UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

Adaptación de Chatbots a Avatares Virtuales con MetaHuman

Animación automática de un Avatar Virtual en Unreal Engine

Adapting chatbots to Virtual Avatars with MetaHuman

Automatic Animation of a Virtual Avatar in Unreal Engine

Realizado por

Antonio Jiménez Godínez

Tutorizado por

David Bueno Vallejo

Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO DE 2022

Resumen

Castellano

En la actualidad se está extendiendo el uso de chatbots y asistentes a través de páginas web, Whatsapp, algunos con voz como el Asistente de Google, Cortana o Siri. A todos ellos les falta un toque más humano que daría una interfaz 3D. En este proyecto se ha trabajado para dar una interfaz 3D genérica en formato de Avatar a esos chatbots. Utilizando tecnologías de texto a voz, analizando los movimientos de la boca que estarían asociados a cada sonido (visema) para generar una interacción en tiempo real lo más realista posible.

En este proyecto se ha realizado un avatar virtual 3D en Unreal Engine 4 utilizando la nueva tecnología de MetaHumans.

Este avatar puede conectarse con un chatbot y, utilizando la tecnología ReadSpeaker Text to Speech, puede obtener los pares de audio y duración de visema a partir de la respuesta del chatbot. El avatar utiliza estos resultados para generar una animación facial de la respuesta.

En este TFG se realiza la parte relacionada con la interacción con el usuario y la animación automática del MetaHuman.

Todo esto se hace con el propósito de ofrecer a los usuarios de este tipo de servicios una experiencia más realista, aprovechando las tecnologías más modernas de generación de voz y creación de avatares virtuales.

En la memoria también se detallará cómo utilizar cada tecnología que ha sido empleada para el desarrollo de este proyecto, ya que gran parte de éste ha consistido en aprender a utilizar tecnologías que se encuentran todavía en desarrollo.

- **Palabras clave** : Unreal Engine, Animación automática, MetaHumans, Chatbot.

Inglés

Nowadays the use of Chatbots is being expanded through web assistants, Whatsapp and some Voice assistants like Google Assistant, Cortana or Siri. A lot of them lack a bit of human touch that could be provided by a 3D interface. In this project we have aimed to give a generic 3D interface to those Chatbots by adding an Avatar. Using Text-to-Speech and analyzing facial movement associated to each sound (viseme) we aim to generate the most realist real time interaction posible.

This Project has consisted in the development of a 3D virtual avatar in Unreal Engine 4 using the new MetaHumans technology.

This avatar can connect with a chatbot and, using ReadSpeaker Text to Speech technology, it can obtain the audio and viseme-duration pairs from the answer of the chatbot. These results are used by the avatar to generate a facial animation of the answer.

This part of the project is centered around the user input and the automatic animation of the MetaHuman.

All of this is done with the purpose of offering the users of this kind of service a more realistic experience, using the most recent voice generating and virtual avatar technologies.

In this document it will be also detailed how to use each technology that has been used in the project, because a big part of it has been the learning process of using technologies that are still in development.

- **Key Words**: Unreal Engine, Automatic Animation, MetaHumans, Chatbot.

Índice

1. Introducción

1.1. Motivación

Recientemente las tecnologías de chatbots han visto una gran aceleración, popularizándose su uso y comercialización. En pocos casos estos chatbots son representados mediante avatares y, cuando lo están, suelen ser de baja calidad.

Con la reciente aparición de la novedosa tecnología de MetaHumans, unos avatares virtuales muy realistas con potencial para una gran expresividad, hemos pensado que al ser conectados con chatbots les darían una gran experiencia a los usuarios.

1.2. Objetivos

Al tratarse de un trabajo en grupo, los objetivos se pueden separar en dos categorías:

- **Grupal**: El objetivo final del trabajo es realizar un avatar virtual en 3D (utilizando MetaHumans de Unreal Engine) capaz de conectarse con un chatbot a través de servicios JSON. Las frases en texto que se reciban del chatbot, se convertirán en voz con la API de textToSpeech de ReadSpeaker. Analizando el audio y el texto se generarán las posiciones de la boca (visemas) y el Avatar reproducirá el sonido asociado. En el otro sentido, se estudiará la entrada de texto para comunicarse con el avatar.
- **Individual**: De todo el proceso descrito anteriormente, en este proyecto se procesarán los datos recibidos por la API de Audio (voz y texto) para generar automáticamente una animación sobre el modelo del MetaHuman al tiempo que se reproduce el sonido para simular el habla del chatbot. También se hará la recogida de la comunicación del usuario usando texto.

1.3. Metodología

Debido a la parte tan alta de innovación del proyecto y a que hay más de una persona implicada en el mismo, la metodología utilizada fue una metodología Ágil basada en Scrum con Sprints de unos 15 días.

1.4. Herramientas Utilizadas

Para este proyecto se han utilizado diversas herramientas:

- **Quixel Bridge**: Se trata de una herramienta que permite el manejo de diferentes assets 3D y su importación a diferentes programas. En este proyecto se ha utilizado para importar un MetaHuman a Unreal Engine.
- **Unreal Engine**: Motor gráfico para videojuegos y producciones virtuales. Se ha utilizado la versión 4.27 como entorno de trabajo para el proyecto.
- **Visual Studio 2019**: Entorno de programación. Se ha utilizado para la parte de programación en C++ necesaria para el proyecto.
- **Live Link Face**: Aplicación de iOS para captura facial. Esta herramienta se encuentra en desarrollo, por lo que todavía no es capaz de grabar fielmente los movimientos faciales. Se ha utilizado para la grabación de las diferentes animaciones de visemas sobre el MetaHuman.

En este documento se explicará cómo emplear las diferentes tecnologías que han sido utilizadas, poniendo énfasis a lo relevante para este proyecto.

2. Como usar MetaHumans en Unreal Engine

2.1. Creación de un MetaHuman

Aunque en este proyecto se ha utilizado uno de los MetaHuman ya generados que se encuentran en Quixel Bridge, es importante para mostrar el potencial de esta herramienta mostrar cómo es posible crear un MetaHuman desde cero.

Para esto accederemos a la página de MetaHuman Creator [1].

En esta página podremos personalizar un MetaHuman completamente. Aunque esta funcionalidad sigue en desarrollo, permite crear avatares con un alto nivel de realismo.

Primero hay que seleccionar uno de los MetaHuman que ofrecen para tomar como base.

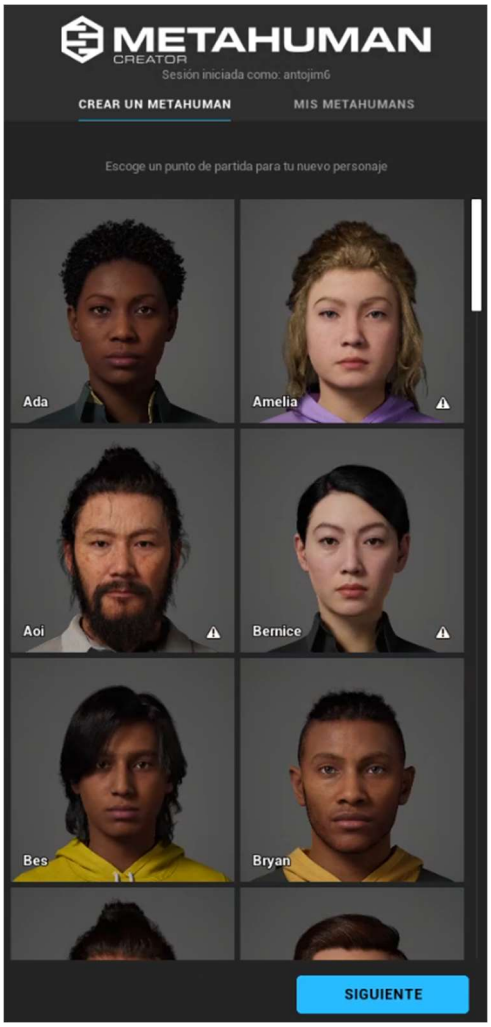

Ilustración 1: MetaHuman base en el creador

Tras esto, el MetaHuman seleccionado será cargado en la pantalla y podremos empezar a modificarlo.

En el lado derecho de la pantalla se muestra la leyenda de todos los controles disponibles para la personalización del avatar y la forma de verlo desde distintas perspectivas.

En la parte izquierda se encuentra el menú que permite la edición del avatar. Este menú está separado en categorías para facilitar la navegación.

Actualmente permite que los usuarios configuren:

- Características faciales
- Tez de la piel
- Tipo de cuerpo
- Maquillaje
- Dientes
- Ropa
- Pelo
    - Cabeza
    - Facial
    - Cejas
    - Pestañas

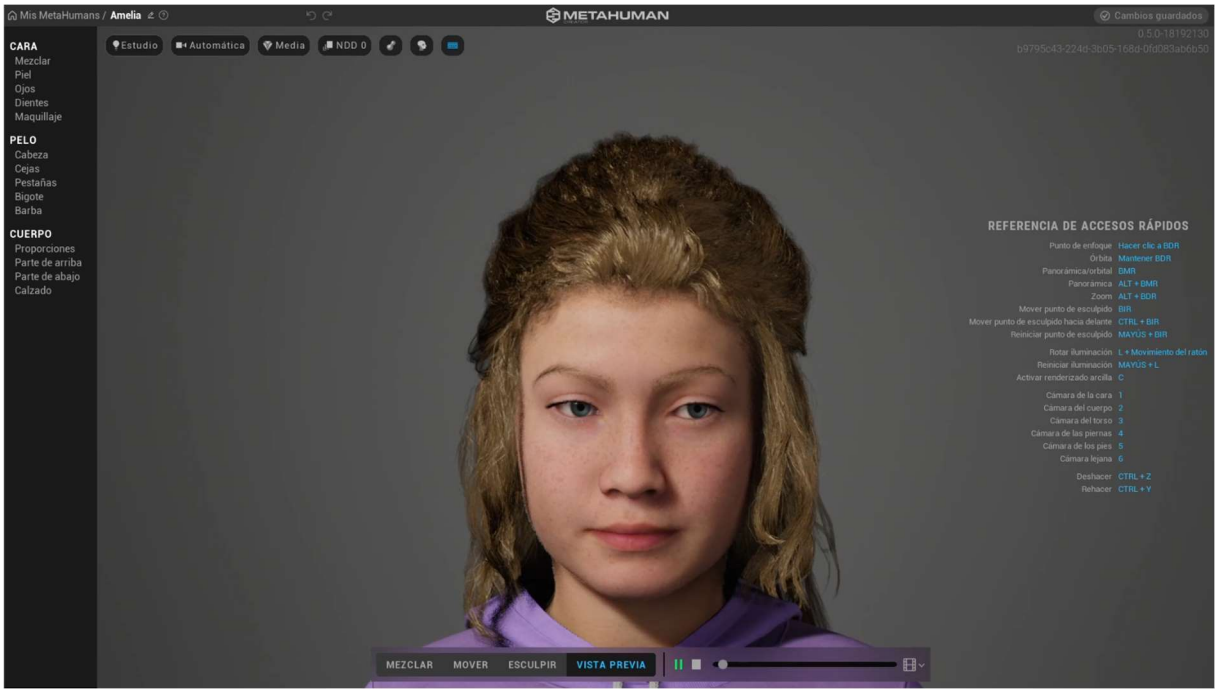

Ilustración 2: Creador de MetaHumans

La aplicación permite editar las condiciones de luz y reproduce continuamente una animación para poder ver el MetaHuman en movimiento.

En la parte superior izquierda podremos renombrar al avatar.

2.2. Importar un MetaHuman a Unreal Engine

Desde Bridge accederemos a la sección de MetaHumans.

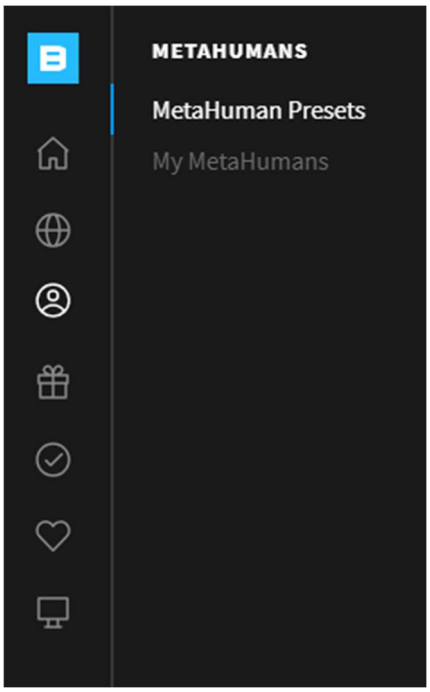

Ilustración 3: Menú de Bridge

Ahí seleccionaremos el MetaHuman que queremos usar y los descargaremos con el botón *Download*. En caso de haber creado nuestro propio avatar, se encontrará en la sección *My MetaHumans*.

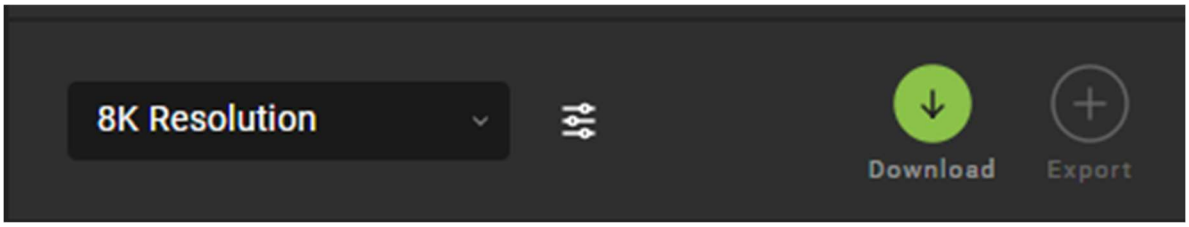

Ilustración 4: Descarga de un MetaHuman

Una vez descargado podremos exportarlo con el botón *Export*. Hay que asegurarse de haber seleccionado correctamente Unreal Engine como objetivo y la versión del motor que estamos utilizando, además del lugar de instalación del plugin y el proyecto por defecto.

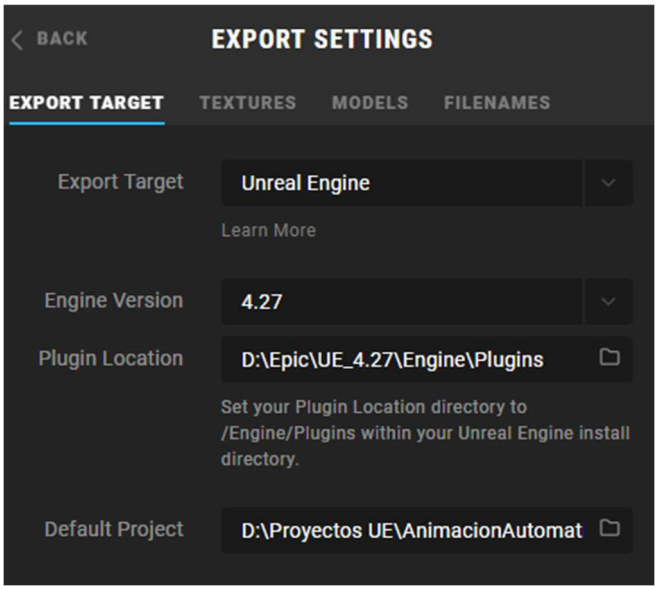

Ilustración 5: Opciones de exportación de Bridge

Ahora, en Unreal Engine nos encontraremos con que el MetaHuman se encuentra en la carpeta *Content* del proyecto.

2.3. Partes de un MetaHuman

Los MetaHumans disponen de una gran cantidad de variables para modificar cada parte del movimiento facial. Esto permite animaciones muy realistas y configurables.

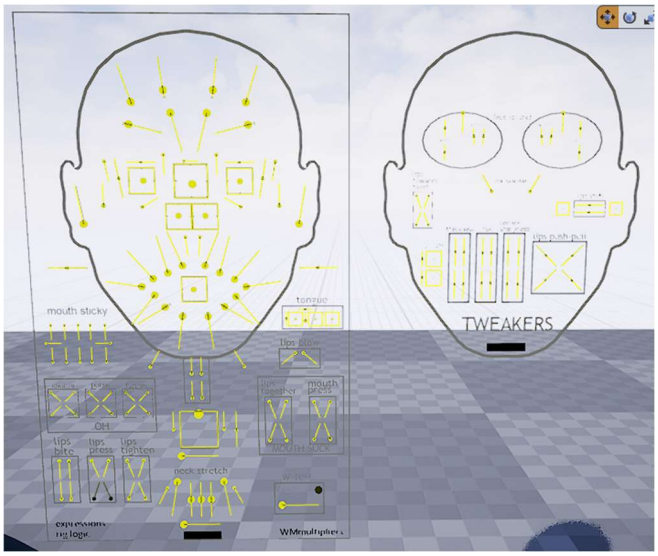

Ilustración 6: Partes de un MetaHuman

Todas estas variables son editables desde la interfaz flotante que aparece en la ilustración, desde el secuenciador, desde los *Blueprints* y desde código C++.

3. Como usar Live Link Face

Para utilizar Live Link Face harán falta dos dispositivos, un dispositivo móvil en el que utilizará la cámara a través de la aplicación y un ordenador personal con Unreal Engine. Ambos han de estar en la misma red local.

Actualmente esta tecnología solo está disponible para teléfonos iOS.

3.1. Dispositivo móvil

Una vez descargada la aplicación de Live Link Face en el dispositivo nos encontraremos con la siguiente pantalla.

Ilustración 7: Aplicación Live Link Face [2]

Entraremos a ajustes utilizando el botón que se encuentra en la parte superior izquierda de la pantalla y veremos la pantalla de opciones. Tras esto, pulsaremos la primera opción, llamada Live Link.

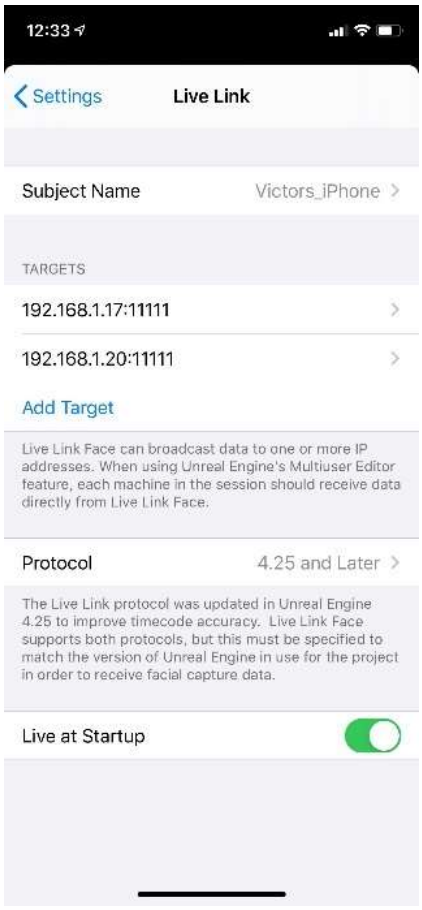

Ilustración 8: Opciones Live Link Face [2]

En esta pantalla definiremos un objetivo, introduciendo un nombre identificativo, la dirección IP del PC en el que estaremos ejecutando Unreal Engine y un puerto.

Tras esto podemos volver a la pantalla inicial y lo que capture la cámara será enviado al PC objetivo.

3.2. Ordenador personal

Lo primero que hay que tener en cuenta para configurar Unreal Engine es que hay que tener habilitados los siguientes plugins:

- Live Link
- ARKit
- ARKit Face Support

En el motor abriremos la pestaña de Live Link desde **Window > Live Link**. El dispositivo debería aparecer listado.

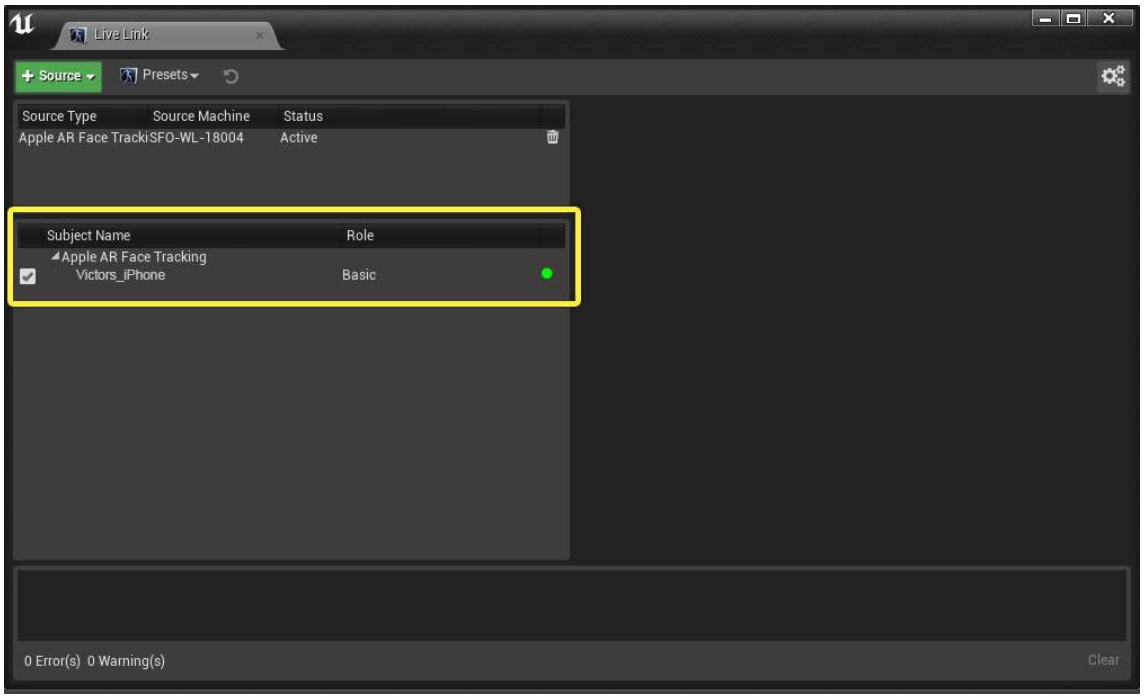

Ilustración 9: Conectar Live Link [2]

Tras esto, iremos a la raíz del Blueprint de nuestro MetaHuman. En la sección *Default*, entrada *LLink Face Subj* seleccionaremos el dispositivo que hemos conectado.

Tras esto el MetaHuman imitará lo reconocido por la aplicación móvil, lo que nos permitirá grabar tomas utilizando la función *Take Recorder* de Unreal Engine. Esto está detallado en el apartado 4.4.

4. Diseño e Implementación

4.1. Investigación previa

Similar a cómo los sonidos del lenguaje pueden ser descritos utilizando fonemas, el movimiento y posición de la boca pueden ser descritos con visemas.

Los visemas son una herramienta utilizada en el campo de la lingüística para estudiar cómo se forman los diferentes sonidos utilizados en el habla. No tienen una correspondencia directa con los fonemas, si no que muchas veces múltiples fonemas están relacionados con un solo visema. Por ejemplo, los fonemas /k, g, ŋ/ corresponden al visema /k/.

Se han realizado investigaciones previamente sobre qué visemas se utilizan en el castellano con el propósito de investigar la sincronización labial de personajes virtuales.

Visema									
Modo / Punto Articulación	Bilabial	labiodental	Linguo-Interdental	Linguo-Dental	Linguo-Alveolar	Post-Alveolar	Liguo-Palatal	Linguo-velar	Uvular
Oclusiva	Sonora [b]			[d]				[g]	
	Sorda [p]			[t]				[k]	
Aproximante Oclusiva	Sonora			[ð]				[ɣ]	
	Sorda								
Fricativa	Sonora [β]				[z]		[ʝ]		
	Sorda	[f]	[θ]		[s]			[x]	[χ]
Nasal	Sonora [m]	[ɱ]			[n]		[ɲ]	[ŋ]	
	Sorda								
Lateral	Sonora				[l]		[ʎ]		
	Sorda								
Africada	Sonora					[ʤ]			
	Sorda						[ʧ]		
Vibrante Simple	Sonora				[ɾ]				
	Sorda								
Vibrante Múltiple	Sonora				[r]				
	Sorda								

Ilustración 10: Tabla de Visemas del Castellano [3]

4.2. Estructura del proyecto

Este proyecto ha sido planteado de forma que pueda ser independiente del de Ignacio Jiménez. El flujo de la ejecución permite la sustitución del tanto del generador de visemas y audio como del Chatbot, ambas partes no cubiertas en este proyecto.

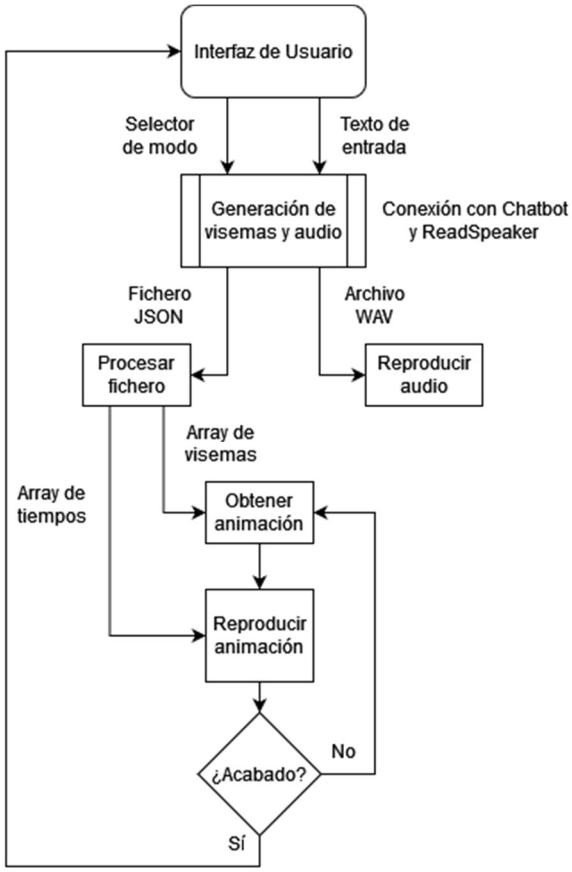

Ilustración 11: Diagrama del proyecto

Esto se ha implementado de forma que este proyecto requiere de otra parte separada que generará los visemas y el audio, para luego ser cargados en Unreal Engine mediante una librería de funciones para Blueprints.

4.3. Escenario

Se ha dispuesto un pequeño escenario para dar una mayor sensación de realismo.

Ilustración 12: Escenario en Unreal Engine

Para esto se han utilizado *assets* gratuitos del Bazar de Unreal Engine [4]. Se ha intentado simular que el avatar se encuentra en un parque, lleno de naturaleza, para producir un entrono agradable a la vista para el usuario.

La cámara está situada frente al avatar y es completamente inmóvil, esto permite ver claramente la vocalización del avatar, que está centrado en la pantalla.

4.4. Captura de Visemas

Para la captura de los visemas se ha hecho uso de Live Link Face utilizando como referencia la siguiente tabla, que lista todos los visemas del castellano, junto con algunos ejemplos.

IPA	X-SAMPA	Descripción	Ejemplo	Visema
Consonantes				
ɾ	4	vibrante simple alveolar	pe**r**o	t
b	b	oclusiva bilabial sonora	**b**estia	p
β	B	fricativa bilabial sonora	be**b**é	B
d	d	oclusiva alveolar sonora	cuan**d**o	t
ð	D	fricativa dental sonora	ar**d**er	T
f	f	fricativa labiodental sonora	**f**ase	f
g	g	oclusiva velar sonora	**g**ato	k
ɣ	G	fricativa velar sonora	tri**g**o	k
j	j	aproximante palatal	hac**i**a	i
ʝ	j\	fricativa palatal sonora	enh**i**elar	J
k	k	oclusiva velar sorda	**c**aña	k
l	l	aproximante alveolar lateral	**l**ino	t
ʎ	L	aproximante palatal lateral	**ll**ave	J
m	m	nasal bilabial	**m**adre	p
n	n	nasal alveolar	**n**ido	t
ɲ	J	nasal palatal	caba**ñ**a	J
ŋ	N	nasal velar	ci**n**co	k
p	p	oclusiva bilabial sorda	**p**ozo	p
r	r	vibrante alveolar	pe**rr**o	r
s	s	fricativa alveolar sorda	**s**aco	s
t	t	oclusiva alveolar sorda	**t**amiz	t
t͡ʃ	tS	africada postalveolar sorda	**ch**ubasco	S
θ	T	fricativa dental sorda	cere**z**a	T
w	w	aproximante velo-labial	f**u**ego	u
x	x	fricativa velar sorda	**j**amón	k
z	z	fricativa alveolar sonora	ra**s**go	s

Vocales				
a	a	vocal abierta anterior no redondeada	t**a**nque	a
e	e	vocal semicerrada anterior no redondeada	p**e**so	e
i	i	vocal cerrada anterior no redondeada	c**i**nco	i
o	o	vocal semicerrada posterior redondeada	b**o**sque	o
u	u	vocal semicerrada anterior no redondeada	p**u**blicar	u

Ilustración 13: Visemas del castellano detallados [5]

Estos visemas fueron grabados utilizando la herramienta *Take Recorder* de Unreal Engine.

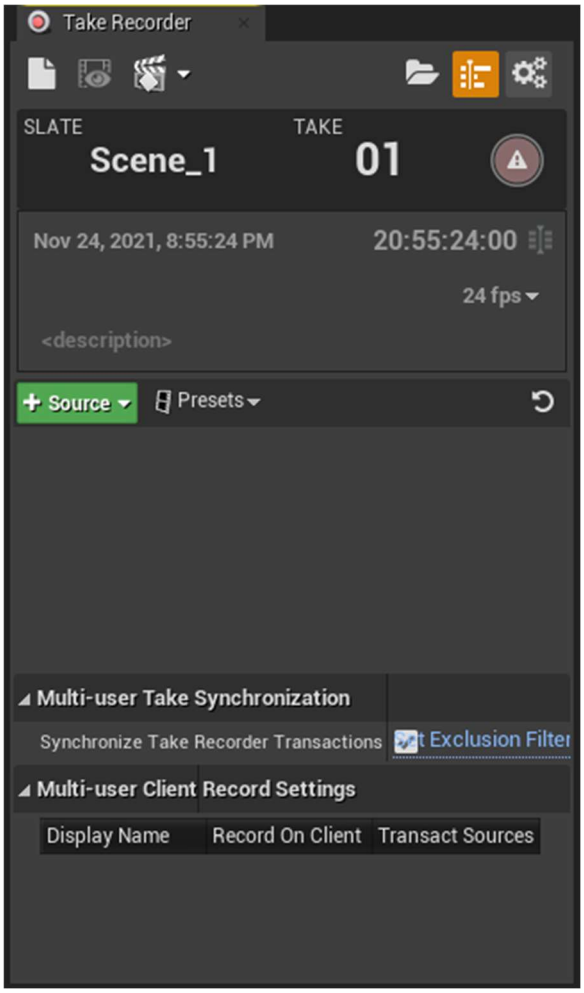

Ilustración 14: Herramienta Take Recorder

Una vez que se tiene funcionando la aplicación Live Link Face, visto en la sección 1, se selecciona el dispositivo móvil como fuente para *Take Recorder* desde: **Source > From LiveLink**.

Tras esto hay que deseleccionar *Use Source Timecode* en la fuente y comprobar que están deseleccionados en *Take Recorder* tanto *Start at Current Timecode* como *Record Timecode* y seleccionado *Record Sources into Sub Sequences*.

Para empezar a grabar se debe entrar en ejecución y luego presionar el botón de grabar, tras haber cambiado el nombre de la grabación y elegido el número de la toma. En este momento se interpreta el visema que se busca animar a la cámara del dispositivo móvil y luego se termina la toma.

El archivo con la secuencia grabada se almacena por defecto en la ruta: **Content > Cinematics > Takes > [Fecha]** .

Ilustración 15: Tomas de Visemas

Ahora hay que entrar en la carpeta que tiene el nombre con formato **[Nombre de grabación]_[Número de toma]_Subscenes** y ahí se encontrará la grabación del visema y la podremos abrir para editarla y transformarla en una animación.

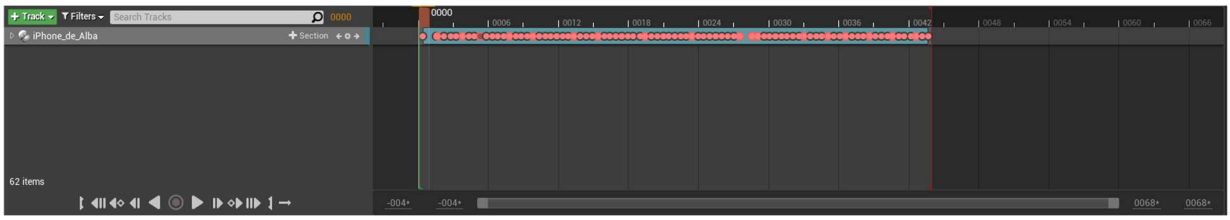

Ilustración 16: Secuenciador

Para que la secuencia pueda ser vista sobre el MetaHuman hay que desbloquear la edición de la secuencia con el candado de la parte superior derecha y después arrastrar el MetaHuman desde el *World Outliner* hasta el secuenciador. Como el cuerpo no es importante podemos deshacernos de la pista llamada *Body*.

Tras esto, para generar la animación pulsaremos el botón izquierdo del ratón sobre Face y en el menú contextual seleccionaremos *Bake Animation Sequence*. Ahora hay

que seleccionar un nombre y el directorio en el que se va a generar, dejando los valores de exportación por defecto.

Para editar esta animación se creará una nueva Secuencia de Nivel desde: **Cinematics > Add Level Sequence** .

A esta secuencia se le añadirá el MetaHuman de la misma forma que a la anterior y también se puede eliminar la pista *Body*. Ahora se le añade una nueva pista a la cara, seleccionando la animación desde: **Track > Animation** .

Se abre el menú contextual sobre *Face* y se selecciona *Bake to Control Rig*, desmarcando la opción *Reduce Keys*.

En este momento, se añade una nueva sección aditiva sobre la pista que se ha generado desde: **Section > Additive** . Tras esto hay dos pistas dentro de la pista nueva, la primera tiene todas las claves y la segunda está vacía. Sobre la pista vacía, abrimos el menú contextual y pulsamos *Key this Section*.

Ahora podremos añadir animaciones por encima de la grabación para modificarla, utilizando esta pista aditiva y la plataforma de control que aparece en Ilustración 6: Partes de un MetaHuman.

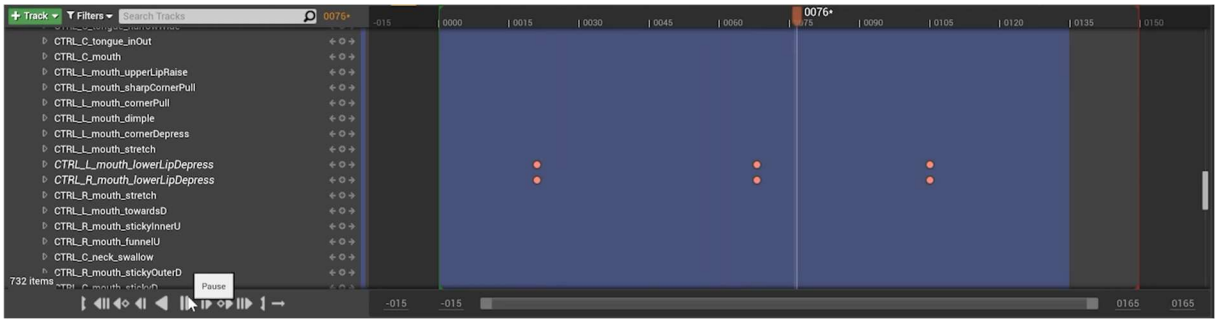

Ilustración 17: Edición de animaciones

Una vez hayamos realizado los cambios, podemos guardarlos como animación de la misma forma que se ha hecho anteriormente.

Una vez tenemos las animaciones de los visemas, crearemos un montaje de animación con cada una de ellas abriendo el menú contextual sobre cada una y seleccionando:

Create > Create AnimMontage . Esto lo hacemos con el propósito de tener transiciones más fluidas entre los distintos visemas cuando se genere automáticamente la animación. Estos montajes de visemas han sido guardados con nombres acordes a la lista de valores que asigna ReadSpeaker a cada visema. Esta asignación puede ser consultada en la página de la Speech API de Microsoft [6].

Los nombres utilizados han seguido el formato: **Visema_[número]_Montaje** .

- 0 = silence
- 1 = ae ax ah
- 2 = aa
- 3 = ao
- 4 = ey eh uh
- 5 = er
- 6 = y iy ih ix
- 7 = w uw
- 8 = ow
- 9 = aw
- 10 = oy
- 11 = ay
- 12 = h
- 13 = r
- 14 = l
- 15 = s z
- 16 = sh ch jh zh
- 17 = th dh
- 18 = f v
- 19 = d t n
- 20 = k g ng
- 21 = p b m

Esta lista contiene más visemas de los que se utilizan en el castellano, pero todos los del castellano están incluidos en ella, por lo que nos es de utilidad para el proyecto.

La relación entre los visemas de la Ilustración 13: Visemas del castellano detallados y la numeración de ReadSpeaker ha sido recogida en esta tabla.

Número	Fonema	Visema
31	silencio	
1	ae ax ah	a
2	aa	
3	ao	o
4	ey eh uh	e
5	er	
6	y iy ih ix	i
7	w uw	u
8	ow	
9	aw	
10	oy	
11	ay	
12	h	
13	r	r
14	l	
15	s z	s
16	sh ch jh zh	J/S
17	th dh	T
18	f v	B/f
19	d t n	t
20	k g ng	k
21	p b m	p

Ilustración 18: Numeración de los visemas

El tener los montajes nombrados según la numeración que utiliza ReadSpeaker permite simplificar el código que los accederá.

4.5. Animación automática

Para la parte de generación automática de la animación se han utilizado los Blueprints de Unreal Engine, junto con una librería de funciones propias.

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Kismet/BlueprintFunctionLibrary.h"
#include "Dom/JsonObject.h"
#include "AutomaticAnimationUtils.generated.h"

typedef TSharedPtr<FJsonObject> JsonObjectPtr;

UCLASS()
class ANIMACIONAUTOMATICA_API UAutomaticAnimationUtils : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

    UFUNCTION(BlueprintCallable)
        static bool LeerVisemas(TArray<FString>& visemas, TArray<float>& tiempos);

    UFUNCTION(BlueprintCallable)
        static void GenerarVisemas(bool esLeer, FString texto);

    UFUNCTION(BlueprintCallable)
        static UAnimMontage* BuscarMontaje(FString visema);

    UFUNCTION(BlueprintCallable)
        static USoundBase* CargarAudio();
};
```

Ilustración 19: Definición de las funciones utilizadas.

Los Blueprints de Unreal Engine son una forma de programación gráfica. Consisten en una serie de nodos interconectados que pasan parámetros entre ellos y realizan funciones sobre los mismos.

Unreal Engine ofrece un gran número de funciones ya preparadas para su uso en este sistema, pero para aumentar el potencial de esta herramienta se permite la creación de librerías de funciones, como la vista en la Ilustración 19: Definición de las funciones utilizadas. En estas librerías se pueden implementar funcionalidades para luego aplicar en los Blueprints.

Hay dos formas de animar el avatar, a través de la conexión con el Chatbot o dándole un texto para que lea en voz alta.

El proceso se puede dividir en tres apartados distintivos: la entrada del usuario mediante una interfaz gráfica, la conexión con ReadSpeaker y la animación automática del avatar.

4.5.1. Interfaz Gráfica

Se ha creado una pequeña interfaz para que el usuario pueda introducir texto para ser procesado.

Ilustración 20: Pantalla en ejecución

La parte con la que se puede interactuar es el cuadro de texto y los dos botones.

Ilustración 21: Cuadro de entrada de texto

La interfaz es creada desde el blueprint de animación al empezar la ejecución y se almacena una referencia para su posterior uso.

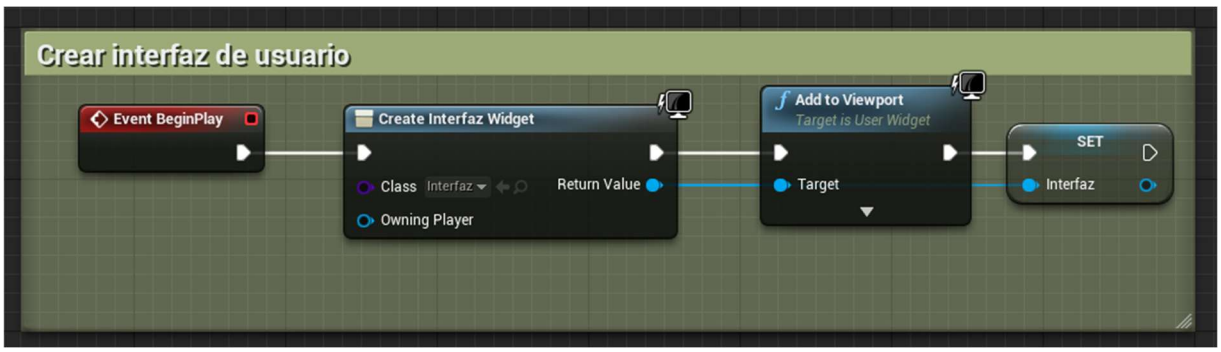

Ilustración 22: Creación de la Interfaz Gráfica

El Blueprint utilizado para esta interfaz es el siguiente.

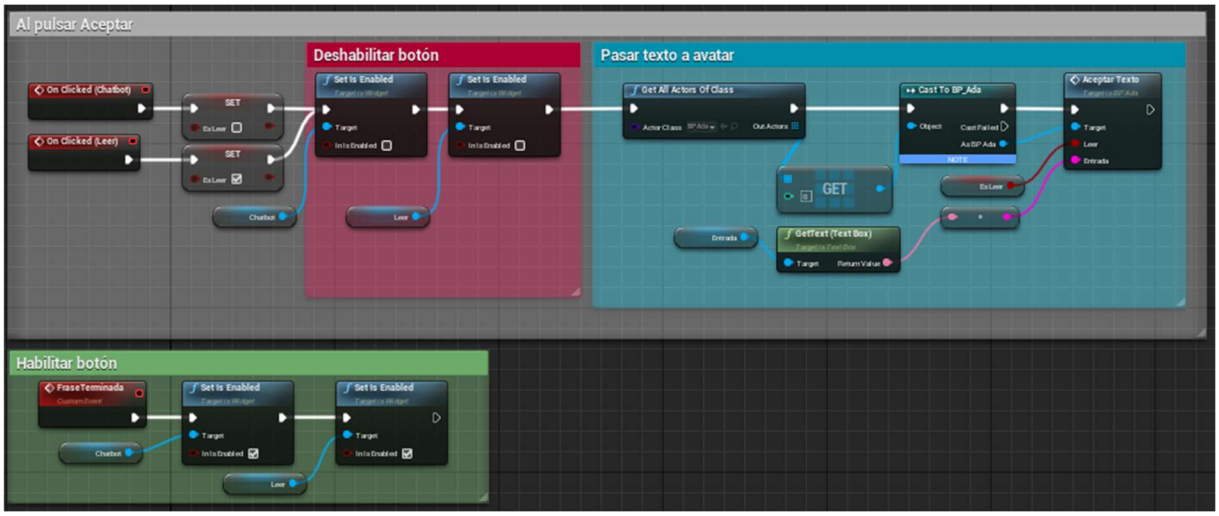

Ilustración 23: Blueprint de Interfaz Gráfica

Tiene dos flujos de ejecución distintos, en base a dos tipos de eventos diferentes:

- Cuando el algún botón es pulsado, deshabilita ambos botones y obtiene una referencia al Blueprint en el que se procesará el texto y se generará la animación del avatar. Tras esto pasa el texto y la información de qué botón se ha pulsado a ese Blueprint utilizando la función Aceptar Texto.
- Cuando sucede el evento Frase Terminada, interpreta que el Blueprint de animación ha terminado y los botones vuelven a ser habilitados.

En el Blueprint de nivel se modifican algunos valores del controlador de la cámara para dejar la escena estática y que se muestre el ratón.

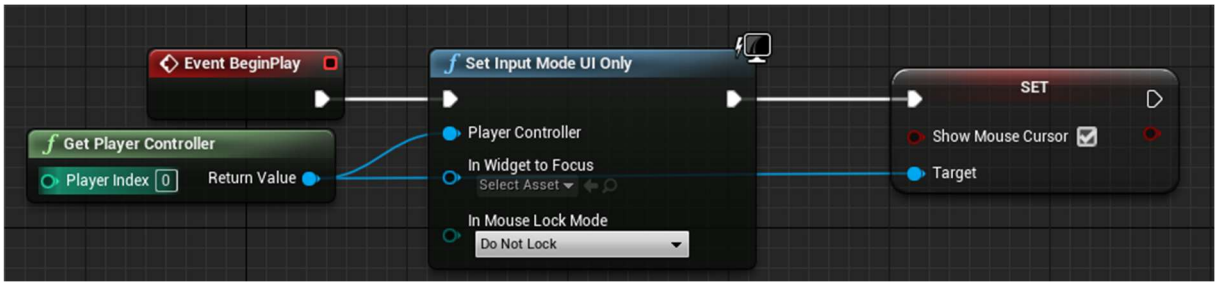

Ilustración 24: Blueprint de Nivel

4.5.2. Conexión con ReadSpeaker

En el Blueprint en el que sucederá la animación automática se recoge el texto de entrada mediante el evento Aceptar Texto, que tiene como parámetros la entrada del usuario y la información de qué botón se ha pulsado.

Estos parámetros son luego transmitidos a la función Generar Visemas, que a su vez los pasa por la librería que procesará la información para producir los visemas y el fichero de audio con la voz.

Ilustración 25: Blueprint de conexión con ReadSpeaker

El código que lleva esto a cabo es el siguiente.

```
// Función para llamar a la librería que generará los visemas y el audio
void UAutomaticAnimationUtils::GenerarVisemas(bool esLeer, FString texto)
{
    if (esLeer) // Caso leer
        tts::ttsText(TCHAR_TO_ANSI(*texto));
    else        // Caso chatbot
        tts::ttsAnswer(TCHAR_TO_ANSI(*texto));
}
```

Ilustración 26: Código de conexión con ReadSpeaker

Esta función llama a la librería que procesará el texto, creando un proceso, y le pasará los parámetros necesarios.

El funcionamiento de esta librería viene detallado en la memoria de mi compañero, Ignacio Jiménez.

El archivo JSON y el archivo de audio son generados en un directorio definido en un fichero de parámetros de la librería.

Para importar la librería se debe modificar el archivo *build* de la librería de funciones en la que se vaya a implementar. Para esto hay que añadir la ruta en la que se encuentran los *headers* necesarios al *path* y luego importar los ficheros de librería. Esto se hace mediante estos métodos:

```
// Ruta hasta las librerías
string LibrariesPath = Path.Combine(ThirdPartyPath, "TTS", "Libraries");

// Librerías estáticas
PublicAdditionalLibraries.Add(Path.Combine(LibrariesPath, "libtts.lib"));
PublicAdditionalLibraries.Add(Path.Combine(LibrariesPath, "libvtapi.lib"));

// Archivos header
PublicIncludePaths.Add(Path.Combine(ThirdPartyPath, "TTS", "Includes"));
```

Ilustración 27: Importar librerías en Unreal Engine

4.5.3. Blueprint de Animación

Una vez se obtiene la salida de ReadSpeaker, se pasa la respuesta por el siguiente proceso.

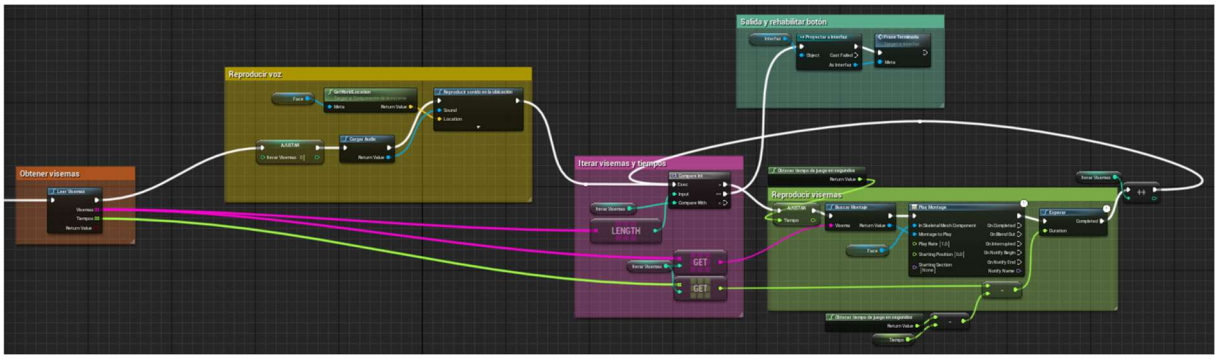

Ilustración 28: Blueprint Animación automática

Este proceso puede ser separado en diferentes partes: la obtención de los visemas, la reproducción del fichero de audio, la iteración de los visemas, la reproducción de los visemas y el final de la ejecución.

4.5.3.1. Obtener Visemas

Se ha definido una función llamada Leer Visemas, que se encarga de buscar el fichero JSON y extraer la información de los visemas de él.

Ilustración 29: Blueprint Leer Visemas

El código que lleva esto a cabo es el siguiente.

```
// Función para leer el archivo JSON generado
bool UAutomaticAnimationUtils::LeerVisemas(TArray<FString>& visemas, TArray<float>& tiempos)
{
    // Variable que almacenará los datos
    FString datos;

    // Ruta hasta el archivo JSON
    FString ruta(Path / "visemas.json");

    // Cargar el archivo en la variable
    bool bArchivoCargado = FFileHelper::LoadFileToString(datos, *ruta);
    FString jsonRaw = FString::Printf(TEXT("{\"data\": %s}"), *datos);

    if (bArchivoCargado) {
        JsonObjectPtr objetoJson = MakeShareable(new FJsonObject());
        TSharedRef<TJsonReader<>> JsonReader = TJsonReaderFactory<>::Create(jsonRaw);

        if (FJsonSerializer::Deserialize(JsonReader, objetoJson)) {
            // Iterar datos del JSON
            auto arr = objetoJson->GetArrayField("data");

            for (int i = 0; i < arr.Num(); i++) {
                visemas.Add(arr[i]->AsArray()[0]->AsString());
                tiempos.Add(arr[i]->AsArray()[1]->AsNumber());
            }
            return true;
        }
    }
    return false;
}
```

Ilustración 30: Código Leer Visemas

Esta función lee los datos del archivo JSON y los divide en dos *arrays*, uno con los visemas y otro con las duraciones de los mismos, ambos de la misma longitud, ya que por cada visema hay un tiempo y viceversa.

Estos *arrays* están definidos como parámetros de entrada por referencia, visemas es un *array* de cadenas de texto, utilizando el tipo **FString** provisto por Unreal Engine, y tiempos es un *array* de números decimales, utilizando el tipo primitivo **float**.

4.5.3.2. Reproducir Voz

Para esta parte se obtiene la posición en el mundo de la cara del MetaHuman y se reproduce el fichero de audio en esa posición.

El propósito de esto es que, si el avatar se implementa en un entorno virtual navegable, la voz del MetaHuman sea un sonido tridimensional, para una mayor inmersión.

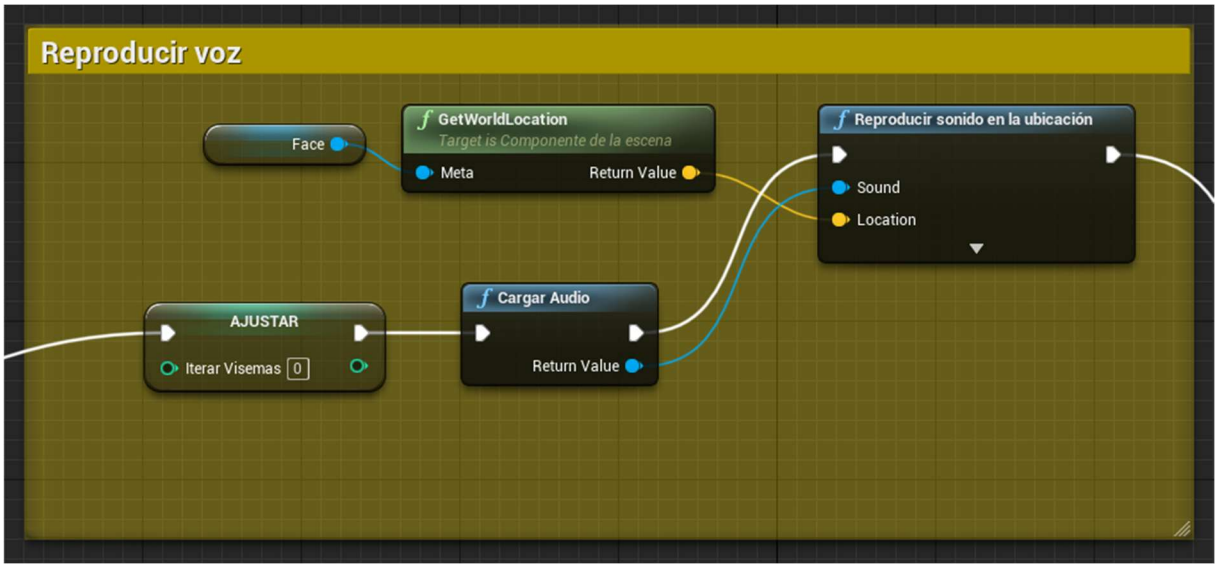

Ilustración 31: Blueprint Reproducir voz

En esta parte también se establece la variable que se va a utilizar para iterar los visemas a cero.

Se ha definido en la librería de funciones el método Cargar Audio, que accede a una ruta por defecto en la que cargará el archivo allí generado por la librería.

```
// Función que carga el archivo de audio generado
USoundBase* UAutomaticAnimationUtils::CargarAudio()
{
    // Ruta hasta el archivo de audio
    FString ruta = "USoundBase'/Game/RecursosProyecto/audio.audio'";

    return LoadObject<USoundBase>(nullptr, *ruta);
}
```

Ilustración 32: Función Cargar Audio

4.5.3.3. Iterar visemas

Esta parte está hecha completamente utilizando las herramientas que proveen los Blueprints de Unreal Engine.

La lógica de esta sección es similar a un bucle *for* clásico, pero no se ha podido utilizar el provisto por los Blueprints de Unreal Engine, debido a que dicho bucle ha de terminar en una *frame* de ejecución mientras el necesario para este proyecto necesita ser ejecutado durante la reproducción del audio.

Ilustración 33: Blueprint Iteración de visemas

Se utiliza la variable Iterar Visemas para controlar la ejecución del iterador.

Esta variable es comparada en cada ciclo con la cantidad de visemas que tenemos, si no hemos iterado todos reproducirá el siguiente visema y si hemos terminado dirigirá el flujo de ejecución a la salida.

Esta comparación se realiza con la función *Compare Int*, que permite separar el flujo de ejecución del Blueprint.

El incremento de la variable de iteración se sucede tras la reproducción de los visemas, tras haber reproducido la animación durante el tiempo especificado en el array de tiempos.

Se toma el valor de la esta variable de iteración para acceder a la posición necesaria tanto en el *array* que contiene los visemas como en el que contiene los tiempos, utilizando el método *GET* que proveen los Blueprints.

4.5.3.4. Reproducir visemas

Esta es la sección en la que se anima al avatar virtual.

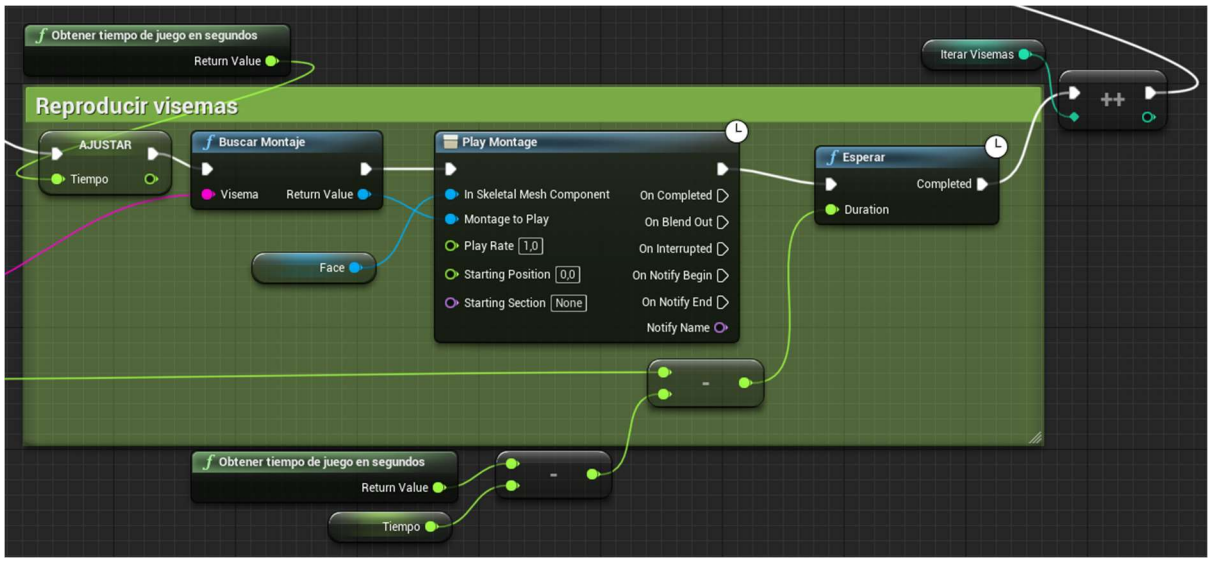

Ilustración 34: Blueprint Reproducir visemas

Se ha creado la función Buscar Montaje, que tiene como entrada el visema a animar en este momento y devuelve el montaje de animación correspondiente a dicho visema.

```
UAnimMontage* UAutomaticAnimationUtils::BuscarMontaje(FString visema)
{
    FString ruta = "/Game/RecursosProyecto/Animaciones/Visema_" + visema + "_Montaje.Visema_" + visema + "_Montaje";
    UAnimMontage* anim = LoadObject<UAnimMontage>(nullptr, *ruta);
    return anim;
}
```

Ilustración 35: Código Buscar Montaje

Esto proceso es sencillo debido a la nomenclatura utilizada al generar los montajes en la sección 4.4.

La salida de esta función es animada sobre la cara del MetaHuman durante el tiempo especificado en el archivo JSON.

Para esto se utiliza la función *Play Montage*, a la que se le pasan como parámetros de entrada la cara a animar y el montaje que se utilizará. La función *Delay* se encarga de que la duración sea la adecuada, tomando el valor correspondiente del *array* de tiempos y restándole el tiempo que ha tardado en procesarse la ejecución de la animación. Esto se hace mediante una sencilla fórmula:

$$e_f = e_0 - (t_f - t_0)$$

Siendo e_f el tiempo final a esperar, e_0 el tiempo de espera del *array*, t_f el momento tras la función *Play Montage* y t_0 el momento previo a la función Buscar Montaje.

Tras esto, se incrementa la variable de iteración de los visemas y se vuelve al iterador.

4.5.3.5. Salida del Blueprint

Esta sección se encarga del final de la ejecución.

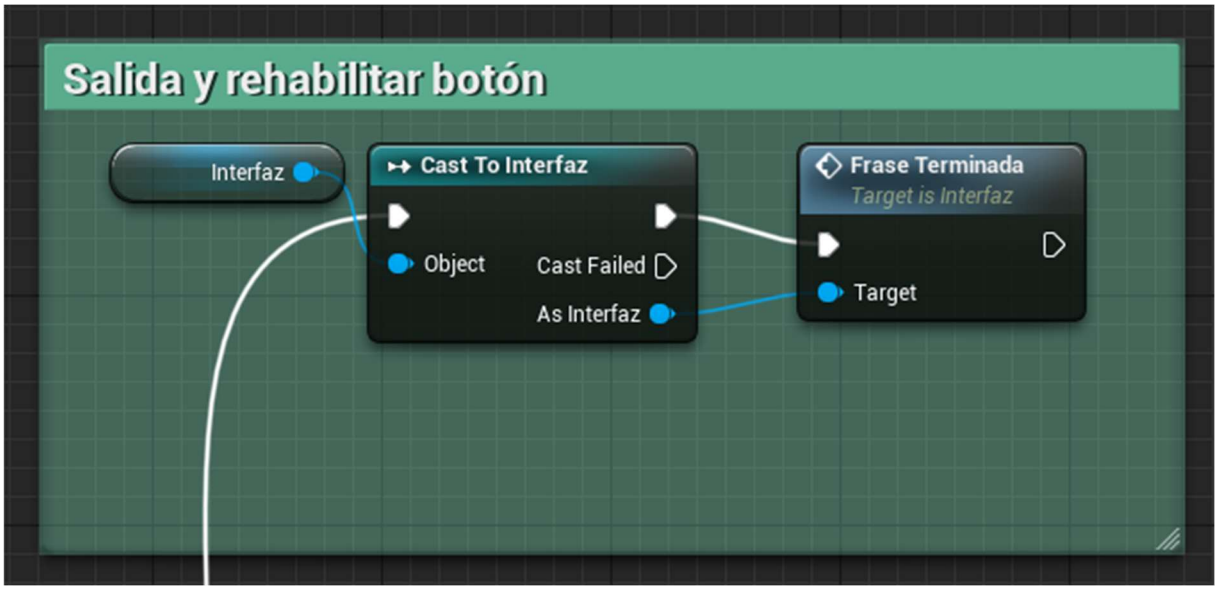

Ilustración 36: Blueprint Fin de ejecución

En este paso se toma la referencia a la interfaz guardada en el momento de su creación y se llama a la función Frase Terminada, para así comunicarle al Blueprint de la interfaz que debe rehabilitar los botones y así permitir que se pueda volver a introducir texto.

4.6. Entregables y uso

Mi parte del trabajo será entregada como dos proyectos separados, ambos en Unreal Engine 4.27.

El primero se tratará del proyecto con las librerías de mi compañero en el que será necesario instalar el motor de ReadSpeaker para su correcto funcionamiento y el segundo será un *mockup* independiente del proyecto de Ignacio en el que se podrán introducir frases predefinidas y obtener su respuesta.

Esto es con el propósito de que se pueda probar mi parte con independencia del otro proyecto. En este *mockup* se podrán introducir las siguientes frases:

- Botón Chatbot:
    - Buenos días -> Hola, espero que tengas un buen día.
    - ¿Qué tal estás? -> Me encuentro muy bien.
- Botón Leer:
    - El cielo está enladrillado, ¿quién lo desenladrillará? El que lo desenladrille, buen desenladrillador será.

En caso de que se introduzca otra cosa responderá con “Texto inválido”.

También se ha hecho una grabación sobre el funcionamiento del proyecto con una demo del mismo. Esta grabación está disponible Youtube bajo el título Animación automática de un Avatar Virtual en Unreal Engine [7].

Para modificar las animaciones utilizadas en el proyecto de Unreal Engine, basta con reemplazar los montajes de animación que se encuentran en la ruta /**Recursos Proyecto/Animaciones** dentro de la carpeta de contenido, dándole a las animaciones nuevas nombres acordes a la nomenclatura especificada en la sección 4.4.

5. Conclusiones

Los MetaHumans de Unreal Engine son una tecnología novedosa con potencial para implementarse en entornos virtuales que aspiran a un gran realismo, como museos virtuales, videojuegos y nuevos chatbots.

Este proyecto es el primer paso hacia ese propósito, con muchas formas de ser expandido, como implementar compatibilidad con más idiomas, entrada de audio por voz, expresividad facial de emociones y gesticulación en el cuerpo, entre otras tantas.

Unreal Engine es una herramienta muy útil para este tipo de proyecto, debido a la facilidad con la que maneja la animación de avatares y la simplicidad en crear entornos realistas en los que implementarlos.

Por otra parte, la documentación de este motor, específicamente la parte correspondiente a los MetaHumans, tiene carencias y no hay muchas fuentes de las que obtener información de forma gratuita acerca de las herramientas que se han utilizado para este proyecto.

Gran parte de este problema viene de que los MetaHuman siguen siendo una tecnología en desarrollo y otra por el menor tamaño de la comunidad de usuarios de Unreal Engine frente a otras tecnologías similares, como el motor Unity. Esto hace que sea difícil encontrar soluciones a problemas que van surgiendo durante el desarrollo, ralentizando así el progreso del proyecto.

Líneas futuras que continúen con el rumbo del proyecto podrían añadir elementos de interactividad con el avatar, implementación de este en un entorno navegable, mejorar la expresividad conforme avance la tecnología de captura de animaciones, entre otras opciones.

Ilustraciones

Bibliografía

[1] Unreal Engine, «MetaHuman Creator,» [En línea]. Available: https://www.unrealengine.com/en-US/metahuman-creator.

[2] Unreal Engine, «Recording Facial Animation from an iOS Device,» [En línea]. Available: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/FacialRecordingiPhone/.

[3] M. Morales Rodríguez, J. Radilla Avila, A. Ramírez, J. Ramírez Saldivar, J. J. González Barbosa y H. Fraire Huacuja, «Silabeo Automático para la Generación de Vi-Silabas en Español,» de *17th International Congress on Computer Science Research (CIICC'11)*, Morelia, Michoacán, 2011/10/26.

[4] TomkaGS, «Unreal Engine Marketplace,» [En línea]. Available: https://www.unrealengine.com/marketplace/en-US/product/stylized-forest-03.

[5] Amazon, «Español (es-Es) - Amazon Polly,» [En línea]. Available: https://docs.aws.amazon.com/es_es/polly/latest/dg/ph-table-spanish.html.

[6] Microsoft, «SpeechVisemeType (SAPI 5.3),» [En línea]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms720881(v=vs.85).

[7] A. Jiménez Godínez, «Animación automática de un Avatar Virtual en Unreal Engine - Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=BOYZld3HmeI.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática

Bulevar Louis Pasteur, 35

Campus de Teatinos

29071 Málaga