Oregon State University
College of Engineering

CS Capstone Design Document

November 27, 2019

Design and implementation of a framework for bio-informed 3D user interaction

Prepared for

Oregon State University College of Engineering

Raffaele de Amicis _____ _____
Signature *Date*

Prepared by

Group 51

BioMR

Ley Aldinger _____ _____
Signature *Date*

Ayush Choudhury _____ _____
Signature *Date*

Kyle Hiebel _____ _____
Signature *Date*

Zunyue Qiu _____ _____
Signature *Date*

Abstract

The project is broken into 5 components, time perception, sensors, API, Unreal Engine 4, and Unity. Time perception can be linked to attention, which in previous research, has been quantitatively measured. Our biometric sensors such as eye-tracking, GSR, and ECG, can all measure attention, thus we can measure time perception quantitatively. The API component will read data from these sensors and correlate the data with events happening in the VR environment. Two different VR environments will be created, one in UE4 and one in Unity, both of which have weather and lighting which are modifiable in real-time by a researcher using the API.

CONTENTS

1 Overview

1.1 Scope

The project will entail the research, design, and implementation of a system architecture which allows for communication between biometric sensors and a game engine. The sensors, virtual reality headset, and game engine are components which will be provided by the client. The individual components will be chosen based on their price and functionality. A custom API will be developed to transmit data between the sensors and game engine. This custom API does not need to be developed from scratch, and will likely be built as an extension of an existing API. Finally, 2 virtual reality applications will be developed in 2 different game engines to validate the system architecture. These VR environments will be created by modifying existing scenes to speed up development.

1.2 Purpose

As virtual reality technology becomes more popular, users are spending increasing amounts of time in virtual worlds. In the near future, some on-site jobs will be replaced with cyber-physical jobs which place the worker in a virtual environment for 8 or more hours. Due to this impending upsurge in the use of virtual reality, it is increasingly important to understand the effects which being in a virtual environment may have on the body. Understanding the physiological effects of VR will enable developers to create virtual reality software designed for optimal user experience, and allows for the development of systems which adapt in response to user physiological data in near real time.

1.3 Intended Audience

This document is intended for the developers, customers, and stakeholders. The overall goal is to generate a working document which outlines how the requirements of the project will be met, and the specific technologies and strategies which will be used to achieve this goal. The design document sits as a tentative road map for the upcoming 6 months of this project and shall be revised until the project reaches completion.

2 Definitions

- AR - Augmented Reality
- API - Application Programming Interface
- ECG - Electrocardiogram
- EMG - Electromyography, an electrical diagnostic medical technique used to assess and record electrical activity produced by skeletal muscles.
- GSR - Galvanic skin response
- HMD - Head Mounted Display
- HTC VIVE - Virtual reality headset. Released 2016.
- Oculus Quest - Virtual reality headset. Released 2019.
- SDK - Software Development Kit
- System Architecture - A conceptual model that defines the structure, behavior, and more views of the system.
- UE4 - Unreal Engine 4 (Game Engine)
- Unity - Game Engine
- VR - Virtual Reality
- XR - Cross Reality. Virtual and Augmented Reality combined.

3 Design Description

3.1 Overview

Fig. 1: The system architecture broken down into it's 5 major components. Researchers will run an experiment using the API, and subjects will interact with the HMD.

To allow a researcher to gain insight on how environmental factors affect a VR user's perception of time, we will develop a system architecture in which a researcher can perform such a study. The researcher will interact with our system architecture through a custom API which will serve as middleware between biometric sensors and a VR scene. A subject will interact with the VR environment through the HMD while biometric sensors will quantitatively measure the subject's perception of time.

The system architecture will be broken down into 5 major components (Figure 1). To begin, the sensors collect biometric data and expose this data to our API through firmware. The API will be able to connect to an arbitrary sensor's firmware to collect, analyze, and process the sensor's data. A 2-way interface will be created between the API and one of two game engines, either UE4 or Unity. A game engine will provide the API with the current activity the subject is performing in VR, and the API will send commands to the game engine to modify the VR environment, per request of the researcher. A test scene will be created in each of the two game engines to allow for the system architecture to be validated. Lastly, the subject will interact with the Head Mounted Display (HMD) allowing them to move around an immersive VR environment.

3.2 Design Stakeholders

The sole stakeholder for this project is professor Raffaele de Amicis. He requires that the resulting product is a system architecture which can quantitatively measure the time perception of subjects based on their VR surroundings.

3.3 Time Perception Design

3.3.1 Overview

This project will use biometric data read from subjects in order to analyze their time perception in VR. There are many factors that could affect people's time duration judgement, distraction, motion misperception, and spatial perception, to name a few. Furthermore, when people's attention has been captured by something, their time duration judgment is often shorter than the actual time duration. Through previous research, we will search for a quantitative method to measure people's attention and time perception.

3.3.2 Detail

The need to measure the time perception of a subject in VR is an important component in connecting bio metrics to the world. One key characteristic that could affect time perception is attention distraction. An experiment made by Lontz concluded that when people try to finish a task with auditory distraction, their time duration judgments are underestimated than the real-time[1]. To learn more about the relationship between time perception and attention, we need to measure attention, which can give insight into our VR case.

According to the article Foundations of Vision, the spacial resolution of photoreceptors in the human retina is not uniform[2]. The density is highest in that part of the retina aligned with the visual axis(the fovea), and the resolution around the fovea is logarithmically decreasing with eccentricity[2]. Also, their results show that observers' scan paths are similar, and to some extent, predictable[2]. Since there are patterns in eye tracking linked to information density, B. Wandell points out that eye-tracking is an essential and common method to measure people's attention.

Furthermore, neurofeedback is another method to measure one's attention. The name of neurofeedback training(NFT) includes skin temperature, heart function, and muscle activity, and these protocols expose individuals real-time neural activity[3]. Furthermore, Berger observed that the learning rates are faster in a 3D virtual reality(VR) environment compared to a 2D control situation[4]. In other words, through (NFT), the observer could monitor the user's attention during the time that the user plays in VR.

3.4 Sensor Design

3.4.1 Sensor Hardware

- As a user, I would like to measure eye movement, galvanic skin response, and electrocardiographic data of a person using a virtual reality program.
  - We will use eye-tracking enabled HMDs to measure and record gaze and eye movement patterns through light reflections from the VR user's pupils and cornea. This will allow us to evaluate the VR user's attention levels and direction of focus.
  - We will use GSR sensors to measure sweat gland activity through the electrical conductivity of the VR user's skin. This will allow us to evaluate the VR user's exertion, stress levels, and level of emotional affect.
  - We will use ECG sensors to measure muscle and nerve activity, as well as the path of electrical signals within the VR user's heart. This will allow us to evaluate the VR user's posture, gait, and muscular movement.
- As a user, I would like to know how to set up my biometric sensors for concurrent use with my virtual reality display and controllers.
  - We will study the set up and use of the eye-tracking enable HMD, GSR sensor, and ECG sensor and create a user guide explaining both visually and in writing an effective method of setting up all three pieces of hardware for concurrent use by one VR user.

3.4.2 Sensor Output

- As a user, I would like to know how the biometric data collected by my sensors relates to time perception.

  - We will research time perception from a biocognitive perspective and create a brief user guide to the evaluation of time perception through eye-tracking, GSR, and ECG.

- As a user, I would like the biometric data collected by my sensors to be displayed and saved in a format I can read.
  - We will create an interface within our API which displays the biometric data collected by each sensor visually and numerically.
  - We will create an interface within our API which displays the biometric data collected by each sensor visually and numerically.

3.4.3 Output Processing

- As a user, I would like the biometric data collected by my sensors to be analyzed with respect to time perception by the API middleware.
  - We will create a set of grading criteria based on our biocognitive research on time perception which can infer distortion in a person's perception of time from the biometric data our sensors can collect.
  - We will establish a grading scale representing the level and qualities of distortion in a person's perception of time with respect to time as measured by the VR program clock, where both the way estimation of duration is distorted (as shorter or longer than clock time) and the amount of distortion present is represented.
  - We will create a grading program within our API which analyzes incoming biometric data from the sensors in near real-time based on our grading criteria and calculates a grade representing the level of time distortion experienced by a user based on our grading scale.
- As a user, I would like my VR programs to change in response to the VR user's perception of time.
  - We will create a library of manipulation functions which change elements of the VR environment when called. The functions we include will be designed to affect user time perception, based off of cognitive research on human perception of time.
  - We will establish a set of default responses to time perception grades produced by our grading program, associated specific grades with specific functions from our manipulation library.
  - We will create an observer program which will respond to grades produced by the grading program by calling functions from our manipulation library associated with the observed grade.
- As a user, I would like to know what changes can be made to the VR environment and what will trigger those responses during a virtual experience.
  - We will create aliases for our grading scale that intuitively convey the meaning of the grade in a manner that is easy for the user to understand.
  - We will create aliases for our manipulation functions that intuitively convey the effect of the function in a manner that is easy for the user to understand.
  - We will create a user interface displaying possible grades, implemented manipulations, and the associations between them.
- As a user, I would like to specify certain time perception grades which my API middleware should respond to, and the manipulation responses the API should make.

  - We will create a user interface which allows users to add connections between time perceptions grades and manipulation functions that are not already connected or remove connections between connected grades and functions.

- As a user, I would like to view the way a VR user experienced time during a virtual experience after that experience has ended.
  - We will create a database of time perception grades recorded over the duration of a virtual experience.
  - We will create an interface which displays the information stored in the time perception database to the user visually and textually
  - We will create a function which allows the user to export displayed data in useful formats.
- As a user, I would like to view biometric data collected at the time of specific events during a VR experience, such as user interaction with an object in the virtual scene.
  - We will create a database of interactions during a VR experience and their associated time stamp as imported from Unity/Unreal Engine, and associate each with keywords representing type or qualities of the interaction.
  - We will create an interface displaying the interaction database information to the user.
  - We will implement searching capabilities allowing the user to filter displayed interactions in the database by name, type, or quality.
  - We will implement a function to search the grades database for time perception grades given at a time nearest to a given timestamp.
  - We will implement a feature allowing the user to view the time perception grade given at the time of a selected interaction.
  - We will implement a feature allowing the user to view all displayed interactions side-by-side with the time perception grade given nearest to the timestamp of each interaction.
- As a user, I would like to view the conditions of the VR scene during the time of a specific recorded biosignal, such as a period of elevated heart rate or prolonged stationary eye positioning.
  - We will create a database of environmental parameters of a virtual scene over the duration of a virtual experience as imported from Unity/Unreal.
  - We will create an interface displaying the parameters of the environment over time.
  - We will implement a feature allowing the user to view the environmental parameters of the virtual scene at the timestamp a time perception grade was given at.

3.5 API Design

3.5.1 Overview

The API can be viewed as a client-server system. In this paradigm, the clients are the researcher, sensors, and game engine, while the API is an always on server. The researcher would like be able to manipulate objects in the game engine, such as weather and lighting. The API waits for a command from the researcher, then sends the proper signals to the game engine upon request. The game engine also acts as a client because it will signal the API when the user is performing certain events. The API also waits for data from biometric sensors, and when this data is available, it will be stored in the API database.

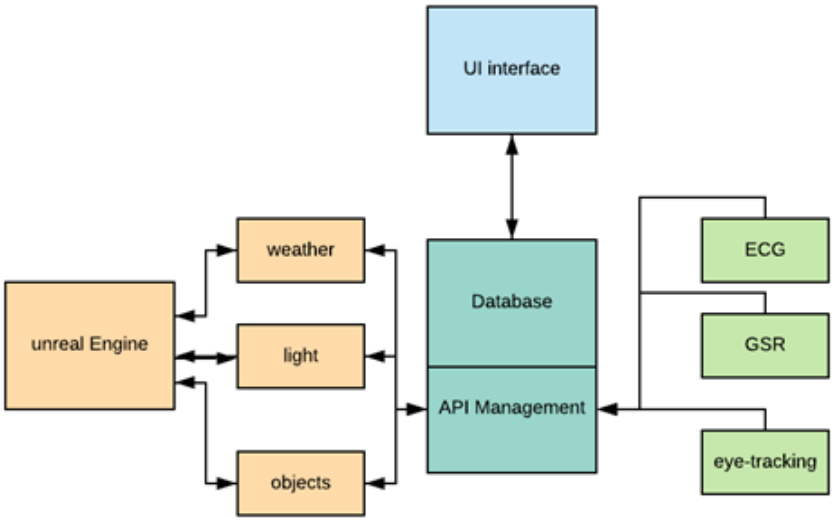


Fig. 2: The API will interface with external components through a management layer. The API acts as a server for the game engine, sensors, and researcher.

3.5.2 API General Design

The API will consist of multiple components (see Figure 2). The API Management layer will be coded in C++ since it will allow for easy communication with the sensors and game engines. The database layer will be coded in JavaScript and will use SQL as a query language. The API will have 2 different public interfaces depending on which system is trying to connect. The researcher will have access to different information than the sensors and game engines.

Specifically, for connecting to the database, we will use an object of class DatabaseConnection which inherits from the PHP data object class. In order to query the database, the SQL statements will be sent to a database connection which will return the requested data.

3.5.3 API Connection to Game Engine

Description

As a researcher, I would like to change objects in my subject's VR environment in real-time.

Design

The user interface will contain an input widget to allow for parameters in the VR environment to be modified. When the researcher makes a change to a parameter, the API will call a registered function in the game engine. These registered functions will be set up by the game engine and serve as a synchronous signal through the C++ API management interface. The game engine must register a function by providing a function pointer before the researcher can modify the parameter. When a function is registered, the initial value will be provided to the API so it can be displayed in the user interface.

3.5.4 API Connection to Sensors

Description

As a researcher, I would like to read biometric sensor data from a subject in VR and store it in a database.

Design

When subjects play in VR, the biometric sensors such as eye-tracking, GSR, ECG, etc. will begin to report data. We will read this data using the process described in 3.4.3. Once the data is stored in local memory, we will use JavaScript and SQL queries to add the data to the database. These calls will be made asynchronously as there may be large amounts of data flowing from the sensors, and we cannot slow down the gameplay. Each entry in the database will contain a timestamp and the sensor from which the data was retrieved.

3.5.5 API User Interface

Description

As a researcher, I would like to have a user interface which allows me to modify the VR environment and view biometric sensor data.

Design

The user interface will let researchers easily see all data that is measured from subjects during VR. The raw data will be displayed in table form and allow for custom SQL queries. The table will be created using either Qt or Windows Forms depending on how the database interface is coded. Figure 3 shows how the user interface will be laid out. By modifying a parameter on the left side of the page, the API will send a signal to the game engine to change that parameter. The researcher can customize their data view by sorting by column, or entering their own SQL query.

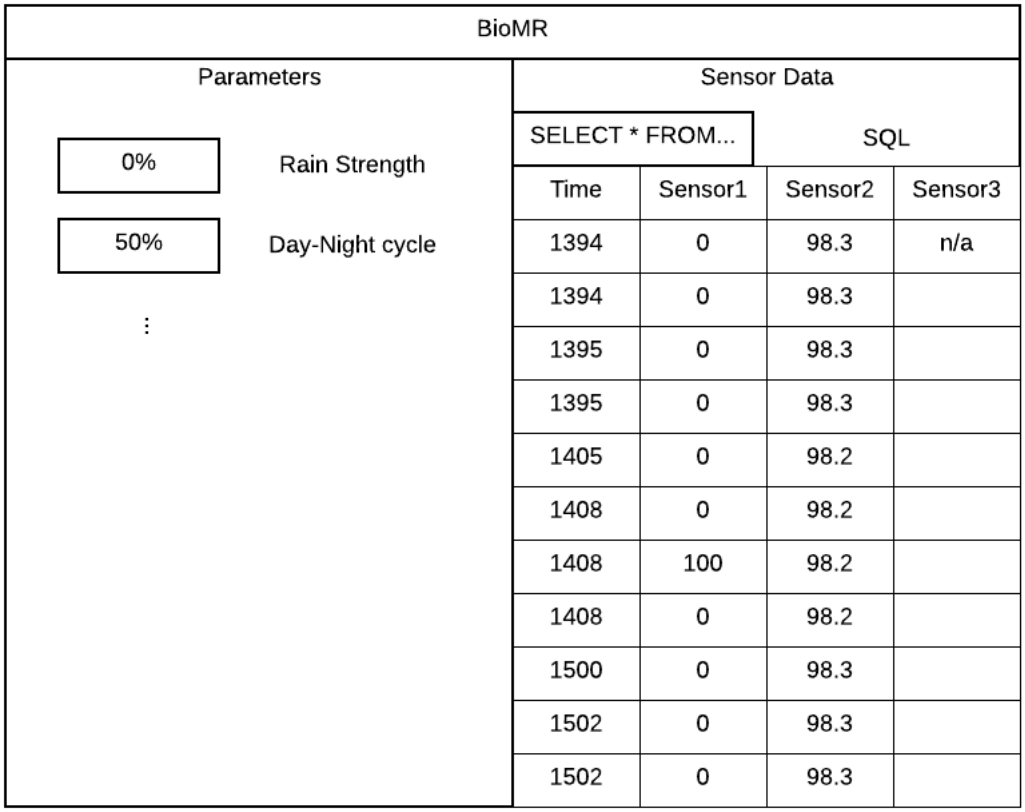


Fig. 3: A paper prototype of the user interface. The left half of the screen will contain parameter controls while the right half allows for data to be viewed.

3.6 Unreal Engine 4 Design

This section outlines the details of a test scene which will be created using Unreal Engine 4. Figure 4 shows all the components which need to be developed for the scene to meet the requirements set by the client. In the architecture of the UE4 scene, data flows from user interactions with the scene, to the API, then back to the VR scene to modify the effects.

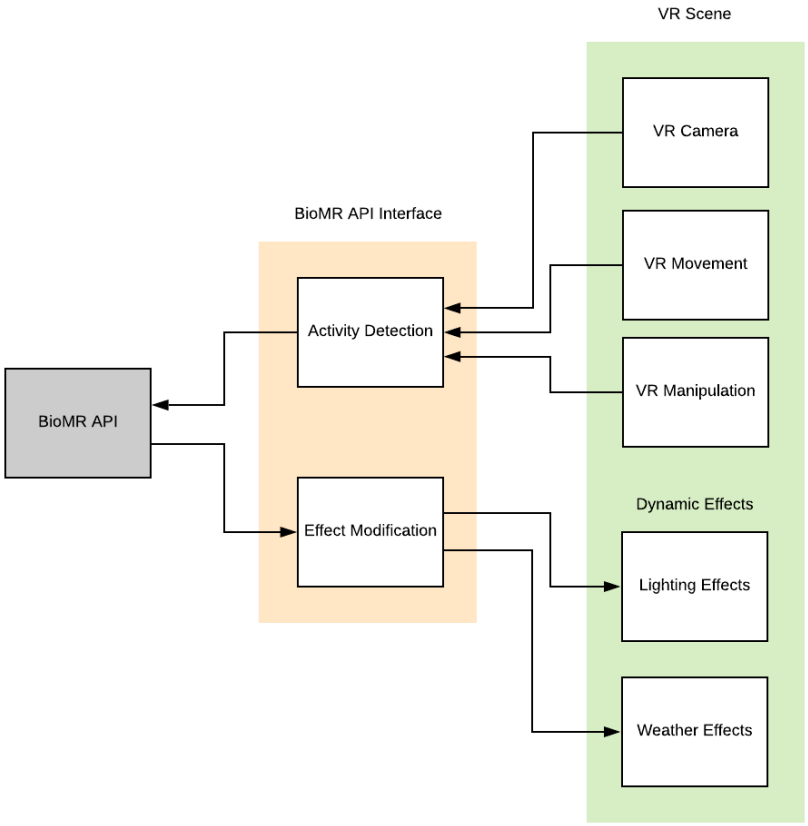


Fig. 4: The full UE4 development stack with multiple layers. The user facing layer (green) contains all the visible elements of the UE4 scene. The middle layer (orange) handles communication between the UE4 application and the BioMR API Plugin. The BioMR API will be included in the project as a statically linked plugin.

3.6.1 VR Scene and Interface

Description

As a researcher, I would like my subject to be able to navigate and interact within an immersive Unreal Engine 4 scene so I can study how certain elements of their environment affect their perception of time.

Design

The scene will be based on the expansive UE4 example scene, A Boy and His Kite, which can be downloaded for free from Epic Games[5]. Pressing the number 4 key takes the player to the location shown in Figure 5, which we have selected as a good starting area due to its proximity to water, deer, and trees. The PlayerStart actor will be moved to this location using the UE4 map editor so that the scene will always start here. A collision volume will be placed using the map editor to limit the playable area at the start[6]. All assets outside the visible area will be deleted to improve performance since the scene's performance is very poor without any modification. The next sections outline how the scene will be prepared for VR operation.

Fig. 5: This preset location is a good starting location because it is close to water, deer and trees. These objects can be used to influence interaction and navigation.

3.6.1.1 VR Camera

Description

As a researcher, I would like my subject to be able to view an Unreal Engine 4 scene in virtual reality so they can be immersed in a virtual environment.

Design

A VR scene can be launched natively in Unreal Engine 4 using the VR Preview command (Figure 6). To allow for VR camera movement, a standing camera will be implemented according to the UE4 documentation[7]. A VR camera will be added as a component to the default scene. Then, the camera's tracking origin will be set to floor level so that the camera will follow the ground height (Figure 7). The camera will be assigned to follow a pawn which will stick to the player at all times. The camera's position will remain constant even though the view in VR can be offset by changing the location of the HMD. This is the default VR camera mode in UE4, and it allows the height of the camera to be calculated off the center floor height of the playable region.

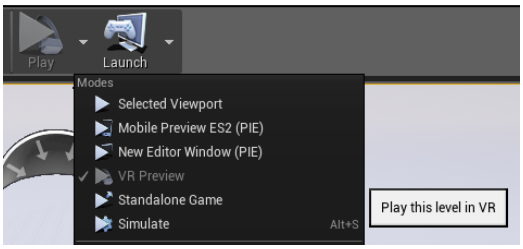


Fig. 6: Starting a scene in VR Preview mode.

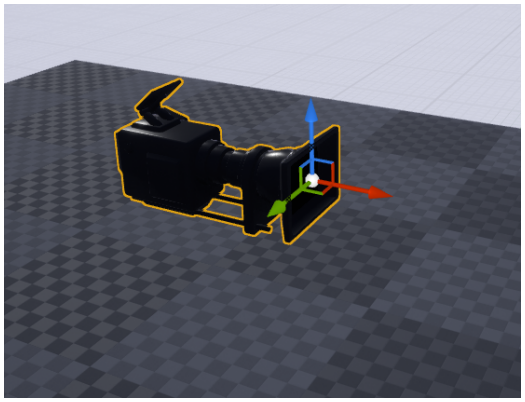


Fig. 7: A VR camera in UE4.

3.6.1.2 VR Movement

Description

As a researcher, I would like my subject to be able to move naturally about a VR scene so that I can measure their time perception during navigation.

Design

To allow movement beyond the boundaries of the VR play area, we will implement teleportation controls. To do this, we will use the player asset from the VR Template provided by Epic Games[8]. This teleportation mechanic requires motion controllers, and works as follows. The movement phase starts when either analog stick on a touch controller is pushed forwards. Now, an arc to a new region is drawn as in Figure 8. The subject now must determine the location they wish to teleport to by moving the controller. The higher the controller, the farther the teleporation distance. The direction is decided by the angle between the HMD and the controller. After the direction of teleport is determined, the new direction is determined by the direction the analog stick is pointing when it is released back to the center.

To implement this type of movement, we need to create a navigation mesh. We will create a navigation mesh by dragging a ”Nav Mesh Bounds Volume” into the UE4 map editor, and let it automatically calculate terrain height. Now, the mesh will be assigned to the player camera so it knows the possible locations to teleport to. This navigation mesh can be used for both VR player movement and AI movement for dynamics in the scene, such as deer.

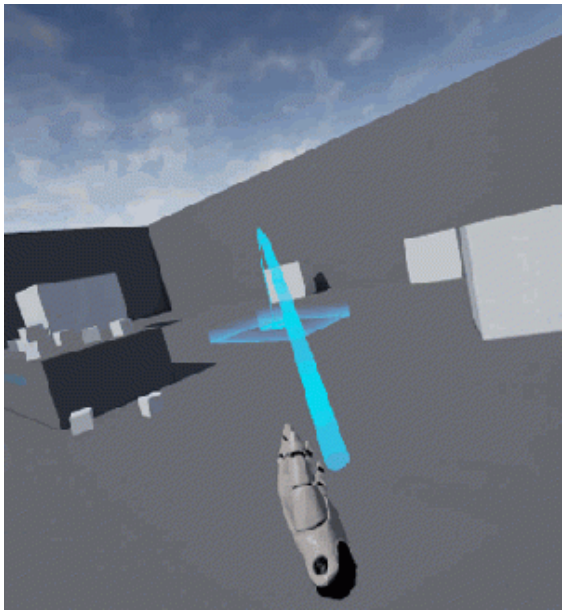


Fig. 8: Teleportation controls

3.6.1.3 VR Manipulation

Description

As a researcher, I would like my subject to be able to intuitively pick up objects in VR so that I can observe differences in time perception while the subject is manipulating objects.

Design

For manipulation, we will use the custom events PickUpObject and DropObject as per the documentation[9]. These events will be linked to the trigger of each motion controller such that pulling the trigger causes an item to be picked up, and releasing the trigger drops an object. Figure 9 shows an example of how the custom PickUpObject event can be used to grab an object within 1000 units of the left touch controller. While an object is picked up, we will force the object to match the rotation and translation of the touch controller that grabbed it using the AttachToComponent. All these functions are accessible from blueprints and will be created using the UE4 blueprint editor.

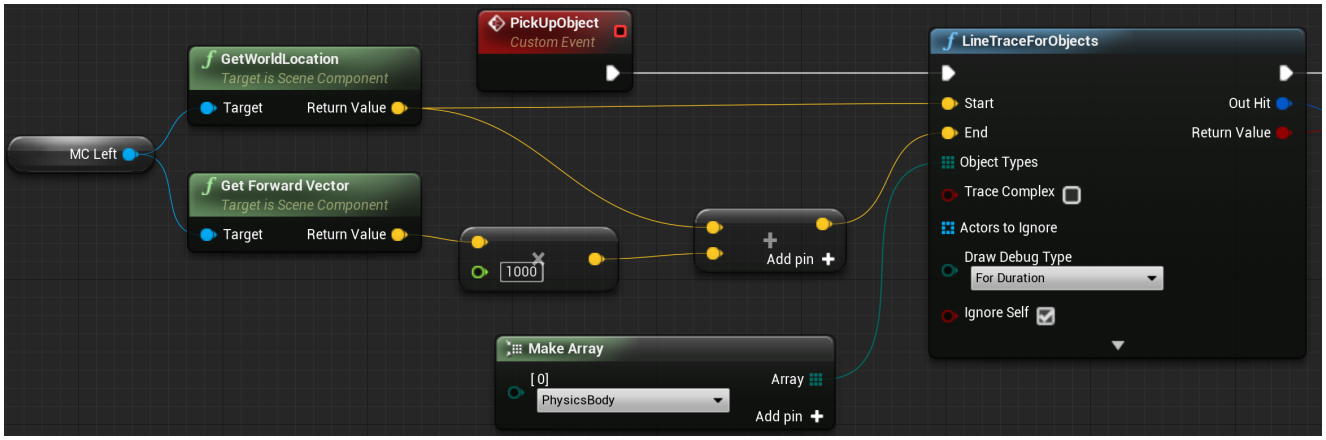


Fig. 9: An example of how to find which object is being picked up. This method fires a ray into the scene in the direction of the motion controller and returns the closest object it intersected of type PhysicsBody[9].

3.6.2 Dynamic Effects

Description

As a researcher, I would like my subject to experience a dynamic environment in VR so that I can measure how their time perception changes in response.

Design

Since the purpose of these effects is to be controlled by the researcher, each effect will expose a set of parameters which can be modified in real time. The effects we have chosen to implement are dynamic lighting and dynamic weather. The following sections outline the details of how these effects will be implemented.

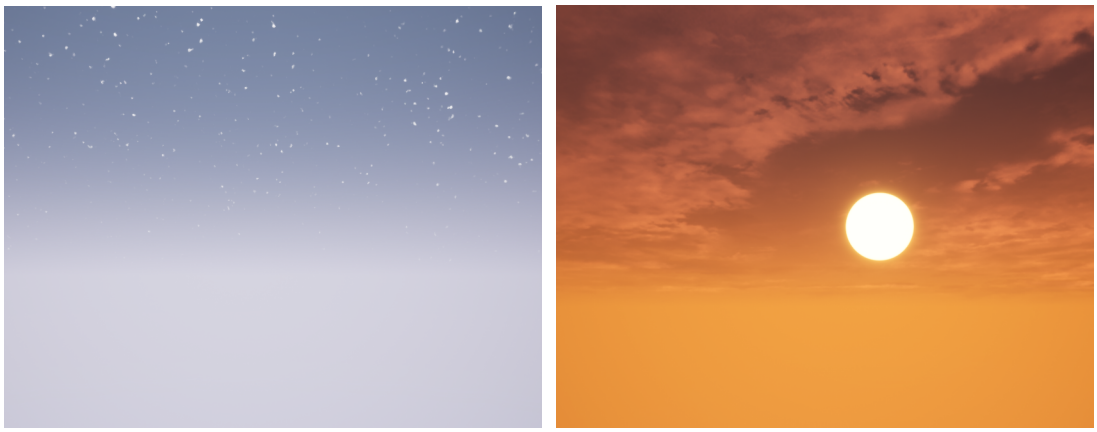
3.6.2.1 Lighting Effects

Description

As a researcher, I would like to be able to change the lighting of a scene in UE4 to determine how lighting affects time perception in VR.

Design

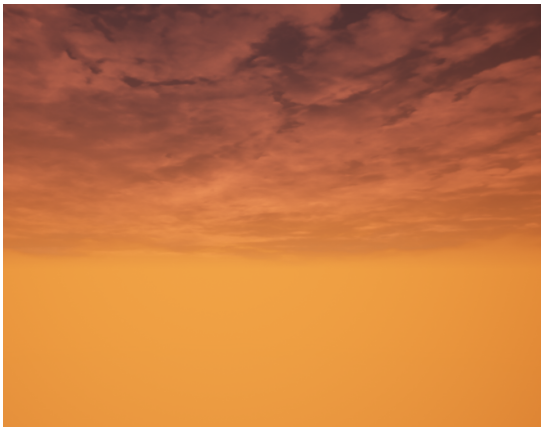
To change the lighting of a scene, we will use GoodSky, a free skybox collection by Unreal Marketplace user UNEASY[10]. To implement Good Sky into a scene, the assets will be imported into the UE4 project, then a BP_GoodSky actor will be placed in the map. The existing skybox actor will be removed so that there is only one skybox. The BP_GoodSky actor implements all the Good Sky features using blueprint functions, and exposes parameters which can be modified in realtime. We will modifiy the parameter ”Time Of Day” which is a float from 0 to 1.0. This value, as its name implies, indicates the time of day in a 24 hour cycle. For the lighting effect, we will adjust the rate at which this parameter increases, thus modifying how long a complete day-night cycle takes. The range will be from 30 seconds to 24 hours to complete a full cycle, giving the researcher a large range of options. The day-night cycle will move between the four lighting presets shown in Figure 10.



(a) Night Sky (0:00)

(b) Dawn Sky (6:00)

(c) Midday Sky (12:00)



(d) Dusk Sky (18:00)

Fig. 10: The four skybox presets in Goodsky which will be used at their corresponding times. Times between the presets will interpolate the parameters to generate each preset.

3.6.2.2 Weather Effects

Description

As a researcher, I would like to be able to change the weather in a scene in UE4 to determine how weather affects time perception.

Design

We will implement a rain weather effect using a UE4 particle system. First, a material will be created for the particle system. We will make the material translucent white with a index of refraction of 1.3. This will cause the raindrop to be visible while distorting the environment through the drop. Next, a particle system will be created using the UE4 particle editor. In the editor, we will enable GPU Sprites to improve performance. We will set the starting position to be 1000 units above the player, and allow the particles to randomize their starting location, velocity and size (Figure 11). We will set the particle collision event to ”destroy” so that particles will destroy themselves upon impact with a physics object such as the ground.

The rain particle effect will expose two parameters to modify the intensity. First is the number of particles and second is the scale of the particles. The number of particles will support ranges from 0 to 1000000, and the scale will

support between 0.2 and 5.0. These values will be exposed to the researcher so they can modify the effect in realtime. Additionally, the rain particle effect will follow the VR camera by attaching to the HMD, per the documentation[11]. This will reduce the overall number of particles which must be drawn because the effect only covers a small area around the player.

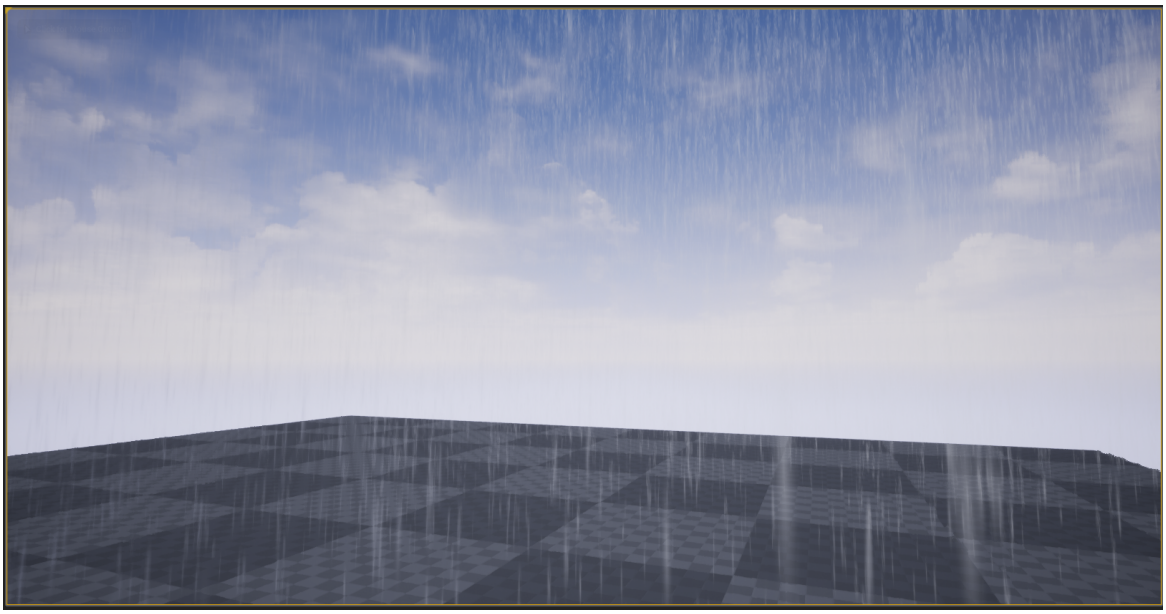


Fig. 11: Rain particles with differing sizes and starting positions.

3.6.3 BioMR API Plugin

Description

As a developer, I would like to be able to use the BioMR API within Unreal Engine 4 so that I can run custom code related to time perception.

Design

To allow the API to be included in a UE4 project, we will encapsulate the API in a UE4 plugin. The plugin type we will use is the Non-Game Module which is an extension or creation of a set of code that exposes an interface to internal code, allowing other modules and game modules to depend upon its functionality[12]. The Plugin will dynamically load the API by including all the header files, and linking a .dll for the implementations. We will use public linking by populating the ”PublicDependencyModuleNames” field in the non-game module build file. In order to call C++ functions from blueprints, we will write wrapper functions that use the ”UFUNCTION” macro for all public functions in the API. In addition, we will create the following interfaces.

3.6.3.1 Activity Detection

Description

As a researcher, I would like to know when my subject is navigating or manipulating an object in VR so that I can correlate their biometric data with an activity.

Design

When a user has an item picked up, they will be considered manipulating. If the user is moving around the world via teleportation, and they have moved in the last 3 seconds, they will be considered navigating. This will be measured by a countdown timer attached to the player object each time they teleport. To notify the API about these events, we will call a public function in the API which accepts the activity type. The API function will store the value in the database. See section 3.5.3 for more API details.

$$activity = \begin{cases} \text{Item in hand} & Manipulating \\ \text{Last moved} < 3 \text{ sec} & Navigating \\ \text{Otherwise} & None \end{cases} \tag{1}$$

3.6.3.2 Effect Modification

Description

As a researcher, I would like to be able to directly modify lighting and weather conditions through the BioMR API in realtime so I can measure how these changes affect time perception.

Design

Upon creation of a dynamic effect, we will register a callback function with the BioMR API which can be used to modify the effect. The callback function (see Figure 12) will take a type, such as an int or float, which will be assigned to the corresponding parameter. For example, a callback function for raindrop size can be registered with type float. A row in the BioMR API user interface appears showing the default value for raindrop size. When the researcher modifies this value, a synchronous call will be made to the registered function. We will create three different callback functions, one for each of the exposed parameters: rain drop size, rain drop count, and day-night cycle speed.

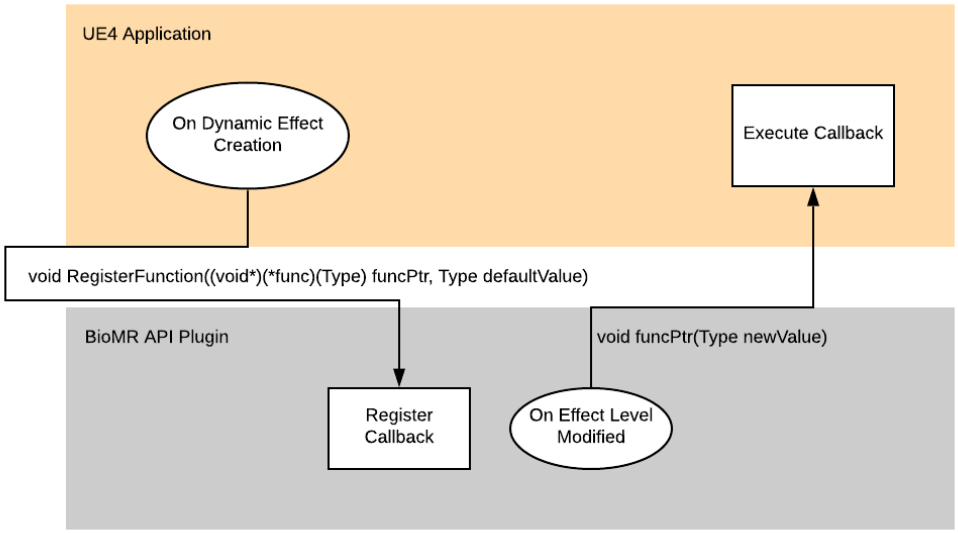


Fig. 12: The API will allow callback functions to be registered by passing a pointer to the function and a default value. When the field in the BioMR UI is modified, the callback function will be called to modify the UE4 scene.

3.7 Unity Design

The framework that will be developed must be usable for application development in the Unity engine. As a proof of concept for this feature a Unity VR application needs to be developed that can allow for user interaction, detect when an interaction is taking place, send and receive data from the BIO-MR API, and alter the lighting and weather effects of the application according to the data received.

3.7.1 VR Scene and Interface

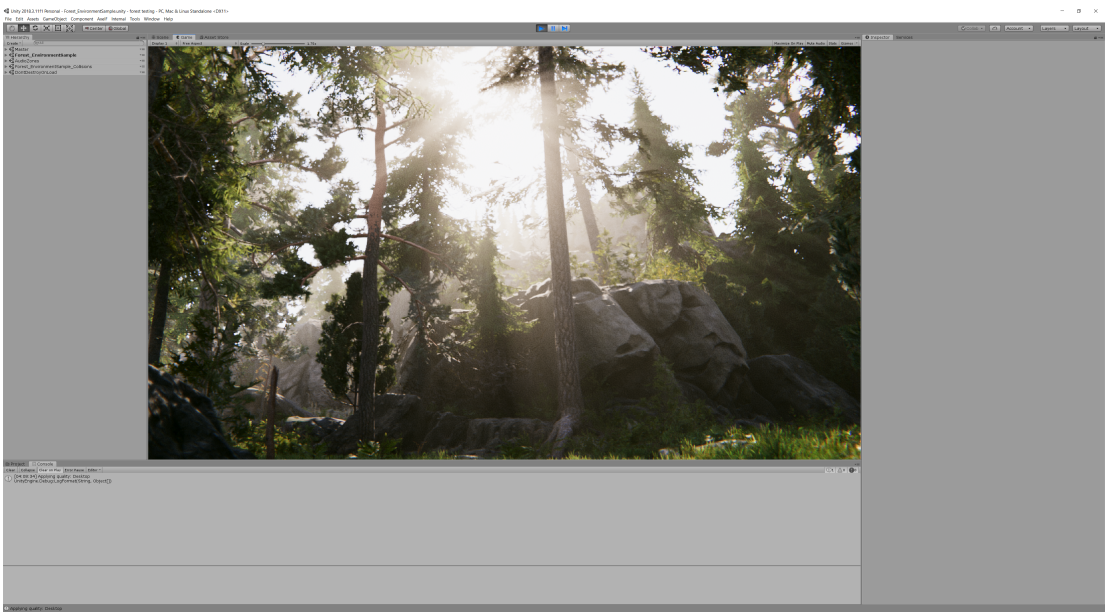


Fig. 13: Book of the Dead scene

Description

As a user I would like to to be able to view and interact with a scene in virtual reality.

Design

The scene that will be used will be based on the ”Book of the Dead” environment from the Unity demo in 2018. This scene will be adapted to be usable in VR, and contain objects that can be interacted with by the user. The SteamVR Plugin for Unity will be used in order to create a Virtual Reality camera and interactions, allowing for cross platform support for any VR headset that SteamVR supports. The SteamVR cameraRig can be brought into the Book of the Dead scene and SteamVR Input can be customized to include teleportation for navigation, and grabbing for manipulation.

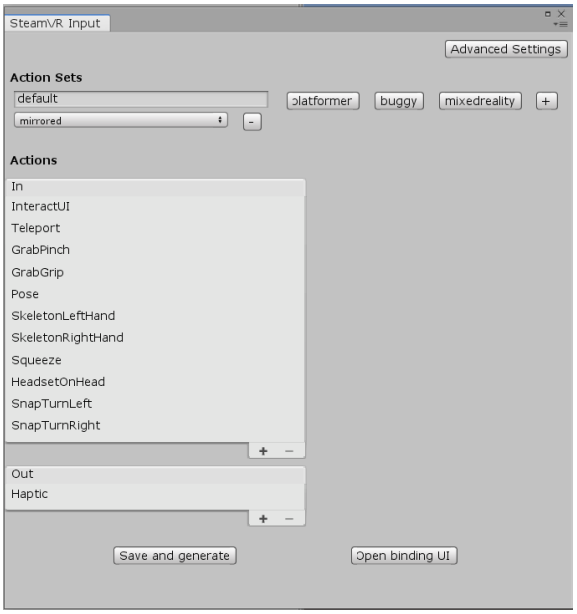
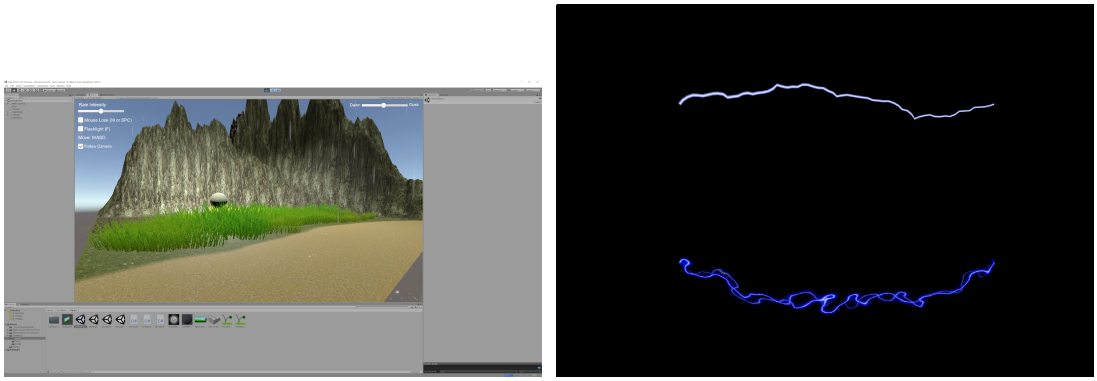


Fig. 14: Built in interaction options in SteamVR

3.7.2 Weather and lighting effects



(a) Rain Example (b) Lightning asset

Fig. 15: Weather effects like rain and lightning can be imported from the asset store.

Description

As a user I would like to to be able to have varying lighting and weather effects.

Design

The first part of this is to implement the effects themselves. Several free asset packages from the Unity asset store will enable the application to change weather conditions such as rain, wind, lightning, and time of day in the scene. The Book of the Dead scene has adjustable wind settings built into it. Lighting is also already built into the scene and can be adjusted to make the scene darker or brighter. Free external packages for lightning, rain, and skyboxes will also be used. The AllSky Free package will provide the sky boxes[13], the Lightning Bolt Effect for Unity package will be used for the lightning[14], and the Rain Maker package will be used to generate rain[15]. The next step is to make the effects dynamic. These assets will each be attached to a Unity script that will be able to turn them on and off and adjust them. For example the wind in the book of the dead can be adjusted to to change it's direction, and intensity. The skybox can be switched out and light intensity changed to alter the time of day. Lighting and rain can be activated and deactivated in the scene to created dynamic weather patterns. This will become useful when they are adjusted according to the BioMR data received from the framework.

(a) Dusk Skybox

(b) Daytime skybox

(c) Nighttime Skybox

(d) Overcast Skybox

Fig. 16: The Skyboxes in All Sky Free allow for different times of day, different amounts of light and different weather.

3.7.3 BioMR API Plugin

Description

As a user I would like to to be able to use the BioMR API in Unity.

Design

The framework that will allow for communication from the biometric sensors will be written in c++ or some other language other than c#. Because Unity only supports c# scripts and JavaScript, a Unity application cannot communicate directly with the framework. Unity does however support dynamically linked libraries (or DLLs) to be imported as a plugin[16]. From there a dedicated script can be used as a middle layer to import the functionality of the DLL into the Unity project. The application will then need to be set up in a way that will allow for the middle layer script to access

the Unity game objects that control the lighting, weather, and time of day. The middle layer will link the various game objects to the output from the plugin functions allowing for the application to adjust objects according to the biometric sensor data. The middle layer will also need to know when the user is interacting with objects in the scene. The plugin functions can take this data as input and tell the middle layer which objects it should alter accordingly.

3.8 User Study

A user study will be conducted to evaluate the completed product through the use of the application in a research experiment. Based on the findings of the study, revisions will be made to the final product design before delivery.

By analyzing the validity of the research through cross-comparison of its qualitative and quantitative data results, we can determine the accuracy of our data collection system. Furthermore, by conducting field study observation on the use of our system in its intended function, we can identify flaws in user experience and in the system's technical execution. The research study itself will explore the system's effect on user time perception, which will be used to evaluate the system's performance.

The details of the research study are as follows:

3.8.1 Research Questions

- How do characteristics A, B, and C of the virtual environment affect a user's subjective estimation of the duration of a navigation task in virtual reality?
- How do characteristics A, B, and C of the virtual environment affect a user's physiological evaluation of time during a navigation task in virtual reality?
- What relationship does a user's ability to accurately estimate the length of a short duration spent outside of VR without a task have with their ability to accurately estimate the duration of a task within VR?

3.8.2 Procedure

3.9 Timeline

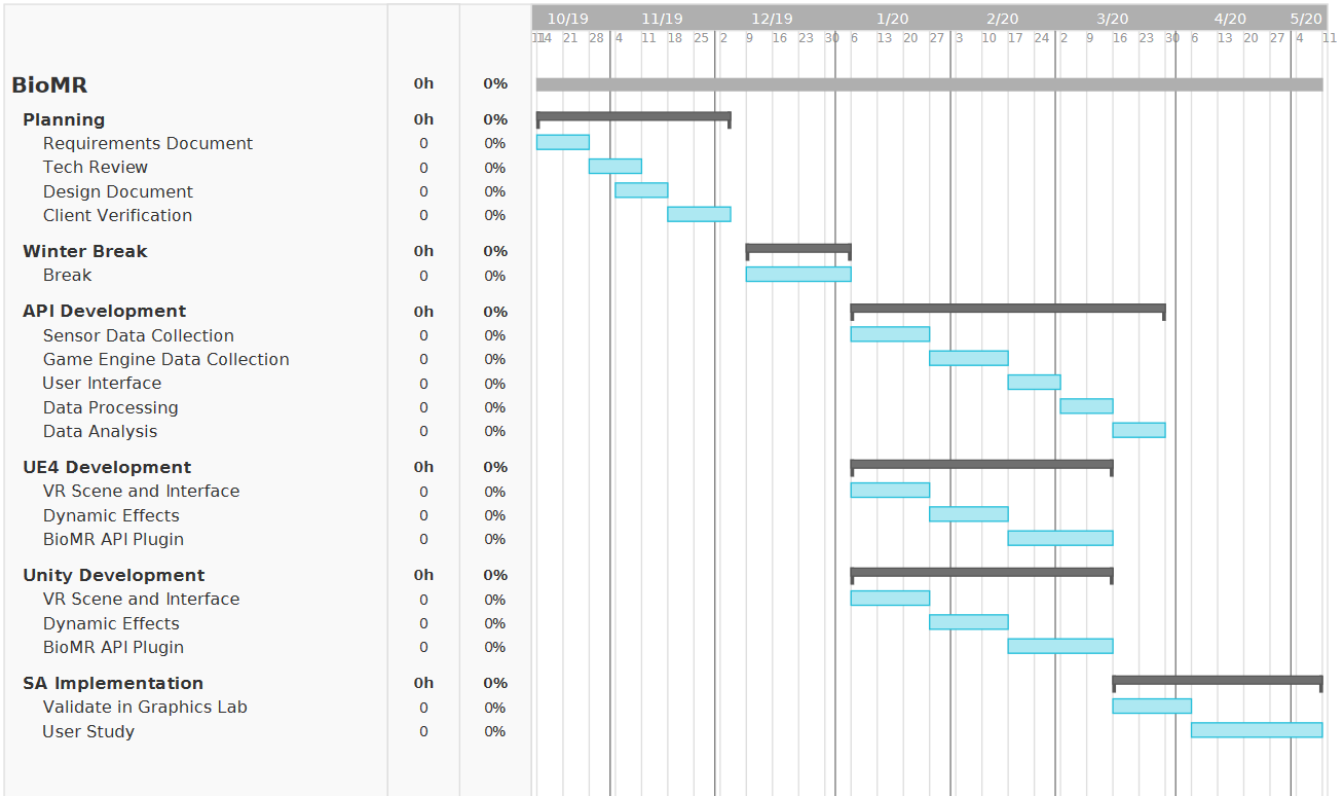


Fig. 17: Our planned schedule of work for the completion of this project by the Engineering Expo in May. Note that the API ”Game Engine Data Collection” must be completed before the ”BioMR API Plugin” can be started for either game engine.

REFERENCES

[1] K. Lontz, "Time perception during retrospective and prospective paradigms with a distraction," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 59, pp. 1367–1371, 2013.

[2] B. Wandell and S. Thomas, "Foundations of vision," *PsycCRITIQUES*, vol. 42, July 1997.

[3] D. Vernon, T. BSc, O. Bazanova, N. PhD, M. PhD, and S. Andersen, "Alpha neurofeedback training for performance enhancement: Reviewing the methodology," *Journal of Neurotherapy*, vol. 13, pp. 214–227, October 2009.

[4] A. M. Berger and E. J. Davelaar, "Frontal alpha oscillations and attentional control: A virtual reality neurofeedback study," *Neuroscience*, vol. 378, pp. 189–197, May 2018.

[5] Epic Games, "A Boy and His Kite," 2015, https://www.unrealengine.com/marketplace/en-US/item/378a4f30ab5e4a168f5f87a1039461f2.

[6] ——, "Volumes Reference," 2019, https://docs.unrealengine.com/en-US/Engine/Actors/Volumes/index.html.

[7] ——, "Set Up a Standing Camera for SteamVR," 2019, https://docs.unrealengine.com/en-US/Platforms/VR/SteamVR/HowTo/StandingCamera/index.html.

[8] ——, "Motion Controller Teleportation," 2019, https://wiki.unrealengine.com/VR_Template#Motion_Controller_Teleportation.

[9] ——, "Using Motion Controllers," 2019, https://docs.unrealengine.com/en-US/Platforms/VR/DevelopVR/UsingTouchControllers/index.html.

[10] UNEASY, "GOOD SKY," May 2018, https://www.unrealengine.com/marketplace/en-US/item/a50e527d893d4182a3488572398064a8.

[11] Epic Games, "Attaching Items to the HMD," 2019, https://docs.unrealengine.com/en-US/Platforms/VR/DevelopVR/VRCameraRefactor/index.html.

[12] ——, "An Introduction to UE4 Plugins," 2019, https://wiki.unrealengine.com/An_Introduction_to_UE4_Plugins.

[13] RPGWHITELOCK, "AllSky Free - 10 Sky / Skybox Set," May 2019, https://assetstore.unity.com/detail/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014.

[14] Digital Ruby, "The Lightning Bolt Effect for Unity," May 2017, https://assetstore.unity.com/detail/tools/particles-effects/lightning-bolt-effect-for-unity-59471.

[15] ——, "Rain Maker - 2D and 3D Rain Particle System for Unity," April 2019, https://assetstore.unity.com/detail/vfx/particles/environment/rain-maker-2d-and-3d-rain-particle-system-for-unity-34938.

[16] Unity Technologies, "Native plug-ins," 2019, https://docs.unity3d.com/Manual/NativePlugins.html.