TABLE OF CONTENTS

Abstract

Mental illnesses are suffered by millions of people around the world, yet for those who suffer from it, it is still difficult to seek help because of the stigma that it carries.

Luckily, new possibilities have been created for mental treatment due to the advances in technology and virtual reality: creating safe intimal spaces where people can heal and learn is no longer a thing of the future.

Creating a digital metaverse can help to destigmatize mental illnesses. It can be of social benefit to provide people with the resources they need to manage their symptoms. With this help, people will feel accepted and cared for, and can open the possibility to build a community that supports and empowers those who are struggling with mental health issues.

Find the documentation, installation files and unity project files within the GitHub repository: https://github.com/PauAnton/Metaverse-for-mental-illness

1. INTRODUCTION

1.1 Object

The primary goal of this project is to create and develop a metaverse setting specifically tailored to support mental health treatments. The focus lies on designing a virtual environment that encompasses suitable spatial arrangements, captivating aesthetics, and engaging gameplay elements, all with the intention of serving as a conducive foundation for hosting therapeutic activities within the metaverse.

The digital metaverse envisioned provides a supportive, immersive environment where people can access information related to their pathology, have guided therapy and guided meditations. Creating a space where therapies and other mental health services can be implemented in a way that is accessible, affordable, and inclusive is the objective.

In addition, another objective of the project is to test and research technological innovations. Developing an ideal environment like such could take years, plus it would have to be constantly updated and tested to match the requisites of a real person. Therefore, in this project one of the focuses will be put on which are the new technologies available for the creation of a space that can be updated, modified and tweaked to match consumer requisites, and the real possibilities that these offer.

1.2. Scope

The development process followed a methodology known as "baby-steps," which emphasizes accomplishing small tasks at a time to minimize time spent in case of any unexpected deviations. This approach also enables the flexibility to expand or refine the project with additional elements or adjust the level of detail based on the actual time required for each task.

Although the outcome may vary, a defined set of minimum goals and objectives were established to ensure the project's success and maintain organizational alignment.

- Investigation of precedent work involving mental health care solutions in the digital era.
- Evaluation and selection of technologies for development.
- Sketch of metaverse environment.
- 3D design of the metaverse.
- Integration of an avatar
- Project report.

1.3. Specifications and requirements

While the project does not adhere to a rigid set of predefined standards, certain specifications and requirements were intentionally established to guarantee future feasibility.

- Use free software.
- Minimize the size of the resultant application.

1.4. Justification

The project's inception stems from the collaboration of Jordi Cusidó Roura, a university teacher and developer deeply involved in healthcare solutions. Jordi envisioned a metaverse designed specifically for the treatment and support of individuals with mental health

conditions. The overarching objective of the project is to explore the current technologies available for metaverse development and assess their feasibility as effective solutions for mental health treatment.

To ensure efficient progress, Jordi devised a plan to divide the project into three distinct components: the creation of the metaverse environment, the development of a user avatar, and the implementation of an AI-powered avatar to serve as a medical representation.

Each segment was assigned to a different student: The project present refers to the first of the three parts: the creation of the metaverse environment.

2. BACKGROUND AND STATE REVIEW

2.1. Metaverse

The term "metaverse" refers to a virtual reality space where users can interact with a computer-generated environment and other users in real time. It is an immersive and interconnected digital universe that combines elements of augmented reality, virtual reality (VR), and the internet.

The concept of the metaverse originated from science fiction and has gained popularity in recent years with the advancement of virtual reality and augmented reality technologies. The term was first coined by Neal Stephenson in his 1992 science fiction novel "Snow Crash" [1] where he described a virtual world accessed by users through a global computer network.

Since the end of the 2010s decade several companies and projects have been exploring the development of the metaverse concept. Technology giants like Meta, Google and Nvidia showed interest and made investments in building virtual reality platforms and technologies.

In light of the emergence of AI in recent years and its significant social influence, certain prominent companies appear to prioritize AI over other technologies such as metaverses. However, this doesn't imply that metaverses will become obsolete. For numerous individuals, the metaverse will serve as a space where numerous AI activities can occur and be seamlessly integrated. This is why many companies continue to place emphasis on metaverses as a core aspect of their business.

2.1.1. State of art of metaverses

Second Life was the first platform to be launched as a metaverse in 2003 as it incorporated many aspects of social media into a persistent three-dimensional world with the user represented as an avatar. Nowadays It provides a digital space for users to socialize, explore, create content, and engage in various activities.

Another platform which operates similarly to Second Life is Spatial.io, but differs by carrying the development in the external software of game development Unity.

As for the bigger companies, in 2019 Facebook (now Meta) launched a social VR world called Facebook Horizon and announced in 2021 that their focus would be on metaverse development. Despite that, in 2023 they announced to pivot away from metaverses to focus on AI. Microsoft used to have a part in metaverse too, with the virtual platform AltspaceVR, but on the 10th of March of 2023 the platform was shut down [2].

On the other hand, NVIDIA developed the platform Nvidia Omniverse [3] which is self-described as "The platform" for creating and operating metaverse applications. It aims to revolutionize the way people work together and create interactive virtual worlds. Omniverse provides a common platform for designing, simulating, and visualizing complex 3D environments, enabling real-time collaboration among multiple users and platforms.

2.2. Metaverses for mental health

The integration of metaverse technology for mental health aims to leverage the immersive and interactive nature of virtual environments to provide innovative approaches for therapy, treatment, and support. By creating virtual spaces and experiences, individuals can engage in therapeutic activities, receive interventions, and explore therapeutic techniques in a controlled and customizable manner.

Many studies are being undertaken in the current time to demonstrate or bust the usefulness of metaverses in mental health and VR applications. At the moment VR and interactive games have already shown promise in various mental health applications, such as exposure therapy for phobias, PTSD treatment, anxiety management, and relaxation techniques [4].

2.2.1. State of art of metaverses for mental health

Because it is a still-studied field and its success is not yet fully determined, not many companies have yet launched a fully focused platform for mental health but there are two notable platforms standing out regarding metaverses and VR.

Inner World [5] is a digital metaverse platform dedicated to personal growth, mindfulness, and well-being. With its curated collection of content, Inner World offers a range of resources such as guided meditations, mindfulness exercises and group therapies. By providing a convenient and accessible platform, Inner World empowers individuals to prioritize their mental and emotional health. The social community aspect allows users to connect with like-

minded individuals, fostering a sense of support and belonging. Inner World can be installed in MacOs, Windows and found in Oculus or Meta.

On the other hand, the company PsyTech [6] developed a VR Training System which focuses on leveraging virtual reality technology to provide mental health interventions. By creating immersive environments, users can engage in therapeutic experiences and gain valuable insights into their mental well-being. This system offers a unique approach to therapy, allowing users to confront fears, manage stress, and develop coping strategies within a controlled and safe virtual setting. It is available on Meta, ViveFocus and Pico3&4 platforms.

3. DESIGN PROPOSAL

3.1. Software

As we embark on the project, it is imperative to carefully select the software tools that will facilitate efficient streamline workflows, keeping in mind the initial requirements. Considering the diverse areas encompassed by the project, such as coding, 3D modeling, databases, and servers, the decision-making process involved leveraging the knowledge gained throughout our degree program.

3.1.1. Maya

Maya offers comprehensive 3D modeling and animation tools specifically designed for game development. Its robust features enable efficient creation of high-quality assets, realistic character animations, and captivating visual effects. Maya seamlessly integrates with popular game engines, streamlining the asset pipeline. Being an industry standard, Maya provides extensive resources and community support.

Having had spent time in Maya with other projects and thus, mastering Maya's interface, keyboard commands, and shortcuts provides a compelling justification for choosing it as the primary 3D modeling application. This familiarity with Maya enhances efficiency, productivity, and ensures a smoother workflow.

In terms of licensing, it should be noted that Maya, being an Autodesk program, is not available for free. However, the utilization of Maya under the student license, offered by the same company, allowed for its legal and cost-free usage.

3.1.2. Unity

Unity is a powerful and versatile game development platform that is primarily used for creating interactive and immersive experiences across various platforms. Unity is widely used for developing video games for different platforms, including mobile, desktop, console, and virtual reality (VR) devices. Its intuitive interface, extensive library of assets, and robust scripting capabilities make it a popular choice among game developers.

Unity Editor provides an intuitive and user-friendly interface, making it accessible to developers of all skill levels. Its visual layout and well-organized menus allow for easy navigation and efficient workflow management. One of the standout features of the Unity Editor is its real-time editing and preview capabilities. Developers can make changes to the game's assets, scenes, and code, and instantly see the results in real time.

Despite my limited experience with Unity, primarily consisting of a single car game project, I determined that Unity is well-suited for the requirements of this particular project. Furthermore, the Unity version “Unity Personal” is free to use for individuals or corporations under 100k yearly revenue and provides a comprehensive set of features and tools for individuals or small teams.

3.1.3. Nvidia Omniverse

In accordance with Jordi's initial project concept of dividing it into three distinct parts, the need arose for a technology that could effectively connect these parts. Given the collaborative nature of the project involving multiple individuals and considering the current advancements in the field, the software initially chosen for this purpose was Nvidia Omniverse.

Nvidia Omniverse facilitates seamless collaboration among project participants while offering free usage. Moreover, it boasts compatibility with prominent 3D development software such as Maya and Blender, as well as the ability to integrate Unity projects.

During the initial phases of the project, a significant challenge arose in relation to the usage of the Nvidia Omniverse. Specific hardware requirements were necessary, such as an RTX Series graphics card, which was not available at the time. Additionally, while a cloud server was intended to be used for operating with Nvidia Omniverse, access to the cloud server was not granted.

Despite not having this specific software available, the project could still be carried out independently. Avatar and avatar animations would be included in the metaverse. Nevertheless, the project would be carried out to allow for future work and updates to implement the AI-powered avatars created by other students involved in the overarching project.

3.2. Functionalities

The initial project requirements entailed establishing an environment for the implementation of avatars. To set a foundation and streamline workflows, several core functionalities were identified at the project's outset. These functionalities included:

- **Scene Navigation:** Creation of mechanisms for users to navigate and explore different scenes within the project environment, providing a seamless and intuitive user experience.
- **User Menu:** Development of a user menu interface that enables users to access and control various features, options, and settings within the project environment.
- **Communication with Web Server:** Integration of communication protocols and mechanisms to establish a connection with a web server, facilitating data exchange, login and registration.
- **Avatar:** Implementation of avatars with animation capabilities, allowing for realistic and dynamic movements of the avatars within the environment.

Additionally, with the initial functionalities completed within the project timeline, there was a possibility to incorporate further enhancements to improve the overall user experience. These functionalities were set to be:

- **Image displayer:** displays visual content related to the user's pathology.
- Guided meditations through YouTube videos displayed inside the metaverse

3.3. Digital environment

The designed environment must prioritize respect and evoke a sense of tranquility to ensure user comfort. It is crucial to bear in mind that the intended audience for the final product consists of individuals who are either mentally ill or in the process of recovering from a mental health issue. Therefore, great care must be taken during the design process to create a safe and respectful environment that caters to their specific needs.

The characteristics of a safe environment could involve specifications such as:

- **Supportive Atmosphere:** The environment should evoke a sense of calmness and positivity by using vivid colors and lighting and spacious rooms.
- **Non-Threatening Design:** The design should avoid elements that may trigger anxiety or distress. It should be free from disturbing or violent imagery, sudden movements, or loud sounds.
- **Accessibility and Ease of Use:** The environment should be designed to accommodate individuals with different abilities and needs.
- **Professional Support and Resources:** Incorporate access to mental health resources, helplines, or information within the metaverse.

The digital environment will primarily be modeled using Maya software, with the intention of exporting it into a Unity project where the metaverse will be developed. In Maya, objects will be created with a focus on maintaining unities in centimeters (cm) and independent modeling to facilitate seamless assembly within the Unity Editor. This approach aims to optimize the workflow and ensure efficient integration of the modeled elements into the Unity project.

4. BACKGROUND KNOWLEDGE

This section aims to provide a theoretical explanation of the various technologies and software utilized in the project. The objective is to offer the reader an overview of how these components function and to provide them with the necessary knowledge to comprehend the project's development outlined in Section 6 and 7. Rather than delving into the specific details and steps of each component, the focus will be on establishing a conceptual understanding of their workings. By presenting this higher-level perspective, readers will gain a comprehensive understanding of how the project has taken shape without the need for an exhaustive breakdown of every individual step or component.

4.1. Autodesk Maya

4.1.1. Maya API

To provide insight into how Maya operates, it is beneficial to focus on its Application Programming Interface (API) and its associated components. The API serves as a bridge between Maya's core functionality and external tools or scripts, enabling developers to extend and customize the software's capabilities.

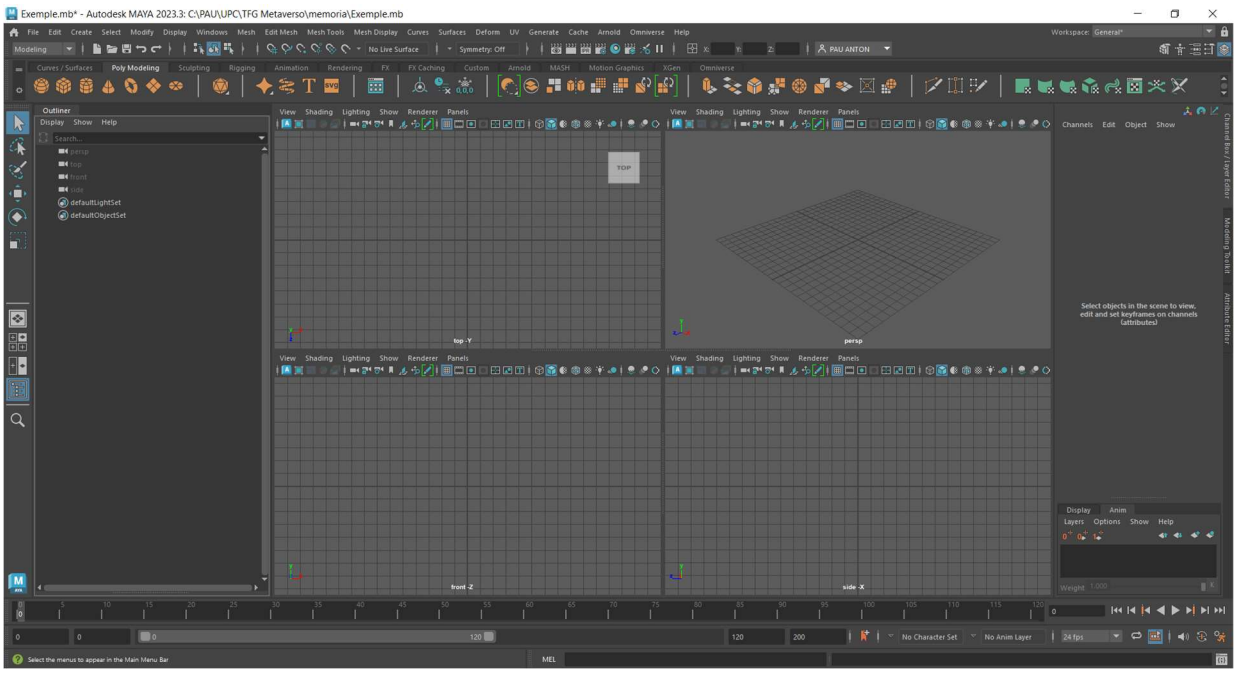


Figure 1: Maya API by default

To enhance productivity and facilitate the various stages of development, Maya offers different workspaces that automatically open relevant tabs and panels. These workspaces are tailored to specific tasks or workflows, streamlining the user interface and providing quick access to essential tools and features.

4.1.1.1. General Workspace:

Toolbar:

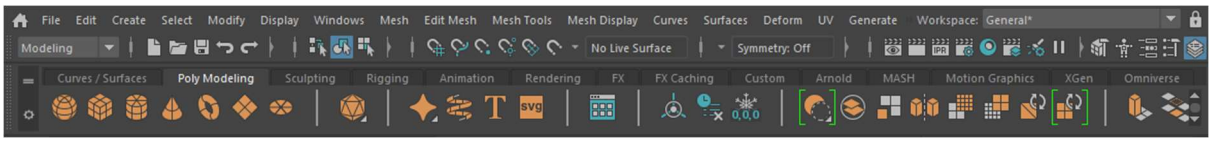


Figure 2: Maya Toolbar

Placed at the top.

The first row contains the kit of basic functionalities (from “File” to “Windows”) and additional features depending on the type of work aimed to do (modeling, rigging, FX, etc.), selected on the selectable placed at the second row in the left part (from “Mesh” to “UV”). The workspace selectable contains parameters to open tabs accordingly (for example: UV Editing opens UV Editor, UV toolkit, and Hypershade tab).

The second row contains shortcuts to some of the basic functionalities (file management, snapping tools, symmetries, etc.)

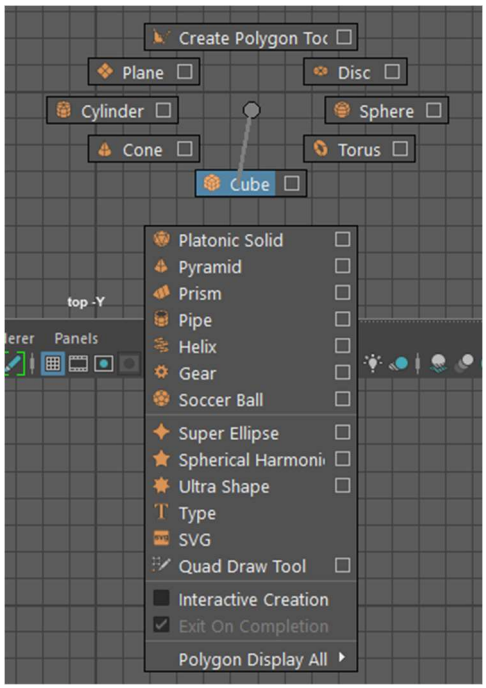


Figure 3: Maya Shift+MB3 shortcut

The third row contains the icons to generate and create surfaces, polygons, etc. Note that not all objects are shown in this toolbar because the most effective way to create is by keyboard and mouse shortcuts. For example, by pressing Shift+MB3 we open the PolyModeling Toolkit

Selection and manipulation tools and Panel Layout:

Found at the left part of the screen.

The first row shows the selected tool for selection (Q), moving (W), rotating (E) and scaling (R). These tools are easily accessed by pressing the keys rather than clicking on them.

The second row on the left is used to change between panel layouts. By pressing Spacebar, it automatically switches from one panel to 4 layout panels. The Outline is the last of the options. The outline tab shows the different objects found in the scene (cameras and lighting) but will also show all the meshes, transformations, groups, etc.

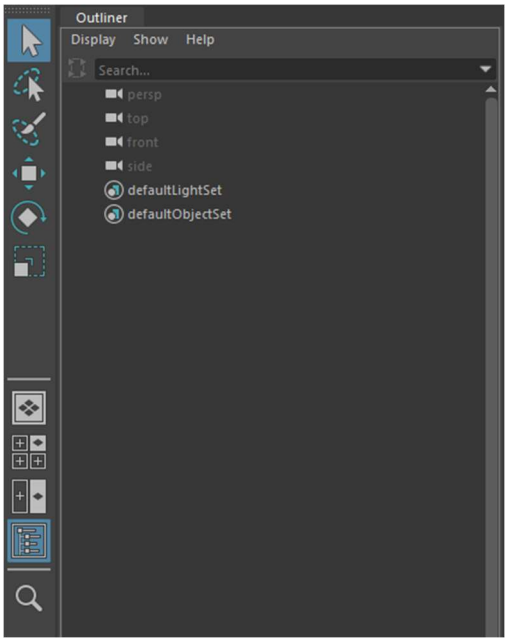


Figure 4: Maya Outliner

In the third row the magnifying glass is used to manually search for a specific tool.

Panel Layout:

The panel layout is where the work can be visualized.

By default, it comes with the top and sides view and one free perspective. Those can be changed with the different tools available.

By pressing the spacebar, we can switch between the 4-panel view or the single perspective panel view.

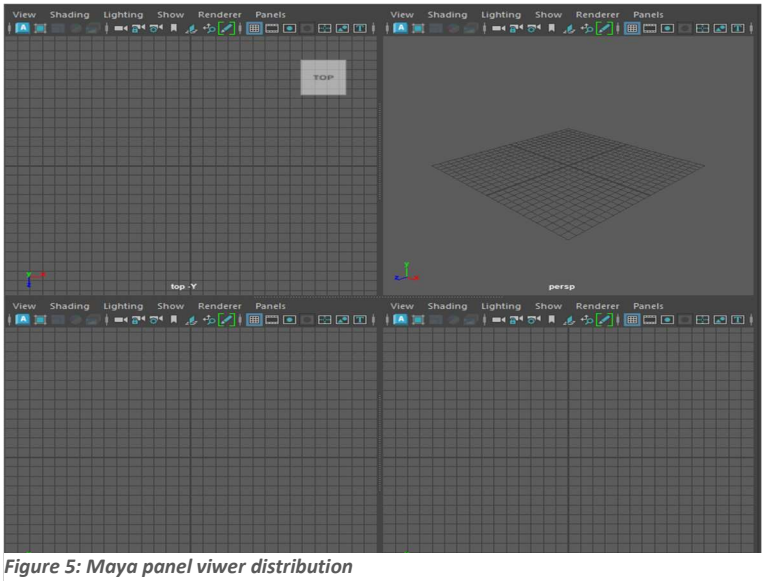


Figure 5: Maya panel viwer distribution

Channel Box/Layer Editor and Attribute editor:

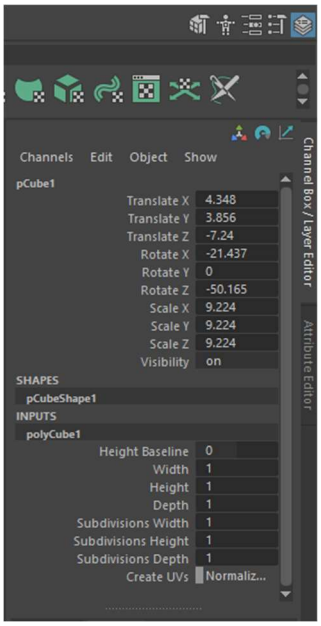


Figure 6: Maya Channel Box

Found at the right side of the screen.

When an object is selected in the channel box it is shown information about its characteristics regarding the position in the scene and the shape (this last also found in the attribute editor).

As for the attribute editor, it shows each object the components it has (shape, polygon characteristics, shading, material, etc.)

With the icons at the right side of the tool bar we can enable/disable these tabs or add others like the modeling toolkit, toggle character controls, etc.

Playback and frame timeline:

Placed at the bottom of the screen.

It contains the basic tools for animations and changes in time. It allows playback features, set animation time, etc.

4.1.1.2. UV Editing Workspace

Maintains the tabs of general workspace except the timeline and opens up three tabs useful for UV Editing (texturizing and material properties).

UV Editor:

Placed by default between the panel views and the attribute editor.

It serves as a dedicated workspace for manipulating and editing the UV maps of a mesh. UV mapping is the process of flattening the 3D surface of a model onto a 2D plane, allowing textures to be applied accurately.

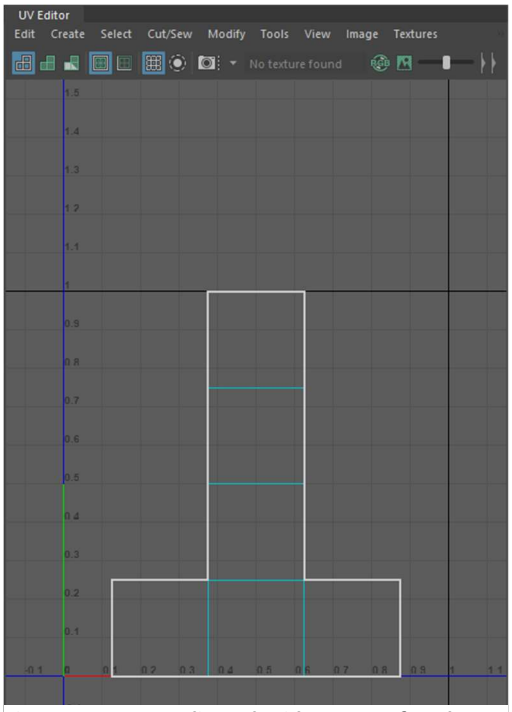


Figure 7: Maya UV Editor tab with UV map of a cube

UV toolkit:

The UV Toolkit in Maya is a panel that opens in the same space where the attribute editor is located. It provides a comprehensive set of tools for manipulating UV maps.

Within the UV Toolkit, there's a range of tools that facilitate various tasks related to UV mapping such as generation tools to assist in generating initial UV layouts automatically or transformation tools which offer tools for scaling, translating, and rotating UVs.

Hypershade:

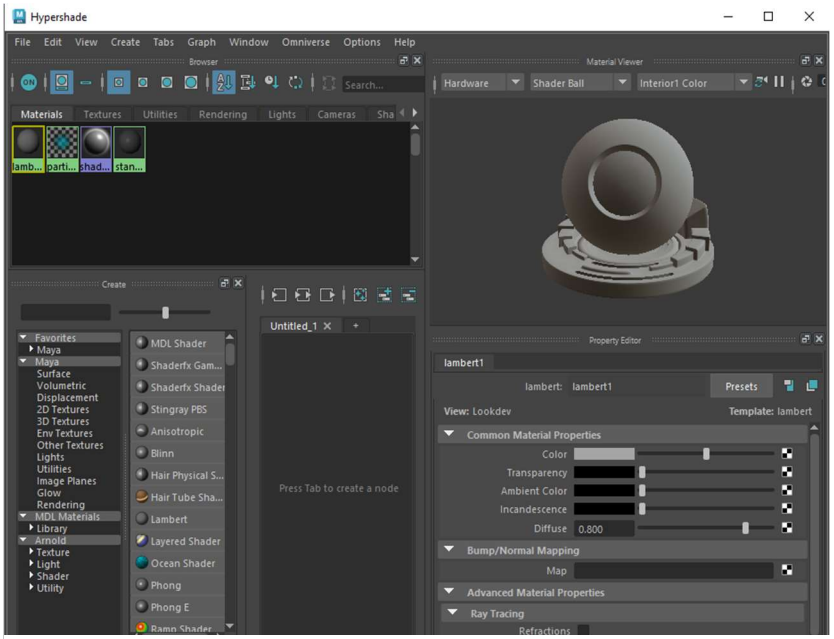


Figure 8: Maya Hypershade tab with the default materials

Hypershade, an essential component of Maya, is accessed as an independent tab due to its comprehensive collection of features related to materials, shading, and texture properties.

Within the Hypershade tab, there is a general list of materials, textures, and other elements located in the middle section of the interface. This section provides an overview of available resources for creating and manipulating materials.

Additionally, Hypershade allows users to create new materials, modify texture mapping, and adjust material properties. Users can also preview the appearance of materials on objects using a default shader ball or a customized viewer.

4.1.2. Modeling process

Explaining the theoretical process of creating a 3D model can be complex. To simplify the explanation, let's break down the steps involved in creating a basic shape and apply it to and example:

1. **Geometry Creation:** The initial step is to create a base geometry using primitive shapes such as cubes, spheres, or cylinders. These basic shapes serve as the foundation for constructing more complex objects.
2. **Edge Loops and Subdivision:** To add more detail and smoothness to the model, edge loops are introduced. Edge loops are sequences of connected edges that define specific regions of the geometry. By strategically placing and manipulating edge loops, artists can enhance the shape's curvature and surface flow. Subdivision surfaces can also be applied to achieve smoothness by subdividing the polygonal geometry.

3. **Polygon Editing:** Once the base geometry is in place, polygon editing tools come into play. These tools allow designers to manipulate vertices, edges, and faces to refine the shape and achieve the desired form. Operations like extruding, beveling, scaling, and smoothing help shape the geometry into the desired design.
4. **UV Mapping:** UV mapping is crucial for applying textures to the model accurately. It involves creating a 2D representation (UV map) of the 3D surface, allowing textures to be placed and aligned properly. Artists use tools like the UV Editor to unfold the geometry, adjust seams, and layout UV coordinates.
5. **Texture and Material Application:** Once the UV mapping is complete, textures and materials can be applied to the model. Artists can create or import textures and define their properties, such as color, transparency, and reflectivity. Materials can be assigned to specific regions of the model, enhancing its visual appearance.

4.1.3. Modeling example: round wooden table

To exemplify the modeling process, a basic round table will be modeled similar to the one present in the final version of the application.

4.1.3.1. Geometry creation

Start by creating a cylinder.

We can rename it. This is important because when having multiple objects in one scene it helps to stay organized.

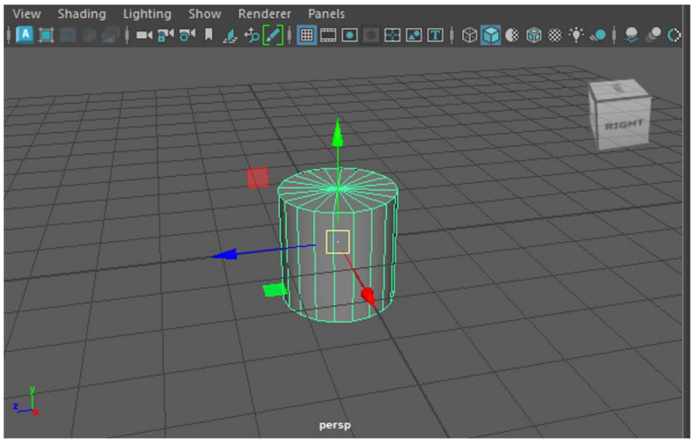


Figure 9: A default cilinder on Maya prespective view panel

4.1.3.2. Edge Loops and Subdivision:

In the channel box we can decrease the subdivisions axis to 8 to simplify the shape and reduce the number of faces.

Add 3 subdivisions caps. This will allow for further manipulation to create legs or add extra detail.

4.1.3.3. Polygon Editing:

Scaling: Set the radius to 75 (remember we are working in a system where 1 unit = 1 cm) and height to 4, to create a table of 1.5 m in diameter and 4 cm thick.

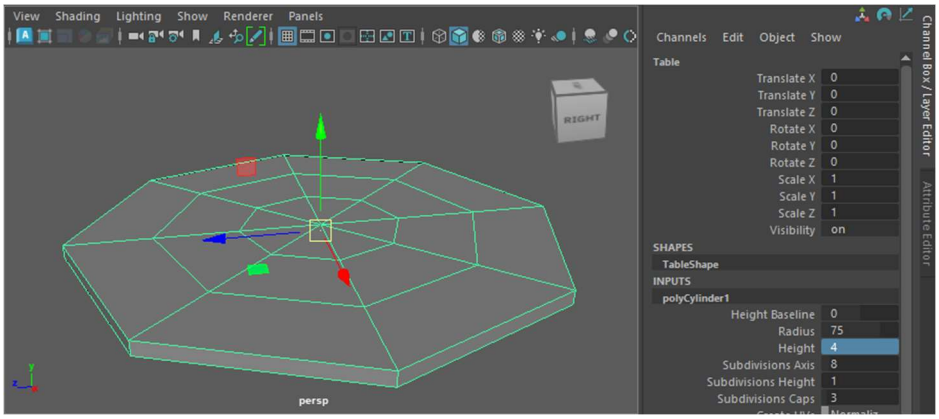


Figure 10: The cilider with diferent subdivisions

Select (double click) the outer edge (not the border) and scale it (R) to take it close to the border. Then with the edge loop tool (in object mode, Shift+MB3) with the equal multiplier checked we can create a vertex. Select the faces and extrude to create a leg (Ctrl+E). We can extrude multiple times to adjust size and create a more elegant shape.

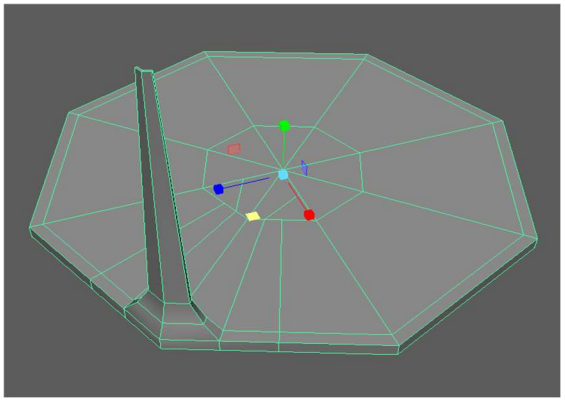


Figure 11: Result of extruding faces to create a leg

For symmetrical objects the mirror tool is very useful. We can erase ¾ parts of the mesh faces and leave the ¼ which has been modified to mirror it.

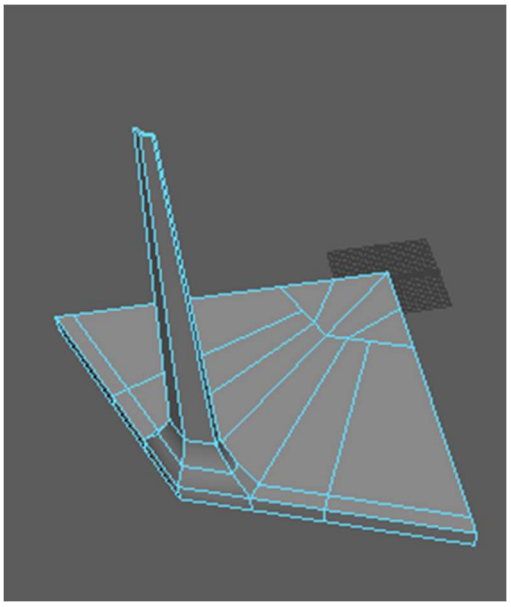


Figure 12: Result of deleting faces of the mesh

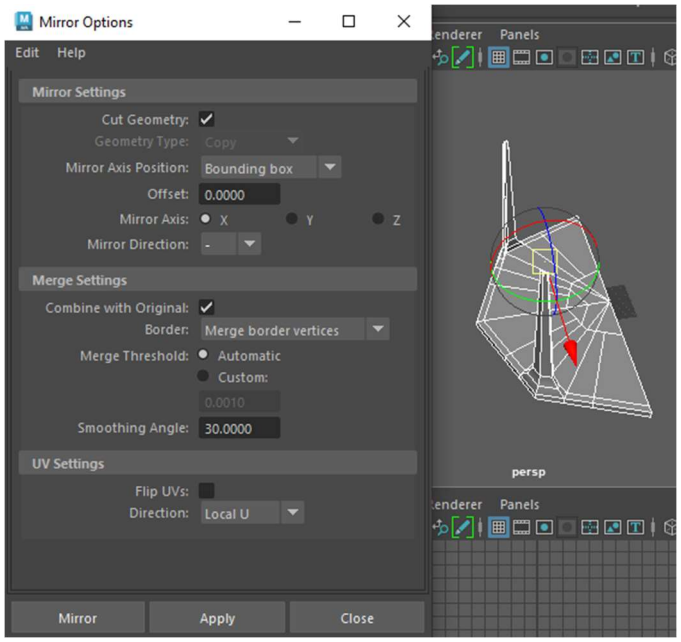


Figure 13: Maya Mirroring tool and the result of mirroring

We can open it from the toolbar “Mesh/MirrorOptions”. As the pivot stands, it is needed to mirror in the X axis, negative direction, and then the Z axis in the same direction.

We can now scale the table to -1 in the Y axis to flip it and this can be the final result for the polygon modeling part.

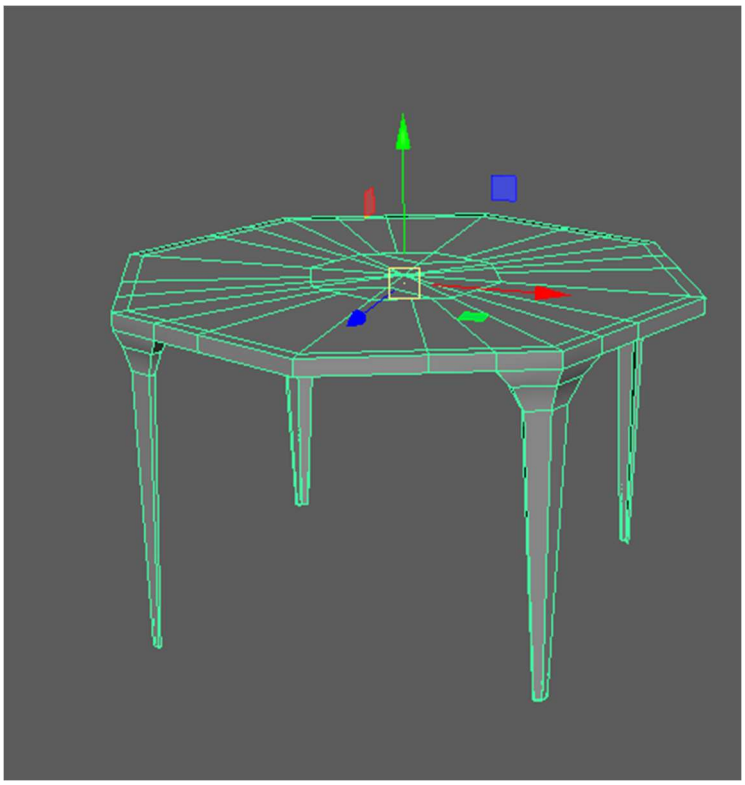


Figure 14: Final result of the modeled table

4.1.3.4. UV Mapping:

Switching to the UV Editing workspace in Maya allows for the creation and manipulation of UVs. By default, Maya generates UV maps automatically, but sometimes the default mapping may not be ideal for achieving seamless and visually appealing textures. To address this issue, manual adjustment of the UV mapping is required.

Ideally the cuts and sews should be on the inside and under the object. In our example, we will have a UV shell for the legs and a UV shell for the top.

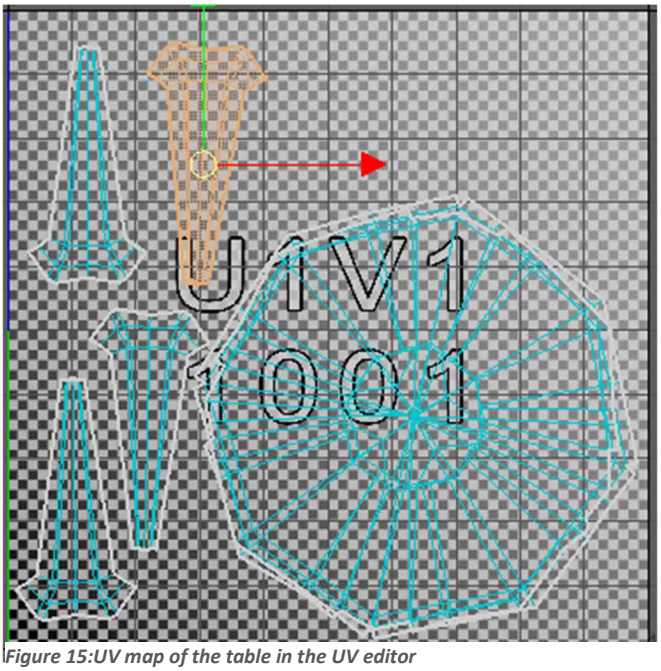


Figure 15:UV map of the table in the UV editor

By cutting the edges and unfolding the UV shells a good UV map can be extracted. The UV maps have fit inside the range (0,1) for both U an V parameters.

4.1.3.5. Texture and Material Application

In Maya, there are various options for adding textures or materials to your 3D models, allowing you to change their appearance and enhance their visual appeal. Here are a few commonly used methods:

- **Standard Surfaces:** Maya offers a range of standard surface materials such as Lambert, Blinn, Phong, etc. These materials provide basic properties like diffuse color, specular reflection, and transparency. You can assign these materials directly to your mesh to change its appearance.
- **Arnold Materials:** Maya also supports the Arnold renderer, which provides advanced material options. Arnold materials offer additional properties and controls to achieve more realistic and visually appealing results. These materials can be used to create effects like subsurface scattering, metallicity, roughness, and more.
- **Texture Mapping:** Maya allows you to apply textures to your models using image files. By utilizing the UV mapping created earlier in the process, you can ensure accurate and precise placement of the texture onto the model's surface.

These are just a few examples of texturing and material application methods within Maya. Depending on your project's requirements and artistic vision, you can explore and experiment with different materials, textures, and shader properties to achieve the desired visual outcome.

For our example, we downloaded a wooden texture from a free texture website and applied it to the table object in Maya.

By creating a new material and selecting the texture file, the UV mapping ensured that the table had a realistic appearance with the wooden texture mapped correctly onto its surface.

Hence, we can observe the result of a simple object being modeled.

Figure 16: Render of the table with Arnold Viewer

Upon observing the final result of our simple object being modeled, it becomes evident that the level of detail, edge sharpness or smoothness, and realism of the texturing can vary based on the specific characteristics and requirements of the project.

4.1.4 File export

Once there's a satisfactory model, the export has to be made in order to upload the model into a Unity Project.

The model is exported in a .FBX file format. Checking the box "Embed materials" will also export material properties such as UV maps and materials.

4.2. Unity [7]

4.2.1. Unity API

The Unity API is known for its intuitive and user-friendly design, making it relatively simple to use. One notable aspect is the similarity in structure and window placement to Maya.

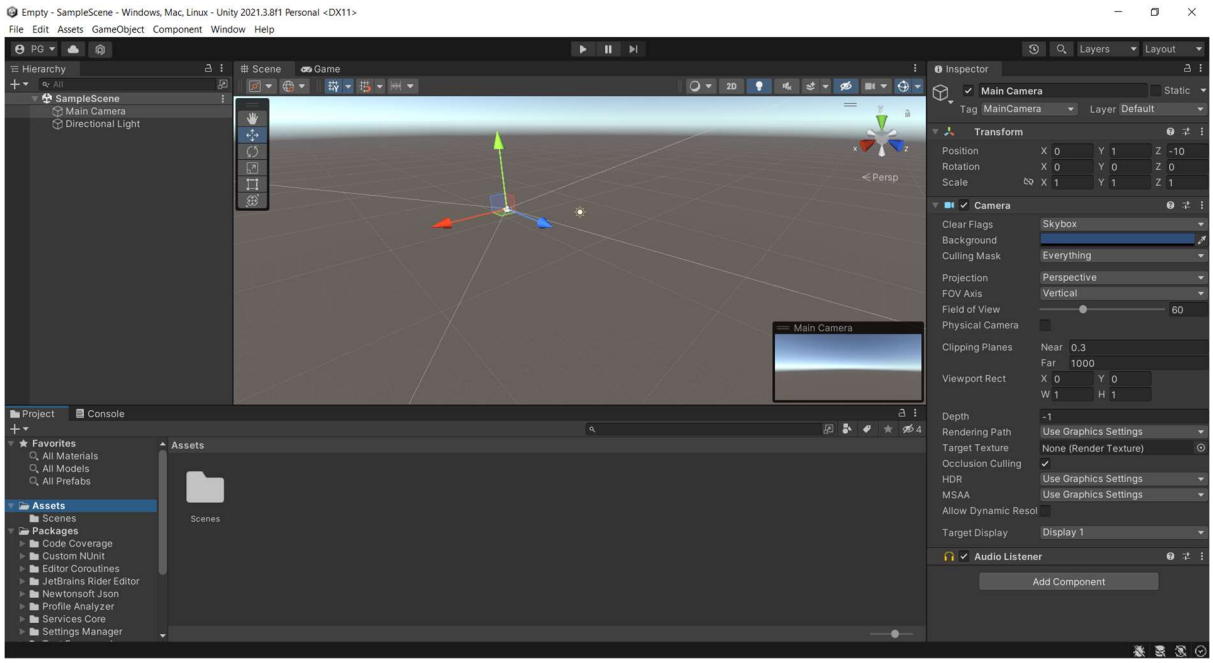


Figure 17: Unity API by default

Toolbar: Located at the top, it provides project management tools and quick access to essential utilities for working with Assets, GameObjects, and Components.

Project and Console Windows: Positioned at the bottom, the Project window serves as a file explorer, allowing the management of project directories. The Packages folder contains necessary packages for the project, while the Assets folder stores project-specific information such as scripts, scenes, and materials. The Console window provides information on compilation and debugging during gameplay.

Hierarchy Tab: Found on the left-hand side, it lists the GameObjects present in the scene, like Maya's Outliner. It helps organize and navigate the hierarchy of objects in the scene.

Inspector Tab: Located on the right-hand side, it displays the components and their properties for a selected GameObject. This tab allows the modification and customization of the behavior and appearance of objects in the scene.

Scene Viewer: Positioned in the middle, it provides a perspective view of the scene, allowing to design and arrange objects within the environment visually.

Game Tab: Also situated in the middle, it shows the view from the perspective of the active camera in the scene. This tab allows you to preview the scene as it would look during gameplay.

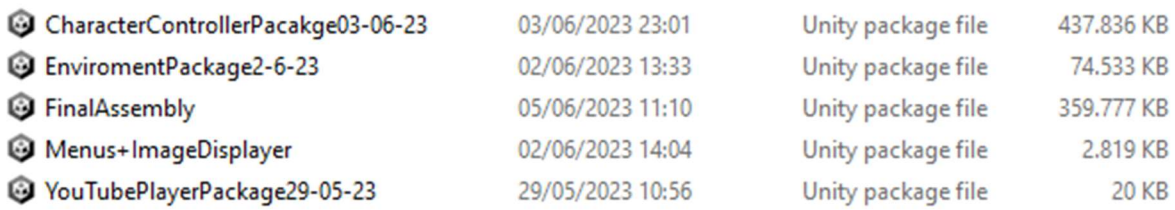
4.2.2. Assets and Packages

In a project, an asset serves as a representation of any item that can be utilized. Assets can originate from external files created outside of Unity, such as 3D models, audio files, images, or other file types supported by Unity. An example of it from this project would be the .FBX files exported from Maya and imported into Unity.

Additionally, Unity provides the capability to create certain asset types within the Unity environment itself. These include Animator Controllers, Audio Mixers, and Render Textures.

On the other hand, packages are compilations of files and data extracted from Unity projects or specific project elements, compressed and stored as a single file. Packages offer a convenient method for sharing and reusing Unity projects and collections of assets.

Packages function similarly to compressed files like .ZIP or .RAR. Within a Unity project, packages can be created containing selected elements or components of the project. These packages can then be imported into other projects, allowing for the assembly of various packages, as is the case in this particular project.



CharacterControllerPacakge03-06-23	03/06/2023 23:01	Unity package file	437.836 KB
EnviromentPackage2-6-23	02/06/2023 13:33	Unity package file	74.533 KB
FinalAssembly	05/06/2023 11:10	Unity package file	359.777 KB
Menus+ImageDisplayer	02/06/2023 14:04	Unity package file	2.819 KB
YouTubePlayerPackage29-05-23	29/05/2023 10:56	Unity package file	20 KB

Figure 18: The packages created for the realization of this project in the file explorer

4.2.3. GameObjects

The GameObject is the most important concept in the Unity Editor: every object in the game is a GameObject. However, a GameObject can't do anything on its own; it needs to be given properties before it can become anything.

A GameObject is a container; by adding pieces to the GameObject container it can become a character, a light, a tree, etc. Each piece added is called a component. Depending on what kind of object is desired, different combinations of components are introduced to a GameObject.

To exemplify, it will be showcased the creation of an avatar and its corresponding animations. To accomplish this, the GameObject serves as the primary entity, encompassing various components such as:

- Character Controller: Enables movement capabilities.
- Movement Scripts: Governs the behavior of the character controller.
- Animator: Facilitates animation functionalities for the GameObject.

Moreover, the same GameObject can simultaneously contain other GameObjects, referred to as children, to aid in achieving the desired outcome.

Taking the avatar as the example, it may include additional GameObject with its different components such as:

- Mesh GameObject: Represents the physical appearance of the avatar.
- Camera GameObject: Follows the avatar, providing dynamic perspectives.

Figure 19: The avatar of Amy in the Unity editor. The hilighted objects are part of the avatar's GameObject

By combining these elements, a GameObject and its child objects work in harmony to create the desired outcome.

4.2.3.1. Components

A GameObject is characterized by the components it contains.

Every GameObject needs the Transform Component, which defines the GameObject's position, rotation, and scale in the game world and Scene view. GameObjects can contain other components as well such as Rigidbody, Collider or Scripts.

To illustrate the role of components, an example is shown by one default GameObject present in every scene: Main Camera.

The inspector lists the components of the Main Camera GameObject. Camera Component and an Audio Listener are the components needed to provide functionality to this GameObject, each containing one or several properties unique to the component type.

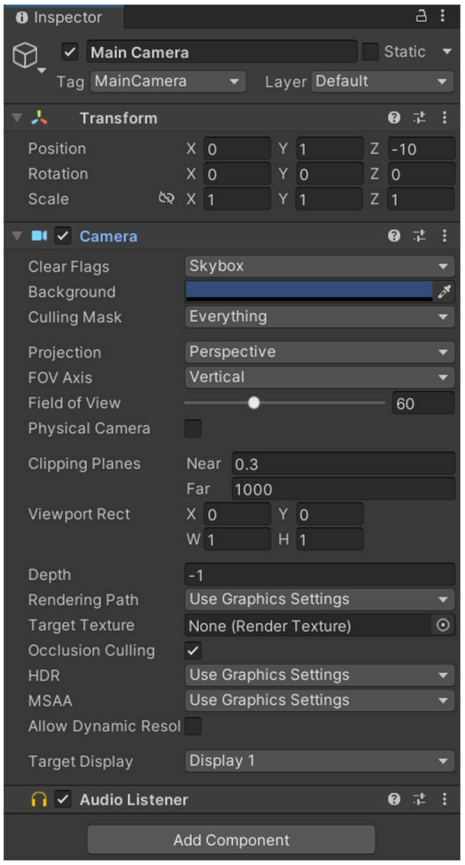


Figure 20: Unity inspector of the MainCamera by default

4.2.3.2. Scripts

In Unity, Scripts are essential for providing custom behaviors and features to the gameplay experience. They allow for unique interactions and functionality that go beyond the built-in Components offered by Unity. Scripts serve as Components and need to be attached to GameObjects in order to define specific behaviors and actions for those objects.

The primary programming language used in Unity is C#, which is widely adopted in the industry for its versatility and performance. Unity also supports its own language called UnityScript, though C# is generally recommended.

The structure of a Script involves defining a public class that matches the filename and derives from the *MonoBehaviour* class. The *MonoBehaviour* class provides essential functionality and integration with the Unity engine. Commonly used methods within Scripts include the *Start()* and *Update()* methods. The *Start()* method is called once when the game starts and is typically used for initialization, while the *Update()* method is called every frame and is useful for continuous actions like movement and event triggering.

4.2.3.3. Prefabs

Unity has a Prefab asset type that allows to store a GameObject object complete with its components and properties. The prefab acts as a template from which one can create new object instances in the scene with the same properties as the prefab. Any edits made to a prefab asset are immediately reflected in all instances produced from it.

In the case of the aforementioned avatar, once it is finalized, it is saved as a prefab. This prefab functionality enables the creation of avatars with various visual appearances while maintaining consistent behavior across all instances.

Figure 21: Hierarchy of AmyCCPrefab Variant GameObject in Unity Hierarchy tab

4.3. Web server

4.3.1. Local server

The project requires a server connection to fully provide with the stated requirements of having a user login and web server communication.

Due to a lack of a free public server the option chosen was to run a simulation locally.

In order to do so, the application MAMP [8] is used to provide a local server.

Figure 22: Logo of MAMP

4.3.2. Database server

The Server provided by MAMP utilizes phpMyAdmin, a popular open-source web-based application written in PHP. It serves as a powerful tool for managing and administering databases.

In order to store, access, and manipulate project data, a database is required. Within the local server environment (localhost), a database can be created using phpMyAdmin.

Within a database, information is organized and stored in tables. Tables are structured as a collection of rows and columns, allowing for efficient organization and retrieval of data. SQL (Structured Query Language) queries are used to manage the tables, enabling operations such as creating, modifying, and retrieving data.

By default, when using phpMyAdmin, after the creation of a database, the root user is created with all privileges. The root user is a superuser with full access and control over the database server. It is commonly used to manage and administer the database, including creating and modifying tables, executing queries, and managing user permissions.

When developing server-side scripts that interact with the database, the root user can be used to establish a connection and perform necessary operations such as inserting, updating, or retrieving data.

4.4. Communication between Unity and web server

To establish a connection and enable real-time interaction between Unity and the server's database, the built-in UnityWebRequest class is utilized. This class provides a convenient way to manage the flow of HTTP communication between Unity and web servers.

4.4.1. UnityWebRequest

UnityWebRequest is a class in the Unity game engine that provides a way to communicate with web servers and retrieve data from or send data to remote resources. It is part of Unity's networking functionality and is commonly used for tasks such as accessing web APIs, downloading files, or interacting with online databases.

UnityWebRequest handles HTTP requests to web servers using various methods such as GET, POST, PUT or DELETE. It supports different types of data formats, including JSON, XML, and binary data.

In addition to basic web requests, UnityWebRequest supports features like downloading files and streaming data, which can be useful for tasks such as downloading assets, images, or audio files from a server.

DownloadHandler and UploadHandler are helper objects attached to a UnityWebRequest. They respectively manage the response and transmission of data during HTTP connections.

5. THE ENVIRONMENT: 3D MODELING

5.1. Design

To create the metaverse, a thoughtful process was undertaken to design the environment and subsequently model its various components. Extensive brainstorming and research were conducted to envision a welcoming and user-friendly environment. The final design concept took shape as a single-story building, comprising a spacious main hall, a therapy room, and a meditation room.

The main hall serves as a central area where users spawn into the metaverse. It was deliberately designed to be open and uncluttered, devoid of excessive elements that may induce a sense of overwhelm. Instead, the focus was on creating a sense of spaciousness and freedom, achieved through the inclusion of tables, chairs, and a small garden at its center. The aim was to evoke a naturalistic ambiance and a feeling of tranquility, with a deliberate avoidance of sharp corners and complex geometries.

The therapy room was conceptualized as a space where an AI-powered avatar, representing a medical professional, would be present to facilitate therapy sessions (imported from the college's project). It is here where users can engage in guided therapy and receive support and assistance.

Conversely, the meditation room was designed as a serene and minimalistic space. Its purpose is to provide a conducive environment for guided meditation sessions. The room features decor elements intended to inspire a sense of tranquility and facilitate a meditative experience.

5.2. 3D Models

To ensure efficient modeling of the various elements in the metaverse, each room and object was created as separate scenes in Maya.

To streamline the process and minimize the number of imported objects/meshes in Unity, the assembly of the metaverse will be done both in Maya and Unity. Each scene will be exported as .FBX files and imported into the Unity project as an asset. This allows the creation of prefab objects, reducing memory requirements and enhancing efficiency.

5.1.2. Main Hall

Structure: Set the room to be 20x15m. Create doors and windows.

Garden: The tree is created with the "Generate / Get brush…" option in the toolbar, then selecting the tree folder and a tree element. Then convert the curves to polygons for .FBX export.

Figure 23: Perspective view of the main hall's structure in Maya

Table and chair:

The modeling process followed a similar approach as described in the example in section 4.1.3.

Specific measurements were applied to ensure realism in the virtual environment. The table was set to a height of 75 cm, while the chair had an absolute height of 125 cm, with a sitting height of 50 cm, simulating real-life proportions.

Figure 24: A table and a chair after texturing in Maya

5.2.2. Therapy room

Structure: The dimensions of the therapy room are 7.5x5m.

Furniture: To create a sense of freedom and provide multiple seating options in the therapy room, various furniture elements were modeled to fit into the space.

One of the furniture arrangements includes a sofa positioned facing the window, accompanied by a small irregular table. This setup allows for a comfortable and relaxed seating arrangement while offering a view towards the outside.

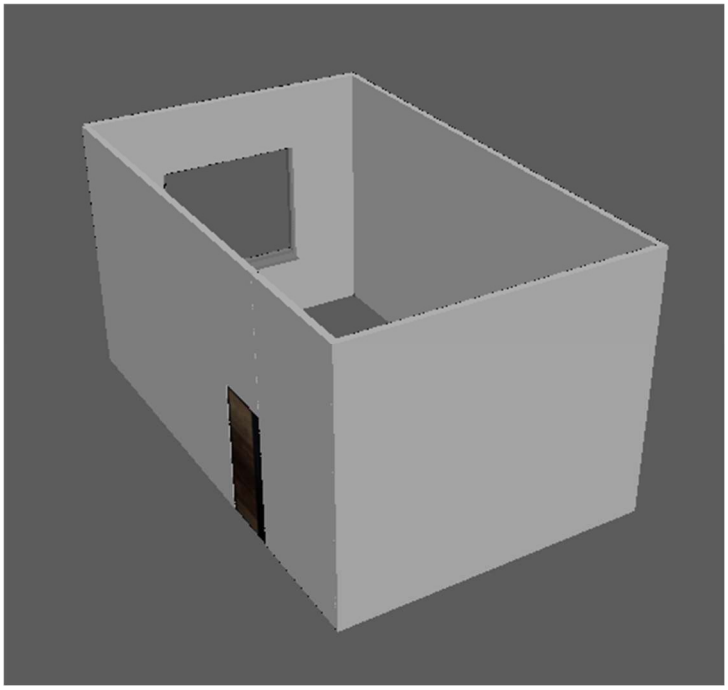


Figure 25: Perspective view of the therapy room's structure

Another seating arrangement consists of two round chairs facing each other, with an eclipsed shaped table placed in between them. This configuration encourages face-to-face interaction and promotes a conducive environment for therapy sessions.

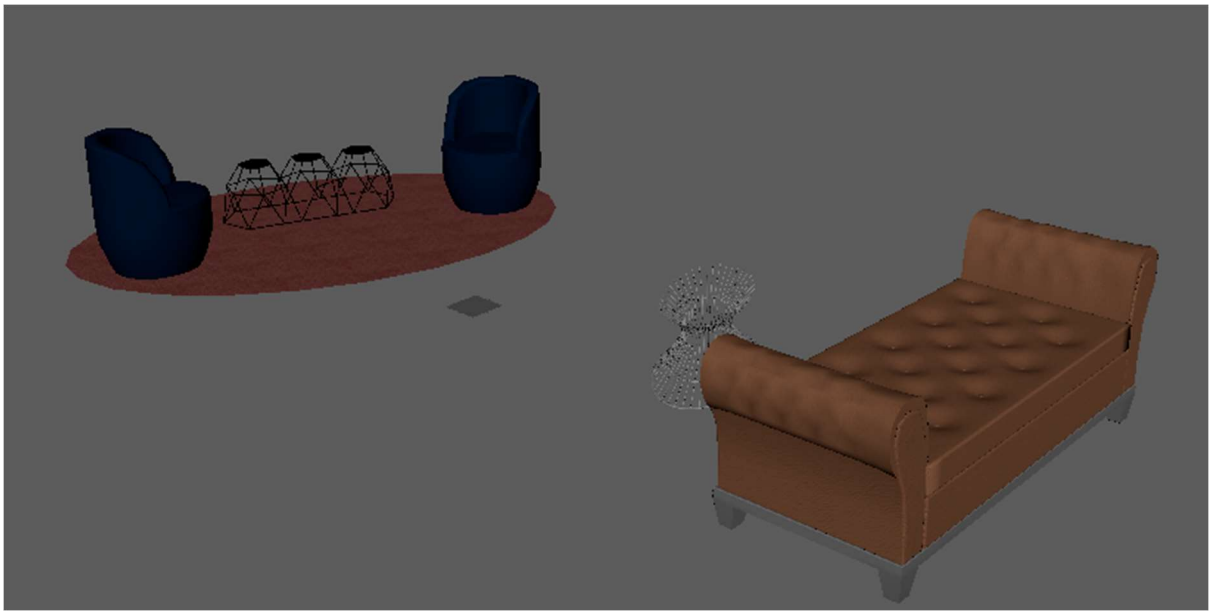


Figure 26: The therapy room's furniture in Maya editor view

5.2.3. Meditation room

Structure: Contrary to the initial design concept, the final shape of the meditation room is a rectangular arrangement with a rounded end to accommodate a circular tatami.

The room has a width of 7.5 meters and a total length of 8.75 meters for the rectangular portion, along with an additional 3.7 meters for the radius of the circular wall.

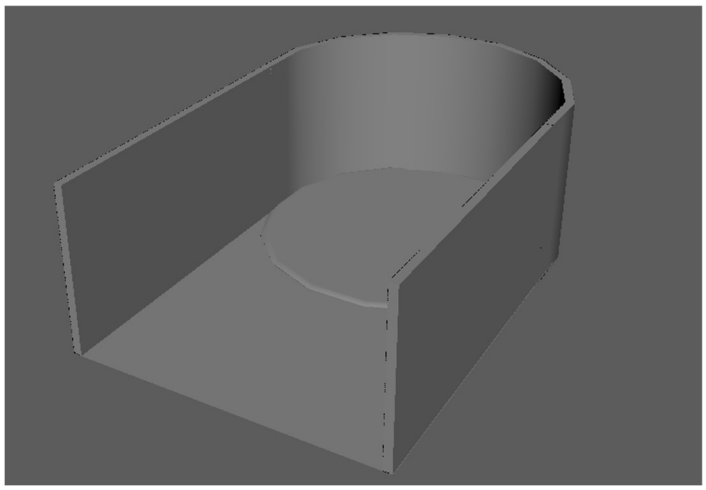


Figure 27: Perspective view of the meditation room's structure

This layout provides a spacious and serene environment for meditation practices. The rectangular shape offers ample space for individuals to sit or lie down comfortably, while the rounded end adds a touch of aesthetic appeal and a focal point for the meditation area. The circular tatami can be placed in the rounded section, serving as a designated space for meditation and promoting a sense of calm and focus.

Furniture: To create an authentic Japanese ambiance in the meditation room, various elements were incorporated into the design. Japanese-style sliding doors, lights, and window frames were modeled to capture the traditional architectural aesthetics.

To bring a touch of nature indoors, various plants were included in the scene. Placing the plants throughout the room, a sense of tranquility and harmony with nature is fostered.

A yoga statue was imported to serve as a focal point within the meditation space.

In this specific case, the arrangement was done in Maya and the whole room was exported as one file.

Figure 28: Perspective view of the meditation room's interior

6. METAVERSE DEVELOPMENT BY PACKAGES

The development of the metaverse is endured in Unity. To maintain a well-organized development process in Unity and facilitate the assembly of the final metaverse project, a systematic approach was adopted, dividing the work into the separate components: environment, avatars, menus, and functionalities.

By breaking down the project into these distinct parts, it becomes easier to focus on individual aspects and ensure clarity and organization throughout the development process. Each component can be treated as a separate package, with its own set of assets, scripts, and scenes. Each package can be developed and refined independently, allowing for focused work in its elements.

Once the individual packages are developed and refined, they can be combined and assembled to create the final metaverse project. This modular approach allows for flexibility and scalability, enabling future updates and modifications to be easily incorporated into the project.

6.1. Environment assembly

To assemble the environment in Unity using the models created in Maya, the first step is to export the objects and scenes as .FBX files from Maya. During the export process, it is important to enable the "Embed media" option to ensure that textures and materials are included within the FBX file itself.

If any of the textures used in the models are external files, they need to be present within the Unity project.

The exported .FBX files are imported into Unity. Unity will process the .FBX files and import the associated meshes, textures and materials.

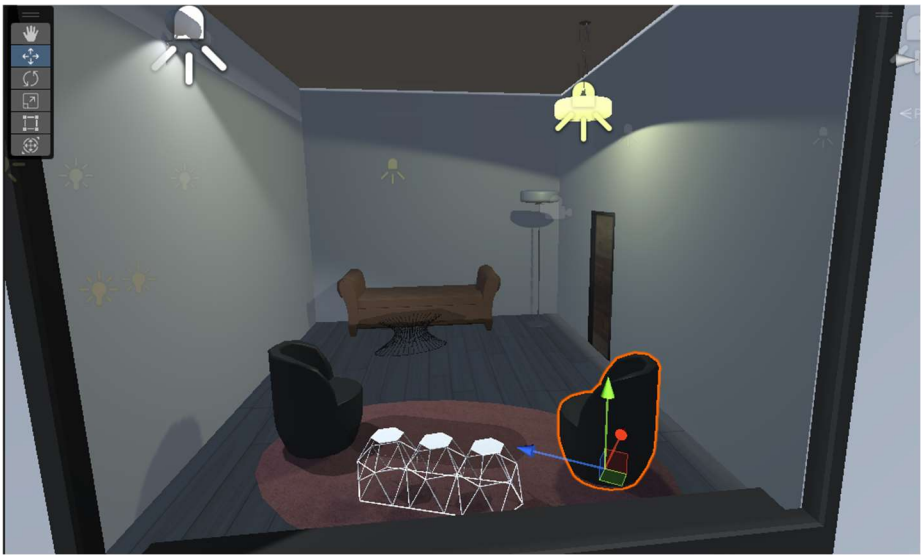


Figure 29: Perspective view of the therapy room in Unity

Once the models are imported, they can be arranged and positioned within the Unity scene to recreate the desired metaverse environment. This involves placing the objects in their appropriate locations, adjusting their scales and rotations if necessary, and setting up the overall composition of the scene.

6.2.1. Double sided textures

Upon importing all the elements into Unity, it became apparent that certain textures and materials, specifically those associated with the plants' leaves (generated using the brush tool in Maya), were not being displayed accurately.

This issue arose because Unity, by default, only renders the side of each plane that possesses the normal vector. In most scenarios, this default behavior is optimal as it reduces the processing required. For instance, for our environment's walls or the floor, there is no need to render the inner faces since they remain unseen.

However, the leaves of the plants are visible from both the inside and outside faces, necessitating the rendering of both sides. To address this problem, a shader capable of displaying both faces needed to be utilized.

Since Unity does not provide such a shader by default, a solution was sought from the Unity Asset Store. The "Free Double Sided Shaders" asset, available in the asset store [9], was downloaded for this purpose.

After installation, a new material was created, and instead of using the standard shader, the shader from the newly imported asset was selected for the material.

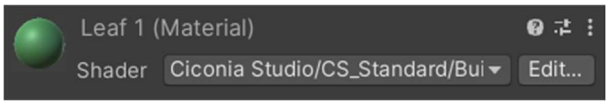


Figure 30: Material Inspector

Figure 31: Top view of a plant with default material shader

Figure 33: Bottom view of a plant with default material shader

Figure 32: View of a plant with CiclonaSturido material shader

6.2.2. Lighting

The environment is primarily illuminated by the default directional light, which simulates outdoor lighting. However, due to the presence of a roof with double-sided textures, it is necessary to ensure proper indoor lighting as well.

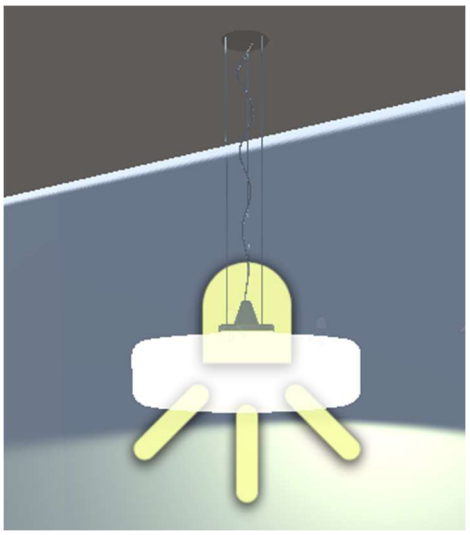


Figure 34: Hanging light in the therapy room

To achieve realistic lighting effects, a collection of lamps was acquired for free from a 3D model bank [10]. These lamps were imported into the project as .FBX files.

For indoor lighting, the "Spotlight" is utilized. By adjusting the parameters of the light component, such as the spot angle and color, the light is configured to achieve the desired effect. Soft shadows are enabled, and the Mixed mode is set to compute shadows only when the player is nearby, thus optimizing the project's performance.

To create a more authentic appearance of light emanating from the lamp, the "emission" box in the material assigned to the lamp is checked. The color of the emission can be adjusted to match the actual light emitted by the light object.

To ensure consistent lighting throughout the environment, lights can be saved as prefab objects and placed strategically to provide adequate illumination in every corner.

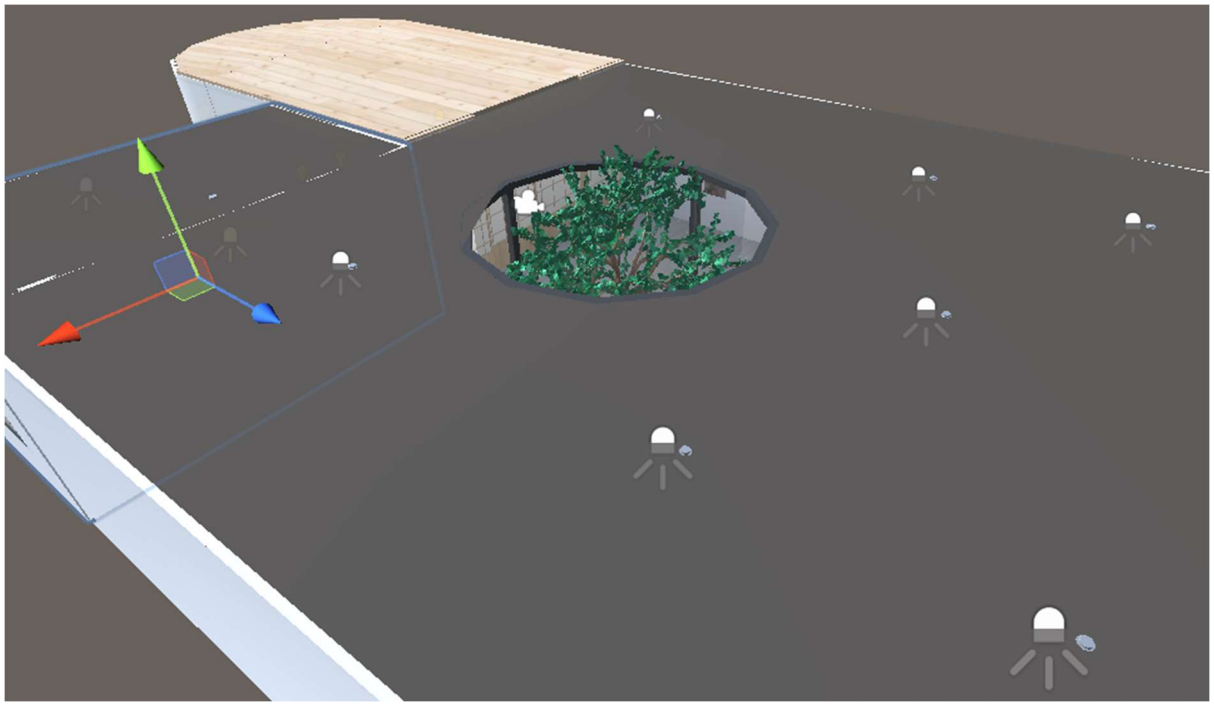


Figure 35: Top view of the enviroment with light icons

6.3.3. Decoration

In addition to the foundational furniture, custom-designed pictures were created to adorn and enhance the aesthetics of both the main hall and the meditation room, adding a final touch to the overall environment. These pictures were modeled to align with the desired decorative theme and style of each respective space. They serve as artistic elements that contribute to the ambiance and visual appeal of the rooms, elevating their overall atmosphere.

Figure 36: Inside view of the enviroment

6.3.4. Mesh Colliders

To ensure that objects in the scene cannot be passed through by the player, it is necessary to enable collision detection. This can be achieved by adding the "Mesh Collider" component to each GameObject that has a mesh representation.

By adding the "Mesh Collider" component, the physical boundaries of the object's mesh are defined, allowing for collision detection with other objects in the environment. This means that when the player interacts with the scene, they will no longer be able to pass through walls, chairs, doors, and any other objects that have the "Mesh Collider" component attached.

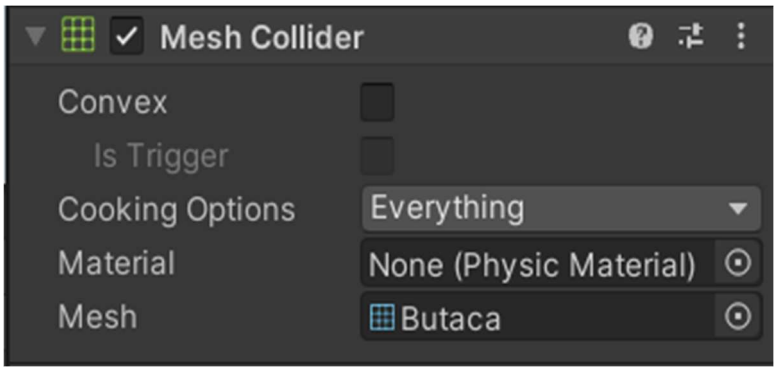


Figure 37: Mesh collider compenent in the Inspector tab

By implementing this collision detection mechanism, the player's movements and interactions within the virtual environment will be constrained by the physical presence of objects, providing a more realistic and immersive experience.

Figure 38: Interior view of the meditation room with the tatami inspector

6.2. Avatar integration

In the metaverse, users navigate and interact with the virtual environment through their avatars. These avatars are designed to resemble humans and are animated to exhibit realistic behaviors, enhancing the overall in-game experience.

This section will explain the different elements that are present within the avatar and its behavior.

- Character Movement
- Camera controller
- Avatar
- Animation

6.2.1. Character controller and movement

The Character Controller is a crucial component responsible for managing the movement and navigation of the avatar within the virtual environment. It acts as the "HitBox" or collision capsule for the avatar, allowing it to interact with other elements in the scene.

To enable player movement, a script is required to provide the desired movement behavior.

The script responsible for controlling the character is the *MovementStateManager.cs* (the same script also includes the management of the animations which will be explained in section 6.2.4).

The following conceptual map illustrates the algorithm endured by the script which controls the movement of the player.

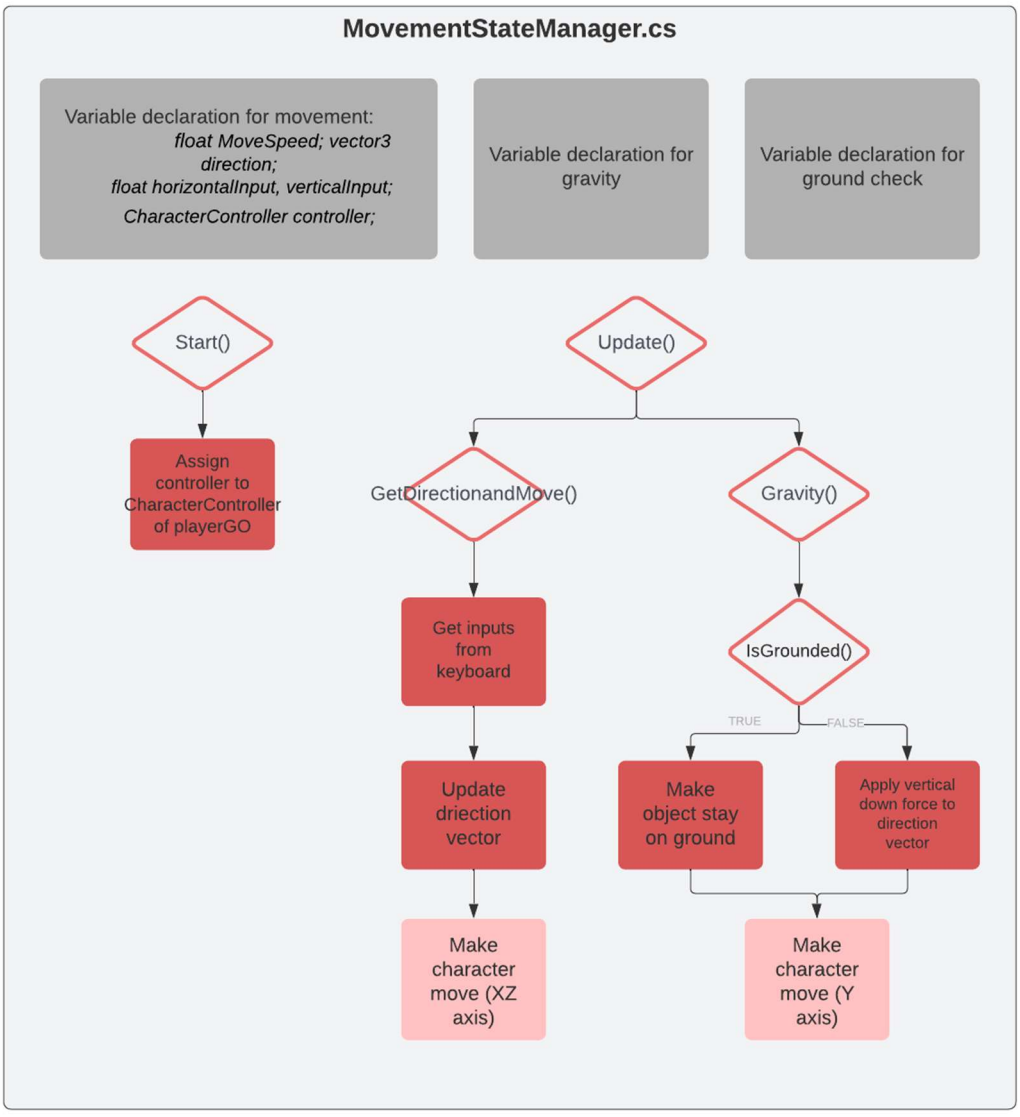


Figure 39: Algorithm of MovementStateManager script for movement

6.2.2. Camera controller

The point of view and camera control play a crucial role in providing a satisfying gaming experience. To achieve smooth and dynamic camera movements that follow the player, the Cinemachine package can be integrated into the Unity project.

Cinemachine is a powerful camera system provided by Unity that allows for camera control.

6.2.2.1. Camera following player.

To make the camera follow the player as it moves around the environment the following steps are undergone:

- Add *Cinemachine brain* component to the main camera.
- Create a new empty GameObject child of the Player.
- Create a new *CineMachine VirtualCamera* and assign the last GameObject. to the "Follow" and "LookAt" parameters. Select "body/3rd person follow" and "aim/Do nothing".
- Add the extension *CinemachineCollider* to the virtual camera. This will make the camera detect collisions and keep the view of the player upon hitting walls or other surfaces.

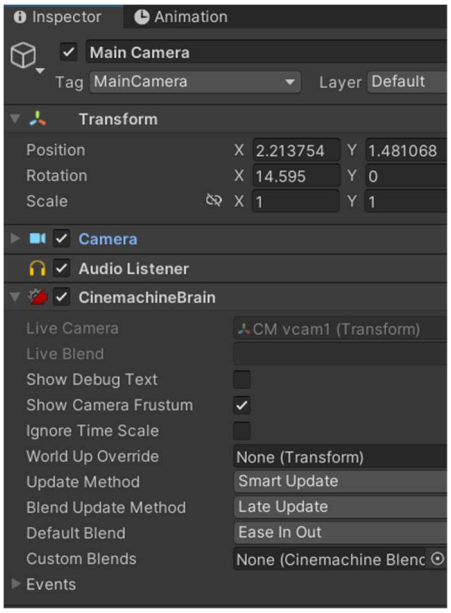


Figure 41: Inspector of the MainCamera with the CinemachineBrain component

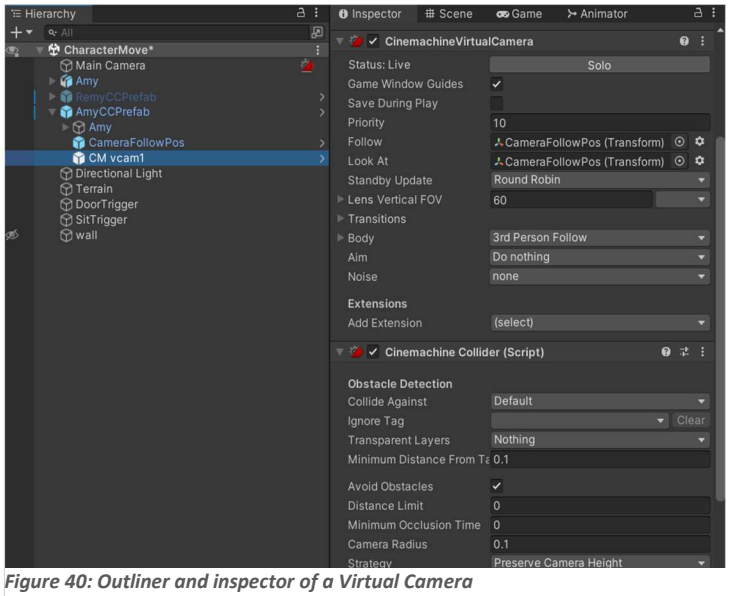


Figure 40: Outliner and inspector of a Virtual Camera

6.2.2.2. Player rotation and perspective

In many 3rd person games (such as Call of Duty or GTA), the camera is controlled independently from the movement of the player. In those cases, the most frequent is that the character rotates to where the camera is pointing.

The script *aimStateManager.cs* provides this functionality. The script is added to the player GameObject (which contains the characterController and movement script components).

The following conceptual map explains the algorithm used by *aimStateManager.cs*

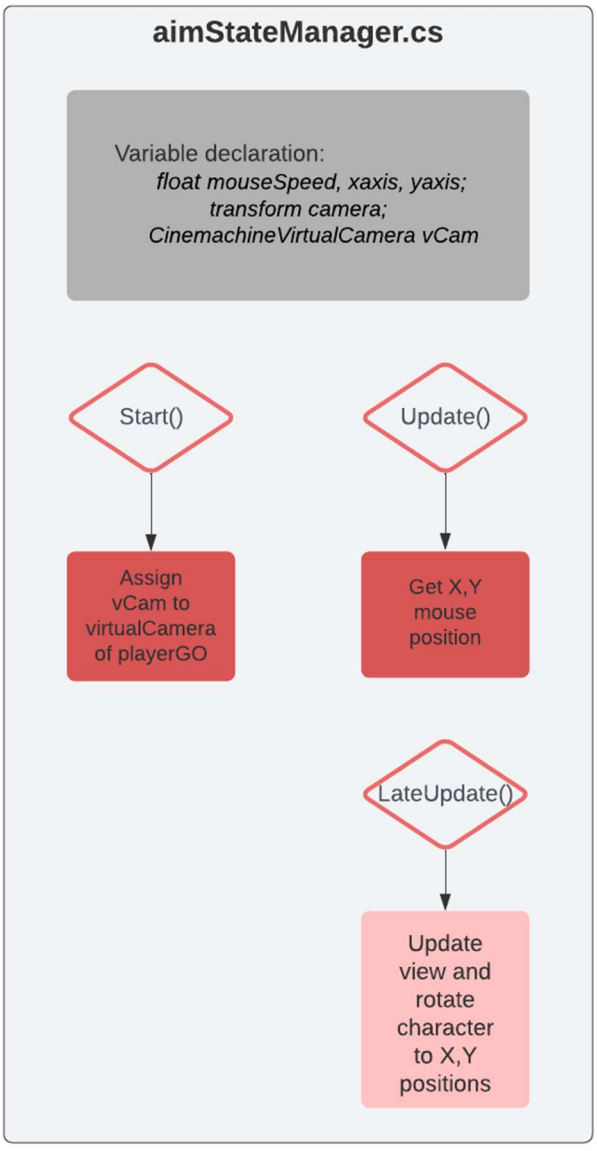


Figure 42: Algorithm of aimStateManager script

6.2.3. Avatar

The avatar refers to the avatar body which represents the physical form of the character within the virtual world. It includes the mesh, textures, and materials that define the avatar's appearance.

The avatars used for this project are downloaded from Mixamo [11].

To utilize the avatars downloaded from the web and prepare them for further use, the following steps are endured:

Click on the avatar asset imported, in the inspector tab configure the avatar as following:

- ➢ Materials:
  - Extract textures
  - Extract materials
- ➢ Rig:
  - Set “Animation type” to “Humanoid”
  - “Avatar definition” to “Create from this model”

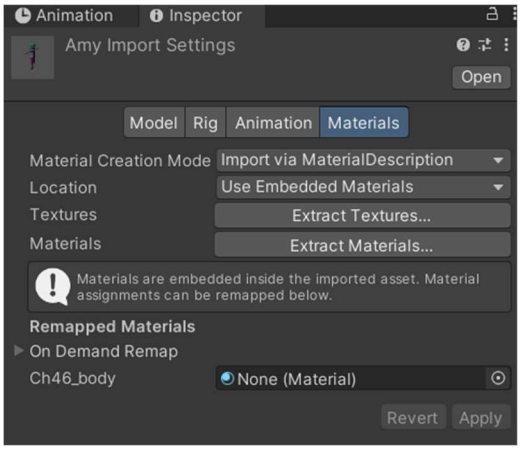


Figure 43: Inspector of imported .fbx avatar file

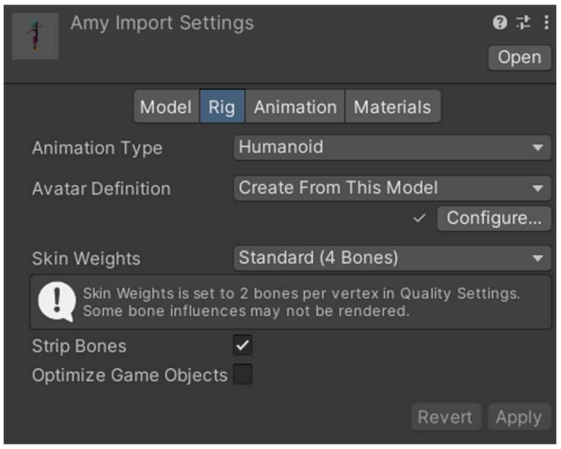


Figure 44: Inspector of the imported .fbx avatar file

Figure 45: Amy avatar asset placed in a terrain

6.2.4. Avatar animation

Mixamo provides a vast collection of animations (pre-made motions like running, walking, etc) that can be applied to all the avatars downloaded from their website.

This feature is highly beneficial as it allows for the utilization of a single animator controller that can be applied to all the avatars.

6.2.4.1. Download animations

The animations needed to download for this project are, regarding each action:

- Idle: Idle state
- Walking: Walk forward, walk left, walk backwards, walk backwards side
- Running Slow run, run left, run backwards, Run backwards left
- Sitting: Stand to sit, sitting idle, sit to stand
- Opening door: opening

The animations, when imported into the project, have to be configured as following:

- Rig:
  - Set “Animation type” to “Humanoid”
  - “Avatar definition” to “Create from other avatar”

6.2.4.2. Animator controller

With all the animations in the project, an animator controller is created. In Unity, the Animator Controller is a tool used for controlling and managing animations for characters or objects. It is a visual state machine used to define and organize different animation states, transitions between states, and the parameters that drive those transitions.

Each state can incorporate either a single motion, such as an opening door animation, or a blend tree that combines various motions, such as walking, which has different animations based on the direction.

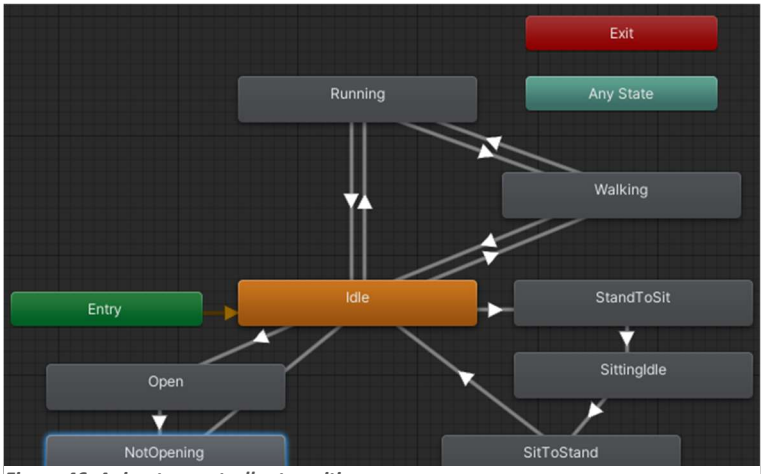


Figure 46: Animator controller transition map

Transitions between these states are depicted by white arrows, indicating how the animation flow progresses. Each transition is equipped with conditions that determine when a specific animation state should commence. These conditions are variable-based and are controlled through scripting.

Parameters:

- *hzInput:* type float, defines left/right direction
- *vInput:* type float, defines up/down direction
- *Walking:* type bool. Upon changing, triggers transition from (false→ true)/to (true→false) to the *walking* state.
- *Running*: type bool. Upon changing, triggers transition from (false→ true)/to (true→false) to the *running* state.
- *Open*: type trigger. Upon triggered, transitions to *open* state.
- *Sit*: type trigger. Upon triggered, transitions to *StandToSit* state.
- *Sitting:* Type bool. If true, triggers transition from *StandToSit* to *SittingIdle* and, if false, from *SittingIdle* to *SitToStand.*

Walking blend tree:

The configuration of the *walking* blend tree to trigger a walking animation depending on the *hzInput* and *vInput.*

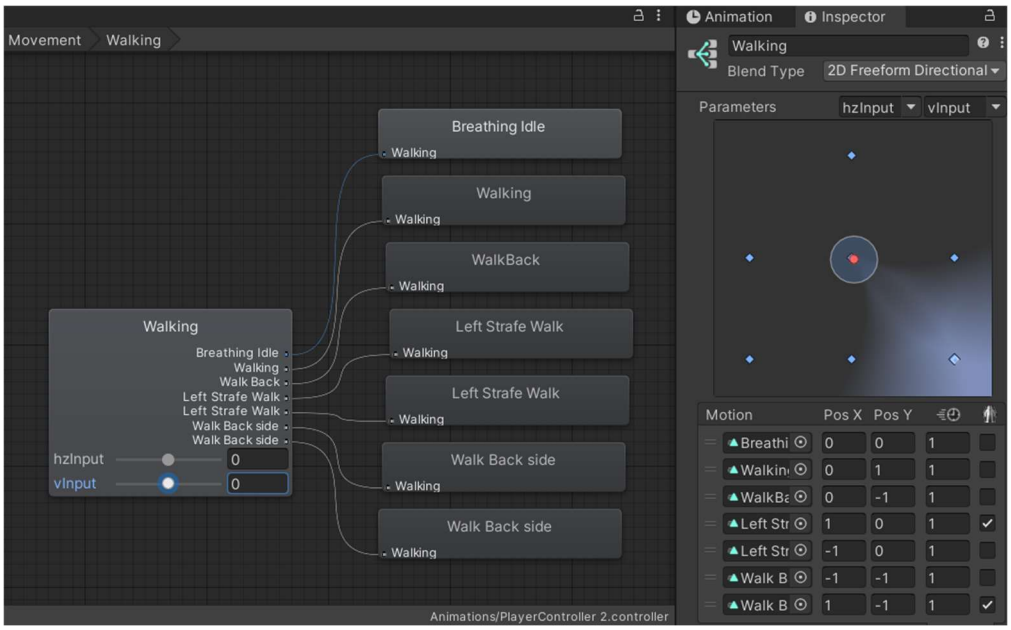


Figure 47: Walking blend tree of animator controller

Running blend tree:

The configuration of the *running* blend tree to trigger a running animation depending on the *hzInput* and *vInput.*

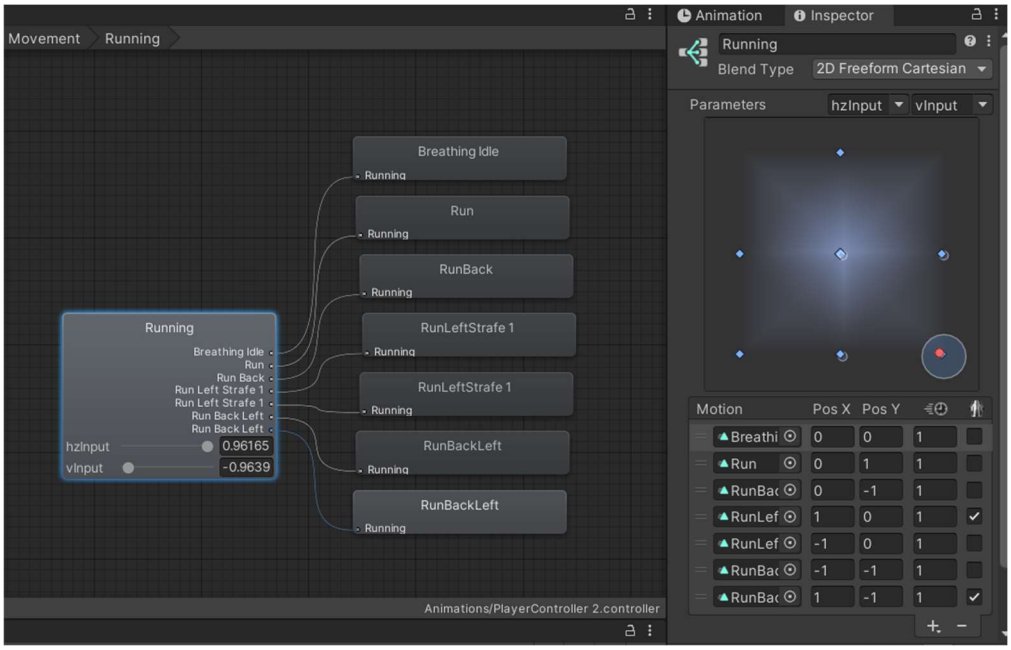


Figure 48: Running blend tree of animator controller

6.2.4.3. Animation scripts

Figure 49: Amy walking forward and sideways

As mentioned previously, the animations in this project are triggered by manipulating the parameters within the animation controller. These parameters are updated through scripts.

In addition to serving as the character controller movement script, the *MovementStateManager.cs* script also facilitates the functionality of transitioning between different scripts assigned to each state of the animation controller.

Each state script is derived from the abstract class *MovementBaseState*, which defines two methods: *EnterState(MovementStateManager movement)*, which is called when transitioning to a new state, and *UpdateState(MovementStateManager movement)*, executed every frame.

To provide a clearer understanding of how the states are controlled, please refer to the accompanying visual aid.

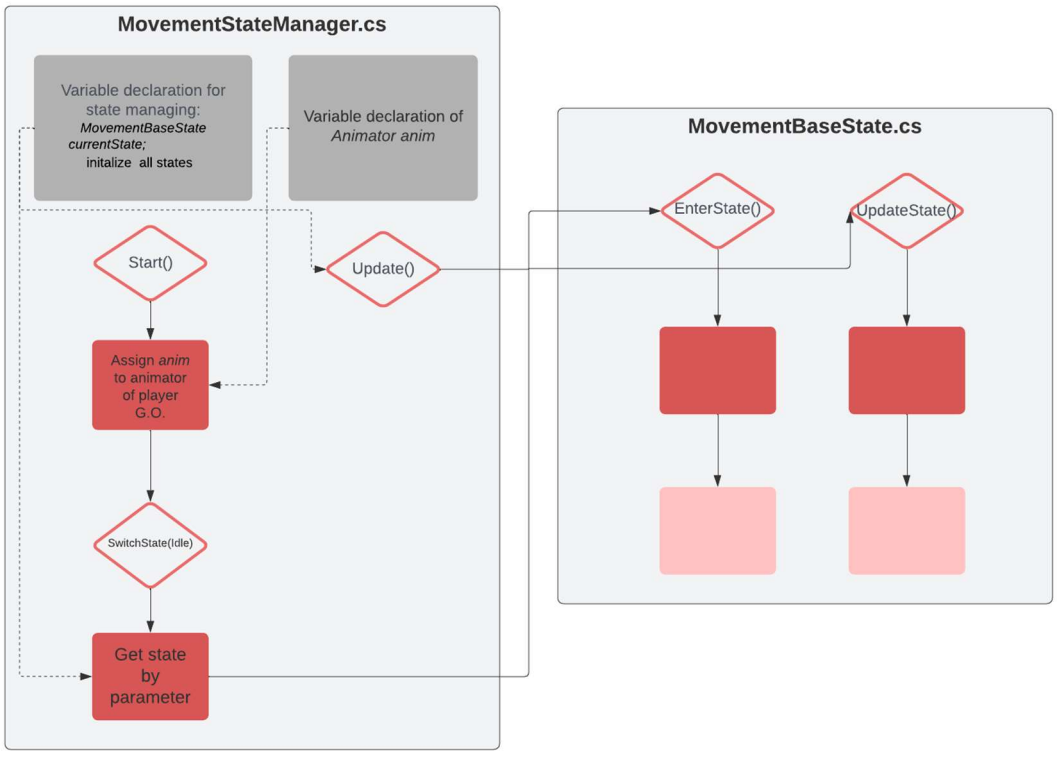


Figure 50: : Algorithm of MovementStateManager script for animations

As mentioned, each animation state in the project has its own script, which declares a new class derived from the *MovementBaseState* class. Consequently, each state script possesses distinct *EnterState()* and *UpdateState()* functions tailored to its specific behavior and requirements.

The figure 51 illustrates the behavior of each state and draws the connection between them.

Please note that to initiate the opening and sitting animations, the character must be within a trigger region with the "Door" or "Sit" tag, respectively. Merely pressing the designated keys (E or C) will not trigger the corresponding animations unless the character is properly positioned within the respective trigger region.

This behavior is implemented within the *MovementStateManager.cs* script, as well as the *Opening.cs* and *SitState.cs* scripts. These scripts collectively manage the interaction between the character and the triggers, ensuring that the appropriate animations are triggered only when the character is within the designated trigger regions with the corresponding tags.

Additionally, it is important to note that while the character is in the opening and sitting states, movement is blocked. This means that the character cannot move or perform other actions during these specific animation states.

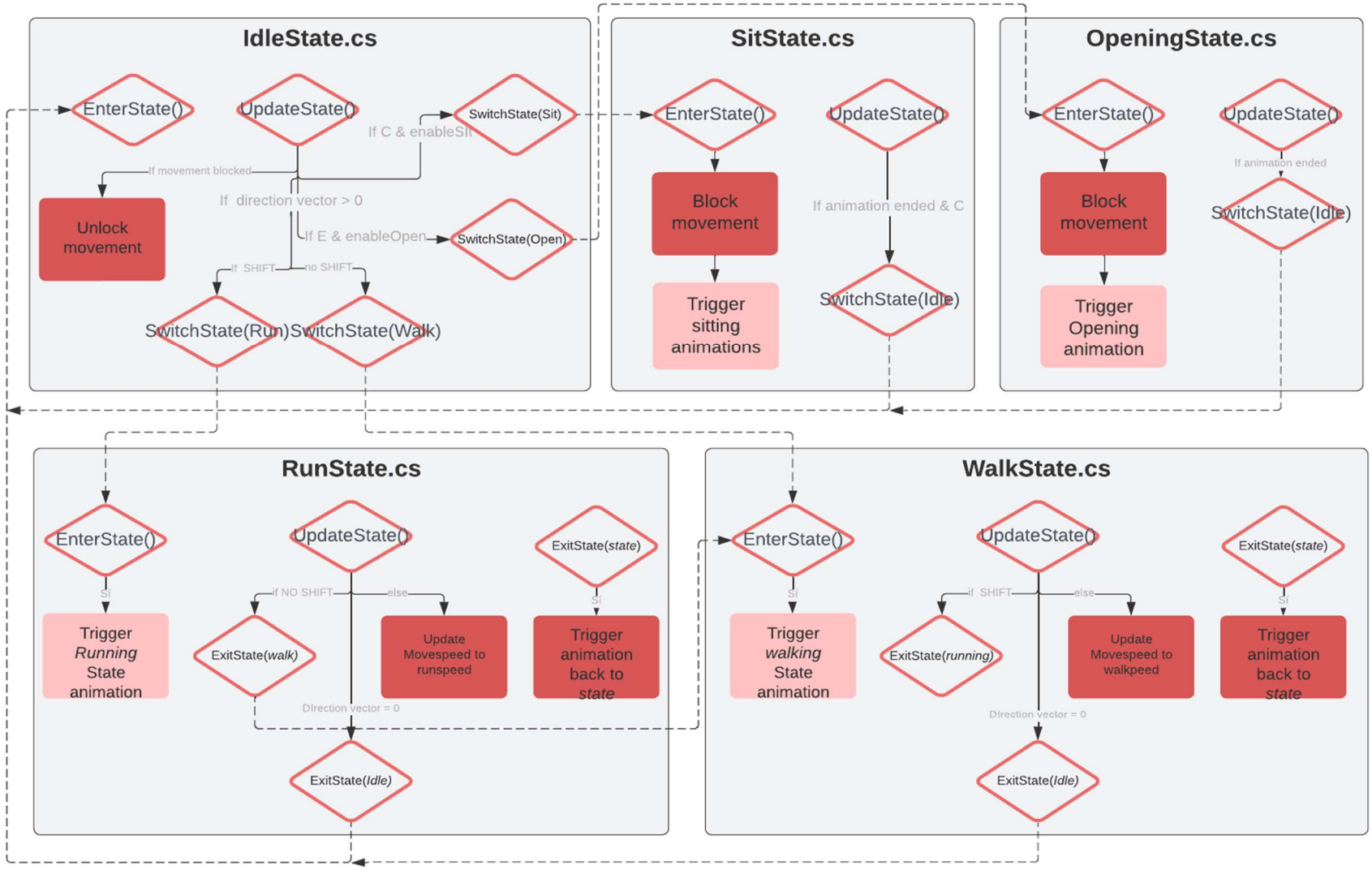


Figure 51: : Algorithm of the scripts of the diferent animation states and their transisons

6.3. Web server

To implement a login menu with user and doctor data, as well as the ability to retrieve information and images from a web server or a database, a database management system such as MySQL can be used in conjunction with Unity

To manage and run the database, it is necessary to set up a web server that can host PHP scripts and provide access to the database.

6.3.1. Local server

Upon installation of the programmary MAMP, a folder named MAMP is created into the local storage. After installation, launching the application will run a local server.

The server runs at the location “localhost” and the root directory for the server is “C:/MAMP/htdocs”. For the convenience of our project, another folder was created inside the server root folder named “sqlconnect”. All server-side scripts and database connections will be located inside this last folder.

While MAMP is active, the URL http://localhost/sqlconnect/ can be accessed with the browser to visualize the server content.

6.3.2. Database

The URL http://localhost/phpMyAdmin/ enters the phpMyAdmin home page. To set the database tables needed, a new database is created under the name *unityaccess*. The tables *patients* and *doctors* are created for login management and the table *images* for image display function.

6.3.2.1. Patients table

Stores the information regarding the patients.

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(10)			No	None		AUTO_INCREMENT	Change Drop More
2	username	varchar(16)	utf8_general_ci		No	None			Change Drop More
3	salt	varchar(50)	utf8_general_ci		No	None			Change Drop More
4	hash	varchar(100)	utf8_general_ci		No	None			Change Drop More
5	pathology	int(10)			No	0			Change Drop More

Figure 52: Patiens table structure

- *id* and *username* are unique and indexes of the table “patients”.
- *salt* and *hash* are the parameters used to store the encrypted password.
- *pathology* stores the information of the pathology the user is classified into.

6.3.2.2. Doctors table

Stores the information regarding the doctors.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	username	varchar(16)	utf8_general_ci		No	None			Change Drop More
3	salt	varchar(50)	utf8_general_ci		No	None			Change Drop More
4	hash	varchar(100)	utf8_general_ci		No	None			Change Drop More

Figure 53: Doctors table structure

The same columns as the *patients* table are required with the exception of the *pathology* column.

6.3.2.3. Image table

Stores the information regarding the file location and name of every image uploaded by any doctor.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	id	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	filename	varchar(255)	utf8_general_ci		No	*None*			Change Drop More
☐	3	pathology	int(10)			No	*None*			Change Drop More
☐	4	directory	varchar(255)	utf8_general_ci		No	*None*			Change Drop More

Figure 54: Image table structure

- *filename:* name of the image
- *directory*: location to the image file
- *pathology:* pathology type which the image provides information about.

6.4. Application menus

6.4.1. Main Menu

When launching the application, the main menu serves as the initial interface that users can interact with. The main menu provides an interface where the user can login or logout and start the game.

At the top, the name “METADATION” of the application is largely displayed and below, the following list of elements is found:

- User information (prompt): Displays in text “User not logged in” or the username if the user has successfully logged in.
- Log In (button): Enabled/disabled if the user has/has not logged in. Launches Log In menu scene (section 6.4.2)
- Start Metaverse (button): Enabled/disabled if the user has/has not logged in. Launches the metaverse scene.
- Log Out (button): Enabled/disabled if the user has not/has logged in. If clicked, user information is reset.
- Doctor Menu (button): Launches Doctor menu scene (section 6.4.4).

The behavior is controlled by the script *mainmenu.cs* which contains the functions that the buttons activate when pressed (*GoToLogin(), GoToMetaverse(), Logout() GoToDoctorLogin()*) as well as managing the visual aspect of the buttons and the prompt text (*Start()*).

The following figure shows the main menu in game mode. Start metaverse and log out buttons are disabled because no user has been logged in.

Figure 55: Mainmenu in game

6.4.2. User Login menu

User LogIn scene manages the connection between the patients database and Unity.

When a user attempts to log in, their entered credentials are sent as a form through a *UnityWebRequest* pointing to the server-side script with the URL : http://localhost/sqlconnect/login.php.

Figure 56: User login menu in game

The server-side script (*login.php*), upon receiving the request, establishes connection with the database and verifies the credentials against the stored data in the *patients* table of *unityaccess* database. If the credentials are valid (*username, hash* and *salt* match), the server sends a response back to Unity to successfully log in the user with the *pathology* type information and automatically transitions back to the main menu.

A user can only try to log in when both input fields have at least 8 characters.

The following figure shows the algorithm used to log in a user.

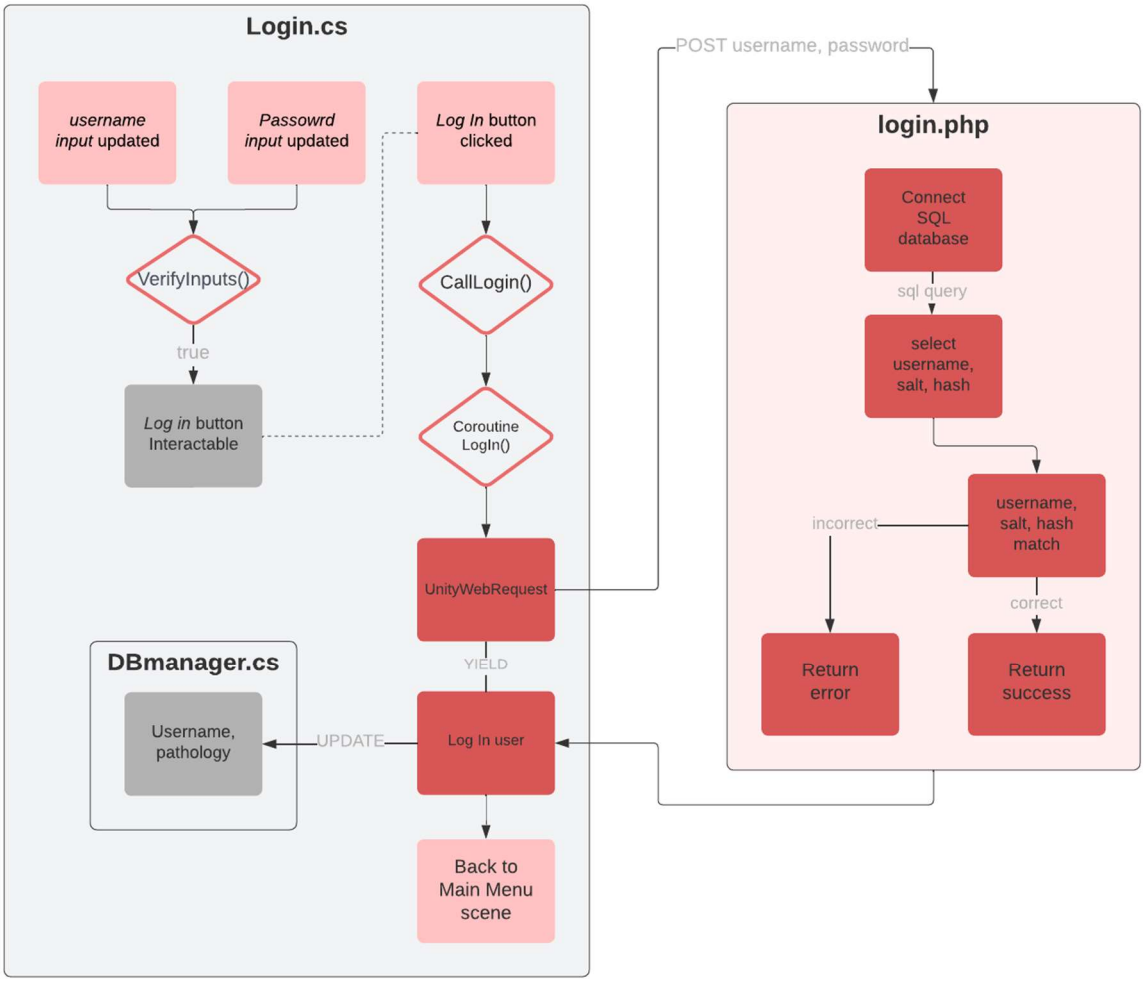


Figure 57: Algorithm of login

6.4.3. Doctor Log In menu

Behaves identically as user log in. The script regulating the behavior is *DoctorLogin.cs* which can create *a UnityWebRequest* pointing to the server-script *doctorlogin.php.*

Upon logging in as a doctor, the application will transition to the doctor menu, providing specific functionalities and options tailored for doctors instead of transitioning back to the main menu.

6.4.4. Doctor Menu

Within the doctor-specific interface, doctors have the ability to register new patients and upload images for the therapy room.

Figure 58: Doctor interface for uploading pictures and registering new patients

6.4.4.1. Patient Registration

Doctors can register new patients by entering their username, password, and pathology type number. This information will be used to create a new patient profile within the database.

If registration is successful (no other patients registered with the same username) the text prompt informs of the event.

The algorithm for registration is similar to that used in login. Again, a connection is endured by *UnityWebRequest* pointing to the register server-script at the URL http://localhost/sqlconnect/register.php. The *register.php* file checks for any matching user with the same username and if there are none, proceeds to encrypt the password and insert a SQL query into the *patients* table.

6.4.4.2. Image Upload

Doctors can upload images from the file explorer. These will be displayed inside the therapy room for users undergoing therapy. These images are specific to each pathology and are intended to provide a visual component to the therapeutic experience.

Pressing the *Import Image* button will open the file explorer. Selecting an image file will close the window and a previsualization will be observed. After introducing the pathology image type, the button *upload image* will be selectable.

If the image has been successfully uploaded, the text prompt will say so.

The following figure shows the algorithm used to upload an image into the web server and update the table *images* from the database [12][13].

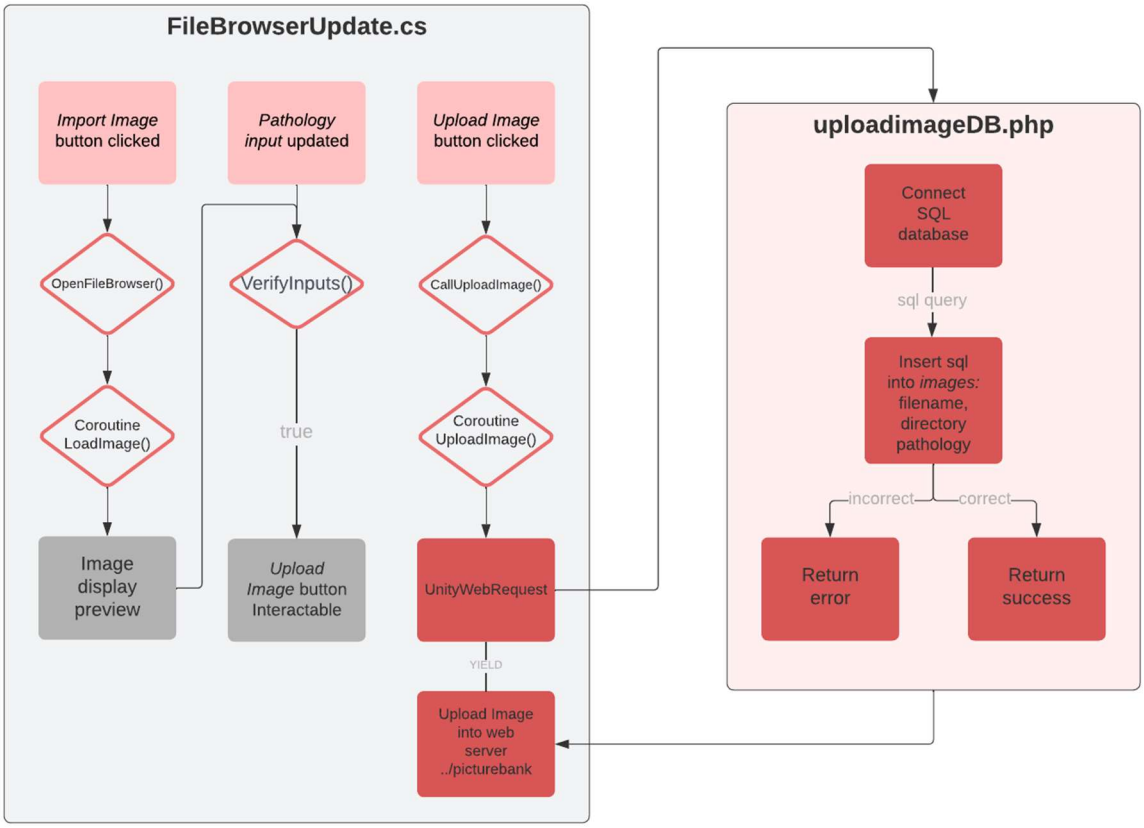


Figure 59: Algorithm for uploading images to ther server

6.4.5. Menu navigation script

To facilitate the registration and login processes, the *changeInput.cs* script is added to a general object inside the canvas of each scene containing input fields.

By pressing the Tab key in the keyboard, the next input field is automatically selected and the Left Shift key gets the previous input field.

By pressing the Intro key, the specified button (public variable of the script), is automatically pressed.

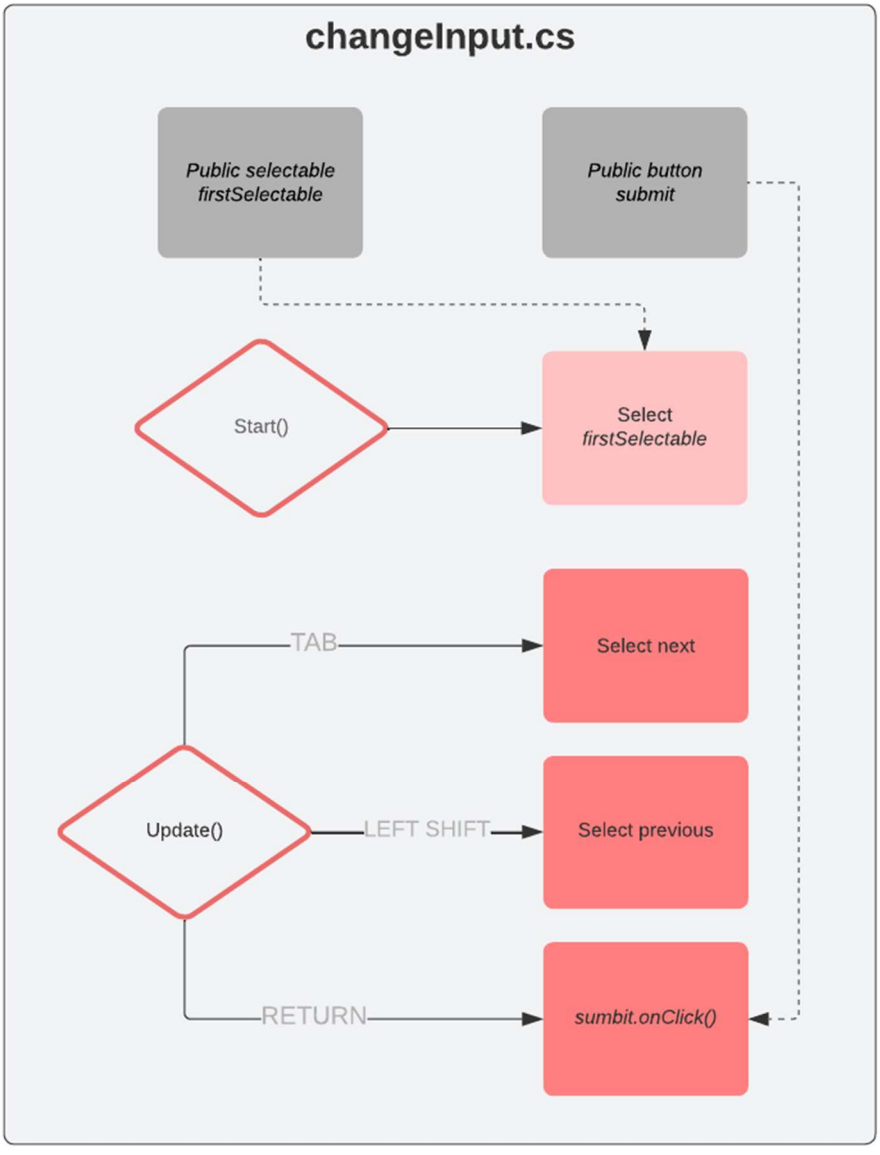


Figure 60: Algorithm used in changInput script

6.5. Image display

The image display component showcases images based on the pathology of the currently logged-in user.

Upon clicking either of the arrows, the image display updates to show the corresponding image. The left arrow allows navigation to the previous image, while the right arrow enables browsing to the next image.

The prompt *imagename* updates to match the name of the image being displayed.

This functionality is controlled by the script *imageDisplayer.cs*.

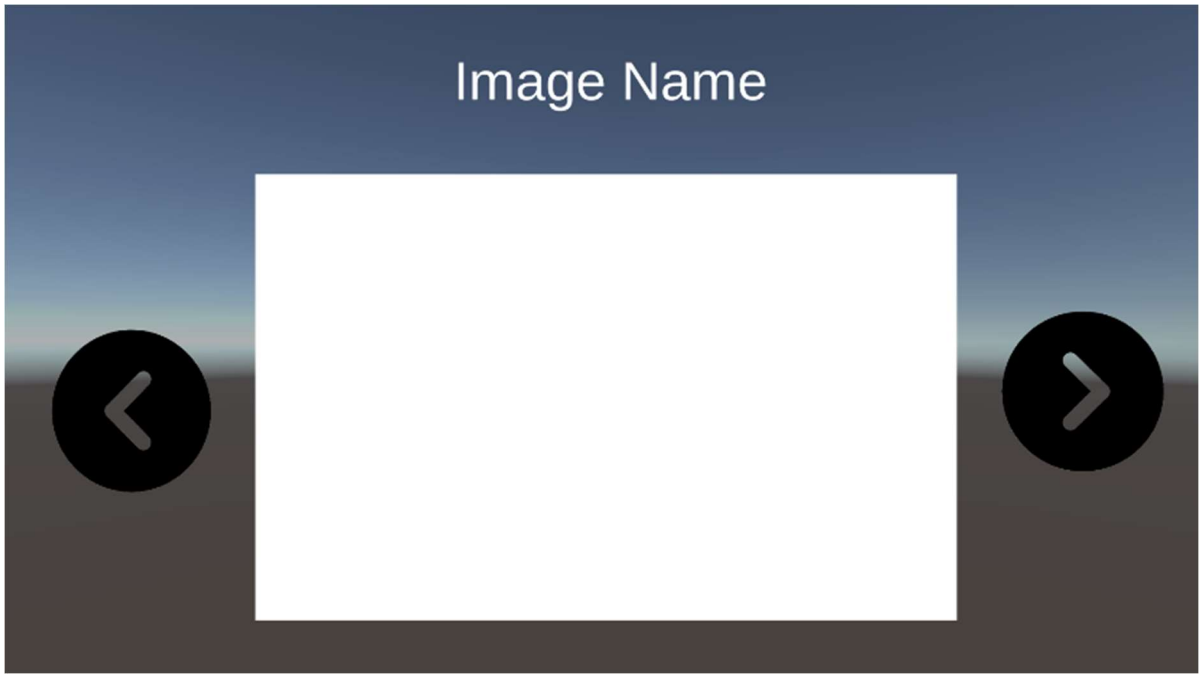


Figure 61: Image displayer in game

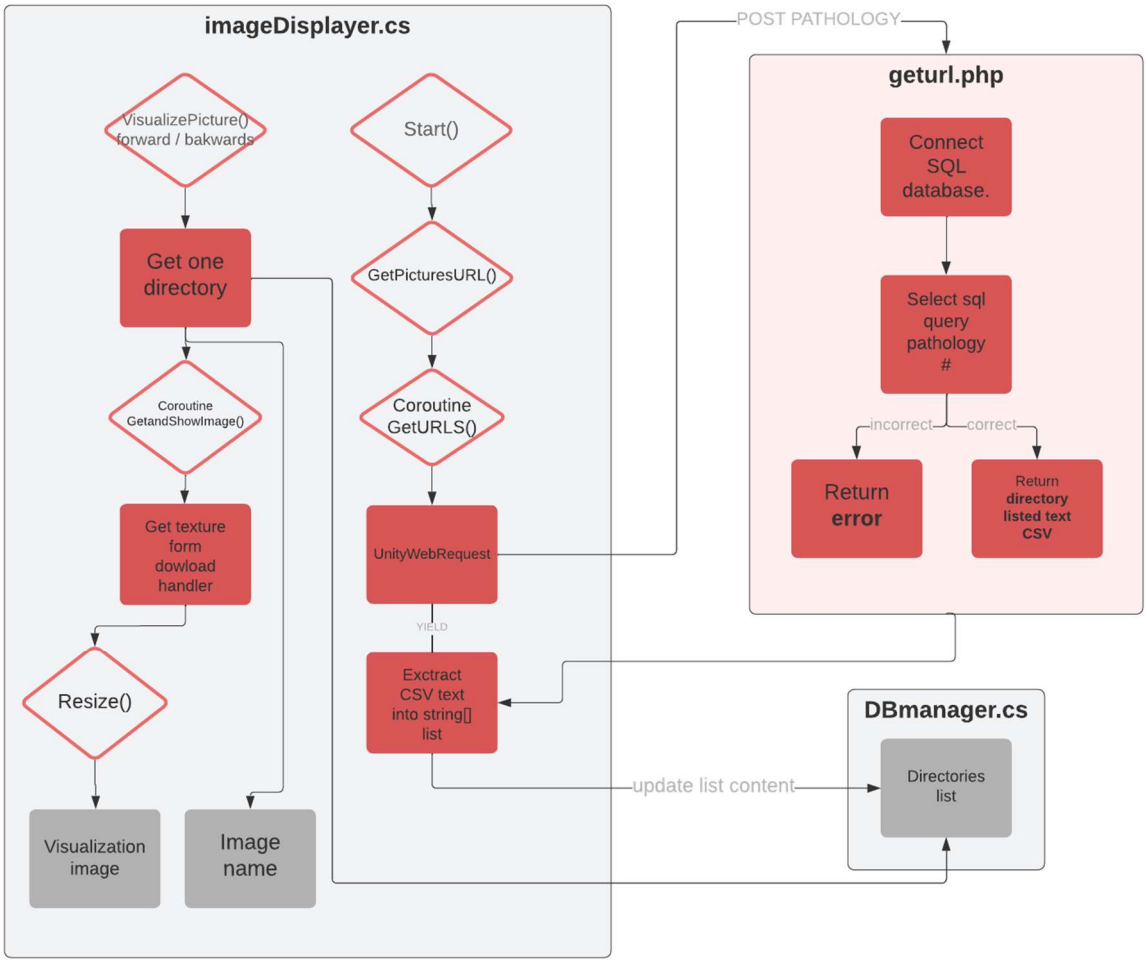


Figure 62: Algorithm implemented for the visuaitzation of images.

6.6. YouTube player

The implementation of a guided meditation feature within the metaverse called for the utilization of a YouTube player. This was accomplished by leveraging the UnityYoutubePlayer project by iBicha, available on the GitHub repository [14].

For our specific project requirements, the goal was to enable the playback of various videos from a curated compilation. This functionality aimed to provide users with the convenience of selecting their preferred meditation video without the need for extensive searching on YouTube.

The original project included a scene featuring a YouTube playlist player, which aligned perfectly with our project's objectives. To incorporate this functionality, only minor modifications were necessary:

- Playlist Controller script: We created a script to initiate the player when a designated button is pressed.
- Background panel: We added a background panel for improved visibility of playlist items.
- All scripts except YoutubePlaylist.cs and VideoPlayerProgress.cs were removed, as they were not required for our specific implementation.

Figure 63: The Youtube player playing a video of yoga

To gain an understanding of the algorithm and behavior of the YouTube playlist player, the accompanying figure serves as a visual representation, outlining its key components and processes.

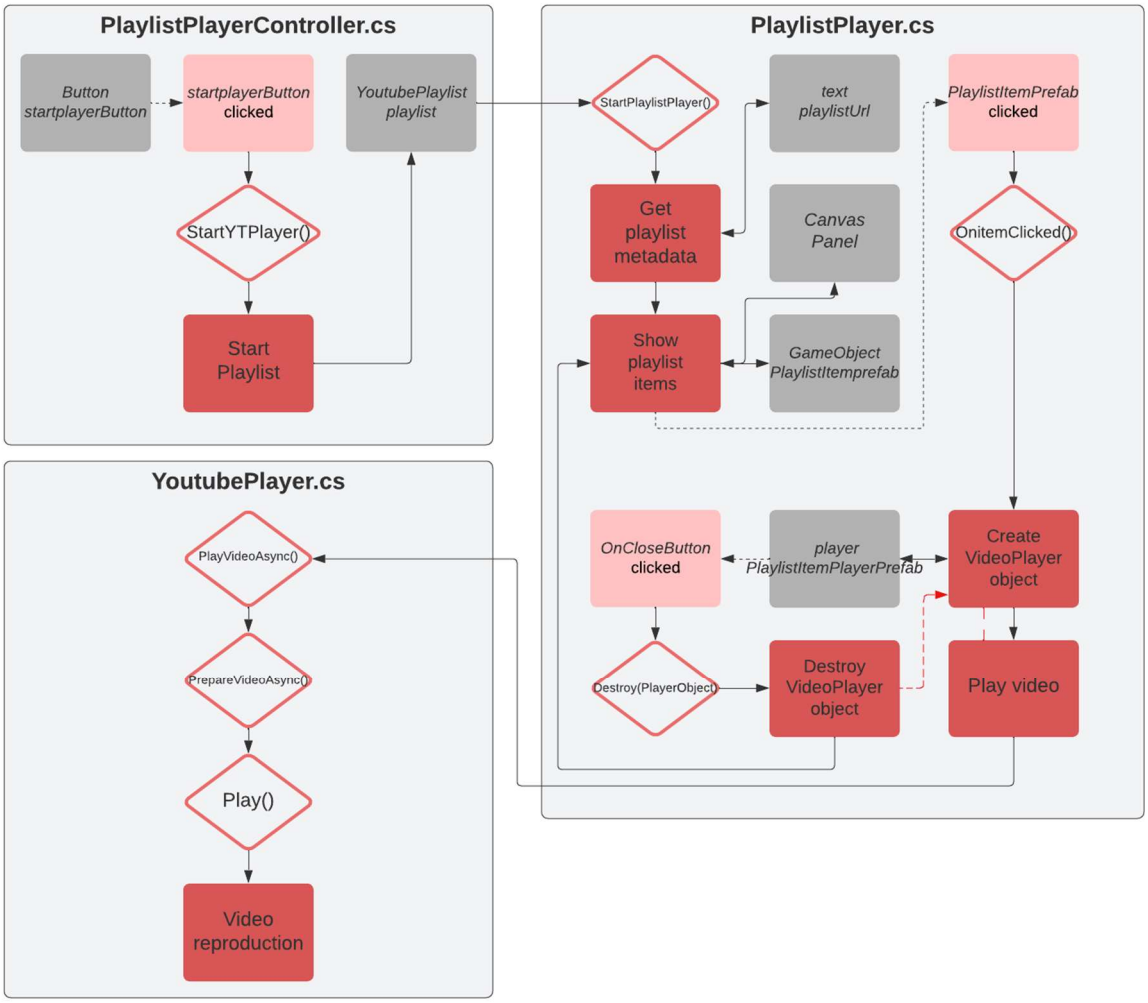


Figure 64: The algorithm used in the youtube playlist player

7. METAVERSE ASSEMBLY

After completing the development of all the packages, the final assembly process commences. The packages are imported into a new Unity Project.

To ensure the proper behavior of each package within the new project, the following requirements and configuration must be met:

- Disable Assembly Version Validation: Edit/Project Settings/Player Settings
- Disable Snap axis box: Edit/Project Settings/Input Manager/ Horizontal and Vertical
- Import CineMachine Package: Window/PackageManager.
- Import Bicha Youtube asset package following the instructions in Readme.md

After importing the packages, several enhancements are implemented to elevate the quality of the user interface and improve the overall playability of the project.

7.1. Choosability of an avatar

Adds the option to choose from 5 different avatars.

The functionality persists within the main menu, providing the option to visualize the avatar. By clicking the designated button, another avatar becomes visible, and the *avatarcode* parameter within the *DBmanager* script is updated accordingly. When transitioning to the Game scene, the avatar chosen in the main menu will spawn within the metaverse environment.

Figure 65: Resulting main menu of the metaverse

7.2. Door animations

By default, the environment package only includes the environment itself and does not provide any animations for opening the doors.

To enable door opening functionality, the following modifications are made to each door:

- Add a box collider trigger to the door object.
- Create an animation controller with states Open and Close
- Create the door animation for opening and closing and assign them to each of the animator controller states.
- Assign the script *OpenDoor.cs.* to the trigger and assign the animator controller of the door in the variable *animDoor* in the inspector tab.

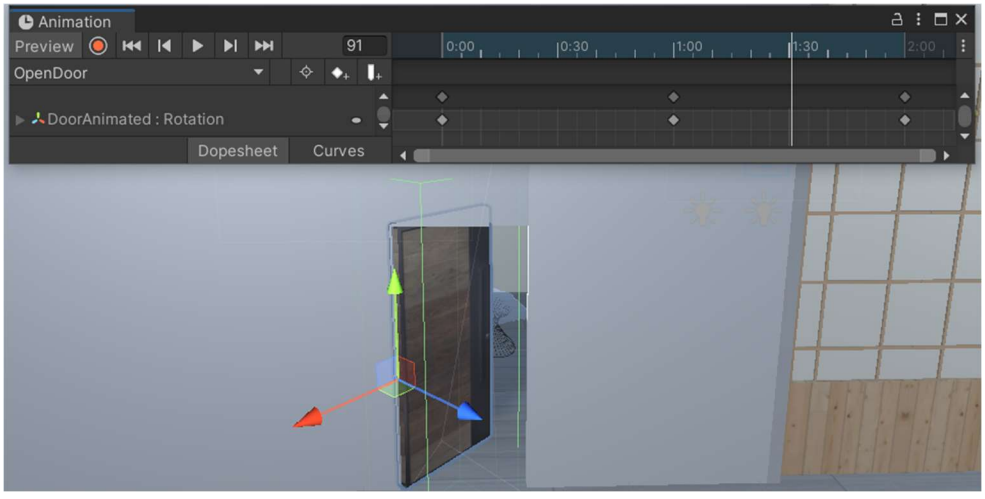


Figure 68: The animation timeline of the door

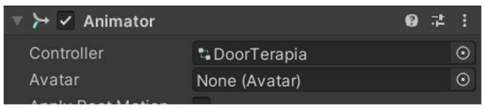


Figure 67: Animator inspector of the door

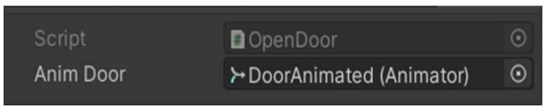


Figure 66: Script inspector of the door trigger

7.2.1. Script algorithm

The required algorithm is quite straightforward: The script contains the functions *OnTriggerEntry()* and *OnTriggerExit()* to set the variable *inZone* to *true* or *false*, respectively, when an object with the "Player" tag (the avatar) enters or exits the trigger zone.

If the condition *inZone* is satisfied and the user presses the E key, the door will change its state (trigger animation) to open or close, depending on its current state.

In the case of an automatic door, the user does not need to press the E key. Entering the zone automatically activates the animation.

7.3. ImageDisplayer and YoutubePlayer into the metaverse

Both the ImageDisplayer and the Youtube player are integrated into a canvas UI, which by default cannot be moved or rotated.

To position these canvases within the scenes, the Render Mode of the canvas needs to be changed to WorldSpace. This adjustment allows for the placement and manipulation of the canvases in the 3D environment.

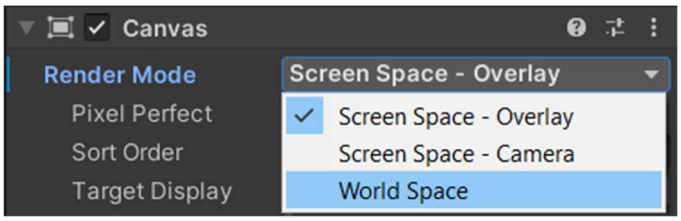


Figure 69: Canvas rendering modes

Once this is done, prefab objects of the ImageDisplayer and the Youtube player are created with the updated WorldSpace configuration and saved as prefab objects. The prefabs are then added to the assembly scene.

The Image Displayer is positioned inside the Therapy room, while the YouTube Player is placed in the center of the tatami within the meditation room. When a video is clicked, a new screen object is created, so the screen prefab also needs to be set in the same location as the YouTube Player. This can be achieved by updating the Playlist Item Player Prefab with the new Screen WorldSpace prefab within the Playlist GameObject.

Figure 70: The youtube playlist player in the meditation room

7.4. Scene Controller

The metaverse scene manager oversees the control of scenes within the virtual world. Acting as a canvas overlay, the scene controller facilitates the spawning of the chosen avatar from the menu.

Visualized as an overlay on the screen, the scene controller serves multiple functions. It provides access to the main menu, showcases game controls, and presents the currently available options.

Figure 72: Text promp when an avatar is able to open a door

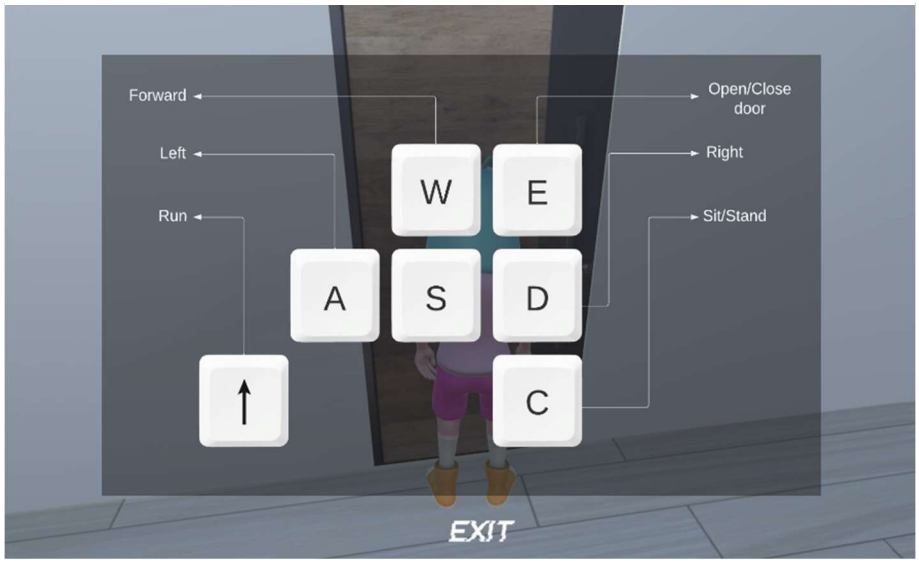


Figure 71: Controls window in game

The scene controller is implemented in the *SceneController.cs* script.

8. GAME BUILD AND DISTRIBUTION

8.1. Building the game

To build the game, navigate to File/Build Settings in Unity and click on the Build button. The game will be built specifically for the Windows x64 platform. However, before initiating the build process, certain settings need to be adjusted to align with our specific requirements.

8.1.1. Quality

The render quality can be adjusted in the Project Settings under the Quality section.

By default, the Standalone platform uses the Ultra quality build setting. However, to enhance the rendering quality, the following settings can be modified:

- Pixel Light Count: 8
- Anti Aliasing: 4x Multi Sampling
- Shadows: Hard and Soft Shadows

8.1.2. Player

The Player settings in the Project Settings tab provide options to customize various aspects of the game, including the company name, product name, version, icon, and default cursor.

In the figure 73 we can see the Player settings for this project

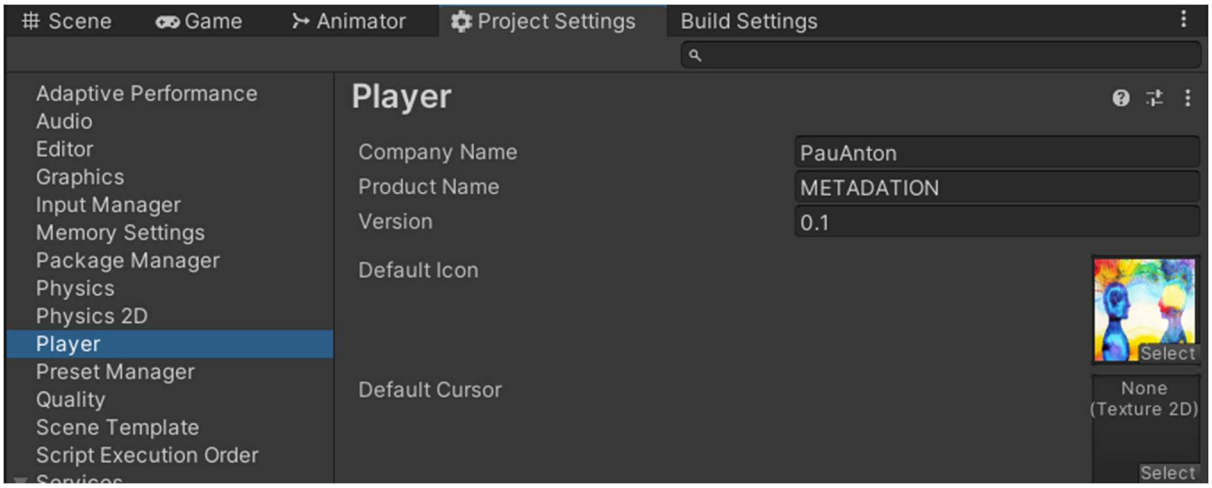


Figure 73: Player settings of the Project Settings tab

8.2. File preparation for distribution

Instead of sharing the standalone .EXE file obtained by building the game, the distribution of the project will be done by creating a comprehensive installer file. This installer file will handle the installation process, including all the necessary files such as the .EXE file and the /data folder, similar to how it is commonly done for applications.

To accomplish this, the Inno Setup Compiler, a third-party application, is utilized [15]. Once installed, following these steps will create the setup file (using version 6.2.2):

- Welcome: Select “Create new script file using the Script Wizard”
- Application information: Write desired application information.
- Application folder: -
- Application files: Select the .exe file produced by Unity as the main executable file and add the other files in the section “Other application files:”
- Application file association: Uncheck “Associate a file type to the main executable.
- Application shortcut: -
- Application documentation: -
- Install mode: -Setup languages: -

- Compiler Settings: Select the folder where the result file of this procedure is going to be located and its name (Metadation setup (x64).
- Compile the result script and save it in the folder selected earlier.

The setup file created using the Inno Setup Compiler can be used to distribute the application to any user.

9. BUDGET AND FEASIBILITY

9.1. Budget

In this section, the budget due to the duration of research, design, 3D modeling, and development as well as for the obtention of software needed for the project will be assessed.

The overall cost is determined by multiplying the number of hours spent on the project by the hourly rate assigned to the project members, in this case myself, and adding it to the infrastructure and indirect costs.

9.1.2. Software

Set as a requirement of the project, all software utilized in this project, including apps, Unity assets and third-party tools like MAMP or Inno were downloaded legally and obtained for free. Therefore, in terms of software expenses, the total cost for the project implementation is zero.

9.1.3. Total budget

Number of hours: 360 h

Licenses: 0 €

Indirect costs (light and infrastructure): 15% of total hourly cost.

For a cost per hour of 15€ the total cost is 6210 €.

9.2. Feasibility

The software used for this project is available for free, and commercialization through licenses is permitted. As a result, there is no cost associated with the deployment of the software.

Considering that updates to the project can be easily implemented by adding new packages or modifying existing ones, the feasibility of the project remains.

Moreover, the deployment process is very straightforward, as it only requires a single file (MetadationSetup.exe) with a size of just 66 MB to be uploaded or downloaded. The installation itself, while currently limited to Windows x64 bit systems, only necessitates approximately 200 MB of local storage.

10. CONCLUSIONS

The primary objective of this project was to design and develop a 3D environment to serve as the foundation for a digital metaverse dedicated to mental health treatment. Extensive research was conducted to identify and select the necessary technologies, software tools for modeling and game production, as well as application distribution methods. The application was intentionally designed to be future-proof, allowing for updates and scalability over time.

The initial project requirements, including the use of free software and the creation of a memory-efficient application, were successfully adhered. With these conditions met, the project was developed to be feasible and sustainable throughout its intended market lifespan.

Despite encountering challenges in terms of conceptualization and implementation, the overall solution and resulting application were successfully developed as a standalone product, offering a fully immersive gaming experience with two functionalities specifically tailored to mental health treatment.

Furthermore, the project effectively laid the groundwork for the integration of AI-powered avatars in the near future. It also presents the potential for future expansion, such as incorporating new packages with enhanced functionalities or updating existing components to improve the overall user experience.

This project serves as a proof-of-concept, demonstrating the creation of a basic metaverse and highlighting the feasibility of developing medical applications with these characteristics. The implementation of metaverses in mental health is a relatively unexplored field with immense possibilities, and this project represents an initial step in understanding their creation and identifying the most suitable technologies for implementation.

In conclusion, despite encountering some unforeseen technological challenges, the primary objective of the project has been successfully accomplished. This project showcases the creation of a basic metaverse and establishes the feasibility of implementing similar technologies for medical purposes, even within low-budget organizations, due to their practicality and ease of distribution.

11. REFERENCES

1. Stephenson, Neal. *Snow Crash*. 1992.

2. Roth, Emma. “AltspaceVR is shutting down as Microsoft's mixed reality division shrinks.” *The Verge*, 21 January 2023, https://www.theverge.com/2023/1/21/23565188/altspace-vr-shutting-down-microsoft-layoffs.

3. “Omniverse Platform for creating and operating metaverse applications.” *NVIDIA*, https://www.nvidia.com/en-us/omniverse/.

4. Cerasa, Antonio. “The promise of the metaverse in mental health: the new era of MEDverse.” *sciencedirect*, November 2022, https://www.sciencedirect.com/science/article/pii/S240584402203050X.

5. “Innerworld.” *Innerworld | Start Your Mental Health Journey In the Metaverse*, https://www.inner.world/home/

6. “PsyTechVR.” *VR Mental Health system for Meta Quest, PICO and HTC*, https://www.psytechvr.com/.

7. “Documentation Manual.” *Unity3d*, https://docs.unity3d.com/Manual/index.html.

8. “Downloads - MAMP.” *MAMP*, https://www.mamp.info/en/downloads/.

9. “Free Double Sided Shaders | VFX Shaders.” *Unity Asset Store*, https://assetstore.unity.com/packages/vfx/shaders/free-double-sided-shaders-23087.

10. “Explore 3D Models.” *Sketchfab*, https://sketchfab.com/3d-models.

11. “Get Animated. Animate 3D characters for games, film and more.” *Mixamo*, https://www.mixamo.com/#/.

12. “AccessingTheUnityFileExplorer.” *GitHub*, https://github.com/The-Ultimate-Developer/AccessingTheUnityFileExplorer.

13. “unity-server-upload.” *GitHub*, https://github.com/francescosoave/unity-server-upload.

14. “UnityYoutubePlayer: Play youtube videos in Unity using youtube-dl and Unity's VideoPlayer.” *GitHub*, https://github.com/iBicha/UnityYoutubePlayer.

15. “Inno Setup Downloads.” *JrSoftware*, https://jrsoftware.org/isdl.php.