FARAUTE, Vol. 17, No. 2: 25-39, 2022 Print ISSN 1698-7418
Online ISSN 2542-3061

ARQUITECTURA DE SOFTWARE PARA LA INTEGRACIÓN DE MÓDULOS EN VIDEOJUEGOS USANDO UNITY 3D

Herrera, M. Torrealba, A. y Llobregat, V.
Universidad de Carabobo. Facultad Experimental de Ciencias y Tecnología.
Dpto. de Computación. Bárbula, Edo. Carabobo, Venezuela.
mherrera@uc.edu.ve, angelav.torrealbas@gmail.com, Vicente.llobregat94@gmail.com

Recibido: 23/09/2025, Revisado: 15/10/2025, Aceptado: 15/11/2025

Resumen

El desarrollo de videojuegos moderno involucra múltiples componentes interactivos, como sistemas de progreso, logros, avatares personalizables y tiendas virtuales, que incrementan considerablemente la complejidad estructural del producto. Para abordar este reto, se propone una arquitectura modular y basada en eventos para Unity 3D, fundamentada en patrones de diseño y buenas prácticas de ingeniería de software, que permite integrar, activar o desactivar diversos módulos funcionales sin comprometer la estabilidad del sistema. La arquitectura se diseñó a partir de una revisión teórica de estilos arquitectónicos y se implementó en un prototipo funcional como caso de estudio. Su efectividad se validó mediante un análisis de calidad basado en el estándar ISO/IEC 25010, evaluando atributos como mantenibilidad, modularidad, portabilidad, usabilidad y tolerancia a fallos. Los resultados demuestran que la arquitectura mejora la flexibilidad, escalabilidad y mantenibilidad, ofreciendo una base sólida para el desarrollo estructurado de videojuegos en contextos académicos e independientes (indie).
Palabras clave: arquitectura de software, videojuegos, modularidad, Unity, escalabilidad.

Software architecture for integrating modules in video games using Unity 3d

Abstract

Modern video game development involves multiple interactive components—such as progress tracking systems, achievements, customizable avatars, and virtual stores—that significantly increase the structural complexity of the product. To address this challenge, we propose a modular, event-driven architecture for Unity 3D, based on design patterns and software engineering best practices, which allows the integration, activation, or deactivation of various functional modules without compromising system stability. The architecture was designed following a theoretical review of architectural styles and implemented in a functional prototype as a case study. Its effectiveness was validated through a quality analysis based on the ISO/IEC 25010 standard, evaluating attributes such as maintainability, modularity, portability, usability, and fault tolerance. The results demonstrate that the architecture improves flexibility, scalability, and maintainability, providing a solid foundation for structured video game development in both academic and independent (indie) contexts.
Keywords: software architecture, video games, modularity, Unity, scalability.

1. Introducción

El desarrollo de software consta de varias etapas, como análisis de requerimientos, diseño, implementación, pruebas y mantenimiento, donde la arquitectura de software juega un papel fundamental al definir la estructura del sistema y orientar decisiones sobre los componentes y la organización del código, tal como describen Bass, Clements y Kazman (2021), quienes resaltan su papel como punto de referencia para la evolución y el mantenimiento del software. En el caso de los videojuegos, esta etapa cobra mayor relevancia debido a la complejidad de integrar múltiples sistemas interactivos en tiempo real.

Un videojuego es una aplicación interactiva que combina gráficos, sonido, mecánicas de juego y condiciones de victoria o derrota, con diversos niveles de dificultad. Su estructura incluye subsistemas como animación, lógica de juego, interfaz de usuario y almacenamiento de progreso, lo que lo convierte en un software complejo. La complejidad de un videojuego depende de sus objetivos y funcionalidades. Mientras algunos juegos presentan una estructura simple, otros incorporan sistemas de progreso, logros, tiendas virtuales, selección de avatar y clasificación de jugadores, lo que aumenta la dificultad de desarrollo, especialmente en términos de organización, escalabilidad y mantenibilidad.

Para organizar esta complejidad, es preciso apoyarse en paradigmas de desarrollo que abarcan desde patrones de diseño hasta *frameworks* y arquitecturas completas. Kaisler (2005) presenta precisamente esta jerarquía de paradigmas (patrones, componentes, arquitecturas, *frameworks*) como estructuras que ayudan a comprender cómo se construyen sistemas robustos y escalables. Esta visión sirvió de base para definir los estilos aplicados en nuestra arquitectura modular. Asimismo, Blancarte (2016) destaca que su correcta aplicación no solo optimiza la estructura interna del software, sino que también facilita la comunicación entre desarrolladores y reduce la curva de aprendizaje en proyectos complejos. Existen patrones arquitectónicos, como el modelo en capas, orientado a eventos o microservicios, y patrones de diseño, como *observer*, *singleton* o *strategy*.

Una arquitectura bien estructurada permite que los componentes se conecten de manera eficiente a través de interfaces y conectores, y que cada elemento tenga atributos que influencian su comportamiento (Figura 1). Esta estructura organizacional facilita la creación de arquitecturas flexibles, desacopladas y escalables, esenciales para el desarrollo de videojuegos modernos.

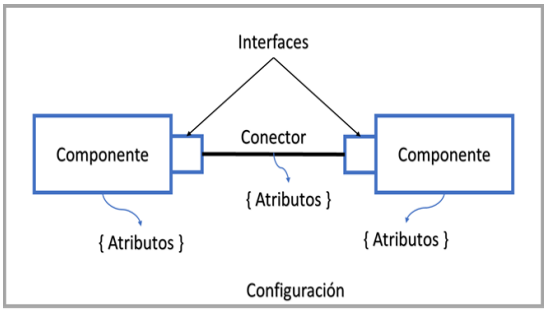

Figura 1. Relación entre componentes, conectores e interfaces en una arquitectura de software. Fuente: Adaptado de Kaisler (2005).

Específicamente, durante la investigación se realizó una revisión bibliográfica detallada sobre los estilos arquitectónicos y patrones de diseño más utilizados en ingeniería de software y desarrollo interactivo. A partir de esta revisión, se propuso una arquitectura modular, escalable y reutilizable para videojuegos desarrollados en *Unity 3D*. Esta arquitectura permite integrar módulos funcionales que se pueden activar o desactivar sin comprometer la estabilidad del sistema, siendo adaptable a cualquier videojuego que comparta principios comunes.

Para validar la propuesta, se implementó un prototipo funcional en un caso de estudio, y se realizaron pruebas orientadas a evaluar atributos de calidad como mantenibilidad, reutilización, tolerancia a fallos y portabilidad. Estos atributos fueron evaluados siguiendo el estándar ISO/IEC 25010, que establece métricas clave para la calidad del software. Los resultados obtenidos respaldan la eficacia de la arquitectura propuesta, como una guía para el desarrollo estructurado de videojuegos complejos.

2. Justificación

La investigación se justifica por la creciente necesidad de una arquitectura eficiente para los videojuegos, especialmente los desarrollados por equipos pequeños, en un contexto donde los juegos AAA dominan el mercado. Según Díaz (2023), a pesar de que las grandes compañías producen la mayoría de los lanzamientos de videojuegos, incluso estos suelen salir al mercado con errores importantes que son corregidos mediante actualizaciones posteriores. Este fenómeno también se presenta en los “juegos indie”, desarrollados por pequeñas empresas o incluso por una sola persona, que enfrentan mayores dificultades debido a la falta de recursos y herramientas adecuadas para gestionar la complejidad arquitectónica de sus proyectos.

Es de hacer notar que, como industria, el desarrollo de video juegos experimenta aciertos y fallos, en cuanto a las métricas relacionadas con el fracaso de éstos debido a problemas de arquitectura. En este sentido, se observaron estadísticas reveladoras en los estudios de Grewal et al. (2022) y Linåker et al. (2024) sobre la industria indie.

En el estudio de Grewal et al. (2022) se encontró que 48% de los juegos indie lanzados en *Steam* enfrentaron retrasos significativos en sus fechas de lanzamiento debido a fallos en la arquitectura de software ya que la falta de modularidad y escalabilidad, dificultó la integración de nuevas funcionalidades.

Asimismo, el trabajo de Linåker et al. (2024), presentó un análisis de 1,000 videojuegos indie lanzados entre 2020 y 2023, en el que más del 40% de los desarrolladores problemas de mantenibilidad y tolerancia a fallos. Esto se debió a decisiones arquitectónicas inadecuadas que afectaron tanto la experiencia del usuario como la capacidad del equipo de desarrollo para corregir errores post-lanzamiento.

Contexto del mercado de videojuegos

Dada la complejidad que implica desarrollar videojuegos, es evidente que los juegos indie enfrentan grandes desafíos al intentar competir con los juegos AAA. Según datos del SteamDB (2023), el número de juegos indie lanzados desde 2020 ha sido considerablemente inferior al de los juegos de mayor presupuesto. La Figura 2 compara el número de juegos indie lanzados desde el año 2020 con respecto al total de juegos (incluyendo juegos indie) durante el mismo periodo, considerando únicamente los juegos de la plataforma de distribución digital *Steam*.

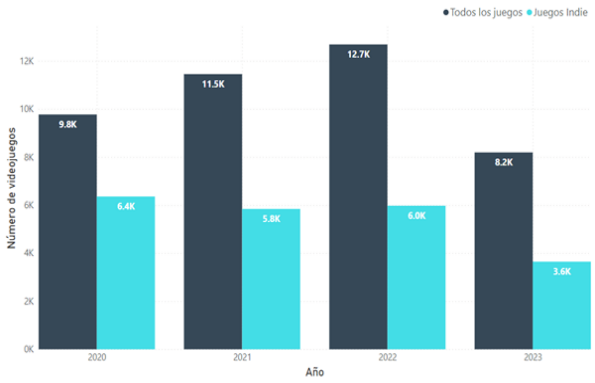

Figura 2. Número de juegos indie comparados con el total de juegos lanzados. Fuente: SteamDB (2023)

Por otro lado, la Figura 3 muestra los ingresos por ventas de los 50 videojuegos más vendidos en la plataforma *Steam* según la categoría del videojuego (Indie, AA, AAA), desde el año 2020. Esto indica que los ingresos generados por los juegos AAA siguen dominando el mercado, mientras que los juegos indie generan una fracción mucho menor de los ingresos totales. Estos datos reflejan cómo los grandes desarrolladores lideran el mercado, mientras que los desarrolladores indie tienen dificultades para obtener visibilidad y éxito financiero.

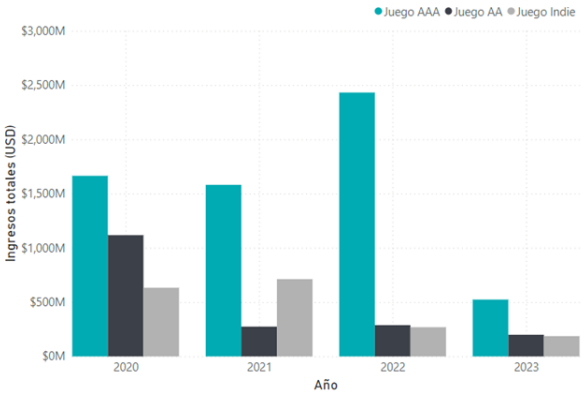

Figura 3. Ingresos totales por categoría en la plataforma Steam para los videojuegos más vendidos. Fuente: SteamDB (2023)

Es importante señalar entonces que los estudios y métricas sobre el fracaso de los videojuegos indie, junto con el análisis del contexto actual del mercado, subrayan la importancia de contar con una arquitectura de software modular, escalable y mantenible para el desarrollo de videojuegos. La propuesta de una arquitectura adecuada no solo facilitaría la integración de múltiples funcionalidades, sino que también ayudaría a los desarrolladores indie a superar los retos técnicos y a mejorar las posibilidades de éxito, en un mercado cada vez más competitivo, siguiendo un enfoque similar al de Pi, Domínguez y Hernández (2017), quienes desarrollaron una arquitectura modular en *Unity* 3D que integraba diversos sistemas funcionales y fue validada mediante métricas de calidad.

3. Metodología

Para llevar a cabo la investigación, se combinaron dos enfoques metodológicos complementarios: una estrategia de

investigación basada en el ciclo de Investigación-Acción, y un enfoque de desarrollo ágil usando *Extreme Programming* (XP). Esta combinación permitió abordar el problema desde una perspectiva teórica y práctica, construyendo una arquitectura aplicada, funcional y evaluable (Figura 4)

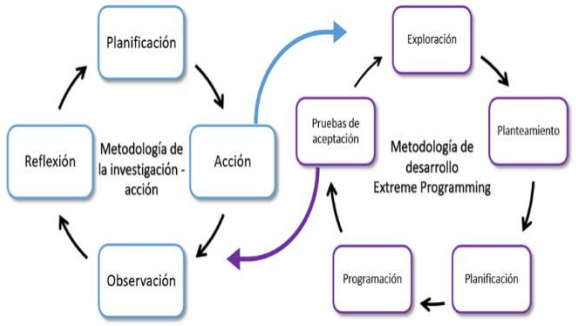

Figura 4. Ciclo completo utilizando la metodología de investigación y de desarrollo. Fuente: Adaptado de Latorre (2005).

Metodología de Investigación-Acción

Fueron consideradas sus cuatro fases, cada una con actividades concretas desarrolladas en el marco del trabajo:

Planificación: Consistió en identificar como problema central, la falta de una arquitectura modular, reutilizable y escalable para videojuegos desarrollados en *Unity* 3D que integrara múltiples módulos funcionales. A partir de esta necesidad, se formuló el objetivo principal del proyecto y se definieron los módulos que se incluirían en el diseño arquitectónico, tales como sistema de progreso, logros, vistas, audio, entre otros. Pressman (2010) enfatiza que un análisis claro de requerimientos y una definición temprana de la arquitectura son esenciales para garantizar la calidad y reducir los costos de mantenimiento.

Acción: Se realizó una revisión bibliográfica detallada, incluyendo fuentes académicas y técnicas sobre estilos de arquitectura, tales como: en capas, orientado a eventos, hexagonal, microkernel, entre otras; y patrones de diseño como: *singleton*, *observer*, *state*, *strategy*, *builder*, entre otros. Este análisis permitió seleccionar y combinar los elementos más adecuados para el contexto de videojuegos, dando lugar al diseño de una propuesta arquitectónica preliminar. Cervantes y Kazman (2016) señalan que un diseño arquitectónico efectivo debe considerar desde el inicio los atributos de calidad esperados, asegurando que las decisiones estructurales permitan cumplirlos a lo largo del ciclo de vida del sistema, principio que se siguió en esta etapa.

Observación: Se implementó un caso de estudio, mediante un prototipo funcional en *Unity 3D* que aplicó la arquitectura diseñada. Durante esta etapa se construyeron los módulos definidos y la documentación de su integración dentro del sistema. Asimismo, se observaron los comportamientos del juego bajo distintas configuraciones (por ejemplo, desactivando módulos específicos para validar independencia y flexibilidad).

Reflexión: Finalmente, se analizaron los resultados del caso de estudio, incluyendo métricas de calidad como modularidad, mantenibilidad y reusabilidad. Este análisis permitió ajustar y mejorar la arquitectura, así como extraer conclusiones sobre su aplicabilidad a otros desarrollos similares.

Aplicación de la metodología *Extreme Programming (XP)*

En paralelo, se aplicó la metodología XP como guía para el desarrollo técnico del prototipo, siguiendo las recomendaciones de Maurer y Martel (2022), quienes destacan que *Extreme Programming* favorece ciclos de desarrollo rápidos, retroalimentación continua y alta adaptabilidad a cambios de requisitos, distribuyendo el trabajo en cuatro etapas prácticas:

Planeación: Se definieron las historias de usuario correspondientes a los módulos funcionales del videojuego. Cada historia representó una necesidad concreta del usuario o del sistema, y sirvió como punto de partida para el diseño e implementación. También se priorizaron los módulos a desarrollar según su impacto en la arquitectura.

Diseño: Se elaboró un primer diagrama de arquitectura con todos los componentes principales y sus interacciones. Esta etapa fue clave para definir cómo se comunicarían los módulos entre sí mediante eventos, servicios o interfaces. El diseño también incluyó decisiones sobre la estructura de carpetas, clases base y relaciones de herencia o composición. Durante esta etapa, también se realizó un análisis de riesgos que permitió identificar problemas potenciales relacionados con el acoplamiento entre módulos y la rigidez en la comunicación.

Implementación: Se desarrollaron los módulos funcionales en *Unity*, cada uno implementado de forma desacoplada, con su propio flujo de datos y lógica interna. Se utilizó la inyección de dependencias para facilitar el intercambio de servicios (por ejemplo, en el sistema de audio), siguiendo las estrategias descritas por Aldazabal (2015), quien indica que desacoplar las dependencias mediante contenedores o inyección directa permite sustituir o actualizar componentes sin modificar el código que los utiliza, favoreciendo así la escalabilidad y mantenibilidad del sistema. También se diseñó un bus de eventos para centralizar la comunicación entre componentes. Se garantizó que cada módulo pudiera activarse o desactivarse sin afectar al sistema general.

Pruebas: Se ejecutaron pruebas de aceptación en el prototipo para validar tanto el funcionamiento individual de los módulos como su integración. Se evaluaron atributos como la madurez del sistema, su tolerancia a fallos, la capacidad de reusar módulos en otros proyectos, y la facilidad de modificar o extender funcionalidades existentes.

4. Resultados y análisis

La arquitectura propuesta fue validada mediante la implementación de un prototipo funcional de videojuego desarrollado en *Unity 3D*, que integró diversos módulos diseñados de forma desacoplada. Esta implementación permitió comprobar la viabilidad del enfoque modular y la eficacia de las decisiones arquitectónicas tomadas (Ver Figura 5).

Figura 5. Pantalla inicial del juego desarrollado como caso de estudio. Fuente: Elaboración propia.

Diseño e implementación de la arquitectura

A partir del análisis de los patrones y estilos arquitectónicos seleccionados, se diseñó una arquitectura de software compuesta por múltiples sistemas funcionales independientes (vistas, audio, texto, imágenes, progreso, base de datos, *popups*, entre otros), intercomunicados mediante un bus de eventos. Esta aproximación sigue el enfoque integral propuesto por Jason (2019), quien en explica cómo los motores de juego profesionales organizan subsistemas modulares y especializados para facilitar evolución, rendimiento y mantenimiento en entornos de alto nivel técnico

Se realizaron dos versiones del diagrama arquitectónico. La primera representó una propuesta inicial, y la segunda incluyó mejoras derivadas del análisis de riesgos y de la implementación progresiva del caso de estudio. Estas versiones permitieron visualizar de forma clara la evolución del diseño arquitectónico y cómo cada componente se ajustaba a principios de bajo acoplamiento y alta cohesión, tal como promueve Martin (2018) en su propuesta de *Clean Architecture*, donde se enfatiza la independencia entre capas y la protección de las reglas de negocio frente a cambios en *frameworks*, interfaces o bases de datos. En la *Figura* 6 se puede observar la primera versión del diagrama de la arquitectura propuesta.

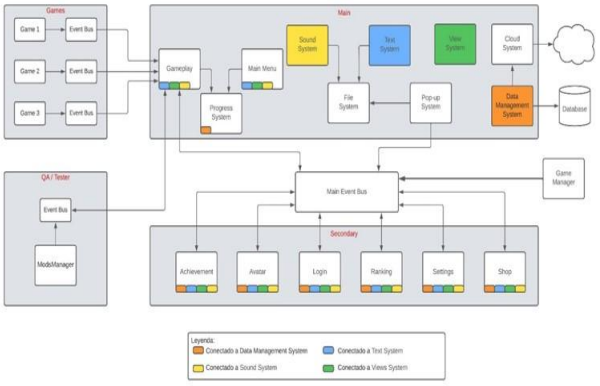

Figura 6. Diagrama de arquitectura del software: primera versión. Fuente: Elaboración propia.

La segunda versión del diagrama de arquitectura se planteó como resultado del análisis de riesgos realizado en la etapa de diseño. En ésta, se incorporó un bus de eventos independiente por módulo, con el objetivo de favorecer una comunicación desacoplada y flexible. Richards y Ford (2020) describen el bus de eventos como un middleware de mensajería que permite a productores y consumidores intercambiar información sin estar directamente acoplados, lo que favorece la escalabilidad, modularidad y mantenibilidad del sistema.

Según Lazzari y Farias (2021), una arquitectura basada en eventos mejora la separación de responsabilidades y permite una evolución más controlada del sistema, al reducir el acoplamiento entre componentes y facilitar la incorporación de nuevas

funcionalidades sin afectar el núcleo del sistema. Además, en el contexto de juegos serios, Söbke y Streicher (2016) destacan que una arquitectura distribuida basada en eventos permite una integración más eficiente de diversos módulos, optimizando el rendimiento y la escalabilidad del sistema.

De acuerdo con Rob Howard (2006), el patrón de diseño *provider* permite una flexibilidad significativa al desacoplar la lógica de negocio de las implementaciones concretas, lo que favorece la extensibilidad del sistema sin afectar su núcleo. De forma similar, Etheredge (2009) describe el *Provider Model* como un mecanismo que abstrae la funcionalidad de un *API* mediante una implementación separada, similar al *Strategy Pattern*, lo cual facilita intercambiar componentes sin modificar el código del cliente.

Este enfoque también se alinea con las recomendaciones de usar patrones modulares para mejorar la sostenibilidad a largo plazo de aplicaciones complejas en entornos dinámicos. Por esa razón, se introdujeron *providers* para gestionar el acceso a servicios internos de cada sistema, mejorando la organización y el control de dependencias. Asimismo, se empleó inyección de dependencias para abstraer servicios y reducir acoplamiento (Aldazabal, 2015). Estos cambios permitieron una mayor escalabilidad y mantenibilidad del sistema, tal como se refleja en la Figura 7.

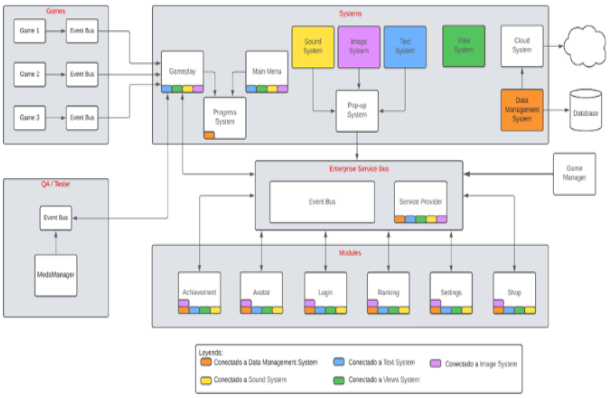

Figura 7. Diagrama de arquitectura del software: segunda versión. Fuente: Elaboración propia.

Caso de estudio y evaluación de calidad

Durante el desarrollo del caso de estudio, se realizaron pruebas orientadas a validar diversos atributos de calidad del sistema, con el objetivo de evaluar la efectividad de la arquitectura propuesta. Para este proceso, se utilizó el estándar ISO/IEC 25010 (2011), que define un modelo de calidad para productos de software, centrado en características clave como mantenibilidad, modularidad, usabilidad, tolerancia a fallos, capacidad de recuperación y portabilidad, entre otros.

La selección de estos atributos se basó en la necesidad de evaluar aspectos fundamentales de la arquitectura del sistema, asegurando que cumpliera con los requisitos esenciales para su escalabilidad, flexibilidad y fiabilidad a lo largo de su ciclo de vida. La aplicación de este estándar proporcionó un marco de referencia claro para evaluar la arquitectura, garantizando que cumpliera con los más altos niveles de calidad, rendimiento y fiabilidad. A continuación, se detallan las pruebas realizadas tomando en cuenta los atributos de calidad mencionados.

Mantenibilidad

La mantenibilidad se identificó como uno de los atributos más sólidos de la arquitectura propuesta, dado que desde su diseño inicial se priorizó la creación de un sistema modular, flexible y fácilmente adaptable a nuevos requisitos. Esta estructura modular permite modificar o extender funcionalidades sin generar impactos significativos en el resto del sistema, lo que simplifica las tareas de mantenimiento y evolución del *software*. A continuación, se detallan las características clave relacionadas con la mantenibilidad:

Modularidad: Tal como se evidenció en los diagramas de arquitectura, el sistema fue diseñado con una clara separación de responsabilidades entre módulos. Cada componente funcional, como *login*, avatar, progreso, audio o vistas, opera de forma independiente, lo cual permite realizar cambios localizados o incluso eliminar por completo algunos módulos sin afectar el funcionamiento global de la aplicación.

Pruebas realizadas:

- **Deshabilitación del módulo de *Login*:** se eliminó de la escena el objeto correspondiente al gestor del *login*. El sistema inició con normalidad, omitiendo la pantalla de autenticación y considerando al usuario como invitado. En este modo, los datos se almacenaron únicamente en la base de datos local, permitiendo jugar y acumular progreso, aunque sin sincronización en la nube. Como se muestra en la Figura 8, el juego arranca directamente en el menú principal, asignando un nombre de usuario y avatar por defecto.

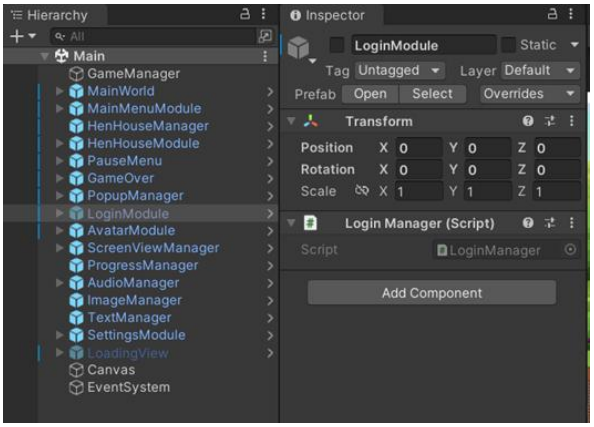

Figura 8. Escena principal de la aplicación en *Unity 3D* con el módulo de *Login* deshabilitado. Fuente: Elaboración propia.

- **Deshabilitación del módulo de Avatar:** Se removió el objeto del gestor del avatar en la jerarquía de Unity. El juego funcionó normalmente, aunque sin mostrar el avatar del usuario. Al acceder a la sección de personalización, ésta se mantuvo oculta, sin generar errores. El resto de los sistemas, como el progreso o la navegación, continuaron funcionando sin inconvenientes (Figura 9).

Figura 9. Menú principal del juego sin el avatar. Fuente: Elaboración propia.

Para resumir, las pruebas demuestran que el sistema responde adecuadamente ante la ausencia de módulos, lo cual confirma un

diseño desacoplado y tolerante a cambios. Asimismo, la modularidad es clave para facilitar el mantenimiento, permitir configuraciones personalizadas y reducir el riesgo de errores al introducir nuevas funcionalidades.

- **Reusabilidad:** Para verificar la reusabilidad, se reutilizaron módulos desarrollados en el caso de estudio, como el módulo de texto y el sistema de audio, en otros proyectos independientes. Los módulos fueron exportados e integrados sin necesidad de realizar adaptaciones significativas, lo que demuestra su bajo acoplamiento y capacidad de reutilización. Esta característica también se vio favorecida por el uso de providers y la centralización de la comunicación mediante eventos (Figura 10), lo que permite portabilidad y adaptación a otros entornos o aplicaciones sin dificultades.

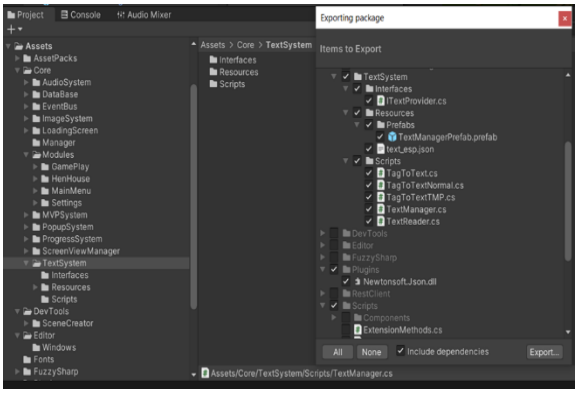

Figura 10. Exportación del módulo de texto. Fuente: Elaboración propia.

- **Tolerancia a fallos y capacidad de recuperación:** El sistema fue sometido a pruebas ante fallos como la pérdida de conexión a internet o al servidor Firebase. En general, la aplicación continuó funcionando, permitiendo al usuario guardar el progreso de forma local. Al recuperar la conexión, los datos se sincronizan automáticamente con el servidor. La única limitación se presentó en el inicio de sesión, que requiere conexión activa. En este caso, el usuario no puede avanzar hasta restablecer el servicio.

- **Usabilidad:** En términos de usabilidad, se evaluó especialmente la operabilidad del sistema, que mide qué tan fácil y eficiente es para los usuarios interactuar con el software. Para ello, la aplicación incluye un sistema de alertas que mejora la operabilidad al ofrecer mensajes claros cuando el usuario realiza acciones inválidas, tales como:
  - Campos vacíos.
  - Contraseñas demasiado cortas.
  - Intentos de ingreso sin conexión.
  - Datos incorrectos.

  Estas alertas permitieron comprender y corregir errores de forma sencilla, mejorando la experiencia de uso, tal como se muestra en la Figura 11. Este enfoque asegura que el sistema sea intuitivo y fácil de operar, lo cual se alinea con los criterios de operabilidad según ISO/IEC 25010.

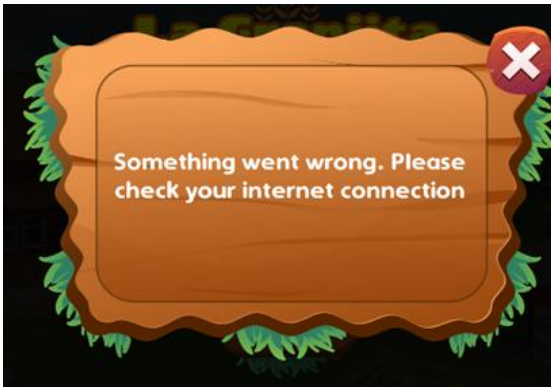

Figura 11. Alerta ante pérdida de conexión a internet. Fuente: Elaboración propia.

- **Portabilidad:** Finalmente, se ejecutó el prototipo en distintos entornos, incluyendo plataformas de escritorio y móviles. En todos los casos, la aplicación funcionó de manera estable, sin requerir ajustes específicos para cada plataforma. Esta característica fue posible, gracias a la modularidad de los sistemas, la abstracción de servicios y el uso de eventos desacoplados (Figura 12), lo que facilitó la adaptabilidad del sistema a diferentes plataformas sin necesidad de ajustes o modificaciones significativas.

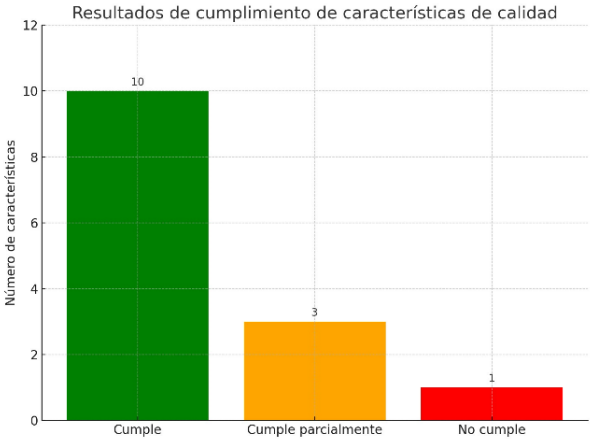

Figura 12. Resultado de cumplimiento de atributos de calidad. Fuente: Elaboración propia.

La portabilidad del sistema se validó con éxito, confirmando que la arquitectura es capaz de funcionar de manera estable en diversas plataformas, como *PC*, *Android y Web*, sin requerir modificaciones específicas para cada entorno.

- **Resultados de pruebas de aceptación por atributos y características:** A este respecto, la Tabla 1 presenta los resultados de las pruebas de aceptación realizadas, indicando el cumplimiento de cada característica evaluada y las observaciones correspondientes. Asimismo, muestra la cuantificación del nivel de cumplimiento de cada atributo de calidad, proporcionando una medida objetiva de qué tan bien la arquitectura responde a los requisitos esenciales para el desarrollo de videojuegos.

Tabla 1. Resultados de las pruebas de aceptación**.** Fuente: Elaboración propia

Atributo	**Característica**	**Cumple**	**Observaciones**
Confiabilidad	Madurez	Parcial	De 50 pruebas, presentó 4 fallos de conexión en WebGL y 3 incidencias de latencia en PC. Android no presentó fallos.
	Disponibilidad	Sí	La aplicación nunca estuvo fuera de servicio; los fallos en WebGL fueron de conexión, no de caída de servicio.
	Tolerancia a fallos	Sí	La app continúa funcionando ante fallos de conexión, excepto en inicio de sesión.
	Capacidad de recuperación	Sí	Recupera funcionalidad al restablecer conexión, sincronizando progreso automáticamente.
Mantenibilidad	Modularidad	Sí	Módulos secundarios deshabilitados sin fallos; deshabilitar audio bloqueó flujo antes de crear clase de respaldo.
	Reusabilidad	Sí	Módulo de texto exportado e integrado en otro proyecto sin errores.
	Capacidad de modificación	Sí	Cambio exitoso de base de datos remota a local sin afectar ejecución.
	Capacidad de prueba	Sí	Rastreo y depuración de eventos con enumeradores facilitó las pruebas.
Portabilidad	Adaptabilidad	Parcial	Funcionó en PC y Android; WebGL presentó fallos de conexión que se resolvieron reemplazando Rest Client.
	Capacidad de instalación	Sí	Instalación sencilla en Android (APK), PC y WebGL (navegador).
	Capacidad de reemplazo	Sí	Cambio de Rest Client por uno compatible con WebGL sin afectar otros módulos.
Usabilidad	Operabilidad	Sí	Alertas implementadas cubren todos los errores probados (campos vacíos, conexión, contraseñas inválidas).
Seguridad	Confidencialidad e integridad	No	Base de datos Firebase sin reglas de seguridad, accesible públicamente.

- De acuerdo con los resultados previos, se evaluaron un total de 14 características de calidad. De éstas, 10 características cumplieron completamente, 3 características tuvieron un cumplimiento parcial (madurez, tolerancia a fallos y adaptabilidad) y 1 característica no cumplió (seguridad - confidencialidad e integridad).

  Asimismo, muestran un porcentaje de cumplimiento total de 71.4%, un cumplimiento parcial del 21.4% y un incumplimiento del 7.2%. En términos generales, la arquitectura propuesta cumple satisfactoriamente en la mayoría de las características evaluadas. Las características con cumplimiento parcial requieren mejoras específicas, como optimizar la compatibilidad en WebGL y reducir la latencia en PC para mejorar la madurez, implementar soluciones offline o en caché para el inicio de sesión con el fin de fortalecer la tolerancia a fallos, y consolidar la adaptabilidad que ya fue solucionada mediante el reemplazo del Rest Client en WebGL.

  Es de hacer notar, que la única característica que no se cumplió, corresponde a la seguridad (confidencialidad e integridad) debido a la

ausencia de reglas de seguridad en la base de datos Firebase. No obstante, es importante señalar que en este caso no se implementó un sistema de seguridad avanzado porque se trata de un proyecto académico, aunque se reconoce que es un aspecto de gran importancia para entornos productivos y aplicaciones reales.

Finalmente, el sistema demostró un alto grado de cumplimiento en las características de calidad evaluadas, destacando su mantenibilidad, reusabilidad, capacidad de modificación, recuperación y operabilidad, y con las mejoras necesarias podría consolidarse como una arquitectura de software robusta y confiable, apta para proyectos de mayor alcance en la industria.

5. Conclusiones

El desarrollo de una arquitectura modular para videojuegos en *Unity 3D* permitió abordar de manera efectiva la complejidad inherente a este tipo de software, caracterizado por la integración de múltiples sistemas funcionales que interactúan entre sí. A través de una revisión exhaustiva de patrones arquitectónicos y de diseño, se identificaron y aplicaron soluciones adaptadas a las necesidades específicas del desarrollo de videojuegos modernos.

La arquitectura propuesta fue diseñada para facilitar la incorporación, modificación o exclusión de módulos como sistema de progreso, logros, audio, interfaz, base de datos, entre otros, sin comprometer la estabilidad del sistema.

Uno de los elementos más relevantes en esta integración, fue la incorporación de un bus de eventos para desacoplar los módulos y establecer un mecanismo de comunicación flexible, eficiente y extensible. Mediante este sistema de eventos, los módulos pudieron responder a acciones o cambios de estado sin requerir referencias directas entre sí, lo cual favoreció la mantenibilidad y la reducción de la complejidad de las dependencias.

Igualmente, la implementación del bus de eventos permitió observar una mejora significativa en la capacidad de escalar el sistema, ya que nuevos módulos pudieron suscribirse fácilmente a eventos existentes o definir eventos propios, sin necesidad de alterar el código base de otros componentes. Esta característica fue especialmente útil, durante las pruebas en las que se habilitaron y deshabilitaron módulos para validar su independencia funcional.

Es así que, los resultados obtenidos mediante pruebas de calidad evidenciaron que la arquitectura cumple con los principales atributos deseables en el desarrollo de software: mantenibilidad, reutilización, escalabilidad, portabilidad y tolerancia a fallos. Asimismo, el enfoque basado en patrones permitió construir un sistema flexible que pudiera servir como base para futuros proyectos, adaptándose a distintos estilos de juego o requerimientos funcionales.

En definitiva, esta propuesta representa una herramienta de valor para desarrolladores independientes o académicos que deseen estructurar sus proyectos de videojuegos desde una perspectiva arquitectónica sólida, basada en buenas prácticas de ingeniería de software y en mecanismos modernos de comunicación entre componentes como los eventos.

6. Agradecimientos

Los autores expresan su agradecimiento a la Universidad de Carabobo, en especial al Departamento de Computación de la Facultad Experimental de Ciencias y Tecnología, por el respaldo académico brindado durante la realización de este trabajo. También se extiende un reconocimiento a los desarrolladores de contenido en la tienda de *Unity* por ofrecer recursos gratuitos y de bajo costo que facilitaron la creación del prototipo utilizado en el caso de estudio.

7. Bibliografía

Aldazabal, L. (07 de marzo de 2015). Code2Read. C# Dependency Injection – ¿Qué estrategias existen para inyectar dependencias?: https://code2read.wordpress.com/2015/03/07/csharp-dependency-injection-estrategias-inyectar-dependencias-di/

Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice, 4th edition. Addison-Wesley Professional.

Blancarte, O. (2016). Introducción a los patrones de diseño - Un enfoque práctico. Ciudad de México: (Publicación independiente). Patrones de diseño.

Cervantes, H., & Kazman, R. (2016). Designing Software Architectures: A Practical Approach. Addison-Wesley Professional.

Etheredge, J. (06 de Julio de 2009). Simple Thread. The Provider Model Pattern, Really?: https://www.simplethread.com/the-provider-model-pattern-really/

Howard, R. (2006, June 29). Provider model design pattern and specification, Part 1. Microsoft Corporation: https://arxiv.org/abs/2110.14699 https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/june/provider-model-design-pattern-and-specification-part-1

ISO/IEC 25010:2011. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Software product quality (ISO/IEC 25010:2011). International Organization for Standardization: https://www.iso.org/standard/35733.html

Jason, G. (2019). Game Engine Architecture Third Edition. New York: CRC Press.

Kaisler, S. (2005). Software Paradigms. New Jersey: Wiley-Interscience.

Latorre, A. (2005). La investigación-Acción: Conocer y cambiar la práctica educativa. Editorial Graó.

Lazzari, L., & Farias, K. (2021). An exploratory study on the effects of event-driven architecture on software modularity. arXiv:

Lazzari, L., & Farias, K. (2023). Uncovering the hidden potential of event-driven architecture: A research agenda. arXiv: https://arxiv.org/abs/2308.05270fugumt.com

Martin, R. (2018). Clean Architecture. Pearson Education, Inc.

Maurer, F., & Martel, S. (7 de agosto de 2022). Extreme Programming: Rapid Development for Web-Based Applications. IEEE Internet Computing, págs. 86-91.

Pi, A., Domínguez, J., & Hernández, A. (2017). Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D. La Habana: Universidad de las Ciencias

Informáticas.

Pressman, R. (2010). Ingeniería del software un enfoque práctico séptima edición. New York: McGraw-Hill.

Richards, M., & Ford, N. (2020). Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, Inc.

Söbke, H., & Streicher, A. (2016). Serious games architectures and engines. En R. Dörner, S. Göbel, M. Kickmeier-Rust, & M. Masuch (Eds.), Entertainment computing and serious games (pp. 148–173).