UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

InkFall: Integración de sistemas y mecánicas en Unity para un videojuego narrativo multigénero de experiencia dinámica

PROYECTO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN REALIDAD VIRTUAL Y VIDEOJUEGOS

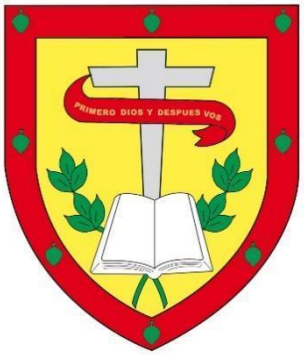
AUTOR: DARWIN ALEXANDER ABAD AUCANCELA

DIRECTOR: ING. JOHN LUIS ESTRADA GUAYTA, M.S.

CUENCA- ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

InkFall: Integración de sistemas y mecánicas en Unity para un videojuego narrativo multigénero de experiencia dinámica

PROYECTO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTOR: DARWIN ALEXANDER ABAD AUCANCELA
DIRECTOR: ING JOHN LUIS ESTRADA GUAYTA, M.S.

CUENCA- ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca

DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD

Declaratoria de Autoría y Responsabilidad

Darwin Alexander Abad Aucancela portador(a) de la cédula de ciudadanía Nº **0105878458**. Declaro ser el autor de la obra: **"InkFall: Integración de sistemas y mecánicas en Unity para un videojuego narrativo multigénero de experiencia dinámica."**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, **31 de agosto de 2026**

F: .....

Darwin Alexander Abad Aucancela

C.I. **0105878458**

Universidad Católica de Cuenca

CERTIFICADO

Certificado

Certifico que la presente Propuesta Tecnológica fue desarrollado por **Darwin Alexander Abad Aucancela** portador(a) de la cedula de ciudadania N.º **0105878458** con el tema "**InkFall: Integración de sistemas y mecánicas en Unity para un videojuego narrativo multigénero de experiencia dinámica.** ", bajo mi supervisión.

Cuenca, **31 de agosto de 2026**

F: .....

Ing. John Luis Estrada Guayta, M.S.

Docente - Tutor

Dedicatoria

Con el mayor cariño y gratitud, dedico este trabajo a mi familia, quienes siempre me han brindado su apoyo, sus consejos y la confianza necesaria para afrontar cada una de las decisiones que he tomado a lo largo de este camino.

A mis padres, Jenny y Darwin, por darme la oportunidad de estudiar y dedicarme a aquello que realmente me apasiona. Gracias por sus enseñanzas, por los valores que me han transmitido y por el apoyo incondicional que me han brindado en cada etapa de mi vida. Gracias a ustedes soy la persona que soy hoy y espero seguir creciendo para convertirme en una mejor versión de mí mismo. Su amor, esfuerzo y dedicación han sido pilares fundamentales para alcanzar esta meta.

A mi Cami, por acompañarme siempre, por seguirme cada una de mis bromas y por estar a mi lado en los momentos más importantes. Has sido un apoyo invaluable en mi vida y espero poder corresponderte siendo también un gran apoyo en la tuya.

A mis abuelos y a mis tíos, quienes siempre han estado presentes con su cariño, confianza y apoyo. Gracias por creer en mí y por ayudarme cuando más lo he necesitado. Su amor incondicional ha sido una fuente constante de motivación para seguir adelante y no rendirme.

Agradecimiento

Al haber terminado esta etapa significativa de mi vida, quiero agradecer a mis seres queridos y mis amigos por ser un apoyo incondicional en mi vida y ayudarme a alcanzar este logro importante.

Agradezco a mi director de tesis, Ing. John Luis Estrada Guayta M.S. por su acompañamiento y constante disposición para compartir sus conocimientos durante el desarrollo de este trabajo. Gracias por su confianza y compromiso a lo largo de este proceso.

Resumen

El presente trabajo tuvo como objetivo integrar los sistemas, mecánicas y recursos del videojuego narrativo multigénero InkFall mediante una arquitectura técnica modular implementada en Unity, capaz de articular los aportes de programación, modelado 3D, interfaz de usuario y audio sin comprometer la estabilidad, el rendimiento ni la mantenibilidad del proyecto. La investigación fue aplicada, de nivel descriptivo, y se desarrolló mediante una metodología iterativa basada en la incorporación progresiva de mecánicas, escenarios y sistemas, acompañada de procesos continuos de validación, corrección y optimización. Durante el desarrollo se definió una arquitectura basada en componentes, administradores y eventos para reducir el acoplamiento entre sistemas y facilitar la incorporación de nuevas funcionalidades. El proceso incluyó la organización de la estructura del proyecto, la configuración del entorno de desarrollo, la integración de recursos gráficos y sonoros, la incorporación de recursos de terceros y la implementación de sistemas de jugabilidad, interfaz y navegación. Las pruebas funcionales y de rendimiento realizadas con las herramientas de análisis de Unity permitieron verificar la integración de los componentes y medir las mejoras obtenidas. Como resultado, el prototipo incrementó su rendimiento de aproximadamente 30 a 63 fotogramas por segundo, equivalente a una mejora cercana al 110 %, y redujo en un 90 % la cantidad de lotes de renderizado generados durante la ejecución. Se concluye que la arquitectura implementada mejoró la fluidez, la estabilidad y la eficiencia del prototipo, y estableció una base técnica modular para su mantenimiento, ampliación y desarrollo colaborativo.

Palabras Clave: *Integración de sistemas, optimización, Unity, arquitectura modular, videojuegos.*

Abstract

This study aimed to integrate the systems, mechanics, and assets of the multi-genre narrative video game *InkFall* through a modular technical architecture implemented in Unity, capable of coordinating the contributions of programming, 3D modeling, user interface, and audio without compromising the project's stability, performance, or maintainability. The research was applied and descriptive in nature and was conducted using an iterative methodology based on the progressive incorporation of mechanics, levels, and systems, supported by continuous validation, refinement, and optimization processes. During development, a component-, manager-, and event-based architecture was defined to reduce system coupling and facilitate the integration of new functionalities. The process included organizing the project structure, configuring the development environment, integrating graphical and audio assets, incorporating third-party assets, and implementing gameplay, user interface, and navigation systems. Functional and performance testing performed using Unity's profiling and analysis tools verified component integration and quantified the improvements achieved. As a result, the prototype increased its performance from approximately 30 to 63 frames per second, representing an improvement of nearly 110%, and reduced the number of rendering batches generated during execution by 90%. It is concluded that the implemented architecture improved the prototype's smoothness, stability, and efficiency while establishing a modular technical foundation for its maintenance, expansion, and collaborative development.

Keywords: *System integration, optimization, Unity, modular architecture, video games.*

Contenido

Índice De Tablas

Índice De Ilustraciones

Introducción

El desarrollo de videojuegos modernos requiere la participación coordinada de diferentes áreas, como programación, modelado 3D, diseño de interfaces, audio y diseño de mecánicas. Aunque cada disciplina produce componentes de manera independiente, el éxito del proyecto depende de que todos estos recursos puedan integrarse correctamente dentro de un único entorno de desarrollo. Cuando este proceso no sigue una arquitectura organizada, pueden aparecer problemas de compatibilidad, dependencias innecesarias, dificultades de mantenimiento y disminución del rendimiento del sistema.

En este contexto, el presente trabajo desarrolla un sistema de integración técnica para el videojuego InkFall, implementado en el motor Unity. La propuesta establece una arquitectura y un flujo de integración que permiten incorporar progresivamente recursos y sistemas desarrollados por diferentes áreas del equipo, garantizando su compatibilidad, estabilidad y funcionamiento conjunto. Asimismo, se definen procedimientos para la organización del proyecto, la validación de los componentes integrados y el mantenimiento de la solución, facilitando la incorporación de nuevas funcionalidades durante el desarrollo del videojuego.

A continuación, se presenta una breve descripción de los capítulos que conforman este trabajo:

Capítulo I. Marco referencial: En este capítulo se presenta el contexto de la investigación, la problemática que motiva el desarrollo del proyecto, la formulación del problema, la justificación, los objetivos generales y específicos, el alcance, las delimitaciones, la metodología empleada, los beneficiarios y los resultados esperados.

Capítulo II. Marco teórico: En este capítulo se exponen los fundamentos conceptuales, metodológicos y tecnológicos que sustentan la propuesta. Se analizan trabajos relacionados con el desarrollo e integración de videojuegos, las tecnologías evaluadas durante la investigación, las arquitecturas y patrones de diseño considerados, las herramientas utilizadas y las decisiones técnicas adoptadas para la implementación del sistema de integración.

Capítulo III. Manual de usuario: En este capítulo se describe el funcionamiento del prototipo desde la perspectiva del usuario final. Se presentan los requisitos para la ejecución del videojuego, la instalación, la configuración inicial, los controles, las mecánicas principales y el uso general de la aplicación.

Capítulo IV. Manual del programador: En este capítulo se documenta técnicamente la implementación de la propuesta. Se describe la arquitectura de integración utilizada, la configuración del entorno de desarrollo, la organización del proyecto, el flujo de integración de recursos, la comunicación entre sistemas, los procedimientos de validación y las consideraciones necesarias para el mantenimiento y la incorporación de nuevas funcionalidades.

Capítulo I

1. Marco Referencial

1.1. Contextualización del problema

En el En el desarrollo de videojuegos modernos, la construcción de una experiencia interactiva requiere la integración de múltiples sistemas y recursos que cumplen funciones diferentes dentro de un mismo entorno de ejecución. El motor, las mecánicas, la interfaz, el audio, la inteligencia artificial, las animaciones y los recursos visuales deben mantener relaciones consistentes para que el videojuego funcione como un sistema conjunto. Jesse Schell, en «The Art of Game Design», plantea que los diferentes elementos que conforman un videojuego deben funcionar de manera complementaria para sostener una experiencia coherente, lo que implica que las decisiones tomadas sobre un sistema pueden afectar el funcionamiento de los demás [1]. Asimismo, Granic, Lobel y Engels señalan que los videojuegos contemporáneos pueden involucrar diferentes dimensiones cognitivas, perceptivas y emocionales, lo que incrementa la importancia de una correcta coordinación entre los elementos que intervienen en la experiencia interactiva [2].

Esta complejidad aumenta cuando el desarrollo involucra diferentes áreas de trabajo. Los recursos de programación, modelado 3D, animación, diseño de niveles, interfaz de usuario y audio son producidos mediante herramientas y procesos distintos, pero posteriormente deben converger dentro de un mismo proyecto.. Ernest Adams con su obra «Fundamentals of Game Design» [3] destaca la necesidad de mantener una relación coherente entre elementos como la interfaz y los sistemas de juego, debido a que inconsistencias en estos componentes pueden dificultar la interacción del usuario con el entorno virtual [3].

Un ejemplo claro de esta integración técnica se observa en el apartado sonoro. Karen Collins, autora de «Game sound» [4] define el audio dinámico (interactivo y adaptativo) como aquel que debe reaccionar tanto a las acciones del jugador como a los estados internos del sistema, lo que requiere una comunicación fluida entre el código lógico y los recursos de audio. Si estos sistemas no se integran de forma coherente, surge lo que Ropstam Game Studio [5] denomina un mal diseño, donde la falta de retroalimentación clara o la inconsistencia de las mecánicas rompen la confianza del jugador y provocan el abandono del proyecto.

Desde esta perspectiva, el problema no se limita a disponer de los recursos necesarios, sino a lograr que estos puedan incorporarse a una estructura común sin generar conflictos entre sistemas. Penny De Byl, en Holistic Game Development with Unity, plantea que la integración de arte, diseño y programación requiere una organización que permita coordinar las responsabilidades de las distintas áreas dentro de una arquitectura común [6]. En Unity, esta necesidad se relaciona directamente con la organización de escenas, GameObjects, componentes, recursos y sistemas que deben coexistir dentro del mismo proyecto [7].

Esta situación se presenta de manera concreta en el desarrollo de InkFall, donde los recursos fueron generados progresivamente por diferentes áreas del equipo y posteriormente incorporados al mismo proyecto de Unity. Durante este proceso se identificaron dificultades relacionadas con la compatibilidad entre recursos, la necesidad de actualizar referencias al sustituir elementos provisionales por recursos definitivos, la configuración de animaciones y controladores, la adaptación de interfaces previamente desarrolladas, la integración de modelos con sistemas de inteligencia artificial y

navegación, y la coordinación entre mecánicas, eventos y cambios de escena. Además, la incorporación de assets de terceros exigió verificar su compatibilidad con la versión del motor, el Universal Render Pipeline y la estructura existente del proyecto antes de utilizarlos en los niveles. La diversidad de estos recursos también generó la necesidad de mantener una organización técnica que evitara la propagación de errores entre sistemas. Javier Pernas en su proyecto de «Desarrollo de un simulador de eventos discretos experimental basado en realidad virtual» evidencia la importancia de separar las responsabilidades de los diferentes niveles del sistema y de establecer mecanismos de comunicación controlados para mantener la estabilidad durante la integración [8].De manera similar, Robert Nystrom, que en sus años de trabajo en Electronic Art, escribe «Game Programming Patterns», señala que una arquitectura sin una separación adecuada de responsabilidades puede generar dependencias excesivas y estructuras de código difíciles de mantener conforme el proyecto crece [9].

En consecuencia, el problema de InkFall se relaciona con la necesidad de incorporar y coordinar recursos y sistemas desarrollados de forma independiente dentro de un único proyecto, sin perder compatibilidad, estabilidad ni capacidad de mantenimiento. Esta situación hace necesario establecer criterios técnicos que permitan organizar los recursos, definir mecanismos de comunicación entre sistemas y validar cada incorporación antes de continuar con el desarrollo.

1.2. Problemática

Durante el desarrollo de InkFall, la incorporación progresiva de recursos provenientes de diferentes áreas generó situaciones que requerían intervención técnica para mantener la coherencia del proyecto. Los sistemas de movimiento, inteligencia artificial,

animación, interfaz, audio, navegación y mecánicas no siempre podían incorporarse directamente en su configuración original, debido a diferencias en referencias, estructuras de objetos, componentes utilizados o configuraciones propias de cada recurso.

En el caso de los recursos visuales, por ejemplo, la sustitución de modelos provisionales por los modelos definitivos requirió actualizar configuraciones de Animator, colliders, referencias y componentes asociados para conservar el funcionamiento de las mecánicas existentes. De manera similar, las armas proporcionadas por el área de modelado debieron integrarse con sistemas de combate previamente funcionales, mientras que los enemigos definitivos fueron incorporados sobre sistemas de inteligencia artificial y navegación ya desarrollados.

También se presentaron necesidades de integración en la interfaz y en la navegación del videojuego. Las interfaces y menús entregados por otras áreas requirieron actualización de referencias para ajustarse a la estructura definitiva del proyecto, mientras que las transiciones entre escenas debieron coordinarse con la gestión de escenas y los eventos correspondientes. En los escenarios, la incorporación de nuevos objetos o modificaciones en la geometría podía requerir actualizar colliders, navegación, iluminación y otros sistemas relacionados.

Estas situaciones evidenciaron que la incorporación individual de recursos no garantizaba por sí misma su correcto funcionamiento dentro del proyecto completo. Una modificación realizada sobre un componente podía afectar otros sistemas con los que mantenía dependencias, generando referencias perdidas, comportamientos inesperados, errores de configuración o problemas de rendimiento.

Por tanto, la problemática se centra en la ausencia de una estrategia de integración técnica que establezca criterios comunes para analizar, incorporar, configurar y validar los diferentes recursos y sistemas que forman parte de InkFall. Sin una estructura de este tipo, la incorporación progresiva de nuevos componentes puede incrementar el acoplamiento entre sistemas, dificultar la identificación de errores y aumentar el esfuerzo necesario para mantener y ampliar el videojuego.

1.3. Formulación del problema

Con base al problema, se formula la siguiente pregunta:

¿Cómo determinar e implementar una arquitectura de integración técnica en Unity que permita incorporar y coordinar los assets y sistemas de InkFall, manteniendo su compatibilidad, estabilidad y funcionamiento conjunto dentro de las diferentes mecánicas y escenarios del videojuego?

1.4. Justificación

La justificación de este trabajo nace de la necesidad de explorar nuevas formas de desarrollo de videojuegos donde la integración técnica no sólo cumpla una función operativa, sino que contribuya directamente a la construcción de la experiencia narrativa e interactiva. En este contexto, el motor de juego actúa como una metaplataforma de software que permite lo que Benjamin Nicoll y Brendan Keogh en «The Unity Game Engine and the Circuits of Cultural Software » denominan como «remezclabilidad profunda» (deep remixability) [10], integrando no solo contenidos de diferentes medios, sino también sus técnicas fundamentales y métodos de trabajo en un sistema unificado, por eso el rol de integración dentro de Unity adquiere una importancia clave, ya que permite articular de manera coherente los distintos componentes del juego, como modelos 3D, sonido e interfaz,

transformándolos en un sistema unificado capaz de responder dinámicamente a las acciones del jugador [11].

Además, este proceso resulta fundamental para garantizar la calidad, estabilidad y escalabilidad del proyecto, especialmente al trabajar con recursos provenientes de diferentes áreas de producción. Autores como Len Bass con «Software Architecture in Practice » mencionan que una integración adecuada no solo optimiza el rendimiento del videojuego, sino que también asegura que los elementos visuales, sonoros y jugables mantengan una coherencia estética y funcional.[12] Con la naturaleza progresiva y colaborativa del desarrollo del videojuego se introduce un nivel adicional de complejidad en el proceso de integración, debido a que los distintos assets y sistemas son incorporados en diferentes etapas del proyecto. Esta dinámica implica la necesidad de gestionar dependencias, controlar versiones y mantener la consistencia del sistema a medida que se agregan nuevos elementos. Así, el desafío no se limita a integrar componentes, sino a establecer un flujo de trabajo que permita su incorporación continua sin comprometer la estabilidad del entorno de desarrollo. Por ello, resulta fundamental definir criterios técnicos y procedimientos que faciliten la adaptación del sistema ante cambios, asegurando que la evolución del proyecto se mantenga controlada y alineada con los objetivos funcionales del videojuego.

En este contexto, el rol de integración técnica adquiere una relevancia fundamental dentro del proyecto, ya que no se limita únicamente a la incorporación de recursos en el motor de juego, sino que implica la gestión, articulación y validación de los distintos sistemas desarrollados por el equipo. Esto supone abordar desafíos relacionados con la compatibilidad entre assets, la comunicación entre módulos y la adaptación de los componentes a una arquitectura común que permita su funcionamiento conjunto. Por tanto,

el problema no solo reside en la integración operativa de elementos, sino en garantizar que dicha integración contribuya activamente a la coherencia funcional, el rendimiento del sistema y la correcta implementación de las mecánicas y la narrativa del videojuego.

1.5. Objetivos

1.5.1. Objetivo General

Implementar un sistema de integración técnica en Unity para la incorporación y configuración de los assets y sistemas del proyecto, mediante una arquitectura orientada a la gestión de sus componentes, con el fin de garantizar la operabilidad entre sistemas, el rendimiento en tiempo real y la coherencia funcional del videojuego InkFall en sus diferentes géneros y mecánicas.

1.5.2. Objetivo Específico

- Determinar los requerimientos técnicos de integración de los assets y sistemas del proyecto mediante el análisis de su estructura, compatibilidad y documentación, con el fin de establecer una arquitectura que permita su integración funcional dentro del videojuego.
- Diseñar la estructura de integración del sistema mediante la aplicación de patrones de diseño en videojuegos, que permita la coordinación e interacción eficiente entre los sistemas visuales, sonoros y de jugabilidad.
- Implementar las mecánicas de juego y la lógica de programación en Unity, garantizando su interacción con los demás sistemas y su coherencia dentro del flujo de jugabilidad.

- Configurar los modelos 3D en Unity, garantizando su compatibilidad con los sistemas de iluminación, físicas, animación y renderizado en los distintos escenarios del videojuego.
- Integrar los sistemas de interfaz de usuario y audio en Unity, asegurando su correcta sincronización con la lógica del juego y su respuesta coherente a las acciones del jugador.
- Validar la integración de los sistemas y assets en Unity mediante pruebas funcionales y análisis de rendimiento, con el fin de asegurar la estabilidad y eficiencia del sistema.

1.6. Alcance

El alcance del presente proyecto, desde el rol de integración, se centra en la unificación, implementación y validación de los distintos componentes del videojuego dentro del entorno de desarrollo Unity, abarcando desde la recepción de los recursos generados por las diferentes áreas hasta su correcta integración en un sistema funcional.

El proyecto contempla la construcción de un prototipo jugable multigénero, en el cual se integran y coordinan elementos visuales,sonoros y lógicos, garantizando su compatibilidad, coherencia y funcionamiento en tiempo real. No se incluye la creación directa de assets (modelado 3D, diseño sonoro o visuales), sino su integración dentro del sistema.

1.7. Delimitaciones

El presente proyecto se delimita al proceso de integración técnica de sistemas y mecánicas dentro del motor de videojuegos Unity para el desarrollo del prototipo funcional de InkFall. El trabajo se desarrolla específicamente desde el rol de integración, por lo que

las actividades realizadas se enfocan en la incorporación, configuración, adaptación y validación de los recursos generados por las diferentes áreas del equipo de desarrollo.

En este contexto, el proyecto no contempla el desarrollo original de modelos 3D, animaciones, ilustraciones, interfaces gráficas, efectos visuales, efectos sonoros o composiciones musicales, ya que dichos recursos son elaborados por las áreas correspondientes y posteriormente integrados dentro del motor Unity.

Asimismo, el trabajo no aborda el desarrollo completo de motores gráficos, algoritmos avanzados de inteligencia artificial, motores físicos personalizados ni herramientas externas de creación de contenido. La implementación se realiza utilizando las funcionalidades proporcionadas por Unity y los paquetes compatibles empleados durante el desarrollo del videojuego.

Las actividades de validación se limitan a pruebas funcionales, de integración y de rendimiento realizadas sobre el prototipo desarrollado. Estas pruebas tienen como objetivo verificar el correcto funcionamiento e interacción de los sistemas implementados, sin evaluar aspectos relacionados con la percepción, satisfacción o experiencia de los usuarios. En consecuencia, no se incluyen estudios de experiencia de usuario, pruebas de aceptación con una muestra de jugadores, análisis estadísticos ni evaluaciones comerciales del producto.

Finalmente, el proyecto se circunscribe al desarrollo del prototipo funcional de InkFall y a la documentación del proceso de integración técnica, sin contemplar etapas posteriores relacionadas con la publicación comercial, mantenimiento evolutivo o distribución del videojuego.

1.8. Metodología

1.8.1. Tipo de proyecto

El presente trabajo corresponde a un proyecto de carácter aplicado, ya que está orientado al desarrollo de una solución tecnológica para resolver un problema específico relacionado con la integración de sistemas y mecánicas dentro del motor de videojuegos Unity. Su finalidad no es generar conocimiento teórico de manera aislada, sino aplicar principios de ingeniería de software y desarrollo de videojuegos para implementar un proceso de integración que garantice la compatibilidad, estabilidad y correcto funcionamiento de los distintos componentes del proyecto InkFall.

1.8.2. Nivel del proyecto

El proyecto posee un nivel descriptivo, debido a que documenta y describe el proceso de integración técnica de los distintos sistemas, assets y mecánicas que conforman el videojuego InkFall dentro del entorno Unity. Para ello se analizan las actividades necesarias para incorporar componentes provenientes de diferentes áreas del desarrollo, como programación, modelado 3D, interfaz de usuario y audio, garantizando su funcionamiento coordinado dentro de una arquitectura común.

1.8.3. Metodología de desarrollo

Para el desarrollo del presente proyecto se adopta una metodología de desarrollo iterativa, orientada a la integración progresiva de los distintos sistemas que conforman el videojuego InkFall dentro del motor Unity. Este enfoque resulta adecuado debido a que el proyecto involucra la incorporación continua de componentes desarrollados por diferentes áreas del equipo, los cuales deben integrarse de forma coordinada para garantizar su correcto funcionamiento dentro de un único entorno de desarrollo.

A diferencia de metodologías secuenciales, el enfoque iterativo permite incorporar gradualmente nuevos componentes al proyecto, realizar pruebas de funcionamiento en cada etapa y aplicar correcciones antes de continuar con la siguiente fase del desarrollo. De esta manera es posible detectar tempranamente problemas de compatibilidad, errores de integración o afectaciones en el rendimiento, reduciendo el riesgo de que estos se acumulen durante el desarrollo del videojuego.

Esta metodología también favorece el trabajo colaborativo, ya que permite integrar continuamente los recursos generados por las distintas áreas del equipo sin depender de la finalización completa del proyecto para realizar las pruebas de funcionamiento.

Aunque este proceso incorpora principios propios del desarrollo iterativo, no sigue de forma estricta un marco de trabajo ágil específico, como Scrum, debido a que las actividades estuvieron condicionadas por la disponibilidad progresiva de assets y sistemas desarrollados por las diferentes áreas del proyecto. En consecuencia, la planificación se organizó en función de las etapas de integración y validación técnica, priorizando la incorporación ordenada de los recursos y la verificación continua de su funcionamiento dentro del entorno Unity.

1.9. Beneficiarios

1.9.1. Directos

Los beneficiarios directos del presente proyecto son los integrantes de equipos de desarrollo de videojuegos, especialmente las áreas de programación, diseño de videojuegos, modelado 3D, interfaz de usuario y audio, quienes disponen de un proceso de integración técnica que facilita la incorporación de los diferentes recursos dentro del motor Unity. Asimismo, el desarrollador encargado de la integración se beneficia al contar con una

estructura organizada que simplifica la implementación, validación y mantenimiento de los distintos sistemas del videojuego.

1.9.2. Indirectos

Los beneficiarios indirectos corresponden a estudiantes, principiantes y investigadores interesados en el desarrollo de videojuegos con Unity, quienes pueden utilizar la documentación generada como referencia para implementar procesos de integración técnica en proyectos similares. De igual manera, el prototipo desarrollado beneficia a la institución académica al constituir un antecedente técnico que puede servir como apoyo para futuros trabajos relacionados con ingeniería de software y desarrollo de videojuegos.

1.10. Resultados esperados

Como resultado del presente proyecto se espera obtener un prototipo funcional del videojuego InkFall, en el que los distintos sistemas, mecánicas y recursos desarrollados por las diferentes áreas del equipo se encuentren correctamente integrados dentro del motor Unity, garantizando su funcionamiento coordinado y estable.

Asimismo, se espera disponer de una arquitectura de integración organizada que facilite la incorporación de nuevos componentes, reduzca las dependencias entre sistemas y permita realizar modificaciones futuras sin afectar el funcionamiento general del proyecto. Esta estructura busca favorecer la mantenibilidad, escalabilidad y reutilización de los distintos elementos implementados durante el desarrollo.

Otro resultado esperado consiste en la documentación del proceso de integración técnica mediante un manual del programador, en el que se describa el entorno de desarrollo, la arquitectura utilizada, el flujo de integración, la comunicación entre sistemas, los

procedimientos de validación y las recomendaciones para el mantenimiento del proyecto. Esta documentación permitirá que otros desarrolladores comprendan la organización del sistema y puedan continuar su desarrollo o incorporar nuevas funcionalidades.

Finalmente, se espera que el proceso de integración aplicado contribuya a mantener un desempeño adecuado del videojuego mediante la ejecución de pruebas funcionales y revisiones básicas de rendimiento, verificando la correcta interacción entre mecánicas, interfaz de usuario, audio, escenarios y demás componentes que conforman el prototipo.

Capítulo II

2. Marco Teórico

2.1. Estado del arte

La literatura revisada evidencia que el desarrollo de videojuegos requiere integrar múltiples subsistemas que operan de manera coordinada dentro de un mismo entorno de ejecución. Alan Thorn, en «Pro Unity Game Development with C#» [13], aborda el desarrollo de videojuegos en Unity desde una perspectiva modular, destacando la necesidad de organizar los diferentes elementos del proyecto de manera que puedan interactuar sin generar una estructura difícil de mantener. Esta perspectiva coincide con la propuesta de Jason Gregory en «Game Engine Architecture» [14], quien describe los motores de videojuegos como sistemas compuestos por diferentes subsistemas especializados, entre ellos el renderizado, la física, el audio y la inteligencia artificial, cuya coordinación determina el funcionamiento general del motor.

Desde una perspectiva de arquitectura de software, Len Bass, Paul Clements y Rick Kazman, en «Software Architecture in Practice» [12], establecen que la arquitectura constituye un mecanismo fundamental para abordar atributos de calidad como la modificabilidad, el rendimiento y la mantenibilidad. Esta perspectiva complementa el análisis de Gregory, ya que mientras este último se concentra en la organización de los subsistemas propios de un motor de videojuegos, Bass, Clements y Kazman plantean principios arquitectónicos aplicables a sistemas de software en general. En ambos enfoques existe coincidencia respecto a la necesidad de establecer una estructura que limite las dependencias entre componentes y facilite la evolución del sistema.

En el contexto específico de Unity, Thorn [13] y Penny De Byl, en «Holistic Game Development with Unity» [6], trasladan estos principios hacia el desarrollo práctico de videojuegos mediante el uso de GameObjects, componentes y Prefabs. De Byl plantea además un enfoque holístico en el que programación, diseño y arte deben considerarse de manera coordinada durante la construcción del videojuego. Por su parte, Saku Eräniitty, en «Creating a Game Scene in Unity» [7], aborda la organización de escenas y recursos durante la construcción de entornos, destacando la importancia de mantener una estructura organizada para facilitar el flujo de trabajo y la incorporación de recursos provenientes de diferentes herramientas. En conjunto, estos trabajos muestran que la organización interna del proyecto constituye un aspecto recurrente en la construcción de videojuegos con Unity, aunque cada autor aborda el problema desde diferentes niveles de abstracción.

Otra línea identificada corresponde a la validación de los sistemas durante el desarrollo. Asmo Jurvanen, en su trabajo de fin de grado «Automated Testing with Unity» [15], analiza la utilización de pruebas automatizadas dentro de Unity y destaca su utilidad para detectar errores de manera temprana y reducir el riesgo de introducir fallos durante el desarrollo colaborativo. Por otro lado, Carboneras Mas, en «Simulador de eventos discretos para Unity» [16], estudia la integración de un sistema externo dentro del entorno Unity y resalta la importancia de establecer mecanismos de comunicación y sincronización adecuados entre los diferentes subsistemas. Aunque ambos trabajos se desarrollan sobre problemáticas diferentes, coinciden en la necesidad de verificar el comportamiento de los sistemas durante el proceso de integración y no únicamente al finalizar el desarrollo.

A partir de la comparación de estas investigaciones se observa que existe coincidencia en tres aspectos principales: la necesidad de una arquitectura modular para

controlar las dependencias entre sistemas, la importancia de una organización estructurada de los recursos dentro del motor y la conveniencia de realizar procesos de validación durante el desarrollo. Sin embargo, las fuentes revisadas abordan estos aspectos de manera parcial y desde perspectivas diferentes. Gregory y Bass, Clements y Kazman se concentran principalmente en los principios arquitectónicos; Thorn, de Byl y Eräniitty abordan la organización y construcción de proyectos en Unity; mientras que Jurvanen y Carboneras Mas se enfocan en mecanismos específicos de validación e integración.

En consecuencia, se identifica una brecha relacionada con la falta de una descripción integrada de estos enfoques aplicada al proceso de incorporación progresiva de recursos y sistemas desarrollados por diferentes áreas dentro de un mismo videojuego. Las investigaciones revisadas proporcionan fundamentos para organizar componentes, estructurar proyectos y validar sistemas, pero no presentan de manera conjunta un procedimiento técnico que articule estos elementos durante la integración de un videojuego multidisciplinario desarrollado en Unity. Esta brecha establece el marco para el desarrollo del presente trabajo, orientado a documentar un proceso de integración técnica que articule análisis, arquitectura, incorporación de recursos y validación dentro de un proyecto de videojuego.

2.2. Bases teóricas

La integración incremental constituye una estrategia de desarrollo en la que los diferentes componentes de un sistema se incorporan progresivamente y se verifican antes de continuar con nuevas integraciones. Carboneras Mas [16] evidencia que la incorporación progresiva de componentes permite identificar problemas de comunicación y sincronización durante etapas tempranas del desarrollo, reduciendo el riesgo de trasladar

errores hacia otros subsistemas. En el contexto de InkFall, este principio sustenta la incorporación progresiva de los recursos desarrollados por las diferentes áreas del proyecto, estableciendo una validación funcional después de cada integración antes de continuar con nuevos componentes.

La aplicación de una estrategia incremental se relaciona directamente con el principio de modularidad. Una arquitectura modular divide el sistema en unidades funcionales con responsabilidades específicas, permitiendo que cada módulo pueda desarrollarse y modificarse con un impacto reducido sobre el resto de la aplicación. En este sentido, Bass, Clements y Kazman [12] relacionan la organización modular de los sistemas con atributos de calidad como la modificabilidad y la mantenibilidad. Para InkFall, la modularidad constituye el principio que permite mantener separados los sistemas de jugabilidad, inteligencia artificial, interfaz, audio, gestión de escenas y recursos visuales, facilitando su integración progresiva.

La modularidad también depende de la relación existente entre cohesión y acoplamiento. Bass, Clements y Kazman [12] señalan que la cohesión se relaciona con el grado en que las responsabilidades de un componente están relacionadas entre sí, mientras que el acoplamiento representa el nivel de dependencia entre diferentes módulos. Una arquitectura con alta cohesión y bajo acoplamiento facilita la modificación y evolución del software, debido a que los cambios pueden limitarse a componentes específicos. Martin Fowleren su página web «Inversion of Control Containers and the Dependency Injection pattern», [17] complementa este principio al plantear que la reducción de dependencias directas entre componentes permite sustituir o modificar funcionalidades sin afectar innecesariamente el resto del sistema. En un videojuego, donde múltiples sistemas

interactúan simultáneamente, estos principios permiten reducir el impacto de las modificaciones realizadas durante la integración.

La reutilización de software constituye otro principio relacionado con la modularidad. Nystrom [9] plantea que los patrones y componentes reutilizables permiten construir nuevas funcionalidades a partir de soluciones existentes, evitando duplicar lógica y reduciendo el costo de mantenimiento. En el desarrollo de videojuegos, este principio puede extenderse tanto al software como a los recursos utilizados en las escenas, incluyendo Prefabs, materiales, animaciones, efectos visuales y otros componentes configurados previamente. En InkFall, la reutilización permite incorporar un mismo recurso en diferentes escenarios y conservar una configuración común, reduciendo la duplicación y facilitando las modificaciones posteriores.

Finalmente, la optimización en tiempo real constituye un principio necesario para mantener un desempeño adecuado durante la ejecución del videojuego. Akenine-Möller, Haines, Hoffman y sus colaboradores explican en su obra «Real-Time Rendering» [18] que la optimización de aplicaciones gráficas implica administrar eficientemente los recursos de procesamiento para mantener una tasa de actualización estable y reducir la carga sobre la CPU y la GPU. Gregory [14] señala además que las optimizaciones deben realizarse sobre sistemas funcionales y a partir de la identificación de problemas concretos, evitando introducir complejidad innecesaria en componentes que aún se encuentran en desarrollo. En consecuencia, dentro de InkFall la optimización se considera una actividad posterior a la integración y validación funcional, orientada a identificar y corregir posibles problemas de rendimiento antes de considerar estable una incorporación.

En conjunto, estos principios proporcionan la base teórica para la estrategia de integración utilizada en el proyecto: la incorporación incremental permite controlar el

proceso, la modularidad organiza los sistemas, la cohesión y el bajo acoplamiento reducen las dependencias, la reutilización evita duplicación de recursos y lógica, y la optimización permite asegurar un desempeño adecuado durante la ejecución.

2.3. Bases metodológicas

El desarrollo de la presente investigación se sustentó en un enfoque aplicado, debido a que los fundamentos estudiados fueron utilizados para resolver una problemática concreta relacionada con la integración técnica de sistemas y recursos dentro del videojuego InkFall. Desde el punto de vista del proceso de desarrollo, se adoptó un enfoque iterativo que permitió incorporar progresivamente los diferentes componentes del proyecto y verificar su funcionamiento durante cada etapa.

Para seleccionar el enfoque de desarrollo se analizaron diferentes metodologías y marcos de trabajo. En primer lugar, se consideró el modelo en cascada (Waterfall), caracterizado por organizar el desarrollo en etapas secuenciales. Este modelo favorece la planificación y documentación anticipada; sin embargo, presenta limitaciones cuando los requerimientos cambian o cuando los componentes se entregan progresivamente durante el desarrollo [19]. Estas características no se ajustaban completamente a InkFall, debido a que los recursos y sistemas eran proporcionados de manera progresiva por las diferentes áreas del equipo.

También se analizaron enfoques ágiles. El Manifiesto por el Desarrollo Ágil de Software de Kent Beck y sus colaboradores establece principios orientados a la colaboración, la entrega frecuente de resultados funcionales y la adaptación ante cambios en los requerimientos [20]. Dentro de este conjunto de enfoques se consideró Scrum, que organiza el trabajo mediante iteraciones y establece responsabilidades como Product Owner, Scrum Master y Developers. Aunque estas características resultan útiles en equipos

que requieren una estructura formal de coordinación, no se consideró conveniente aplicar Scrum de manera estricta debido al tamaño reducido del equipo y a la ausencia de dichos roles especializados. Asimismo, se analizó Kanban, metodología que utiliza un flujo visual de trabajo y permite administrar las tareas mediante estados y límites de trabajo en proceso [19]. Su flexibilidad resultaba adecuada para organizar las actividades del equipo; sin embargo, por sí sola no describía de manera suficiente el proceso técnico de integración, validación y ajuste realizado durante el desarrollo de InkFall. A partir de este análisis se adoptó un enfoque de desarrollo iterativo adaptado a las características del proyecto. La unidad básica de planificación correspondió a iteraciones de una semana. Al inicio de cada iteración se establecía el objetivo técnico correspondiente y se definían las actividades necesarias para alcanzarlo. Se documentó mediante un cronograma de trabajo organizado por semanas, en el que se registraron las actividades planificadas y ejecutadas durante las diferentes etapas del proyecto, complementado con un informe semanal en el que se registraron aspectos como los objetivos cumplidos, el progreso alcanzado, los próximos objetivos, las evidencias de trabajo, así como los riesgos y problemas identificados. Esta documentación permitió establecer objetivos técnicos para cada iteración y realizar un seguimiento progresivo de la integración de los recursos y sistemas de InkFall.

La Tabla 1 presenta la distribución de las principales actividades desarrolladas durante el proyecto.

Tabla 1

Cronograma semanal de actividades

N°	ACTIVIDAD	MES						MEDIOS DE VERIFICACIÓN
		I	II	III	IV	V	VI	

1	Definición de la estructura del proyecto en unity (Carpetas, nombres, escenas base)	**x**						Documento de organización del proyecto y evidencia de la jerarquía de carpetas y escenas en Unity.
2	Configuración inicial del entorno de desarrollo	**x**						Proyecto ejecutándose correctamente en el editor de Unity con configuración base funcional.
3	Definición de la arquitectura basada en componentes y eventos		**x**					Diagrama de arquitectura del sistema y scripts base implementados en el proyecto.
4	Preparación de sistemas para futura integración (UI, audio, gameplay)		**x**					Escenas de prueba con sistemas base (UI, audio, gameplay) configurados y operativos.
5	Integración de personajes en Unity			**x**				Personajes correctamente importados, configurados y visibles en escena con sus componentes activos.
6	Implementación del primer nivel dentro del sistema base			**x**				Escena jugable del nivel 1 funcional dentro del entorno Unity.
7	Ajustes iniciales de animación, físicas y lógica			**x**				Evidencia de ejecución en tiempo real con animaciones, colisiones y comportamiento funcional.
8	Pruebas funcionales básicas			**x**				Registro documentado de pruebas iniciales con verificación de funcionalidades principales.
9	Integración de escenarios en unity				**x**			Escenarios correctamente implementados y cargados dentro del proyecto.
10	Implementación de integración con objetos				**x**			Interacción funcional entre jugador y objetos validada en entorno de ejecución.

11	Integración del segundo nivel				**x**			Escena jugable del nivel 2 integrada y funcional.
12	Ajustes de iluminación, colisiones y navegación				**x**			Escenarios configurados con iluminación, físicas y navegación correctamente implementadas.
13	Integración del sistema de sonido (efectos y ambientación)					**x**		Evidencia de reproducción de efectos y ambientación sincronizados con eventos del juego.
14	Integración de la interface de usuario (HUD, menús)					**x**		Interfaz funcional (HUD y menús) operativa dentro del flujo del juego.
15	Implementación de los eventos dinámicos (cambios por reinicio y progreso)					**x**		Comportamiento del sistema modificado en función de eventos (progreso, reinicio), validado en ejecución.
16	Pruebas de coherencia general del sistema					**x**		Informe de pruebas de integración que evidencie la interacción correcta entre sistemas.
17	Optimización de rendimiento (renderizado, memoria, assets)						**x**	Registro comparativo de métricas de rendimiento (FPS, uso de memoria) antes y después de la optimización
18	Corrección de errores e inconsistencias						**x**	Listado documentado de errores detectados y corregidos durante el proceso.
19	Pruebas completas						**x**	Ejecución integral del sistema validada mediante pruebas realizadas dentro del motor.
20	Preparación de build final y documentación						**x**	Build ejecutable del videojuego y entrega de documentación técnica del proyecto (manual de usuario).

Nota: Cronograma de actividades semanales, Elaboración propia

El cronograma evidencia la aplicación sistemática del enfoque iterativo, debido a que las actividades fueron desarrolladas de manera progresiva y organizadas en función de los recursos disponibles y de las necesidades técnicas identificadas en cada etapa. Las actividades de análisis, integración, configuración, pruebas y corrección se distribuyeron en diferentes iteraciones, permitiendo incorporar los recursos conforme fueron entregados por las distintas áreas del proyecto y realizar ajustes antes de continuar con nuevas integraciones.

El proceso técnico utilizado para la integración se organizó de acuerdo con cinco etapas: análisis, diseño, integración, validación y optimización. Durante el análisis se identificaron los requerimientos técnicos, dependencias y condiciones de compatibilidad del recurso. En la etapa de diseño se determinó su ubicación dentro de la arquitectura y la forma en que se comunicaría con los sistemas existentes. Posteriormente, durante la integración, el componente fue incorporado al proyecto y configurado de acuerdo con la arquitectura establecida. La validación permitió comprobar su funcionamiento y detectar posibles incompatibilidades, mientras que la optimización se realizó cuando las pruebas de rendimiento evidenciaron oportunidades de mejora

En conclusión, el desarrollo de InkFall se fundamenta en un proceso iterativo de integración, más que en la aplicación estricta de un marco de trabajo específico. Este enfoque permitió adaptar el ritmo de desarrollo a la disponibilidad de assets, mecánicas y recursos provenientes de las distintas áreas, manteniendo la estabilidad del proyecto durante todo el proceso de implementación.

Tabla 2

Cuadro comparativo de Metodologías

Metodología	Ventajas	Limitaciones para InkFall
Waterfall	Alta planificación y documentación	Dificulta incorporar assets y mecánicas que se desarrollan de forma progresiva.
Scrum	Adaptación al cambio, trabajo colaborativo, iteraciones cortas	Requiere roles formales (Scrum Master, Product Owner) y ceremonias que no se ajustaban al tamaño del equipo.
Kanban	Gestión visual de tareas, flexibilidad y flujo continuo	Facilita la organización, pero no define un proceso técnico de integración.
Desarrollo Iterativo	Integración progresiva, validación continua y adaptación a la entrega de recursos	Requiere disciplina para realizar pruebas frecuentes y documentar cada integración.

Nota: Elaboración propia

2.4. Bases tecnológicas

El desarrollo de un videojuego requiere la selección de herramientas tecnológicas que permitan implementar las mecánicas, integrar recursos provenientes de diferentes áreas y mantener un flujo de trabajo estable durante todo el proyecto. Para la construcción de InkFall se analizaron distintas alternativas tecnológicas considerando aspectos como compatibilidad, facilidad de integración, disponibilidad de recursos, rendimiento y experiencia previa del equipo de desarrollo.

2.4.1. Motor de desarrollo

Uno de los primeros aspectos evaluados fue la selección del motor de videojuegos. Entre las alternativas más utilizadas actualmente destacan Unity y Unreal Engine, ambos ampliamente empleados tanto en la industria como en el ámbito académico.

De acuerdo con Gregory en «Game Engine Architecture», un motor de videojuegos proporciona la infraestructura necesaria para administrar gráficos, física, audio, animación, escenas y ejecución del juego, permitiendo que los desarrolladores concentren sus esfuerzos

en la implementación de la lógica del proyecto en lugar de construir estos sistemas desde cero [14].

Alberto Sastre dice en su informe para la página web Deusto Formación que aunque Unreal Engine ofrece una calidad gráfica superior y herramientas avanzadas para producciones AAA, Unity proporciona una mayor flexibilidad para proyectos independientes, una amplia disponibilidad de recursos reutilizables y una arquitectura basada en componentes que facilita la integración progresiva de nuevos sistemas[21].

Además, como señalan Nicoll y Keogh, Unity se ha consolidado como uno de los motores más utilizados para proyectos independientes gracias a su ecosistema de herramientas, su comunidad y la disponibilidad de recursos desarrollados por terceros [10].

Por estas razones se seleccionó Unity como entorno principal de desarrollo de InkFall. Esta decisión también estuvo respaldada por la experiencia previa del equipo con el motor, la existencia de múltiples assets compatibles y la facilidad para integrar componentes desarrollados por diferentes áreas dentro de un mismo proyecto.

2.4.2. Lenguaje de programación

Para el desarrollo de la lógica del videojuego se seleccionó C#, lenguaje utilizado de forma nativa por Unity para la implementación de scripts y componentes.

Según Alan Thorn, con su obra específica de este lenguaje para unity «Pro Unity Game Development with C#», la integración entre Unity y C# permite desarrollar sistemas modulares mediante clases, componentes reutilizables y programación orientada a objetos, facilitando la creación de videojuegos de gran complejidad [13].

Otra alternativa considerada fue C++, utilizado principalmente en Unreal Engine por ofrecer un mayor control sobre el hardware y optimizaciones de bajo nivel. Sin embargo, para un proyecto desarrollado completamente en Unity, C# representa una

solución más adecuada debido a su integración directa con el motor, menor complejidad de implementación y mayor rapidez durante el desarrollo.[21]

En consecuencia, toda la lógica correspondiente a mecánicas, administradores, inteligencia artificial, interacción entre sistemas e interfaz fue implementada utilizando C#.

2.4.3. Pipeline de renderizado

Otro aspecto tecnológico importante corresponde al sistema de renderizado.

Unity ofrece tres alternativas principales:

- Built-in Render Pipeline
- Universal Render Pipeline (URP)
- High Definition Render Pipeline (HDRP)

La documentación oficial de Unity indica que el Universal Render Pipeline (URP) está diseñado para ofrecer un equilibrio entre calidad visual y rendimiento, siendo especialmente recomendado para proyectos multiplataforma y videojuegos que requieren ejecutarse sobre hardware con recursos limitados. Además, Unity orienta el desarrollo de las versiones recientes del motor hacia los Scriptable Render Pipelines, entre los cuales se encuentra URP, incorporando en ellos las principales mejoras y nuevas funcionalidades. En versiones posteriores del motor, como Unity 6.5, el Built-in Render Pipeline comienza a considerarse una tecnología heredada (deprecated), reforzando la elección de URP como una alternativa con mayor proyección de mantenimiento y evolución para proyectos de largo plazo [22].

Mientras que HDRP proporciona un nivel gráfico superior orientado a hardware de alta gama (PC y consolas de última generación) a costa de una mayor carga computacional, URP permite implementar técnicas de alta fidelidad visual con un impacto controlado en el

rendimiento. Entre estas destacan el Oclusión Ambiental y el uso de Adaptive Probe Volumes, que mejoran significativamente la iluminación global sin los requisitos masivos de HDRP [22].

Considerando que InkFall incorpora escenarios amplios, efectos visuales y un enfoque orientado al rendimiento, se seleccionó Universal Render Pipeline (URP) como sistema de renderizado principal.

2.4.4. Sistema de entrada

Para la gestión de los controles del jugador se analizó el uso del sistema clásico Input Manager y el nuevo Input System desarrollado por Unity.

La documentación oficial indica que el nuevo Input System permite centralizar todas las acciones del jugador en un único archivo de configuración (Input Actions), facilitando la administración de múltiples dispositivos de entrada y reduciendo la dependencia entre el código y las configuraciones de control.[23] Además, el uso de este sistema se alinea con el concepto de mapeo (mapping) definido por Jesse Schell, quien explica que la transformación de una entrada física en una acción en el mundo virtual es lo que define la interactividad del sistema[1]. Al organizar todas las acciones desde un único recurso, se facilita lo que Navarro Estrella describe en su trabajo «Videojuego RPG en Unity» como un controlador centralizado (usualmente bajo un patrón Singleton), el cual permite que los personajes en escena detecten eventos de forma eficiente y desacoplada [24].

Debido a estas ventajas, InkFall implementa el Input System, permitiendo organizar todas las acciones del jugador desde un único recurso y facilitando la incorporación de nuevas mecánicas sin modificar la estructura general del proyecto.

2.4.5. Sistema de navegación e inteligencia

Para el desplazamiento de los enemigos se evaluó la posibilidad de desarrollar un sistema propio de navegación o utilizar las herramientas integradas del motor.

Ian Millington explica en «Artificial Intelligence for Games» que la utilización de mallas de navegación (Navigation Mesh) constituye una de las soluciones más eficientes para el desplazamiento de agentes dentro de entornos tridimensionales, ya que permite calcular rutas evitando obstáculos y reduciendo el costo computacional de los algoritmos de búsqueda [25].

Por este motivo se utilizó NavMesh, integrado con los agentes de navegación de Unity, permitiendo implementar el movimiento básico de los enemigos y facilitar su adaptación a los diferentes escenarios del videojuego.

2.4.6. Herramientas y assets complementarios

Además de las tecnologías principales, durante el desarrollo de InkFall se utilizaron herramientas y assets de terceros con el propósito de agilizar la construcción de escenarios, mejorar la calidad visual y reducir el tiempo de implementación de elementos que no forman parte del objetivo principal del proyecto.

Para la construcción de los escenarios se contempló el uso del gestor de terrenos nativo de Unity (Terrain) o el uso del asset Gaia desarrollado por Procedural Worlds, una herramienta especializada en la creación y edición de terrenos para Unity. Gaia fue la opción elegida porque, aunque incorpora funciones avanzadas para la generación de paisajes, vegetación, iluminación y configuración del entorno, su funcionamiento continúa apoyándose en el sistema de Terrain nativo del motor. Esta herramienta permitió acelerar la creación de los escenarios sin modificar el flujo de trabajo propio de Unity, facilitando posteriormente la integración de los diferentes recursos del videojuego.

Para la construcción del interior del segundo nivel se evaluaron dos alternativas: utilizar los recursos 3D básicos proporcionados por Unity y personalizar sus materiales y texturas, o emplear un paquete especializado de assets arquitectónicos. Se optó por la segunda alternativa debido a la rapidez de implementación y a la posibilidad de reutilizar elementos modulares durante la construcción de los escenarios. Así, se usó un paquete de recursos desarrollado por Finward Studios, compuesto por paredes, pisos, techos, elementos arquitectónicos y diversos objetos decorativos (props). La utilización de este tipo de assets permitió disponer de una biblioteca consistente de recursos reutilizables, reduciendo considerablemente el tiempo dedicado al modelado de elementos estructurales y facilitando la construcción modular de los escenarios.

Para la implementación de efectos visuales se empleó un paquete desarrollado por VEFECTS, el cual incorpora partículas y efectos prefabricados compatibles con Unity. Estos recursos fueron utilizados para complementar diferentes mecánicas y mejorar la presentación visual del videojuego, evitando la necesidad de desarrollar cada efecto desde cero.

Adicionalmente, el proyecto utilizó herramientas propias del ecosistema de Unity como Cinemachine que a comparación del componente base «Camera», integra muchas mejoras y opciones para un control avanzado de la cámara como el uso de track and dolly, las zonas muertas de seguimiento, además de que es una excelente herramienta para la creación de cinemáticas o escenas cinematográficas (cut scenes) dentro del motor, con la ventaja técnica de que no requieren la escritura de código adicional para su funcionamiento[7] . Además del uso TextMesh Pro, para la representación de texto dentro de la interfaz gráfica. Cinemachine permitió desacoplar el comportamiento de la cámara respecto a la lógica del jugador, mientras que TextMesh Pro proporcionó una representación

tipográfica de mayor calidad y flexibilidad que el sistema de texto tradicional del motor [10].

Todos los assets de terceros utilizados durante el desarrollo fueron obtenidos a través de la Unity Asset Store, respetando los términos establecidos en la licencia estándar de Unity Asset Store (Standard Unity Asset Store EULA). Antes de su incorporación al proyecto se verificó su compatibilidad y en caso de no serlo, se analizó si funcionaba adecuadamente con la versión de Unity empleada, el Universal Render Pipeline (URP) y la arquitectura general del sistema, con el fin de garantizar su correcto funcionamiento durante la integración.

La selección de estas herramientas respondió al objetivo de concentrar el esfuerzo de desarrollo en la integración de sistemas y mecánicas del videojuego, reutilizando recursos especializados cuando su implementación desde cero no aportaba un valor significativo al proyecto.

2.4.7. Control de versiones

Finalmente, durante el desarrollo del proyecto se empleó Git como sistema de control de versiones.

Se analizó varias opciones, entre ellas el uso del control de versiones nativo de unity(Unity Version Controller) y opciones de almacenamiento en la nube como la de google, al final se decidió por el uso de Git y su extensión GitHub desktop debido a la facilidad de uso del software, su familiaridad con el mismo y el dominio y popularidad que tiene en comparación a sus competidores[26]. El uso de Git permitió mantener un historial de cambios sobre el proyecto, facilitar el trabajo colaborativo entre los integrantes del equipo y reducir el riesgo de pérdida de información durante el proceso de integración de

nuevas mecánicas, modelos y escenarios [14] [13].

Su utilización también permitió revertir modificaciones cuando una integración afectaba el funcionamiento general del proyecto, favoreciendo un desarrollo incremental y más seguro.

2.5. Modelos, estándares o arquitecturas

2.5.1. Arquitecturas de software

Antes de definir la estructura del proyecto se analizaron diferentes arquitecturas utilizadas en el desarrollo de videojuegos, con el propósito de seleccionar una alternativa que facilitara la integración de recursos provenientes de distintas áreas del equipo y permitiera mantener el proyecto conforme aumentara su complejidad.

Una de las primeras alternativas consideradas fue la arquitectura monolítica, caracterizada por concentrar la mayor parte de la lógica del sistema en un conjunto reducido de componentes altamente dependientes entre sí [12]. Aunque este enfoque simplifica el desarrollo inicial de proyectos pequeños, conforme aumenta el número de mecánicas y sistemas resulta más difícil mantener el código, localizar errores e incorporar nuevas funcionalidades sin afectar otros módulos del proyecto [9].

Posteriormente se analizó la arquitectura basada en componentes, ampliamente utilizada en motores de videojuegos modernos. Gregory explica que este enfoque organiza el sistema mediante componentes independientes que cumplen responsabilidades específicas y pueden combinarse para construir comportamientos complejos, favoreciendo la reutilización y reduciendo el acoplamiento entre módulos [14]. Esta arquitectura coincide además con la filosofía de funcionamiento de Unity, donde los objetos del juego se

construyen mediante GameObjects a los que se agregan distintos componentes funcionales [27].

Como complemento a esta arquitectura también se estudió el uso de una arquitectura basada en administradores (Managers), práctica ampliamente utilizada en proyectos desarrollados con Unity, como es el caso del proyecto RPG de Navarro Estrellas. Este enfoque consiste en centralizar determinadas responsabilidades dentro de componentes especializados, por ejemplo, para controlar escenas, audio, interfaces, enemigos o sistemas específicos del juego. En lugar de que cada objeto gestione de manera independiente la información compartida, los Managers actúan como puntos de coordinación entre los diferentes sistemas, reduciendo las dependencias directas y facilitando el mantenimiento del proyecto, los principios de separación de responsabilidades y organización modular descritos por Robert Nystrom respaldan este tipo de organización [24][9].

Después del análisis realizado, se decidió adoptar una arquitectura basada en componentes, complementada con el uso de Managers para coordinar los principales sistemas del videojuego. Esta combinación permite mantener separadas las responsabilidades de cada componente mientras la comunicación entre mecánicas, interfaz, audio y escenarios permanece organizada mediante administradores y eventos, facilitando tanto la integración progresiva como el mantenimiento posterior del proyecto.

2.5.2. Patrones de diseño

Además de la arquitectura general, se analizó el uso de patrones de diseño como soluciones reutilizables a problemas recurrentes de ingeniería de software. Nystrom destaca que estos patrones son fundamentales para evitar que la base de código se convierta en una estructura entrelazada y compleja, facilitando la limpieza y mantenibilidad del proyecto conforme éste crece [9].

En este sentido, se priorizaron patrones orientados a la separación de responsabilidades. Garcia Ruiz introduce diversos patrones de diseño aplicados al desarrollo de videojuegos en Unity en su obra «Sobreviviendo como desarrollador indie en Unity». Entre ellos el Patrón de Componentes (Component Pattern) actúa como el pilar fundamental en Unity, permitiendo que las entidades (GameObjects) funcionen como colecciones de piezas independientes que ejecutan funciones específicas sin acoplarse rígidamente al resto del sistema. Complementariamente, el Patrón de Estado (State Pattern) proporciona una estrategia para gestionar comportamientos complejos de personajes e inteligencia artificial de forma segmentada, permitiendo transiciones lógicas entre estados como «atacar» o «deambular» de manera organizada. Para la administración centralizada de recursos y la comunicación entre objetos, resulta vital el patrón Singleton. Este diseño restringe la creación de objetos a una instancia única con acceso global, lo cual es la arquitectura estándar para implementar controladores críticos (Managers) de audio, entrada y lógica de juego persistente [28].

Asimismo, para la optimización del rendimiento, se analizó el patrón de Grupo de Objetos (Object Pool), Akenine-Möller y sus colaboradores recomiendan evitar la creación y destrucción continua de objetos dentro del ciclo principal de ejecución, proponiendo la reutilización de instancias previamente reservadas en memoria para disminuir la carga del recolector de basura y mejorar el desempeño del sistema [18], este es un proceso que mitiga la carga sobre el procesador y el recolector de basura al reutilizar instancias de una colección pre asignada en memoria, evitando el coste de crear y destruir objetos constantemente durante la ejecución.

En InkFall, estos principios sirven como referencia técnica para organizar la lógica mediante administradores especializados y una distribución modular de las mecánicas. Esta

integración permite que los diversos sistemas funcionen de manera coordinada pero desacoplada, asegurando una arquitectura escalable y eficiente.

2.5.3. Estándares de desarrollo en Unity

El propio motor Unity establece una serie de prácticas recomendadas para organizar los proyectos mediante GameObjects, Componentes y Prefabs. Según la documentación oficial de Unity Technologies, cada GameObject puede incorporar diferentes componentes especializados que encapsulan funcionalidades independientes, permitiendo construir comportamientos complejos a partir de módulos reutilizables [27].

Asimismo, Unity recomienda el uso de Prefabs para reutilizar objetos configurados previamente, mantener la consistencia entre escenas y facilitar la actualización simultánea de múltiples instancias [29].

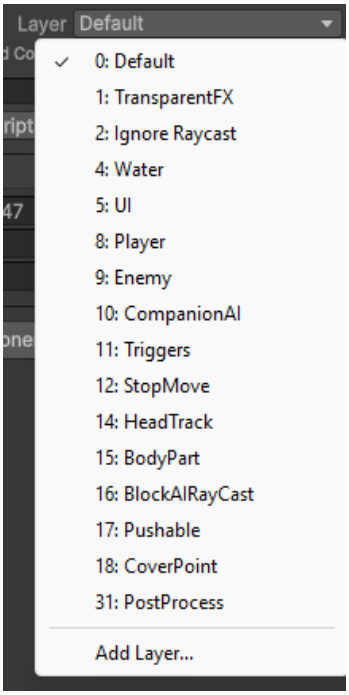
Para asegurar la escalabilidad técnica de un proyecto como InkFall, es vital adoptar una organización sistémica desde el inicio del desarrollo. Autores como Alan Thorn y Nathaly Amaya y sus colaboradores con «Análisis de una experiencia de aprendizaje basada en juegos digitales » mencionan que una adecuada selección y organización de los elementos que conforman un videojuego constituye un factor clave para facilitar la toma de decisiones durante el desarrollo y, al mismo tiempo, evitar el desperdicio de recursos, favoreciendo una implementación más eficiente y un mejor aprovechamiento de los componentes del proyecto, esto implica mantener una estructura de carpetas estricta para categorizar mallas, texturas, audio y scripts.

En este flujo de trabajo, el empleo de nombres adecuado en la jerarquía es crucial, un objeto no debe identificarse con términos genéricos como «Cube01», sino por su función y tipo, un ejemplo es el nombre que reciben los huesos de animación del personaje «Rig:RightLeg», permitiendo gestionar la escena de forma ágil sin depender de la

visualización constante en el editor. Complementariamente, como menciona el manual de UI Toolkit oficial de Unity, el uso de etiquetas (tags) y capas (layers) resulta esencial para categorizar objetos y optimizar procesos técnicos como el trazado de rayos o la detección selectiva de colisiones [31].

Ilustración 1

Lista de capas (Layers) en unity



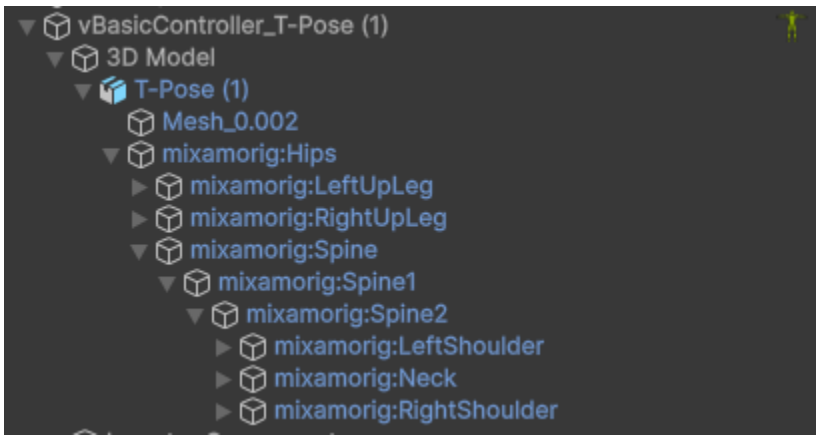
Nota: Elaboración propia

Desde la perspectiva de la ingeniería, Navarro Estrella señala que Unity organiza los elementos mediante una estructura jerárquica en forma de árbol, como se observa en la ilustración 2. Esta organización permite agrupar objetos relacionados bajo un mismo GameObject padre, facilitando la administración de las transformaciones, la organización de la escena y la coordinación de componentes que pertenecen a una misma entidad del videojuego [24]. Esta estructura también simplifica el mantenimiento del proyecto, ya que posibilita gestionar conjuntos de objetos como una única unidad lógica cuando la mecánica así lo requiere. Finalmente, para la agrupación y optimización de escenas, Alan Thorn

recomienda emplear GameObjects raíz vacíos como contenedores de los diferentes módulos del entorno, siguiendo un enfoque de desarrollo modular. esta práctica facilita la manipulación espacial de secciones completas del nivel y permite aplicar,si las características de los objetos lo permiten, técnicas de procesamiento por lotes estáticos (static batching). Esta técnica técnica consolida múltiples objetos bajo una sola operación de renderizado, reduciendo las llamadas de dibujado (draw calls) y garantizando la fluidez en escenarios con alta densidad de elementos [13].

Ilustración 2

Objeto «padre» y sus «hijos»



Nota: Elaboración propia

2.6. Marco conceptual

- Motor de videojuegos: es un software o kit de desarrollo que permite a los usuarios emplear componentes reutilizables como: físicas, renderizado, audio e inteligencia sin tener que reescribirlos para cada proyecto, agrupa todos los subsistemas que operan de forma coordinada a través de un bucle de juego. Su función técnica es gestionar tareas computacionales de bajo nivel para que el desarrollador se enfoque en el diseño y la interactividad [16][28].

- Integración de Sistemas: Proceso mediante el cual diferentes módulos de software y recursos multimedia se coordinan para funcionar de forma conjunta dentro de un videojuego. En Unity, este proceso se apoya en un orden definido de ejecución que sincroniza la lógica del juego, la simulación física y el renderizado, permitiendo la interacción coordinada entre los distintos subsistemas [32]. Asimismo, Paul Barry y David Griffiths con su obra «Head First Programming» principios de programación como la encapsulación favorecen que cada módulo mantenga responsabilidades claramente definidas y se comunique mediante interfaces estructuradas, reduciendo el acoplamiento y facilitando la integración del sistema [33].
- Assets: Es cualquier archivo de datos o recursos multimedia utilizado en un proyecto, pueden ser mallas 3D, texturas, audios, animaciones, materiales y scripts. Constituyen la unidad básica de información que se importa en el editor y que el motor procesa para construir el mundo virtual [26].
- GameObjects: Entidad básica de Unity que representa cualquier objeto presente en una escena. Su funcionalidad depende de los componentes que tenga asociados, pudiendo representar elementos visuales, físicos, de interfaz o lógica del juego [27].
- Componente: Unidad funcional que se incorpora a un GameObject para proporcionarle una característica o comportamiento específico, como físicas, interacción, animación o lógica de juego. En Unity, los componentes personalizados suelen heredar de la clase MonoBehaviour, lo que les permite integrarse con el ciclo de ejecución del motor. Frank Lanzinger destaca que este modelo basado en componentes favorece el desarrollo modular y la incorporación progresiva de nuevas mecánicas [34], mientras que García Ruiz señala que permite desacoplar la

lógica de los objetos y reutilizar funcionalidades sin modificar la estructura general del proyecto [28].

- Prefab: Es un recurso que permite almacenar un GameObject configurado con todos sus componentes, valores y objetos hijos como una plantilla reutilizable. Actúa como un «molde» técnico que facilita la creación de múltiples instancias idénticas en diferentes escenas, permitiendo realizar actualizaciones globales sobre todas ellas de forma simultánea. Son herramientas esenciales para la construcción modular de entornos y garantizar la consistencia visual y lógica en el diseño de niveles [29].
- Escenas (Scenes): Corresponde a una unidad individual de un proyecto que contiene los entornos, menús y niveles específicos del videojuego. Puede considerarse como un archivo único donde se disponen los obstáculos, personajes y decoraciones para construir el juego piezas por piezas. Al ejecutarse, el motor carga la información de escena en una jerarquía de memoria para procesar y renderizar las entidades activas en cada ciclo [28].
- Evento: Se define como un cambio de estado dentro del sistema originado por una acción del usuario, la interacción entre objetos o el transcurso del tiempo, los eventos permiten que los diferentes componentes de una aplicación respondan a un suceso determinado sin necesidad de mantener una comunicación directa entre ellos [16]. En Unity, este mecanismo se implementa mediante funciones del ciclo de vida, delegados y la API UnityEvent [35].
- NavMesh: Es una «malla de navegación» técnica que define las superficies transitables dentro del mundo virtual para los agentes controlados por el sistema. Constituye la base necesaria para implementar algoritmos de enrutamiento y

seguimiento de rutas permitiendo que los personajes no jugables (NPC) se desplacen evitando obstáculos de forma inteligente. Su correcta configuración en la escena es esencial para que la inteligencia artificial pueda interactuar con el entorno espacial de manera convincente [25].

- Dificultad dinámica: Consiste en calibrar continuamente los desafíos del sistema basándose en las habilidades y el comportamiento del usuario durante la partida. Este enfoque técnico busca mantener al jugador en su «zona de desarrollo próximo». evitando la frustración por retos imposibles o aburrimiento por falta de estímulos. Se implementa mediante el ajuste en tiempo real de variables de la IA, la frecuencia de recursos o la reactividad del entorno virtual [30] [13].
- Bucle del juego (Game Loop): Es el patrón fundamental que rige el flujo cíclico de ejecución de un videojuego, procesando eventos y respuestas calculadas en tiempo real. En cada ciclo, el bucle actualiza las entradas del jugador, computa la lógica física y de la IA, y finalmente renderiza el resultado visual en la pantalla. Esta secuencia lógica garantiza que el mundo virtual funcione como un sistema determinista, preservando la ilusión de fluidez y movimiento constante [14].
- Draw Call:Instrucción enviada por la CPU a la GPU para representar un objeto en pantalla. Un número elevado de draw calls incrementa la carga de procesamiento, por lo que Unity recomienda reducirlas mediante técnicas como batching, instancing o la reutilización de materiales compartidos [37].

2.7. Decisiones técnicas

2.7.1. Integración de assets de terceros

Durante el desarrollo de InkFall se decidió aprovechar assets desarrollados por terceros para acelerar la implementación de funcionalidades que no constituían el objetivo principal del proyecto. Entre estos recursos se encuentran sistemas de movimiento del personaje, paquetes de construcción de escenarios, herramientas para la generación de terrenos y efectos visuales.

Esta decisión se fundamenta en el criterio planteado por Ian Millington en «Game Physics Engine Development», quien señala que reutilizar soluciones previamente desarrolladas permite reducir el tiempo de implementación y aprovechar sistemas ampliamente probados, siempre que su integración se realice de forma controlada dentro de una arquitectura modular [36].

La decisión no consistió únicamente en importar estos recursos al proyecto, sino en seleccionar e incorporar aquellos elementos que resultaban compatibles con la arquitectura definida para InkFall. Cada asset fue evaluado previamente para determinar si era compatible con la versión de Unity utilizada, con el Universal Render Pipeline (URP) y con la estructura general del videojuego. Posteriormente, únicamente se incorporaron los componentes necesarios, descartando escenas de ejemplo, scripts y recursos que no aportan al desarrollo de InkFall.

Este enfoque permitió reducir los tiempos de desarrollo sin comprometer la organización de la arquitectura del proyecto, evitando incorporar recursos innecesarios y facilitando el mantenimiento y la evolución del sistema.tando incorporar dependencias innecesarias y facilitando el mantenimiento del sistema.

2.7.2. Integración incremental

La integración de los diferentes componentes se realizó de forma incremental. En lugar de incorporar todos los recursos simultáneamente, cada sistema fue integrado y validado de manera independiente antes de continuar con la siguiente etapa del desarrollo.

Este procedimiento permitió detectar incompatibilidades entre assets, errores de configuración y problemas de rendimiento en etapas tempranas del proyecto. Además, facilitó la incorporación progresiva de nuevos recursos provenientes de las diferentes áreas del equipo, reduciendo el riesgo de afectar componentes previamente implementados.

La estrategia también simplificó el proceso de depuración, ya que cualquier error podía asociarse con la última integración realizada.

2.7.3. Uso de Administradores (Managers)

Una de las principales decisiones de arquitectura fue centralizar determinadas funcionalidades mediante administradores o Managers. En lugar de distribuir la lógica de control entre múltiples objetos de la escena, se crearon componentes responsables de coordinar sistemas específicos, como la gestión de escenas, los rompecabezas, la interfaz o determinadas mecánicas del juego.

Esta decisión permitió disminuir el acoplamiento entre componentes, facilitar la reutilización de código y mantener responsabilidades claramente definidas para cada sistema. Asimismo, simplificó la incorporación de nuevas funcionalidades, ya que las modificaciones se concentraban en un único punto de control sin afectar directamente al resto de los objetos de la escena.

2.7.4. Organización modular del proyecto

Durante la integración se procuró mantener una arquitectura modular, donde cada componente cumpliera una responsabilidad específica dentro del proyecto. Los sistemas

relacionados con el jugador, los enemigos, la interfaz, el audio y los escenarios permanecieron separados, comunicándose únicamente cuando era necesario mediante referencias controladas, eventos o administradores.

Esta decisión permitió mantener una arquitectura escalable y facilitó el mantenimiento del proyecto, ya que las modificaciones realizadas sobre un sistema tienen un impacto reducido sobre el resto de los componentes.

2.7.5. Organización del proyecto por niveles

Debido a que InkFall es un videojuego narrativo multigénero, cuya experiencia se divide en diferentes libros o niveles, se decidió organizar gran parte de los recursos siguiendo esa misma estructura.

Los modelos 3D, texturas, efectos visuales, elementos de interfaz, scripts y demás recursos fueron agrupados según el nivel al que pertenecían, manteniendo además carpetas independientes para los recursos compartidos entre todo el proyecto. Esta organización facilita localizar rápidamente los archivos relacionados con cada escenario, reduce la posibilidad de modificar recursos incorrectos y simplifica el mantenimiento conforme aumenta el tamaño del proyecto.

2.7.6. Estrategia de validación

Como parte del proceso de integración se estableció una estrategia de validación continua. Cada nuevo componente incorporado debía superar una serie de verificaciones antes de considerarse integrado al proyecto.

Estas validaciones incluyeron la revisión de errores en la consola de Unity, comprobación de referencias desde el Inspector, pruebas funcionales de la mecánica implementada y análisis básico del rendimiento mediante las herramientas Stats y Profiler.

Solo después de verificar el correcto funcionamiento del nuevo recurso se continuaba con la integración de componentes adicionales.

La aplicación sistemática de este procedimiento permitió detectar problemas de integración de forma temprana y mantener la estabilidad general del proyecto durante todo el desarrollo.

Capítulo III

3. Manual del usuario

3.1. Introducción

El presente Manual de Usuario tiene como finalidad proporcionar la información necesaria para la correcta instalación, ejecución y utilización del prototipo del videojuego Inkfall. Este documento está dirigido a cualquier usuario que desee interactuar con el sistema, ofreciendo una guía clara sobre los requisitos del software, los controles, las mecánicas principales y el funcionamiento general del juego.

Inkfall es un videojuego narrativo desarrollado en el motor Unity como parte de un proyecto académico. El prototipo está compuesto por dos niveles que presentan diferentes mecánicas de juego y un sistema de dificultad dinámica que modifica determinadas condiciones de la partida en función del desempeño del jugador.

Este manual describe paso a paso el proceso para ejecutar el videojuego, conocer la interfaz, utilizar los controles y comprender las principales mecánicas implementadas.

3.2. Requisitos del sistema

Tabla 3

Requisitos mínimos del sistema

Componente	Requisitos
Sistema Operativo	Windows 10 de 64 bits
Procesador	Intel Core i5 de 8va generación o AMD Ryzen 5 equivalente
Memoria RAM	8 GB
Tarjeta gráfica	NVIDIA GTX 1650 o AMD Radeon RX 570, con 4 GB de VRAM
Almacenamiento	8 GB disponibles

Nota: Elaboración propia

3.3. Instalación

Para ejecutar el videojuego Inkfall es necesario contar con un equipo que cumpla con los requisitos mínimos del sistema y seguir el procedimiento de instalación descrito a continuación.

3.3.1. Descarga del juego

Obtenga el archivo comprimido que contiene el videojuego desde el medio proporcionado por los desarrolladores (memoria USB, enlace de descarga o repositorio institucional).

3.3.2. Descompresión de los archivos

Ubique el archivo descargado y descomprímalo en la carpeta de su preferencia utilizando el explorador de archivos de Windows o cualquier software de descompresión compatible.

3.3.3. Ejecución del juego

Abra la carpeta del proyecto y ejecute el archivo denominado **Inkfall.exe** para iniciar el videojuego.

3.3.4. Inicio del videojuego

Una vez ejecutado el archivo, se mostrará la pantalla principal del videojuego, desde la cual el usuario podrá acceder al menú principal y comenzar la experiencia de juego.

3.4. Interfaz del juego

La interfaz de usuario de Inkfall fue diseñada para ofrecer una navegación clara e intuitiva desde el inicio del videojuego. Su estructura permite acceder de forma organizada a las diferentes funciones disponibles antes de comenzar la partida, manteniendo una distribución uniforme de los elementos visuales y una identidad gráfica coherente con la propuesta artística del proyecto.

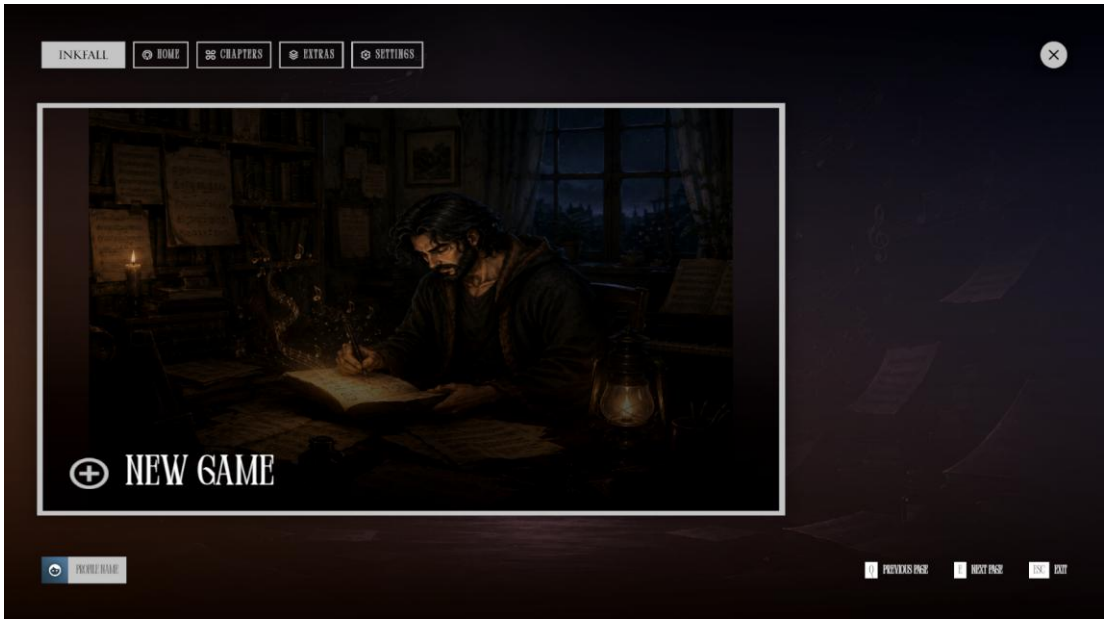
Al ejecutar el videojuego, el usuario visualizará una pantalla de bienvenida con el logotipo de Inkfall y el mensaje "Press Any Key to Start". Para acceder al menú principal, únicamente deberá presionar cualquier tecla del teclado.

3.4.1. Menú principal

El menú principal constituye el centro de navegación del videojuego y reúne las principales opciones disponibles para el usuario. La interfaz está organizada mediante pestañas ubicadas en la parte superior, permitiendo cambiar entre las diferentes secciones sin abandonar el menú principal.

Ilustración 3

Menú principal



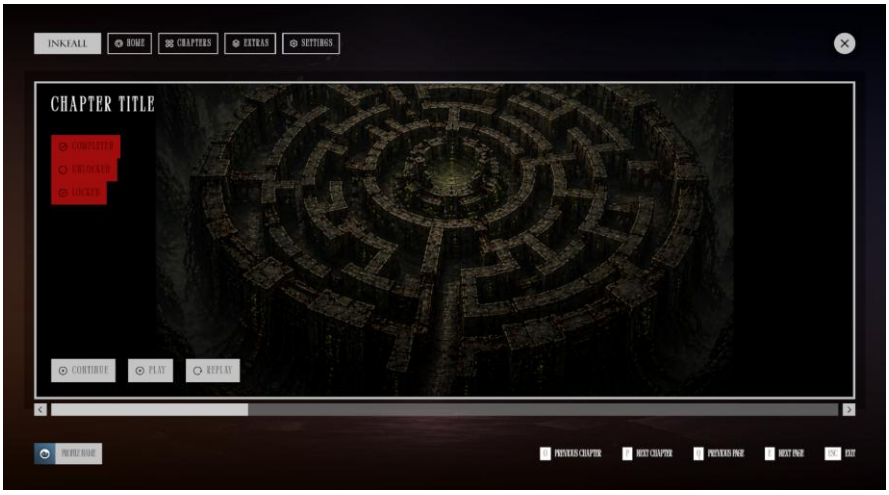
Nota: Menú de inicio del juego, Elaboración propia

En la ilustración 3 se visualiza la pestaña Home donde se presenta la opción New Game, desde donde el usuario puede iniciar una nueva partida y acceder al contenido del videojuego. Además, en la parte inferior se muestran los controles disponibles para desplazarse entre las páginas del menú o salir de la aplicación.

En la ilustración 4 se tiene la pestaña Chapters (capítulos) en donde se permite visualizar los capítulos disponibles dentro del prototipo. Cada capítulo incluye una imagen representativa, una breve descripción y las opciones para iniciar o repetir el nivel correspondiente.

Ilustración 4

Menú, sección de capítulos



Nota: Menu de selección de capítulos, Elaboración propia

La pestaña Extras contiene información complementaria del videojuego, como los créditos.

Ilustración 5

Menú, sección de extras

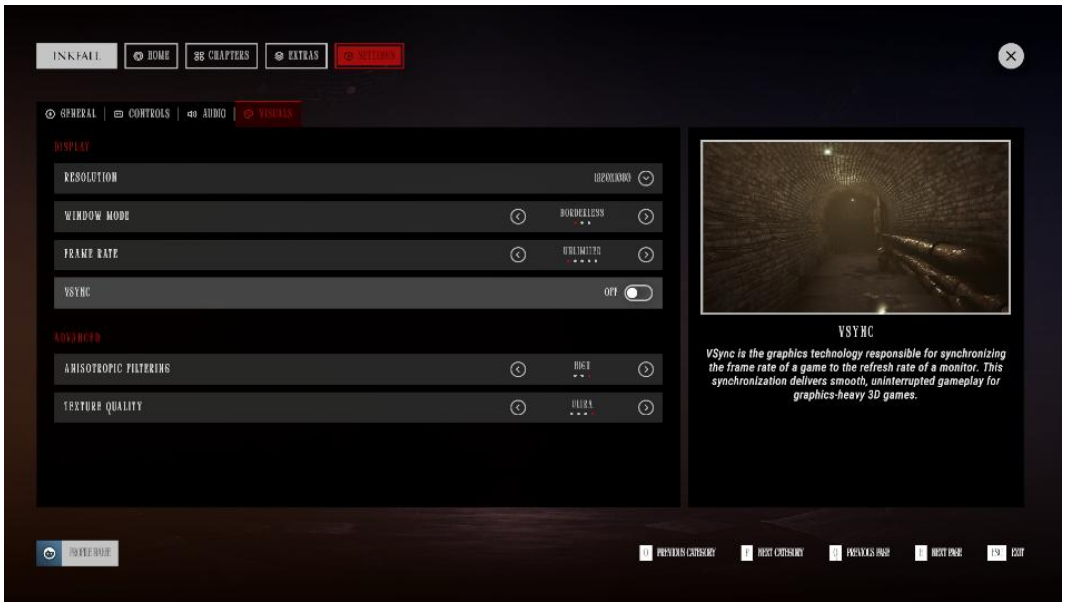
Nota: Menú de extras, Elaboración propia

Finalmente, la pestaña Settings permite modificar diferentes parámetros del videojuego. Dentro de esta sección el usuario puede acceder a opciones generales,

configuración de controles, audio, aspectos visuales y funciones de accesibilidad, adaptando la experiencia de juego de acuerdo con sus preferencias.

Ilustración 6

Menú sección de ajustes



Nota: Menú de configuraciones, Elaboración propia

3.4.2. Pantalla del juego

Durante la partida se mostrará la interfaz principal (HUD), la cual proporciona al jugador la información necesaria para el desarrollo del juego. Su diseño mantiene la identidad visual de InkFall, utilizando elementos gráficos inspirados en el logotipo del videojuego. El indicador de vida está representado por el círculo presente en el logotipo, el cual refleja el estado de salud del personaje durante la partida. En el centro se encuentra una gota que indica el nivel de dificultad seleccionado, permitiendo al jugador identificar esta información de forma rápida sin afectar la experiencia visual.

En el Capítulo 1, la interfaz incorpora información adicional relacionada con el sistema de combate, mostrando el arma equipada, el inventario de armas y la cantidad de

munición disponible. Estos elementos permiten al jugador administrar sus recursos durante los enfrentamientos.

Por otra parte, en el Capítulo 2, la interfaz se adapta a la mecánica propia de ese escenario. Debido a que la jugabilidad está basada en el uso de un encendedor como elemento principal, los indicadores relacionados con las armas son reemplazados por la información necesaria para esta mecánica, manteniendo una interfaz clara y acorde con la experiencia del jugador.

3.5. Controles

Tabla 4

Controles y su botón del teclado

Acción	Tecla
Mover	WASD
Cámara	Mouse
Disparar	Click izquierdo
Apuntar	Click derecho
Saltar	Espacio
Interactuar	E
Recargar	R
Habilidad especial	F
Cambiar arma	Rueda del mouse
Abrir puerta	Click izquierdo
Recargar	R

Nota: Elaboración propia

3.6. Como jugar

Al iniciar Inkfall, el jugador tomará el control del protagonista en 2 escenarios distintos. El objetivo es explorar los escenarios, interactuar con los elementos del entorno y superar los desafíos planteados en cada nivel. A medida que el jugador progresa, encontrará mecánicas diferentes que requieren combinar exploración, observación y habilidad para completar cada sección del prototipo.

3.6.1. Inicio

Al ejecutar el juego se mostrará el menú principal, desde donde el usuario podrá iniciar una nueva partida o seleccionar uno de los 2 niveles. Una vez seleccionada la opción correspondiente, el juego cargará el nivel y el jugador aparecerá en el punto inicial del escenario.

Durante la partida, el jugador podrá desplazarse libremente utilizando los controles establecidos e interactuar con los diferentes elementos presentes en el entorno para avanzar en la experiencia.

3.6.2. Primer nivel

El primer nivel está orientado a las mecánicas de combate y exploración. El jugador deberá recorrer un laberinto mientras enfrenta enemigos controlados por inteligencia artificial.

Para avanzar es necesario:

- Explorar el escenario para encontrar la ruta correcta
- Utilizar el arma para eliminar a los enemigos que bloquean el camino
- Evitar recibir demasiado daño durante los enfrentamientos
- Continuar avanzando hasta alcanzar la salida del nivel

Si la vida del jugador llega a cero, la partida finalizará y el juego se reiniciará aplicando el sistema de dificultad dinámica.

3.6.3. Segundo nivel

El segundo nivel cambia completamente la dinámica del juego. En este escenario, la supervivencia depende del manejo adecuado de un encendedor que actúa como la principal fuente de luz del jugador, así el nivel transforma el juego a un nivel de terror.

Durante el recorrido, el encendedor puede apagarse debido a movimientos bruscos ya sea de la cámara o del personaje. Cuando esto ocurre, el jugador deberá encenderla lo antes posible para evitar permanecer en la oscuridad durante un tiempo prolongado.

Mientras explora el escenario, el jugador deberá recorrer las habitaciones, interactuar con los objetos necesarios y encontrar la ruta y la llave que le permita completar el nivel evitando ser alcanzado por la amenaza presente en el escenario.

3.6.4. Sistema de dificultad

Inkfall incorpora un sistema de dificultad dinámica que aumenta progresivamente el comportamiento del juego conforme el jugador acumula intentos fallidos.

Cada vez que el jugador muere, el sistema registra el intento y aplica cambios en determinados elementos del juego. Estos cambios pueden afectar el comportamiento de los enemigos, las condiciones del escenario o las mecánicas disponibles, generando una experiencia diferente en cada nuevo intento.

Este sistema forma parte de la propuesta principal del prototipo y busca demostrar una alternativa a los modelos tradicionales de dificultad basados únicamente en el incremento de parámetros numéricos o en la disminución de la dificultad por fracaso para hacer más fácil el nivel.

3.7. Créditos

Tabla 5

Integrantes del proyecto y su rol

Integrante	Rol
Joan Dután	Programador principal
Alejandro Ortiz	Diseño y arte visual
Juan José Bacuilima	Modelado 3D y animación
Alexander Abad	Integración de sistemas

Nota: Elaboración propia

Capítulo IV

4. Manual del programador

4.1 Arquitectura general del sistema

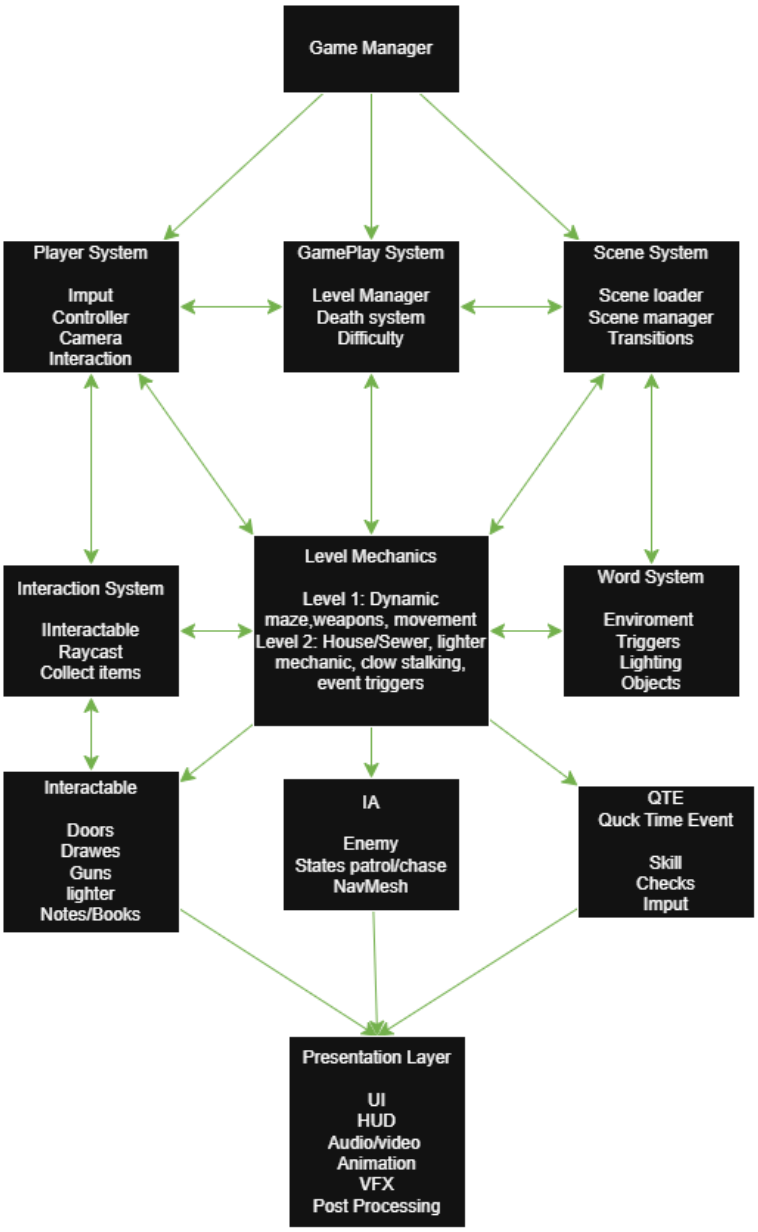
Esta sección presenta la arquitectura general del videojuego InkFall, describiendo la forma en que se organizan los principales subsistemas dentro del entorno de desarrollo. Su propósito es ofrecer una visión global de la estructura del proyecto antes de detallar la implementación técnica de cada componente en las secciones posteriores del manual.

La arquitectura general contempla la integración de los recursos generados por las diferentes áreas del equipo de desarrollo, incluyendo programación, modelado 3D, animación, diseño de niveles, interfaz de usuario y audio. Todos estos elementos convergen dentro del proyecto de Unity mediante una estructura organizada que facilita su incorporación progresiva y su funcionamiento conjunto. Las siguientes secciones desarrollan con mayor detalle los elementos que conforman esta arquitectura, incluyendo las tecnologías empleadas, la organización técnica del proyecto, la arquitectura de clases y componentes, la integración de recursos y la comunicación entre los distintos sistemas del videojuego.

El videojuego fue desarrollado sobre el motor Unity siguiendo una organización mediante módulos especializados que trabajan de forma coordinada para controlar el comportamiento del jugador, las mecánicas propias de cada nivel, los enemigos, la interacción con el entorno y los elementos de presentación.

Ilustración 7

Arquitectura técnica



Nota: Elaboración propia

El Game Manager actúa como el núcleo orquestador del proyecto, coordinando de forma modular los tres pilares principales de la arquitectura: Player System, GamePlay System y Scene System. Para mantener una arquitectura limpia y desacoplada, estos subsistemas se comunican mediante un esquema de señales y eventos bidireccionales. El módulo Player System gestiona la entrada de usuario (input), el control del personaje y la

cámara. Mantiene una conexión directa con el Interaction System para detectar y manipular elementos del escenario (como puertas, cajones u objetos inspeccionables) mediante raycasts o colisiones, sin que el controlador principal del jugador requiera conocer la lógica interna de cada objeto interactivo. Asimismo, se conecta con World y Scene System para interactuar con triggers y entornos. La relación entre Player System y GamePlay System es bidireccional a través de eventos. El jugador proporciona de forma continua su posición, orientación y acciones, lo que permite al sistema de Gameplay evaluar y ejecutar las reglas del nivel actual. Por ejemplo, en la mecánica del encendedor del nivel 2, los movimientos bruscos o el desplazamiento del jugador notificados por el Player System reducen la intensidad de la llama. De esta manera, las reglas del juego responden al comportamiento del usuario sin sobrecargar la lógica del controlador del personaje. El GamePlay System delega la ejecución de reglas concretas en el módulo Level Mechanics, adaptándose a los diferentes niveles del juego: En el nivel 1, gestiona la lógica del laberinto dinámico, la generación de oleadas de enemigos y el control de armas. En el nivel 2, controla los eventos de persecución y la activación del subsistema QTE (Quick Time Event). Cuando el encendedor se apaga, Gameplay invoca el QTE para procesar las pulsaciones del jugador; al completarse con éxito, se envía una señal para restaurar la luz. El subsistema IA System recibe condiciones del GamePlay System y del Player System para alterar los estados de los enemigos (patrulla, persecución, ataque), devolviendo eventos de estado que se usan para para determinar consecuencias como el daño o la derrota del jugador. Finalmente, todos los subsistemas emiten señales hacia la Presentation Layer. Esta capa centraliza la respuesta visual y sonora del proyecto, integrando el trabajo de arte 3D, animación, post-procesado, VFX, audio y HUD/UI en una experiencia unificada.

La arquitectura general también contempla la integración de los recursos generados por las diferentes áreas del equipo de desarrollo, incluyendo programación, modelado 3D, animación, diseño de niveles, interfaz de usuario y audio. Todos estos elementos convergen dentro del proyecto de Unity mediante una estructura organizada que facilita su incorporación progresiva y su funcionamiento conjunto. Las siguientes secciones desarrollan con mayor detalle los elementos que conforman esta arquitectura, incluyendo las tecnologías empleadas, la organización técnica del proyecto, la arquitectura de clases y componentes, la integración de recursos y la comunicación entre los distintos sistemas del videojuego.

4.2 Tecnologías, motores y herramientas utilizadas

El proyecto InkFall utiliza Unity 6 LTS (6000.3.10f1) como motor principal. Durante el desarrollo se seleccionó esta versión por ofrecer estabilidad para proyectos de larga duración, mantener compatibilidad con los paquetes empleados y recibir actualizaciones enfocadas principalmente en la corrección de errores. El proyecto está configurado con Universal Render Pipeline (URP) para equilibrar calidad visual y rendimiento, además de facilitar la compatibilidad entre los assets incorporados.

Como entorno de programación se utilizó Visual Studio Code, ligeramente superior en cuanto a velocidad a su versión base Visual Studio, integrado con Unity para la edición de scripts en C#. Durante el desarrollo proporcionó herramientas de autocompletado, depuración, navegación entre clases y detección temprana de errores.

El sistema de cámaras utiliza Cinemachine, paquete oficial de Unity que permite desacoplar el comportamiento de la cámara respecto al resto de mecánicas del juego. Su incorporación simplificó la configuración del seguimiento del jugador, los cambios de

posición y las transiciones de cámara sin necesidad de desarrollar un controlador personalizado.

La gestión de las entradas del jugador se realiza mediante New Input System, centralizando todas las acciones en un único archivo de Input Actions. Esta organización facilita la modificación o incorporación de nuevos controles sin afectar la lógica de las mecánicas existentes.

Para la representación de texto en la interfaz gráfica se utiliza TextMesh Pro, herramienta integrada en Unity que ofrece una mejor calidad de renderizado y mayores posibilidades de personalización respecto al sistema de texto tradicional. Todos los elementos del HUD, menús y mensajes del videojuego utilizan este sistema.

Durante la construcción de los escenarios se empleó Gaia 6, herramienta utilizada para la generación y edición de terrenos. Aunque Gaia incorpora múltiples utilidades adicionales, en este proyecto se utilizó principalmente como apoyo para la creación del relieve y la configuración inicial del terreno, aprovechando que el asset tiene como núcleo la herramienta nativa de Unity «Terrain» así podemos realizar ajustes específicos según las necesidades de cada nivel.

Los escenarios también incorporan recursos del paquete House Props and Furniture, desarrollado por Finward Studios. Sus elementos fueron reorganizados y adaptados con cambios de texturas para construir los distintos ambientes del videojuego de acuerdo con el diseño establecido para cada nivel.

Los efectos visuales utilizan el paquete de VFX, «Stylized VFX Bundle» de VEFECTS, empleado para incorporar partículas, efectos ambientales y elementos visuales asociados a diferentes mecánicas del juego. Cada efecto fue adaptado para mantener

coherencia con la estética general del proyecto, ya sean colores, tiempo de duración, cantidad de partícula y similares, además de asegurar su compatibilidad con URP.

Para el control de versiones se gestionó mediante Git, complementado con almacenamiento en la nube para compartir el proyecto entre los integrantes del equipo. Esta herramienta permitió mantener un historial de cambios, recuperar versiones anteriores del proyecto y facilitar el trabajo colaborativo durante el proceso de integración.

La validación técnica del proyecto se apoya en las herramientas integradas de Unity. Console permite identificar errores de compilación y referencias faltantes; Profiler se utiliza para analizar el consumo de CPU, memoria y otros recursos; la ventana Stats facilita el seguimiento de indicadores gráficos como FPS, batches y número de triángulos; mientras que NavMesh se emplea para generar las superficies de navegación utilizadas por los personajes controlados por inteligencia artificial.

Tabla 6

Herramientas y dependencias utilizadas durante el desarrollo

Herramienta	Versión	Propósito
Unity	6000.3.10f1 LTS	Motor principal para la integración y desarrollo del videojuego.
New Input System	Paquete oficial de Unity	Gestión de entradas del jugador.
Cinemachine	3.1.7	Paquete oficial de Unity, sistema de cámaras.
TextMesh Pro	Integrado en Unity	Representación de textos e interfaz gráfica.
Gaia 6	4.2.5	Constructor de escenario con el componente Terrain de Unity
House props y furniture	1.2.0	Paquete de objetos e infraestructura de una casa
Stylized VFX Bundle	1.0.1	Paquete de efectos visuales y partículas
Unity Profiler	Integrado en Unity	Evaluación del rendimiento y consumo de recursos.
Unity Console	Integrado en Unity	Identificación y depuración de errores.
NavMesh	Integrado en Unity	Configuración de navegación de personajes con IA.
Git	2.55.0	Control de versiones del proyecto.
Drive	Integrado en Google	Almacenamiento de versiones

Nota: Elaboración propia

4.3 Organización técnica del proyecto

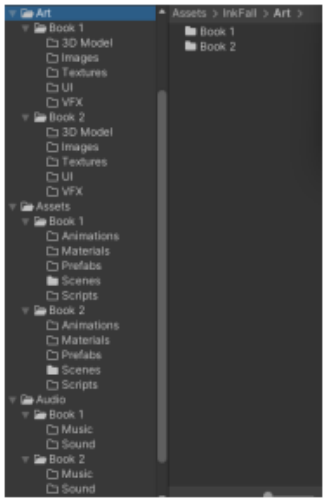
La organización técnica del proyecto InkFall se estableció a partir de una estructura común en Unity, sobre la cual se incorporaron progresivamente los diferentes recursos y sistemas desarrollados para los niveles del videojuego. El proyecto fue configurado utilizando Unity 6000.3.10f1 LTS con Universal Render Pipeline (URP) como sistema de renderizado. Durante la configuración inicial se instalaron mediante el Package Manager las dependencias requeridas para el desarrollo, entre ellas Cinemachine y New Input System, mientras que herramientas como TextMesh Pro, NavMesh, Profiler y Console corresponden a funcionalidades integradas en el motor. Esta configuración permitió

establecer una base técnica común para la posterior incorporación de personajes, escenarios, objetos, sistemas de gameplay, interfaz de usuario y audio.

Como parte de la estructura inicial se definió una organización de carpetas orientada a facilitar la incorporación y administración de los recursos durante las diferentes etapas de desarrollo. Los elementos principales del proyecto se distribuyeron en categorías generales como Art, Assets, Audio, Scripts y Core, las cuales se subdividieron de acuerdo con los niveles o libros que conforman el videojuego. Dentro de estas categorías se separaron los recursos específicos de cada nivel y los componentes compartidos entre todo el proyecto. Por ejemplo, dentro de Art/Book1 se almacenaron los recursos gráficos correspondientes al primer nivel, incluyendo modelos 3D, texturas, imágenes, efectos visuales y elementos de interfaz. Esta organización permitió mantener diferenciados los recursos específicos de cada escenario respecto de los componentes generales del proyecto, facilitando su localización y reduciendo dependencias innecesarias entre niveles.

Ilustración 8

Organización de carpetas y archivos



Nota: Organización de carpetas elaborada en la semana inicial de desarrollo, *Elaboración propia*

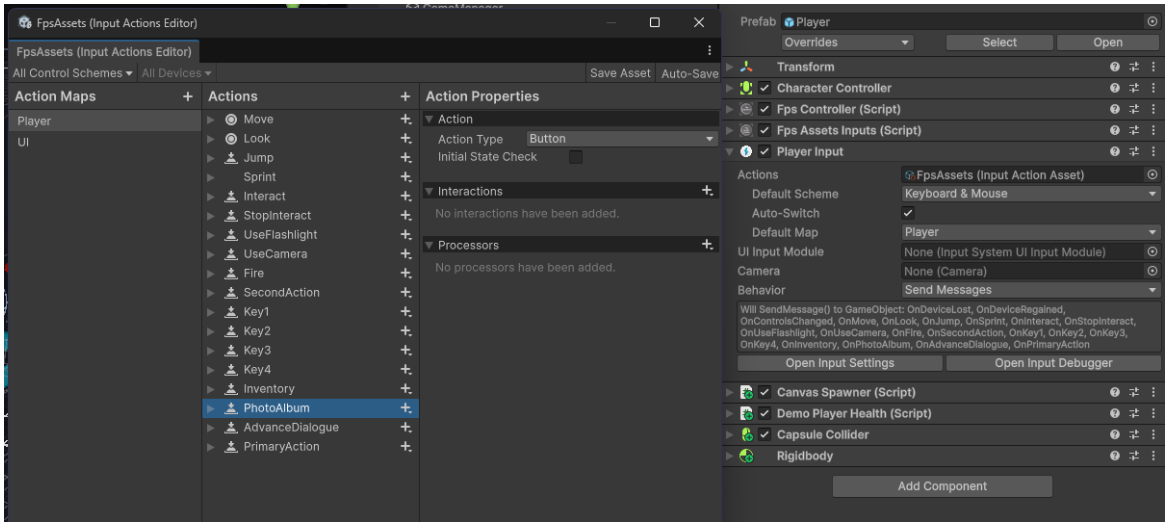
Sobre esta estructura se definió la arquitectura técnica utilizada para la integración de los diferentes sistemas del videojuego. La organización se basó en componentes y

eventos, de manera que las funcionalidades pudieran distribuirse entre los GameObjects y los sistemas correspondientes sin concentrar toda la lógica en un único elemento. Los componentes se utilizaron para añadir comportamientos específicos a los objetos, mientras que los eventos permitieron establecer comunicación entre sistemas cuando una acción del jugador o un cambio de estado requería una respuesta en otros elementos del proyecto. Esta arquitectura se utilizó como base para relacionar los sistemas de gameplay, personajes, objetos interactivos, interfaz, audio y eventos asociados al progreso del videojuego.

Una vez establecida la estructura y arquitectura inicial, se inició la integración de los elementos funcionales del videojuego. La primera etapa correspondió a la incorporación de los personajes dentro del sistema base del proyecto. El objeto Player concentró los componentes necesarios para el control del jugador, incluyendo el sistema de entrada, movimiento y cámara. Las entradas del jugador fueron implementadas mediante New Input System, utilizando un archivo de Input Actions centralizado y referenciado desde el componente Player Input del prefab Player. Esta organización permitió mantener en un único punto las acciones utilizadas por el videojuego y facilitó su vinculación con los scripts responsables del comportamiento del jugador.

Ilustración 9

Input action y su referencia en el player input del prefab Player



Nota: Ventana del Input Action Editor y su referencia dentro del componente player input, Elaboración propia

El sistema de cámara se integró mediante Cinemachine, incorporado al proyecto a través del Package Manager. El objeto Camera se configuró como parte del sistema del Player y se utilizó el componente Cinemachine Camera para controlar el seguimiento y la visualización del personaje. Entre sus parámetros se estableció un campo de visión de 60°, además de las configuraciones relacionadas con la distancia y comportamiento de seguimiento. La separación entre la lógica de movimiento del jugador y la lógica de cámara permitió modificar el comportamiento de visualización sin alterar directamente el sistema de control principal.

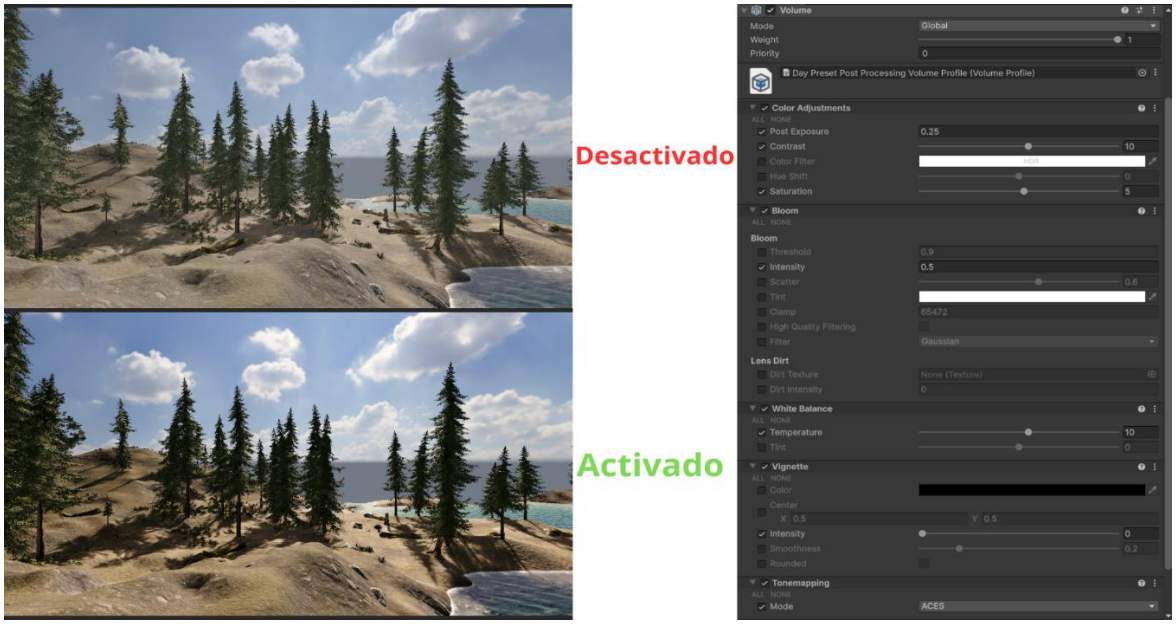
Durante la integración de los personajes también se incorporaron los sistemas de animación, físicas y lógica de interacción requeridos por cada nivel. Los componentes de movimiento y comportamiento se configuraron de manera conjunta con los elementos visuales del personaje para garantizar que sus acciones respondieran correctamente a las entradas del usuario y a las condiciones definidas por el gameplay. Esta etapa permitió

disponer de una primera versión funcional sobre la cual se realizaron pruebas iniciales antes de continuar con la incorporación de nuevos recursos.

Paralelamente a la integración del personaje se inició la incorporación del primer nivel dentro de la estructura base del proyecto. Para la construcción del escenario se utilizó Gaia 6 como herramienta de apoyo para la generación inicial del terreno. Una vez creada la configuración base, el relieve fue ajustado mediante la herramienta Stamper para adaptar la topografía a las necesidades específicas del nivel. Posteriormente se incorporaron los modelos, materiales y demás elementos visuales necesarios, verificando su compatibilidad con Universal Render Pipeline para mantener un comportamiento coherente de los recursos dentro del proyecto. La integración visual de los escenarios también incluyó la configuración de iluminación y postprocesado. El proyecto utilizó el sistema de postprocesado integrado en URP mediante componentes Global Volume, permitiendo establecer perfiles visuales específicos para cada nivel. En el primer nivel se utilizó como punto de partida el perfil proporcionado por Gaia 6 y se realizaron ajustes en parámetros como Bloom y Vignette con el propósito de adaptar la iluminación y mantener la cohesión visual del escenario.

Ilustración 10

Componente Global Volume y comparación, primer nivel

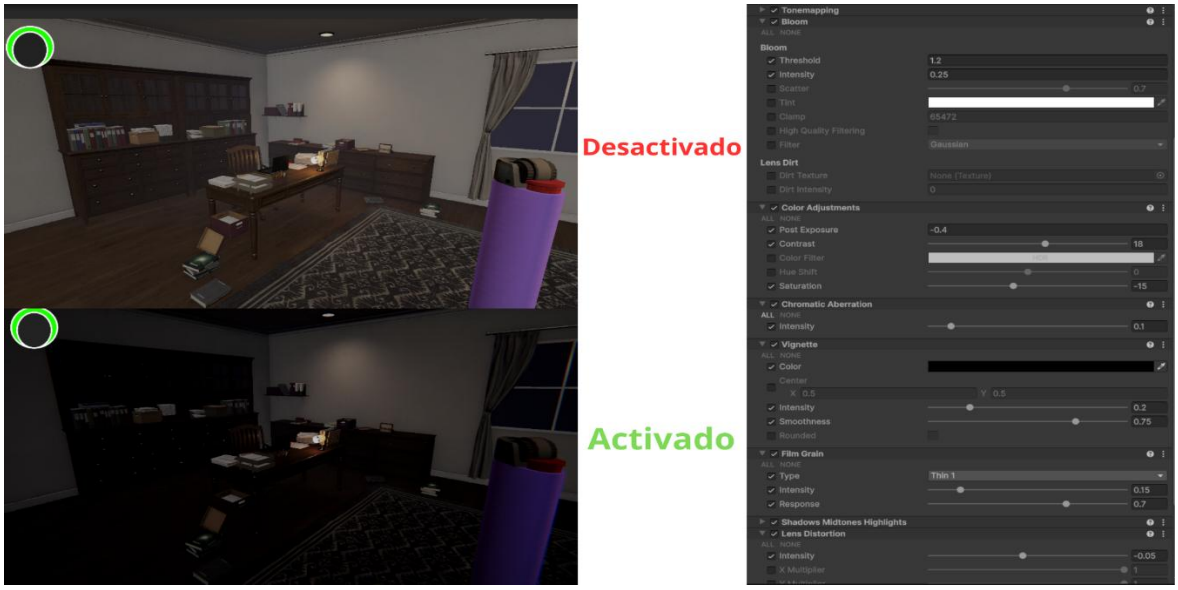


Nota: Comparativa entre el postprocesado activo y desactivado del nivel 1, Elaboración propia

Para el segundo nivel se configuró un perfil de postprocesado independiente, orientado a generar una ambientación más oscura y perturbadora. Para ello se incorporaron efectos de Chromatic Aberration, Lens Distortion, Vignette, Bloom, Film Grain, Tonemapping y Color Adjustments, complementados con una configuración de niebla de baja intensidad integrada en la iluminación del escenario. Esta configuración permitió diferenciar visualmente el nivel y adaptar su presentación a las características de su género y temática.

Ilustración 11

Componente Global Volume y comparación, segundo nivel



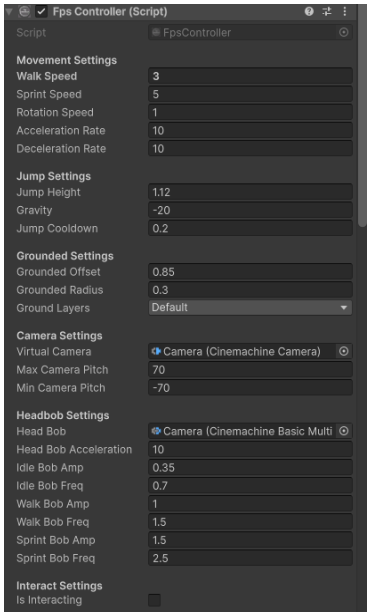
Nota: Comparativa entre el postprocesado activo y desactivado del nivel 2, Elaboración propia

Después de integrar los escenarios se incorporaron los objetos interactivos y los componentes necesarios para establecer su relación con las mecánicas del videojuego. Los objetos fueron configurados mediante componentes específicos de interacción, colisión y comportamiento, vinculándolos con los scripts y eventos correspondientes. De esta forma, los elementos visuales incorporados al escenario dejaron de funcionar como recursos independientes y pasaron a formar parte del sistema de gameplay.

La navegación de los personajes controlados mediante inteligencia artificial fue integrada utilizando NavMesh. Debido a que la navegación depende de la geometría del escenario, durante la incorporación y modificación de paredes, obstáculos y terrenos fue necesario ajustar la superficie navegable y regenerar la malla de navegación para garantizar la correcta generación de rutas de los agentes. De esta manera, la integración de los escenarios contempló no únicamente sus elementos visuales, sino también las condiciones necesarias para su interacción con los sistemas de movimiento y comportamiento.

Ilustración 12

Visualización del componente controller



Nota: Componente Fps Controller y sus variables y referencias modificables, Elaboración propia

La incorporación del segundo nivel siguió la misma estructura general de integración utilizada para el primero, manteniendo separados los recursos específicos de cada escenario y reutilizando los sistemas compartidos del proyecto. Esta organización permitió incorporar las características particulares del segundo nivel sin duplicar innecesariamente los componentes generales. Al mismo tiempo, las configuraciones visuales, de navegación, colisiones y objetos fueron adaptadas a las necesidades particulares del nivel.

Una vez integrados los elementos principales de gameplay, personajes, escenarios y objetos, se incorporaron progresivamente los sistemas complementarios del proyecto. El sistema de audio se integró mediante la incorporación de efectos de sonido y ambientación vinculados a las acciones y eventos generados durante la ejecución del videojuego. Esta relación permitió que la respuesta sonora se produjera en función del estado del gameplay

y de las acciones realizadas por el jugador, en lugar de tratar el audio como un conjunto de recursos independientes.

La interfaz de usuario se integró posteriormente mediante TextMesh Pro para la representación de textos dentro del HUD, menús y mensajes del videojuego. Los elementos de interfaz fueron vinculados a los estados y eventos generados por el sistema de gameplay para actualizar la información mostrada al jugador en función del progreso y las acciones realizadas. Esta integración permitió mantener una relación directa entre la lógica del videojuego y la información presentada en pantalla.

Finalmente, se incorporaron los eventos dinámicos asociados al progreso y reinicio del videojuego. Estos eventos permitieron coordinar cambios de estado que afectan a diferentes sistemas de manera simultánea, incluyendo los procesos relacionados con el reinicio de una partida, la progresión del jugador y las modificaciones previstas en la experiencia de juego. El uso de eventos permitió mantener separada la lógica de los diferentes sistemas y establecer respuestas coordinadas sin depender de una comunicación directa entre todos los componentes involucrados.

Debido a la naturaleza multigénero de InkFall, la organización técnica no se limitó a la integración independiente de cada nivel, sino que requirió mantener una estructura común que permitiera incorporar diferentes mecánicas, escenarios y recursos dentro de un mismo proyecto de Unity. Para ello, los sistemas compartidos se mantuvieron separados de los componentes específicos de cada nivel, permitiendo reutilizar elementos generales como el sistema de entrada, control del jugador, cámara, interfaz y otros componentes centrales, mientras que las funcionalidades particulares de cada nivel se mantuvieron dentro de sus respectivas estructuras. Esta organización permitió que los diferentes niveles funcionaran dentro de un mismo proyecto sin perder su identidad mecánica y visual.

La integración se desarrolló de manera progresiva y estuvo alineada con el cronograma establecido, en el cual cada semana correspondió a un conjunto específico de actividades de configuración, incorporación, adaptación y validación. La secuencia partió de la definición de la estructura del proyecto y la configuración del entorno, continuó con la arquitectura basada en componentes y eventos, y posteriormente avanzó hacia la integración de personajes, gameplay, escenarios, objetos y sistemas de soporte. Una vez incorporados estos elementos, se realizaron las pruebas de integración necesarias para detectar inconsistencias en la interacción entre sistemas antes de continuar con las siguientes etapas.

En las fases finales del desarrollo se realizaron pruebas de coherencia general del sistema para verificar el comportamiento conjunto de los componentes incorporados. Posteriormente se efectuaron actividades de optimización orientadas al rendimiento del renderizado, uso de memoria y administración de assets, seguidas de la corrección de errores e inconsistencias detectadas durante las pruebas. Finalmente, se realizaron pruebas completas del proyecto y se preparó la build final junto con la documentación técnica correspondiente. De esta manera, la organización técnica del proyecto evolucionó de forma iterativa durante las veinte semanas de trabajo, permitiendo incorporar progresivamente los recursos y sistemas desarrollados por las diferentes áreas y mantenerlos integrados dentro de una única estructura funcional en Unity.

4.4 Arquitectura de clases y componentes

La arquitectura técnica de integración implementada en InkFall se estructuró mediante la combinación de componentes propios de Unity, scripts de comportamiento, interfaces, relaciones de herencia, managers y eventos, con el propósito de establecer una organización modular que permita coordinar los diferentes sistemas que intervienen en la

ejecución del videojuego. Esta estructura permite separar las responsabilidades de cada funcionalidad y establecer mecanismos de comunicación entre los sistemas sin concentrar toda la lógica del videojuego en una única clase o componente.

La organización de los sistemas se realizó considerando tres elementos principales: los GameObjects y prefabs presentes en las escenas, los componentes y scripts asociados a cada objeto y los managers encargados de administrar funcionalidades que requieren una coordinación centralizada. A partir de esta estructura se establecieron relaciones entre los diferentes componentes mediante referencias, interfaces y eventos, de acuerdo con las necesidades de cada sistema.

Cada sistema mantiene una responsabilidad asociada a una función específica dentro del proyecto. La lógica relacionada con el jugador, la inteligencia artificial, la interfaz de usuario, el audio, la gestión de escenas y las mecánicas particulares de cada nivel se implementa mediante componentes y scripts independientes. En los casos donde una funcionalidad requiere varios comportamientos, estos se distribuyen entre diferentes componentes del mismo GameObject o entre objetos especializados(hijos), permitiendo modificar una función sin necesidad de alterar directamente las demás.

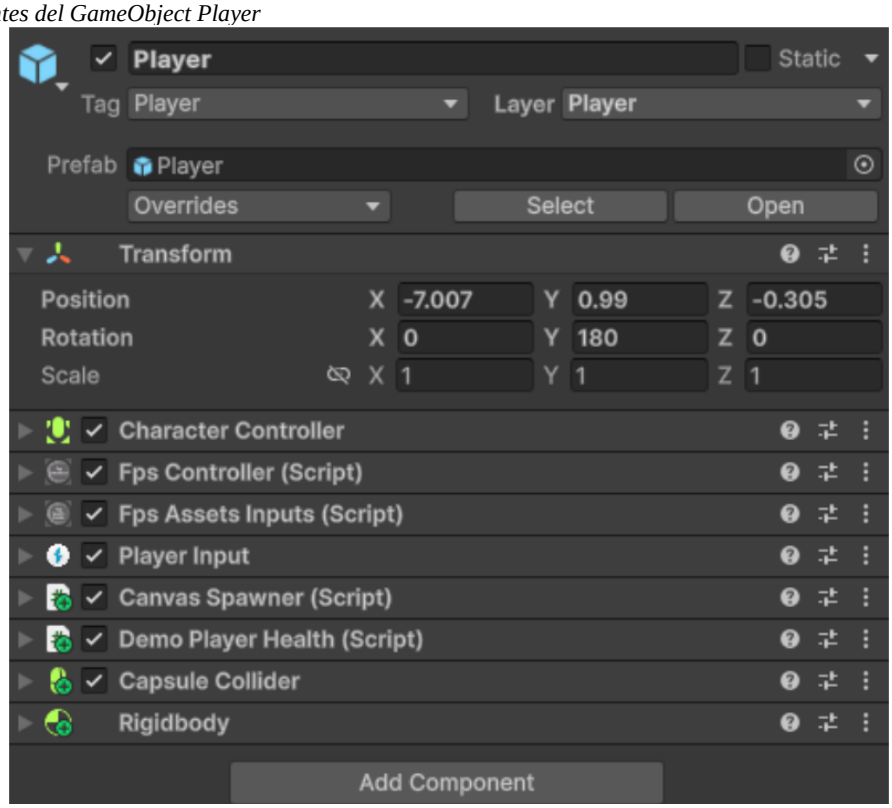
La utilización de componentes constituye uno de los elementos principales de la estructura de integración de Unity. Cada GameObject incorpora los componentes necesarios para cumplir con su función dentro de la escena, mientras que los scripts desarrollados contienen la lógica específica que determina su comportamiento. Esta distribución permite reutilizar funcionalidades mediante prefabs y reducir la dependencia entre objetos individuales de las escenas.

En el caso del jugador, el prefab correspondiente contiene los componentes necesarios para controlar su comportamiento dentro del videojuego. La estructura del

Player integra componentes relacionados con la entrada del usuario, movimiento, animación, interacción y cámara, de manera que cada comportamiento pueda mantenerse separado dentro de su correspondiente componente o script. Esta organización permite modificar el sistema de movimiento o interacción sin tener que modificar directamente los componentes encargados de la animación u otras funciones del personaje.

Ilustración 13

Componentes del GameObject Player



Nota: Componentes esenciales dentro del player, abarca el movimiento, estadísticas del personaje, físicas y eventos, Elaboración propia

Otro mecanismo utilizado para establecer comportamientos comunes entre diferentes objetos corresponde al uso de interfaces. En InkFall se implementó la interfaz IInteractable como contrato para los elementos que pueden ser utilizados por el jugador. Los objetos que requieren interacción implementan dicha interfaz y proporcionan su propia lógica para responder a la acción correspondiente.

Esta estructura permite que el sistema de interacción detecte un objeto que cumple con el contrato de IInteractable sin necesitar conocer directamente la clase concreta a la que pertenece. Por ejemplo, diferentes objetos interactivos pueden implementar la misma interfaz, aunque cada uno ejecute una acción distinta al ser utilizado. En consecuencia, el componente encargado de detectar la interacción trabaja con la interfaz y delega la ejecución de la acción al objeto correspondiente. La Ilustración 14 muestra los hijos que heredan de Iinteractable.

Ilustración 14

Relación de herencia entre objetos interactuables

Nota: Elaboración propia

Cuando varios componentes requieren compartir una estructura de comportamiento, también se utilizan relaciones de herencia. Estas relaciones permiten establecer una clase base con atributos o comportamientos comunes y extenderla mediante clases especializadas. Esta organización resulta aplicable a los elementos que comparten

características generales, permitiendo que cada clase derivada implemente únicamente las particularidades de su funcionamiento.

La coordinación de sistemas generales se realiza mediante Managers. Cada manager se encarga de administrar una responsabilidad específica y proporciona las funciones necesarias para que otros componentes puedan utilizar dicho sistema. Dentro del proyecto se encuentran managers asociados a la gestión de escenas, audio y mecánicas específicas de determinados niveles.

Los managers se incorporan a las escenas mediante GameObjects específicos y contienen los scripts responsables de administrar las operaciones correspondientes. El uso de estos administradores evita duplicar controladores de una misma funcionalidad y permite centralizar operaciones que deben ser utilizadas por diferentes objetos.

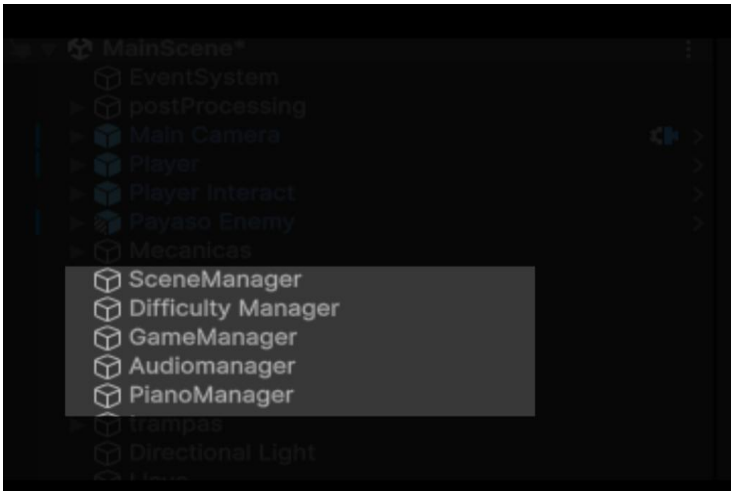
Por ejemplo, el AudioManager concentra las operaciones relacionadas con la reproducción y control de los recursos de audio, evitando que cada objeto que necesita reproducir un sonido tenga que implementar individualmente la lógica de administración del audio. De forma similar, el SceneManager centraliza las operaciones relacionadas con la transición y carga de escenas utilizadas durante el flujo del videojuego.

En el caso del sistema de piano utilizado por uno de los niveles, el PianoManager centraliza la lógica necesaria para controlar la secuencia de interacción del piano. Este manager se relaciona con el script encargado de determinar o verificar el orden de las teclas que deben ser presionadas, permitiendo validar la acción del jugador y determinar si la secuencia ejecutada corresponde con la esperada por la mecánica.

La Ilustración 15 presenta los managers incorporados dentro de una escena de InkFall.

Ilustración 15

Managers del nivel 2 dentro de la jerarquía del motor



Nota: Managers de escenas, audio y del piano dentro del inspector, Elaboración propia

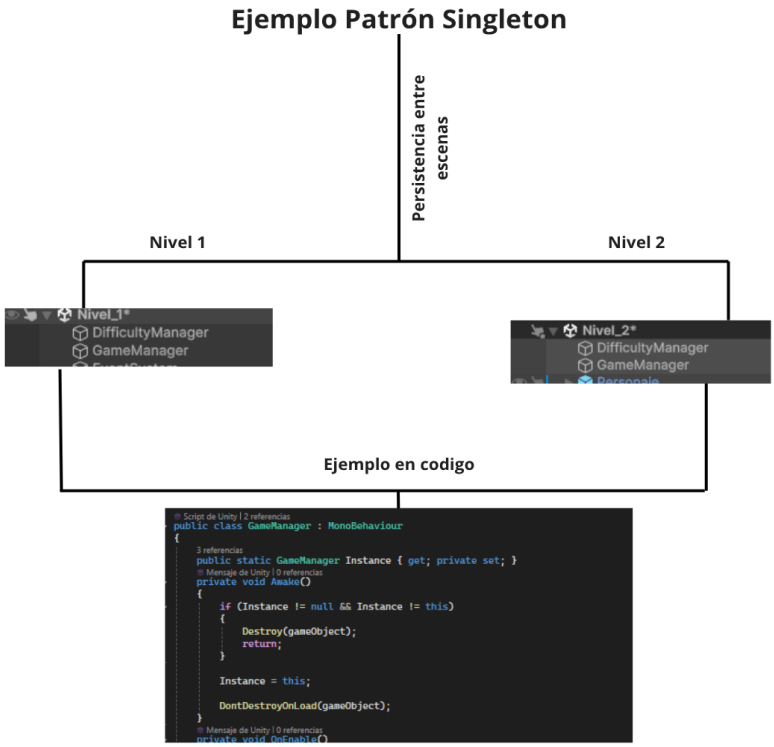
Para determinados managers se empleó el patrón Singleton, con el propósito de mantener una única instancia del administrador correspondiente durante la ejecución del juego. La implementación verifica la existencia de una instancia previa y evita la generación de duplicados, permitiendo que el sistema sea accesible desde diferentes componentes que requieran sus funciones.

La implementación del patrón Singleton resulta especialmente útil en sistemas que deben mantener un estado único o centralizado, como los relacionados con la gestión de escenas, audio u otras funcionalidades generales. Cuando un manager se encuentra configurado como persistente entre escenas, su instancia se mantiene durante la transición de un nivel a otro, evitando que se cree nuevamente el mismo controlador en cada escena.

La Ilustración 16 presenta la implementación del patrón Singleton utilizada en los managers, evidenciando la existencia de una instancia única y, cuando corresponde, la configuración utilizada para conservar el objeto durante el cambio de escenas.

Ilustración 16

Patron singleton usado en managers



Nota: Elementos de ejemplo que usan el patrón singleton y su verificación de persistencia dentro del código, diagrama. Elaboración propia

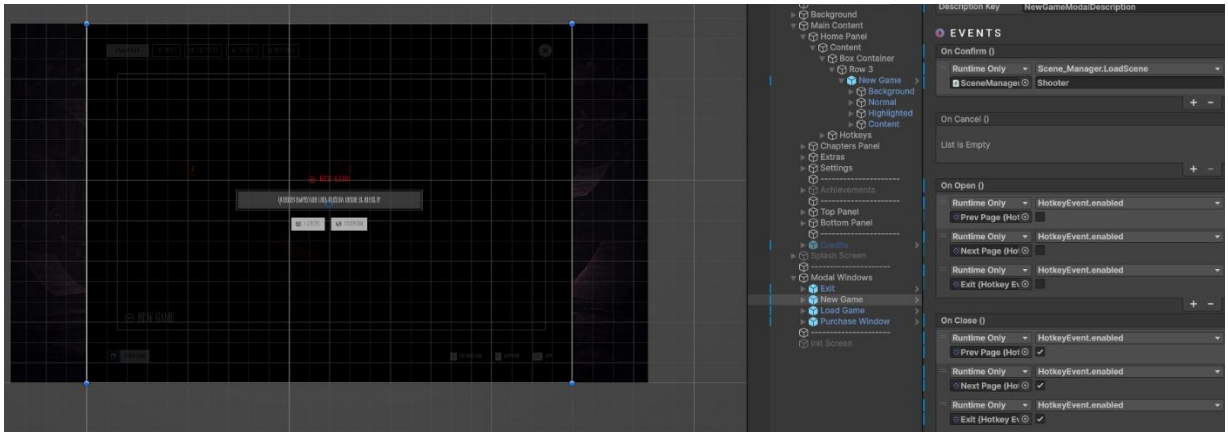
La comunicación entre los componentes del proyecto se establece mediante referencias directas controladas, llamadas a funciones y eventos, dependiendo del tipo de relación requerida. Las referencias directas se utilizan cuando un componente necesita acceder explícitamente a otro objeto para ejecutar una operación determinada, mientras que los eventos se utilizan cuando un sistema necesita notificar un cambio de estado a diferentes componentes sin establecer una dependencia directa con cada uno de ellos.

La utilización de eventos permite separar el sistema que genera una acción de aquellos que reaccionan a ella. Uno de los casos más representativos dentro de InkFall corresponde a los menús y elementos de la interfaz de usuario, donde los eventos permiten

controlar la interacción entre botones y diferentes elementos visuales. Por ejemplo, al interactuar con un botón del menú para avanzar entre ventanas, el componente asociado al botón ejecuta el evento correspondiente, mientras que los elementos de la interfaz que reciben dicha acción responden de manera independiente, ya sea activando o desactivando paneles, modificando su visualización o ejecutando la funcionalidad asociada. De esta manera, el botón no necesita controlar directamente todos los elementos involucrados en la transición, sino que comunica la acción mediante el evento y cada componente responde según la función que tiene asignada.

Ilustración 17

Eventos dentro del menú



Nota: Eventos de «On Confirm» «On Open» y «On Close» del botón de jugar dentro del motor Unity, Elaboración propia

Cuando se incorpora una nueva mecánica al proyecto, su integración se realiza utilizando los sistemas ya existentes antes de establecer nuevas dependencias entre objetos de las escenas. Cuando una mecánica necesita cambiar el estado del juego, reproducir un sonido, actualizar la interfaz o realizar una transición de escena, utiliza el manager o sistema correspondiente, siempre que dicha funcionalidad ya se encuentre centralizada.

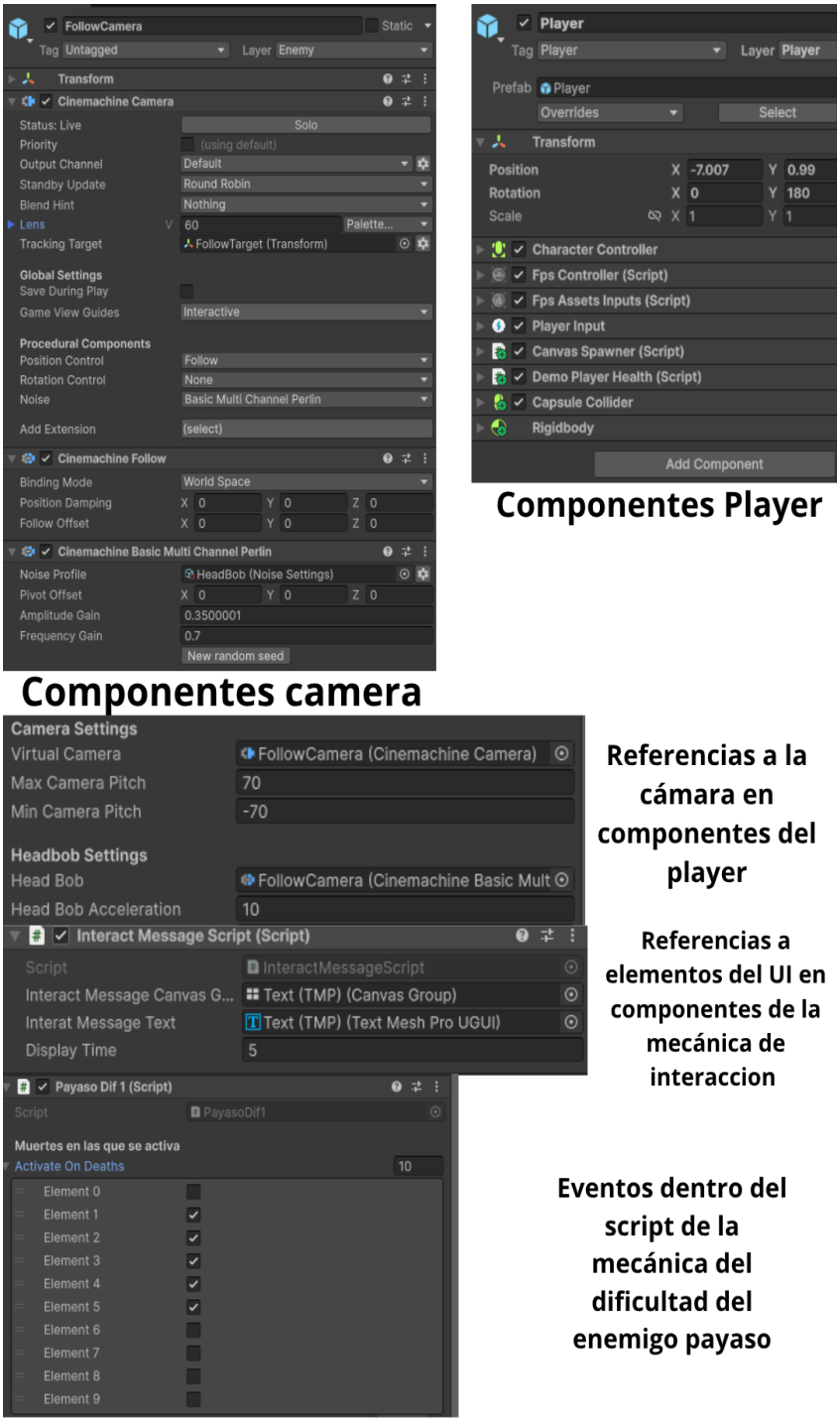
Este criterio permite conservar una estructura organizada y reduce el número de dependencias directas entre los objetos de una escena. Por ejemplo, una mecánica no necesita contener una implementación independiente del sistema de audio para reproducir un efecto, sino que solicita dicha operación al AudioManager. De forma equivalente, una mecánica que requiere realizar una transición entre niveles puede utilizar el SceneManager en lugar de controlar directamente la carga de escenas.

Los managers no sustituyen a los componentes encargados de ejecutar las mecánicas. Su función consiste en administrar las operaciones que necesitan ser compartidas o coordinadas entre diferentes objetos. La lógica particular de un enemigo, de un objeto interactivo o de una mecánica específica permanece en los componentes correspondientes, mientras que el mánager controla las operaciones generales asociadas a su sistema.

Las escenas contienen principalmente los elementos visuales, GameObjects, prefabs y configuraciones específicas de cada nivel. La lógica reutilizable permanece encapsulada en scripts y componentes que pueden ser utilizados en diferentes escenas mediante referencias y configuraciones determinadas en el Inspector de Unity. Esta separación permite mantener en cada escena únicamente los elementos necesarios para su funcionamiento y reutilizar los sistemas generales sin necesidad de duplicar su código.

Ilustración 18

Colección de componentes, referencias y eventos



Nota: Uso de componentes acorde a la función del objeto, así como referencias y eventos usado dentro del motor

Elaboración propia

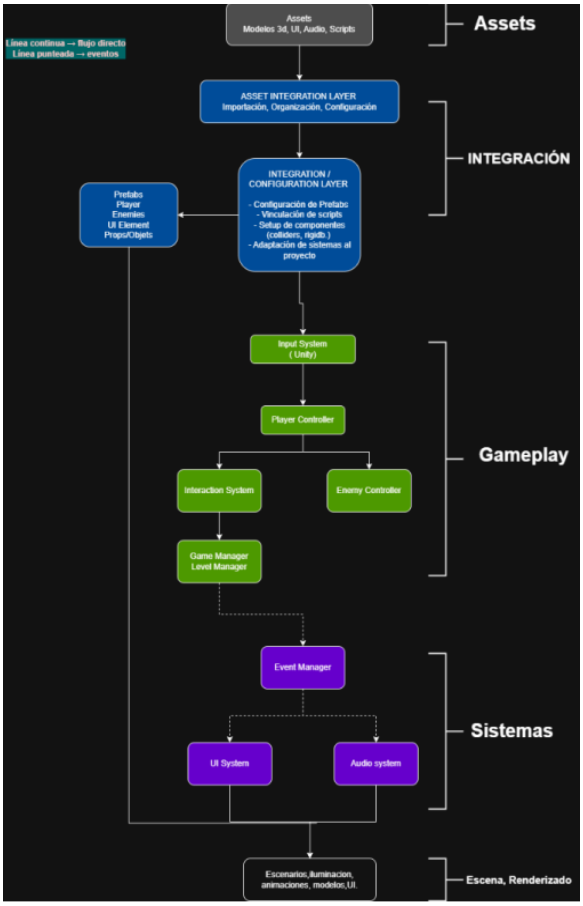
La utilización de prefabs permite encapsular configuraciones de GameObjects junto con sus componentes y referencias necesarias para que puedan ser reutilizados. Cuando un elemento debe aparecer en diferentes escenas o repetirse dentro de un mismo nivel, su configuración puede almacenarse como prefab, conservando la estructura de componentes necesaria para su funcionamiento y reduciendo la necesidad de configurar nuevamente cada objeto.

La relación entre escenas, prefabs, componentes, scripts, interfaces, managers y eventos conforma la arquitectura de integración utilizada en InkFall. Las escenas contienen la composición específica de cada nivel; los prefabs permiten reutilizar configuraciones de objetos; los componentes y scripts contienen la lógica funcional; las interfaces establecen contratos comunes entre sistemas; los managers centralizan determinadas operaciones y los eventos permiten comunicar cambios entre componentes que no necesitan mantener una referencia directa entre sí.

La Ilustración 19 muestra el diagrama general de la arquitectura de integración de InkFall, estableciendo las relaciones entre los principales sistemas del proyecto. En el diagrama se identifican el Player, los sistemas/managers, las escenas, los objetos interactivos y los mecanismos de comunicación utilizados para relacionarlos.

Ilustración 19

Diagrama de la arquitectura general de integración de InkFall



Nota: Elaboración propia

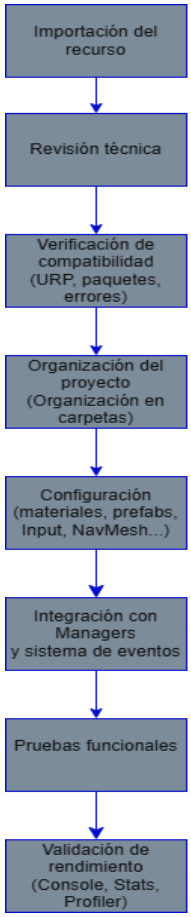
A partir de esta estructura, la integración técnica permite mantener separadas las responsabilidades de los diferentes sistemas y facilita la incorporación de nuevas funcionalidades. La modificación de una mecánica específica puede realizarse dentro de los componentes responsables de dicha funcionalidad, mientras que los sistemas generales permanecen centralizados y reutilizables. De esta manera, la arquitectura implementada permite gestionar la interacción entre los diferentes elementos técnicos del videojuego y mantener una estructura coherente durante la integración de las mecánicas correspondientes a los distintos niveles de InkFall.

4.5 Procedimiento para ampliar el sistema

Para mantener la estabilidad, escalabilidad del proyecto y reducir problemas de compatibilidad entre los diferentes sistemas, durante el desarrollo de InkFall se estableció un flujo de integración común para todos los recursos incorporados al motor Unity, con su diagrama en la ilustración 20. Este procedimiento se aplicó tanto para modelos 3D y escenarios como para assets de terceros, prefabs y sistemas desarrollados específicamente para el videojuego.

Ilustración 20

Flujo general de trabajo para la integración de recursos



Nota: Elaboración propia

El proceso comienza con la importación del recurso dentro del proyecto. Una vez incorporado, se debe verificar que todos los archivos hayan sido importados correctamente y que no existan errores de compilación en la consola de Unity. Cuando el recurso provenga de la Unity Asset Store o de un paquete Unity Package, también es recomendable comprobar que sea compatible con la versión del motor utilizada y que no genere conflictos con otros paquetes previamente instalados.

Después de la importación, revisar la estructura del recurso antes de comenzar su utilización. Muchos paquetes incluyen escenas de demostración, documentación, materiales de ejemplo o recursos adicionales que no serán utilizados dentro del proyecto. Por este motivo, únicamente se integran los elementos necesarios, manteniendo organizada la estructura de carpetas y evitando incorporar archivos innecesarios que incrementen el tamaño del proyecto.

Si el recurso corresponde a un sistema completo, como un controlador de personaje, una inteligencia artificial o una herramienta de terceros, se recomienda probar inicialmente su funcionamiento en una escena de prueba. Una vez verificado que el sistema funciona correctamente de forma independiente, se procede a adaptar su comportamiento para integrarlo con la arquitectura utilizada en InkFall, reutilizando los Managers, eventos y componentes existentes en lugar de sustituir la estructura ya implementada.

En el caso de modelos 3D y recursos visuales, el siguiente paso consiste en verificar aspectos como la escala, orientación, materiales, texturas y shaders antes de incorporarlos a una escena definitiva. Posteriormente, los modelos se convierten en prefabs cuando su reutilización es necesaria en diferentes escenarios, facilitando así su mantenimiento y actualización.

La integración de escenarios sigue un procedimiento similar. Inicialmente se construye la estructura base del nivel mediante terrenos, paredes, pisos, techos y elementos decorativos. Posteriormente se incorporan los objetos interactivos, las zonas de navegación mediante NavMesh, la iluminación, los efectos visuales y el resto de mecánicas propias de cada nivel, comprobando que todos estos elementos funcionen correctamente dentro de la escena.

Finalmente, después de integrar cualquier recurso, se realizan pruebas funcionales para verificar que no existan errores de compilación, referencias perdidas, conflictos con otros sistemas o disminuciones significativas del rendimiento. Solo una vez superadas estas verificaciones el componente pasa a formar parte de la versión principal del proyecto.

Seguir este flujo de integración permitió incorporar recursos provenientes de distintas fuentes manteniendo una estructura homogénea durante todo el desarrollo, facilitando tanto la adaptación de assets de terceros como el mantenimiento y evolución del proyecto.

4.6 Integración entre diseño, arte y programación

Una vez que el recurso ha sido importado y verificado, proceder con su integración dentro de la estructura del proyecto. Dependiendo del tipo de componente, el proceso puede variar; sin embargo, es recomendable mantener los mismos criterios utilizados durante el desarrollo para conservar la organización y evitar conflictos entre sistemas, como criterio común en la integración es la correcta colocación en la estructura de carpetas.

4.6.1 Modelos 3D

Los modelos 3D se incorporan inicialmente dentro de la estructura de carpetas correspondiente a cada nivel del videojuego. Antes de utilizarlos en una escena, se verifica que la escala, orientación, materiales y texturas sean compatibles con el resto del proyecto.

Cuando un modelo proviene de una fuente externa, también es necesario comprobar que sus materiales sean compatibles con Universal Render Pipeline (URP) y, de ser necesario, convertir los shaders antes de continuar con la integración. Posteriormente se configuran los componentes necesarios para su funcionamiento, como colliders, rigidbodies, puntos de interacción o animaciones, dependiendo de la función que cumplirá dentro del videojuego

4.6.2 Personajes

Una vez incorporados los modelos definitivos del personaje y de los enemigos, se procedió a sustituir los modelos provisionales utilizados durante el desarrollo inicial, para los enemigos se utilizaron objetos temporales para validar el comportamiento de la inteligencia artificial. Posteriormente se configuró el componente Animator, asignando el controlador correspondiente y verificando que todas las animaciones estuvieran correctamente enlazadas.

Ilustración 21

Enemigos

Nota: Integración de los modelos 3d de los enemigos en sus respectivos escenarios , Elaboración propia

Durante esta etapa se revisaron parámetros como Loop Time, velocidad de reproducción (Speed), transiciones entre estados y condiciones de cambio dentro del

Animator Controller, garantizando que las animaciones respondieran correctamente a las acciones ejecutadas por los diferentes sistemas del videojuego.

En el caso del personaje principal, las animaciones de locomoción, salto, ataque e interacción fueron ajustadas para integrarse con el controlador de movimiento existente. La integración requirió configurar el componente Animator, adaptando el Animator Controller suministrado con el sistema de movimiento para utilizar las animaciones definitivas del personaje. La locomoción se organizó mediante un Blend Tree, el cual permite interpolar automáticamente entre las animaciones de reposo, caminar y correr de acuerdo con la velocidad del jugador. Adicionalmente, el controlador se estructuró en múltiples Layers, separando la locomoción de las acciones del personaje y de las animaciones superpuestas (Overlay), lo que permite ejecutar acciones como apuntar o interactuar sin interrumpir el movimiento base. Para controlar las transiciones entre estados se configuraron parámetros del Animator relacionados con desplazamiento, carrera, salto, acciones de combate y variables empleadas por el sistema de Inverse Kinematics (IK), como el control de influencia de las manos durante la manipulación de armas. Esta organización permitió reutilizar el controlador de movimiento existente, manteniendo una estructura flexible para incorporar nuevas animaciones sin modificar la lógica principal del sistema.

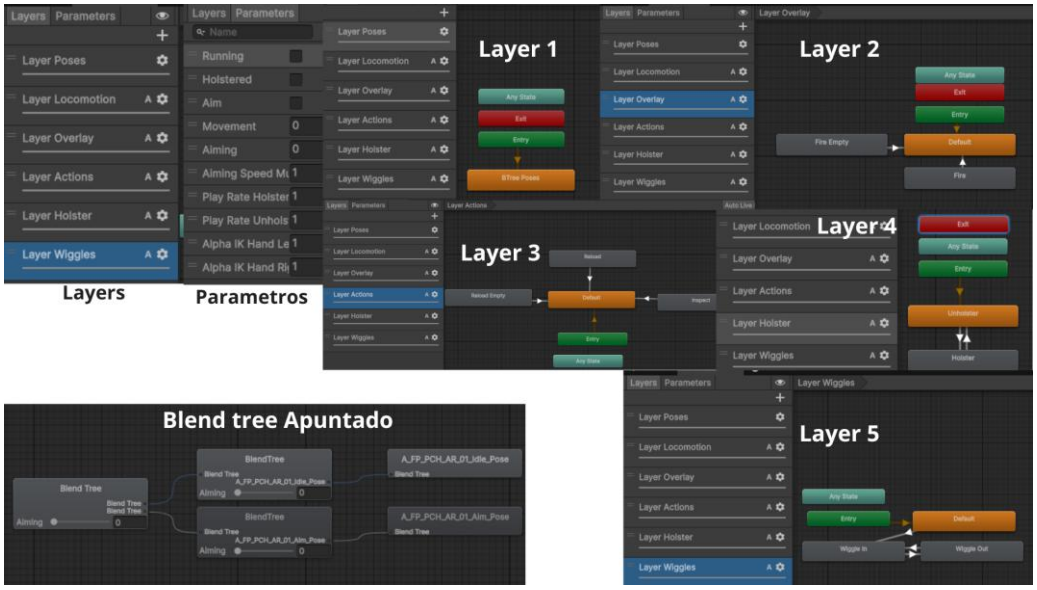
En los enemigos se siguió un procedimiento similar. Los modelos definitivos sustituyeron a los objetos temporales utilizados durante el desarrollo, incorporando el modelo de la criatura tipo araña para el primer nivel y el enemigo con forma de payaso para el segundo. Cada enemigo fue configurado con su correspondiente Animator Controller, cuya locomoción también se organizó mediante un Blend Tree para interpolar las animaciones de reposo, desplazamiento y carrera según la velocidad calculada por el NavMesh Agent. Además, se configuraron los estados de persecución, diferentes

secuencias de ataque y la animación de muerte, controlados mediante parámetros específicos del Animator, como los disparadores para los distintos ataques (Attack 1, Attack 2 y Attack 3), la animación de carrera de ataque (Run Attack) y las variables utilizadas por la máquina de estados. Finalmente, se configuraron los componentes NavMesh Agent, colliders, puntos de ataque y el resto de referencias necesarias para reutilizar la lógica de inteligencia artificial implementada previamente, evitando modificar el comportamiento general del sistema.

En ambos casos se configuraron los componentes Animator, NavMesh Agent, colliders, puntos de ataque y el resto de las referencias necesarias para reutilizar la lógica de inteligencia artificial desarrollada previamente.

Ilustración 22

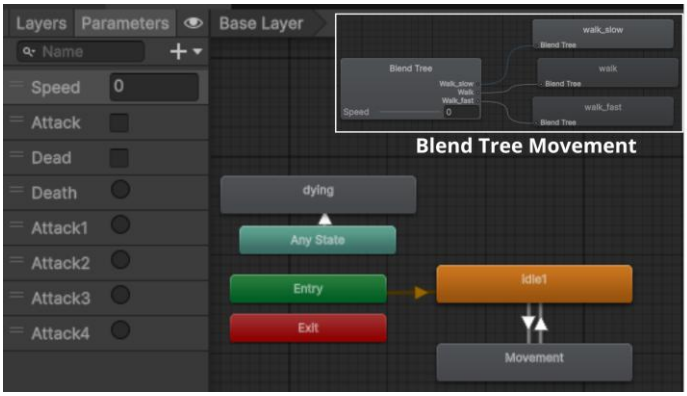
Organización del Animator Controller del personaje principal



Nota: Colección de layers, parámetros y blend tree importantes del animator del personaje, Elaboración propia

Ilustración 23

Organización del Animator Controller del enemigo (parámetros y Blend Tree)



Nota: Colección de parámetros, organización y blend tree del animator de enemigos, Elaboración propia

4.6.3 Armas

Las mecánicas de combate ya disponían de un sistema funcional desarrollado por otras áreas del proyecto; sin embargo, éste utilizaba armas temporales empleadas únicamente para pruebas. Durante el proceso de integración dichas referencias fueron sustituidas por los modelos definitivos proporcionados por el equipo de modelado 3D. Cada arma fue configurada como un Prefab independiente, conservando los scripts existentes y actualizando las referencias hacia los nuevos modelos, puntos de disparo, animaciones y efectos visuales correspondientes. Asimismo, se verificó la correcta sincronización entre las animaciones del arma, el sistema de disparo y las acciones ejecutadas por el jugador.

Este procedimiento permitió mantener la lógica previamente implementada sin necesidad de modificar el funcionamiento del sistema de combate.

4.6.4 Prefabs

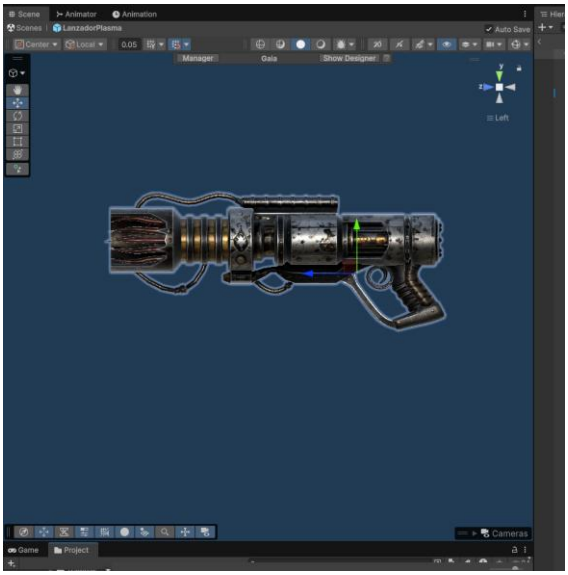
Antes de utilizar un prefab en el proyecto, verificar que todas sus referencias internas permanezcan correctamente asignadas y que los scripts asociados funcionen sin errores. Durante el desarrollo los recursos reutilizables fueron convertidos en Prefabs con el fin de centralizar su configuración y facilitar su utilización en diferentes escenas del videojuego.

Este procedimiento permitió mantener una única versión de cada objeto, de modo que cualquier modificación realizada sobre el Prefab se reflejara automáticamente en todas sus instancias.

Entre los principales Prefabs implementados se encuentran elementos estructurales del escenario, como paredes, pisos y componentes modulares del entorno; los personajes principales y enemigos; armas, objetos interactivos, como interruptores y llaves; además de elementos funcionales como el encendedor utilizado por el jugador y diferentes sistemas de iluminación.

Ilustración 24

Prefab arma de plasma



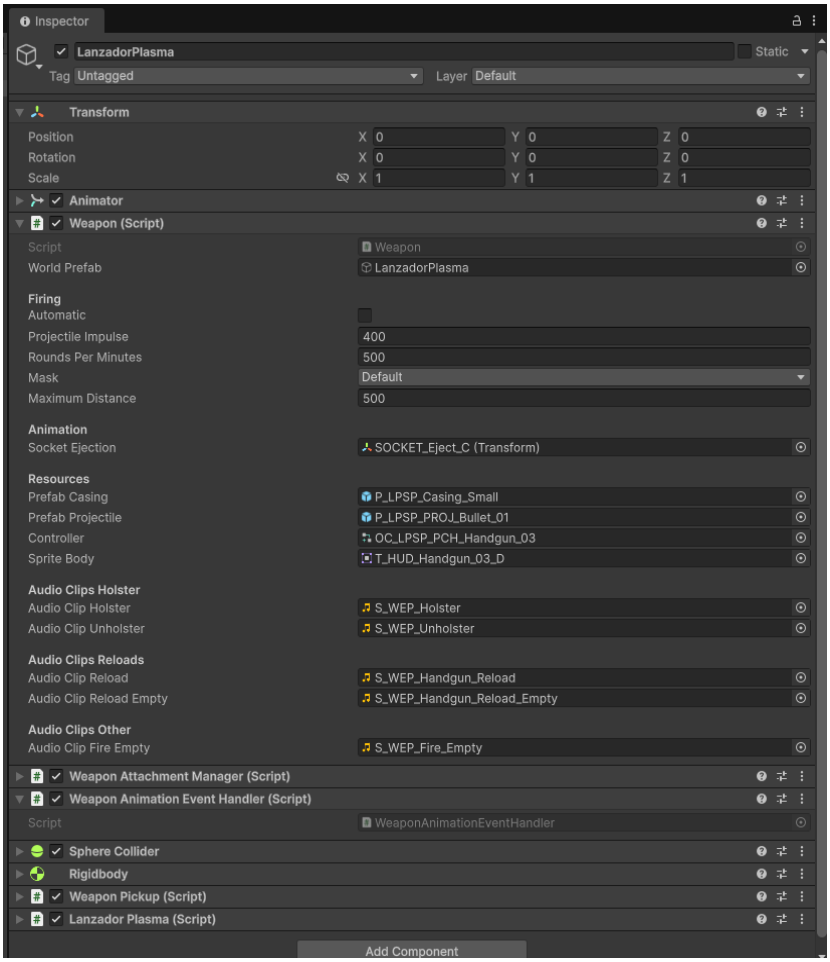
Nota: Prefab del arma de plasma visto desde Unity, Elaboración propia

Cada Prefab fue configurado con los componentes necesarios para su funcionamiento, incluyendo colliders, rigidbodies cuando fueron requeridos, scripts personalizados, Animator, fuentes de iluminación, eventos y demás referencias necesarias según la función que desempeñaba dentro del videojuego. En el caso de objetos interactivos, como interruptores y llaves, la configuración permitió reutilizar la misma lógica en

diferentes niveles modificando únicamente las referencias específicas de cada escena. Un ejemplo representativo corresponde al encendedor. Este recurso fue implementado como un Prefab que integra el modelo 3D, la fuente de luz, los scripts de control y un collider empleado para detectar la proximidad de los enemigos. Gracias a esta configuración, el mismo objeto puede reutilizarse en cualquier escena conservando todas sus funcionalidades sin necesidad de configuraciones adicionales.

Ilustración 25

Componentes del prefab Arma



Nota: referencias y scripts vinculados al prefab del arma, se evidencia las referencias a sonidos, otros prefabs, objetos y valores de ajustes del arma; Elaboración propia

Para preservar la modularidad de la arquitectura implementada en InkFall, se evitó en medida de lo posible establecer referencias directas entre Prefabs y objetos concretos de

una escena. En su lugar, la comunicación con otros sistemas se realizó mediante Managers y eventos, permitiendo reutilizar un mismo recurso en distintos escenarios sin modificar su estructura interna.

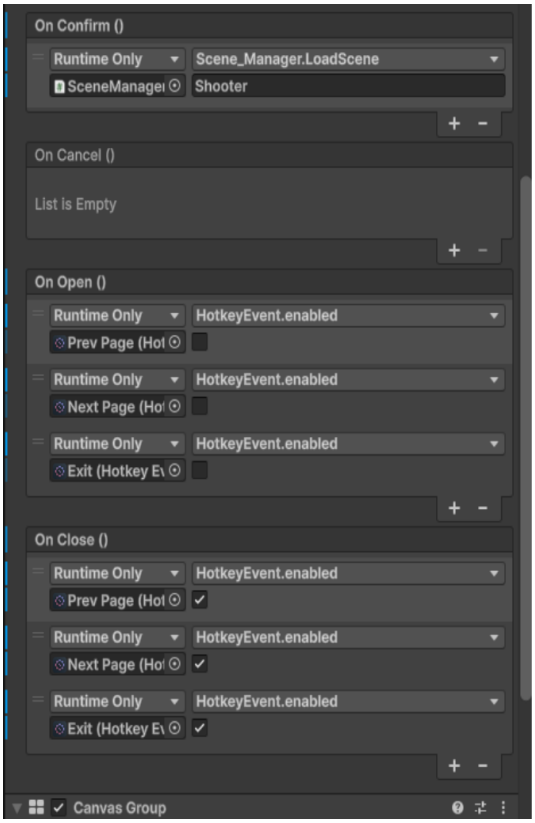
4.6.5 Interfaz y menús

La interfaz gráfica fue proporcionada inicialmente con su funcionamiento implementado. El proceso de integración consistió principalmente en adaptar las referencias internas de botones, paneles y elementos visuales para ajustarlas a la estructura definitiva del proyecto. Se verificó la correcta asignación de los objetos utilizados por los diferentes menús, indicadores del HUD y pantallas del videojuego, actualizando aquellas referencias que cambiaron durante la reorganización de escenas y recursos. Asimismo, el menú principal fue integrado utilizando el sistema de interfaz desarrollado para el proyecto. Durante esta etapa se configuraron las acciones asociadas a cada botón, enlazando las funciones correspondientes mediante eventos del Inspector.

Las transiciones entre escenas se implementaron mediante eventos y el SceneManager, como se ve en la ilustración 26, verificando que la carga de cada nivel se realizara correctamente y que los datos necesarios permanecieran disponibles durante el cambio de escena.

Ilustración 26

Eventos de la ventana de "New Game"



*Nota: E*ventos dentro del inspector, con su referencia al Scene manager y eventos para cuando se abre y cierra la ventana de confirmación, Elaboración propia

4.6.6 Mecánicas

Las mecánicas fueron integradas de manera progresiva conforme fueron dispuestas por el equipo responsable. Antes de incorporarlas a la escena principal, cada funcionalidad fue validada de forma aislada para comprobar su comportamiento y compatibilidad con la arquitectura del proyecto. Posteriormente, se realizaron pruebas dentro de la escena para verificar su interacción con los demás sistemas involucrados.

Durante la integración se reutilizaron los Managers, eventos y componentes existentes, evitando duplicar funcionalidades o establecer dependencias directas entre sistemas. Cuando fue necesario adaptar recursos desarrollados por otras áreas, se

modificaron las referencias, parámetros y componentes requeridos para mantener la compatibilidad con la estructura del proyecto, preservando la lógica previamente implementada. Esta adaptación permitió conectar las mecánicas recibidas con los sistemas generales del proyecto sin modificar innecesariamente su funcionamiento interno.

Entre las principales mecánicas integradas se encuentran el sistema de movimiento, el sistema de interacción con objetos, la apertura de puertas mediante llaves, la activación de interruptores para controlar la iluminación, el funcionamiento del encendedor utilizado por el jugador y las mecánicas de combate de los diferentes niveles. Estas funcionalidades requirieron su vinculación con diferentes sistemas del proyecto. El movimiento se relacionó con los componentes de control del jugador y las colisiones del escenario; la interacción con objetos se vinculó con los elementos interactuables y los eventos correspondientes; las puertas y los interruptores requirieron comunicación con los sistemas de llaves e iluminación; mientras que el encendedor se integró con los elementos del escenario que dependen de su activación como el enemigo y los QTE (Quick Time Events). Las mecánicas de combate, por su parte, se vincularon con los sistemas de armas, enemigos, daño y eventos propios de cada nivel.

Durante este proceso también se identificaron problemas de integración que requirieron ajustes sobre los componentes involucrados. Entre ellos se encontraron comportamientos incorrectos en el movimiento del personaje al desplazarse junto a las paredes, así como velocidades de apertura inadecuadas en determinadas puertas y cajones interactivos. Estos problemas fueron corregidos mediante ajustes en los componentes responsables de las colisiones y mediante la modificación de los parámetros de velocidad de las interacciones, respectivamente. Posteriormente, las funcionalidades fueron sometidas

nuevamente a pruebas para comprobar que las modificaciones no afectaran su interacción con los demás sistemas.

En todos los casos se verificó la correcta comunicación entre los distintos sistemas antes de incorporar las funcionalidades a la versión principal del videojuego. De esta manera, la integración de las mecánicas se evaluó no solo desde su funcionamiento individual, sino también desde su comportamiento dentro del flujo general de cada nivel..

4.6.7 Escenarios

La construcción de cada escenario comienza revisando el esquema del nivel dado en el white box proporcionado por el equipo, después se organizan los recursos estructurales del nivel, como paredes, pisos, techos y elementos decorativos (props). En InkFall estos elementos se colocan individualmente para adaptar cada escenario a las necesidades de la narrativa y de las mecánicas de juego. Como se observa en la Ilustración 27, los escenarios se construyen mediante la combinación de recursos modulares, permitiendo modificar su distribución durante el desarrollo sin depender de una única estructura. Durante esta etapa también se revisa la configuración de los modelos incorporados, considerando sus materiales, colliders, escalas y transformaciones para adecuarlos a las características de cada escenario

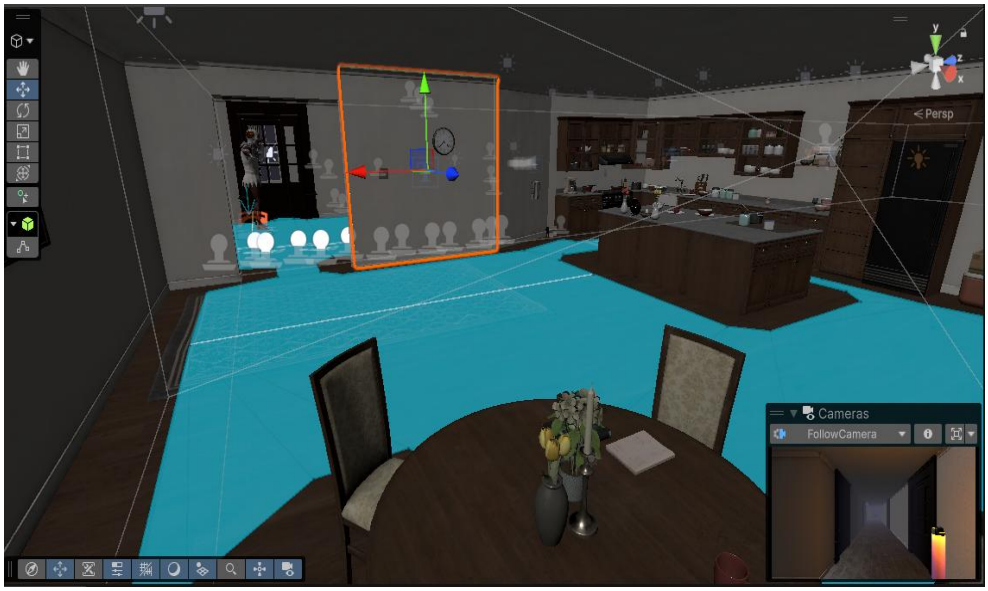
Una vez definida la estructura del escenario, se incorporan los objetos interactivos, la iluminación, los efectos visuales, los puntos de aparición de enemigos y los sistemas de navegación mediante NavMesh. Estos elementos se integran de acuerdo con los requerimientos establecidos para cada nivel, vinculándolos con las mecánicas y sistemas previamente incorporados al proyecto. Por ejemplo, los objetos interactivos como las puertas se configuran para responder a los eventos correspondientes, mientras que las superficies del escenario se preparan para permitir la navegación del jugador y de los

agentes controlados por inteligencia artificial. Durante la configuración de la navegación se revisa la superficie transitable del escenario mediante NavMesh, realizando ajustes cuando los elementos estructurales o interactivos generan zonas que impiden el desplazamiento esperado. De igual manera, se comprueba la interacción entre colliders, agentes de navegación y elementos del escenario para evitar bloqueos o comportamientos incorrectos durante la ejecución.

Finalmente, se realiza una prueba funcional de la escena para verificar que tanto el jugador como los agentes controlados por inteligencia artificial puedan desplazarse correctamente por el nivel y que los elementos interactivos, efectos visuales, iluminación y mecánicas presentes respondan de acuerdo con su configuración. Esta verificación permite identificar problemas de integración específicos del escenario antes de incorporarlo a la versión principal del proyecto

Ilustración 27

Uso de recursos



Nota: Se usaron recursos como: construcción (pared naranja), props(objetos de cocina/comedor) y NavMesh (piso azul); Elaboración propia

4.6.8 Verificación y optimización

Después de incorporar cualquier recurso al proyecto, se realiza una prueba funcional completa de la escena (Play) para comprobar que el nuevo componente interactúe correctamente con el resto de los sistemas. Durante esta revisión se verifica la ausencia de errores de compilación, referencias perdidas, conflictos entre componentes y problemas de funcionamiento. Este procedimiento permite detectar errores de integración en etapas tempranas y mantener una estructura organizada durante todo el desarrollo de InkFall, facilitando la incorporación de nuevos recursos y el mantenimiento del proyecto.

Como parte del proceso de integración, también se aplican medidas de optimización sobre los recursos incorporados y las escenas, con el propósito de reducir la carga de procesamiento y mantener un rendimiento adecuado durante la ejecución en tiempo real. Estas medidas incluyen la configuración de Occlusion Culling para evitar el procesamiento de elementos que permanecen ocultos respecto a la cámara, así como el establecimiento de Layer Culling Distances para limitar la distancia de renderizado de determinados elementos según su importancia dentro de la escena. Adicionalmente, se revisan las características de los modelos y componentes incorporados para identificar configuraciones que puedan generar una carga innecesaria sobre el procesamiento gráfico. Estas acciones se realizan durante la integración para asegurar que los recursos no solo funcionen correctamente, sino que también se encuentren configurados de acuerdo con los requerimientos de rendimiento del proyecto.

4.7 Integración de sistemas

La integración entre los diferentes sistemas de InkFall se diseñó procurando reducir las dependencias directas entre componentes. Siempre que fue posible, las interacciones se realizaron mediante administradores (Managers), eventos o referencias previamente

configuradas desde el Inspector, evitando que los distintos objetos de la escena dependieran directamente unos de otros. Este criterio facilita el mantenimiento del proyecto y disminuye la posibilidad de introducir errores al modificar una mecánica existente.

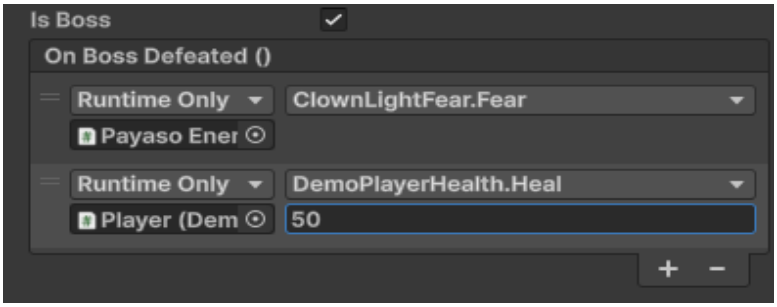
Cuando un sistema requiere interactuar con otro, la primera alternativa consiste en utilizar alguno de los Managers disponibles dentro del proyecto. Estos administradores centralizan la lógica correspondiente a cada sistema y exponen únicamente las funciones necesarias para que otros componentes puedan utilizarlas, evitando accesos innecesarios a la implementación interna de cada módulo.

En aquellos casos donde una acción debe producir múltiples respuestas, la comunicación se realiza mediante eventos. Este mecanismo permite que varios sistemas reaccionen a un mismo cambio de estado sin establecer referencias directas entre ellos. Por ejemplo, cuando la vida del jugador llega a cero, el sistema genera un evento que activa la interfaz de reinicio, reinicia la escena correspondiente y actualiza el nivel de dificultad dinámica. De forma similar, cuando el jugador completa un nivel, el administrador de escenas realiza la transición hacia el siguiente escenario sin que las mecánicas del jugador necesiten conocer cómo se ejecuta dicho proceso.

En la ilustración 28 se muestra un ejemplo de configuración de eventos utilizada durante el desarrollo del proyecto, donde diferentes componentes responden a una misma acción, en este caso, si el enemigo es considerado un enemigo nivel jefe, se le activa una nueva mecánica y con el mismo evento añadimos vida al jugador.

Ilustración 28

Gestor de eventos del enemigo



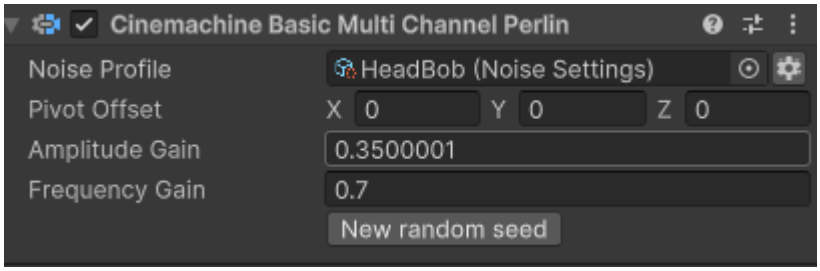
Nota: Elaboración propia

Cuando la comunicación requiere acceder a componentes específicos de la escena, las referencias se asignan previamente desde el Inspector utilizando campos serializados. Antes de ejecutar el proyecto, es recomendable verificar que todas estas referencias permanezcan correctamente configuradas, ya que una referencia nula puede impedir el funcionamiento de la mecánica o producir errores durante la ejecución.

La ilustración 29 y 30 se presenta un ejemplo de referencias configuradas desde el Inspector en la cual asignamos la máquina de estados Head Bob al apartado de ruido (noise) de la cámara, si no estuviese bien referenciado, el controlador del personaje que maneja la intensidad de movimiento de cabeza (Head Bob) no funcionase correctamente pues también debe recibir una referencia del componente cámara.

Ilustración 29

Referencia correcta en el GameObject Camera



Nota: Referencia de la máquina de estado en la cámara, Elaboración propia

Ilustración 30

Referencia correcta en el GameObject Player

Nota: Referencia de la cámara en el player, para controlar la máquina de estados, Elaboración propia

Finalmente, después de incorporar una nueva mecánica o modificar un sistema existente, se recomienda realizar pruebas funcionales que involucren a todos los componentes relacionados. Este procedimiento permite comprobar que la comunicación entre sistemas continúa funcionando correctamente y que los cambios realizados no afectan el comportamiento general del proyecto.

4.7.1 Pruebas técnicas y validación

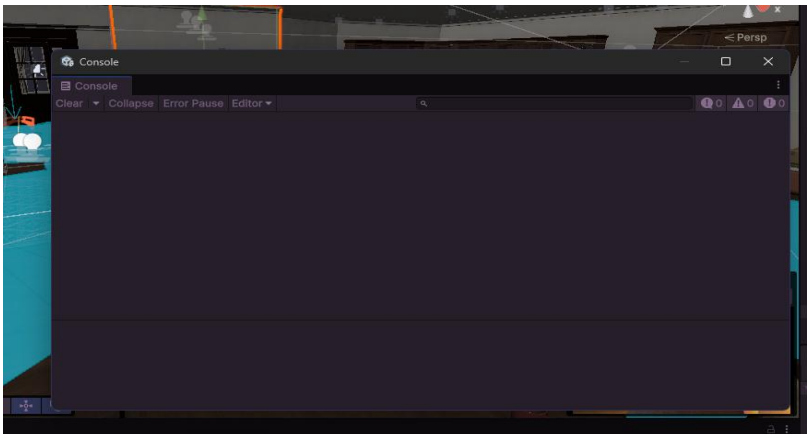
Una vez integrado un nuevo recurso o sistema dentro del proyecto, se realizó un proceso de validación antes de incorporarlo a la versión principal del videojuego. El

objetivo fue comprobar que cada componente funcionara correctamente de forma individual y que su incorporación no afectara el comportamiento del resto de los sistemas.

La primera verificación consistió en ejecutar la escena correspondiente y comprobar que no existieran errores de compilación ni mensajes de advertencia en la consola de Unity. Como se observa en la Ilustración 31, durante la ejecución utilizada como evidencia para esta validación no se registraron errores (Errors) ni advertencias (Warnings) en la consola. Esta comprobación permitió verificar que los componentes incorporados no generaran conflictos de compilación en el escenario evaluado, aunque no constituye una garantía de ausencia de errores en todas las situaciones de ejecución.

Ilustración 31

Visualización de la consola en modo de juego



Nota: Elaboración propia

Si el recurso incorporaba scripts o referencias a otros objetos, se verificó además que todas las dependencias permanecieran correctamente asignadas desde el Inspector. Un inconveniente frecuente durante el proceso de integración fue que algunos componentes u objetos permanecieran deshabilitados, provocando la pérdida de funcionalidad o referencias. Por ello, se revisó que todos los componentes requeridos se encontraran

habilitados y que los objetos necesarios permanecieran activos dentro de la jerarquía de la escena.

Posteriormente, se realizaron pruebas funcionales sobre las mecánicas y recursos integrados. En el caso de los modelos 3D y prefabs, se comprobó que su posición, escala, colisiones y materiales fueran correctos. En el caso de las mecánicas de juego, se validó que estas respondieran adecuadamente a las acciones del jugador y que su comportamiento fuera consistente con el diseño previsto.

Cuando la integración involucró sistemas existentes, también se comprobó que la comunicación entre ellos continuara funcionando correctamente. De esta manera, se verificó que las interacciones del jugador, la interfaz, el audio y los eventos asociados respondieran de forma esperada después de incorporar los nuevos componentes.

Para documentar los resultados de las pruebas funcionales, se registraron los casos evaluados, las condiciones de ejecución y el resultado obtenido en cada comprobación. Esta evaluación permitió identificar de forma cualitativa el comportamiento de los sistemas integrados y, al mismo tiempo, cuantificar las pruebas satisfactorias y las incidencias detectadas durante el proceso de validación.

A partir de las pruebas realizadas, se consideró como resultado satisfactorio aquel caso en el que el recurso o sistema respondiera de acuerdo con el comportamiento esperado y no produjera errores que impidieran continuar la ejecución. Las incidencias encontradas durante estas comprobaciones fueron corregidas y posteriormente sometidas a una nueva ejecución para verificar su solución.

Tabla 7

Resultados de las pruebas funcionales

Sistema	Pruebas	Incidencia detectada	Corrección
Movimiento	4	El personaje quedaba adherido a las paredes al mantener presionada la tecla W.	Se ajustó la posición del Sphere Collider encargado de detectar la colisión con el piso
Interacción	4	Las puertas se desplazaban a una velocidad excesiva mientras se mantenía presionado el clic para ejecutar la interacción.	Se modificó la variable de velocidad del movimiento y se hizo pública para permitir su ajuste desde el Inspector de Unity hasta obtener un valor adecuado.
IA	6	El NavMesh del enemigo no hacia el bake de superficies más alta como gradas	Configurar los parámetros de altura del componente antes de realizar el bake, hasta encontrar la adecuada para las gradas
Eventos	2	Ninguna	Ninguna
Audio	2	Ninguna	Ninguna

Nota: Elaboración propia

Los resultados de las pruebas funcionales permitieron identificar problemas relacionados principalmente con la interacción entre los componentes físicos y la configuración de las mecánicas. En el sistema de movimiento se detectó una interferencia entre el Sphere Collider del personaje y las superficies verticales, provocando que el personaje quedara adherido a las paredes al desplazarse hacia adelante. La modificación de la posición del collider permitió eliminar este comportamiento. En el sistema de interacción con puertas, se identificó una velocidad de desplazamiento superior a la esperada durante la apertura. Para solucionar este comportamiento, la variable encargada de controlar la velocidad fue expuesta en el Inspector de Unity, permitiendo realizar ajustes durante las pruebas hasta establecer un valor adecuado. El sistema de la IA de enemigo se encontraba limitada por el escenario, con el bake del componente NavMesh el enemigo no podía subir superficies con cierta altura como el caso de gradas, después de probar aumentar el valor

en el parámetro de altura se realizó un bake correcto para el movimiento del enemigo Después de aplicar las correcciones, se realizaron nuevas ejecuciones para comprobar que los comportamientos identificados no volvieran a presentarse.

Además de las pruebas funcionales, se realizó una revisión del rendimiento utilizando las herramientas proporcionadas por Unity. La ventana Stats permitió supervisar información relevante de la escena, como el número de batches, triángulos, vértices y la tasa de fotogramas por segundo (FPS). Como se muestra en las Ilustraciones 32 y 33, durante las pruebas de integración se realizaron mediciones de rendimiento mediante las herramientas Stats y Profiler de Unity.

Las mediciones se efectuaron bajo un entorno de prueba controlado, cuyos parámetros se presentan en la Tabla 8.

Tabla 8

Entorno para las pruebas de rendimiento

Parámetro	**Valor**
Calidad	Uso de la plantilla para PC. «PC_RPAsset »
Procesador	Intel(R) Core(TM) i7-11700F
Memoria Ram	16 GB
Tarjeta gráfica	Nvidia RTX 4060
Número de ejecuciones	5
Plataforma de ejecución	Editor de Unity en 1920x1080

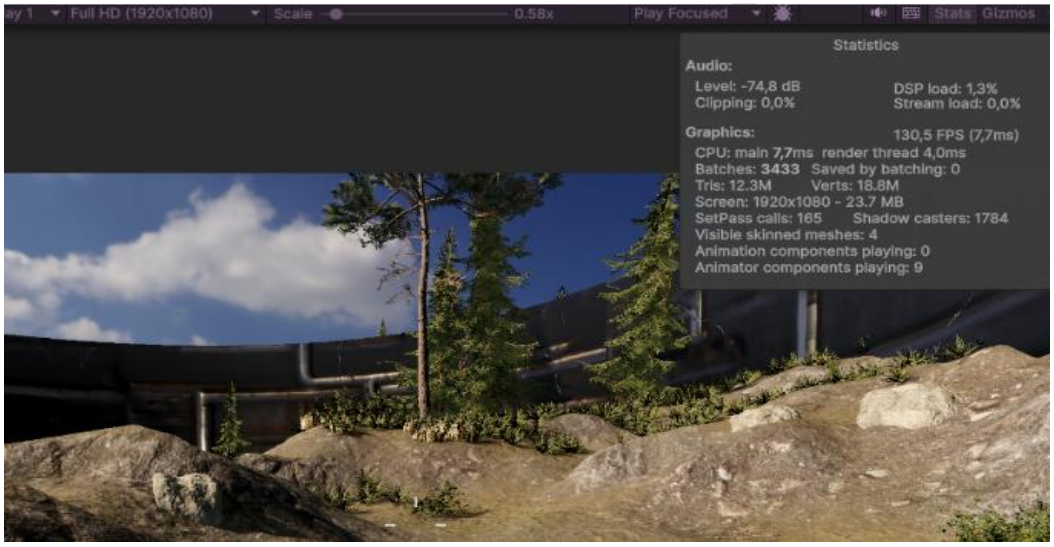
Nota: Elaboración propia

Las pruebas se realizaron utilizando una computadora equipada con un procesador Intel(R) Core(TM) i7-11700F, 16 GB de memoria RAM y una tarjeta gráfica Nvidia RTX 4060, ejecutando el proyecto desde el Editor de Unity con la plantilla para PC “PC_RPAsset”, además, de tener activado los efectos del post-procesado y niebla. Se realizaron cinco ejecuciones siendo las 2 primeras con técnicas de optimización como el occlusion culling y el layer culling distances desactivadas y teniendo variaciones teniendo la cámara en el centro del nivel, esto con el fin de observar el comportamiento del proyecto

bajo condiciones realistas de prueba y comprobar si la optimización causa un impacto significativo en el proyecto.

Ilustración 32

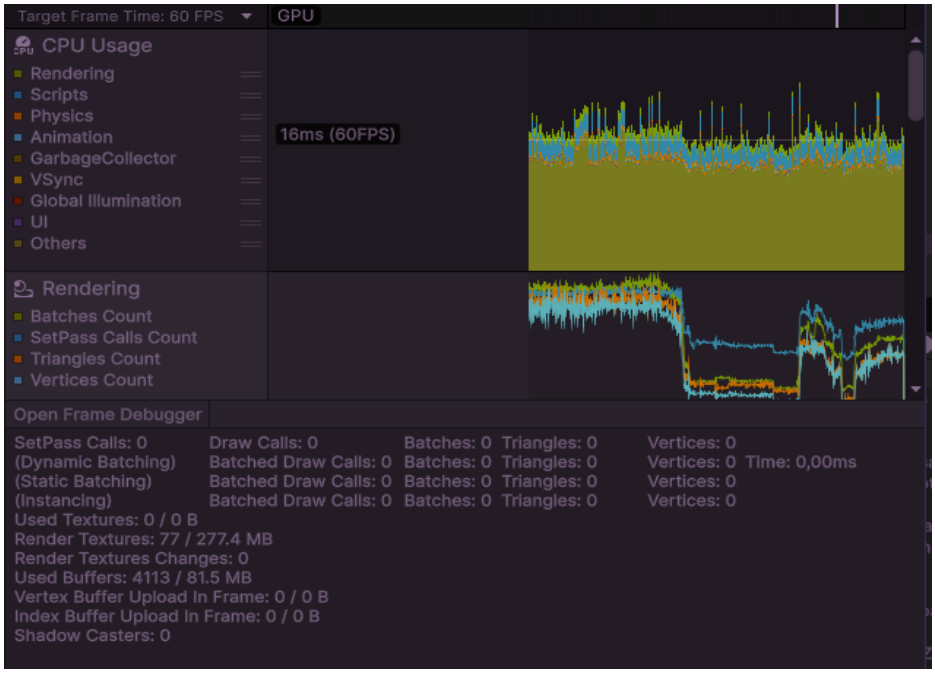
Visualización de la ventana de estadísticas



Nota: Elaboración propia

Ilustración 33

Visualización de la ventana de Profiler



Nota: Profiler donde podemos observar picos en el uso de la CPU en rendering y scripts, *Elaboración* propia

La Ilustración 33 muestra un ejemplo del análisis realizado mediante el Profiler durante la etapa de optimización. Esta herramienta facilitó la identificación de posibles cuellos de botella y permitió comprobar si la incorporación de nuevos recursos afectaba el rendimiento general del videojuego.

Así mismo se registraron los valores obtenidos durante las cinco ejecuciones realizadas con la herramienta de estadísticas (Stats). Se consideraron principalmente la tasa de FPS, el número de triángulos renderizados y el número de batches, debido a que estos indicadores permiten observar el comportamiento de la escena durante la ejecución y detectar posibles variaciones de rendimiento.

Tabla 9

Entorno para las pruebas de rendimiento

Ejecución	Condiciones	FPS	Triángulos (millones)	Batches
1	Sin optimización, personaje en la mitad del nivel	**80,4**	**29,3**	**13689**
2	Sin optimización, personaje en el inicio del nivel	**84,9**	**20,3**	**6959**
3	Optimizado, personaje en la mitad del nivel	**126,9**	**15,3**	**4117**
4	Optimizado, personaje en el inicio del nivel	**150,3**	**10,4**	**2722**
5	Optimizado, personaje en el inicio del nivel	**151,2**	**10,2**	**2700**

Nota: Elaboración propia

Los resultados obtenidos evidencian una mejora en el rendimiento del nivel después de aplicar las acciones de optimización sobre los assets y elementos integrados. En las pruebas realizadas con el personaje ubicado en la mitad del nivel, se obtuvo un incremento del FPS promedio de 80,4 a 126,9, equivalente a un aumento aproximado del 57,8 %, mientras que la cantidad de triángulos se redujo de 29,3 a 15,3 millones, representando una disminución del 47,8 %, y los batches pasaron de 13 689 a 4 117, lo que corresponde a una

reducción del 69,9 %. Por otra parte, considerando las pruebas realizadas desde el punto inicial del nivel, el FPS promedio pasó de 84,9 a 150,3, alcanzando un incremento aproximado del 77,0 %, acompañado de una reducción de 20,3 a 10,4 millones de triángulos, equivalente al 48,8 %, y de 6 959 a 2 722 batches, correspondiente a una reducción del 60,9 %. Finalmente, la quinta ejecución, realizada bajo las mismas condiciones de la cuarta, registró 151,2 FPS, 10,2 millones de triángulos y 2 700 batches, mostrando una variación mínima respecto a la prueba anterior y permitiendo comprobar la estabilidad del rendimiento alcanzado.

Una vez superadas las pruebas funcionales y de rendimiento, se realizó una revisión general del escenario para identificar posibles referencias perdidas, comportamientos inesperados o degradaciones en el desempeño del proyecto. En caso de detectar algún inconveniente, este fue corregido antes de continuar con la integración de nuevos recursos.

De forma complementaria, la evaluación cualitativa permitió observar el comportamiento de los sistemas durante la ejecución. Las pruebas permitieron comprobar la correcta respuesta de las interacciones, la activación de los elementos de interfaz, la reproducción de audio y la comunicación entre los diferentes sistemas integrados. Cuando se detectaron comportamientos no esperados, estos fueron registrados como incidencias y sometidos a una nueva prueba después de su corrección.

La validación se realizó de forma incremental durante todo el desarrollo de InkFall. Cada recurso fue verificado inmediatamente después de su integración, permitiendo detectar incompatibilidades, referencias perdidas y problemas de rendimiento antes de incorporar nuevos componentes. Este enfoque permitió documentar las incidencias encontradas, comprobar su corrección mediante nuevas ejecuciones y obtener evidencia tanto cuantitativa como cualitativa del comportamiento de los sistemas integrados.

4.8 Escalabilidad y mantenimiento

Para facilitar el mantenimiento del proyecto, conservar la estructura de integración utilizada durante el desarrollo. Antes de modificar una mecánica existente, identificar los sistemas con los que interactúa y comprobar que los cambios no afecten el funcionamiento del resto de los componentes.

Al incorporar nuevos recursos, evitar modificar directamente los archivos originales de los assets de terceros y mantener la organización de carpetas por libro (Book1, Book2, etc.) y por categoría (Art, Scripts, Audio, Assets y Core). Evitar almacenar recursos de un nivel dentro de otro, ya que esto dificulta el mantenimiento y la localización de archivos. Si es necesario realizar cambios en un modelo, Prefab o sistema importado, crear una variante o una copia del recurso y conservar siempre la versión original como respaldo. Este procedimiento facilita futuras actualizaciones y reduce el riesgo de perder configuraciones realizadas por el desarrollador original. Cuando se agreguen nuevas mecánicas, procurar reutilizar la arquitectura existente antes de implementar soluciones independientes. Siempre que sea posible, utilizar los administradores (Managers), eventos y componentes ya implementados para mantener una estructura consistente y reducir el acoplamiento entre sistemas.

Cuando un asset de terceros reciba una actualización, realizar las pruebas inicialmente sobre una copia del proyecto antes de reemplazar la versión utilizada durante el desarrollo, ya que las nuevas versiones pueden modificar prefabs, scripts o configuraciones existentes.

Después de realizar modificaciones importantes, ejecutar pruebas funcionales sobre la escena afectada antes de continuar con el desarrollo, también se recomienda crear un

commit o una copia de seguridad del estado actual del proyecto. Esto facilita la recuperación del sistema en caso de que los cambios introduzcan errores de integración.

Además de comprobar el funcionamiento de la nueva implementación, verificar que las mecánicas previamente integradas continúen respondiendo correctamente y que no existan errores de compilación, referencias perdidas o conflictos entre componentes.

En caso de incorporar nuevos escenarios o modificar niveles existentes, actualizar los elementos relacionados con la navegación mediante NavMesh, las colisiones y la iluminación cuando sea necesario. Posteriormente, generar nuevamente la malla de navegación y realizar pruebas de recorrido para comprobar que tanto el jugador como los personajes controlados por inteligencia artificial puedan desplazarse correctamente dentro del escenario. Además del código, documentar cualquier modificación relevante realizada sobre la arquitectura, la configuración de escenas, los Managers, los sistemas de eventos y las dependencias entre componentes. Mantener esta información actualizada facilita la incorporación de nuevos desarrolladores y permite continuar el proyecto sin necesidad de reconstruir el funcionamiento de los sistemas existentes.

Finalmente, una serie de buenas prácticas, recomendadas no solo para aplicar en InkFall, sino en cualquier proyecto relacionado con Unity o motores de juego son las dadas a continuación.

Tabla 10

Buenas prácticas de mantenimiento

Recomendación	Finalidad
No modificar assets originales	Facilita futuras actualizaciones
Mantener una estructura clara de carpetas	Evita desorganización
Reutilizar Managers existentes	Reduce el acoplamiento y duplicación entre escenas
Validar referencias antes de ejecutar	Evita referencias vacías y pérdida de tiempo
Ejecutar pruebas después de cada integración	Detecta errores tempranamente
Documentar cambios importantes	Facilita el mantenimiento

Nota: Elaboración propia

4.9 Conclusiones y recomendaciones

4.9.1 Conclusiones

La implementación del sistema de integración técnica en Unity permitió incorporar y coordinar los assets y sistemas desarrollados para InkFall dentro de una estructura común, obteniendo un funcionamiento integrado de las mecánicas, sistemas visuales, audio, interfaz y gestión de escenas. Los resultados de las pruebas funcionales y de rendimiento evidenciaron que la solución implementada permitió ejecutar el prototipo de acuerdo con los requerimientos establecidos para sus diferentes géneros y mecánicas.

El análisis de la estructura, compatibilidad y documentación de los recursos permitió identificar los requerimientos técnicos necesarios para su incorporación al proyecto y establecer criterios de integración aplicables a los diferentes assets y sistemas. Como resultado, fue posible incorporar recursos provenientes de distintas áreas del desarrollo manteniendo su compatibilidad con la estructura definida para el proyecto.

La arquitectura implementada mediante componentes, administradores, eventos y patrones de diseño permitió establecer mecanismos de comunicación entre los sistemas de jugabilidad, interfaz, audio y gestión de escenas. Las pruebas funcionales realizadas sobre

las interacciones entre estos sistemas evidenciaron un comportamiento coherente con los flujos de jugabilidad definidos para el prototipo.

La implementación de las mecánicas y la lógica de programación permitió integrar el control del jugador, las mecánicas específicas de los niveles, la inteligencia artificial, las armas y los eventos dentro de la estructura desarrollada. Las pruebas funcionales permitieron identificar y corregir errores de interacción y comportamiento, obteniendo un funcionamiento acorde con las mecánicas planteadas para cada nivel.

La configuración de los modelos 3D permitió incorporarlos al proyecto en conjunto con los sistemas de iluminación, físicas, animación, navegación y renderizado utilizados en Unity. Las verificaciones realizadas durante la integración y las pruebas funcionales posteriores permitieron comprobar su funcionamiento dentro de los diferentes escenarios del prototipo.

La integración de los sistemas de interfaz de usuario y audio permitió vincular sus elementos con la lógica del videojuego mediante los mecanismos de comunicación definidos en la arquitectura. Las pruebas funcionales evidenciaron una respuesta coherente de las interfaces y eventos sonoros ante las acciones del jugador y los cambios de estado del juego.

Finalmente, la validación mediante pruebas funcionales y análisis de rendimiento permitió comprobar el comportamiento de los sistemas integrados y evaluar el impacto de las medidas de optimización aplicadas. En las pruebas de rendimiento realizadas desde el inicio del nivel, el FPS aumentó de 84,9 a 150,3, correspondiente a un incremento del 77,0 %, con reducciones del 48,8 % en triángulos y del 60,9 % en batches. Estos resultados permiten establecer que las medidas de optimización aplicadas contribuyeron a mejorar el rendimiento del prototipo dentro del entorno de evaluación utilizado.

4.9.2 Recomendaciones

Para futuras etapas del desarrollo de InkFall se recomienda mantener la arquitectura de integración definida durante este trabajo, incorporando nuevos sistemas y recursos siguiendo los criterios establecidos para la organización del proyecto, la comunicación entre componentes y la reutilización de administradores, eventos y prefabs. Esto permitirá preservar la modularidad del software y reducir el impacto de futuras modificaciones sobre el resto de la aplicación.

Se recomienda verificar previamente la compatibilidad de cualquier asset o herramienta de terceros con la versión del motor Unity utilizada, el Universal Render Pipeline (URP) y las dependencias existentes en el proyecto antes de proceder con su incorporación. Asimismo, resulta conveniente mantener una estructura uniforme de carpetas, escenas y recursos para facilitar el mantenimiento y la localización de los diferentes elementos del videojuego.

Durante la incorporación de nuevas mecánicas, modelos o sistemas se recomienda reutilizar la arquitectura existente y evitar la creación de dependencias directas entre objetos de la escena. El empleo de administradores especializados, eventos y componentes reutilizables contribuye a mantener un bajo nivel de acoplamiento, facilita la escalabilidad del sistema y simplifica el mantenimiento del código conforme aumenta la complejidad del proyecto.

Es recomendable continuar realizando pruebas funcionales y evaluaciones periódicas de rendimiento durante todo el ciclo de desarrollo, utilizando herramientas como Unity Console, Stats y Profiler para detectar oportunamente problemas de integración, referencias perdidas o degradaciones en el desempeño del sistema antes de incorporar nuevas funcionalidades a la versión principal del proyecto.

Referencias

[1] J. Schell, *The Art of Game Design: A Book of Lenses*. Burlington, MA, EE. UU.: Morgan Kaufmann, 2008.

[2] I. Granic, A. Lobel y R. C. M. E. Engels, «The Benefits of Playing Video Games», *American Psychologist*, vol. 69, n.º 1, pp. 66–78, ene. 2014, doi: 10.1037/a0034857.

[3] E. Adams, *Fundamentals of Game Design*, 3.ª ed. Berkeley, CA, EE. UU.: New Riders, 2014.

[4] K. Collins, *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. Cambridge, MA, EE. UU.: MIT Press, 2008.

[5] Ropstam Game Studio, «Game Design: When Difficulty Goes Too Far», 2026. [En línea]. Disponible en:https://ropstamgames.com/difficult-games-vs-bad-game-design/ .

[6] P. de Byl, *Holistic Game Development with Unity: An All-in-One Guide to Implementing Game Mechanics, Art, Design and Programming*, 3.ª ed. Boca Raton, FL, EE. UU.: CRC Press, 2019.

[7] S. Eräniitty, «Creating a Game Scene in Unity», Tesis de grado, Häme University of Applied Sciences, Hämeenlinna, Finlandia, 2020.

[8] J. Pernas Álvarez, «Desarrollo de un simulador de eventos discretos experimental basado en realidad virtual», Trabajo final de máster, Escola Politécnica Superior, Universidade da Coruña, La Coruña, España, 2019.

[9] R. Nystrom, *Game Programming Patterns*. Genever Benning, 2014. [En línea]. Disponible en: https://gameprogrammingpatterns.com/contents.html .

[10] B. Nicoll y B. Keogh, T*he Unity Game Engine and the Circuits of Cultural Software*. Palgrave Pivot, 2019.

[11] M. Sukoinen, «Audio Implementation Methods in Unity», Tesis de grado, Turku University of Applied Sciences, Turku, Finlandia, 2019.

[12] L. Bass, P. Clements y R. Kazman, *Software Architecture in Practice*, 3.ª ed. Upper Saddle River, NJ, EE. UU.: Addison-Wesley, 2012.

[13] A. Thorn, *Pro Unity Game Development with C#*. NY, EE. UU.: Apress, 2014

[14] J. Gregory, *Game Engine Architecture*, 3.ª ed. Boca Raton, FL, EE. UU.: CRC Press, 2018.

[15] T. Jurvanen, «Automated Testing with Unity», Tesis de grado, Centria University of Applied Sciences,Kokkola, Finlandia, 2023.

[16] Carboneras Mas, «Simulador de eventos discretos para Unity». Trabajo final de máster, Universitat Politècnica de València, Valencia, España, 2019.[En línea]. Disponible en: https://riunet.upv.es/handle/10251/129933

[17] M. Fowler, «Inversion of Control Containers and the Dependency Injection pattern», 2004. [En línea]. Disponible en: https://martinfowler.com/articles/injection.html .

[18] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki y S. Hillaire, *Real-Time Rendering*, 4.ª ed. Boca Raton, FL, EE. UU.: CRC Press, 2018.

[19] Asana, «Kanban vs. Scrum: ¿Cuál es la diferencia?». [En línea]. Disponible en: https://asana.com/es/resources/waterfall-agile-kanban-scrum

[20] K. Beck et al., «Manifiesto por el Desarrollo Ágil de Software», 2001. [En línea]. Disponible en: https://agilemanifesto.org/iso/es/manifesto.html.

[21] A. Sastre, «Programar con Unity vs. Unreal Engine: ¡pros y contras!». [En línea]. Disponible en: https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/pros-contras-programar-unity-vs-unreal-engine .
[22] Unity Technologies, «Unity 6.5 User Manual», 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/index.html .
[23] Unity Technologies, «Assets and media», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/assets-and-media.html .
[24] R. Navarro Estrella, «Desarrollo de un videojuego RPG en Unity», Trabajo fin de grado, Universidad de Alicante, Alicante, España, 2018.
[25] I. Millington, *Artificial Intelligence for Games*. San Francisco, CA, EE. UU.: Morgan Kaufmann, 2006.
[26] Unity Technologies, «Introduction to assets in Unity», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/AssetWorkflow.html .
[27] Unity Technologies, «Introduction to GameObjects», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/GameObjects.html .
[28] P. J. García Ruiz, «Sobreviviendo como desarrollador indie en Unity», Trabajo fin de grado, Universidad de Alicante, Alicante, España, 2020.
[29] Unity Technologies, «Prefabs», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/Prefabs.html .
[30] N. Amaya-Olarte, M. L. Torres-Barreto y K. R. Plata-Gómez, «Análisis de una experiencia de aprendizaje basada en juegos digitales», Revista Electrónica de Investigación Educativa, vol. 26, n.º e08, pp. 1-15, 2024.
[31] Unity Technologies, «UI Toolkit», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/UIElements.html .
[32] Unity Technologies, «Event function execution order», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/execution-order.html .
[33] P. Barry y D. Griffiths, *Head First Programming*. Sebastopol, CA, EE. UU.: O'Reilly Media, 2009.
[34] F. Lanzinger, 2*D Game Development with Unity*. Boca Raton, FL, EE. UU.: CRC Press, 2021.
[35] Unity Technologies, «Inspector-configurable custom events», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/unity-events.html .
[36] I. Millington, *Game Physics Engine Development*. Burlington, MA, EE. UU.: Morgan Kaufmann, 2007.
[37] Unity Technologies, «Reduce rendering work on the CPU or GPU», Unity User Manual, 2026. [En línea]. Disponible en: https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html .