OMNI-IA GAME: ARQUITECTURA MODULAR Y FLUJO DE TRABAJO LOCAL PARA LA GENERACIÓN ASISTIDA DE RECURSOS DE VIDEOJUEGOS

OMNI-IA GAME: MODULAR ARCHITECTURE AND LOCAL WORKFLOW FOR AI-ASSISTED VIDEO GAME ASSET GENERATION

Jaime Alberto Arango
Estudiante del Programa de Ingeniería Multimedia,
Universidad Nacional Abierta y a Distancia (UNAD)
https://orcid.org/0000-0003-1782-4068
Correo: jaarango@unadvirtual.com

Javier Gerardo Reina Granados
Docente, Universidad Nacional Abierta y a Distancia (UNAD)
https://orcid.org/0000-0002-9424-9522
Correo: javier.reina@unad.edu.co

Néstor Fabián Gorrón Gómez
Docente, Universidad Nacional Abierta y a Distancia (UNAD)
https://orcid.org/0000-0002-7296-7005
Correo: nestor.gorron@unad.edu.co

Cómo citar: Reina Granados, J. G., Gorrón Gómez, N. F. ., & Arango, J. A. (2026). OMNI-IA Game: arquitectura modular y flujo de trabajo local para la generación asistida de recursos de videojuegos. Documentos De Trabajo ECBTI, 7(1). https://doi.org/10.22490/ECBTI.10308

RESUMEN

El desarrollo de videojuegos obliga a coordinar, en un mismo proceso, recursos visuales, textuales, sonoros y de programación. Para desarrolladores independientes y equipos pequeños, esta combinación puede traducirse en un flujo de trabajo fragmentado, con múltiples aplicaciones y servicios que deben configurarse y utilizarse por separado. En este contexto, la inteligencia artificial generativa ofrece nuevas posibilidades para apoyar tareas de ideación, producción preliminar y creación de recursos. Este documento de trabajo documenta el desarrollo de OMNI-IA GAME, un prototipo de escritorio con enfoque *local-first* que articula herramientas de inteligencia artificial y servicios heterogéneos dentro de un flujo común de generación, revisión y exportación para el prototipado de videojuegos. La solución integra un componente visual basado en ComfyUI, servicios de lenguaje ejecutados mediante Ollama y *llama-server*, y módulos de audio para efectos, música y síntesis de voz, coordinados desde una capa de orquestación desarrollada con Python y FastAPI. Su contribución no consiste en proponer nuevos modelos generativos, sino en explorar una arquitectura de integración local que reduzca la fragmentación del flujo y permita obtener resultados reutilizables en distintos entornos de desarrollo. El proceso se organizó mediante una adaptación de la metodología SUM, con fases de concepto, planificación, elaboración iterativa, beta y cierre. Las pruebas funcionales muestran que el prototipo puede generar *sprites*, escenarios, textos, *scripts* y recursos sonoros, exportarlos en formatos definidos e incorporarlos posteriormente a Unity como comprobación de interoperabilidad. Los tiempos registrados se midieron manualmente en una configuración principal de *hardware* y se presentan como referencias del proceso, no como un *benchmark* generalizable. El documento delimita así la arquitectura, el flujo metodológico y el estado funcional alcanzado, y deja para trabajos posteriores las evaluaciones experimentales de rendimiento, calidad, consistencia y experiencia de usuario.

Palabras clave: inteligencia artificial generativa; desarrollo de videojuegos; *local-first*; generación de *assets*; Unity; prototipado.

ABSTRACT

Video game development requires visual, textual, audio, and programming resources to be coordinated within a single production process. For independent developers and small teams, this can result in fragmented workflows involving multiple applications and services that must be configured and used separately. Generative artificial intelligence offers new possibilities for supporting ideation, early-stage production, and asset creation. This working paper documents the development of OMNI-IA GAME, a local-first desktop prototype that orchestrates heterogeneous AI tools and services within a unified workflow for the generation, review, and export of assets for video game prototyping. The solution integrates a visual component based on ComfyUI, language services executed through Ollama and llama-server, and audio modules for sound effects, music, and speech synthesis, all coordinated through an orchestration layer built with Python and FastAPI. Its contribution lies not in proposing new generative models but in exploring a local integration architecture designed to reduce workflow fragmentation and deliver reusable outputs for different development environments. Development followed an adaptation of the SUM methodology, encompassing the concept, planning, iterative elaboration, beta, and closing phases. Functional tests show that the prototype can generate sprites, environments, text, scripts, and audio assets, export them in predefined formats, and subsequently integrate them into Unity as an interoperability check. Execution times were manually recorded using a primary hardware configuration and are reported as development references rather than as a generalizable benchmark. The paper therefore documents the architecture, methodological workflow, and current functional state of the prototype, while leaving experimental evaluations of performance, quality, consistency, and user experience for future work.

Keywords: generative artificial intelligence; video game development; local-first; asset generation; Unity; prototyping.

INTRODUCCIÓN

El desarrollo de videojuegos reúne tareas que deben avanzar de forma coordinada: producción visual, diseño sonoro, narrativa, programación e integración técnica. En proyectos independientes o en equipos reducidos, esa diversidad puede convertirse en una carga operativa cuando cada actividad exige una aplicación, una configuración o un servicio diferente. En los últimos años, la inteligencia artificial generativa se ha incorporado a este escenario como herramienta de apoyo para explorar ideas, producir variantes y generar contenidos preliminares. Mao et al. (2026) relacionan estas técnicas con la evolución de la generación procedimental de contenido y documentan su aplicación en elementos visuales, narrativos y otros componentes utilizados en videojuegos.

El reto no consiste únicamente en disponer de modelos capaces de generar imágenes, texto, código, música o efectos sonoros. En la práctica, estas capacidades suelen distribuirse entre herramientas independientes, lo que obliga a cambiar de interfaz, administrar configuraciones distintas y trasladar manualmente los resultados de una etapa a otra. La literatura reciente sobre generación de *assets* también destaca la necesidad de integrar mejor estas herramientas con los entornos habituales de diseño y desarrollo y de conservar formatos de salida reutilizables (Fukaya et al., 2026). A partir de esta necesidad resulta pertinente explorar un enfoque *local-first*, en el que la mayor parte del procesamiento se realiza en el propio equipo una vez instalados los modelos y servicios necesarios.

La búsqueda de flujos más integrados no es exclusiva de este proyecto. Existen plataformas que ya agrupan capacidades generativas orientadas a la producción digital y de videojuegos. Layer ofrece acceso, mediante su plataforma y API, a modelos para recursos 2D, 3D, de video y audio, además de *workflows* que encadenan varias operaciones. Rosebud AI dispone de generación de recursos 2D, como *sprites*, iconos y fondos, que pueden descargarse en PNG e incorporarse a motores como Unity, Godot y GameMaker. Meshy se concentra en la generación de modelos 3D y documenta rutas de exportación e importación hacia Unity, Unreal Engine y Godot (Layer, s. f.; Rosebud AI, s. f.; Meshy, 2026). Estos antecedentes muestran que la integración entre generación y producción de *assets* ya es una línea activa de desarrollo. En consecuencia, OMNI-IA GAME no se presenta como la primera solución integrada de este tipo, sino como una exploración con un alcance técnico diferente.

Tabla 1. Comparación sintética de antecedentes tecnológicos y alcance de OMNI-IA GAME

Solución	Cobertura principal	Tipo de integración	Relación con OMNI-IA GAME
Layer	2D, 3D, video y audio	Plataforma y API gestionadas; *workflows* multietapa	Cobertura multimodal amplia mediante servicios del proveedor.
Rosebud AI	*Assets* 2D: *sprites*, íconos y fondos	Entorno web; descarga de archivos PNG para su uso en motores	Orientado a la generación rápida de recursos 2D y su transferencia a motores.
Meshy	Generación de modelos 3D	Servicio web; exportación e importación a motores	Especialización en 3D y rutas documentadas hacia Unity, Unreal Engine y Godot.
OMNI-IA GAME	Recursos visuales, texto y *scripts*, efectos de sonido (SFX), música y síntesis de voz (TTS)	Aplicación de escritorio *local-first* con servicios heterogéneos	Explora la orquestación *local-first* y la exportación multiformato dentro de un flujo común.

Fuente: elaboración propia con base en Layer (s. f.), Rosebud AI (s. f.) y Meshy (2026).

OMNI-IA GAME se sitúa en ese escenario con un propósito más acotado: no desarrolla modelos nuevos ni pretende competir con plataformas comerciales por la amplitud de servicios. El prototipo prioriza la orquestación *local-first* de herramientas heterogéneas —generación visual, lenguaje, *scripting*, efectos, música y síntesis de voz— dentro de una interfaz común, con exportación en formatos utilizados por Unity, Godot, Unreal Engine y proyectos web. La brecha que se aborda no es la ausencia de herramientas generativas, sino la articulación de servicios que puedan ejecutarse principalmente en el equipo del usuario y que permitan mantener la continuidad entre generación, revisión, exportación e interoperabilidad con herramientas utilizadas en las etapas posteriores del desarrollo. En la versión documentada, ComfyUI concentra el flujo visual, mientras que Ollama y *llama-server* permiten ejecutar modelos de lenguaje de forma local (Comfy-Org, s. f.; Ollama, s. f.). El término *local-first* indica aquí una prioridad por el procesamiento local; no supone que todos los servicios integrados deban funcionar necesariamente sin conexión.

El prototipo se desarrolla como trabajo de grado de Jaime Alberto Arango, en la modalidad de Proyecto Aplicado del programa de Ingeniería Multimedia de la Universidad Nacional Abierta y a Distancia (UNAD). La iniciativa se vincula con las actividades de investigación formativa del Semillero ARCADE, adscrito al grupo de investigación Byte in Design. Este contexto permite abordar el proyecto no solo como un ejercicio de implementación de *software*, sino también como una oportunidad para documentar un flujo de trabajo reproducible para el uso de

la inteligencia artificial aplicada al entretenimiento digital dentro de la Escuela de Ciencias Básicas, Tecnología e Ingeniería (ECBTI).

El alcance de este documento de trabajo se mantiene en el nivel de prototipo. El propósito es explicar cómo está organizada la solución, cómo se estructuró su desarrollo y qué funcionamiento ha podido comprobarse hasta el momento. Por ello, no se plantean todavía comparaciones exhaustivas entre modelos, mediciones de productividad ni evaluaciones formales de experiencia de usuario. El objetivo es describir la arquitectura modular y el avance funcional de OMNI-IA GAME, así como la integración de herramientas locales de inteligencia artificial para generar recursos visuales, textuales y sonoros y transferirlos de manera preliminar, a un flujo de prototipado en Unity.

METODOLOGÍA

Esta sección describe el enfoque de desarrollo adoptado, la adaptación de las fases de SUM al proyecto y las condiciones empleadas para las pruebas funcionales del prototipo. La metodología se utiliza como marco para organizar y documentar el proceso de construcción, no como objeto independiente de evaluación.

Enfoque general del desarrollo

El proyecto se aborda como un desarrollo tecnológico aplicado, centrado en construir y probar progresivamente una solución funcional. Para ordenar ese proceso, se tomó como referencia SUM, una metodología formulada para el desarrollo de videojuegos que recoge principios ágiles de Scrum y Extreme Programming (XP). Acerenza et al. (2009) la proponen para contextos de equipos pequeños, ciclos relativamente cortos y revisiones frecuentes del producto. En OMNI-IA GAME, SUM se utiliza como guía para organizar el avance y no como un conjunto rígido de prácticas que deban aplicarse de manera literal.

La relación con Scrum se refleja principalmente en la priorización de funcionalidades, el trabajo incremental y la revisión continua de versiones ejecutables. SUM añade una secuencia pensada para proyectos de videojuegos: concepto, planificación, elaboración, beta y cierre. Esta estructura permite separar las decisiones iniciales, la implementación de los módulos y la verificación funcional, de modo que cada avance pueda revisarse antes de incorporar nuevas capacidades.

Flujo general basado en SUM

El desarrollo se organizó en cinco fases adaptadas de SUM: concepto, planificación, elaboración, beta y cierre.

Concepto

En esta fase se definieron el problema, el usuario potencial y el alcance inicial del prototipo. OMNI-IA GAME se planteó como una aplicación de escritorio capaz de centralizar la interacción con herramientas locales de IA para generar recursos visuales, textuales y sonoros. También se estableció que el proyecto no entrenaría modelos propios, sino que integraría servicios existentes mediante una interfaz y una capa común de orquestación.

Planificación

El sistema se organizó en bloques de trabajo: arquitectura e interfaz, generación visual, generación textual, generación de audio, exportación de resultados, integración en motor y documentación. Estos componentes funcionaron como referencia de *backlog* para ordenar prioridades. También se consideraron requisitos no funcionales, entre ellos la ejecución local, una navegación comprensible, la estabilidad de la comunicación entre servicios y la compatibilidad de los archivos generados con herramientas posteriores del flujo de producción.

Elaboración

La implementación se realizó de manera iterativa. Primero se consolidaron la arquitectura base y la comunicación entre interfaz y *backend*; después se incorporaron los módulos visual, textual y sonoro, las opciones de exportación y las primeras pruebas en motor. Cada iteración partió de una funcionalidad concreta, continuó con su implementación y prueba básica, y finalizó con una revisión de incidencias y prioridades. Esto permitió volver sobre componentes ya desarrollados cuando aparecieron problemas de integración o ajustes necesarios en los formatos y la interfaz.

Beta

En este proyecto, la fase beta corresponde a una verificación funcional interna y no a una publicación abierta del producto. Las pruebas se orientaron a comprobar la comunicación entre la interfaz y los servicios locales, la obtención de salidas recuperables y la posibilidad de utilizar los archivos generados fuera de OMNI-IA GAME. Unity se empleó como entorno de comprobación para verificar, en una escena 2D, la incorporación de recursos exportados por el prototipo.

Cierre

El cierre reúne la versión funcional disponible, las evidencias de ejecución y la documentación del proceso. En este documento de trabajo representa un punto de corte para registrar lo desarrollado y analizar su comportamiento inicial; no implica que el proyecto esté concluido ni que sus componentes hayan sido evaluados de forma exhaustiva. Las pruebas comparativas y las evaluaciones más profundas se reservan para etapas posteriores.

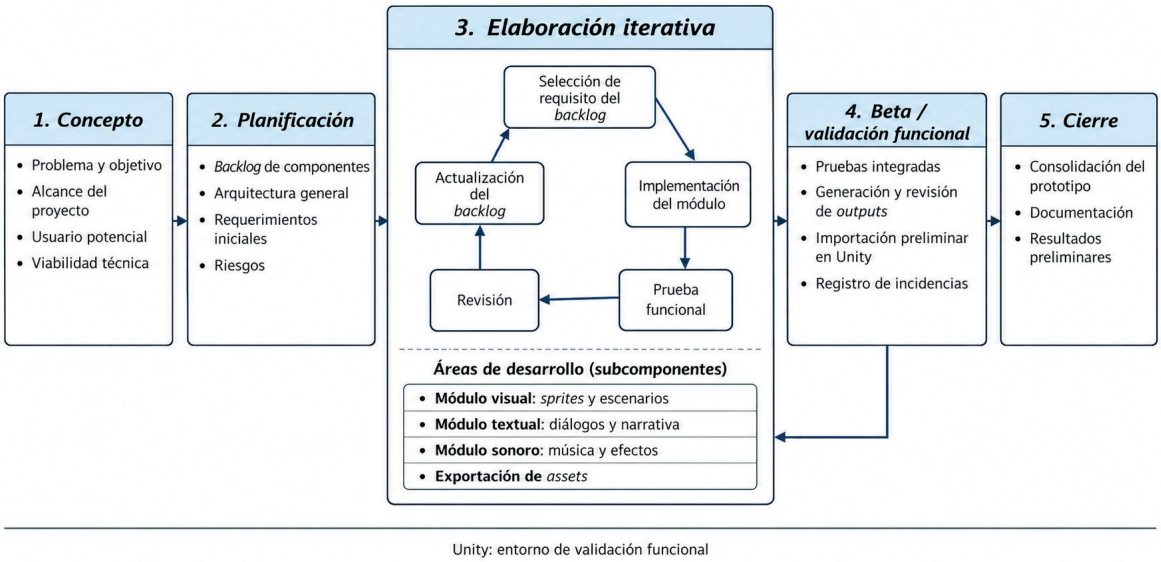
Flujo técnico y entorno de prueba

En términos operativos, el usuario selecciona un módulo, introduce un *prompt* y ajusta los parámetros disponibles; la aplicación envía la solicitud al servicio correspondiente; el modelo realiza la inferencia local, y el resultado vuelve a la interfaz para ser revisado y, cuando corresponde, exportado. Aunque cada módulo utiliza tecnologías distintas, la aplicación conserva una lógica de interacción común. ComfyUI se utiliza en el flujo visual; Ollama y *llama-server* gestionan los modelos de lenguaje, y los servicios de audio cubren efectos de sonido, música y síntesis de voz.

Las pruebas principales se realizaron en Windows 11 Pro sobre un equipo con procesador AMD Ryzen 5 5600X de seis núcleos, 40 GB de RAM y una GPU NVIDIA RTX 3090 con 24 GB de VRAM. Los tiempos se observaron manualmente durante ejecuciones funcionales realizadas por el desarrollador en las plataformas y módulos utilizados. Para lenguaje se utilizaron, entre otros, los identificadores locales gemma4:12b y qwen3.5:8b; el *pipeline* visual se probó principalmente con Z-Image Turbo; y para síntesis de voz se emplearon VibeVoice y Edge TTS. Durante el desarrollo también se exploraron Krea2, Ace Step, Minimax H3, LTX 2.5, WAN 3 y SDXL 2. Los nombres se conservan tal como estaban configurados en el entorno de trabajo y no se emplean aquí para establecer comparaciones entre modelos. Además del desarrollador, seis personas realizaron pruebas exploratorias no estandarizadas en equipos con configuraciones de *hardware* diferentes. Estas ejecuciones se consideraron únicamente como observaciones complementarias de funcionamiento y no se incorporaron a los intervalos de tiempo ni se trataron como una evaluación formal con usuarios.

La verificación se planteó en términos funcionales. En esta etapa, una salida se consideró satisfactoria cuando el módulo completaba la generación sin un error que impidiera recuperar el resultado, el archivo podía abrirse o reproducirse según su tipo, se exportaba en el formato previsto y, cuando correspondía, podía incorporarse a una herramienta posterior del *pipeline*. Para *scripts* se comprobó la exportación en la extensión seleccionada; para audio, la reproducción de los archivos, y para recursos visuales, la recuperación en PNG y su importación posterior. Estos criterios permiten comprobar la continuidad técnica, pero no equivalen a una evaluación estética, semántica o de utilidad productiva. Tampoco se realizaron repeticiones controladas de un mismo *prompt* ni se fijaron semillas y parámetros deterministas. En *pipelines* de difusión, la aleatoriedad puede reducirse mediante semillas y configuraciones específicas, aunque la reproducibilidad exacta no está garantizada entre versiones, plataformas o *hardware* (Hugging Face, s. f.). Por ello, este documento de trabajo se concentra en el funcionamiento y la interoperabilidad y no en la consistencia generativa o la experiencia de usuario.

Figura 1. Adaptación del flujo metodológico SUM al desarrollo de OMNI-IA GAME



Fuente: elaboración propia con base en datos de Acerenza et al. (2009).

RESULTADOS

Los resultados corresponden a pruebas funcionales realizadas durante el desarrollo. Su propósito es mostrar qué partes del flujo ya operan de forma integrada y cómo se comportan en el equipo de prueba, no establecer una comparación entre modelos ni medir el rendimiento con criterios experimentales. En el estado documentado, OMNI-IA GAME genera recursos visuales, texto, *scripts* y audio, los exporta en formatos definidos y permite incorporar recursos gráficos y sonoros a un proyecto de Unity.

Tabla 2. Configuración del entorno de prueba y formatos de salida utilizados en las pruebas funcionales

Elemento	Configuración / salida
Equipo de prueba	AMD Ryzen 5 5600X (6 núcleos); 40 GB de RAM; NVIDIA RTX 3090 con 24 GB de VRAM; Windows 11 Pro.
Texto y *scripting*	Ollama y *llama-server*; identificadores locales probados, entre ellos gemma4:12b y qwen3.5:8b.
Pipeline visual	ComfyUI; Z-Image Turbo como modelo base en la configuración de las pruebas visuales.

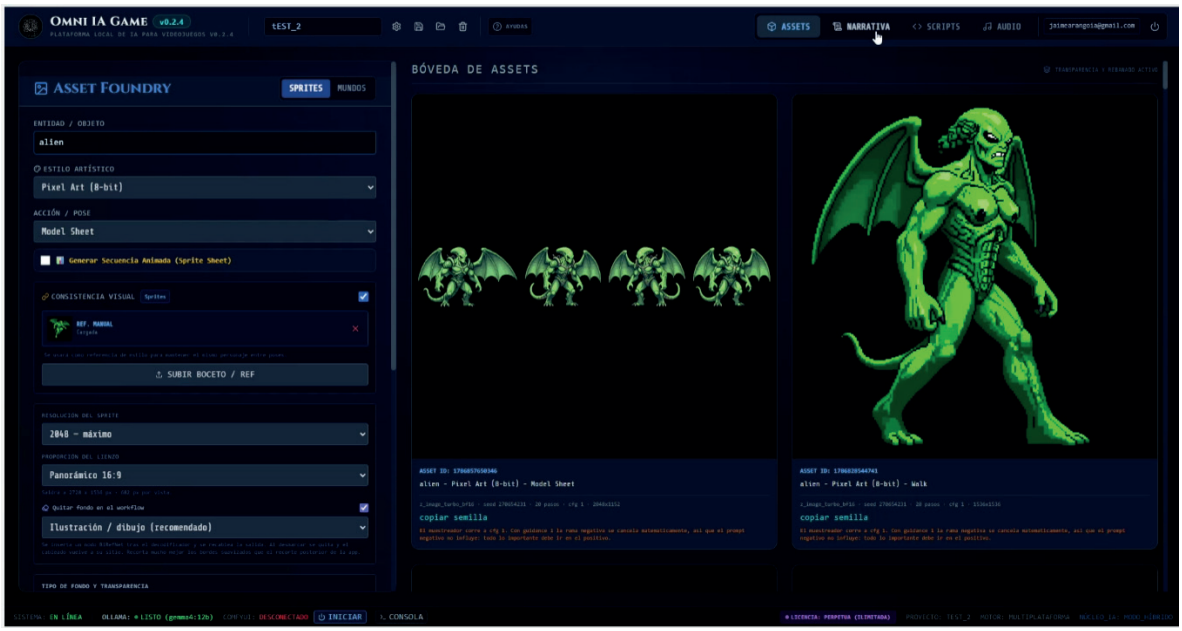
Elemento	Configuración / salida
Formatos visuales	PNG para *sprites*, escenarios y objetos.
Formatos de audio	WAV y MP3.
Formatos de *scripts*	.cs (Unity), .gd (Godot), .cpp (Unreal Engine) y .json (proyectos web).

Fuente: elaboración propia a partir de la configuración y los formatos utilizados en las pruebas del prototipo.

Resultados del módulo visual

En el módulo visual, Z-Image Turbo se utilizó como modelo base dentro de ComfyUI. En la instalación utilizada durante las pruebas, el modelo ocupaba aproximadamente 22 GB en disco. En el equipo principal, la carga inicial hizo que la primera generación tomara entre 3 y 5 minutos; una vez que el modelo permanecía cargado en VRAM, las solicitudes posteriores se resolvieron generalmente entre 5 y 7 segundos. De acuerdo con las observaciones manuales del desarrollador, los escenarios tendieron a requerir algo más de tiempo que los personajes cuando la solicitud incorporaba mayor complejidad. OMNI-IA GAME utiliza PNG para *sprites*, escenarios y objetos, lo que permite trasladar estos recursos a herramientas posteriores del flujo 2D.

Figura 2. Interfaz del módulo visual con ***prompt*** de entrada y recursos gráficos generados



Fuente: captura de pantalla del prototipo OMNI-IA GAME proporcionada por el desarrollador.

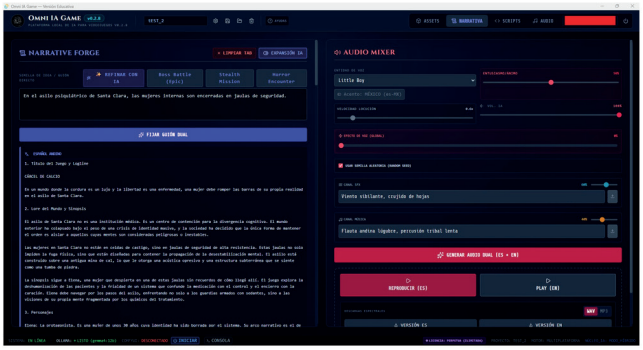
Resultados de los módulos textual, *scripting* y audio

El módulo textual funciona con modelos locales ejecutados mediante Ollama y *llama-server*. En las pruebas realizadas con el identificador local gemma4:12b, la carga inicial observada estuvo entre 15 y 17 segundos. La primera solicitud completa —texto, guiones, documentos de diseño de juego (GDD) o *scripts*— tardó entre 1 minuto 30 segundos y 2 minutos. Con el modelo residente en VRAM, las solicitudes posteriores se resolvieron aproximadamente entre 7 y 10 segundos. La aplicación permite exportar *scripts* en .cs para Unity, .gd para Godot, .cpp para Unreal Engine y .json para proyectos web. Estos formatos corresponden a opciones de exportación del prototipo; el presente trabajo no evalúa la compilación ni el comportamiento del código generado dentro de cada motor.

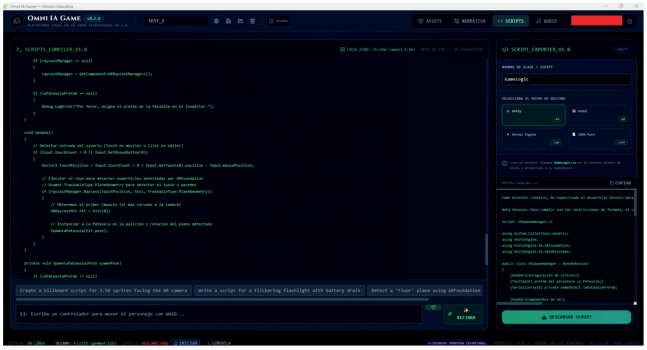
El componente sonoro cubre tres tipos de salida. En las pruebas manuales, los efectos de sonido (SFX) de aproximadamente 5 a 7 segundos se generaron en intervalos de 7 a 10 segundos. Para música ambiental o banda sonora, se observaron entre 5 y 7 minutos en la primera ejecución y entre 3 y 4 minutos en solicitudes posteriores; las generaciones que incluían voces tendieron a requerir más tiempo que las instrumentales. En síntesis de voz, Edge TTS produjo resultados entre 3 y 10 segundos según la longitud del texto. Con VibeVoice integrado mediante ComfyUI, la primera generación observada tomó entre 5 y 7 minutos y las posteriores entre 3 y 4 minutos. Los recursos sonoros se exportaron en WAV y MP3.

Figura 3. Evidencias funcionales de los módulos textual y de ***scripting***: (a) generación narrativa con síntesis de voz, (b) generación y exportación de ***scripts***

(a) Narrativa y síntesis de voz



(b) Generación y exportación de scripts



Fuente: capturas de pantalla del prototipo OMNI-IA GAME proporcionadas por el desarrollador.

Tiempos preliminares de ejecución

La tabla 3 resume los intervalos observados manualmente por el desarrollador durante la construcción y las pruebas del prototipo en el equipo principal. En varios procesos locales, una parte importante del tiempo inicial estuvo asociada con la carga del modelo en memoria; una vez residente en VRAM, el tiempo de respuesta disminuyó de forma apreciable. No se aplicó un número fijo de repeticiones por tarea ni se utilizó cronometraje automatizado. Las pruebas exploratorias de las seis personas adicionales tampoco se agregaron a estos intervalos debido a la heterogeneidad de sus equipos y a la ausencia de un protocolo común. En consecuencia, los valores describen el comportamiento observado en esa configuración y no constituyen un *benchmark* reproducible ni una comparación de rendimiento entre modelos.

Tabla 3. Tiempos preliminares observados manualmente durante las pruebas funcionales en el equipo principal

Proceso	Primera ejecución / carga inicial	Ejecuciones posteriores
Texto con gemma4:12b (identificador local)	Carga: 15-17 s; primera solicitud: 1 min 30 s-2 min	7-10 s
Imagen/escenario con Z-Image Turbo (configuración utilizada)	3-5 min	5-7 s
SFX de 5-7 s	No registrado por separado	7-10 s observados
Música ambiental/banda sonora	5-7 min	3-4 min
Edge TTS	No registrado por separado	3-10 s, según la longitud
VibeVoice	5-7 min	3-4 min, según la longitud

Nota. Los valores no constituyen un *benchmark* generalizable.

Fuente: elaboración propia a partir de observaciones manuales realizadas por el desarrollador durante las pruebas funcionales.

Integración preliminar en Unity

La comprobación en Unity se utilizó para cerrar el recorrido técnico del recurso dentro del flujo de trabajo. Los archivos exportados desde OMNI-IA GAME se incorporaron a un proyecto y se configuraron dentro de una escena 2D mediante los mecanismos

de importación del motor (Unity Technologies, s. f.-a, s. f.-b, s. f.-c). Esta prueba no corresponde a la implementación de un videojuego completo ni demuestra el uso de los *assets* dentro de una línea de producción terminada. Su finalidad fue comprobar la interoperabilidad: verificar que los archivos producidos por el prototipo podían ser aceptados por Unity y continuar dentro de un flujo convencional de prototipado.

Figura 4. Flujo básico de exportación e integración de recursos generados por OMNI-IA GAME en Unity



Fuente: elaboración propia a partir del flujo de exportación e integración definido para OMNI-IA GAME.

Los intervalos registrados corresponden únicamente a la configuración principal descrita y a observaciones manuales del proceso. El *hardware* disponible, el modelo seleccionado y la complejidad de la solicitud pueden modificar los tiempos de respuesta, por lo que estos valores no deben extrapolarse a otros equipos. Tampoco se realizó una comparación sistemática entre modelos, un análisis estadístico, una evaluación formal con usuarios ni una valoración objetiva de la calidad artística o la utilidad productiva. Del mismo modo, no se estudió la consistencia entre generaciones repetidas de un mismo *prompt*. Estas limitaciones delimitan el alcance de los resultados y evitan interpretarlos como una evaluación experimental del sistema.

DISCUSIÓN

Los resultados sitúan la contribución de OMNI-IA GAME en la integración tecnológica, no en la creación de nuevos modelos generativos. Layer, Rosebud AI y Meshy muestran que ya existen soluciones capaces de reducir parte de la fragmentación asociada con la generación de *assets*. La diferencia operativa explorada en OMNI-IA GAME es la prioridad dada a una arquitectura *local-first* que reúne servicios heterogéneos y mantiene una lógica común de interacción y exportación. El presente estudio no demuestra que este enfoque sea más eficiente, económico o usable que las plataformas existentes; únicamente documenta que la integración propuesta funciona en el nivel

de prototipo y permite mantener la continuidad entre la generación de recursos y las etapas posteriores del *pipeline*.

Las pruebas permiten sostener ese funcionamiento básico, pero no permiten establecer superioridad. La disminución de los tiempos una vez cargados los modelos en VRAM describe una característica observada en el equipo principal; las mediciones manuales y la ausencia de repeticiones estandarizadas impiden convertir esos intervalos en una conclusión sobre el rendimiento. De manera similar, la importación en Unity demuestra que los archivos generados son compatibles con el motor en las condiciones probadas, no su calidad final dentro de un videojuego. Las pruebas exploratorias realizadas por seis personas adicionales confirman que se ensayó el prototipo fuera del equipo principal, pero, al no compartir *hardware*, tareas ni protocolo, no aportan evidencia comparable sobre usabilidad o desempeño.

La valoración funcional adoptada distingue entre continuidad técnica y calidad del recurso. En este trabajo se comprobó que las salidas pudieran generarse, recuperarse, exportarse y, cuando correspondía, integrarse en una herramienta posterior. No se evaluaron de forma sistemática la calidad estética, la coherencia semántica, la utilidad productiva ni el grado de intervención humana requerido. Tampoco se estudió la consistencia de los *outputs* mediante repeticiones controladas. Estas dimensiones requieren otro diseño metodológico, con métricas definidas, condiciones reproducibles y usuarios representativos. El valor de esta etapa está, por tanto, en documentar un *pipeline* operativo y delimitar con claridad qué aspectos todavía necesitan una evaluación experimental.

CONCLUSIONES

Los resultados permiten concluir que OMNI-IA GAME ha alcanzado, en esta etapa, un nivel funcional de integración que permite recorrer un flujo básico de generación, revisión, exportación e interoperabilidad. La contribución del prototipo se sitúa en la articulación *local-first* de herramientas heterogéneas para recursos visuales, textuales y sonoros dentro de una interfaz común, no en el desarrollo de modelos propios ni en una supuesta superioridad frente a plataformas existentes. La importación satisfactoria de archivos en Unity ofrece una comprobación técnica de que las salidas pueden continuar en una etapa posterior del prototipado.

La adaptación de SUM permitió ordenar el desarrollo por componentes y revisar cada incremento antes de incorporar nuevas capacidades. Concepto, planificación, elaboración iterativa, beta y cierre funcionaron como una estructura práctica para documentar decisiones, integrar módulos y verificar el comportamiento del prototipo. En este trabajo, la metodología no se evalúa como objeto de estudio, sino como soporte para organizar un desarrollo tecnológico en progreso.

Las observaciones de ejecución muestran, además, que, en el equipo principal, la carga inicial de los modelos influye de manera importante en el tiempo de respuesta. Una vez residentes en VRAM, los intervalos disminuyeron de forma apreciable. Sin embargo, el procedimiento manual, la ausencia de repeticiones controladas y la diversidad de *hardware* de las pruebas exploratorias impiden generalizar estos valores. El trabajo tampoco aporta todavía evidencia suficiente para establecer conclusiones sobre la calidad de los recursos, la consistencia generativa, la intervención humana o la experiencia de usuario.

En consecuencia, OMNI-IA GAME debe entenderse como un prototipo de integración tecnológica en desarrollo. Una etapa posterior podrá establecer protocolos reproducibles para estudiar el rendimiento y la consistencia, criterios objetivos y subjetivos para valorar la utilidad de los recursos generados y pruebas con usuarios bajo condiciones comparables. Estas líneas no forman parte del alcance de este documento de trabajo, pero se desprenden de las limitaciones identificadas y constituyen una continuación natural del proyecto.

REFERENCIAS

Acerenza, N., Coppes, A., Mesa, G., Viera, A., Fernández, E., Laurenzo, T. y Vallespir, D. (2009). *Una metodología para desarrollo de videojuegos: Versión extendida* (Reporte Técnico RT 09-13). Instituto de Computación, Facultad de Ingeniería, Universidad de la República. https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/3420

Comfy-Org. (s. f.). *ComfyUI* [Repositorio de código fuente]. GitHub. https://github.com/Comfy-Org/ComfyUI

Fukaya, K., Daylamani-Zad, D., y Agius, H. (2026). Heuristics for ai-driven graphical asset generation tools in game design and development pipelines: A user-centered approach. *International Journal of Human-Computer Interaction*, 1-22. https://doi.org/10.1080/10447318.2026.2632170

Hugging Face. (s. f.). *Reproducibility* [Diffusers documentation]. https://huggingface.co/docs/diffusers/using-diffusers/reusing_seeds

Layer. (s. f.). *Documentation*. Layer AI. https://layer.ai/docs

Mao, X., Yu, W., Okawara, Y., Zhan, X., Yamada, K. D., y Zielewski, M. R. (2026). Procedural content generation via generative artificial intelligence. *Interdisciplinary Information Sciences*, *32*(2), 117-127. https://doi.org/10.4036/iis.2026.R.01

Meshy. (2026, 28 de julio). *Import Meshy Models into Any Engine: Unity, Unreal, Roblox, Godot* [Guía de importación]. Meshy Help Center. https://help.meshy.ai/en/articles/11973241-import-meshy-models-into-any-engine-unity-unreal-roblox-godot

Ollama. (s. f.). *Introduction*. Ollama Documentation. https://docs.ollama.com/api/introduction

Rosebud AI. (s. f.). *2D Assets* [Generador de recursos 2D]. https://dev.play.rosebud.ai/2d-assets

Unity Technologies. (s. f.-a). *Audio file compatibility* [Manual de Unity]. https://docs.unity3d.com/6000.3/Documentation/Manual/AudioFiles-compatibility.html

Unity Technologies. (s. f.-b). *Introduction to importing assets* [Manual de Unity]. https://docs.unity3d.com/6000.3/Documentation/Manual/ImportingAssets.html

Unity Technologies. (s. f.-c). *Set up sprites* [Manual de Unity]. https://docs.unity3d.com/2023.1/Documentation/Manual/sprites-setup.html