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

Construcción y programación de un entorno tridimensional interactivo y funcional para un videojuego de computadora mediante Unity

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

AUTOR: RICARDO ESTEBAN VÁZQUEZ BANEGAS

DIRECTOR: ING. JHEYSON STEVEN GAONA PINEDA, MTR.

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

Construcción y programación de un entorno tridimensional interactivo y funcional para un videojuego de computadora mediante Unity

PROYECTO DE INTEGRACION CURRICULAR PREVIO A LA OBTENCION DEL TITULO DE INGENIERO EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTOR: RICARDO ESTEBAN VÁZQUEZ BANEGAS

DIRECTOR: ING. JHEYSON STEVEN GAONA PINEDA, MTR.

CUENCA - ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca

DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD

Declaratoria de Autoría y Responsabilidad

Ricardo Esteban Vazquez Banegas portador de la cédula de ciudadanía N.º **0106250913**. Declaro ser el autor de la obra: **"Construcción Y Programación De Un Entorno Tridimensional Interactivo Y Funcional Para Un Videojuego De Computadora Mediante Unity"**, 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 del 2026**

F: .....

Ricardo Esteban Vazquez Banegas

C.I. **0106250913**

Universidad Católica de Cuenca

CERTIFICADO

Certificado

Certifico que el presente Trabajo de Investigación fue desarrollado por **Ricardo Esteban Vazquez Banegas** portador(a) de la cédula de ciudadanía N.º **0106250913** con el tema **"Construcción y programación de un entorno tridimensional interactivo y funcional para un videojuego de computadora mediante Unity"**, bajo mi supervisión.

F: .....

Ing. Jheyson Steven Gaona Pineda, Mtr.

Docente - Tutor

Dedicatoria

A mi mamá Tania y a mi papá René, por ser mi mayor soporte. Gracias por enseñarme a ser buena persona, por guiarme y por apoyarme incondicionalmente sin importar lo difícil que se pusiera el camino. Todo el esfuerzo que han hecho por mí toda la vida se resume en esto.

A Cristina y Paúl, por ser los mejores hermanos que la vida me pudo haber dado. Gracias por escucharme, por haber estado a mi lado y por darme siempre los ánimos precisos para seguir adelante sin rendirme.

A mi abuelita Guillermina, el corazón de la familia. Gracias por confiar en mí (a veces más de lo que yo mismo lo hago), por querernos tanto a mí y a mis hermanos, y por darme el amor más puro del mundo. Esto también es tuyo, abuelita.

Este trabajo lleva un pedacito de cada uno de ustedes, porque fueron quienes me sostuvieron durante todo este tiempo.

Agradecimiento

Al haber conseguido uno de mis logros más importantes de mi vida, quiero agradecer a mis seres queridos por ser un apoyo incondicional en mi vida y ayudarme a alcanzar este logro importante.

Agradezco a mis padres por el apoyo y cariño que siempre me han brindado siempre, por haberme enseñado a nunca rendirme y haber sido unos grandes ejemplos para seguir, todos mis logros también son los suyos

Agradezco a mi ñaña Cris, la cual siempre confió en mí en cada momento, y la cual me apoyó para poder culminar mi etapa universitaria, que a pesar de la distancia siempre ha estado pendiente.

Agradezco a mi abuelita por el cariño que siempre me ha dado, por las veces que ha orado por mi para llegar a ser un profesional, y por haber sido mi segunda madre

Agradezco también a mis amigos los cuales siempre me han apoyado a seguir adelante y cumplir todos mis sueños y metas

Resumen

El desarrollo de entornos interactivos es clave para construir experiencias inmersivas, lográndolo mediante la integración continua entre el diseño de nivel que da forma y estructura al mundo virtual y la programación que aporta lógica, mecánicas y funcionalidad al espacio jugable, transformando así un escenario estático en un entorno vivo y dinámico. Este proceso implica planificar el terreno, distribuir estratégicamente los elementos del mapa e implementar sistemas de interacción fluida, equilibrando escala, navegabilidad y ambientación. Este proyecto describe la creación del entorno del videojuego "Ashes of Faith", desarrollando el Level Design a partir de un mockup conceptual, junto con la programación integral de su lógica en Unity y C#. El trabajo se centra en tres pilares: (i) programación del entorno espacial y su interactividad, (ii) implementación de la interfaz de usuario, y (iii) optimización del rendimiento, garantizando una visualización fluida sin fatiga visual. El desarrollo se organiza bajo la metodología ágil Scrum, en ciclos cortos que facilita ajustes rápidamente al nivel y sus mecánicas ante cambios imprevistos. Para validar el entorno se realizan pruebas de rendimiento con Unity Profiler (FPS y uso de recursos) y pruebas de usabilidad con un grupo piloto, midiendo tiempos y dificultades de navegación. Los resultados mostraron una experiencia satisfactoria, con rendimiento estable y usuarios capaces de completar las tareas sin mayores dificultades, confirmando el cumplimiento de los estándares técnicos y de usabilidad establecidos.

Palabras Clave: *Diseño de nivel, Programación de entorno, Usabilidad, Unity.*

Abstract

The development of interactive environments is key to creating immersive experiences. This is achieved through the continued integration of level design, which gives shape and structure to the virtual world, and programming, which provides logic, mechanics, and functionality to the playable space, thereby transforming a static setting into a living, dynamic environment. This process involves planning the terrain, strategically distributing map elements, and implementing fluid interaction systems, while balancing scale, navigability, and atmosphere. This project describes the creation of the environment for the video game "Ashes of Faith," from a conceptual mockup to the level design and the comprehensive programming of its logic in Unity and C#. The work focuses on three pillars: (i) programming the spatial environment and its interactivity, (ii) implementing the user interface, and (iii) optimizing performance to ensure smooth rendering without eye strain. Development is organized using the Scrum agile methodology, in short cycles that facilitate rapid adjustments to the level and its mechanics in response to unforeseen changes. To validate the environment, performance tests were conducted using the Unity Profiler (FPS and resource usage), and usability tests were conducted with a pilot group to measure navigation times and difficulties. The results showed a satisfactory experience, with stable performance and users able to complete tasks without major difficulties, confirming compliance with the established technical and usability standards.

Keywords: *Level Design, Environment Programming, Usability, Unity.*

Índice de contenido

Índice de tablas

Índice de ilustraciones

Introducción

En la actualidad, el desarrollo de entornos virtuales interactivos ha adquirido gran relevancia debido al crecimiento de la industria de los videojuegos y a la necesidad de construir experiencias inmersivas para los usuarios. No obstante, a menudo se ostenta la calidad gráfica en los escenarios tridimensionales en función de los métodos de diseño aplicados que tienen como punto de partida la elaboración de un jugador típico y lo que hace que existan barreras mecánicas, confusión espacial y una gran carga cognitiva en la exploración. Frente a esta realidad, surge la necesidad de diseñar un entorno tridimensional funcional y accesible para el videojuego "Ashes of Faith", sustentado en el motor gráfico Unity y el lenguaje C#, que optimice los procesos de interacción y navegación, mejore la experiencia del jugador y establezca bases sólidas de diseño de niveles (Level Design).

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

Capítulo I: En el presente capítulo es donde se comienza a desarrollar el marco referencial de la investigación, se plantea y, por lo tanto, se formula el problema. Se describe la justificación de este, los objetivos generales y específicos, el alcance técnico y la metodología ágil utilizada para la planificación y el desarrollo del proyecto.

Capítulo II: Se desarrolla el marco teórico en donde se abordan los principales conceptos que fundamentan el estudio, tales como el estado del arte del diseño de niveles y el análisis comparativo de trabajos relacionados en la industria. Asimismo, se exponen las características de las tecnologías, motores gráficos (Unity, Unreal Engine), lenguajes de programación (C#, C++) y metodologías de desarrollo (Scrum) que sustentan la construcción del entorno.

Capítulo III: Se describe el manual de usuario, en el cual se detallan los requisitos técnicos mínimos del sistema, las instrucciones para la instalación y ejecución del aplicativo, así como el esquema de controles. Adicionalmente en este capítulo se desarrolla el funcionamiento de la interfaz gráfica y las mecánicas de usabilidad ambiental para que el jugador final llegue a tener un uso adecuado de la navegación del juego.

Capítulo IV: Se presenta el manual del programador, en el cual se detalla el diseño y la arquitectura del modelo de software adaptado al entorno del videojuego. Mediante la programación en C#, se documenta el ensamblaje de los scripts, la organización de las escenas y la integración de las mallas de navegación, finalizando con la exposición de las pruebas funcionales, de rendimiento y de usabilidad aplicadas para validar el sistema.

Capítulo I

1. Marco Referencial

1.1. Contextualización del problema

En el siguiente punto se describe con claridad la situación y la necesidad técnica que dan origen a este proyecto. Se muestran los principales problemas de usabilidad y las limitaciones presentes en los motores gráficos, las cuales, el producto busca resolver para mejorar la experiencia optima del jugador.

El desarrollo de los videojuegos ha llegado a ser un ámbito que no duda en crecer más dentro del ámbito informático, aunque en la realización de mundos en 3D para ordenador, casi siempre se tiene como referente a un jugador medio[1], [2]. Esto hace que se produzcan grandes obstáculos de usabilidad que generan la exclusión de un colectivo importante de personas [1], [2]. Durante mucho tiempo, la accesibilidad y la facilidad de uso se han considerado como un arreglo superficial de última hora, en vez de ser enunciadas como una de las bases desde el principio de un proyecto y una determinación de la experiencia de usuario [3], [4].

1.2. Planteamiento del problema

Al analizar los juegos de exploración en tercera persona, se observa que entender el espacio, medir la profundidad y procesar los cambios de una iluminación muy realista requiere un esfuerzo mental enorme. La elevada carga cognitiva llega a colmar, a frustrar y a confundir totalmente a quienes se encuentran jugando debido al movimiento a través de mapas amplios[5], [6]. A ello hay que añadirle que los controles tradicionales son muy rígidos

y castigan severamente cualquier error por causa de imprecisión o lentitud al accionar sus teclas, sometiendo a los jugadores a unas severas barreras mecánicas que limitan de una forma u otra la forma de jugar[7], [8].

Esta dificultad cobra mayor importancia al trabajar con herramientas como el motor Unity, donde es muy común que la creación de los niveles le dé más prioridad a que el juego se vea estéticamente bien antes de asegurar que sea funcional y claro para la vista[9], [10]. El problema empeora debido a la falta de reglas técnicas desde la fase de preparación, lo que lleva a construir escenarios con zonas de choque invisibles muy duras y sistemas que no dan ninguna opción de flexibilidad para que el usuario ajuste los controles a su comodidad[11], [12].

1.3. Formulación del problema

Todo esto hace que el juego mantenga problemas clásicos y tenga una curva de dificultad totalmente artificial [13], [14]. Si no se atiende esta necesidad mediante herramientas que suavicen las barreras motoras y visuales, el juego terminará desorientando y excluyendo a una gran cantidad de personas, arruinando por completo su experiencia al interactuar con el mundo virtual[15], [16].

Pregunta: ¿De qué manera la programación de un entorno tridimensional y la implementación de una arquitectura modular basada en componentes contribuyen a la percepción de usabilidad y accesibilidad espacial en los usuarios del videojuego Ashes of Faith?

1.4. Justificación

El desarrollo del videojuego "Ashes of Faith" surge de la necesidad de mejorar la funcionalidad y la forma en la que se perciben los mundos virtuales construidos para computadora. En la actualidad, no es suficiente con ofrecer gráficos de alta calidad; el juego debe ser fácil de entender, fluido y estar pensado completamente en la comodidad de quien lo utiliza[1], [10]. Cuando el diseño del escenario, la configuración de los controles y los menús no trabajan en conjunto de forma armónica, el jugador se confunde rápidamente y la experiencia de juego fracasa[7], [17]. Por esta razón, el proyecto busca unificar la construcción del entorno tridimensional con respuestas visuales claras y un rendimiento óptimo de la máquina, garantizando así una interacción sin interrupciones[5], [16].

Desde una perspectiva social, la incorporación de opciones de inclusión permite que el entretenimiento digital llegue a un público mucho más amplio[8], [15]. Mediante la eliminación de las barreras técnicas, las personas con alguna limitación tanto física como cognitiva podrán vivir la aventura como cualquier otro, pero, a su vez, se favorecen espacios de ocio más justos y equitativos. [6], [13]. La meta principal de dicho mecanismo es la de cambiar la forma convencional (histórica, ortodoxa) mediante la que solemos navegar por esos mundos alternativos, explorando un tipo de interacción mucho más natural o amigable con el mundo virtual[3], [18]. De esta forma, la empatía deja de ser un concepto secundario y se convierte en una pieza fundamental del diseño de software [8], [15].

Finalmente, en el ámbito educativo e institucional, este trabajo se plantea como un ejemplo práctico de gran utilidad para la facultad. El proyecto ofrece una guía real y aplicada sobre cómo estructurar entornos digitales que funcionen correctamente desde su base técnica,

sirviendo como material de apoyo e inspiración para los futuros desarrolladores de la carrera[9], [11], [19]. Al documentar todo este proceso, se deja un recurso valioso que demuestra cómo aplicar metodologías de diseño accesible en motores gráficos reales, enriqueciendo el aprendizaje práctico de otros estudiantes[11], [19].

1.5. Objetivo General

Desarrollar un entorno tridimensional interactivo para el videojuego de computadora "Ashes of Faith" mediante el uso del motor gráfico Unity y el lenguaje C#, para garantizar una experiencia de navegación, interacción y rendimiento óptima para el usuario.

1.6. Objetivos Específicos

- Analizar las barreras de interacción y los requisitos técnicos de los entornos 3D mediante la revisión de normativas de usabilidad, para determinar los criterios de accesibilidad y diseño universal que guiarán la construcción del prototipo.
- Programar la lógica del entorno 3D implementando sistemas de colisión tolerantes, físicas adaptativas y carga asíncrona, para asegurar una navegación espacial fluida e ininterrumpida por parte del jugador.
- Implementar una interfaz de usuario escalable y de alto contraste mediante la estructuración de menús accesibles, para reducir la carga cognitiva y el esfuerzo mental al configurar y explorar el juego.
- Optimizar el rendimiento técnico del prototipo aplicando técnicas de perfilado (*profiling*) y gestión de memoria, para mantener una tasa de fotogramas estable que prevenga la desorientación visual del usuario.

- Evaluar la usabilidad y la accesibilidad del entorno y de la interfaz. Realizar pruebas de rendimiento y aplicamos las pautas de diseño universal.

1.7. Alcance del proyecto

En esta sección se establecen los límites técnicos del proyecto, definiendo con claridad hasta dónde llegará la implementación del software. A continuación, se explican los módulos de trabajo específicos que se desarrollarán para el videojuego, así como aquellas áreas de producción que no se usarán, garantizando así un enfoque realista y centrado en la ingeniería.

El alcance de este trabajo se enfoca estrictamente en las tareas de ingeniería de software para el prototipo del videojuego "Ashes of Faith" en computadora, asumiendo los roles de programación del entorno, diseño de interfaz y optimización técnica[10], [20]. Para organizar el desarrollo en Unity, el sistema se estructura en los siguientes cuatro módulos técnicos:

- **Módulo 1: Programación del entorno y navegación.** Este primer módulo aborda el ensamblaje técnico de los escenarios tridimensionales. Se utilizarán mallas de navegación inteligente (NavMesh) para asegurar rutas de exploración claras y espacialmente guiadas[4], [18]. Además, contempla el desarrollo de sistemas de carga asíncrona (Scene Management) para crear transiciones fluidas entre zonas y evitar pantallas de espera abruptas que rompan la inmersión del jugador [20].
- **Módulo 2: Físicas y colisiones tolerantes.** El segundo módulo se enfoca en programar las interacciones físicas del entorno. Se implementarán áreas de choque (hitboxes) dinámicas y con tolerancia suficiente para que en los momentos indicados

no se castiguen los problemas de motricidad del jugador al saltar, escapándose o manipulando los objetos, con lo cual se recortan las barreras mecánicas[7], [20].

- **Módulo 3: Interfaz de usuario (UI/UX) y audio espacial**. El tercer módulo incluye el desarrollo del entorno visual del sistema (Front-end) y los feedback sonoros. Se implementará una interfaz de usuario escalable orientada a la usabilidad básica y el alto contraste estático, dejando opciones de remapeo personalizable como alcance para trabajos futuros.
- **Módulo 4: Optimización y rendimiento técnico.** El último módulo está destinado a asegurar la fluidez y estabilidad del prototipo. Se aplicarán técnicas de ocultamiento de geometría no visible (Occlusion Culling) y perfilado de código constante (Profiling) para evitar fugas de memoria, asegurando un consumo eficiente de recursos y una tasa de fotogramas estable que prevenga la fatiga visual interactiva [10].

1.8. Delimitación

Para asegurar el cumplimiento de los objetivos técnicos, es fundamental delimitar las áreas que quedan fuera de este proyecto. No se abordará la creación de recursos artísticos desde cero, lo que excluye el modelado 3D, el texturizado, las animaciones, la composición musical y la redacción narrativa[10]. De igual manera, la programación de inteligencias artificiales (IA) complejas para los enemigos no formará parte de los entregables, ya que todos los esfuerzos de desarrollo estarán dedicados a estructurar un entorno accesible, una interfaz clara y un rendimiento óptimo [4], [20].

1.9. Beneficiarios

La ejecución del presente proyecto genera un impacto positivo que se califica según el grado de relación con el producto final desarrollado:

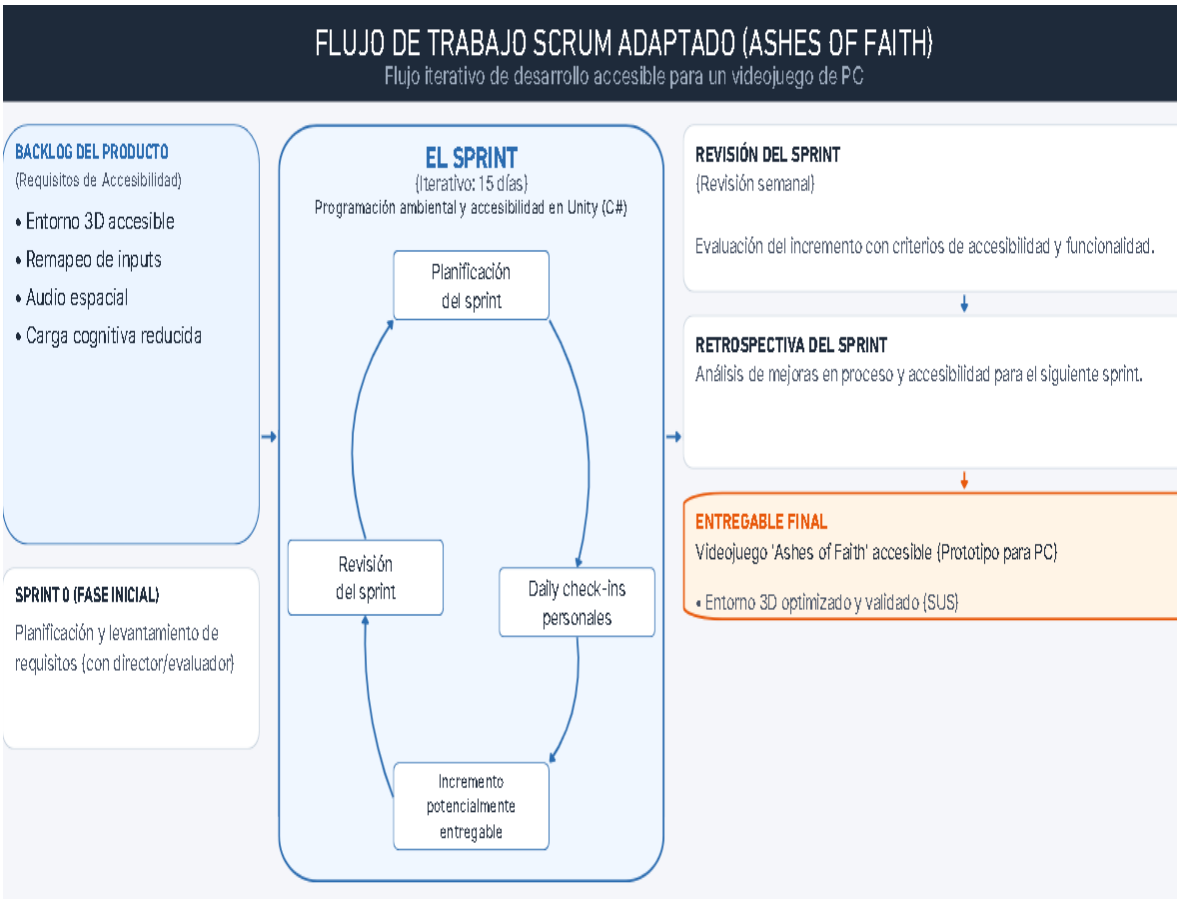
- **Beneficiarios indirectos:** Los beneficiarios indirectos se identifican a los futuros desarrolladores de software y diseñadores de niveles (Level Designers). A través de la documentación generada en el proyecto, se les proporciona una base técnica validada empíricamente sobre como estructurar, programar y optimizar la usabilidad de entornos 3D.
- **Beneficiarios directos**: Son los jugadores de computadora, quienes obtendrán una experiencia interactiva más accesible al reducirse la desorientación espacial y la sobrecarga cognitiva en entornos 3D.

1.10. Metodología general

En este apartado se demuestra la estructura de trabajo y los procedimientos que guiarán el desarrollo del proyecto. Se explica el ambiente de trabajo seleccionado, el acoplamiento de los roles de producción y las fases secuenciales de los "sprints", como objetivo de enseñar cómo se agregarán las prácticas de diseño alcanzable en cada fase de creación del videojuego.

Ilustración 1

Representación gráfica de flujo de trabajo Scrum de creación propia



Nota: Elaboración propia

La metodología usada para el desarrollo de la programación del entorno y las características de usabilidad se apoya en el marco de trabajo ágil Scrum. Este planteamiento ordena el ciclo de vida del software en iteraciones pequeñas, continuas y progresivas llamadas "sprints". La naturaleza repetitiva de Scrum resulta altamente recomendada en el desarrollo de videojuegos interactivos y sistemas de accesibilidad, ya que permite adaptar las mecánicas y la construcción del motor frente a cambios o hallazgos imprevistos obtenidos de las pruebas de usabilidad[3], [10].

Dado el contexto de este trabajo, el marco tradicional de Scrum se ha adaptado estructuralmente para un equipo de desarrollo unipersonal. En este ejemplo, el investigador

se encarga del rol principal de "Development Team" (encargado de la programación, el diseño de niveles en Unity y la implementación técnica). El rol de "Product Owner" será representado de forma externa por el director del proyecto y los pares revisores, quienes se encargarán de validar los incrementos, priorizar los requerimientos técnicos y aprobar los módulos establecidos[21]. En lugar de las reuniones diarias, se establecerán revisiones semanales de seguimiento para evaluar el porcentaje de progreso del prototipo y planificar ajustes en la programación.

1.11. Resultados esperados

1. Fase de análisis y requisitos (Sprint 0). En esta etapa inicial se establecen las bases técnicas del proyecto. Se enfocará en detallar el "Product Backlog" atendiendo únicamente en las barreras de interacción espacial. Para aquello, se realizará un análisis comparativo utilizando las normas "Xbox Accessibility Guidelines (XAGs)" y las "Game Accessibility Guidelines"[4], [20]. En lugar de un desarrollo completamente práctico, se usarán vocabularios de diseño y evaluación heurística temprana para identificar los problemas de carga cognitiva y visual que presenta el diseño de niveles habitual en computadora [22].

2. Fase de diseño y arquitectura modular (Sprints 1 y 2). En estos periodos, la prioridad será configurar la lógica principal en Unity para asegurar que todo corra sin problemas. Se creará una arquitectura de software que permita separar la lógica central del juego de los sistemas de ayuda y entrada de datos [10]. Simultáneamente, se realizará el "blockout"(caja gris) del entorno 3D, creando la estructura de nivel bajo guías de inmersión y accesibilidad para asegurar que los espacios ofrezcan un margen de tolerancia adecuado [9].

3. Fase de desarrollo e implementación (Sprints 3 al 4). Esta es la fase central de recopilación, donde se concretan las secciones técnicas. Se programarán shaders personalizados y filtros adaptativos para lograr un entorno visual de alto contraste y reducir la sobrecarga cognitiva[5], [6]. Se agregará métodos de audio espacial para ayudar al jugador en la movilidad y localización de objetivos[18], [23]. Finalmente, se implementarán los scripts de navegación y el soporte para el remapeo de controles, disminuyendo la fatiga motora[7].

4. Fase de pruebas, validación y refinamiento (Sprints 4 y 5). En las iteraciones finales, se compilará el prototipo para su realización en entornos locales. La validación del sistema se estructurará mediante pruebas cualitativas de jugabilidad con un grupo piloto de usuarios, observando empíricamente su respuesta ante las mallas de colisión y el entorno interactivo[13]. Para evaluar la capacidad técnica y el nivel de accesibilidad alcanzado, se aplicarán métricas de usabilidad específicas. Estas permitirán realizar ciclos de pruebas para pulir el código, solucionar problemas y perfeccionar la interacción antes de la compilación definitiva [2].

Capítulo II

2. Marco teórico

Este capítulo apoya el proyecto con investigación teórica y técnica enfocada en el diseño de nivel. El capítulo describe, las tecnologías, las metodologías y los referentes que el proyecto usa para crear la programación ambiental, la interfaz de usuario y la optimización de rendimiento en el motor gráfico Unity.

2.1. Antecedentes de investigación

En este apartado se analizan proyectos reconocidos de la industria de los videojuegos que han enfrentado y resuelto problemas de usabilidad, sirviendo como guía fundamental para el desarrollo del presente proyecto.

2.1.1. Shadow of the Tomb Raider (2018)

Es un videojuego de acción y aventura en tercera persona desarrollado por el estudio Eidos-Montréal. Este título se lanzó al mercado permitiendo al jugador la capacidad de explorar selvas densas, ruinas antiguas y tumbas oscuras llenas de acertijos, enfrentándose a un entorno natural muy detallado y complejo que exige un alto nivel de observación [24].

Seguidamente se enumeran temas de interés con respecto a este videojuego, con el propósito de ofrecer una visión exhaustiva y amplia de la extensa gama de vivencias que proporciona Shadow of the Tomb Raider ante la desorientación del jugador al momento de la exploración del entorno:

- **Mitigación de la sobrecarga visual:** A diferencia de otros juegos de aventura, Shadow of the Tomb Raider se centró en resolver el problema del cansancio mental que genera el exceso de detalles en pantalla, ayudando al jugador a identificar qué elementos son decorativos y cuáles son interactivos.
- **Personalización modular de la dificultad:** Innovó al romper la clásica selección de dificultad global (Fácil, Normal, Difícil). Permitió separar la experiencia en tres pilares, brindándole el control exacto de cuánta ayuda visual desea usar el jugador, aparte de la dificultad del combate.
- **Señalización no intrusiva:** Su enfoque de diseño evitó llenar la pantalla con flechas o menús flotantes, optando por agregar pistas de navegación directamente sobre las texturas del mundo real (como pintura blanca desgastada en los bordes de las montañas).

El sistema de legibilidad y asistencia en Shadow of the Tomb Raider es dinámico y versátil, lo que facilita a los jugadores ajustar el entorno visual a sus necesidades. Aquí se muestra un resumen detallado de cómo funciona en el entorno de juego:

- **Pistas en el escenario:** Se programaron variables que cambian las texturas del ambiente de forma dinámica, agregando manchas de pintura o relieves brillantes para marcar las rutas seguras de escalada solo si el jugador activa la asistencia.
- **Audio de proximidad:** Emite un sonido tenue cuando hay un objeto importante o cuando hay una ruta próxima, lo que permite conocer el camino correcto sin consultar el mapa en cada momento.

- **Tiempo extra de reacción:** Da más margen de tiempo cuando el personaje debe saltar o agarrarse de un borde, evitando que el personaje caiga si se presiona el botón un poco tarde.
- **Resultados y aportes:** Los resultados obtenidos mostraron que permitir al jugador escoger la legibilidad del entorno reduce drásticamente la frustración y la desorientación. El presente trabajo establece la realización de una organización modulada la que hace referencia que, al desarrollar un menú interno, las opciones a seleccionar permiten el cambio o bien la modificación del contraste visual sin que esto cambie nada con respecto a las reglas básicas del juego

Seguidamente la ilustración 2, muestra la perspectiva sobre ruta guiada, evidenciando su estilo artisco a nivel de detalle sobre la estructura de la superficie, evidecnia un camino y objeto interactuable.

Ilustración 2

Presentación visual de la jugabilidad en Shadow of the Tomb Raider

Nota: Obtenido de Eidos Montréal [24]

2.1.2. Gears 5 (2019)

Es un videojuego de disparos en tercera persona y acción intensa desarrollado por The Coalition. El título exige a los jugadores avanzar en equipo o en solitario enfrentándose a hordas de enemigos en escenarios destruidos y caóticos, requiriendo tradicionalmente reflejos rápidos y una coordinación precisa con el mando [25]. Aquí se enumeran temas de interés únicos con respecto a este videojuego, con el propósito de ofrecer una visión exhaustiva de su aporte específico a la usabilidad:

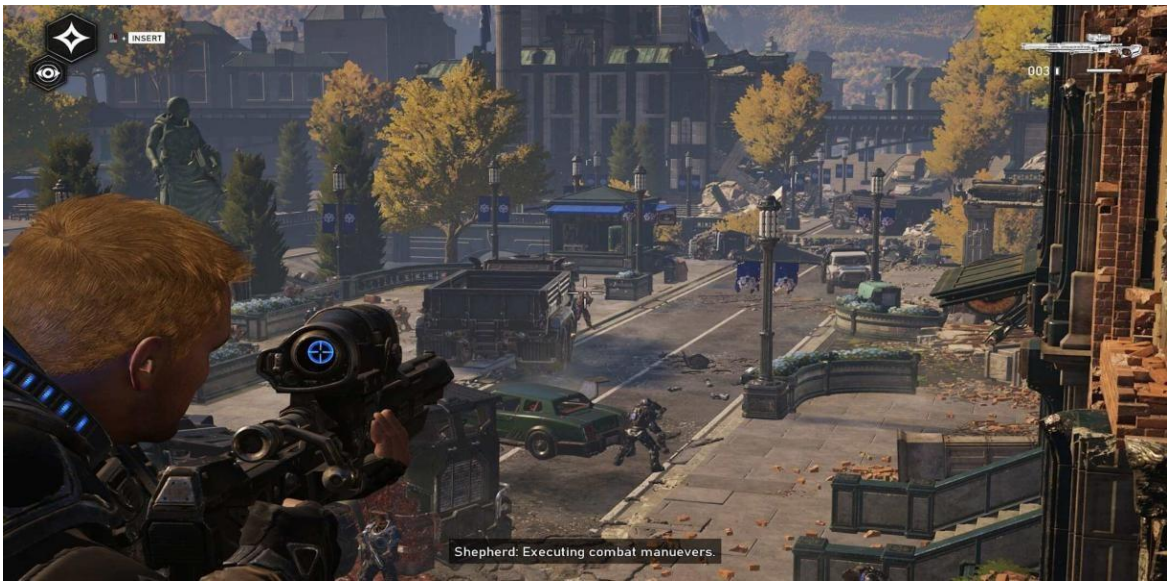
- **Accesibilidad motriz profunda:** Su innovación principal se estableció en adaptar un género que usualmente exige mucha destreza física para que sea totalmente jugable por personas con movilidad limitada en sus manos o dedos.
- **Reducción de la fatiga física:** El juego eliminó por completo los momentos donde se exige presionar un botón a gran velocidad repetidamente para sobrevivir (Quick Time Events), transformándolos en acciones de mantener presionado un solo botón, cuidando la integridad física del usuario.
- **Tolerancia algorítmica al error:** En lugar de exigir que el jugador sea un experto apuntando, el sistema asume parte de la carga mecánica, perdonando la falta de precisión milimétrica durante los momentos de alta tensión.

El sistema de combate y movilidad en Gears 5 se adaptó para ser altamente tolerante con los errores mecánicos del usuario. En esta sección se presenta un resumen detallado de su funcionamiento:

- **Zonas de impacto perdonables:** En la programación, ampliaron las zonas de impacto (hitboxes) de los enemigos. Entendiendo que el juego perdona al jugador y registra aun así el impacto incluso si su puntería no es perfecta.
- **Asistencia al apuntar:** Agregaron un sistema de ayuda al apuntado de objetivo que funciona como una especie de magnetismo a la mira del jugador usando cálculos espaciales.
- **Seguimiento automático de cámara:** Permite que la cámara se mantenga fija en el enemigo más cercano, evitando la necesidad de mover la cámara constantemente para no perderlo de vista.
- **Avisos visuales de peligro:** Muestra indicaciones muy claras en la pantalla avisando al jugador en que dirección vienen los diferentes ataques enemigos, ayudando a reaccionar a tiempo sin depender solo del sonido.
- **Resultados y aportes:** El juego logró retener a una gran cantidad de personas con limitaciones de movilidad fina.

Ilustración 3

Presentación visual de la jugabilidad en Gears 5



Nota: Obtenido de The Coalition. [25]

2.1.3. The Last of Us Part II (2020):

Es un videojuego de acción, aventura y supervivencia dramática desarrollado por Naughty Dog. Este título sumerge al jugador en un mundo donde la exploración silenciosa, la gestión de recursos limitados y los enfrentamientos sorpresa obligan a estar en alerta máxima al entorno tridimensional[26]. A continuación, se menciona los temas de interés únicos con respecto a este videojuego, con el propósito de ofrecer una visión exhaustiva de su aporte específico a la usabilidad:

- **Independencia sin retroalimentación visual:** Su mayor hito fue diseñar un sistema que permite a un jugador con ceguera total completar la historia de principio a fin, moverse por el mundo y defenderse sin necesitar la asistencia de una persona vidente a su lado.

- **Sustitución sensorial mediante audio:** El juego logró transformar la información que normalmente se percibe con los ojos (como la distancia de un obstáculo o la ubicación de una puerta) en un mapa sonoro con una precisión demasiado alta.
- **Abstracción gráfica por alto contraste:** Implementó una solución técnica para la poca visibilidad que elimina todo el fotorrealismo del juego y lo convierte en un lienzo gris, resaltando únicamente lo que tiene utilidad interactiva con colores puros.

El sistema de navegación de The Last of Us Part II demuestra que la orientación espacial puede reemplazarse por completo de la información visual. Enseguida se presenta un resumen detallado de su funcionamiento:

- **Sonidos que guían (Audio 3D):** Sincronizaron el camino invisible por donde camina el personaje con un sonido espacial direccional (HRTF) que emite una señal para guiar al jugador paso a paso.
- **Modo de alto contraste:** Pinta todo el fondo de color gris liso y resalta a los enemigos y objetos importantes con colores brillantes (como rojo o amarillo), haciendo que sea muy fácil saber con qué interactuar.
- **Escaneo del entorno:** Al presionar un botón, el juego envía un radar invisible que detecta objetos útiles a través de las paredes y avisa mediante un sonido o una vibración en el control.
- **Resultados y aportes:** Demostró que programar ayudas visuales y sonoras directas en el entorno permite a cualquier usuario orientarse en el espacio.

Ilustración 4

Presentación visual de la jugabilidad en The Last of Us II

Nota: Obtenido de Naughty Dog[26]

2.1.4. God of War Ragnarök (2022):

Es un videojuego de acción y aventura en tercera persona desarrollado por Santa Monica Studio. El juego permite al jugador la capacidad de explorar reinos extensos de la mitología nórdica, resolver acertijos ambientales complejos y combatir de forma dinámica contra criaturas colosales en mapas de gran magnitud[27]. Enseguida se enumeran temas de interés únicos con respecto a este videojuego, con el propósito de ofrecer una visión exhaustiva de su aporte específico a la usabilidad:

- **Orientación activa en mapas no lineales:** Su aporte único fue resolver el problema de la frustración por extravío en niveles enormes e intrincados, sin romper la inmersión del jugador ni obligarlo a mirar un mapa en pausa constantemente.
- **Automatización de la perspectiva:** A diferencia de otros juegos que solo dibujan una línea en el suelo para guiar al jugador, este título utilizó la cámara del propio

juego como una herramienta de dirección activa que toma el control momentáneo para señalar la ruta correcta.

- **Flexibilidad cognitiva en la resolución de problemas:** Entendió que la accesibilidad no es simplemente física o visual, sino que también puede considerarse mental; y así incluyó mecanismos que otorgan más tiempo al cerebro del usuario para pensar posibilidades para soluciones de trampas o acertijos.

La ayuda para situarse en God of War Ragnarök se vale de la propia arquitectura matemática del nivel para funcionar sin que la gente se dé cuenta. A continuación, se ofrece un resumen detallado de su funcionamiento:

- **Cámara inteligente**: Desarrollaron una función de "Asistencia de Navegación" donde, al presionar un botón, el código calcula la ruta más corta sobre el mapa invisible (*NavMesh*) y rota la cámara de forma automática apuntando hacia el camino correcto.
- **Ampliación de tiempos en acertijos:** Permite que los mecanismos de los escenarios o de las trampas se muevan mucho más lentos, dejando bastante tiempo para pensar y cruzar sin sentir presión.
- **Recogida automática de recursos:** El personaje irá recogiendo automáticamente salud u objetos de mejora del suelo, siempre y cuando pase sobre ellos, dejando fuera la presión de botones extra en la exploración.

- **Resultados y aportes:** Definieron que aprovechar la geometría del nivel para ayudar a mover la cámara evita el abandono del juego.

Ilustración 5

Presentación visual de la jugabilidad en God of War Ragnarök

Nota: Obtenido de Santa Monica Studio[27]

2.1.5. Cuadro comparativo de trabajos relacionados

En esta sección, se examinan y comparan las similitudes, diferencias, puntos fuertes y áreas de oportunidad entre los videojuegos de: Shadow of the Tomb Raider, Gears 5, The Last of Us Part II y God of War Ragnarök, proporcionando una visión integral y crítica que enriquecerá la comprensión del tema abordado. Por este motivo, en la siguiente tabla se muestra resumidamente las comparaciones entre todos los trabajos que se mencionaron con anterioridad.

Tabla 1

Cuadro comparativo de aspectos únicos de los trabajos relacionado

Aspecto	Shadow of the Tomb Raider	Gears 5	The Last of Us Part II	God of War Ragnarök
Diseño de niveles	Entornos verticales y biomas selváticos complejos; criptas laberínticas que exigen una alta carga de observación y análisis espacial.	Escenarios mayormente lineales y semiabiertos, diseñados estructuralmente para canalizar la acción y el movimiento entre zonas seguras.	Combinación de pasillos narrativos y áreas amplias en ruinas urbanas; el diseño fomenta la búsqueda minuciosa en cada habitación.	Reinos mitológicos de gran magnitud, estructurados con rutas interconectadas y mallas de navegación complejas (*NavMesh*).
Navegación espacial	Exploración guiada mediante el uso de texturas dinámicas (manchas de pintura o relieves) que marcan las rutas de escalada.	Avance directo y guiado por la propia arquitectura del nivel, llevando al jugador de un punto de conflicto a otro de forma natural.	Navegación apoyada fuertemente por un radar de audio direccional (3D) que guía al jugador a través del entorno paso a paso.	Exploración asistida por una cámara inteligente que calcula la ruta más corta y gira automáticamente hacia el objetivo.
Interacción ambiental	Alta dependencia de las físicas del entorno para resolver rompecabezas espaciales, habilitando zonas de escalada, rápel y buceo.	Uso táctico de los elementos del escenario como coberturas; cuenta con colisiones tolerantes (*hitboxes* amplios) al interactuar.	Uso del entorno para el sigilo (hierba alta, agua, espacios estrechos) y recolección manual de recursos esparcidos en el mapa.	Interacción directa con la geometría del nivel para resolver obstáculos físicos (congelar agua, mover maquinarias pesadas).
Ambientación visual	Atmósfera inmersiva que simula ecosistemas hostiles y oscuros, propensos a generar sobrecarga de información visual en el usuario.	Entornos bélicos y caóticos de ciencia ficción, fuertemente marcados por la destrucción arquitectónica y el clima extremo.	Atmósfera tensa y melancólica; la iluminación, la vegetación invasiva y las sombras dictan el ritmo de la exploración.	Atmósfera épica y fantástica, con paletas de colores contrastantes que diferencian claramente cada reino mitológico.
Dinámicas del escenario	El entorno funciona como un rompecabezas en sí mismo; los peligros naturales obligan a planificar cada movimiento.	El escenario actúa como una zona de combate fluida; las barreras físicas dictan el ritmo de avance y retroceso del jugador.	El entorno es impredecible; las zonas de luz y sombra son mecánicas vitales para evitar la detección de los enemigos.	El entorno presenta obstáculos integrados en la arquitectura que requierenusar habilidades específicas para desbloquear el camino.

Accesibilidad en el entorno	Ajustes modulares que permiten al usuario elegir qué tan legibles son las pistas del entorno sin cambiar la dificultad del juego.	Eliminación de mecánicas de desgaste físico rápido al interactuar con puertas o bloqueos en el nivel (mantener botón en lugar de presionar rápido).	Filtros visuales de alto contraste que abstraen el nivel a tonos grises y resaltan únicamente los elementos interactivos con colores vivos.	Ajustes de tiempo integrados en el código del entorno para que las trampas o acertijos se muevan más lento, otorgando más tiempo de reacción.

Nota: Elaboración propia

La comparación presentada en la tabla busca destacar la variedad de posibilidades y estilos en la integración de niveles y el diseño ambiental, por ello no se centra en elegir opción alguna o específica para el desarrollo del trabajo, sino en ofrecer una visión amplia de las múltiples opciones disponibles que brindan una amplia gama de combinaciones posibles de preferencias o aspectos específicos que se ajusten de mejor manera a la planificación y propuesta del proyecto de Ashes of Faith

2.2. Bases teóricas

Este apartado detalla la evolución en la creación de mundos virtuales y los conceptos principales que guían el proyecto, con el objetivo central de lograr que el videojuego sea disfrutable y comprensible para cualquier persona.

2.2.1. Integración de Niveles

Tradicionalmente, los escenarios tridimensionales para computadora se construyen asumiendo un jugador sin limitaciones físicas o visuales, lo que genera barreras restrictivas y altos niveles de frustración[7], [8]. Por mucho tiempo, la inclusión de opciones de accesibilidad ha sido considerada como un arreglo de último momento, en lugar de ser el pilar principal del diseño de software[5], [15].

La integración del nivel es un paso fundamental en el desarrollo, ya que es el momento donde se unen todas las partes que se programaron por separado. Aquí es donde las mecánicas, la interfaz, las físicas y el entorno empiezan a funcionar en conjunto [14], [28]. Al trabajar en Unity [29], esta etapa consiste básicamente en conectar los scripts de C# con los elementos del escenario. El objetivo es que todos los módulos se comuniquen entre sí y que el código se ejecute correctamente al interactuar con el entorno 3D[30].

Además, armar todo el nivel requiere tener cuidado con el rendimiento y el uso de la memoria. Para no volver pesado al proyecto, se implementan métodos de carga asíncrona y se optimiza la forma en que se instancian ciertos objetos o destruyen los prefabs en la escena[31]. Todo este proceso de armado se organizó usando la metodología Scrum [32]. Trabajar de esta manera, mediante sprints permite probar el rendimiento poco a poco, encontrar y eliminar bugs sobre la marcha, y así asegurando que todo corra sin problema alguno antes de exportar la versión final [3].

2.2.2. Programación de entornos

Los desarrolladores modernos usan una técnica la cual es la de “Occlusion Culling” donde el mismo escenario ayuda al jugador a optimizar el rendimiento del juego dejando de renderizar y calcular los objetos 3D no visibles o que se encuentren fuera de la visión del jugador[14], [29]. Esto significa colocar luces, caminos y zonas de choque para guiar a la persona naturalmente, implementando sonidos en 3D que le indiquen la ruta correcta sin mirar fijamente la pantalla a todo momento[5], [33].

Un entorno bien programado es fundamental para que el jugador no se pierda o se frustre mientras explora. Según investigaciones orientadas a la usabilidad de videojuegos, las personas necesitan que las interacciones con el entorno virtual sean predecibles e intuitivas;

Esto significa, que si interactúan con objeto o elemento del entorno, el código responda de manera fluida a sus acciones [28]. Si la respuesta de los inputs tiene retraso, o las colisiones o triggers en Unity están mal escalados, el jugador siente que el juego está mal o incompleto. Por esta razón, el código debe asegurar que los límites físicos y las zonas de interacción sean precisas desde el primer momento [14].

Además, la ciencia detrás del diseño de juegos y los factores humanos demuestra que el jugador necesita retroalimentación constante del escenario para entender lo que está pasando. Esto significa que la programación debe devolverle una señal inmediata, como un sonido, un efecto visual o un cambio en la interfaz y no solo registrarse cuando el usuario choca o interactúa con algo[17]. Lograr esto a nivel de código hace que el escenario deje de ser un simple modelo 3D estático y se convierta en un espacio vivo, donde navegar y resolver los retos del nivel se sienta lógico y conectado con lo que el jugador quiere hacer[1].

2.2.3. Arquitectura de interfaz de Usuario

En este punto se explicará como la interfaz de usuario sirve para mostrar información clave y la conexión del usuario con el mundo virtual, y porque conviene programarla mediante módulos para cuidar el rendimiento y evitar la saturación a la pantalla.

Para que estos atajos funcionen bien y no perjudiquen el rendimiento, se necesita crear el programa por partes o módulos[13]. De esta manera se separan las reglas del juego de los controles, permitiendo así que el jugador cambie a gusto propio las opciones y el equipo siga rindiendo al máximo[11]. En el fondo, la interfaz sirve como un puente que convierte todo lo que ocurre en el mundo virtual en información visual muy clara para que se pueda entender al momento [28].

La forma en que la pantalla se comunica con el entorno 3D debe ser rápida para que el juego no se vuelva lento. Al usar herramientas como Unity [29], lo ideal es no forzar a la computadora a gastar recursos revisando a cada segundo si algo cambió en el mundo. En su lugar, se hace que el sistema actúe únicamente cuando pase algo concreto. Haciendo que cuando ocurra una acción como recoger un objeto o abrir una puerta, el mismo escenario mandé un aviso que actualice la pantalla de forma automática, manteniendo la parte visual totalmente separada del mundo físico [30].

Por último, ordenar todo de esta manera evita que la pantalla sea molesta o se sobrecargue de información innecesaria. Si la vista se llena de textos o indicadores innecesarios, la persona pierde rápidamente la atención y se desconecta de la aventura[6]. Para evitar esto, la interfaz debe adaptarse: tiene que mostrar datos clave solo cuando se necesiten y ocultarlos rápidamente, logrando que explorar el escenario siga siendo el centro de atención en todo momento[28].

2.3. Bases metodológicas

En este apartado se aborda el concepto de metodología de desarrollo, explicando cómo estas formas de trabajo funcionan como un mapa para guiar y organizar todo el proceso de creación. De igual manera, se revisan opciones ágiles más usadas actualmente para conservar el orden de las tareas, ayudar a la comunicación del equipo y evitar retrasos.

2.3.1. Metodología ágil (Scrum)

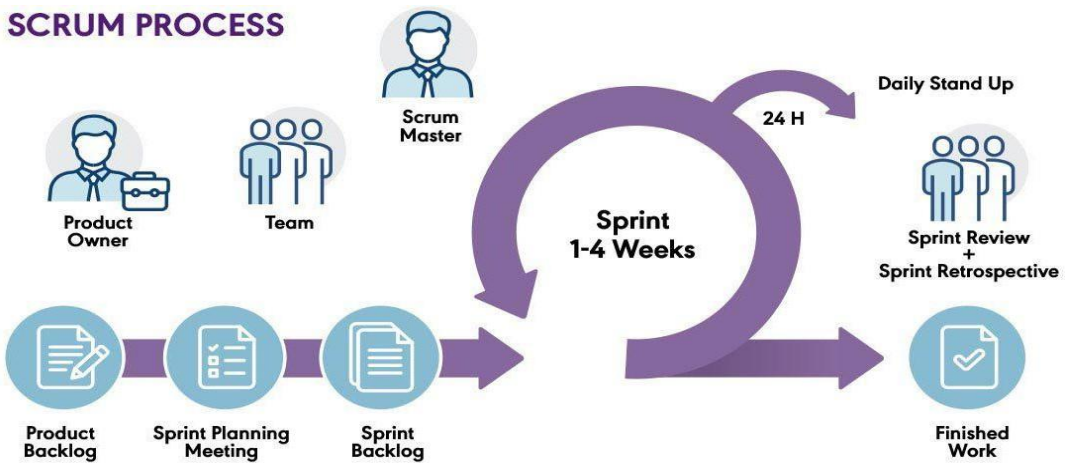
Esta forma de organizar el trabajo se basa en la idea de que es imposible predecir todo lo que ocurrirá en un proyecto largo. Por ello, el equipo divide una tarea enorme en piezas mucho más pequeñas y manejables, priorizando siempre lo que es más importante [34]. El trabajo se realiza en ciclos cortos de tiempo, conocidos como iteraciones (que generalmente

duran entre dos y cuatro semanas). Al final de cada periodo, el equipo no entrega excusas ni documentos incompletos, sino un prototipo del programa funcionando perfectamente [32].

Lo que hace a Scrum tan popular es su enfoque en la comunicación constante. Se realiza reuniones cortas, de quince minutos todos los días, donde el equipo simplemente dialoga sobre qué lograron ayer, qué harán hoy y si existe algún obstáculo que los esté frenando[32]. Además, incluye momentos específicos al final de cada ciclo para reflexionar sobre qué se hizo bien y qué se puede mejorar, logrando que el equipo sea más rápido y eficiente con el paso del tiempo.

Ilustración 6

Metodología ágil Scrum



Nota: Obtenido de pm-partners.com[35]

2.3.2. Metodología ágil (Kanban)

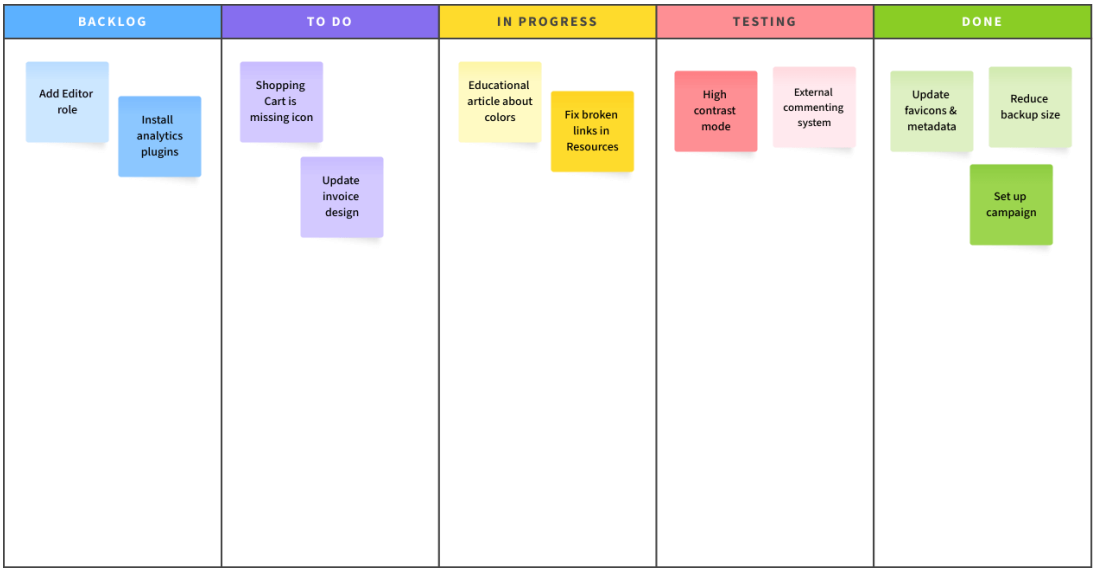
Esta metodología no se basa en fechas de entrega rígidas, sino que organiza las tareas de manera constante y muy visual. Para lograrlo, utiliza una pizarra dividida en columnas que muestran en qué etapa está cada actividad, por ejemplo: "Por hacer", "Haciéndose" y

"Terminado" [36]. Cada trabajo se anota en una pequeña tarjeta que avanza por estas columnas a medida que se va completando.

La gran ventaja de este sistema es que prohíbe hacer demasiadas cosas a la vez. Si el espacio de tareas en progreso está lleno, nadie puede empezar algo nuevo hasta terminar lo que ya tiene asignado[36]. Al mirar la pizarra, es muy fácil notar si el trabajo se estanca en algún punto, lo que permite que todos dejen lo suyo un momento para ayudar a solucionarlo. Evitando así que el equipo se frustre y el avance se mantenga a un ritmo tranquilo pero seguro.

Ilustración 7

Metodología ágil Kanban



Nota: Obtenido de Moqups.com[37]

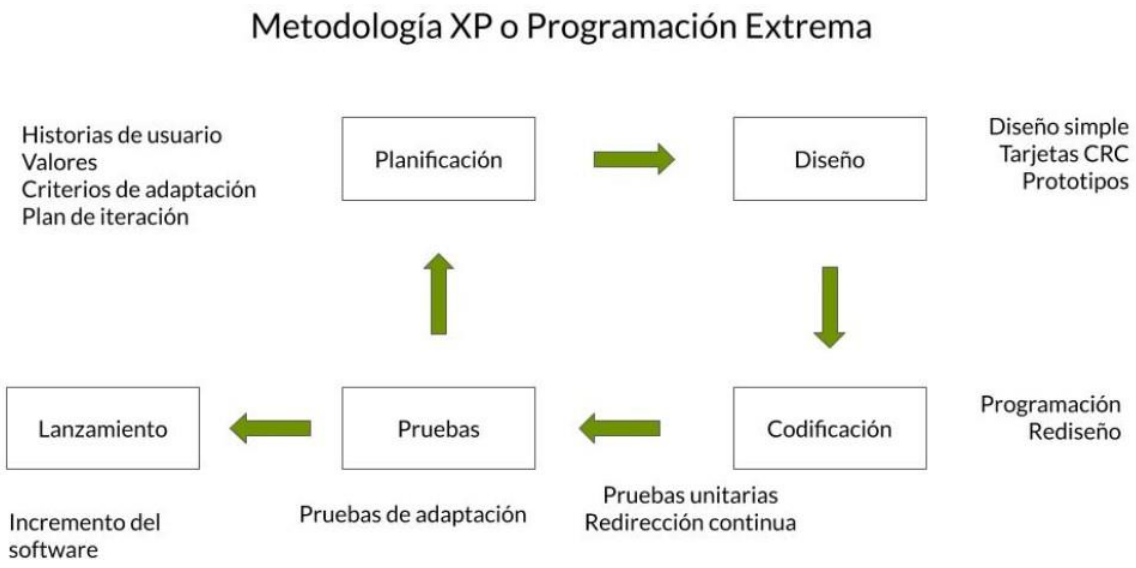
2.3.3. Metodología ágil Programación Extrema (XP)

La Programación Extrema (o XP) se concentra en asegurar que las bases del proyecto estén bien armadas desde el principio, aceptando que los planes pueden cambiar en cualquier momento [38]. Trabajar en parejas usando una sola computadora es una de sus prácticas más útiles: mientras uno escribe, el otro observa la pantalla para detectar errores al instante y sugerir mejores ideas. Esta revisión doble garantiza un trabajo mucho más pulido y ayuda a que ambos aprendan juntos.

Asimismo, XP recomienda hacer todo de la forma más sencilla posible, sin complicarse intentando resolver problemas que tal vez nunca sucedan[38]. Es obligatorio comprobar que cada cambio nuevo funcione bien antes de juntarla con el resto del proyecto. De este modo, si un cambio daña algo, el equipo se da cuenta de inmediato y lo arregla, en lugar de acumular fallos ocultos que después serían muy difíciles de corregir.

Ilustración 8

Metodología ágil Programación Extrema (XP)



Nota: Obtenido de Sinnaps.com[39]

2.3.4. Cuadro comparativo de Metodologías

Para comprender de manera mucho más clara cómo se diferencian estas tres formas de organizar el trabajo, el siguiente cuadro presenta una comparación directa de sus características principales. Realizar esta comparación nos ayuda a identificar qué enfoque se adapta mejor a las necesidades de comunicación, ritmo y exigencia técnica que requiere la construcción del entorno virtual.

Tabla 2

Tabla comparativa entre Metodologías: Scrum, Kanban, y Programación Extrema (XP)

Característica	**Scrum**	**Kanban**	**Programación Extrema (XP)**
Enfoque principal	Cumplir metas pequeñas en tiempos fijos, revisando el avance constantemente.	Mantener un ritmo continuo de trabajo apoyado en lo visual.	Asegurar que la estructura del código sea de la más alta calidad posible.
Organización del tiempo	Se divide rígidamente en ciclos de semanas, con fechas límite marcadas.	No existen ciclos obligatorios; el trabajo simplemente fluye día a día.	Se adapta a cambios muy rápidos; las pruebas y entregas son casi diarias.
Control de las tareas	Mediante reuniones diarias rápidas y asignando objetivos por ciclo.	Usando una pizarra con tarjetas y limitando las tareas en curso.	Mediante el trabajo en parejas frente a la pantalla y pruebas constantes.
Manejo de errores	Se detectan al final del ciclo durante las reuniones de revisión del equipo.	Se notan enseguida si las tarjetas se empiezan a estancar en la pizarra.	Se corrigen al instante gracias a la revisión de dos personas al mismo tiempo.
Ideal para...	Proyectos que necesitan estructura, orden y ver resultados palpables seguido.	Equipos que buscan evitar el estrés, trabajando por prioridades sin pausas.	Desarrollos técnicos complejos donde no se permiten fallos internos.

Nota: Elaboración propia

La comparación resultó en que la metodología de trabajo adecuada y factible para el proyecto es Scrum. La construcción del entorno virtual requiere un alto nivel de organización y un seguimiento continuo, necesidades que cubre Scrum al dividir el tiempo de trabajo en ciclos prefijados. La factibilidad para este caso concreto se fundamenta en que esta

metodología permite no alcanzar la desorganización al fijar objetivos pequeños a corto plazo. De este modo, también requiere revisiones frecuentes y asegura que el proceso se desarrolle de forma ordenada y entregue elementos funcionales, reales y tangibles de forma continua a lo largo de todo el proceso.

2.4. Bases tecnológicas

En este apartado se describen las tecnologías utilizadas para el desarrollo de los niveles y ambientes, con la finalidad de justificar la elección del motor gráfico y de las herramientas de programación que permiten construir un entorno tridimensional estable y funcional. También, se requiere enseñar cómo estas herramientas ayudan a la implementación de sistemas de usabilidad y rendimiento revisados anteriormente.

2.4.1. Unity (Motor Gráfico)

Es un motor gráfico desarrollado por Unity Technologies y se utiliza para crear videojuegos, aplicaciones y experiencias en entornos tanto en 2D como en 3D[29], lo que posibilita la generación de simulaciones para una amplia gama de dispositivos que van desde consolas y dispositivos móviles, hasta aplicaciones para computadoras y realidad virtual[29].

Unity se usa de manera profesional para crear videojuegos independientes; además, trabaja en conjunto con el lenguaje de programación C-Sharp (C#)[40], permitiendo una integración y desarrollo de secuencias de comandos estables y orientadas a objetos. Esta herramienta tiene la capacidad de importar recursos, así como modelos generados que son más conocidos como "assets", que constituyen simplemente la base fundamental de los recursos que cualquiera vaya a necesitar para iniciarse en la creación de entornos interactivos. El aprendizaje en Unity se desarrolla como una experiencia sucesiva y accesible para los

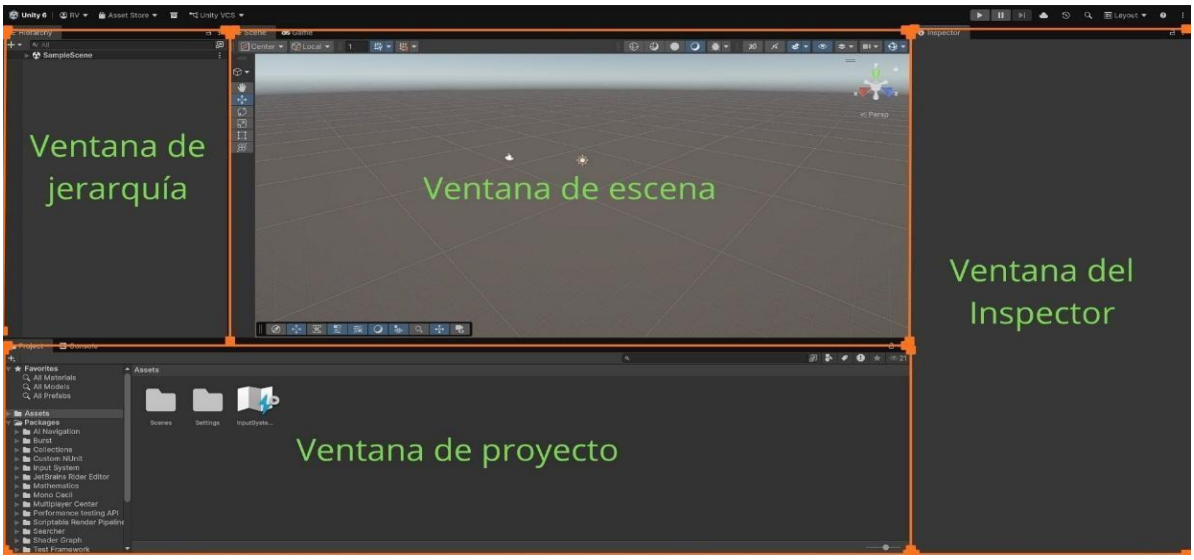
desarrolladores, teniendo como punto inicial una interfaz intuitiva del editor donde se aprecian las diversas herramientas básicas para manipular objetos, crear escenas y estructurar la usabilidad espacial del nivel.

- **Ventana del proyecto:** Aquí se presentan todos los assets y recursos disponibles para la construcción del proyecto.
- **Vista de la escena:** Permite la visualización y la edición de forma tridimensional de la escena que se está visualizando en ese instante.
- **Ventana de jerarquía:** Muestra la estructura jerárquica y la forma de organizar y agrupar los diferentes objetos 3D dentro de la escena que se acaba de abrir.
- **Ventana del inspector:** Muestra las diversas propiedades del objeto que hemos seleccionado, dándonos la opción de editar precisamente y visualizar esas características geométricas y físicas

A continuación, se presenta en la ilustración 9 la herramienta de Unity con sus respectivas interfaces.

Ilustración 9

Representación visual de interfaz visual del motor gráfico Unity



Nota: Elaboración propia

2.4.2. Unreal Engine (Motor Gráfico)

Es un conjunto de herramientas y software de desarrollo de videojuegos creado por Epic Games. Se destaca en la industria del desarrollo y el entretenimiento interactivo para producir contenido tanto en 2D como en 3D, abarcando videojuegos de alto presupuesto (AAA), simulaciones y medios audiovisuales[41]. Ofrece una amplia gama de funciones que van desde gráficos avanzados y físicas realistas, hasta animación e inteligencia artificial, por lo que se ha convertido en una opción popular entre desarrolladores y profesionales del sector [41].

El impacto de Unreal Engine en la industria va más allá de su éxito en juegos; ha transformado cómo se idean, desarrollan y presentan los entornos virtuales actualmente. Su estructura sólida y capacidad para crear ambientes visuales únicos han elevado el estándar de calidad, permitiendo a los desarrolladores generar propuestas que integran precisión y realismo con una gran flexibilidad multiplataforma. Este motor se conoce por tener una curva de aprendizaje más pronunciada en comparación con otros motores, principalmente debido a su enfoque en la precisión gráfica. Utiliza C++ como lenguaje principal para su programación y cuenta con un sistema de programación visual llamado Blueprints, que permite crear una lógica de juego sin una necesidad de escribir código tradicional.

Unreal Engine dispone de las siguientes ventanas principales que abarcan el desarrollo de videojuegos:

- **Barra de pestañas:** Proporciona acceso a herramientas y comandos para trabajar en niveles superiores del editor.
- **Barra de herramientas:** Ofrece accesos rápidos a diversas herramientas y operaciones de uso común.
- **Actores:** Agrega modos de edición especializados al flujo de trabajo del escenario.

- **ViewPoints (Puntos de vista):** Muestra visualmente el mundo virtual en pleno desarrollo.
- **Navegador de contenido:** Organiza y muestra todos los contenidos agregados o colocados en el proyecto.
- **Esquema mundial:** Presenta una estructura jerárquica de todos los objetos en la escena, permitiendo su selección y modificación directa.
- **Detalles:** Permite ver y editar materiales utilizados en los objetos del usuario, además de facilitar la configuración de sus propiedades físicas.

A continuación, se presenta en la siguiente ilustración la herramienta de Unreal Engine con sus interfaces correspondientes.

Ilustración 10

Representación visual de interfaz visual del motor gráfico de Unreal Engine

Nota: Elaboración propia

2.4.3. Lenguaje de programación C#

Es un lenguaje de programación creado por Microsoft que se ha convertido en una de las opciones más populares y accesibles para crear videojuegos, destacando especialmente porque es el idioma central que da vida a los entornos interactivos dentro de Unity[29], [40]. Su diseño logra un equilibrio entre potencia y facilidad de lectura, lo que permite a las personas escribir instrucciones claras y lógicas para que la computadora entienda exactamente qué acciones debe realizar en cada momento, acelerando el proceso de creación[40].

A diferencia de otros lenguajes, C# cuenta con sistemas automáticos que hacen el trabajo pesado de fondo, como un recolector interno que se encarga de limpiar la memoria de la máquina eliminando los datos innecesarios[40]. Esto representa una gran ventaja para el rendimiento del equipo, ya que evita que el sistema se vuelva lento por errores de memoria, permitiendo que la atención se centre por completo en el diseño de los escenarios y la experiencia visual [40].

Para trabajar de manera fluida con este lenguaje, se utilizan varios conceptos básicos que ayudan a mantener todo el sistema ordenado:

- **Scripts:** Son los archivos de texto principales donde se escriben todas las normas y comportamientos que darán vida a los personajes, las cámaras y los objetos en la pantalla.
- **Clases:** Funcionan como moldes o plantillas generales; definen cómo debe ser un elemento dentro del mundo virtual, qué características lo componen y qué acciones es capaz de realizar.

- **Variables:** Son pequeños cajones de almacenamiento en la memoria donde se guarda información temporal y cambiante, como la cantidad de salud, la velocidad de movimiento o el tiempo que falta para un evento.
- **Funciones:** Son bloques de código que agrupan una serie de pasos para realizar una acción muy concreta. De esta forma, en lugar de escribir los mismos pasos cien veces, solo se llama a la función para que el personaje salte o abra una puerta.
- **Condicionales:** Son las reglas de decisión que controlan el juego, las cuales permiten que el código pregunte si algo es verdadero o falso antes de actuar, como revisar si el usuario tiene una llave antes de permitirle pasar a otra zona.

2.4.4. Lenguaje de programación C++

Es un lenguaje de programación sumamente robusto y rápido, considerado desde hace años como el estándar principal en la industria de los videojuegos de gran tamaño, siendo el motor base detrás de herramientas gráficas tan potentes como Unreal Engine[41], [42]. Su mayor atractivo radica en que le da al creador un control casi absoluto sobre el equipo, aprovechando hasta el último recurso del procesador y la tarjeta gráfica sin desperdiciar capacidad [42].

Aunque tiene la fama de ser un lenguaje complicado y exige escribir el código con extremo cuidado y mucha atención al detalle, los resultados compensan totalmente el esfuerzo.

Es la herramienta ideal cuando se construyen mundos inmensos donde las físicas, el movimiento de muchos objetos y las luces dinámicas deben calcularse en milisegundos, garantizando que no haya ningún problema de rendimiento [42].

Al programar utilizando C++, se manejan elementos muy específicos para alcanzar ese nivel máximo de velocidad y eficiencia:

- **Gestión manual de memoria:** A diferencia de otros lenguajes que limpian solos, aquí es la persona quien decide exactamente en qué momento se crea un dato y cuándo se destruye, asegurando que no se desperdicie nada.
- **Punteros:** Son como atajos directos que apuntan a la ubicación exacta de un dato dentro del cerebro de la máquina, lo que permite leer o modificar grandes cantidades de información a una velocidad altísima sin tener que hacer copias pesadas.
- **Archivos de cabecera:** Son documentos separados que sirven como un índice rápido o un resumen visual, donde se le anticipa a la máquina todas las partes que componen a un objeto antes de tener que leer el código completo, manteniendo el proyecto muy ordenado.
- **Compilación:** Es el proceso final de traducción, donde todo el texto humano escrito se convierte de una sola vez al idioma más básico y puro de la máquina, creando un programa final que se ejecuta sin frenos.
- **Programación orientada a objetos:** Es una forma de organizar el entorno tratando a cada cosa visible en la pantalla (como un obstáculo o una trampa) como si fuera un objeto real con sus propias características, ayudando a construir mundos complejos pieza por pieza.

2.4.5. Git

Es un sistema de control de versiones sumamente confiable que funciona en la práctica como una gran máquina del tiempo para el código y los archivos del entorno [43].

Su propósito principal es guardar un historial fotográfico completo de absolutamente todos los cambios que se le hacen a un programa desde el primer día de su desarrollo, protegiendo el trabajo para que nunca se pierda si la computadora llega a fallar o si se borra algo accidentalmente [43].

En la forma en la que se desarrollan programas interactivos hoy en día, es una herramienta obligatoria tanto si se trabaja en solitario como si se hace en un equipo numeroso. Su mayor beneficio es la tranquilidad que ofrece, ya que permite a los creadores experimentar con ideas nuevas o cambiar partes importantes del código visual con la seguridad de que, si algo se rompe, siempre pueden presionar un botón y regresar a la versión de ayer que funcionaba a la perfección [43].

El trabajo ordenado en Git se fundamenta en las siguientes herramientas clave que garantizan la estabilidad del trabajo:

- **Repositorio:** Es la base de datos principal, una especie de bóveda segura donde se almacena todo el historial de cambios, las versiones antiguas y los archivos actuales de la computadora.
- **Commits:** Son puntos de guardados manuales acompañados de un mensaje explicativo, los cuales registran con exactitud matemática qué líneas de texto se agregaron, se cambiaron o se borraron en ese instante.
- **Ramas (Branches):** Son caminos alternativos de trabajo que se separan del tronco principal. Permiten crear una copia aislada del juego para intentar programar algo nuevo, sin riesgo de dañar la versión central que ya es estable.

- **Fusiones (Merges):** Es la acción final de unir el trabajo que se hizo en una rama experimental de vuelta con el código principal, integrando todas las mejoras de forma segura sin perder la información.
- **Resolución de conflictos:** Es un sistema de seguridad que se activa automáticamente cuando se intenta juntar dos cambios distintos que afectan a la misma línea de código, pausando el proceso para que la persona decida manualmente cuál versión es la correcta antes de guardar.

2.5. Arquitecturas, modelos y estándares

En este apartado se presenta las arquitecturas de software utilizadas en el desarrollo de videojuegos, excluyendo la arquitectura basada en componentes por considerarse un paradigma o estilo de programación más que una arquitectura de software[31].

Tabla 3

Tabla de arquitecturas modelos y estándares

Arquitectura	Descripción	Casos de uso
Modelo-Vista-Controlador (MVC)	Separa la lógica del juego, la presentación y la entrada del usuario[31]	Juegos con interfaces complejas, estrategia y simulación
Arquitectura en capas	Divide el sistema en capas como presentación, lógica, física, IA, audio y persistencia[44]	La mayoría de los videojuegos medianos y grandes
Arquitectura orientada a eventos	Los sistemas se comunican mediante eventos[31]	Videojuegos con múltiples sistemas interactuando

Nota: Elaboración propia

Arquitecturas más utilizadas actualmente

En el desarrollo de videojuegos es bastante común encontrarse la arquitectura en capas para la organización de los subsistemas; y la arquitectura orientada a eventos para, en cierta medida, desacoplar la lógica. En los juegos con conectividad se utiliza el modelo Cliente-Servidor, que va siempre acompañado, por lo general, de microservicios para la gestión eficiente del backend de los juegos multijugador.

Diferencia entre arquitectura, paradigma y patrón

La arquitectura de software se refiere a la estructura global del sistema, esto es, Clean Architecture o Cliente-Servidor[44]. El paradigma establece el enfoque de desarrollo, que puede ser la programación orientada a objetos o la centrada en componentes [30]. Por último, el patrón de diseño proporciona soluciones a problemas algorítmicos concretos y puede ser un Singleton u Observer [31].

Por ello, Component-Based Architecture, Entity Component System (ECS) y la programación orientada a componentes suelen clasificarse como paradigmas o estilos de organización del código, más que como arquitecturas de software de alto nivel[30]

2.6. Herramientas de desarrollo

La elección de las herramientas de software más apropiadas resulta esencial para la ejecución y optimización del proyecto. A continuación, se presenta un estudio de evaluación técnica comparativa de los principales motores gráficos y lenguajes de programación que detallan las características que justifican la elección más apropiada del entorno de trabajo para el desarrollo del videojuego, esto se basa en virtud a la investigación que se realiza en la sección 2.4 bases tecnológicas, teniendo los siguientes resultados.

Tabla 4

Comparación entre Motores Gráficos

Característica	Unity	Unreal Engine
Facilidad de aprendizaje	Interfaz muy amigable e intuitiva, ideal para empezar a crear mundos rápidamente.	Requiere más tiempo de estudio y dedicación para dominar todas sus ventanas y opciones.
Calidad visual principal	Muy flexible; es excelente para estilos visuales variados y juegos de todo tamaño.	Destaca fuertemente por lograr gráficos hiperrealistas y simulaciones muy detalladas.
Consumo de recursos	Es más ligero y permite que los escenarios funcionen bien en casi cualquier equipo.	Exige componentes físicos mucho más potentes para aprovechar su máximo potencial.

Idioma de funcionamiento	Trabaja de la mano con C#, lo que facilita escribir las acciones y reglas del mundo.	Utiliza C++, lo que otorga un control absoluto sobre el funcionamiento interno de la máquina.

Nota: Elaboración propia

Tabla 5

Comparación entre Lenguajes de programación

Característica	C#	C++
Facilidad de escritura	Es mucho más fácil de leer, lo que acelera el ritmo de trabajo y evita confusiones.	Exige mucha más atención a los detalles y reglas de escritura bastante estrictas.
Limpieza de memoria	Es automática; el propio sistema busca y borra la información que ya no se está usando.	Es completamente manual; la persona debe programar exactamente qué borrar y cuándo.
Velocidad de respuesta	Es muy rápido y sumamente estable para la gran mayoría de las interacciones virtuales.	Es extremadamente rápido, ideal para que miles de objetos reaccionen en milisegundos.
Nivel de control	Mantiene un equilibrio seguro, evitando que pequeños descuidos cierren el programa de golpe.	Brinda un control total de la computadora, pero un pequeño error puede detener todo el sistema.

Nota: Elaboración propia

En conclusión, aunque herramientas como Unreal Engine y C++ brindan una potencia gráfica y un control absoluto sobre la máquina, la combinación de Unity y C# ofrece el entorno más amigable y equilibrado.

2.7. Marco conceptual

Para facilitar la comprensión técnica del proyecto, este apartado establece los conceptos teóricos y metodológicos fundamentales de la investigación. A continuación, se definen los conceptos claves orientados a la usabilidad, diseño de niveles y la programación dentro del entorno del motor gráfico de Unity, detallando su aplicación directa en el desarrollo del proyecto

Tabla 6

Definición de conceptos clave

Concepto	Definición aplicada al proyecto
Diseño de niveles (Level Design)	Proceso de conceptualización, estructuración geométrica y ensamblaje del entorno 3D
Triggers	Zonas de colisión pasiva que no obstruyen el movimiento físico del jugador
Prefabs	Plantillas de objetos preconfiguradas que se utilizan para instanciar elementos dentro del entorno
Scripts	Archivos de texto redactados en el lenguaje de programación C#
NavMesh	Estructura de datos generada sobre la topología del terreno que delimita las áreas transitables
Hitbox	Volumen geométrico invisible que define los límites de interacción de un objeto
White Box (Caja Gris)	Versión funcional temprana del nivel construida con geometría básica carente de arte final (texturas)
Profiling	Proceso de monitoreo algorítmico en tiempo real (medición de FPS, uso de RAM y CPU)

Nota: Elaboración propia

2.8. Relación entre la teoría y la propuesta tecnológica

La construcción del entorno interactivo para el videojuego "Ashes of Faith" no se ajusta a un proceso empírico, sino que expresa las bases de que las investigaciones se traducen en soluciones técnicas directas por medio de las tres bases del proyecto como sigue:

- **Desde las bases teóricas:** Los principios de usabilidad espacial y de accesibilidad son planteados algorítmicamente para su uso en el motor gráfico. De este modo, el problema de reducir tanto la carga cognitiva como la carga mecánica es resuelto de forma directa mediante el uso de mallas de navegación (NavMesh), sistemas de sonido direccional, colisionadores tolerantes (triggers), los que automatizan la interacción que no requiere una precisión motora.
- **Desde las bases metodológicas:** El enfoque ágil e iterativo con el que se desarrolló el nivel implica, lógicamente, la arquitectura del software que se ha elegido. El uso

de prototipos de caja blanca (white box) implica que código sea muy escalable y modificable. Por eso, podemos afirmar que la forma de estructurar la lógica a través de prefabs instanciados ha permitido que el entorno fuera depurado y refinado continuamente en todas las fases de testeo o sprint, sin tener que reescribir la base del código.

- **Desde las bases tecnológicas:** Los niveles de rendimiento y optimización de las actuales herramientas de desarrollo (como el motor Unity y el lenguaje C#) están ejercidos en la medida requería por el sistema, de modo que una de las apuestas se implementaría mediante una arquitectura orientada a eventos (para un menor consumo del procesador central) acompañada por técnicas de renderizado como Occlusion Culling o la gestión asíncrona de las escenas (en la medida que esto liberaría la memoria RAM).

Capítulo III

3. Manual de usuario

Este capítulo documenta los parámetros necesarios para la correcta ejecución y navegación dentro del videojuego. Aquí se detallan las especificaciones técnicas de usabilidad, el esquema de los controles, la distribución de las interfaces de usuario (UI/HUD) y como recorrer por las diversas pantallas del videojuego.

3.1. Especificaciones de uso

Para la ejecución del videojuego y la su correcta ejecución en el entorno tridimensional, se establecen los siguientes requerimientos de hardware y periféricos:

- **Dispositivos de Entrada:** Teclado estándar QWERTY y Ratón (Mouse) de dos botones con rueda de desplazamiento (Scroll)
- **Entorno de Despliegue:** PC
- **Resolución recomendada:** 1920 x1080 píxeles (Relación de aspecto 16:9).

3.2. Configuración de Controles

La ejecución de acciones se realiza utilizando un teclado alfanumérico para el desplazamiento espacial del personaje jugable y la orientación del ángulo visual de la cámara, junto a los botones del ratón que sirven para las acciones de combate.

Tabla 7

Esquema de controles e interfaz de entrada

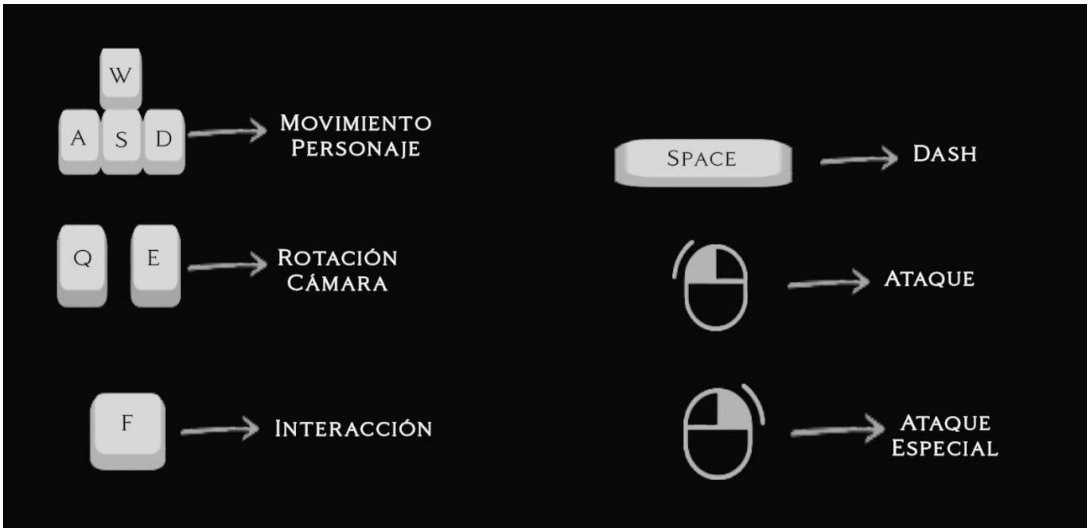
Acción Mecánica	Periférico / Tecla	Descripción Funcional dentro del Sistema
Movimiento de Personaje	W, A, S, D	Controla el movimiento del personaje jugable en el plano tridimensional (X, Z).
Rotación de Cámara	Q, E	Ajusta la perspectiva visual y la orientación angular de la cámara.

Interacción	F	Activa el diálogo con personajes no jugables (NPCs) e interacción con objetos del escenario.
Esquive / Impulso (Dash)	SPACE	Ejecuta un desplazamiento rápido para evadir ataques enemigos.
Ataque Básico	Clic Izquierdo	Activa la secuencia de ataque principal con el arma equipada.
Ataque Especial	Clic Derecho	Despliega la técnica especial sujeta al tiempo de recarga (Cooldown).

Nota: Elaboración propia

Ilustración 11

Esquema y distribución de los botones



Nota: Elaboración propia

3.3. Interfaz de Usuario (UI) y HUD

La interfaz gráfica presenta la información de las pantallas del juego desde la pantalla de inicio, los elementos permanentes en pantalla durante el gameplay (Heads-Up Display - HUD) y las pantallas de pausa para la gestión de información (menús).

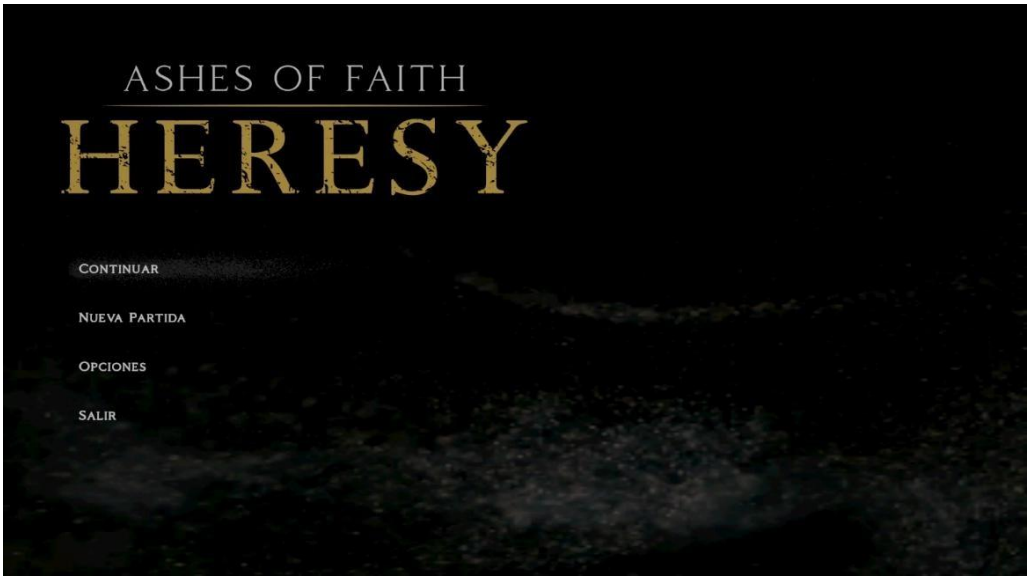
3.3.1. Menú de inicio del juego

- **Continuar:** Permite reanudar la sesión de la partida en el último punto de guardado donde se quedó el usuario la última vez que jugo.

- **Nueva Partida:** Empieza la historia del juego desde cero, comienza una nueva aventura.
- **Opciones:** Muestra un panel de ajustes técnicos configurables como sonido, video y controles.
- **Salir:** Cierra el videojuego de forma segura y vuelve al escritorio de Windows.

Ilustración 12

Interfaz gráfica del menú de inicio



Nota: Elaboración propia

3.3.2. Indicadores del HUD

- **Barra de Vida (Salud):** Se encuentra ubicada en la parte inferior izquierda de la pantalla. Muestra la salud del jugador dentro de un marco alargado oscuro y metálico.
- **Indicador de Poder (Cooldown):** Ubicado por encima de la barra de vida. Indica si el ataque especial está disponible para usarse y el tiempo de espera para volver a usarlo.

Ilustración 13

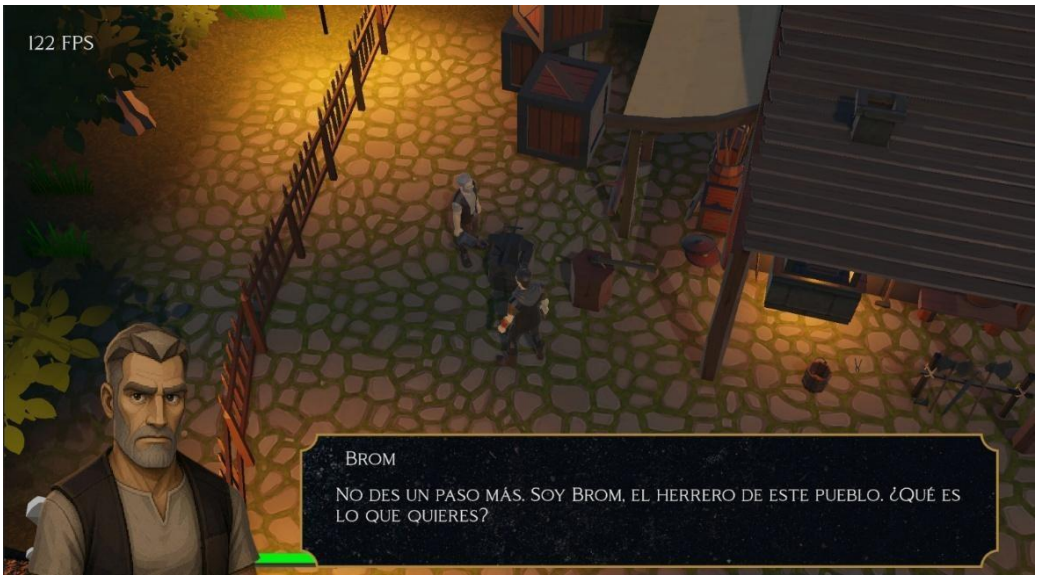
Vista de la perspectiva de cámara e indicadores del HUD (salud y cooldown)

Nota: Elaboración personal

- **Panel de Diálogos:** Franja horizontal oscura con borde dorado que se despliega en la parte inferior de la pantalla al presionar la tecla de interacción (F) frente a un personaje no jugable (NPC). Su propósito es mostrar los diálogos entre el jugador y los NPCs.

Ilustración 14

Ejecución del sistema de diálogos con el NPC Brom



Nota: Elaboración personal

3.3.3. Sistema de Menú de Pausa

- **Reanudar:** Permite volver a la pantalla de juego de forma instantánea, cerrando el menú de pausa y retomando el juego.
- **Salir:** Finaliza la partida de juego y regresa al menú principal.

Ilustración 15

Interfaz de pausa

Nota: Elaboración personal

3.4. Procedimiento de Instalación

- **Ejecutar el archivo de Instalación:**
  - Con el archivo ejecutable de instalación AshesOfFaith_Setup.exe, hacer doble clic sobre él para ejecutarlo (o seleccionar la opción Ejecutar como Administrador para otorgar los permisos del sistema).
- **Configuración de la Ruta de Instalación:**
  - En la pantalla de bienvenida del instalador, hacer clic en el botón Siguiente.
  - Definir la ruta donde se colocarán los archivos del juego (por defecto: C:\Program Files\Ashes of Faith: Heresy). Si se desea cambiar la ubicación de la instalación, seleccionar Examinar y elegir la nueva ubicación en el directorio.

- **Copias de Seguridad y Accesos Directos:**
    - Seleccionar la casilla Crear acceso directo en el Escritorio para crear un icono de acceso rápido en el escritorio y facilitar el ingreso posterior al videojuego.
    - Hacer clic en el botón Instalar para iniciar el proceso de instalación de datos, copia de librerías DLL y registro de recursos del motor.
- **Finalización de la Instalación y Puesta en Marcha:**
    - Una vez completado el proceso de instalación, la interfaz del asistente mostrará la pantalla de finalización.
    - Marcar la casilla Ejecutar Ashes of Faith: Heresy y hacer clic en Finalizar.
    - Entonces el aplicativo estará listo para su ejecución.

Capítulo IV

4. Manual de programador

En el presente capítulo se describe la construcción técnica usada para el entorno y diseño de niveles del videojuego “Ashes of Faith” en el motor de Unity. Este capítulo presenta la arquitectura general, las tecnologías empleadas, la organización de recursos y la estructura de clases y componentes desarrollados para el ambiente. También se aborda la programación de mecánicas ambientales, la implementación de la topología mediante White Box, el sistema de puntos de control (Checkpoints), y las características de usabilidad y accesibilidad del entorno.

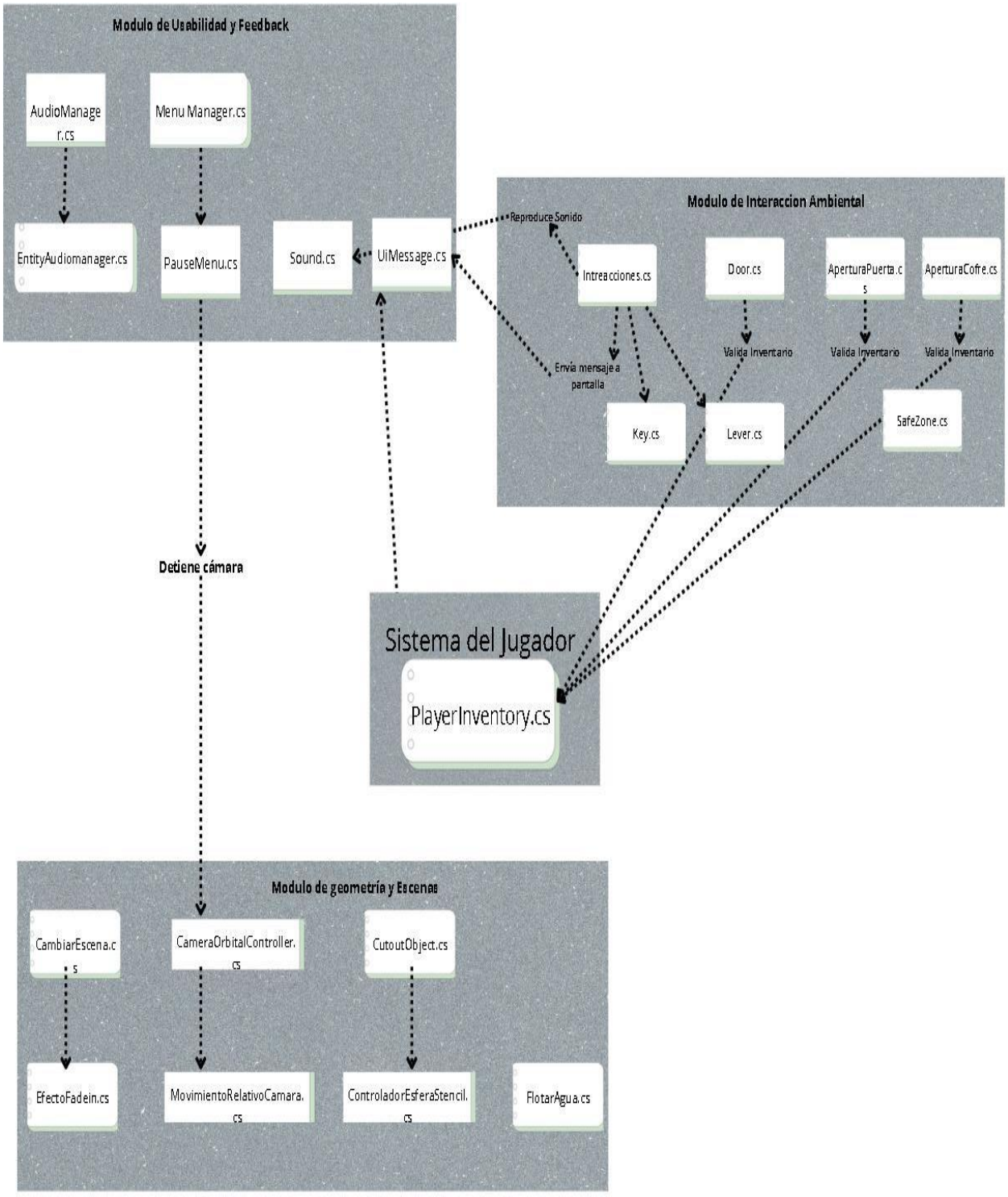
Asimismo, detalla la integración de estos sistemas con la inteligencia artificial de los enemigos y el controlador del jugador. Finalmente, se presenta la distribución del trabajo mediante metodología ágil y la estructura de clases generada para mantener la escalabilidad del código.

4.1. **Arquitectura general del sistema**

La arquitectura del sistema se basa en un modelo de diseño puramente modular con la intención primordial de garantizar la autonomía de los diversos bloques, lo cual incrementa la flexibilidad del código y evita que una sola unidad sea responsable de todo el proceso. La tecnología se apoya en la programación para definir lo que debe hacer y el funcionamiento interno de las aplicaciones de negocio. En la medida en que se desarrollan, los requisitos cambian y deben adaptarse.

Ilustración 16

Diagrama de flujo de la arquitectura general del sistema



Nota: Elaboración propia

En la **Ilustración 16** de la arquitectura superior se observa la combinación integrada del sistema, lo que incluye los lenguajes utilizados, el motor gráfico y los scripts. Se observa evidencias del uso del motor gráfico Unity como la base para el proceso de renderizado y simulación de físicas. El código es fundamentalmente escrito en el idioma C#, aplicando bibliotecas nativas para calcular los datos sobre una ruta y superponer información en pantalla. La comunicación empieza con una interacción física en el entorno, validando las conexiones lógicas a través del inventario de la persona, acabando en una fluidez de respuestas coherentes en los tres componentes principales.

- **Módulo de Interacción Ambiental**: Este componente maneja todas las respuestas mecánicas e interactivas durante la acción del personaje dentro del escenario. Las señales de física y las colisiones se envían al script principal Intreacciones.cs, quien analiza las colisiones entre los contenedores de colisión (triggers). La información resultante impulsa al generador de scripts que controlan los comportamientos, tales como Lever.cs (que manipula la apertura de interruptores mecanismos) y Key.cs (que permite la recopilación de artefactos claves). Otros scripts también afectan la topología del escenario, como es el caso de Door.cs, AperturaPuerta.cs, AperturaCofre.cs y SafeZone.cs, que interactúan entre sí para actuar de forma conjunta para darle sentido al juego. Por último, SafeZone.cs, una unidad de integración en dicha función configura límites de radio físico seguros dentro del cual el sistema de daños funcionará durante el tiempo que esté dentro de ellos.
- **Módulo de Geometría y Escenas**: El módulo de geometría y escenas controla el flujo espacial y la renderización con base en la transición de la malla. La secuencia de niveles se realiza en las clases CambiarEscena.cs y EfectoFadeIn.cs. A nivel de

espacio, las manipulaciones de los vectores de rotación y de los vectores de seguimiento tienen lugar en la cámara CameraOrbitalController.cs y su correspondiente script MovimientoRelativoCamara.cs. En términos de espacio, en tal contexto (debido al problema de la oclusión en espacios compactos), se decide crear y encapsular la geometría del mundo en los scripts de las clases CutoutObject.cs y ControladorEsferaStencil.cs, el segundo de los cuales se encarga de controlar si la geometría puede incidir en la visibilidad de otras entidades. También cambia dinámicamente su material para hacerlo transpuesta, ya que interfiera con el objeto principal. Además, empleamos un objeto físico adicional; la clase FlotarEnAgua.cs asocia fuerzas vectoriales a los objetos cuando estos interactúan con volúmenes líquidos.

- **Módulo de Usabilidad y Feedback:** Este módulo es el de la retroalimentación audiovisual, un elemento clave para que la carga cognitiva del usuario se reduzca. La clase UiMessage.cs intercepta las áreas de interés y llama a la aparición de los textos guías en pantalla. Al mismo tiempo, el script Sound.cs, que funciona según los gestores de mezcla AudioManager.cs y EntityAudioManager.cs, espacializa los efectos acústicos y los produce. Todas estas interacciones dependen del nivel de controladores existente en MenuManager.cs y PauseMenu.cs, los cuales determinan que, cuando se navega entre menús, la rapidez de cambio de la escala de tiempo a cero se produzca. Esto significa que debes tener la simulación física fijada de forma habitual al navegar entre menús.

4.1.1. Tecnologías, motores y herramientas utilizadas

En este capítulo describe los grupos de plataformas, bibliotecas y aplicaciones utilizadas para desarrollar el trabajo. La combinación de estos elementos facilitó el diseño

lógico del escenario, la realización de las colisiones y la definición del comportamiento visual.

- **Unity 6.3 LTS (VÉRTICE 6000.3.8f1):** Se eligió este motor gráfico debido a su adaptabilidad con entornos tridimensionales, la capacidad de diseñar bajo componentes y la fuente potencial del motor de iluminación. Se eligió la versión de larga vida útil (LTS) de este motor gráfico para mantener la confiabilidad técnica durante la fase final de la producción.

A continuación, se observa los requisitos para el correcto funcionamiento del motor gráfico de Unity.

Sistema Operativo

- **Windows:** Windows 10 o posterior de 64 bits.
- **Linux:** Ubuntu 18.04 y 20.04.

Hardware

- **Procesador (CPU):** Arquitectura x64 con soporte para el conjunto de instrucciones SSE2.
- **Memoria RAM:** Mínimo de 8GB (se recomienda 16GB para proyectos grandes)
- **Tarjeta Gráfica (GPU):** Compatible con DirectX 10, DirectX 11, DirectX 12, Metal o Vulkan, dependiendo de cada equipo
- **Visual Studio:** Es un sistema de desarrollo integral utilizado para el código fuente en C#. Está ligado directamente con Unity, lo que permite ordenar las bibliotecas, depurar los scripts desde ese lugar y administrar eficazmente las reglas del rompecabezas en espacio de manera efectiva.

- **C#:** Es el lenguaje de programación que permite escribir toda la lógica bajo el mapa para transiciones, activador de puertas, cambio de materiales, y comportamiento de usuarios. Ha proporcionado el control más efectivo de las mallas, las colisiones e interfases.
- **Unity UI y TextMeshPro**: Paquetes nativos diseñados para la renderización de texto e interfaces gráficas. Fueron utilizados fundamentalmente para organizar visualmente los mensajes en el script UiMessage.cs y para el manejo de los menús de navegación.

4.2. **Arquitectura de clases y componentes**

La metodología seguida fue Ágil Scrum, empleándose para desarrollarla durante cuatro meses. Estas actividades se dividieron en cinco etapas, con el propósito de crear los escenarios, cámaras, menús y las sensaciones percibidas. Los requerimientos fueron establecidos en un conjunto de Product Backlog, organizado de acuerdo con el valor y relevancia personal de cada parte.

Tabla 8

Información general de la bitácora

Campo	Información
Nombre	Ricardo Esteban Vazquez Banegas
Titulo	Bitácora del proyecto (Level Design)
Descripción	Diseño de los escenarios volumétricos, programación de interacciones, gestión de cámaras y configuración de los flujos de transición
Estimación	Cuatro meses

Nota: Elaboración propia

Se agrupa las historias de usuario para que el nivel de viabilidad pudiera organizarlas de manera adecuada. Se clasifican como altas las historias relacionadas con cámaras y

acciones básicas ya que representan la base del diseño espacial; las historias sobre diagnóstico y la interfaz tuvieron diferentes urgencias debido al tipo de funcionalidades operativas requeridas.

Tabla 9

Product Backlog de los sistemas de entorno

ID	Descripción resumida	Prioridad	Historia	Estado
US-01	Se requiere una cámara de seguimiento que rote para visualizar adecuadamente la topología del entorno	Alta	Cámara orbital	Completo
US-02	Se requiere un sistema que vuelva translucidos los obstáculos para no perder de vista al personaje	Alta	Visibilidad	Completo
US-03	Se requiere habilitar las interacciones físicas con palancas y contenedores con recursos	Alta	Interacciones	Completo
US-04	Se requiere que los accesos principales permanezcan cerrados hasta validar un ítem clave	Alta	Progresión de nivel	Completo
US-05	Se requiere emitir efectos de sonido y textos contextuales al aproximarse a un objeto interactúale	Media	Feedback ambiental	Completo
US-06	Se requiere cargar nuevos mapas sin interrupciones severas, utilizando fundidos visuales	Alta	Flujo de escenas	Completo
US-07	Se requiere un sistema de pausa para congelar la simulación y mostrar opciones de interfaz	Alta	Menús e interfaz	Completo
US-08	Se requiere un monitor de rendimiento en pantalla para detectar zonas con alta densidad geométrica	Media	Diagnostico (FPS)	Completo

Nota: Elaboración propia

En los primeros Sprints se distribuyen las asignaciones siguiendo el ciclo lógico de creación. Las figuras geométricas y las cámaras se encargaron de establecer las condiciones iniciales para desarrollar el prototipo.

Tabla 10

Planificación de los sprints

Componente	Sprint	Historias
Sistemas de Cámara	Sprint 1	US-01: Cámara orbital; US-02: Visibilidad
Interacciones Básicas	Sprint 2	US-03: Interacciones.
Progresión de Escenario	Sprint 3	US-04: Progresión de nivel.
Audio y Usabilidad	Sprint 4	US-05: Feedback Ambiental;

		US-08: Diagnóstico (FPS).
Flujo y Menús	Sprint 5	US-06: Flujo de escenas; US-07: Menús e Interfaz.

Nota: Elaboración propia

4.3. Desarrollo

Con vistas a resolver las problemáticas de las barreras de interacción y la desorientación espacial en entornos virtuales, se construye y desarrolla un entorno tridimensional interactivo en forma de prototipo de tipo caja gris (blockout o White box). A nivel ingenieril, la integración de esos sistemas en ese prototipo resuelve uno de los retos de mitigar, por medio de un espacio definido por la funcionalidad y la accesibilidad antes de la integración de recursos artísticos y de alto consumo, la alta carga cognitiva y las severas barreras motoras que colocan los niveles convencionales.

Más específicamente, la construcción del sistema se basa en cuatro módulos técnicos clave:

1. **Programación del entorno y la navegación:** Se construye la escena con un sistema de navegación por medio de mallas de navegación inteligente (NavMesh) para permitir que se puedan seguir rutas guiadas, además de utilizar sistemas de carga asíncrona (Scene Management) que permitan hacer transiciones entre escenas suficientemente suaves.
2. **Físicas y colisiones tolerantes:** Regiones de colisión que fueron determinadas para dar respuesta a los errores mecánicos provocados por el usuario gracias a la implementación de un sistema que se toma su tiempo para despreciar la resistencia mecánica al interactuar y/o explotar.

3. **Interfaz de usuario (UI/UX) y sonido espacial:** Se programan menús y HUDs de alto contraste y escalables, junto con el uso de un algoritmo de sonido direccional (HRTF) para corroborar acciones y reducir la saturación visual.
4. **Optimización y rendimiento técnico**: Se implementan diferentes rutinas y técnicas algorítmicas de perfilado de código (Profiling) y ocultamiento visual (Occlusion Culling) que llevan a cabo la fluidez y la ausencia de fatiga visual.

La organización del desarrollo original se guía por una metodología Scrum, pero en esta aproximación se realiza una adaptación estructural de este marco de trabajo ágil para ajustarlo al equipo de desarrollo unipersonal. Implementación, programación, etc. recaen en un único desarrollador, y el rol de revisar y priorizar requisitos (conocido como Product Owner) se lleva a cabo externamente mediante reuniones periódicas semanales, de seguimiento del desarrollo, con la dirección del proyecto.

El flujo de trabajo y el backlog de las tareas se dividen en las siguientes fases e interacciones (Sprints):

- **Fase de análisis y requisitos (Sprint 0):** Son aquellos en el que se definen los pilares técnicos del proyecto. Con una evaluación heurística en lo concerniente a la parte procedimental se describe los problemas de carga cognitiva y carga visual con vocabularios de diseño y normas de accesibilidad estandarizables.
- **Fase de diseño y arquitectura modular (Sprints 1 y 2):** La arquitectura lógica ahondada para el motor gráfico a los efectos de ir diferenciando los sistemas base de las ayudas de entrada; al mismo tiempo se desarrolla la caja gris (blockout) en el entorno 3D con especificaciones espaciales y un margen de ajuste físico.

- **Fase de desarrollo e implementación (Sprints 3 al 4):** Determinación sobre la elaboración de las soluciones técnicas del ambiente; se considera los shaders particulares, y los filtros de contrastes adaptativos, se incorpora las funcionalidades del audio espacial para la localización de los objetivos y el remapeo de los controles para la disminución de la fatiga motora.
- **Fase de pruebas, validación y refinamiento (Sprints 4 y 5):** Ejecución de iteraciones de pulido sobre el código fuente; pruebas cualitativas de jugabilidad sobre un grupo de usuario piloto y aplicación de métricas de usabilidad para solucionar problemas de interacción.

Por último, la configuración del build y el despliegue para el entregable final se compila para realizar su ejecución nativa en los entornos locales del ordenador. Para el rendimiento obtenido, se implementa la técnica de Occlusion Culling, la cual interrumpe el cálculo de los objetos tridimensionales que se encuentran fuera del campo de visión. Junto con la técnica de profiling constante, evita fugas de memoria y garantiza una utilización de recursos muy eficiente con una tasa de FPS muy estable.

4.3.1. Sprint 1: Sistema de Cámara

El objetivo de este primer sprint es establecer las bases de la perspectiva del jugador y asegurar una correcta navegación visual para el ambiente 3D.

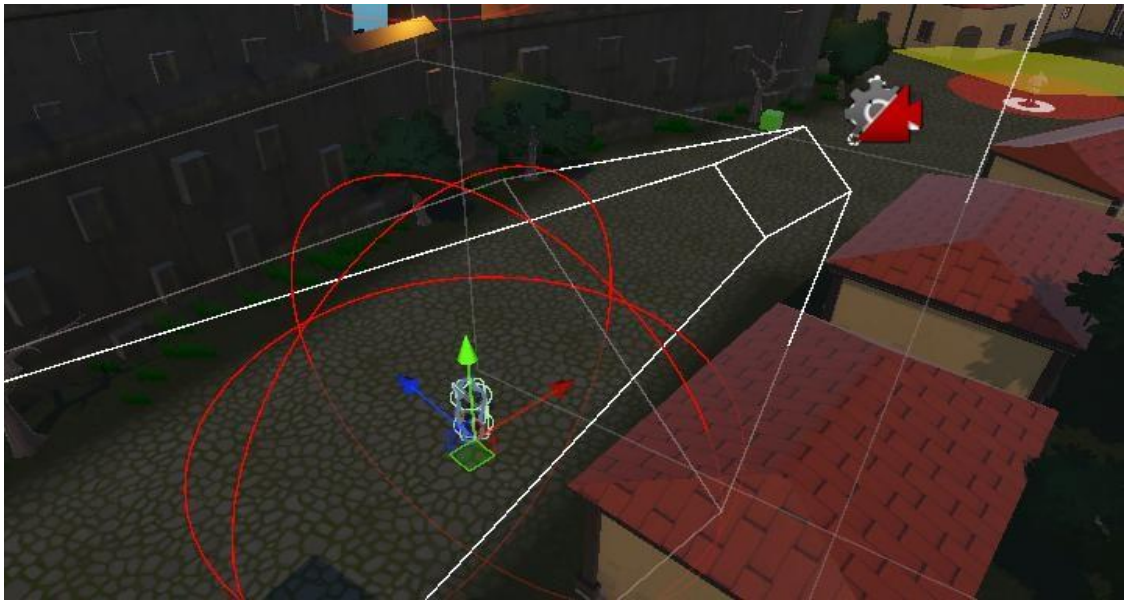
US-01: Cámara Orbital

- **Descripción:** Desarrollo e implementación mediante scripts en el lenguaje de programación C# de un sistema de una cámara orbital que siga al jugador en cada momento

- **Tareas clave:** Realizar la programación del control de rotación (ejes Y y X), configurar el suavizado de movimiento y delimitar los ángulos de inclinación de la cámara

Ilustración 17

Integración de cámara orbital



Nota: Elaboración propia

Como se visualiza en la **Ilustración 17**, para la cámara se empleó un cinemachine, el cual órbita alrededor del personaje principal. La cámara está configurada mediante una virtual cámara seleccionada la opción "Orbital Transposer", también permite girar la cámara de izquierda a derecha al presionar las teclas "E" y "Q".

US-02: Visibilidad

- **Descripción:** Garantizar la visibilidad constante del personaje principal logrando que nunca se pierda de vista atrás de los elementos del nivel.
- **Tareas clave:** Implementar un raycasting que vaya desde la cámara hasta el jugador. Crear la programación de un sistema que aplica una transparencia a los

objetos intermedios o un acercamiento dinámico de la cámara hacia el jugador para mantener una línea de visión.

Ilustración 18

Acercamiento de cámara hacia al personaje cuando existe un obstáculo u objeto que intervenga entre el personaje y la cámara

Nota: Elaboración propia

Posee un script, el cual permite que la cámara se acerque hacia un pivote colocado en el jugador para mantenerlo visible en todo momento

Pruebas y validación del sprint

- Verificación de las correctas colisiones de la cámara con los elementos del nivel
- Realizar pruebas de usabilidad asegurando que el movimiento de la cámara no genere ningún tipo de mareo o malestar visual y que el personaje sea visible en cualquier momento del juego

4.3.2. Sprint 2: Interacciones Básicas

El objetivo de este segundo sprint es el de dar vida al mundo del juego creando la programación de mecánicas del entorno que permite al jugador una interacción intuitiva con el entorno

US-03: Interacciones

- **Descripción:** Crear el sistema central, el cual se encarga de gestionar como el personaje principal se relaciona con los diferentes objetos, NPCs o los elementos claves del ambiente
- **Tareas clave:** Realizar las interfaces de interacción en C#, configurar correctamente los Colliders o triggers, y programar los eventos de respuesta de objetos, la apertura de puertas, e inicio de diálogos con los NPCs.

Ilustración 19

Demostración de interacciones del personaje principal con los objetos del entorno

Nota: Elaboración propia

Como se observa en la ilustración 19, 1. Interacción con cofres, 2. Interacción con puerta del hogar, 3. Interacción con puerta final del nivel, 4. Interacción con puerta de

cementerio. Todas estas interacciones se desarrollan usando dos scripts respectivamente (AperturaPuerta.cs, y AperturaCofre.cs), los cuales funcionan cuando el jugador colisiona con los triggers de los objetos.

Pruebas y validación del sprint

- Comprobación de rangos de interacción y hitboxes
- Validación de una respuesta correcta de mecánicas a los inputs del controlador y teclado

4.3.3. Sprint 3: Transición entre escenarios

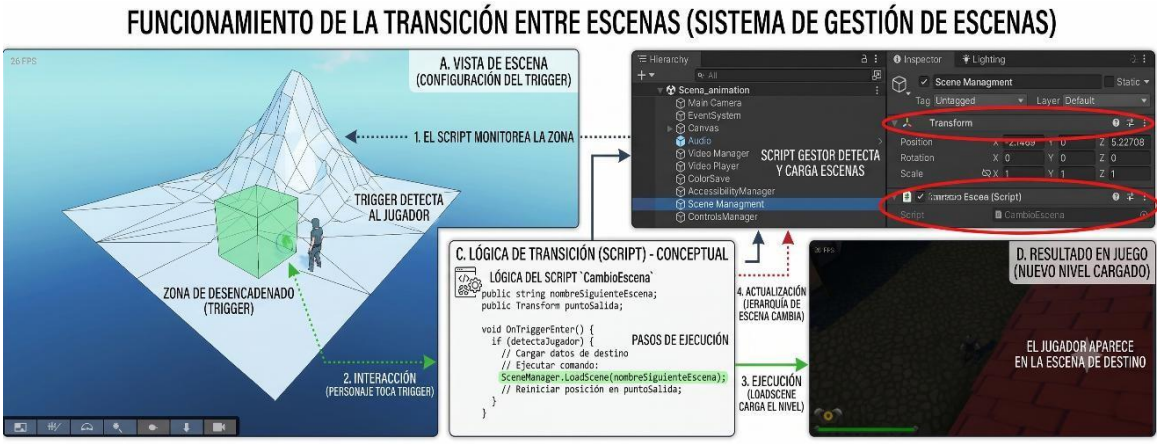
El objetivo de este tercer sprint es realizar la lógica de estructura ambiental y el flujo de niveles para guiar la experiencia del jugador en el nivel

US-04: Progresión de nivel

- **Descripción:** Programar sistemas que evalúan las diferentes condiciones de avance o victoria dentro una zona específica del juego
- **Tareas clave:** Implementar controladores que rastrean los objetivos completados (como la recolección de llaves) y gestionar el estado de los objetos ambientales (apertura de rutas principales bloqueadas) al terminar eventos.

Ilustración 20

Diagrama del funcionamiento de la transición entre escenas



Nota: Elaboración propia

Con el fin de estructurar la progresión de la partida, se implementa un sistema de validación lógica que tiene la tarea de verificar si los objetivos de la zona se han completado. Esto se lleva a cabo mediante un script que controla los triggers de salida y consulta el inventario del personaje principal. Al cumplir los requisitos de para continuar avanzando en el nivel, el entorno actualiza su estado habilitando el acceso a las zonas finales del mapa o abriendo las compuertas que bloquean el camino, sin ejecutar aun la carga del siguiente nivel.

Pruebas y validación del sprint

- Playtesting completo del recorrido de todo el nivel para asegurar que las zonas bloqueadas y accesibles del entorno funcionen correctamente sin interrumpir el flujo de la partida.

4.3.4. Sprint 4: Audio y Usabilidad

El objetivo de este cuarto sprint es reforzar la inmersión del usuario y proporcionar herramientas que permiten asegurar el rendimiento óptimo del juego.

US-05: Feedback Ambiental

- **Descripción:** Integración de respuestas auditivas y efectos visuales que reaccionan a diferentes acciones o interacciones del jugador con el entorno.
- **Tareas clave:** Crear la lógica de programación de la emisión de audio espacial, efectos de partículas en colisiones, y asegurar que el Feedback acústico ayude a la usabilidad y accesibilidad.

Las respuestas auditivas fueron implementadas con los gestores AudioManager y EntityAudioManager, que se encargan de reproducir los efectos sonoros del entorno. Junto a las respuestas auditivas se integró la emisión de partículas visuales y textos contextuales, para lo cual se desarrolló el script UiMessage, el cual es capaz de responder automáticamente de forma eficaz a las acciones y colisiones del jugador. La finalidad era confirmar las interacciones del jugador y al mismo tiempo reducir su carga cognitiva.

US-08: Diagnóstico (FPS)

- **Descripción:** Crear una herramienta de debug e interfaz técnica para medir el rendimiento del juego en tiempo real.

- **Tareas clave:** Desarrollar el script que calcule y muestre los frames per second (FPS) en tiempo real, permitiendo optimizar escenas con alta carga de objetos dentro del entorno

Ilustración 21

Fotogramas por segundo (FPS) y stats en tiempo real

Nota: *Elaboración propia*

Se creó un script de diagnóstico que calcula los FPS y las estadísticas de carga del sistema en tiempo real y permite visualizarlo en pantalla. Esta herramienta de debug es un monitor de rendimiento que muestra de forma precisa las zonas del mapa que son de alta densidad geométrica para ser posteriormente optimizadas.

Pruebas y validación del sprint

- Revisión de la correcta mezcla de audio para así evitar una saturación de canales
- Pruebas de estrés o frustración y monitoreo constante de FPS en zonas sobrecargadas del mapa

4.3.5. Sprint 5: Flujo y menús

El objetivo de este quinto y último sprint es conectar todos los sistemas existentes en un solo ciclo de juego completo y a la vez accesible mediante una interfaz de usuario detallada

US-06: Flujo de escenas

- **Descripción:** Gestión de transiciones entre todos los estados de juego (Menú Principal, Carga de nivel, Gameplay, Pantalla de pausa, y Game Over)
- **Tareas clave:** Implementar un administrador de escenas para cargas asíncronas, asegurando transiciones mucho más suaves y fluidas entre niveles.

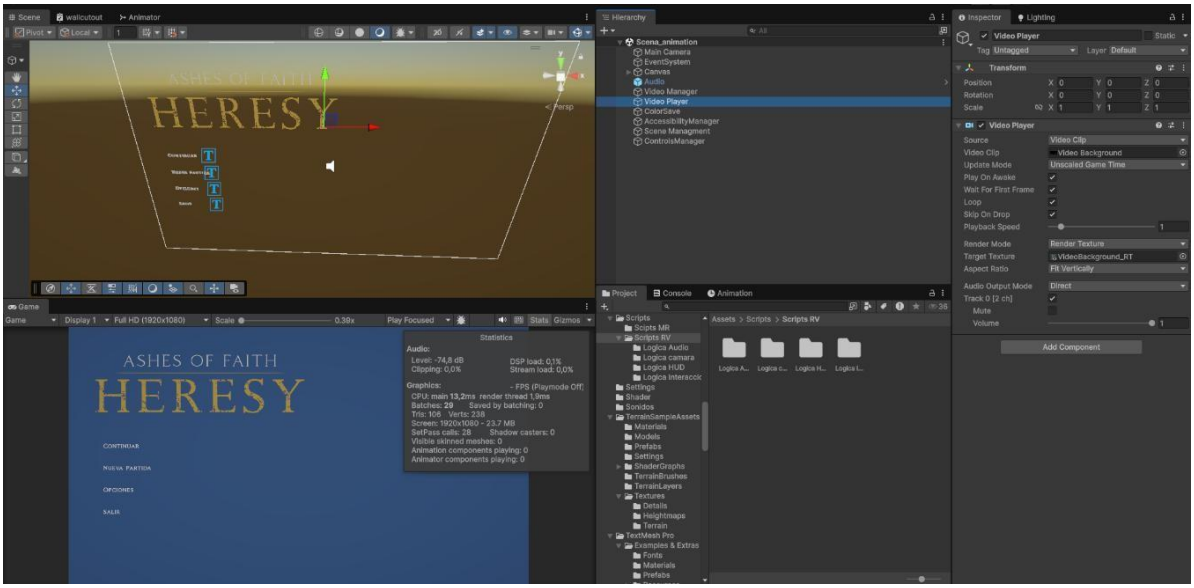
El flujo global fue controlado mediante una programación de un administrador de escenas cuya finalidad estaba centrada en realizar cargas asíncronas entre todos los diferentes estados del juego, que, aunque va desde el menú principal y la pantalla de pausa al gameplay y la pantalla de Game Over, sí que está implementado de una forma más general. Utilizado junto con scripts como CambiarEscena y EfectoFadeIn, el sistema permite realizar una transición visual en el uso de fundidos entre pantallas para evitar esos cortes bruscos o pantallas de carga muy largas que puedan romper la experiencia de usuario en el juego

US-07: Menú e Interfaz

- **Descripción:** Desarrollo de la interfaz de usuario principal del videojuego
- **Tareas clave:** Crear la programación de los elementos visuales del Canvas (botones, panel de opciones, HUD). Asegurar una navegación correcta tanto con el ratón como con un gamepad, aplicando los principios de usabilidad y accesibilidad.

Ilustración 22

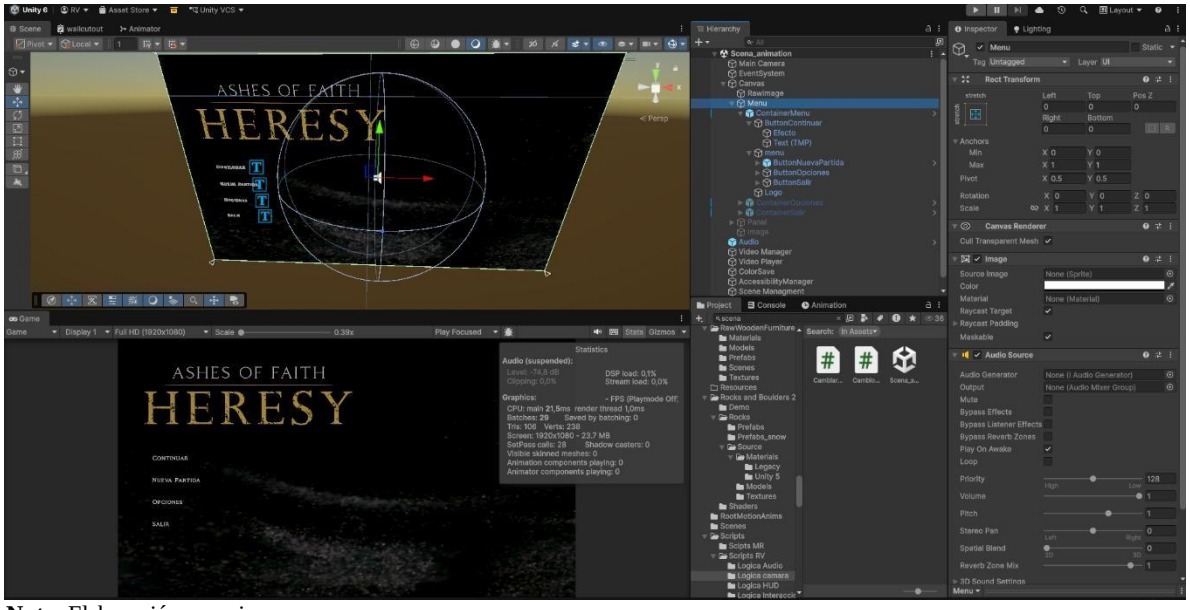
Implementación de un VideoPlayer al menú principal



Nota: Elaboración propia

Ilustración 23

Contenido de opciones del menú principal



Nota: Elaboración propia

Para el desarrollo de la interfaz principal, se creó un Canvas interactivo dinámico mediante el uso de los paquetes nativos Unity UI y TextMeshPro para procesar el texto y los elementos gráficos. Por su parte la regla de navegación queda definida gracias a los scripts MenuManager y PauseMenu, empleando los paneles de menús, así como el visor frontal para hacer match con un diseño a alto contraste y la activación de una respuesta correcta independientemente del uso del ratón o del gamepad.

Pruebas y validación del sprint

- Pruebas de inicio a fin del bucle principal del videojuego
- Validación de las respuestas de la interfaz de usuario en diferentes resoluciones de pantalla.

La culminación satisfactoria de estos cinco sprints brinda una muy buena base técnica para Ashes of Faith, para el desarrollo del juego y, en la medida en que se otorga este orden a la parte de desarrollo, se trata en primer lugar los sistemas visuales básicos y luego se continúa con el afianzamiento de la interfaz y el flujo de juego al mismo tiempo que se consolidan la estabilidad de las mecánicas centrales y la facilidad para el jugador. En el mismo sentido, el hecho de planificar de esta manera la parte del desarrollo formaliza el entorno de proyecto de tal forma que se puede añadir de forma natural, posteriormente, la parte de la inteligencia artificial y sistemas de gameplay que ya existen, manteniendo al mismo tiempo un fuerte control del rendimiento y cumpliendo lo que se considera mejor ingeniería de software para la documentación del título

4.4. Organización del proyecto

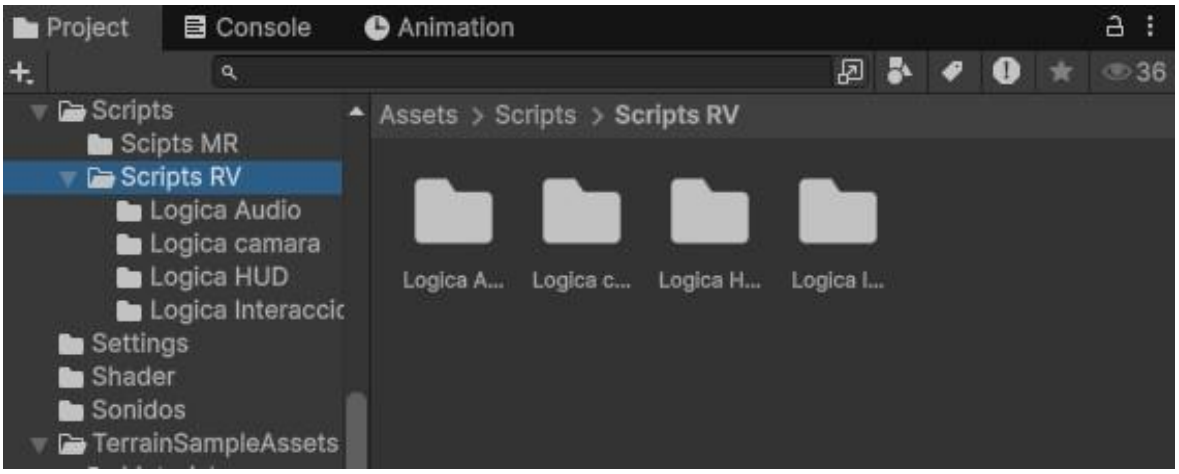
Para garantizar el orden y la eficiencia, el directorio del proyecto tiene una organización con carpetas separadas por el tipo de componente de cada uno. Esto elimina los

conflictos entre los modelos estáticos tridimensionales, los controladores de cámara, la sección de sonidos y la sección de scripts de lógica ambiental.

La organización técnica se distribuye de la siguiente manera, tal como se puede evidenciar en la ilustración inferior:

Ilustración 24

Estructura de directorios en el gestor del proyecto



Nota: Elaboración propia

- **Estructura del Código Fuente (*Scripts*):** La totalidad de los algoritmos que se redactan se centraliza bajo un directorio general. Con el fin de mantener la cohesión, los archivos se agrupan según sus dependencias procesales: lógicas condicionantes y de apertura geométrica, controladores de posicionamiento esférico de cámara, administradores de fluido de las escenas y controladores de audio.
- **Estructura de Componentes Modulares (*Prefabs*):** Con la finalidad de optimizar la creación del mapa, los elementos repetitivos del Level Design se agrupan como Prefabs. Elementos interactivos como: mecanismos de palanca, áreas sin daño y llaves de acceso se almacenan como submódulos ya configurados. Esto permite

que el diseño técnico de la fase white box se arme ágilmente con instanciaciones de piezas que ya contienen sus mallas de colisión y scripts de respuesta incrustados.

- **Estructura de Entornos (*Scenes*):** Las distintas áreas de trabajo se separan lógicamente utilizando el gestor de escenas. Se establece un entorno aislado diseñado de una forma práctica que se concibe para ejecutar depuraciones sobre las rutinas que gestionan los triggers sin comprometer la integridad de los niveles de producción. A la vez, las escenas correspondientes a la entrega final exhiben un claro aislamiento entre los estados de bloqueo contra las colisiones (White Box) y aquellas fases de inyección del arte final.

La organización implementada lleva a que el tratamiento de la totalidad de las carpetas del proyecto sea mucho más sencillo y ordenado, dado que se separa, con mucho cuidado, los códigos de programación, los objetos interactivos y los escenarios de prueba y se evita, de este modo, confundirse y equivocarse en el momento de introducir los gráficos visuales definitivos. El resultado de esto es que, el ambiente de trabajo queda estructurado de una manera muy sencilla, lo que permite, por un lado, modificar el escenario, corregir o introducir datos o incluso incorporar otros nuevos niveles, sin perjuicio alguno de destruir algo que ya está construido.

4.5. Pruebas y resultados

Con la finalidad de validar de forma integral el diseño del nivel y para garantizar el seguimiento por parte del desarrollo de los requisitos técnicos y de accesibilidad, se propone el diseño y puesta en práctica de un repertorio de pruebas combinadas. Esta forma de evaluar hace uso de pruebas funcionales de caja blanca (white box), auditorías de rendimiento o

pruebas prácticas de usabilidad espacial dentro del proyecto. Este tipo de elección técnica es relevante ya que cubre la totalidad de los factores implicados: las pruebas funcionales captan fallos algorítmicos y los huecos de topología, los perfiles del rendimiento verifican que el software es viable, y las pruebas de usabilidad comprueban que se llega a la pertinente reducción del despiste del usuario final.

La ejecución estructurada de estos ensayos se lleva a cabo bajo los siguientes parámetros e instrumentos de control:

- **Pruebas funcionales e integridad geométrica**: Varias iteraciones de pruebas de estrés se llevan a cabo, dirigidas a los límites del prototipo white box. Las pruebas van dirigidas a forzar desplazamientos extremos a través de paredes (para verificar las cajas de colisión y asegurarnos de que no haya huecos en la geometría). En paralelo, se supervisa el estado de las celdas de la malla de navegación (NavMesh) y se utilizan scripts de debugging para validar que los eventos lógicos de ocasionar aperturas se bloqueen y se lleven a cabo solo si se dan las condiciones apropiadas.

    - **Pruebas de rendimiento y de consumo de recursos:** El análisis de viabilidad técnica que se lleva a cabo es un diagnóstico con herramientas de análisis avanzado (UnityProfiler) y recolección de métricas en tiempo real. Para la ejecución de estas auditorias, se utilizo un equipo compuesto por un procesador Intel Core i7 de 13.ª generación, 16 GB de memoria RAM y una tarjeta gráfica NVIDIA GeForce RTX 460, renderizando el entorno a una resolución de 1920x1080 pixeles

Ilustración 25

Análisis de rendimiento computacional del entorno mediante (FPS)

Nota: Elaboración propia

- **Pruebas de Usabilidad y Carga Cognitiva:** La validación centrada en la experiencia interactiva ha sido llevada a cabo mediante la realización de iteraciones prácticas con un grupo piloto de 20 usuarios representativos. La herramienta de evaluación que se utiliza es la observación empírica de las trayectorias de navegación que luego es completada por la administración de un cuestionario estructurado basado en una escala de valoración de 1 a 5 (donde 1 es "Totalmente en desacuerdo" y 5 "Totalmente en de acuerdo"): durante la prueba se explora variables relevantes del Level Design como la orientación espacial, la claridad de las mecánicas, el conocimiento de los objetivos y la ambientación. (Estos resultados se pueden observar en la sección de Anexos).

En conclusión, el ambiente de control operativo en las condiciones técnicas requeridas muestra la buena realización del videojuego. Los resultados obtenidos verifican que la puesta en práctica de las técnicas de optimización disminuye satisfactoriamente y

esperada los cuellos de botella gráficos. Las métricas estadísticas obtenidas sobre la muestra de 20 usuarios muestran una percepción favorable de orientación y claridad de interacción en el grupo evaluado, sugiriendo de forma descriptiva que el diseño del nivel asiste la navegación del jugador.

4.5.1. Limitaciones del estudio

A pesar de los buenos resultados obtenidos durante la evaluación del entorno, se debe admitir que este proyecto tiene también limitaciones metodológicas. En primer lugar, la validación de usabilidad es el resultado exclusivo de una prueba piloto, a 20 usuarios, cuyo tamaño y composición impide generalizar los resultados a la totalidad de la población. Como mencionamos en segundo lugar, la cobertura de los instrumentos de recolección de datos corresponde a un cuestionario de percepción (en función de una escala de 1 a 5), donde se encuentran registradas las valoraciones subjetivas que hacen los participantes, pero no constituyen una medida cuantitativa singular de variables complejas como la carga cognitiva producida, el nivel de frustración efectivamente vivido o cuántos errores mecánicos aparecen. Finalmente, el diseño de la evaluación no incluía un estudio comparativo entre la versión del nivel con las ayudas implementadas en una versión estándar (sin asistencia), lo que impide arribar a la conclusión de que está siendo reducida en aislamiento las barreras de interacción.

4.6. Conclusiones

Una vez concluido el desarrollo y la estructuración técnica del proyecto para el videojuego “Ashes of Faith”, se lograron las siguientes conclusiones:

- **Validación espacial mediante prototipado:** La creación del primer nivel del escenario mediante la implementación de la técnica de la caja gris fue un paso clave,

ya que permitió validar dimensiones, distancias de salto y las escalas de regiones antes de introducir el arte final, con el propósito de que el diseño del nivel tuviese más en cuenta la jugabilidad que únicamente la parte visual.

- **Interacción ambiental accesible:** Al entrar en el área de diseño de niveles interactivo, se establece que el abandono de las exigencias de una precisión estricta permite mitigar las barreras mecánicas existentes para lo que se busca. La implementación de detonadores espaciales (triggers) y cajas de colisión en tamaños decrecientes a lo largo y ancho de la geometría del nivel hizo posible automatizar los eventos y perdonar la imprecisión motriz del usuario, al tiempo que mantenía un flujo de la actividad del juego ininterrumpido.
- **Impacto del diseño espacial en el rendimiento:** Se manifiesta que el diseño arquitectónico del nivel va de la mano con el desempeño técnico. La adecuada segmentación del escenario aprovechando los algoritmos de ocultamiento geométrico (Occlusion Culling), indispensable para estabilizar (FPS), fue la que permitió definir el diseño del nivel, ya que con el objetivo de renderizar únicamente aquellas partes visibles del escenario se favoreció no sólo prevenir la fatiga visual del jugador, sino que se aseguró la fluidez de la experiencia.

4.7. Recomendaciones

A partir de los resultados obtenidos y de la arquitectura técnica establecida para el proyecto, se sugieren las siguientes líneas de acciones para las futuras etapas de desarrollo que podrán realizarse el en presente proyecto

- **Ampliación de pruebas de usabilidad:** Se sugiere llevar a cabo sesiones de playtesting (o prueba de jugabilidad) con un grupo de personas más amplio y variado.

Evaluar el prototipo a nivel empírico lo cual permitirá refinar aún más el escalado de las áreas de colisión tolerantes y validar la efectividad de las señales de la retroalimentación de tipo espacial.

- **Expansión de funciones de accesibilidad:** En el caso de futuras ediciones o actualizaciones del entorno, puede ser interesante empezar a observar la posibilidad de emplear menús de interfaz dinámica que permitan un remapeo de la totalidad de los controles y que se pueda intercambiar las paletas de colores de alto contraste, de tal forma que el proyecto se ciñera todavía más a las normativas como las Xbox Accessibility Guidelines.
- **Reutilización del diseño modular:** Se recomienda capitalizar la arquitectura del software desacoplada alcanzada en este nivel. Al mantener los subsistemas completamente independientes, así como la instanciación basada en prefabs, el equipo de diseño de niveles podrá acelerar notablemente la producción y el ensamblaje de nuevas áreas para el videojuego sin necesidad de volver a escribir el código fuente.

Bibliografía

[1] D. Grammenos, A. Savidis, y C. Stephanidis, «Designing universally accessible games», *Comput. Entertain.*, vol. 7, n.º 1, pp. 1-29, feb. 2009, doi: 10.1145/1486508.1486516.

[2] J. R. Porter y J. A. Kientz, «An empirical study of issues and barriers to mainstream video game accessibility», en *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, Bellevue Washington: ACM, oct. 2013, pp. 1-8. doi: 10.1145/2513383.2513444.

[3] J. Kulik, J. Beeston, y P. Cairns, «Grounded Theory of Accessible Game Development», en *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, Montreal QC Canada: ACM, ago. 2021, pp. 1-9. doi: 10.1145/3472538.3472567.

[4] A. Wilson y M. Crabb, «W3C Accessibility Guidelines for Mobile Games», *Comput Game J*, vol. 7, n.º 2, pp. 49-61, jun. 2018, doi: 10.1007/s40869-018-0058-7.

[5] P. Ng y K. Nesbitt, «Informative sound design in video games», en *Proceedings of The 9th Australasian Conference on Interactive Entertainment: Matters of Life and Death*, Melbourne Australia: ACM, sep. 2013, pp. 1-9. doi: 10.1145/2513002.2513015.

[6] M. Á. Oliva-Zamora y C. Mangiron, «How to create video games with cognitive accessibility features: a literature review of recommendations», *Univ Access Inf Soc*, vol. 24, n.º 3, pp. 2805-2818, ago. 2025, doi: 10.1007/s10209-025-01231-5.

[7] AbleGamers, «Accessible Controller Configurations and Input Mapping for PC Gaming».

[8] K. Brown, «Making Games Better for Gamers with Disabilities».

[9] L. Morgado, D. Beck, y P. O'Shea, «Bridging the gaps: an updated mapping of the uses of immersive learning environments», *Virtual Reality*, vol. 29, n.º 3, p. 134, ago. 2025, doi: 10.1007/s10055-025-01208-y.

[10] R. Ossmann y K. Miesenberger, «Guidelines for the Development of Accessible Computer Games», en *Computers Helping People with Special Needs*, vol. 4061, K. Miesenberger, J. Klaus, W. L. Zagler, y A. I. Karshmer, Eds., en Lecture Notes in Computer Science, vol. 4061. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 403-406. doi: 10.1007/11788713_60.

[11] T. Westin, «Game accessibility course design modules in higher education», *Front. Comput. Sci.*, vol. 6, p. 1182541, abr. 2024, doi: 10.3389/fcomp.2024.1182541.

[12] T. Westin, K. Bierre, D. Gramenos, y M. Hinn, «Advances in Game Accessibility from 2005 to 2010», en *Universal Access in Human-Computer Interaction. Users Diversity*, vol. 6766, C. Stephanidis, Ed., en Lecture Notes in Computer Science, vol. 6766. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 400-409. doi: 10.1007/978-3-642-21663-3_43.

[13] J. Beeston, C. Power, P. Cairns, y M. Barlet, «Accessible Player Experiences (APX): The Players», en *Computers Helping People with Special Needs*, vol. 10896, K. Miesenberger y G. Kouroupetroglou, Eds., en Lecture Notes in Computer Science, vol. 10896. , Cham: Springer International Publishing, 2018, pp. 245-253. doi: 10.1007/978-3-319-94277-3_40.

[14] A. Järvinen, «Theory as Game: Designing the Gamegame».

[15] P. Cairns, C. Power, M. Barlet, G. Haynes, C. Kaufman, y J. Beeston, «Enabled Players: The Value of Accessible Digital Games», *Games and Culture*, vol. 16, n.º 2, pp. 262-282, mar. 2021, doi: 10.1177/1555412019893877.

[16] C. Widden, «The Impact of Accessibility Features on Player Experience in Video Games».

[17] J. Zheng, «Human factors in game design: The importance of accessibility», *ACE*, vol. 42, n.º 1, pp. 102-110, feb. 2024, doi: 10.54254/2755-2721/42/20230696.

[18] J. Sánchez, L. Guerrero, M. Sáenz, y H. Flores, «A Model to Develop Videogames for Orientation and Mobility», en *Computers Helping People with Special Needs*, vol. 6180, K. Miesenberger, J. Klaus, W. Zagler, y A. Karshmer, Eds., en Lecture Notes in Computer Science, vol. 6180. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 296-303. doi: 10.1007/978-3-642-14100-3_44.

[19] O. Naranjo, G. Perez, M. L. Ibanez, y F. Peinado, «Developing a Video Game Accessibility Toolkit for the Deaf and Hearing Impaired».

[20] MIcrosoft, «Xbox Accessibility Guidelines (XAGs) for PC and Console».

[21] M. Vigele, «Accessibility in Practice: A Case Study on a Viennese Video Game Studio», p. 87 pages, 2024, doi: 10.34726/HSS.2024.112763.

[22] P. Cairns, C. Power, M. Barlet, y G. Haynes, «Future design of accessibility in games: A design vocabulary», *International Journal of Human-Computer Studies*, vol. 131, pp. 64-71, nov. 2019, doi: 10.1016/j.ijhcs.2019.06.010.

[23] R. Andrade, M. J. Rogerson, J. Waycott, S. Baker, y F. Vetere, «Playing Blind: Revealing the World of Gamers with Visual Impairment», en *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow Scotland Uk: ACM, may 2019, pp. 1-14. doi: 10.1145/3290605.3300346.

[24] Eidos Montréal, «Eidos-Montréal, "Shadow of the Tomb Raider"».

[25] Xbox y MIcrosoft, «The Coalition, "Gears 5"»

[26] Naughty Dog y PlayStation, «Accessibility options for The Last of Us Part II».

[27] Santa Monica y PlayStation, «Santa Monica Studio, "God of War Ragnarök"».

[28] D. Pinelle, N. Wong, y T. Stach, «Heuristic evaluation for games: usability principles for video game design», en *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Florence Italy: ACM, abr. 2008, pp. 1453-1462. doi: 10.1145/1357054.1357282.

[29] Unity, *Unity*.

[30] J. Gregory, *Game engine architecture*, Third edition. en An A.K. Peters book. Boca Raton London New York: CRC Press, Taylor & Francis Group, 2019.

[31] R. Nystrom, *Game programming patterns*. s.l.: genever benning, 2014.

[32] Ken Schwaber y Jeff Sutherland, *The Scrum Guide*.

[33] C. Chen *et al.*, «SoundSpaces: Audio-Visual Navigation in 3D Environments», 21 de agosto de 2020, *arXiv*: arXiv:1912.11474. doi: 10.48550/arXiv.1912.11474.

[34] A. Stellman y J. Greene, *Learning Agile: understanding Scrum, XP, Lean, and Kanban*, First edition. Sebastopol, CA: O'Reilly Media, 2015.

[35] «Metodologia scrum».

[36] D. J. Anderson, *Kanban: successful evolutionary change for your technology business*. Sequim, Washington: Blue Hole Press, 2010.

[37] «Metodologia Kanban».

[38] K. Beck y C. Andres, *Extreme programming explained: embrace change*, 2. ed., 11. print. en The XP Series. Boston: Addison-Wesley, 2012.

[39] «Metodologia programacion extrema XP».

[40] C#, *C#*.

[41] Unreal Engine, *Unreal Engine*.

[42] B. Stroustrup, *The C++ programming language: C++ 11*, 4. ed., 4. print. Upper Saddle River, NJ: Addison-Wesley, 2015.

[43] Scott Chacon y Ben Straub, *Pro Git*.

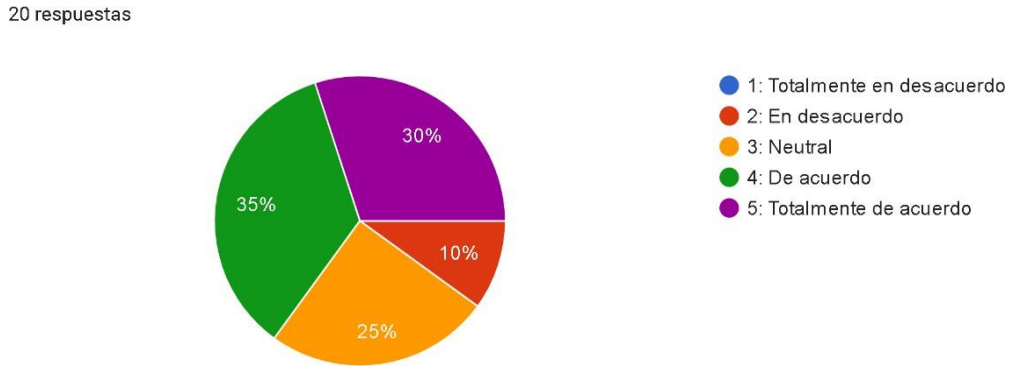
[44] I. Sommerville, «SOFTWARE ENGINEERING».

Anexos

En lo que respecta a la orientación espacial, se puede observar que un 65% de la muestra respondió de manera satisfactoria (35% "de acuerdo" y 30% "totalmente de acuerdo") a la pregunta: "¿es fácil orientarse y encontrar el camino por el pueblo?", mientras que un 25% se mantiene en una posición "neutral". De esta manera, la señalización implícita y la arquitectura del nivel habrían orientado al conjunto de los usuarios implicados, en efecto.

Ilustración 26

Resultados de evaluación sobre orientación espacial

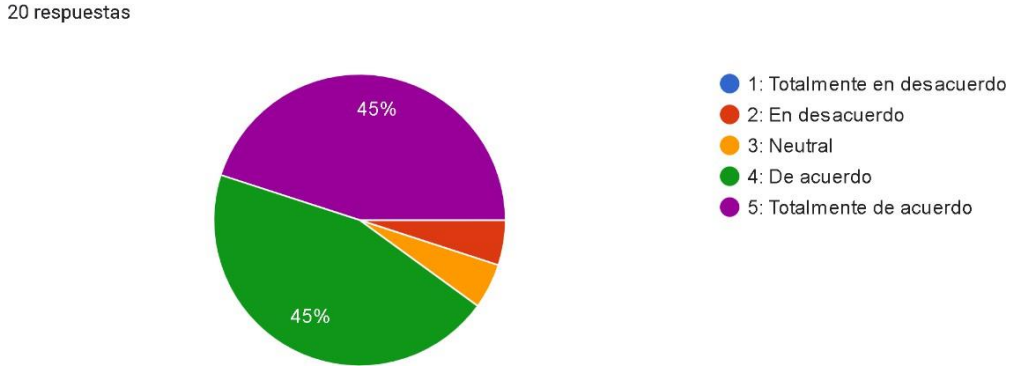


Nota: Elaboración propia

En segundo lugar, se estudia la programación del módulo de interacción. Al responder a la pregunta sobre si la interacción con los objetos del entorno fue clara y fácil de entender, se recibe una favorabilidad total del 90%, distribuyéndose en un 45% "Totalmente de acuerdo" y 45% "De acuerdo". Esto demuestra la eficacia de los scripts Intreacciones.cs y UiMessage.cs para disminuir la fricción mecánica entre el usuario y los puzzles del entorno.

Ilustración 27

Resultados de evaluación sobre claridad de interacciones

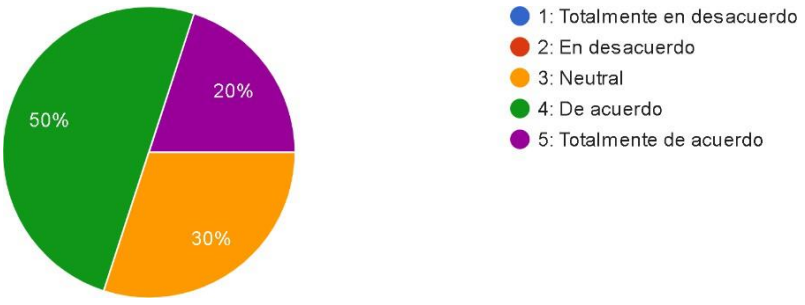


Nota: Elaboración propia

En este tercer punto se realiza valoraciones sobre la carga cognitiva y la claridad del flujo de juego. Al consultar acerca de la comprensión previa al juego, se responde de manera favorable para la mayoría de los evaluados (50% "De acuerdo" y 20% "Totalmente de acuerdo") sin que se den casos de preguntarse nada severamente (0% en desacuerdo,) esto sirve para validar el ritmo de avance (pacing) diseñado para el avance del nivel.

Ilustración 28

Resultados de evaluación sobre comprensión de objetivos



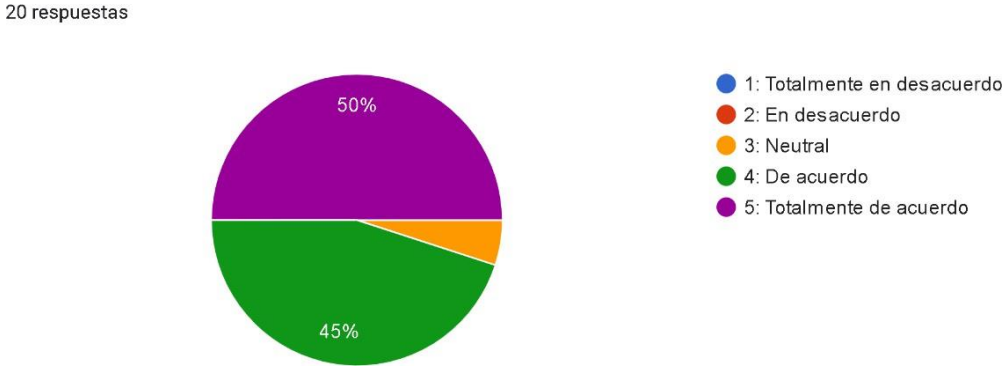
Nota: Elaboración propia

Por último, se interpreta la coherencia entre la programación del entorno y el arte final. En cuanto a la ambientación de los escenarios y si ésta incrementa la experiencia de

juego, también aquí, una gran mayoría opina favorablemente, reflejándose en un 95% de los casos, en donde el 50% "Totalmente de acuerdo" y 45% "De acuerdo".

Ilustración 29

Resultados de evaluación sobre impacto de la ambientación



Nota: Elaboración propia