



Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN DESARROLLO Y DISEÑO DE VIDEOJUEGOS

Curso Académico 2023/2024

Trabajo Fin de Grado

**INTEGRACIÓN DE INTELIGENCIA ARTIFICIAL COMO HERRAMIENTA EN EL
DESARROLLO DE VIDEOJUEGOS**

Autor: Baldomero Rodríguez Árbol

Director: Dan Casas Guix

©2024 *Baldomero Rodríguez Árbol*

Algunos derechos reservados

Este documento se distribuye bajo la licencia "Atribución 4.0 Internacional" de Creative Commons, disponible en: <https://creativecommons.org/licenses/by/4.0/deed.es>

RESUMEN

En la actualidad, el uso de redes neuronales está en constante crecimiento, lo que hace fundamental contar con herramientas que gestionen eficientemente estas tecnologías para maximizar su potencial. En respuesta a esta necesidad, se ha desarrollado el programa **Artificial Intelligence Manager For Developers** (AIM4D), concebido como una herramienta auxiliar que apoya tareas específicas a través de redes neuronales generativas. AIM4D está diseñado para que usuarios sin conocimientos técnicos puedan aprovechar las ventajas de las redes neuronales, especialmente en el desarrollo de videojuegos y gráficos 3D. La interfaz y funcionalidades de AIM4D están adaptadas a este entorno, proporcionando una solución accesible y eficiente para la integración de inteligencia artificial.

AIM4D se caracteriza por su arquitectura modular, que permite la utilización simultánea de múltiples redes neuronales. Esta modularidad facilita la integración y el manejo de diversas tecnologías de IA, permitiendo a los desarrolladores cargar y gestionar diferentes modelos de manera flexible y eficaz. El programa también cuenta con un modo servidor, que mejora significativamente la colaboración y el acceso remoto a recursos de inteligencia artificial, optimizando el flujo de trabajo y permitiendo la gestión de peticiones en línea. Esta capacidad es crucial para equipos de desarrollo distribuidos y para cualquier entorno que requiera una solución robusta y escalable para el uso de redes neuronales.

Como parte del proyecto, se ha desarrollado una demo en Unity para probar y demostrar las funcionalidades de AIM4D. Esta demo permite a los usuarios conectarse al servidor, enviar peticiones para generar avatares 3D y texturas, y visualizar los resultados en tiempo real.

Palabras clave

Inteligencia Artificial, Deep Learning, Large Language Models, Diffusion Models, Python, SocketIO, Unity.

CONTENIDO

RESUMEN.....	3
Palabras clave	3
ÍNDICE DE TABLAS	6
ÍNDICE DE FIGURAS	7
1. INTRODUCCIÓN.....	8
1.1. Motivación	8
1.2. Trabajos relacionados.....	9
1.3. Objetivos	9
2. METODOLOGÍA, LENGUAJE Y TECNOLOGÍAS	10
2.1. Desarrollo de AIM4D	10
2.1.1. Interfaz	10
2.1.2. Formato de redes	10
2.2. Modo servidor	12
2.2.1. Servidor	12
2.2.2. Mensajes	12
2.3. Demo en Unity.....	12
2.3.1. Desarrollo	12
2.3.2. Interacción.....	13
3. DESARROLLO.....	14
3.1. Arquitectura de AIM4D	14
3.1.1. Estructura de componentes del programa.....	14
3.1.2. Integración modular de redes neuronales	14
3.1.3. Gestión de redes con ficheros .aim4d	15
3.2. Interacción de las redes neuronales	16
3.2.1. Proceso de carga y utilización de redes neuronales en GPU.....	16
3.2.2. Interfaz nativa con tkinter y pyrender.....	16
3.3. Diseño de la interfaz de AIM4D	17
3.3.1. Menú principal	17
3.3.2. Gestor de modelos	18
3.3.3. Modo servidor	20
3.3.4. Modo usuario	22
3.3.5. Mapa de navegación	23

3.3.6. Aviso de errores y advertencias.....	23
3.4. Desarrollo del modo servidor.....	24
3.4.1. Funcionalidades específicas del servidor	24
3.4.2. Mensajes de Entrada.....	25
3.4.3. Mensajes de Salida	25
3.5. Desarrollo de la demo	25
3.5.1. Funcionalidades de la demo en Unity	25
3.5.2. Interfaz Gráfica de la demo en Unity.....	26
3.5.3 Lógica de la Aplicación.....	28
3.5.4. Detalles Adicionales	29
3.5.5. Integración con AIM4D	29
3.5.6. Ejemplo de proceso de petición y respuesta	30
4. ANÁLISIS DE RESULTADOS	32
4.1. Exposición cualitativa de resultados.	32
4.2. Análisis del flujo de datos	35
4.3. Análisis de distintos modos de ejecución	36
5. CONCLUSIONES	38
5.1. Conclusiones sobre el desarrollo y uso de AIM4D	38
5.2. Éxitos, limitaciones y aprendizajes del proyecto	38
5.3. Propuestas de mejoras y futuras líneas de trabajo	38
REFERENCIAS.....	40
ANEXOS	42
Anexo A: Código completo del proyecto	42

ÍNDICE DE TABLAS

Tabla 1: Tiempos de ejecución seccionados	36
Tabla 2: Tiempos medios de ejecución con diferentes configuraciones	37
Tabla 3: Tiempos de carga de los modelos.....	37

ÍNDICE DE FIGURAS

Ilustración 1: Boceto del ecosistema planteado por AIM4D	8
Ilustración 2: Estructura de componentes de AIM4D	14
Ilustración 3: Código de la interfaz model_interface.py	15
Ilustración 4: Ejemplo de fichero .aim4d	15
Ilustración 5: Renderizado de un avatar con Pyrender	17
Ilustración 6: Pantalla de menú principal de AIM4D	18
Ilustración 7: Pantalla de gestor de modelos sin modelos cargados	19
Ilustración 8: Pantalla de gestor de modelos y explorador de archivos	19
Ilustración 9: Pantalla de gestor de modelos con dos redes cargadas	20
Ilustración 10: Pantalla de modo servidor.....	21
Ilustración 11: Pantalla de modo usuario con dos modelos cargados.....	22
Ilustración 12: Mapa de navegación de AIM4D	23
Ilustración 13: Ventana emergente de error	24
Ilustración 14: Elementos de la escena del cliente de Unity	26
Ilustración 15: Pantalla de configuración del servidor de la demo en Unity	27
Ilustración 16: Pantalla principal con dos peticiones de la demo en Unity	27
Ilustración 17: Pantalla de carga de la demo en Unity.....	28
Ilustración 18: Resultado tras las inferencias de la demo en Unity	28
Ilustración 19: Mensaje de petición de ejemplo	30
Ilustración 20: Mensaje de respuesta de ejemplo	31
Ilustración 21: Mensaje de petición erróneo	31
Ilustración 22: Mensaje de respuesta a petición errónea	31
Ilustración 23: Resultado de la petición P01	32
Ilustración 24: Resultado de la petición P02	33
Ilustración 25: Resultado de la petición P03	33
Ilustración 26: Resultado de la petición P04	34
Ilustración 27: Resultado de la petición P05	34
Ilustración 28: Resultado de la petición P06	35

1. INTRODUCCIÓN

1.1. Motivación

El uso de inteligencias artificiales (IAs) dentro del proceso de creación de contenido digital está ganando importancia, lo que resalta la necesidad de herramientas de gestión que hagan su uso más intuitivo y accesible. Este gestor es completamente modular, lo que significa que su arquitectura permite la integración y manejo de distintos componentes de manera independiente y flexible. AIM4D permite utilizar varias redes neuronales simultáneamente, ofreciendo a los usuarios la capacidad de hacer peticiones a una o varias redes al mismo tiempo, lo que maximiza la eficiencia y versatilidad en el desarrollo de contenido digital.

AIM4D se plantea como un ecosistema cuyo núcleo principal reside en un programa alojado en una máquina con altas capacidades de uso de GPU, donde se pueden seleccionar y cargar las redes neuronales deseadas. Una vez configuradas estas redes (en este proyecto, se han explorado la red de "Forma del cuerpo" y la de "Textura"), el gestor puede recibir peticiones locales a través de una interfaz ("Modo usuario") o peticiones externas de clientes remotos ("Modo servidor"). Esto permite que dispositivos con capacidades computacionales limitadas puedan beneficiarse de las potentes capacidades de las redes neuronales sin ningún problema. En este proyecto, se ha demostrado este uso mediante una demo en Unity^[1], que actúa como cliente remoto para enviar peticiones y recibir resultados en tiempo real.

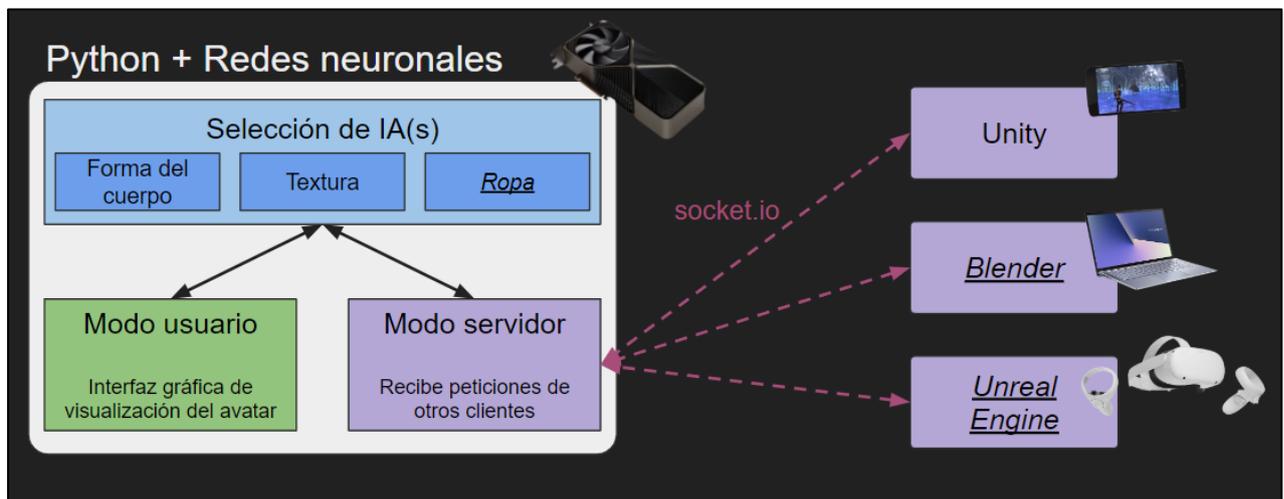


Ilustración 1: Boceto del ecosistema planteado por AIM4D

1.2. Trabajos relacionados

Actualmente existen gestores de redes neuronales entre los que destacan:

- LMStudio^[2], enfocado en el uso de redes neuronales generativas de texto. Ofrece un explorador de redes conectado a internet que permite descargar modelos de forma cómoda e intuitiva. Permite tanto el uso directo de las redes a través de un ‘chatbox’ como la configuración de un servidor que procese peticiones externas.
- Stable Diffusion Web UI^[3], que permite cargar, configurar y utilizar modelos de difusión para la creación de imágenes.

AIM4D se distingue de los gestores anteriormente mencionados por estar específicamente diseñado para el entorno del desarrollo de videojuegos y gráficos 3D, adaptando su interfaz y funcionalidades a dicha tarea. Además, al cumplir una función de intermediario entre el usuario y las redes, permite tener cargadas en memoria varias redes al mismo tiempo, siempre que el dispositivo lo permita, cumpliendo así con las peticiones de naturaleza multidisciplinar propias del desarrollo de videojuegos (generación de geometría, texturas, animaciones...).

1.3. Objetivos

Los objetivos del proyecto son:

1. Crear un programa modular enfocado en el uso de inteligencias artificiales (IAs) como herramientas en el desarrollo de videojuegos. La modularidad permitirá a los desarrolladores integrar y manejar diferentes componentes de manera flexible y eficiente, facilitando la actualización y personalización de las funcionalidades según las necesidades del proyecto.
2. Implementar un modo servidor que pueda recibir y procesar peticiones online. Este modo permitirá a los usuarios acceder a las funcionalidades del gestor de manera remota y colaborar en tiempo real, haciendo posible la integración de AIM4D en entornos de trabajo distribuidos y mejorando la accesibilidad a las herramientas de IA.
3. Integrar en el sistema la implementación de dos redes neuronales con tecnologías diferentes y demostrar su funcionamiento en peticiones a una de las redes o a ambas.
4. Desarrollar una demo que ejemplifique el uso de AIM4D por parte de un usuario medio. La demo demostrará el control de la información de extremo a extremo, mostrando cómo un usuario puede interactuar con el sistema de manera intuitiva y eficiente.

2. METODOLOGÍA, LENGUAJE Y TECNOLOGÍAS

2.1. Desarrollo de AIM4D

El lenguaje elegido para desarrollar AIM4D ha sido Python, debido a su frecuente uso en el ámbito del Deep Learning y a la vasta cantidad de paquetes y librerías desarrolladas por la comunidad, lo que permite la integración de modelos muy variados. Python ofrece opciones robustas para el desarrollo de la interfaz gráfica mediante Tkinter^[4], una biblioteca estándar de Python para crear interfaces gráficas de usuario (GUIs), y para el renderizado de modelos 3D mediante Pyrender^[5], cumpliendo así con todos los requisitos necesarios para el desarrollo del programa.

La sintaxis clara y concisa de Python facilita el desarrollo rápido y eficiente, permitiendo a los desarrolladores centrarse en la lógica de la red en lugar de en detalles complejos del lenguaje. Además, Python cuenta con una amplia gama de bibliotecas y frameworks para machine learning y redes neuronales, como TensorFlow^[6] y PyTorch^[7], que son esenciales para el proyecto. Estas herramientas proporcionan soporte avanzado para la construcción, entrenamiento y despliegue de modelos de inteligencia artificial.

La compatibilidad multiplataforma de Python garantiza que AIM4D pueda ejecutarse en diferentes sistemas operativos sin necesidad de modificaciones significativas. Por último, la gran comunidad de desarrolladores de Python asegura un robusto soporte y abundantes recursos para resolver problemas y mejorar el desarrollo. Todo esto hace de Python una elección óptima para el desarrollo de AIM4D.

2.1.1. Interfaz

Se ha utilizado la librería Tkinter para el desarrollo de la interfaz gráfica de AIM4D debido a su estrecha integración con Python, simplicidad y facilidad de uso, lo que permite un desarrollo rápido y eficiente. Tkinter, como biblioteca estándar de Python, ofrece una compatibilidad multiplataforma que asegura que las aplicaciones desarrolladas sean funcionales y adaptables en diversos sistemas operativos. Su ligereza y flexibilidad permiten la creación de interfaces gráficas intuitivas y eficientes sin un consumo excesivo de recursos.

Además, Tkinter cuenta con una amplia documentación y una comunidad activa, lo que facilita la resolución de problemas y asegura un mantenimiento a largo plazo. Estos factores combinados hacen de Tkinter una elección ideal para el desarrollo de la interfaz gráfica de AIM4D, garantizando un entorno de desarrollo robusto y sostenible.

2.1.2. Formato de redes

En el proyecto se han integrado al sistema dos redes neuronales diferentes: SMPLLM^[8] y SMPLitex^[9].

SMPLLM es una red basada en el modelo de lenguaje grande (LLM) LLaMA-3^[10], el cual ha sido sometido a un proceso de fine-tuning específico para generar avatares tridimensionales humanos a partir de descripciones en lenguaje natural. El modelo

LLaMA-3 es conocido por su capacidad para comprender y generar texto de alta calidad, y su adaptación para la generación de avatares 3D aprovecha estas capacidades para interpretar descripciones detalladas y variadas de características físicas.

Los avatares generados por SMPLLM están basados en el modelo SMPL-X^[11] (Skinned Multi-Person Linear Model - eXpressive), que es un modelo paramétrico ampliamente utilizado en la creación de avatares humanos realistas. La salida de la red SMPLLM consiste en los parámetros de forma del modelo SMPL-X, representados por diez números decimales. Estos parámetros definen aspectos específicos de la morfología del avatar, como altura, proporciones y otras características físicas, permitiendo la creación de avatares detallados y personalizados.

SMPLitex, por su parte, es una red basada en la tecnología de Diffusers^[12], una técnica avanzada de generación de imágenes que se utiliza para crear texturas detalladas y realistas mediante la difusión de ruido en un proceso iterativo que refina la imagen. La función principal de SMPLitex es generar mapas de textura para los modelos SMPL basados en descripciones en lenguaje natural.

Al recibir una descripción verbal, SMPLitex genera una imagen que corresponde al mapa de textura del modelo SMPL. Este mapa de textura puede incluir detalles como el color de la piel, ropa, tatuajes, cicatrices, y otros elementos visuales que contribuyen a la apariencia del avatar. La capacidad de generar texturas a partir de descripciones permite una personalización extensa y una representación visual coherente con las especificaciones del usuario.

La integración de estas redes en AIM4D se realiza a través de un sistema modular que permite cargar y gestionar múltiples redes neuronales de manera eficiente. Los usuarios pueden seleccionar las redes que desean, y el sistema gestiona la carga y ejecución de estas redes, asegurando que los avatares y texturas se generen de acuerdo con las descripciones proporcionadas.

Para integrar redes neuronales de distintas tecnologías en el proyecto, no se ha utilizado una librería específica, sino que se ha desarrollado un sistema basado en herencia. Este enfoque permite definir el comportamiento de cada red al ser cargada en memoria o al realizar inferencias. Los scripts hijos, específicos para cada red neuronal, contienen las librerías necesarias, como Diffusers o Transformers^[13], que son esenciales para el funcionamiento de los modelos de IA.

Para referenciar estos scripts en tiempo de ejecución, se han creado ficheros personalizados con la extensión .aim4d. Estos ficheros contienen información relevante que permite al gestor principal integrar y gestionar cada red neuronal de manera eficiente. La estructura modular y extensible de este sistema asegura que AIM4D pueda adaptarse a diferentes tipos de redes neuronales y tecnologías, proporcionando flexibilidad y escalabilidad al proyecto.

2.2. Modo servidor

2.2.1. Servidor

La tecnología utilizada para implementar la funcionalidad del servidor es SocketIO^[14], una biblioteca que permite manejar comunicaciones en tiempo real. SocketIO facilita la integración con Python y otros lenguajes, como C# para Unity, permitiendo la transmisión eficiente de datos entre el servidor y los clientes.

SocketIO soporta WebSockets^[15], un protocolo que proporciona canales de comunicación full-duplex sobre una sola conexión TCP, garantizando una comunicación rápida y confiable. Esto es fundamental para asegurar que las peticiones y respuestas entre el cliente y el servidor sean manejadas de manera eficiente, minimizando la latencia y mejorando la experiencia del usuario.

Además, la amplia documentación y la comunidad activa de SocketIO facilitan su implementación y la resolución de problemas. Estos recursos proporcionan un soporte robusto y abundante para los desarrolladores, asegurando que cualquier inconveniente técnico pueda ser abordado rápidamente y que el sistema pueda mantenerse y escalarse de manera efectiva.

2.2.2. Mensajes

La comunicación que se establece entre los clientes y el servidor se basa en el uso de mensajes estructurados. Cada mensaje puede contener varias peticiones, y cada petición incluye una cabecera y un cuerpo del mensaje. La cabecera indica al sistema qué red neuronal se pretende utilizar, mientras que el cuerpo del mensaje contiene el prompt para la red neuronal seleccionada.

Una vez que el servidor recibe estas peticiones, procesa cada una utilizando la red neuronal especificada en la cabecera. Tras realizar la inferencia, el servidor emite una respuesta que contiene una cantidad igual de respuestas, cada una con su correspondiente cabecera. Esto asegura que el cliente reciba los resultados de cada inferencia realizada por las redes neuronales, manteniendo una comunicación clara y estructurada entre el cliente y el servidor.

2.3. Demo en Unity

2.3.1. Desarrollo

Como representación del uso por parte de un usuario promedio del gestor AIM4D, se ha desarrollado una demo en Unity. Esta demo permite conectarse al servidor y realizar peticiones desde cualquier dispositivo, independientemente del sistema operativo o de las capacidades computacionales del mismo. El proceso comienza cuando el usuario envía un mensaje desde la aplicación en Unity; este mensaje es procesado por el servidor, que realiza la inferencia con las redes neuronales correspondientes y luego devuelve la respuesta.

El usuario puede ver el resultado directamente en la misma aplicación de Unity, lo que proporciona una experiencia fluida y accesible. Esta demo no solo muestra la funcionalidad de AIM4D, sino que también destaca su capacidad para operar en una amplia gama de dispositivos y plataformas, demostrando su versatilidad y eficiencia en el uso de redes neuronales para el desarrollo de videojuegos y gráficos 3D.

2.3.2. Interacción

En la escena del proyecto de Unity se han integrado dos cuadros de texto donde se pueden introducir los prompts para dos redes neuronales distintas. La primera red neuronal está encargada de generar avatares 3D de cuerpos humanoides basados en la descripción proporcionada, utilizando el modelo SMPLLM. La segunda red neuronal genera la textura para dicho avatar 3D, también en función de una descripción verbal, utilizando el modelo SMPLitex.

El resultado de ambas redes se combina y se representa en un cuerpo 3D, utilizando la librería SMPL-X para Unity. Esto permite visualizar el avatar generado junto con su textura correspondiente ya aplicada, proporcionando una experiencia interactiva y visualmente cohesiva para el usuario.

3. DESARROLLO

3.1. Arquitectura de AIM4D

3.1.1. Estructura de componentes del programa

El programa principal de AIM4D está contenido en el script `aim4d_main.py`. Este programa gestiona la lógica principal de la interfaz y la navegación a través de los distintos menús utilizando Tkinter. Además, `aim4d_main.py` contiene el “modo usuario”, que permite realizar peticiones a las redes neuronales directamente desde la interfaz nativa de AIM4D.

Para ejecutar el servidor asíncrono `SocketIO`, que escucha y maneja las peticiones de los clientes, se utiliza el script `aim4d_server.py`. Este script recibe las peticiones de los clientes, las procesa y luego las traspasa a `aim4d_main.py` para que las redes neuronales realicen las inferencias necesarias. De esta manera, `aim4d_server.py` y `aim4d_main.py` trabajan conjuntamente para ofrecer tanto una interfaz nativa de usuario como la capacidad de procesar peticiones remotas, asegurando una experiencia fluida para el usuario.

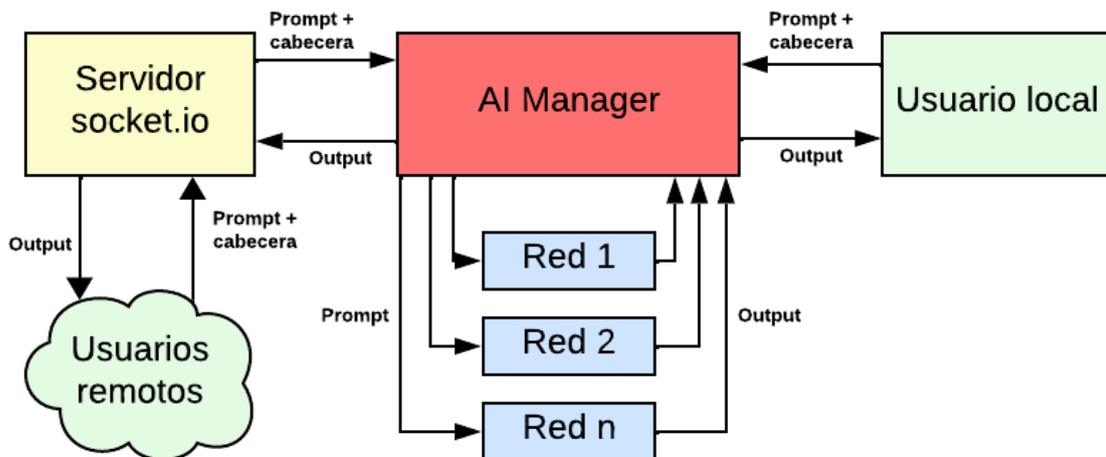


Ilustración 2: Estructura de componentes de AIM4D

3.1.2. Integración modular de redes neuronales

Cuando una petición es recibida, ya sea a través del servidor o de la interfaz nativa, el programa principal delega la ejecución específica de cada red a su correspondiente script, que hereda de `model_interface.py`. Esta interfaz define los métodos que deben implementar todos los scripts hijos para gestionar cada red de forma satisfactoria.

```

from abc import ABC, abstractmethod

class ModelInterface(ABC):
    @abstractmethod
    def load_model(self, model_info):
        pass

    @abstractmethod
    def run_model(self, input_data):
        pass

    @abstractmethod
    def remove_model(self):
        pass

    @abstractmethod
    def display_interface(self, root):
        pass
  
```

Ilustración 3: Código de la interfaz `model_interface.py`

Como ejemplo, se proporcionan los scripts `smpllm_model.py` y `smplitex_model.py`, los cuales heredan de la interfaz `interface_model.py`. Estos scripts contienen la lógica necesaria para cargar, procesar y realizar inferencias con las redes neuronales específicas: `smpllm_model.py` está encargado de generar avatares 3D de cuerpos humanoides basados en descripciones, mientras que `smplitex_model.py` se encarga de generar las texturas para dichos avatares, también basándose en descripciones verbales.

Este enfoque basado en herencia y definición de interfaces asegura que todas las redes integradas en AIM4D sigan un marco común de funcionamiento, facilitando la extensibilidad y mantenimiento del sistema. Cada red puede ser manejada de manera consistente, garantizando que las peticiones sean procesadas de manera individualizada según los requisitos de cada modelo.

3.1.3. Gestión de redes con ficheros `.aim4d`

Los ficheros `.aim4d` son archivos de texto que contienen pares clave-valor con la información necesaria para el correcto funcionamiento de las redes neuronales implicadas en AIM4D. Estos ficheros deben incluir, como mínimo, el nombre de la red, el identificador de la cabecera de la red (para que los mensajes puedan referenciarla), y una referencia al script de tipo `aim4d_model` que controla su lógica.

La estructura de un fichero `.aim4d` podría ser la siguiente:

```

model_name = "SMPLLM"
model_description = "Avatar generation given human description"
base_model_id = "meta-llama/Meta-Llama-3-8B"
model_header = "smpllm"
aim4d_logic = "smpllm_model.SMPLLMModel"
  
```

Ilustración 4: Ejemplo de fichero `.aim4d`

Estos archivos son leídos por el script ``aim4d_main.py``, que extrae la información y la almacena en diccionarios para su fácil acceso y gestión. Este enfoque permite que AIM4D gestione de manera dinámica y flexible múltiples redes neuronales, asegurando que cada red pueda ser referenciada y controlada adecuadamente durante la ejecución del programa. El uso de estos ficheros facilita la configuración y extensión del sistema, permitiendo agregar nuevas redes neuronales simplemente mediante la creación de nuevos archivos de configuración, sin necesidad de modificar el código principal.

3.2. Interacción de las redes neuronales

3.2.1. Proceso de carga y utilización de redes neuronales en GPU

Cuando se inicia el programa por primera vez, por defecto no hay ninguna red cargada. El usuario debe seleccionar qué red o redes desea utilizar, proporcionando el archivo ``aim4d`` correspondiente a través de un explorador de archivos.

Cuando se elige un archivo de esta forma, el programa inicia la red, la carga en la memoria de la GPU y la deja lista para recibir peticiones. Además, el programa calcula el porcentaje de uso de la GPU mediante el método `'get_gpu_memory_usage()'` de `'aim4d_main.py'` y muestra esta información al usuario, permitiéndole decidir si puede añadir nuevas redes en función de los recursos disponibles.

El proceso de arranque de cada red puede variar y debe estar especificado en el script del modelo, tal como lo exige la interfaz ``aim4d_model.py``. Esta interfaz asegura que cada modelo implementa los métodos necesarios para su inicialización, carga en memoria, y procesamiento de inferencias. Este enfoque garantiza que, independientemente de las diferencias entre redes neuronales, el programa principal puede manejarlas de manera uniforme.

3.2.2. Interfaz nativa con tkinter y pyrender

La interfaz nativa, reconocida en el programa como “modo usuario”, permite hacer peticiones a las redes cargadas directamente desde AIM4D, en contraste con el “modo servidor”, que espera peticiones externas de otros dispositivos conectados a la misma red. Esta interfaz es modular, de manera que los espacios de input y los resultados se muestran en función de las redes cargadas. Al igual que ocurre con la carga de los modelos en memoria, los métodos que componen la interfaz gráfica están especificados en la interfaz `'model_interface.py'` para asegurar que cualquier script que herede del mismo contemple el comportamiento de la red en cuanto a la interfaz gráfica.

La interfaz nativa admite el uso de las redes para la creación de gráficos 3D. Por ejemplo, cuando se solicita un avatar, este puede ser renderizado en una ventana gestionada por Pyrender, una biblioteca de renderizado 3D en Python. Esto permite a los usuarios visualizar los resultados de sus peticiones directamente dentro de la aplicación AIM4D, proporcionando una experiencia integrada y eficiente.

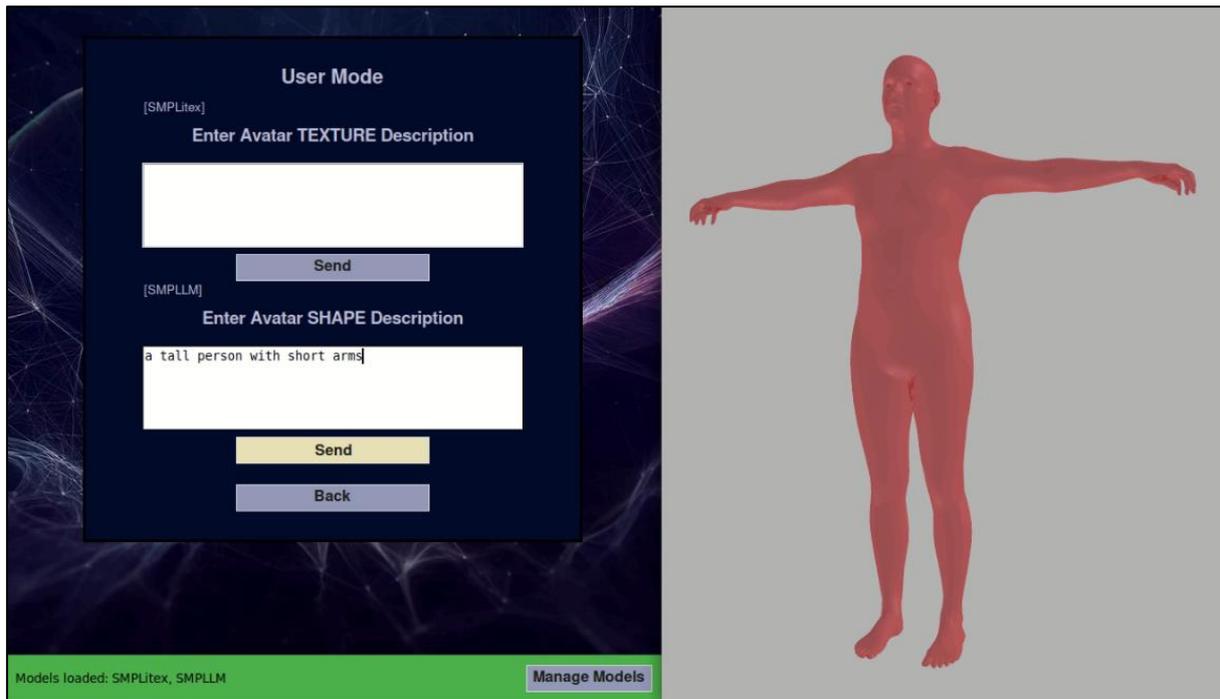


Ilustración 5: Renderizado de un avatar con Pyrender

La modularidad de la interfaz asegura que, independientemente de las redes cargadas, los usuarios puedan interactuar con ellas de manera coherente y adaptable. Esto facilita la personalización y expansión del sistema, permitiendo agregar nuevas funcionalidades y redes sin necesidad de reestructurar la interfaz principal.

3.3. Diseño de la interfaz de AIM4D

La interfaz de AIM4D se divide en cuatro escenas, siguiendo las funcionalidades mencionadas anteriormente.

- Menú principal.
- Gestor de modelos.
- Modo servidor.
- Modo usuario.

3.3.1. Menú principal

Esta escena presenta el nombre del programa y ofrece botones para acceder a los dos modos principales: Modo Servidor y Modo Usuario, así como un botón adicional para cerrar el programa. En la parte inferior de la pantalla se encuentra siempre visible una barra de estado que indica si hay algún modelo cargado. Esta barra se muestra en verde si hay modelos presentes y en rojo si no los hay. En caso de haber modelos cargados, se muestra su nombre. Esto permite al usuario entender de un vistazo el estado actual del sistema. A la derecha de la barra de estado, hay un botón “Manage Models” que facilita de manera intuitiva la adición o eliminación de modelos en el sistema.

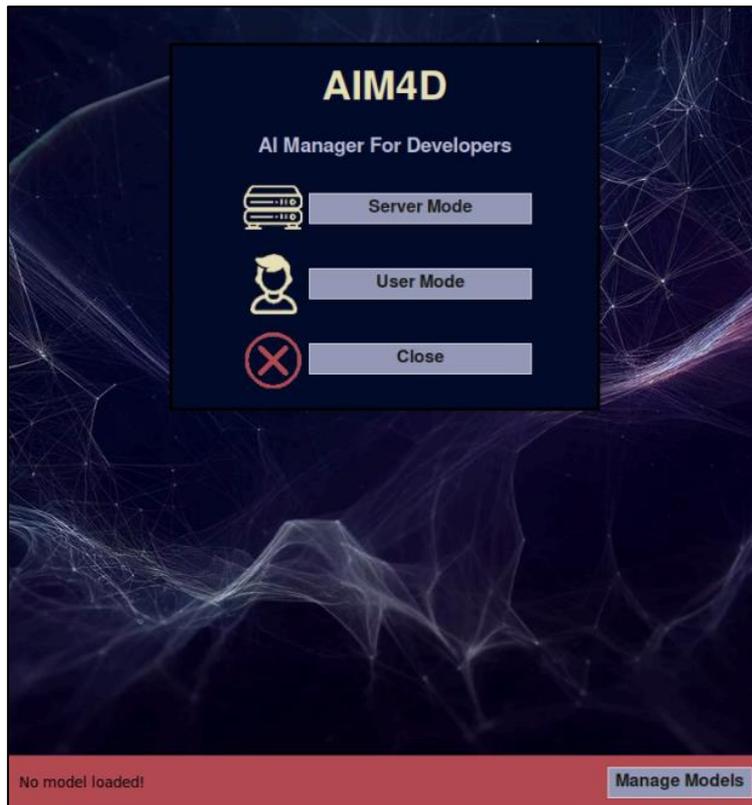


Ilustración 6: Pantalla de menú principal de AIM4D

3.3.2. Gestor de modelos

La interfaz del gestor de modelos muestra una lista de todos los modelos actualmente cargados en la GPU junto con sus descripciones y un botón con la opción de eliminarlo de la lista, o bien un mensaje en caso de que no haya ningún modelo cargado. Incluye un botón titulado “Add Model” que permite al usuario añadir nuevos modelos a la lista. Al hacer clic en este botón, se abre un explorador de archivos que busca ficheros con la extensión .aim4d en el sistema. Una vez seleccionado un fichero compatible, el gestor carga el modelo en memoria y muestra en una barra la cantidad de memoria de GPU utilizada y la cantidad disponible en el ordenador. Esto permite al usuario gestionar eficientemente los recursos disponibles.

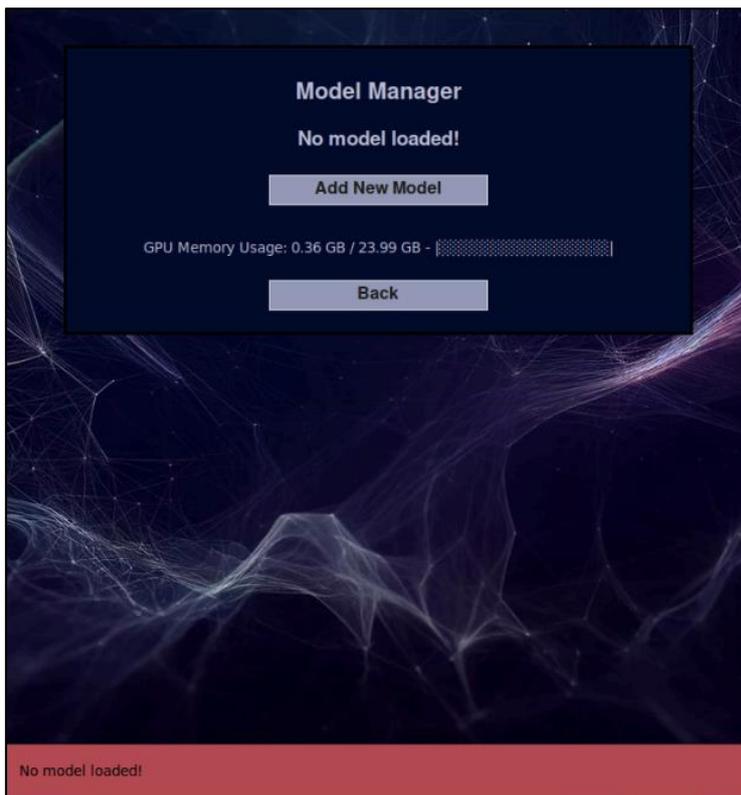


Ilustración 7: Pantalla de gestor de modelos sin modelos cargados

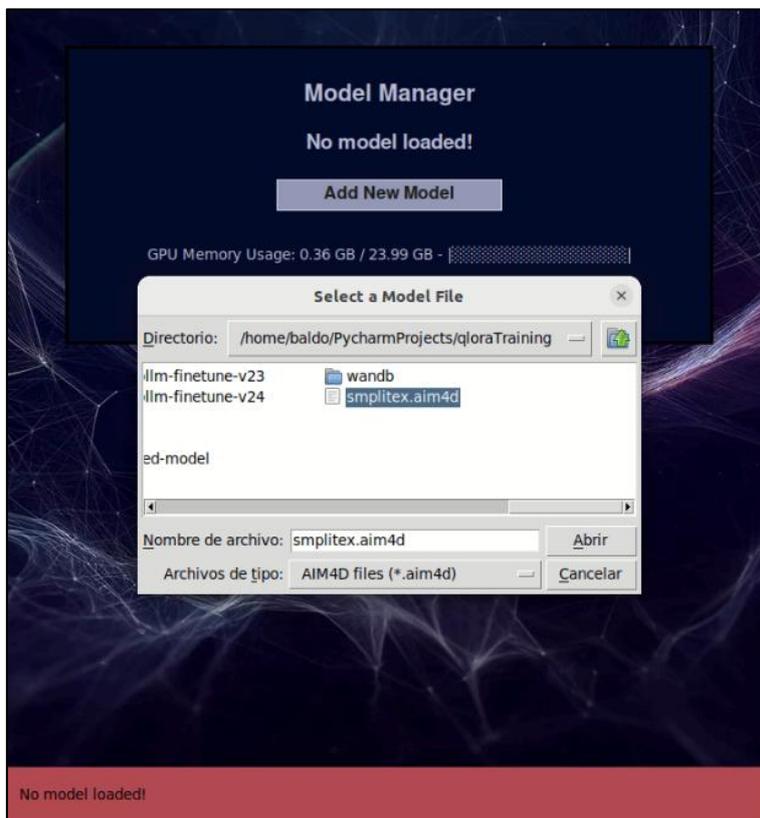


Ilustración 8: Pantalla de gestor de modelos y explorador de archivos

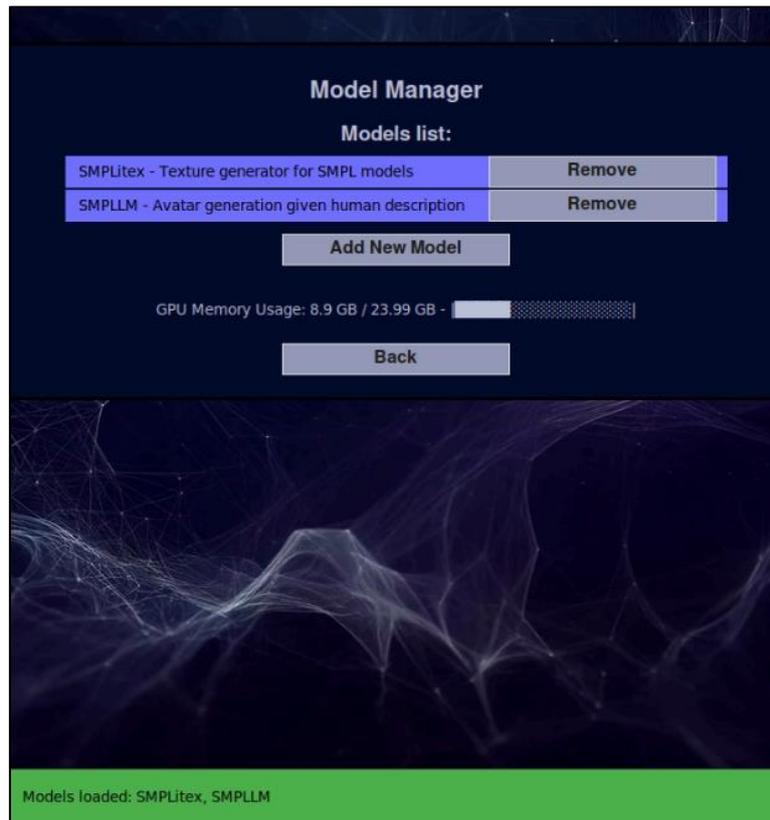


Ilustración 9: Pantalla de gestor de modelos con dos redes cargadas

3.3.3. Modo servidor

La interfaz del modo servidor ofrece una terminal sencilla donde se registran, con fecha y hora, todos los eventos relevantes, tales como la conexión de nuevos usuarios, mensajes de entrada y salida, y desconexión de usuarios. Los usuarios pueden abrir y cerrar el servidor utilizando los botones correspondientes. Además, desde esta escena, se puede acceder al Gestor de Modelos mediante el botón en la barra de estado en la parte inferior de la pantalla. Esto asegura una fácil gestión y monitoreo de las actividades del servidor.

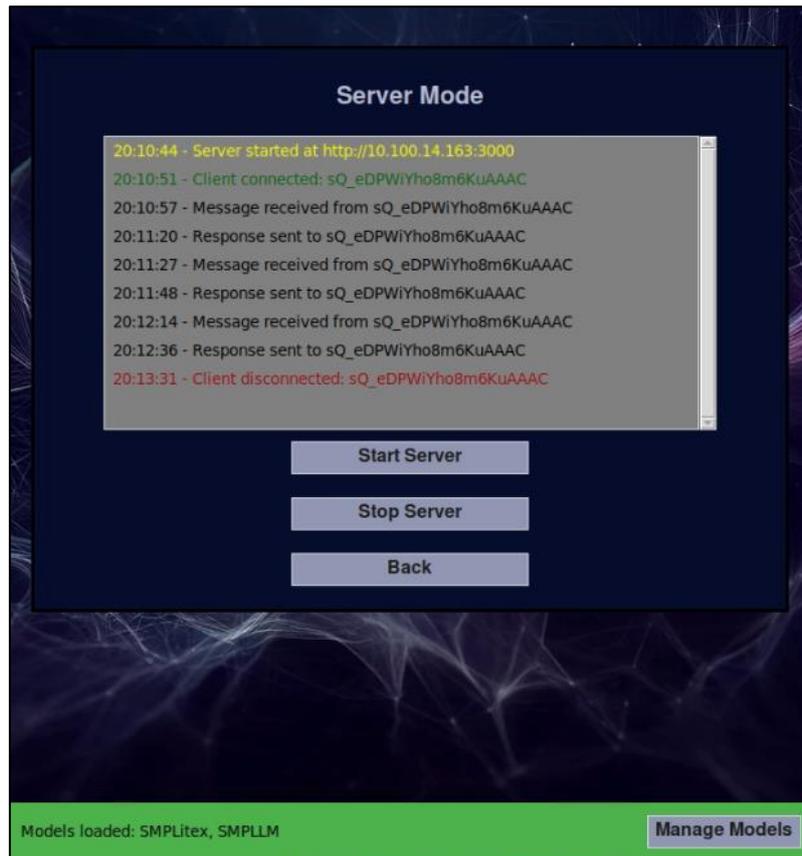


Ilustración 10: Pantalla de modo servidor

3.3.4. Modo usuario

El Modo Usuario presenta una interfaz dinámica que varía en función de los modelos cargados. Cada modelo añade sus propios elementos a la escena, proporcionando al usuario la opción de utilizar esa red directamente sin pasar por el servidor. Por ejemplo, las redes SMPLLM y SMPLitex ofrecen cuadros de texto para introducir las entradas a las redes y botones para enviar las solicitudes. Los resultados se muestran de manera específica según la red: en el caso de SMPLLM, se muestra el avatar generado, y en el caso de SMPLitex, se presenta el mapa de textura. Al igual que en el Modo Servidor, el estado actual de las redes cargadas se muestra en la parte inferior de la pantalla, y se puede acceder al Gestor de Modelos a través del botón correspondiente.

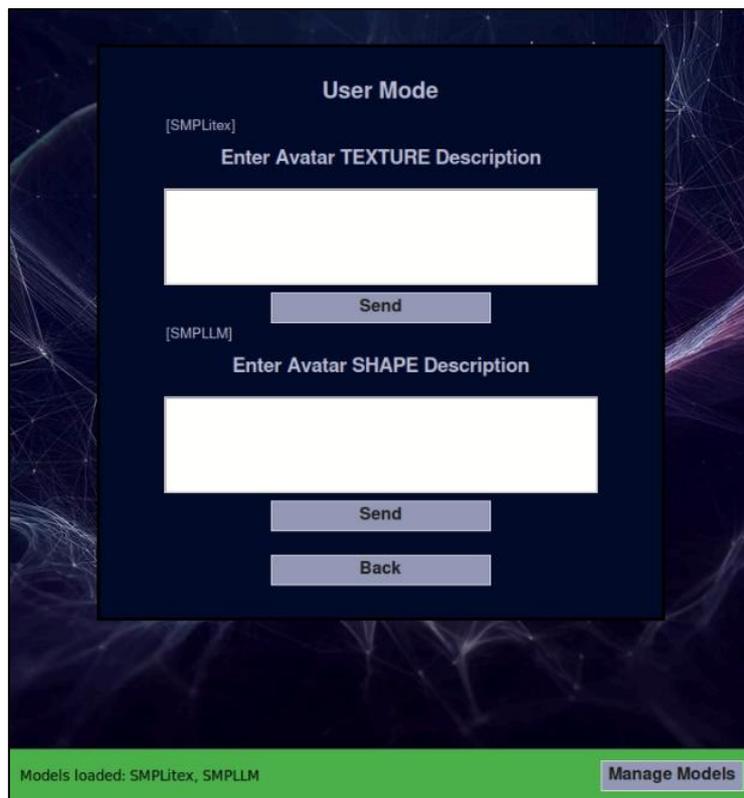


Ilustración 11: Pantalla de modo usuario con dos modelos cargados

3.3.5. Mapa de navegación

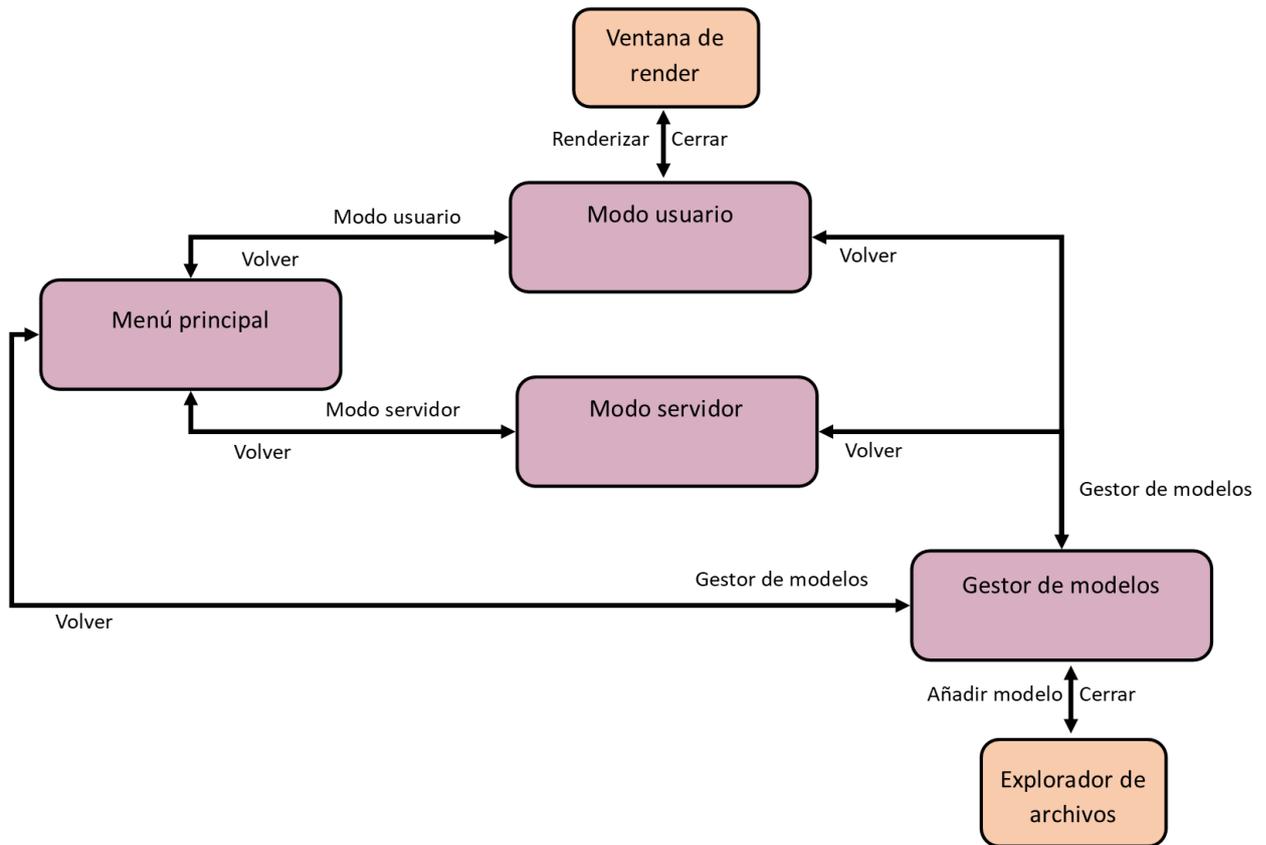


Ilustración 12: Mapa de navegación de AIM4D

3.3.6. Aviso de errores y advertencias

En caso de que se genere un error o una advertencia, AIM4D avisará al usuario utilizando el sistema de ventanas emergentes (pop-ups) de Tkinter. Estas ventanas ofrecen feedback al usuario y son importantes para representar de forma rápida cuál es el error. Por ejemplo, si el archivo .aim4d no contiene la información necesaria para cargar la red, se mostrará un mensaje de error detallado, permitiendo al usuario identificar y corregir el problema rápidamente. Este sistema de notificación asegura que los usuarios estén siempre informados de cualquier problema y puedan tomar las medidas necesarias para resolverlo.

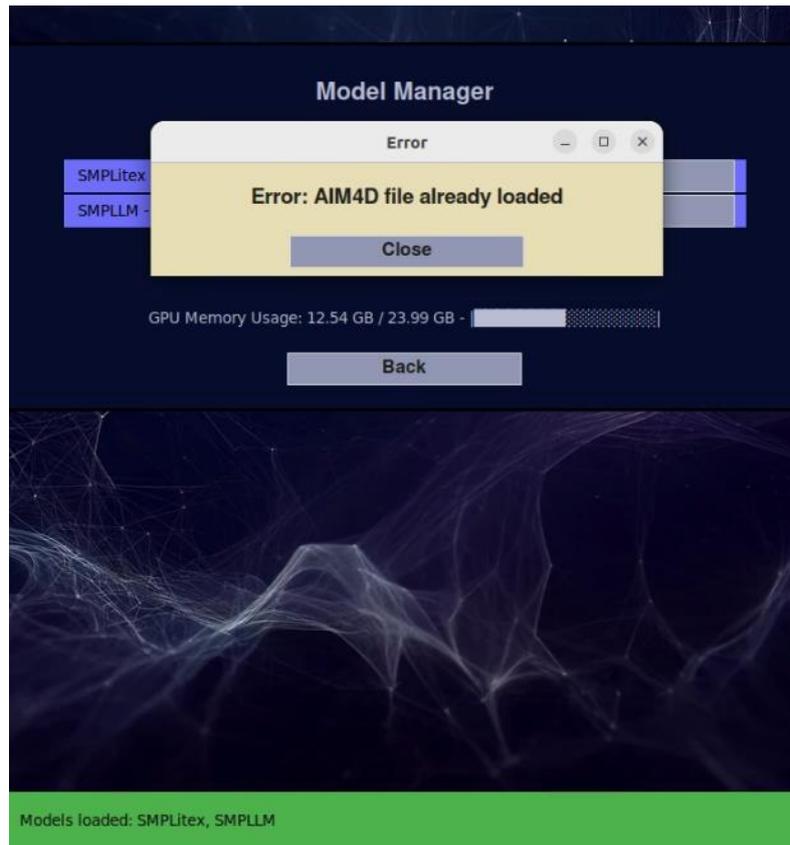


Ilustración 13: Ventana emergente de error

3.4. Desarrollo del modo servidor

3.4.1. Funcionalidades específicas del servidor

El servidor es gestionado por el script ``aim4d_server.py``, que arranca un servidor en la dirección IP local y, por defecto, en el puerto 3000. En este mismo script se definen los comportamientos asociados a varios eventos clave:

- Conexión de un usuario: Cuando un usuario se conecta, el servidor registra la conexión y muestra un mensaje de log con la hora en la interfaz de AIM4D.
- Desconexión de un usuario: De manera similar, cuando un usuario se desconecta, el servidor registra la desconexión y actualiza el log con la fecha y hora en la interfaz.
- Recepción del mensaje de un usuario: Al recibir un mensaje de un usuario, el servidor procesa la petición, delega la ejecución a la red neuronal correspondiente y envía la respuesta de vuelta al usuario. Cada recepción de mensaje también se registra con un log detallado en la interfaz de AIM4D.

Estos comportamientos aseguran una gestión eficiente y transparente de las conexiones y peticiones de los usuarios. El servidor muestra un log con fecha y hora en la interfaz de AIM4D ante cualquier evento, lo que facilita el uso y la depuración del sistema. Esto

permite a los desarrolladores y usuarios monitorear y diagnosticar el funcionamiento del servidor en tiempo real, mejorando la capacidad de respuesta ante posibles problemas.

Los mensajes que el servidor gestiona están en formato JSON y existen dos tipos: mensajes de entrada y mensajes de salida.

3.4.2. Mensajes de Entrada

Los mensajes de entrada son enviados por un usuario remoto y contienen los prompts para las redes neuronales que se desea utilizar. Cada mensaje está compuesto por una o varias peticiones, y cada petición contiene:

- Cabecera (string): Texto que identifica la red neuronal específica que se está utilizando. En el caso de la demo, los valores pueden ser “smpllm” o “smplitex”.
- Prompt (string): Texto de entrada para la red neuronal correspondiente.

3.4.3. Mensajes de Salida

Los mensajes de salida son creados por el servidor y tienen una estructura similar a los mensajes de entrada. Contienen una o varias respuestas, y cada respuesta incluye:

- Cabecera (string): Texto que identifica la red neuronal utilizada. En el caso de la demo, los valores pueden ser “smpllm” o “smplitex”.
- Output (string): Texto que representa la salida de la red neuronal. Puede ser:
 - Texto plano.
 - Un conjunto de números, como en el caso de la salida del modelo SMPLLM.
 - Representación de una imagen en formato base64, como en el caso de la salida del modelo SMPLitex.
 - En caso de que haya habido un error durante la inferencia, el output tendrá el valor “MODEL_ERROR” seguido del mensaje de error específico.

Este esquema de mensajes asegura una comunicación clara y estructurada entre el cliente y el servidor, y la utilización de formato JSON facilita la serialización y deserialización de datos, garantizando compatibilidad y facilidad de uso en diferentes entornos y plataformas. Para explotar esta compatibilidad se ha desarrollado un sencillo programa de demostración en un entorno totalmente distinto (Unity, programado en C#) donde se probarán las funcionalidades que un usuario podría explotar de forma remota desde su dispositivo.

3.5. Desarrollo de la demo

3.5.1. Funcionalidades de la demo en Unity

La demo desarrollada en Unity contiene los siguientes elementos:

- Un avatar 3D importado de la extensión SMPL-X para Unity donde se aplicarán los cambios solicitados a la red.

- Una sencilla escena compuesta por un fondo gris y una plataforma verde para situar el avatar.
- Elementos de referencia para comparar el tamaño del avatar con objetos del mundo real, como una silla o una tabla de alturas, permitiendo verificar visualmente la altura del avatar.



Ilustración 14: Elementos de la escena del cliente de Unity

3.5.2. Interfaz Gráfica de la demo en Unity

La aplicación contiene:

- Un menú de configuración del servidor donde se puede escribir la IP y el puerto del servidor con el que se quiere contactar, pudiendo estar alojado en el mismo dispositivo (localhost) o en cualquier otro dispositivo con visibilidad mutua en la red.
- Cuadros de texto: Uno para el prompt de solicitud para la red basada en SMPLLM y otro para la red basada en SMPLitex.
- Botones para enviar ambas descripciones o alguna de ellas por separado.
- Un cuadro de información donde se muestran los parámetros de forma (betas) del avatar SMPL-X de la escena, así como un botón para copiar esta información al portapapeles.
- Una pantalla de carga para indicar que la red está procesando la petición.

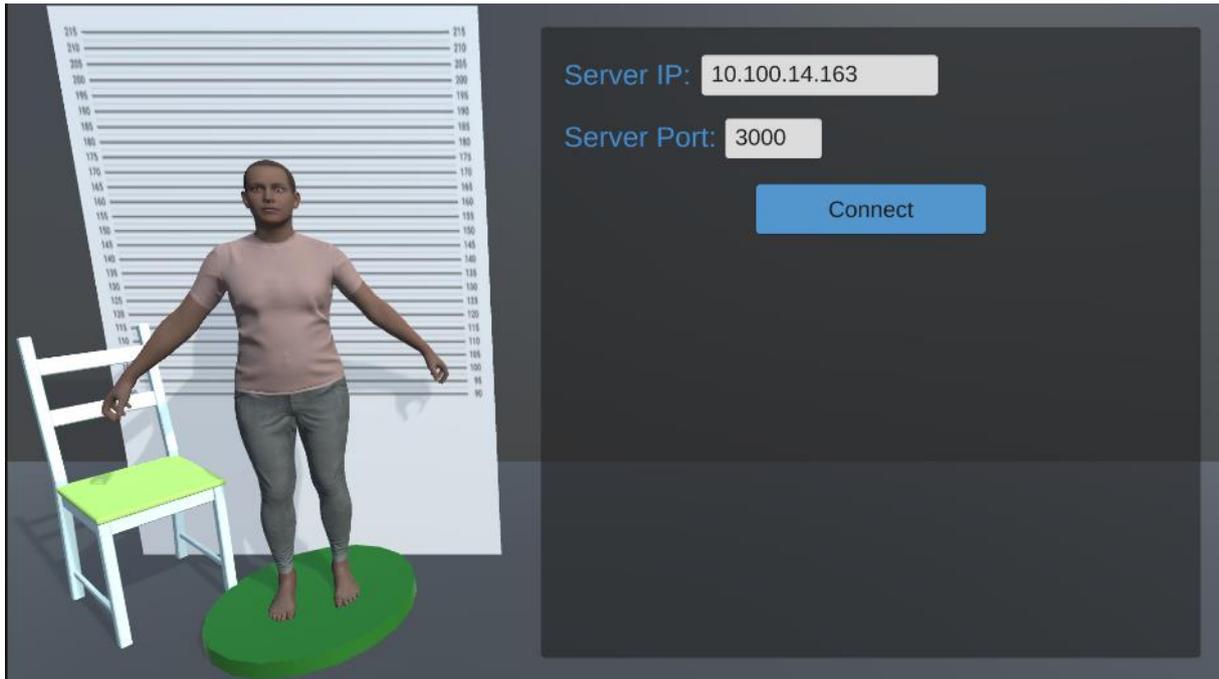


Ilustración 15: Pantalla de configuración del servidor de la demo en Unity

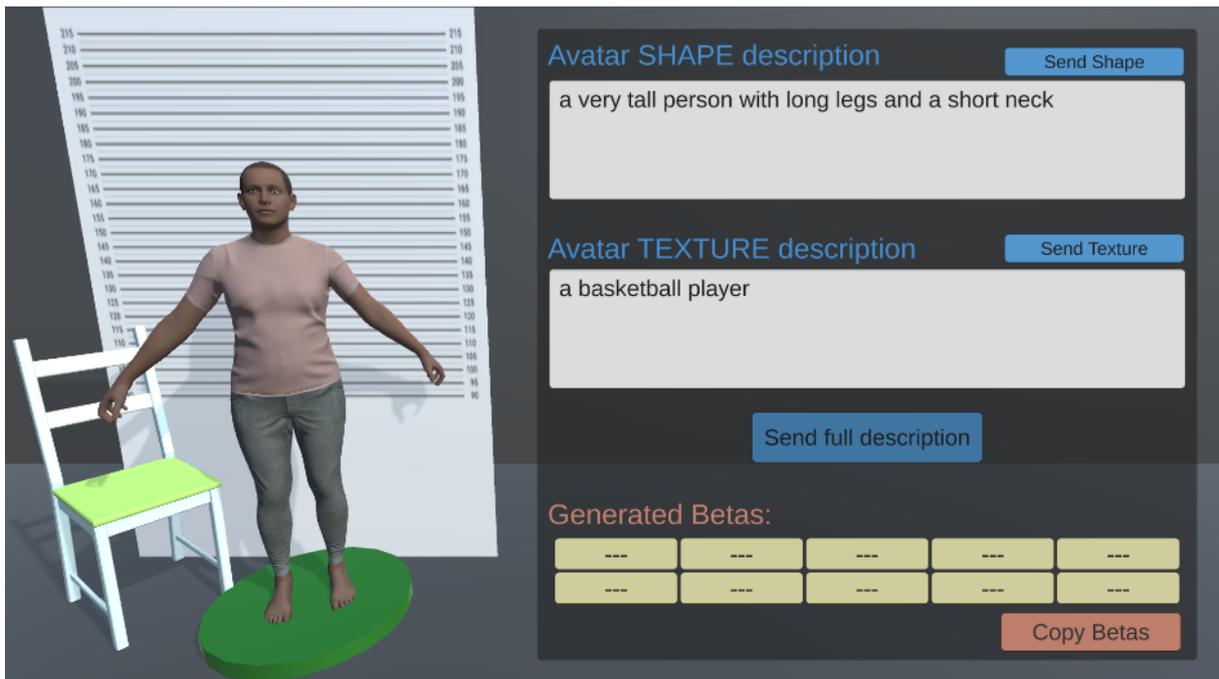


Ilustración 16: Pantalla principal con dos peticiones de la demo en Unity

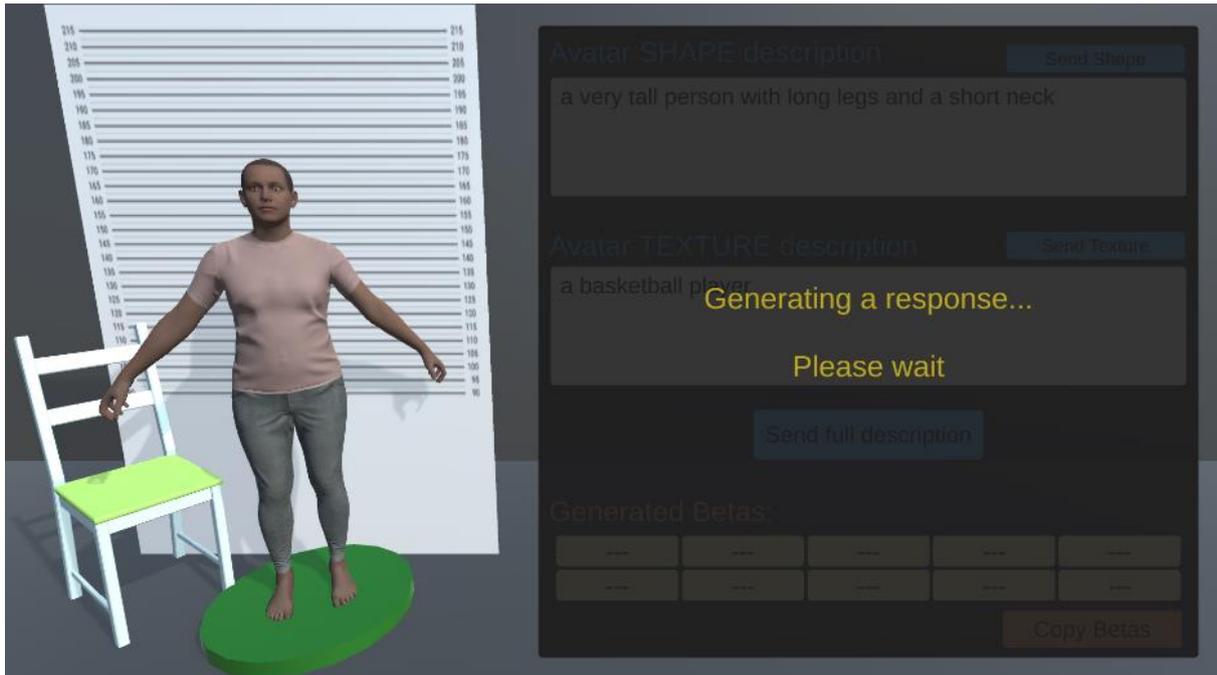


Ilustración 17: Pantalla de carga de la demo en Unity



Ilustración 18: Resultado tras las inferencias de la demo en Unity

3.5.3 Lógica de la Aplicación

La lógica de la aplicación está organizada para garantizar una experiencia de usuario fluida. El 'GameManager.cs' se encarga de aplicar los cambios solicitados en la escena tales como la actualización de la forma del avatar con el método 'SetBetaShapes()' de SMPL-X y la actualización de la textura del avatar, guardando la imagen en un formato válido para Unity en el método 'SaveImage()'.

El script 'SocketIOCustomClient.cs' gestiona la comunicación con el servidor SocketIO, indicando la lógica de los eventos de conexión y recepción de mensaje (métodos 'socket.OnConnected()' y 'socket.On("reply")'), así como ofrecer métodos públicos para ser llamados cuando se quiere enviar un mensaje desde el script principal.

3.5.4. Detalles Adicionales

Se añaden implementaciones adicionales para facilitar el uso de la aplicación tales como girar el avatar usando el ratón (método 'TurnModel()' de 'GameManager.cs' y la posibilidad de los usuarios de copiar la información de los parámetros de forma del modelo al portapapeles usando el botón inferior que llama al método 'CopyBetasButton()' de 'GameManager.cs'.

La interfaz gráfica permite una configuración flexible del servidor y facilita la entrada de prompts para dos redes neuronales distintas. Los usuarios pueden enviar descripciones para la generación de avatares y texturas de manera conjunta o separada.

3.5.5. Integración con AIM4D

Para integrar la aplicación con el ecosistema de AIM4D, basta con conectarse al servidor usando un cliente SocketIO y enviar mensajes utilizando el formato adecuado. Estos mensajes se construyen en el script 'SocketIOClient.cs', combinando las cabeceras predeterminadas con los prompts que se encuentran en los cuadros de texto, y dándoles formato utilizando JSON.

Conexión al Servidor:

Utilizando el cliente SocketIO en SocketIOClient.cs, la aplicación se conecta al servidor especificado en el menú de configuración. El usuario puede ingresar la IP y el puerto del servidor AIM4D.

Construcción de Mensajes:

En SocketIOClient.cs, se construyen los mensajes de entrada combinando las cabeceras predeterminadas (por ejemplo, "smpllm" para la red de avatares y "smplitex" para la red de texturas) con los prompts proporcionados por el usuario en los cuadros de texto.

Los mensajes se formatean en JSON para asegurar que sean interpretados correctamente por el servidor AIM4D. Un ejemplo de estructura de mensaje en JSON podría ser:

Envío y Recepción de Mensajes:

Una vez contruidos, los mensajes JSON se envían al servidor utilizando SocketIO.

El servidor AIM4D procesa las peticiones, realiza las inferencias necesarias y devuelve las respuestas en el mismo formato JSON.

SocketIOClient.cs gestiona la recepción de estas respuestas, actualizando la escena en Unity con los resultados obtenidos, como la generación de un avatar 3D o la aplicación de una textura.

3.5.6. Ejemplo de proceso de petición y respuesta

Los pasos en la utilización de un usuario medio del cliente de Unity para generar la forma de un avatar y su textura serían los siguientes:

1. Configuración y Conexión:
 - a. El usuario introduce la dirección IP y el puerto del servidor en los campos correspondientes del menú de configuración. En este caso, se introduce la dirección del servidor alojado en la misma red con IP local 10.100.14.163 y el puerto 3000 (puerto por defecto de AIM4D).
 - b. La aplicación establece una conexión con el servidor SocketIO utilizando los datos proporcionados.
2. Envío de Peticiones:
 - a. El usuario introduce los prompts para la generación del avatar y la textura en los cuadros de texto específicos.
 - b. Al pulsar el botón de envío, se construye un mensaje en formato JSON que contiene ambas peticiones. Por ejemplo:

```
{
  "requests": [
    {
      "header": "smp11m",
      "prompt": "A tall, muscular avatar."
    },
    {
      "header": "smp1itex",
      "prompt": "A firefighter."
    }
  ]
}
```

Ilustración 19: Mensaje de petición de ejemplo

- c. El mensaje se envía al servidor a través de la conexión SocketIO.
3. Pantalla de Carga: Mientras el servidor procesa las peticiones, la aplicación muestra una pantalla de carga, indicando al usuario que se está trabajando en la solicitud. Esta pantalla de carga ayuda a gestionar las expectativas del usuario durante el tiempo de procesamiento.
4. Recepción y Aplicación de Respuestas:
 - a. El servidor AIM4D procesa las peticiones, realiza las inferencias necesarias con las redes neuronales correspondientes, y envía una respuesta en formato JSON de vuelta a la aplicación.
 - b. La respuesta puede tener una estructura similar a la siguiente:

```
{
  "responses": [
    {
      "header": "smp11m",
```

```

"output": "[1.567, 1.806, 1.050, 0.953, 0.776, -0.237, 0.144, 1.202, 0.500,
0.852]"
},
{
"header": "smplitex",
"output": "3ifed9215jdi3nv94..." [Imagen en formato Base64]
}
]}
  
```

Ilustración 20: Mensaje de respuesta de ejemplo

- c. `SocketIOClient.cs` recibe esta respuesta, decodifica los resultados y aplica los cambios a la escena en Unity.
5. Actualización de la Escena:
- a. Los parámetros del avatar y la textura se aplican al modelo 3D en la escena.
 - b. El avatar 3D generado por la red SMPLLM se actualiza con los parámetros recibidos.
 - c. La textura generada por la red SMPLitex se aplica al avatar.
 - d. La pantalla de carga desaparece, mostrando al usuario los resultados finales en la escena.

Este flujo de trabajo asegura que el usuario pueda interactuar de manera intuitiva con la aplicación, realizando peticiones complejas de generación de contenido digital a través de una interfaz sencilla y obteniendo resultados visuales directamente en la escena de Unity.

En caso de que el sistema cuente con más redes, el usuario solo tendría que referenciarlas añadiendo su correspondiente cabecera. Si por error el usuario introduce una cabecera que no existe en el sistema, el servidor devolverá un mensaje de error explicativo, como en el siguiente ejemplo:

```

{
  "requests": [
    {
      "header": "ssmmp11",
      "prompt": "Example description."
    }
  ]
}
  
```

Ilustración 21: Mensaje de petición erróneo

```

{
  "responses": [
    {
      "header": "ssmmp11",
      "output": "MODEL_ERROR: Model header not found"
    }
  ]
}
  
```

Ilustración 22: Mensaje de respuesta a petición errónea

4. ANÁLISIS DE RESULTADOS

4.1. Exposición cualitativa de resultados.

A continuación, se exponen algunos ejemplos de resultados obtenidos tras el uso de la demo en Unity realizando peticiones al servidor de AIM4D.

Petición P01

Descripción de forma: “Una persona muy alta, con las piernas y los brazos muy largos”

Descripción de textura: “A LA Lakers basketball player”



Ilustración 23: Resultado de la petición P01

Petición P02

Descripción de forma: “Una persona muy ancha de hombros”

Descripción de textura: “A firefighter”



Ilustración 24: Resultado de la petición P02

Petición P03

Descripción de forma: “A tall person with very long legs”

Descripción de textura: “A Real Madrid soccer player”



Ilustración 25: Resultado de la petición P03

Petición P04

Descripción de forma: “A person with very thin hips and very broad shoulders”

Descripción de textura: “A security guard”



Ilustración 26: Resultado de la petición P04

Petición P05

Descripción de forma: “A big and strong person”

Descripción de textura: “A military soldier”



Ilustración 27: Resultado de la petición P05

Petición P06

Descripción de forma: “A very slim and super tall man”

Descripción de textura: “A zombie”



Ilustración 28: Resultado de la petición P06

4.2. Análisis del flujo de datos

Para estudiar los tiempos de ejecución se ha utilizado el temporizador ‘Stopwatch’ de la librería ‘System.Diagnostics’ en Unity y la librería ‘time’ de Python. Se ha medido el tiempo que transcurre entre cada paso del flujo de datos desde que se hace una petición hasta que se recibe una respuesta del sistema. Se distinguen los siguientes pasos:

1. Codificación del mensaje de petición.
2. Envío del mensaje al servidor.
3. Decodificación del mensaje de petición.
4. Inferencia de cada una de las redes.
5. Codificación del mensaje de respuesta.
6. Envío del mensaje al cliente.
7. Decodificación del mensaje de respuesta.

A continuación, se muestran los tiempos de cada paso para tres tipos de peticiones: forma de avatar, textura y ambos a la vez.

Petición realizada	Forma de avatar	Textura	Forma de avatar + textura
1. Codificación de petición	55 ms	56 ms	55 ms
2. Envío a servidor	122 ms	115 ms	110 ms
3. Decodificación de petición	13 ms	15 ms	16 ms
4. Inferencia de red(es)	6120 ms	13355 ms	6512 ms
			12220 ms
5. Codificación de respuesta	54 ms	61 ms	81 ms
6. Envío a cliente	109 ms	121 ms	118 ms
7. Decodificación de respuesta	22 ms	88 ms	78 ms
Tiempo total	6,495 s	13,811 s	19,190 s

Tabla 1: Tiempos de ejecución seccionados

Tras la medición de resultados, se comprueba que el grueso del tiempo de procesamiento consiste en el tiempo de inferencia de la red o redes neuronales ejecutándose. El tiempo de codificación, pese a ser despreciable con respecto al tiempo de inferencia, es mayor en el caso de las imágenes debido a que el sistema las decodifica en formato base64 utilizado en el envío y las guarda en memoria para ser visualizadas por Unity.

El sistema distribuido basado en el envío de peticiones a un servidor resulta conveniente, ya que los tiempos de envío son bajos y despreciables con respecto al tiempo de inferencia, el cual sería mayor en una máquina local o inalcanzable para muchos dispositivos con menor capacidad.

4.3. Análisis de distintos modos de ejecución

Para estudiar el impacto de tener varias redes cargadas al mismo tiempo en memoria de GPU sobre el rendimiento se han realizado 25 peticiones automáticamente para cada uno de los siguientes casos:

- Peticiones tanto de forma de avatar como de textura.
- Peticiones de forma de avatar con ambos modelos cargados en GPU.
- Peticiones de textura con ambos modelos cargados en GPU.
- Peticiones de forma de avatar con únicamente ese modelo cargado en GPU.
- Peticiones de textura con únicamente ese modelo cargado en GPU.

También se ha registrado el tiempo que tarda en preparar cada uno de los modelos cuando son listados y cargados en memoria GPU antes de su uso.

Los resultados son los siguientes:

Operación	Modelos cargados	Tiempo medio de ejecución
Forma de avatar + textura	SMPLLM y SMPLitex	22,115 s
Forma de avatar	SMPLLM y SMPLitex	6,617 s
Forma de avatar	SMPLLM	6,484 s
Textura	SMPLLM y SMPLitex	13,450 s
Textura	SMPLitex	13,373 s

Tabla 2: Tiempos medios de ejecución con diferentes configuraciones

Modelo	Tiempo de carga
SMPLLM	9,223 s
SMPLitex	10,030 s

Tabla 3: Tiempos de carga de los modelos

A la luz de los resultados obtenidos, se puede concluir que mantener varias redes cargadas simultáneamente, aunque tenga un impacto evidente en el uso de la memoria de la GPU, no implica un peor rendimiento en el tiempo de inferencia del sistema. Dado que los tiempos de carga de los modelos representan una porción significativa del tiempo de ejecución, es posible determinar que es preferible configurar el sistema con múltiples redes simultáneamente, en lugar de utilizar un único modelo a la vez, como plantean otros softwares de gestión de redes neuronales. Esto se debe a que cambiar de modelo requeriría volver a cargarlo. Siempre que haya suficiente espacio en la memoria, tener ambos modelos cargados será más eficiente si se espera al menos una solicitud para cada uno de los modelos.

5. CONCLUSIONES

5.1. Conclusiones sobre el desarrollo y uso de AIM4D

El desarrollo de AIM4D ha resultado en la creación de una herramienta robusta y versátil destinada a gestionar redes neuronales en el ámbito del desarrollo de videojuegos y gráficos 3D. Gracias a un enfoque modular, AIM4D facilita la integración de múltiples redes neuronales de diversas tecnologías, haciendo estas tecnologías avanzadas más accesibles a usuarios con distintos niveles de conocimientos técnicos. La elección de Python ha sido fundamental, debido a su clara sintaxis, la disponibilidad de bibliotecas de machine learning y su compatibilidad multiplataforma. La implementación de una interfaz gráfica mediante Tkinter y la integración con Unity para la visualización 3D ha permitido a los usuarios interactuar de manera intuitiva con el sistema y ver los resultados en tiempo real.

5.2. Éxitos, limitaciones y aprendizajes del proyecto

AIM4D ha logrado integrarse eficazmente con dos redes neuronales permitiendo la carga y gestión de estas a través de ficheros de configuración personalizados. La integración con el servidor AIM4D permite a la aplicación Unity aprovechar las potentes capacidades de las redes neuronales gestionadas por AIM4D sin necesidad de implementar complejas rutinas de procesamiento local. Esto descarga el procesamiento intensivo de la GPU del dispositivo del usuario, permitiendo que incluso dispositivos con capacidades limitadas puedan beneficiarse de tecnologías avanzadas de inteligencia artificial. La utilización de JSON para la comunicación asegura que los mensajes sean fácilmente extensibles y compatibles con futuros cambios en los modelos de redes neuronales o en los formatos de entrada y salida. Este enfoque modular y escalable facilita la integración y el uso de AIM4D.

El proyecto también ha enfrentado desafíos importantes, como la integración de redes neuronales de diferentes arquitecturas. Este proceso ha proporcionado lecciones sobre las distintas arquitecturas y cómo realizar la carga, descarga e inferencia de estas de manera modular mediante el uso de herencia. Por otro lado, la gestión de procesos entre distintas máquinas en el servidor ha sido un reto significativo, aportando como solución utilizar métodos asíncronos y síncronos de manera eficiente, aplicando el método adecuado para cada tarea. La capacidad de AIM4D para gestionar múltiples redes y procesar peticiones en tiempo real demuestra el éxito de estos esfuerzos, aunque siempre hay espacio para mejorar en términos de optimización y funcionalidad.

5.3. Propuestas de mejoras y futuras líneas de trabajo

AIM4D tiene un gran potencial para futuras expansiones. Entre las propuestas de mejora se incluye la optimización del rendimiento del sistema para manejar más eficientemente

los recursos de la GPU y CPU, así como la paralelización del uso de las redes neuronales para realizar inferencias simultáneas, reduciendo el tiempo de procesamiento. También es importante considerar la incorporación de nuevas redes neuronales y funcionalidades que amplíen las capacidades actuales del sistema.

Otra línea de mejora sería la ampliación de la compatibilidad con otras plataformas de desarrollo, lo que permitiría a más usuarios beneficiarse de las capacidades de AIM4D. Además, se podrían implementar herramientas adicionales para facilitar la depuración y el monitoreo en tiempo real, mejorando así la experiencia de desarrollo y mantenimiento.

Un aspecto clave para el crecimiento de AIM4D es la creación de una comunidad activa de usuarios y desarrolladores. Esto podría incluir el desarrollo de una "store" o base de datos comunitaria donde los usuarios puedan subir, compartir y buscar modelos de redes neuronales. Integrar un buscador que permita descargar modelos directamente de internet, en lugar de tener que buscarlos en el dispositivo, haría el sistema más accesible. Fomentar una comunidad activa proporcionaría un flujo constante de retroalimentación y nuevas ideas, asegurando que AIM4D evolucione de manera sostenible y se mantenga a la vanguardia de la tecnología de redes neuronales aplicadas a la creación de contenido digital.

REFERENCIAS

1. *Unity Technologies*, 2005: **Unity**. Plataforma de desarrollo en tiempo real utilizada para crear videojuegos y experiencias interactivas en 2D, 3D, realidad virtual (VR) y realidad aumentada (AR).
 - a. Enlace a su web: <https://unity.com/es>
 - b. Enlace a su documentación: <https://docs.unity3d.com/Manual/index.html>
2. *LMStudio*, 2023: **LMStudio**. Entorno de desarrollo que facilita la creación, entrenamiento y despliegue de modelos de lenguaje natural utilizando diversas técnicas de procesamiento del lenguaje natural (NLP).
 - a. Enlace a su web: <https://lmstudio.ai>
3. *Automatic111*, 2023: **Stable Diffusion Web UI**. Interfaz web popular y fácil de usar para la implementación de modelos de difusión en la creación de arte generativo y aplicaciones similares.
 - a. Enlace al GitHub del proyecto: <https://github.com/AUTOMATIC1111/stable-diffusion-webui>
4. *Steen Lumholt, Guido van Rossum*, 1991: **Tkinter**. Módulo de la biblioteca estándar de Python que proporciona herramientas para la creación de interfaces gráficas de usuario (GUI).
 - a. Enlace a su documentación: <https://docs.python.org/es/3/library/tkinter.html>
5. *Matthew Matl*, 2018: **Pyrender**. Librería en Python para la renderización de escenas 3D, que permite trabajar con gráficos de alto rendimiento y visualización en aplicaciones de computación gráfica.
 - a. Enlace a su documentación: <https://pyrender.readthedocs.io/en>
6. *Google Brain Team*, 2015: **TensorFlow**. Biblioteca de código abierto desarrollada por Google para el aprendizaje automático, que facilita la construcción y el entrenamiento de modelos de aprendizaje profundo.
 - a. Enlace a su web: <https://www.tensorflow.org>
7. *Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Meta AI*, 2016: **PyTorch**. Marco de aprendizaje automático de código abierto desarrollado por Facebook, conocido por su flexibilidad y facilidad de uso en la investigación y desarrollo de redes neuronales profundas.
 - a. Enlace a su web: <https://pytorch.org>
8. *Balduino Rodríguez*, 2024: **SMPLLM**. Red neuronal basada en LLaMA-3 capaz de generar los parámetros de forma de un avatar SMPL-X a partir de una descripción en lenguaje natural.
 - a. Enlace al GitHub del proyecto: <https://github.com/baldoarbol/tfg-gic/tree/main>
9. *Dan Casas, Marc Comino-Trinidad*, 2024: **SMPLitex: A Generative Model and Dataset for 3D Human Texture Estimation from Single Image**. Red que ofrece un

método para estimar y manipular la apariencia 3D completa de humanos capturados a partir de una sola imagen o de una descripción en lenguaje natural.

- a. Enlace al GitHub del proyecto: <https://github.com/dancasas/SMPLitex>
10. *Meta AI*, 2024: **LLaMA-3**. Modelo de lenguaje de última generación desarrollado por Meta, diseñado para realizar tareas de procesamiento del lenguaje natural con mayor precisión y eficiencia.
 - a. Enlace a su web: <https://llama.meta.com/llama3>
 - b. Enlace a su página de HuggingFaces: <https://huggingface.co/meta-llama/Meta-Llama-3-8B>
11. *G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, M. J. Black*, 2019: **SMPL-X, Expressive Body Capture: 3D Hands, Face, and Body from a Single Image**. Modelo paramétrico de cuerpo humano en 3D que incluye rostro y manos, utilizado para la captura y animación precisa de la forma y pose humanas a partir de datos de imágenes y videos.
 - a. Enlace a su web: <https://smpl-x.is.tue.mpg.de>
12. *HuggingFaces*, 2022: **Diffusers**. Biblioteca para la implementación de modelos de difusión, que son una clase de modelos generativos utilizados en la creación de imágenes y otras aplicaciones de generación de contenido.
 - a. Enlace a su página de HuggingFaces: <https://huggingface.co/docs/diffusers/index>
13. *A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin*, 2017: **Attention Is All You Need (Transformers)**. Arquitectura de red neuronal diseñada para manejar datos secuenciales y paralelos, ampliamente utilizada en el procesamiento del lenguaje natural (NLP) y la traducción automática.
 - a. Enlace a su página de HuggingFaces: <https://huggingface.co/docs/transformers/es/index>
14. *Guillermo Rauch*, 2023: **SocketIO**. Biblioteca que permite la comunicación en tiempo real entre clientes y servidores a través de WebSockets, facilitando la implementación de aplicaciones interactivas y colaborativas.
 - a. Enlace a su web: <https://socket.io>
 - b. Paquete de SocketIO utilizado en Unity: <https://github.com/itisnajim/SocketIOUnity>
15. *Winston Chang, Joe Cheng, Alan Dipert, Barbara Borges*, 2011: **WebSockets**. Protocolo de comunicación que proporciona canales de comunicación full-duplex sobre una sola conexión TCP, utilizado para aplicaciones que requieren interacción en tiempo real.
 - a. Enlace a su documentación: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

ANEXOS

Anexo A: Código completo del proyecto

Todo el código desarrollado para el proyecto está en el repositorio de Github: <https://github.com/baldoarbol/tfg-gddv>, así como la lista de todas las dependencias y librerías estándar de Python utilizadas.

Acceso al código en Python de AIM4D: <https://github.com/baldoarbol/tfg-gddv/tree/main/servidor-python>

Acceso al código y escena en Unity de la demo: <https://github.com/baldoarbol/tfg-gddv/tree/main/cliente-unity/aim4d-unity-client>