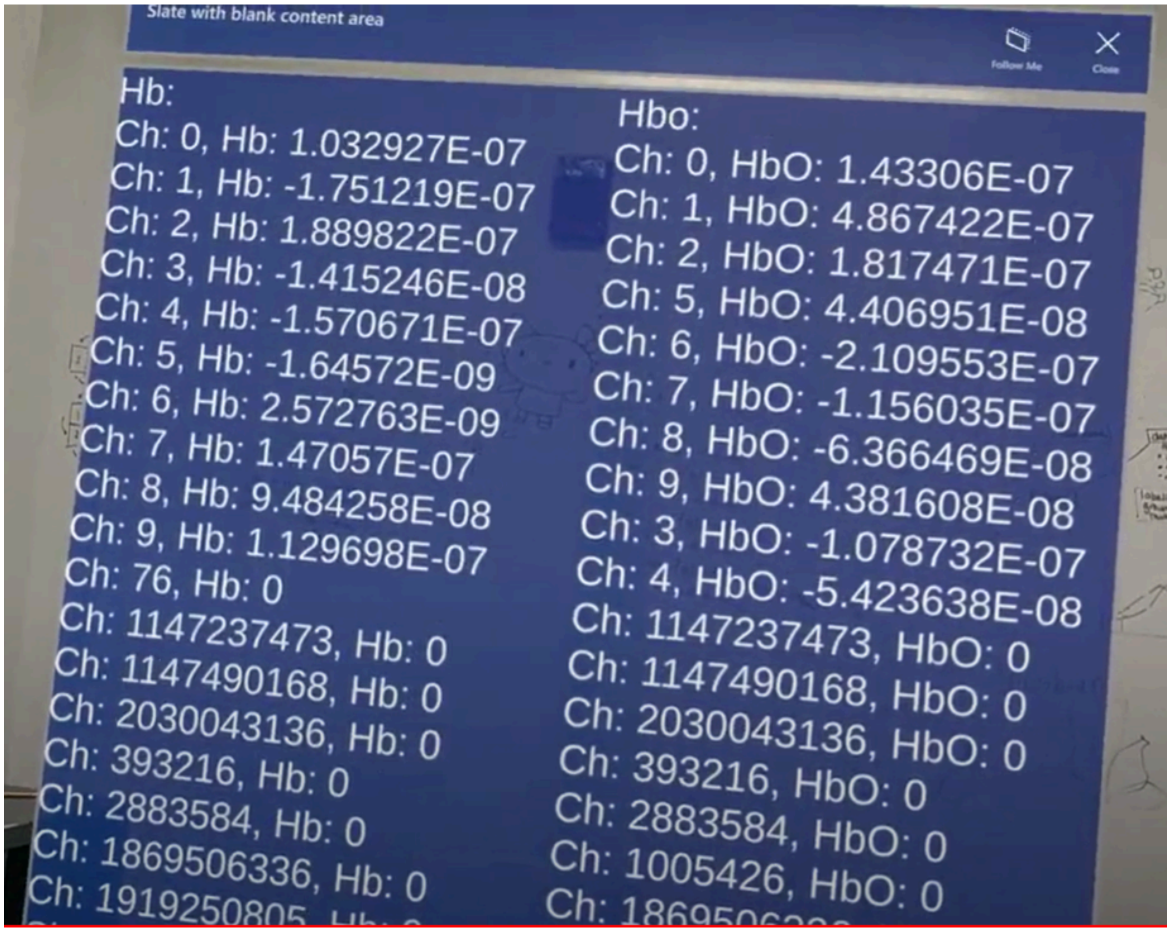
5.1.2 fNIRS

Once we were able to work with the fNIRS headset, we shifted our research so that it was more focused specifically on our project. Using the NIRx website, it was difficult to navigate exactly where we needed to find information specifically for our project, and the location of this information was only made clear once we received direction from our advisors. To interact with the fNIRS device multiple different connections need to be made. One of the major recurring issues that we faced was finding a consistent and easily replicable way to connect different devices. With the NIRSport 2 device already connected to the processing software, Aurora, the next connection we needed to establish was between Aurora and TSI. Another issue that we faced when working with the fNIRS headset was that, because we were still in the process of researching, we had trouble evaluating how much time and energy should go into working with the headset. For example, as a group, we thought that we should create a montage for the fNIRS headset at the beginning stages of our project. However, we severely underestimated how long it would take to create this montage from scratch. After meeting with our advisors, we established a list of priorities based on the advisors' expectations and what we could complete as a group.

5.1.3 Turbo-Satori/Aurora

One of the challenges that we faced when configuring TSI and Aurora was that we could not use the public WiFi networks that already existed such as campus WiFi because there was too much interference on the network. The solution was to use a separate WiFi router and connect the software using that network to avoid interference from unrelated devices. Once the connection was made, however, the challenge of configuring TSI to meet the needs of our project arose. TSI is complex software that includes numerous features, some that were useful to our project and some that were not. Without much experience using the software, it was jarring to look at for the first time and made it difficult to begin tuning the software. Thus we had to research more into what each feature offered.

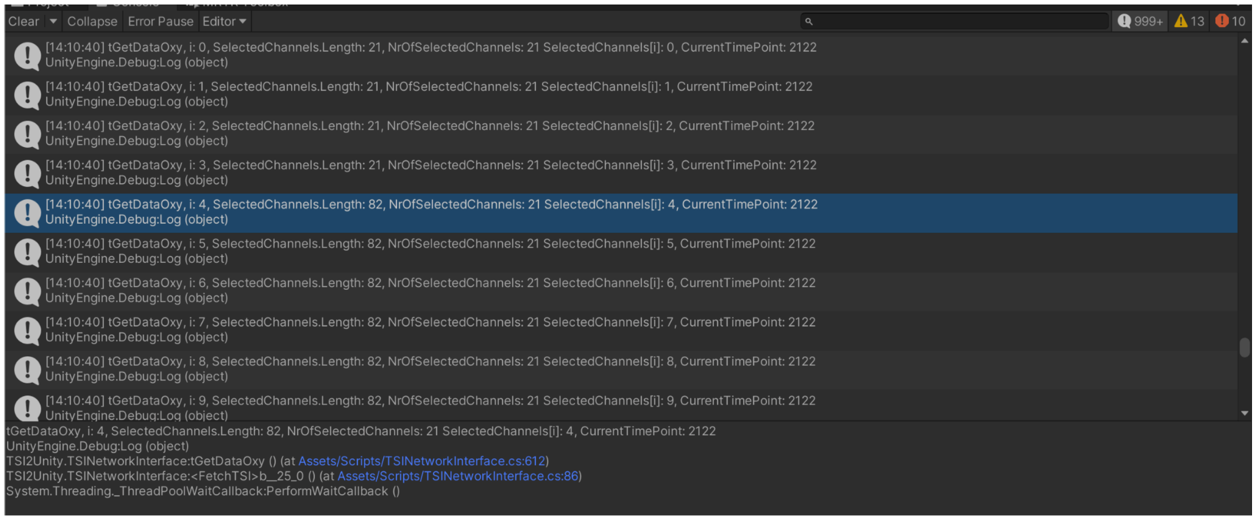
Another major issue we faced was that TSI would constantly crash when streaming data to Unity. We initially presumed the problem to be that Unity was frequently requesting too much data from TSI from too many channels. Using this presumption, we tried finding the right amount of channels to request information from so that we could both receive enough information for the data to be useful while also limiting the number of channels in order to avoid crashing the software. We ended up settling on 21 selected channels. However, this did not stop TSI from crashing and only made the crashes less frequent. We would only realize later what the problem was, as it was related to another major bug.



```
Hb:                                  Hbo:
Ch: 0, Hb: 1.032927E-07              Ch: 0, HbO: 1.43306E-07
Ch: 1, Hb: -1.751219E-07             Ch: 1, HbO: 4.867422E-07
Ch: 2, Hb: 1.889822E-07              Ch: 2, HbO: 1.817471E-07
Ch: 3, Hb: -1.415246E-08             Ch: 5, HbO: 4.406951E-08
Ch: 4, Hb: -1.570671E-07             Ch: 6, HbO: -2.109553E-07
Ch: 5, Hb: -1.64572E-09              Ch: 7, HbO: -1.156035E-07
Ch: 6, Hb: 2.572763E-09              Ch: 8, HbO: -6.366469E-08
Ch: 7, Hb: 1.47057E-07               Ch: 9, HbO: 4.381608E-08
Ch: 8, Hb: 9.484258E-08              Ch: 3, HbO: -1.078732E-07
Ch: 9, Hb: 1.129698E-07              Ch: 4, HbO: -5.423638E-08
Ch: 76, Hb: 0                        Ch: 1147237473, HbO: 0
Ch: 1147237473, Hb: 0                Ch: 1147490168, HbO: 0
Ch: 1147490168, Hb: 0                Ch: 2030043136, HbO: 0
Ch: 2030043136, Hb: 0                Ch: 393216, HbO: 0
Ch: 393216, Hb: 0                    Ch: 2883584, HbO: 0
Ch: 2883584, Hb: 0                   Ch: 1005426, HbO: 0
Ch: 1869506336, Hb: 0                Ch: 18695
Ch: 1919250805
```

Figure 10. HoloLens displaying Hb/Hbo channel data, including unexpected data

When attempting to display the readings from TSI onto Unity and the Hololens, we discovered an error where the channels would display odd channel data, such as including too many channels with abnormal channel ID numbers (see Figure 10). At first, we thought that this error was caused by some form of 32/64-bit incompatibility between TSI and Hololens since we never saw these kinds of errors when testing in Unity on a laptop. While attempting to debug the issue on Hololens however, we discovered that the same error was also occurring on Unity through console debugging. Our advisor then pointed out that the issue was likely not related to the 32/64-bit incompatibility, but to the Unity code that requested and received data from TSI. To figure out what the problem was, we rigorously combed through the code responsible for sending data back and forth, printing out to the console relevant variables to search for any mistakes. Eventually, we found the issue to originate from a validation statement that was not working as intended. The validation statement checked if the number of currently selected channels was equal to the expected number of selected channels, before moving on to loop through all the currently selected channels. The problem was that the list of currently selected channels had a small chance to change its length at random, meaning that its length would sometimes increase in the middle of the loop through the currently selected channels (see Figure 11). This would cause the code to start looking for channel IDs beyond those that existed, causing abnormal values to pop up as channel IDs. This bug is also the cause of the earlier issue of TSI randomly crashing; the Unity code would send a request to TSI for abnormal channel ID values that don't exist, causing TSI to crash.



```
[14:10:40] tGetDataOxy, i: 0, SelectedChannels.Length: 21, NrOfSelectedChannels: 21 SelectedChannels[i]: 0, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 1, SelectedChannels.Length: 21, NrOfSelectedChannels: 21 SelectedChannels[i]: 1, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 2, SelectedChannels.Length: 21, NrOfSelectedChannels: 21 SelectedChannels[i]: 2, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 3, SelectedChannels.Length: 21, NrOfSelectedChannels: 21 SelectedChannels[i]: 3, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 4, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 4, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 5, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 5, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 6, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 6, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 7, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 7, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 8, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 8, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
[14:10:40] tGetDataOxy, i: 9, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 9, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
tGetDataOxy, i: 4, SelectedChannels.Length: 82, NrOfSelectedChannels: 21 SelectedChannels[i]: 4, CurrentTimePoint: 2122
UnityEngine.Debug:Log (object)
TSI2Unity.TSINetworkInterface:tGetDataOxy () (at Assets/Scripts/TSINetworkInterface.cs:612)
TSI2Unity.TSINetworkInterface:<FetchTSI>b__25_0 () (at Assets/Scripts/TSINetworkInterface.cs:86)
System.Threading._ThreadPoolWaitCallback:PerformWaitCallback ()
```

Figure 11. Selected Channels.Length, the number of currently selected channels, suddenly changing from 21 to 82 mid-loop and post-validation.

5.1.4 Unity

Throughout the process of working with Unity, we encountered errors when attempting to connect the computer running Aurora and TSI to the computer that was running Unity. The challenge was that to connect the two computers, we had to input the IP address of the computer running the TSI into the computer running Unity. However, by default, the IP address of the computer running TSI would change each time it connected to the WiFi router. A solution that we attempted was creating a scene in Unity where the IP address of the computer running TSI could be entered. This attempt did not fix the issue, and we could not figure out what the problem was. In the interest of time, we decided to return to hardcoding the IP in the Unity code and move on. Another challenge that we faced with Unity was attempting to move our story script from a hard-coded approach to a modular approach. This meant that we moved all of our game text from within the code to an Excel spreadsheet.

5.1.5 Scrapped Ideas

At the start of the project, we had a lot of ideas, some of which have persisted throughout, and some that we had to discard due to time constraints, being too difficult to implement, or outside the scope of our project.

One such example would be our initial plan on how to incorporate BCI into our game. The creation of a BCI game requires different classifiers for the brain data created by the fNIRS headset so that the HbO and Hb data can be separated into different brain states. With surface-level research on the topics of which brain states to include in our game, we initially had many states that we thought we could incorporate into a story-focused game, including calm, focused/unfocused, excited, confused, frustrated, and stressed. We were excited at the prospect of including these states in our game.

However, after researching more on how fNIRS works as well as consulting our advisors, we realized that our ideas were too unrealistic to achieve in the time available, and were not within the scope of our project. Just determining whether it is possible to identify that number of cognitive states just from reading HbO and HbR is by itself enough to be a project on its own. Observing that number of cognitive states from only blood oxygen levels was not something that we could cover in a project lasting less than a year, much less doing so at the same time as writing/designing a game, coding it in AR with Unity, and learning how to stream data from fNIRS to HoloLens. We had to limit our observed cognitive states to two: a specific cognitive state and not that specific cognitive state. In the end, we decided that focused and unfocused cognitive states would serve our project the best since they were a straightforward pair of cognitive states that could be used in any situation.

Another idea we had at the start was to use a machine learning (ML) algorithm to determine the thresholds of HbO and HbR for the focused and unfocused states. Utilizing an ML algorithm would allow the game to calibrate HbO/HbR thresholds for focus state based on the current player instead of using fixed thresholds that would be accurate for some players, but not nearly as accurate for others. After we started working on the project, we put that idea on our backlog, and as we developed our game, we soon realized that we would not have enough time to learn to create and develop an ML algorithm, so we had to discard that idea.

5.2 Next Steps

5.2.1 Expanding the Game

Currently, the game is designed in a modular way with the current game script entirely within a CSV file that is read in Unity, allowing future groups to easily edit and add to the game story. Additionally, the game was also designed in a way where the use of buttons and a slate in Unity was not necessary for the game to run and was instead just the way we presented the text to users. This allows other groups with more time to potentially further develop the UI or include visual storytelling into the game. The goal of this project, in conjunction with building a HoloLens-based augmented reality game that adapts its narrative style based on cognitive load data retrieved using the fNIRS neuroimaging technique, was also to develop the game so that we had a framework that was easy to understand. One of the more time-consuming aspects of this project, connecting the fNIRS device to Unity to read the user's focus state, was not entirely completed by us. Despite not actually incorporating this into our game, we created a framework that hopefully allows other developers to integrate the use of cognitive states into their own projects.

5.2.2 Creating a Custom Montage

To get the most accurate readings from the fNIRS device, it is important to read only the areas of the brain that relate to the cognitive states that are used within the game. For example, for our project, we focused on a user’s focus levels to change states throughout the game. As a result, the montage that we used was a preexisting one that highlighted the prefrontal cortex area of the brain, or the area that is most relevant to focus levels within an individual. Future project groups could do more research on creating a montage specifically for the prefrontal cortex. A unique challenge however is creating the montage in a way that also allows users to wear the HoloLens device in conjunction with the fNIRS device. In our testing with preexisting montages from past projects, we had to compromise by removing some of the sensors to allow the HoloLens to sit comfortably on a player's head. Future project groups could work to further optimize the montage to create the highest level of accuracy from the fNIRS headset while still allowing the HoloLens to be worn by users.

6. Conclusion

The goal of this project was to establish an infrastructure for the future development of AR games that adapt their narrative style based on cognitive data retrieved using fNIRS and identify avenues of development as well as challenges involved with integrating brain signals into a game. By the end of this project, we completed our goal by creating an AR murder mystery game that sets up the use of focus states to enhance its storytelling, developed in a way that allows future improvement upon the game or the creation of a completely different game with a similar story structure. We hope that future groups can make use of our project, either furthering it or creating something completely new.

Appendices

Appendix A - Script CSV Guide

This is a quick guide on how the story script CSV works, and how to add to it.

Python/Unity code reads the CSV row by row, with the first column (column index 0) determining the purpose of the row and what the code needs to do with it. If you're familiar with programming, then each row is a function call, column 0 is the function name, and the rest of the columns are arguments for the function.

Dictionary for column 0 keywords:

Col0: comment
Col1: whatever you want to say
Col2: whatever you want to say
Col3: whatever you want to say
Col4: whatever you want to say
Col5: whatever you want to say
Col6: whatever you want to say
Description: Just for making comments in the script. These rows are skipped by the code.

Col0: dialogue
Col1: text to be output
Col2:
Col3:
Col4:
Col5:
Col6:
Description: The content of Col1 is displayed on the main screen. Used a lot throughout the script to tell the story.

Col0: state
Col1:
Col2:
Col3:
Col4:
Col5:
Col6:

Description: Triggers code to determine whether the user is focused or not depending on the recorded focus state. A variable representing the state is set based on this.

Col0: state_dialogue
Col1: focused dialogue output
Col2: unfocused dialogue output
Col3:
Col4:
Col5:
Col6:
Description: Depending on the state of the player, displays the contents of either Col1 (if focused) or Col2 (if unfocused).

Col0: choice_3
Col1: response 1
Col2: response 2
Col3: response 3
Col4:
Col5:
Col6:
Description: Gives the player 3 response choices to choose from. Usually followed by path_3.

Col0: path_3
Col1: dialogue 1 (chose response 1)
Col2: dialogue 2 (chose response 2)
Col3: dialogue 3 (chose response 3)
Col4:
Col5:
Col6:
Description: Displays dialogue based on the user's most recent choice (usually from choice_3).

Col0: state_choice_3
Col1: focused, response 1
Col2: focused, response 2
Col3: focused, response 3
Col4: unfocused, response 1
Col5: unfocused, response 2
Col6: unfocused, response 3
Description: Depending on the player's state, gives the player a set of 3 response choices. Usually followed by state_path_3.

Col0: state_path_3
Col1: dialogue 1 (focused, chose response 1)
Col2: dialogue 2 (focused, chose response 2)
Col3: dialogue 3 (focused, chose response 3)
Col4: dialogue 4 (unfocused, chose response 1)
Col5: dialogue 5 (unfocused, chose response 2)
Col6: dialogue 6 (unfocused, chose response 3)
Description: Depending on the state_choice_3, gives the player specific dialogue.

Col0: choice_2
Col1: response 1
Col2: response 2
Col3:
Col4:
Col5:
Col6:
Description: Gives the player 2 response choices to choose from. Usually followed by path_2.

Col0: path_2
Col1: dialogue 1 (chose response 1)
Col2: dialogue 2 (chose response 2)
Col3:
Col4:
Col5:
Col6:
Description: Displays dialogue based on the user's most recent choice (usually from choice_2).

Col0: point_change
Col1: "{Name} {integer points changed}" (focused, chose response 1)
Col2: "{Name} {integer points changed}" (focused, chose response 2)
Col3: "{Name} {integer points changed}" (focused, chose response 3)
Col4: "{Name} {integer points changed}" (unfocused, chose response 1)
Col5: "{Name} {integer points changed}" (unfocused, chose response 2)
Col6: "{Name} {integer points changed}" (unfocused, chose response 3)
Description: Triggers code to make changes to suspicion/point values. Used after state_choice_3. Examples: "Baker -1" subtracts one point from Baker. "Carpenter 2" adds two points to Carpenter.

Col0: wait

Col1: dialogue
Col2: response 1
Col3: response 2
Col4:
Col5:
Col6:
Description: Intended to stop all dialogue and player responses for one minute, before asking the player if they are done (dialogue being the story/detective asking the player if they are ready to continue, and responses 1 and 2 being yes and no). Usually followed by wait_response.
As of writing this guide, the one-minute pause part is not implemented (on backlog).

Col0: wait_response
Col1: dialogue 1
Col2: dialogue 2
Col3:
Col4:
Col5:
Col6:
Description: Depending on whether the player responded with yes or no to a "wait", gives the player specific dialogue. Responding with no also loops back to the previous "wait".

Col0: accuse
Col1: name 1
Col2: name 2
Col3: name 3
Col4:
Col5:
Col6:
Description: This row looks at the stored suspicion values, and (assuming the order of the stored values in the code and the names in Col1, 2, and 3 are the same) returns the Col# value as dialogue. And yes, if you want to change the names of the suspects, the order of the suspects in the CSV, or add/remove suspects, you will need to change the code accordingly.

Appendix B - Design of Receipt used for the AR Game

7 Privet Drive, London, United Kingdom

CENTRAL BRANCH

The Good Generals

Terms: Net Cash	It is agreed that the title to the property herein described shall remain in the seller until the full purchase price is paid in full. Upon default of any payment, all installments shall at once become due and payable and the seller may retake possession of said property. This notice applies to all property contained within this receipt.	Our Order Number.A 503
(1)	- Whetstone - Amount paid on 15/12/1929-----------> A whetstone used to sharpen large knives.	£0.29 £0.19 Remaining: £0.10
(2)	- Replaceable Blades - Amount paid on 15/12/1929-----------> Package of small, cheap blades that can be placed into the hilt of a whittling knife.	£0.15 £0.09 Remaining: £0.06
(3)	- Bundle of Mugwort - Amount paid on 15/12/1929-----------> Mugwort to be used for medicinal purposes (sedative).	£0.13 £0.06 Remaining: £0.10

Appendix C - AR Game Unity/C# Code

Link to repository:
https://github.com/atnguyen01/BCI-GAME

Appendix D - AR Game HoloLens and Unity Video Demos

AR Game Demo on HoloLens:
https://drive.google.com/file/d/1qF3C2HGxzMV_8_oFD83aSTgUt8p14KGM/view?usp=sharing

Unity Demo Video:
https://drive.google.com/file/d/1kHedwXQZRiNc_rDTLCsgNEBQjWWZ-flO/view?usp=sharin
g

References

Arjoranta, J. (2017). Narrative tools for games: Focalization, granularity, and the mode of narration in games. Games and Culture, 12(7–8), 696–717. https://doi.org/10.1177/1555412015596271

Brain Support. (2020, March 27). *Aurora fNIRS - NIRSport 2 Acquisition software. How acquire data with NIRS?* Brain Support. https://www.brainlatam.com/knowledge-base/aurora-fnirs-nirsport-2-acquisition-software.-how-acquire-data-with-nirs-222

Brain Support. (2021, February 8). *How to use Lab Streaming Layer (LSL) for data streaming for NIRS*. How to use lab streaming layer (LSL) for data streaming for Nirs. https://www.brainlatam.com/knowledge-base/how-to-use-lab-streaming-layer-lsl-for-data-streaming-for-nirs-300

Brain Innovation. (n.d.). *Brain Innovation - Products - Turbo-Satori*. Brain Innovation. Retrieved October 11, 2023, from https://www.brainvoyager.com/products//products/turbosatori.html

Britton, J. W., Frey, L. C., Hopp, J. L., Korb, P., Koubeissi, M. Z., Lievens, W. E., Pestana-Knight, E. M., & St. Louis, E. K. (2016). Introduction. In E. K. St. Louis & L. C. Frey (Eds.), *Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants [Internet]*. American Epilepsy Society. https://www.ncbi.nlm.nih.gov/books/NBK390346/

Casey, A. (2020, June 25). *Introduction to fNIRS and NIRx*. https://www.youtube.com/watch?v=KtibfRpV1gI

Causse, M., Chua, Z., Del Campo, N., Matton, N., & Peysakhovich, V. (2017). Mental workload and neural efficiency quantified in the prefrontal cortex using fNIRS. Scientific Reports, 7(1), 5222. https://doi.org/10.1038/s41598-017-05378-x

Chen, M., Solovey, E., & Gillian, S. (2023). Unity-fNIRS. https://github.com/WPIHCILab/Unity-fNIRS/tree/main

Ciena, L. B. (2016, August 10). *Understanding the differences between virtual reality, augmented reality and mixed reality*. Network World. https://www.networkworld.com/article/3106205/understanding-the-differences-between-virtual-reality-augmented-reality-and-mixed-reality.html

Foxman, M. (2019). *United We Stand: Platforms, Tools and Innovation With the Unity Game Engine*. Social Media + Society, 5(4). https://doi.org/10.1177/2056305119880177

Jones, A., Letendre, M., & Nerden, E. (2023). *Sewn Into Memory: Reliving Memories Through An AR Quilt*. Worcester Polytechnic Institute. https://digital.wpi.edu/concern/student_works/2f75rc40d?locale=en

Kim, S.-K., Kim, S. K. & Yu, A. H., (2021). Study on character design affecting interactive games. International Journal of Applied Science and Engineering Review, 02(06). https://doi.org/10.52267/IJASER.2021.2602

Larsen, B. A., & Schoenau-Fog, H. (2020). Making the Player the Detective. *International Conference on the Foundations of Digital Games*, 1–8. https://doi.org/10.1145/3402942.3402969

Li, R., Yang, D., Fang, F., Hong, K. S., Reiss, A. L., & Zhang, Y. (2022). *Concurrent fNIRS and EEG for Brain Function Investigation: A Systematic, Methodology-Focused Review*. *Sensors (Basel, Switzerland)*, 22(15), 5865. https://doi.org/10.3390/s22155865

Link, S. J. (2023). Defining Detective Fiction. In S. J. Link, *A Narratological Approach to Lists in Detective Fiction* (pp. 17–38). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-33227-2_2

Microsoft. (2022, December 16). *Mrtk2-unity developer documentation—Mrtk 2*. Microsoft. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05

Microsoft. (2023, March 13). *About HoloLens 2*. https://learn.microsoft.com/en-us/hololens/hololens2-hardware

Niantic. (2023, September 15). *Catching pokémon in ar+ mode—Pokémon go help center*. Niantic. https://niantic.helpshift.com/hc/en/6-pokemon-go/faq/28-catching-pokemon-in-ar-mode/

Porokh, A. (2023, May 3). Storytelling in video games [The Art of Storytelling: How is Storytelling Used in Video Games?]. Kevuru Games; Kevuru Games. https://kevurugames.com/blog/the-art-of-storytelling-how-is-storytelling-used-in-video-games/

Reid, A. (2020, February 27). Ethical worldbuilding in games [Ethical Worldbuilding in Games]. Massive Entertainment. https://www.massive.se/blog/games-technology/ethical-worldbuilding-in-games/

Solovey, E. T., & Putze, F. (2021). Improving HCI with brain input: Review, trends, and outlook. *Foundations and Trends® in Human–Computer Interaction*, *13*(4), 298–379. https://doi.org/10.1561/1100000078

Turbo-satori: Real-time analysis software. (n.d.). NIRx Medical Technologies. Retrieved March 20, 2024, from https://nirx.net/turbosatori

Unity Technologies. (2023, September 29). Unity Manual. Unity. https://docs.unity3d.com/Manual/UnityManual.html

Wright, G. (2023, February). *What is augmented reality gaming (AR gaming)? –* TechTarget Definition. WhatIs.Com. https://www.techtarget.com/whatis/definition/augmented-reality-gaming-AR-gaming