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

Diseño, desarrollo y evaluación de un prototipo de plataformas 2D con mecánica de riesgo-recompensa: Veins to Light

PROYECTO DE INTREGRACIÓN CURRICULAR PREVIO A LA OBTENCION DEL TÍTULO DE INGENIERO EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTOR: ESTEBAN MATEO UYAGUARI CÓRDOVA

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

Diseño, desarrollo y evaluación de un prototipo de plataformas 2D con mecánica de riesgo-recompensa: Veins to Light

PROYECTO DE INTREGRACIÓN CURRICULAR PREVIO A LA OBTENCION DEL TÍTULO DE INGENIERO EN REALIDAD VIRTUAL Y VIDEOJUEGOS

AUTOR: ESTEBAN MATEO UYAGUARI CÓRDOVA

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

Esteban Mateo Uyaguari Córdova portador(a) de la cédula de ciudadanía Nº **0107190167**. Declaro ser el autor de la obra: **"Diseño, desarrollo y evaluación de un prototipo de plataformas 2D con mecánica de riesgo-recompensa: Veins to Light."**, 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, **26 de agosto de 2026**

F: .....

Esteban Mateo Uyaguari Córdova

C.I. **0107190167**

Universidad Católica de Cuenca

CERTIFICADO

Certificado

Certifico que la presente Propuesta Tecnológica fue desarrollado por **Esteban Mateo Uyaguari Córdova** portador(a) de la cedula de ciudadanía N.º **0107190167** con el tema "**Diseño, desarrollo y evaluación de un prototipo de plataformas 2D con mecánica de riesgo-recompensa: Veins to Light.**", bajo mi supervisión.

Cuenca, **26 de agosto de 2026**

F: .....

Ing. Jheyson Steven Gaona Pineda, Mtr.

Docente - Tutor

Dedicatoria

Dedico este trabajo a mis padres por ser el pilar fundamental de mi vida y el ejemplo de esfuerzo, perseverancia y dedicación que me ha inspirado a alcanzar cada una de mis metas, agradezco su amor incondicional y por confiar siempre en mí.

Se lo dedico de igual manera a mi hermana por su cariño, comprensión y apoyo constante, su compañía y sus palabras de aliento han sido una fuente de motivación para afrontar los retos y culminar esta importante etapa de mi formación profesional.

Este logro representa el resultado del esfuerzo de muchos años y está dedicado a quienes me han acompañado brindándome su confianza y apoyo incondicional.

Mateo Uyaguari Córdova

Agradecimiento

Agradezco primero a Dios por darme la fortaleza, salud y la sabiduría necesaria para superar cada desafío y permitirme culminar esta importante etapa de mi vida.

Expreso mi más profundo agradecimiento a mis padres por su amor, paciencia y apoyo incondicional, ellos me impulsaron a seguir adelante mientras creo en mis capacidades y enseñarme que la dedicación y el esfuerzo son la base para alcanzar cualquier objetivo.

Agradezco a mi hermana por estar presente con su apoyo, comprensión y palabras de aliento en los momentos más exigentes de este proceso.

A mi tutor de tesis por su orientación, dedicación y valiosos conocimientos compartidos durante el desarrollo de este trabajo, sin él este trabajo no hubiera sido posible.

A mis compañeros por compartir conmigo experiencias, aprendizajes y momentos que hicieron de esta etapa una experiencia de crecimiento tanto académico como personal.

Finalmente, agradezco a todas las personas que contribuyeron al desarrollo de este proyecto y me acompañaron durante esta etapa dejando a su vez una huella significativa en mi formación.

Mateo Uyaguari Córdova

Resumen

Las mecánicas de riesgo-recompensa constituyen un enfoque de diseño de videojuegos que busca equilibrar los beneficios obtenidos por el jugador frente a las consecuencias derivadas de asumir determinados riesgos durante la partida. Este tipo de mecánicas promueve la toma de decisiones estratégicas al convertir recursos limitados en elementos fundamentales para la progresión y el desarrollo de la experiencia de juego. Sin embargo, en los videojuegos de plataformas 2D la barra de vida suele limitarse a una función de supervivencia, con poca participación en las mecánicas principales del juego. El presente trabajo se enfoca en el desarrollo de un prototipo funcional de un videojuego de plataformas 2D denominado "Veins to Light", implementado en el motor de Unity y utilizando lenguaje de programación C#. Para su desarrollo se emplea la metodología ágil Scrum, organizada en iteraciones denominados sprints que facilitaron la implementación progresiva de cada uno de los sistemas del prototipo, estructurado en cuatro componentes principales: (i) personajes, (ii) mecánicas de juego, (iii) diseño artístico 2D y (iv) diseño de nivel. Estos componentes se integran mediante una arquitectura modular que coordina el sistema de movimiento, la gestión de vida, las habilidades del jugador, los enemigos, la interfaz de usuario y la progresión entre niveles. La validación se realizó mediante una encuesta Likert a 30 participantes, con excelente consistencia interna ($\alpha = 0.933$) y una valoración promedio de 4.51/5, evidenciando una percepción favorable hacia la mecánica principal, la integración de sistemas y la experiencia de juego, respaldando así la viabilidad de la propuesta.

***Palabras clave**: Riesgo-recompensa, Videojuego 2D, Plataformas 2D, Unity.*

Abstract

Risk-reward mechanics constitute a game design approach that seeks to balance the benefits obtained by players against the consequences of taking specific risks during gameplay. This type of mechanics encourages strategic decision-making by transforming limited resources into essential elements for progression and the overall gameplay experience. However, in 2D platformer games, the health bar is typically restricted to a survival function, with minimal involvement in the core gameplay mechanics. This paper focuses on the development of a functional prototype of a 2D platformer game titled Veins to Light, developed using the Unity game engine and the C# programming language. The project was carried out following the Scrum agile methodology, organized into iterative development cycles known as sprints, which facilitated the incremental implementation of each system within the prototype. The prototype is structured around four main components: (i) characters, (ii) gameplay mechanics, (iii) 2D art design, and (iv) level design. These components are integrated through a modular architecture that coordinates the movement system, health management, player abilities, enemy behavior, the user interface, and level progression. The prototype was evaluated through a Likert-scale survey administered to 30 participants, achieving excellent internal consistency (Cronbach's $\alpha = 0.933$) and an average rating of 4.51 out of 5. The results indicate a positive perception of the core gameplay mechanics, system integration, and overall gameplay experience, supporting the feasibility of the proposed approach.

Keywords: *Risk-reward mechanics, 2D video game, 2D platformer, Unity*

Índice de contenido

Índice de tablas

Índice de figuras

Índice de ilustraciones

Introducción

La industria de los videojuegos ha experimentado un crecimiento sostenido en los últimos años, consolidándose como uno de los sectores tecnológicos y de entretenimiento con mayor impacto a nivel mundial, lo que ha impulsado el desarrollo de nuevas herramientas, motores gráficos y metodologías que permiten crear experiencias interactivas cada vez más complejas, aunque dentro del género de plataformas 2D muchas propuestas siguen apoyándose en mecánicas tradicionales que, si bien han demostrado ser efectivas, ofrecen un margen limitado para la innovación en sus sistemas centrales de jugabilidad.

Uno de los elementos que ha permanecido prácticamente invariable en este género es la barra de vida del personaje, la cual suele cumplir únicamente una función pasiva vinculada con la supervivencia, pues en la mayoría de los videojuegos de plataformas la vida representa el margen de error del jugador frente a los obstáculos y enemigos sin influir de manera directa en las decisiones estratégicas que se toman durante la partida.

En este contexto surge Veins to Light, un prototipo de videojuego de plataformas 2D desarrollado en Unity con el lenguaje de programación C#, cuya propuesta consiste en implementar una mecánica de riesgo-recompensa que convierte la barra de vida en un recurso activo para ejecutar habilidades de movimiento, de modo que el jugador debe decidir constantemente entre conservar su salud para aumentar sus probabilidades de supervivencia o consumirla para obtener ventajas de movilidad que le permitan superar desafíos con mayor eficiencia.

El presente trabajo se centra en el diseño e implementación de los sistemas que conforman el prototipo, empleando una arquitectura que facilite el desarrollo, mantenimiento y escalabilidad del proyecto, entre cuyos principales componentes se encuentran el sistema

de movimiento del personaje, la gestión de vida y habilidades, el sistema de enemigos, los puntos de control, la interfaz de usuario y la progresión entre niveles, organizando el desarrollo mediante la metodología ágil Scrum que permite una implementación iterativa de las funcionalidades a través de sprints orientados a objetivos específicos.

Como resultado, esta propuesta tecnológica demuestra la viabilidad de implementar una mecánica basada en la gestión activa de la vida como recurso principal del juego al ofrecer una alternativa al diseño convencional de los videojuegos de plataformas 2D, y al mismo tiempo evidencia la aplicación de conocimientos de ingeniería de software y desarrollo de videojuegos mediante el uso de Unity y C#.

Capítulo I

1. Marco referencial

Este capítulo reúne los elementos que sirven para delimitar y orientar el proyecto. Primero se expone la problemática vinculada con la comprensión de los fundamentos lógicos aplicados a la programación de mecánicas de videojuego, después se presenta la justificación y se definen los objetivos junto con el alcance, y finalmente se describe la metodología empleada durante el desarrollo.

1.1. Contextualización del problema

La industria de los videojuegos ha experimentado un crecimiento sostenido durante los últimos años, en especial el género de plataformas 2D que continúa siendo una alternativa vigente tanto para estudios independientes como para desarrolladores consolidados, debido a la accesibilidad de sus herramientas de desarrollo, la amplia aceptación por parte de los jugadores y la posibilidad de implementar mecánicas centradas en la precisión, la exploración y la habilidad.

A pesar de su permanencia en el mercado, el desarrollo de videojuegos de plataformas 2D continúa basándose, en gran medida, en estructuras de diseño tradicionales, esto se ve reflejado en títulos que mantienen mecánicas lineales y sistemas de progresión convencionales que garantizan una experiencia de juego funcional, pero que al mismo tiempo reducen las oportunidades de innovación y dificultan la incorporación de nuevas dinámicas de interacción [1]. Diversos estudios recientes señalan que los desarrolladores suelen priorizar fórmulas previamente consolidadas, lo que conduce a propuestas con características similares y limita la exploración de mecánicas alternativas dentro del género [2].

La barra de vida es uno de los sistemas más comunes, este elemento funciona como un indicador de supervivencia que disminuye cuando el personaje recibe daño y determina el margen de error antes de que el jugador pierda la partida [3][4]. Aunque este enfoque ha demostrado ser efectivo para representar el estado del personaje, también refleja una tendencia hacia el uso de sistemas independientes que cumplen funciones específicas dentro del juego, evidenciando el contexto en el que surgen nuevas propuestas orientadas a integrar de forma más estrecha los diferentes sistemas que conforman la experiencia de juego.

1.2. Planteamiento del Problema

En la mayoría de estos videojuegos, el sistema de gestión de vida funciona únicamente como un indicador pasivo de supervivencia, disminuyendo la salud del jugador cuando recibe daño y estableciendo el margen de error antes de perder la partida. Todo resulta en que esta no participe activamente en las mecánicas principales ni que mantenga una relación directa con otros sistemas que conforman la jugabilidad, desaprovechando su potencial como un recurso capaz de influir en la toma de decisiones del jugador [3][4].

Desde la perspectiva del desarrollo de software, esta situación se encuentra estrechamente relacionada con el diseño de la arquitectura del videojuego. Habitualmente, los módulos encargados de administrar la salud, el movimiento, las habilidades y la interfaz de usuario son implementados de manera independiente, generando una separación rígida entre subsistemas. Esta estructura dificulta el desarrollo de mecánicas en las que un mismo recurso controle múltiples comportamientos del juego, reduce la flexibilidad del sistema, limita la reutilización de componentes y aumenta la complejidad para mantener la consistencia del estado del juego cuando diferentes sistemas deben interactuar de forma coordinada [3].

Diversas investigaciones han demostrado que las mecánicas de riesgo-recompensa favorecen el engagement, incrementan la inmersión y promueven una toma de decisiones más estratégica al obligar al jugador a administrar recursos limitados [4][5][6]. Sin embargo, este enfoque ha sido escasamente aplicado cuando la salud del personaje se plantea como un recurso dinámico e integrado con las mecánicas de movimiento y habilidades dentro de los videojuegos de plataformas 2D, evidenciando una oportunidad para explorar nuevas propuestas tanto desde el diseño de juego como desde la arquitectura de software.

En consecuencia, el problema radica en la prevalencia de arquitecturas de software tradicionales en los videojuegos de plataformas 2D, las cuales mantienen el sistema de salud aislado de los subsistemas de movimiento y habilidades. Esta separación limita la implementación de mecánicas interconectadas basadas en riesgo-recompensa, reduce la flexibilidad y reutilización de los componentes del software, dificulta la mantenibilidad del código y restringe el desarrollo de experiencias de juego más dinámicas.

1.3. Formulación del problema

¿Cómo desarrollar una arquitectura de software para un videojuego de plataformas 2D que integre el sistema de salud con las mecánicas de movimiento y habilidades mediante una mecánica de riesgo-recompensa, con el fin de enriquecer la experiencia de juego?

1.4. Justificación

El presente proyecto encuentra su justificación en el aporte que realiza al área de desarrollo de videojuegos siendo particularmente en la implementación de sistemas de software para títulos de plataformas 2D mediante el uso del motor Unity y el lenguaje de programación C#, lo que permite aplicar de manera práctica conocimientos de ingeniería de software junto con principios de programación orientada a objetos y diseño de sistemas

interactivos, integrando en un mismo entorno los distintos componentes que hacen posible la construcción de un videojuego funcional.

Desde el punto de vista tecnológico, el proyecto plantea una alternativa frente a la implementación convencional de la barra de vida, transformándola de un recurso pasivo de supervivencia en un sistema activo que interviene directamente en las mecánicas de movimiento del jugador, lo que exige la integración de distintos subsistemas como el control del personaje, la administración de recursos, las habilidades, la interfaz de usuario, los enemigos y la progresión del juego, permitiendo con ello demostrar la viabilidad técnica de una arquitectura capaz de coordinar estos componentes de manera eficiente y coherente dentro de la experiencia interactiva [3][4].

Asimismo, el desarrollo del prototipo contribuye a la aplicación de buenas prácticas de programación al fomentar la creación de sistemas desacoplados, reutilizables y escalables, lo que facilita tanto el mantenimiento del código como la incorporación de nuevas funcionalidades en etapas posteriores de desarrollo, y al mismo tiempo aprovecha las posibilidades que ofrecen Unity y C#, permitiendo aplicar patrones de diseño y metodologías ampliamente utilizadas en la industria de los videojuegos, fortaleciendo con ello la calidad técnica del software construido.

Diversas investigaciones señalan que este tipo de dinámicas favorecen el engagement, la inmersión y la toma de decisiones estratégicas al situar al jugador frente a la administración de recursos limitados [5][6]; en este sentido, la propuesta de una mecánica donde la vida se convierte en un recurso consumible abre la posibilidad de explorar soluciones tecnológicas capaces de integrar múltiples sistemas de manera coordinada.

El desarrollo de este proyecto se presenta como una propuesta tecnológica que pone en evidencia la aplicación de competencias propias de la ingeniería en el ámbito del

desarrollo de videojuegos, integrando programación, diseño de sistemas interactivos y arquitectura de software dentro de un prototipo funcional que sirve como demostración práctica, además de comprobar la factibilidad de implementar una mecánica mediante una estructura de software organizada y escalable.

1.5. Objetivo general

Desarrollar un prototipo funcional de un videojuego de plataformas 2D utilizando Unity y el lenguaje de programación C#, implementando una arquitectura que permita integrar un sistema de gestión de vida concebido como recurso activo para la ejecución de habilidades de movimiento, con el fin de demostrar la viabilidad técnica de una mecánica de riesgo-recompensa aplicada al desarrollo de videojuegos, articulando en un mismo entorno los distintos componentes que conforman la experiencia interactiva.

1.6. Objetivos específicos

- Diseñar la arquitectura de software del prototipo, definiendo los módulos y la interacción entre los sistemas de movimiento, gestión de vida, habilidades, enemigos, interfaz de usuario y progresión del juego.
- Implementar en Unity los sistemas funcionales que conforman el prototipo, integrando la mecánica de conversión de la barra de vida en un recurso consumible para la ejecución de habilidades de movimiento.
- Integrar y validar el funcionamiento de los diferentes sistemas del videojuego mediante pruebas funcionales que permitan verificar si el juego cumple con las métricas de jugabilidad.

1.7. Alcance del proyecto

El presente proyecto abarca el diseño, desarrollo e integración de un prototipo funcional de un videojuego de plataformas 2D construido en el motor Unity mediante el lenguaje de programación C#, y su alcance se centra de manera específica en el desarrollo de los sistemas de software que sostienen la jugabilidad del prototipo con el fin de demostrar la viabilidad técnica de una mecánica de riesgo-recompensa basada en la utilización de la barra de vida como recurso activo para la ejecución de habilidades de movimiento, articulando en un mismo entorno los distintos componentes que hacen posible la experiencia interactiva.

Para cumplir con este objetivo, el proyecto contempla la implementación de los siguientes módulos de software:

- Sistema de control del jugador, incluyendo desplazamiento horizontal, salto, salto variable, coyote time, jump buffer, deslizamiento en paredes, salto en pared y dash direccional.
- Sistema de gestión de vida, encargado de administrar el consumo, recuperación y actualización del recurso utilizado tanto para la supervivencia como para la ejecución de habilidades especiales.
- Sistema de habilidades de movimiento, integrado con la gestión de vida para controlar el uso de las habilidades y sus restricciones.
- Sistema de enemigos, incluyendo comportamientos básicos, detección del jugador, patrones de ataque e interacción con el sistema de daño.
- Sistema de checkpoints y respawn del jugador, permitiendo el almacenamiento del progreso dentro del nivel y la restauración del estado del personaje.

- Sistema de administración del juego, responsable del control de estados, cronómetro, transición entre escenas y gestión general del flujo de ejecución.
- Sistema de interfaz de usuario (HUD) destinado exclusivamente a la visualización de información relacionada con los sistemas implementados como la barra de vida, cronómetro y demás indicadores funcionales.
- Integración de los módulos implementados mediante una arquitectura que facilite el mantenimiento, la reutilización y la escalabilidad del software.

Como producto final se desarrollará un vertical slice funcional compuesto por tres niveles lineales con progresión de dificultad, conformados por un nivel tutorial y dos niveles base, suficientes para demostrar el correcto funcionamiento e integración de los sistemas implementados y validar la viabilidad técnica de la mecánica principal propuesta.

1.8. Delimitaciones

El presente proyecto no contempla el desarrollo de un videojuego comercial completo ni la producción de contenido artístico de alta fidelidad como ilustraciones finales, animaciones avanzadas, efectos visuales complejos, diseño sonoro profesional o narrativas extensas, y del mismo modo quedan fuera de su alcance la implementación de funcionalidades en línea, la incorporación de inteligencia artificial avanzada, la generación procedural de niveles, los sistemas de guardado complejos, la optimización para múltiples plataformas y los procesos vinculados con la publicación o comercialización del producto.

1.9. Beneficiarios

1.9.1. Beneficiarios directos

Jugadores de videojuegos de plataformas 2D, quienes dispondrán de una experiencia de juego que incorpora una mecánica de riesgo-recompensa basada en la integración del sistema de salud con las habilidades y el movimiento.

Desarrolladores del proyecto, al contar con una arquitectura de software que facilita la integración entre los diferentes subsistemas del videojuego, mejora la mantenibilidad del código y favorece la escalabilidad para incorporar nuevas funcionalidades.

1.9.2. Beneficiarios indirectos

Estudiantes e investigadores del área de desarrollo de videojuegos e ingeniería de software, quienes podrán utilizar el proyecto como un caso de estudio sobre la implementación de arquitecturas y el diseño de mecánicas interconectadas en videojuegos de plataformas 2D.

Desarrolladores independientes y pequeños estudios de videojuegos, que podrán tomar como referencia las soluciones arquitectónicas y las estrategias de integración implementadas en el proyecto para aplicarlas o adaptarlas en desarrollos futuros.

1.10. Metodología general

El desarrollo del prototipo funcional del videojuego se aborda desde una perspectiva orientada principalmente a la ingeniería de software y a la programación de sistemas interactivos en tiempo real, lo que implica la integración de algoritmos de control de físicas, la definición de la lógica de estado de los personajes, la gestión dinámica de recursos en memoria y la construcción de interfaces de usuario reactivas capaces de responder de manera inmediata a las acciones del jugador; en este sentido, la metodología seleccionada debe

ofrecer la flexibilidad necesaria para enfrentar la complejidad inherente a la lógica de los videojuegos.

El marco metodológico se fundamenta en el enfoque ágil Scrum para dar cumplimiento a estos requerimientos, complementado con un modelo de desarrollo iterativo e incremental que resulta especialmente pertinente en el ámbito de los videojuegos, donde las metodologías tradicionales en cascada suelen mostrar poca eficiencia debido a la necesidad constante de evaluar y refinar la respuesta de las mecánicas interactivas, la estabilidad de los algoritmos de control y el rendimiento del código dentro del motor gráfico.

El enfoque ágil seleccionado permite dividir la construcción del sistema en ciclos cerrados de trabajo denominados Sprints, lo que facilita la integración progresiva de módulos de código.

La adopción de Scrum en este proyecto se justifica por su capacidad de estructurar de manera sistemática el flujo de trabajo en entornos de desarrollo de software, ya que a través de las iteraciones se establece una prioridad clara en la programación de los componentes del núcleo técnico antes de avanzar hacia la lógica secundaria o la integración de elementos del entorno, lo que asegura que el sistema central disponga de una base sólida y extensible que pueda sostenerse adecuadamente en la fase de despliegue y facilitar la evolución del prototipo en etapas posteriores.

Asimismo, el proceso incremental garantiza que cada Sprint concluya con un artefacto funcional de software para verificar de manera continua la correcta ejecución de las clases, componentes e interfaces de Unity sin comprometer la estabilidad global del proyecto.

A. Adaptación de Roles en Scrum:

Dado el contexto del proyecto de titulación y la naturaleza del desarrollo, la asignación de roles dentro del marco Scrum se define de la siguiente manera:

- Product Owner: Asume la responsabilidad central de la visión técnica y funcional del producto. Gestiona, prioriza y mantiene el Product Backlog, asegurando que las historias de usuario y los requerimientos del sistema se alineen con los objetivos de investigación y programación del prototipo.
- Scrum Master: Este rol es desempeñado por el director del proyecto, quien supervisa la correcta aplicación de las prácticas metodológicas, apoya en la resolución de bloqueos técnicos o conceptuales y valida el cumplimiento de las entregas incrementales en cada ciclo.
- Equipo de Desarrollo: Actúa como integrante del equipo de desarrollo técnico. Se encarga del diseño de la arquitectura de software, programación de algoritmos en C# dentro de Unity 2D, integración del sistema de entradas, estructuración de físicas y colisiones, e implementación de la interfaz de usuario y sistemas de retroalimentación.

B. Product Backlog:

El Product Backlog constituye una lista dinámica y priorizada que reúne los requerimientos de software, las historias de usuario y las tareas técnicas necesarias para la construcción del prototipo, funcionando como un artefacto que organiza el flujo de trabajo y permite dividir las funcionalidades del sistema en módulos independientes y medibles.

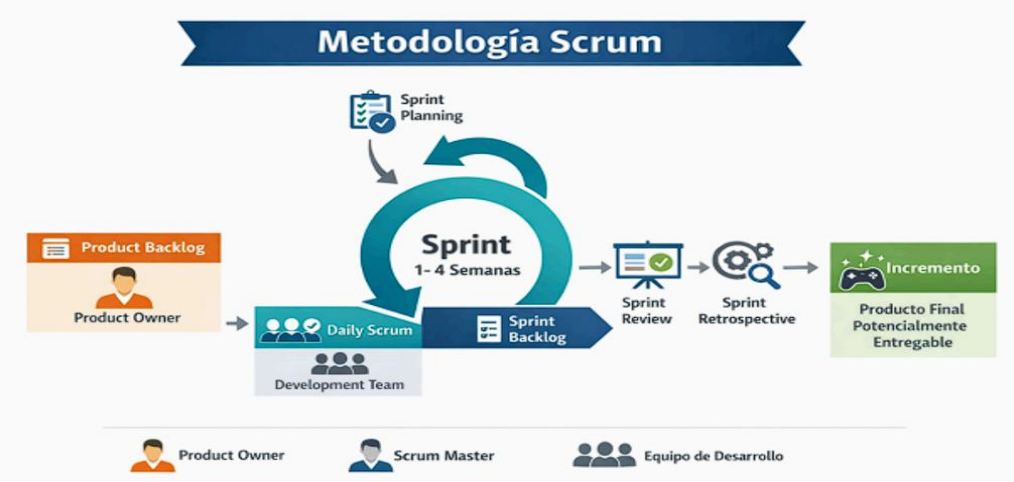
C. Planificación y Ejecución de Sprints:

El proceso de desarrollo se organiza en ciclos iterativos de trabajo, orientados a la entrega de incrementos funcionales de código y lógica del juego:

- Sprint 1 Fundamentos del Gameplay: Implementación del movimiento y salto del personaje, junto con la habilidad especial de dash. Desarrollo del sistema de consumo de vida ligado a habilidades y la visualización mediante HUD dinámico.

- Sprint 2 Feedback y Desafío Inicial: Integración de retroalimentación visual y sonora al recibir daño o usar habilidades, programación de enemigos básicos con comportamiento simple y sistema de daño, implementación de feedback visual de riesgo-recompensa para mostrar claramente el sacrificio de vida y optimización inicial de rendimiento y físicas.
- Sprint 3 Niveles y Progresión: Diseño e implementación de dos niveles con progresión de dificultad, incorporación de checkpoints para reinicios parciales y desarrollo de un nivel tutorial para enseñar las mecánicas, seguido de programación del menú principal y opciones de pausa/reinicio.
- Sprint 4 Validación y Entrega Final: Realización de testing con usuarios para evaluar la experiencia y recolectar datos.

Figura 1
Recorrido de la metodología scrum y sus miembros



Nota: Activision.

1.11. Resultados esperados

Como resultado del presente proyecto se espera obtener un prototipo funcional de un videojuego de plataformas 2D desarrollado en Unity, el cual implementará una arquitectura

de software que integre el sistema de salud con las mecánicas de movimiento y habilidades mediante un enfoque de riesgo-recompensa permitiendo demostrar la viabilidad técnica de la propuesta.

Capitulo II

2. Marco teórico

En este capítulo se exponen los fundamentos teóricos que sustentan el desarrollo de la propuesta tecnológica, abordando conceptos vinculados con la evolución de los videojuegos de plataformas 2D, las características propias de su diseño y desarrollo, las mecánicas de riesgo-recompensa y las tecnologías empleadas para la implementación del prototipo; del mismo modo, se incluyen trabajos relacionados que permiten contextualizar la investigación y respaldar las decisiones técnicas asumidas durante el proceso de construcción del videojuego, ofreciendo así un marco sólido que articula teoría y práctica.

2.1. Bases teóricas

El estado del arte reúne investigaciones y desarrollos previos vinculados con la temática de esta propuesta tecnológica y su análisis permite reconocer los principales enfoques empleados en el diseño y construcción de videojuegos de plataformas 2D, así como las distintas formas en que se han implementado mecánicas de riesgo-recompensa y las soluciones técnicas aplicadas en proyectos similares; de esta manera, la revisión se convierte en un referente que no solo fundamenta las decisiones de diseño e implementación adoptadas durante el desarrollo del prototipo, sino que también ayuda a establecer con claridad los elementos que diferencian la propuesta presentada en esta investigación.

2.1.1. Historia de los videojuegos de plataformas 2D

Los videojuegos de plataformas 2D comenzaron a consolidarse en la década de 1980 como una de las primeras formas de interacción dinámica con entornos digitales, y títulos como Donkey Kong (1981) introdujeron la mecánica del salto como elemento central, mientras que Super Mario Bros (1985) terminó de definir el género al establecer un modelo

de diseño basado en niveles progresivos y recompensas, marcando de esta forma un cambio cultural dentro de la industria al ofrecer experiencias accesibles y repetibles que podían disfrutarse tanto en arcades como en consolas domésticas, y cuya aparente simplicidad escondía un profundo potencial de inmersión que se convirtió en la base para el desarrollo posterior de propuestas más complejas [7].

Los videojuegos de plataformas 2D a medida que pasaba el tiempo evolucionaron hacia experiencias cada vez más sofisticadas, incorporando elementos narrativos y mecánicas de exploración que ampliaron las posibilidades del género; títulos como Metroid (1986) y Castlevania (1989) introdujeron sistemas de progresión y mundos interconectados, dando origen al subgénero conocido como “metroidvania ”, cuya relevancia se explica, según Juul y McDonald, por la capacidad de sus mecánicas básicas para generar significados culturales y emocionales en los jugadores más allá de su aparente simplicidad [7].

Este mismo tipo de videojuegos atravesaron una transición caracterizada durante la década de los 90 gracias a la incorporación de narrativas más complejas y estilos visuales cada vez más diversificados, que a su vez permitió ampliar el horizonte creativo del género; en este proceso, según McDonald, su vigencia se sostuvo gracias a la capacidad de las mecánicas básicas de adaptarse a nuevas formas de diseño y a distintos contextos culturales [7]. Ejemplos como Sonic the Hedgehog (1991), con una estética dinámica y veloz, o Rayman (1995), reconocido por su estilo artístico singular y su énfasis en la exploración.

Ilustración 1
Sistema de mundos interconectados de Metroid (1986)

Nota: Nintendo.

El género sufrió una pequeña revolución por los desarrolladores independientes, quienes encontraron en este formato un terreno fértil para nuevas ideas; obras como Braid (2008), Super Meat Boy (2010) y Fez (2012) renovaron el género al introducir mecánicas novedosas y un marcado componente artístico impulsando a su vez el movimiento indie[1] [7]. Este resurgimiento puso en evidencia la flexibilidad del género para incorporar nuevas formas de expresión, cosas desde la experimentación con la física del entorno hasta la construcción de narrativas metafóricas, viéndose reflejados en estudios recientes los cuales señalan que su persistencia se explica por la capacidad de generar continuidad entre generaciones de jugadores, manteniendo un lenguaje visual y mecánico en títulos contemporáneos como Ori and the Will of the Wisps (2020) y Metroid Dread (2021) [8].

[1] Indie: término utilizado para referirse a videojuegos desarrollados por equipos independientes, generalmente con menor dependencia de grandes editoras y mayor libertad creativa.

2.1.2. Características de desarrollo y diseño en videojuegos 2D

Los pilares fundamentales que sostienen el género son mecanicas como el movimiento del avatar, la progresión de niveles y la interacción con obstáculos y enemigos, y según McDonald la calidad del movimiento constituye el núcleo de la experiencia del jugador, pues define la sensación de control y agencia [7]; la progresión se articula en la estructura de niveles donde la dificultad aumenta de manera gradual y se introducen nuevas mecánicas que mantienen el interés y el desafío con el objetico de generar junto con recompensas como ítems o vidas extra una rutina de motivación [9].

El desarrollo del género ha estado marcado por la incorporación de mecánicas específicas que enriquecen la experiencia del jugador, y en este sentido Whitehead et al. proponen un marco analítico para el diseño de niveles en plataformas 2D, destacando la relevancia de los “grupos de ritmo” y de las transiciones entre caminos como factores que determinan el pacing y la complejidad del reto [9]; de manera complementaria, Richtr y Kyseľová examinan el papel de las mecánicas de cuerda y balanceo, presentes desde Pitfall (1982) hasta títulos más recientes, mostrando cómo estas dinámicas introducen variaciones en el movimiento y aportan diversidad a la progresión [10].

Ilustración 2

Mecánica de cuerda del videojuego Pitfall (1982)

Nota: Activision.

En investigaciones recientes sobre el diseño de niveles en plataformas 2D se ha subrayado la relevancia de introducir elementos inesperados que mantengan la atención del jugador y eviten la monotonía, y en este sentido Chakraborttii y Ferreira proponen un marco conceptual basado en el modelo VCL (Violation of Expectation, Caught Off Guard, and Learning), el cual demuestra poder generar una respuesta emocional significativa para potenciar la inmersión mediante la incorporación de sorpresas en la geometría de los niveles [11].

Por otro lado, Johansson et al. examinan el impacto que la generación procedural de contenido tiene en la jugabilidad de los títulos de plataformas 2D y concluyen que incluso sistemas ligeros de generación automática pueden producir niveles atractivos y estructuralmente válidos [12]; sin embargo, advierten que la calidad de la experiencia depende en gran medida de la capacidad del sistema para adaptarse al nivel de habilidad del

jugador, ya que quienes se inician en el género tienden a percibir mayor dificultad en la mecánica básica antes de enfrentarse a la complejidad del diseño.

2.1.3. Mecánicas de riesgo-recompensa

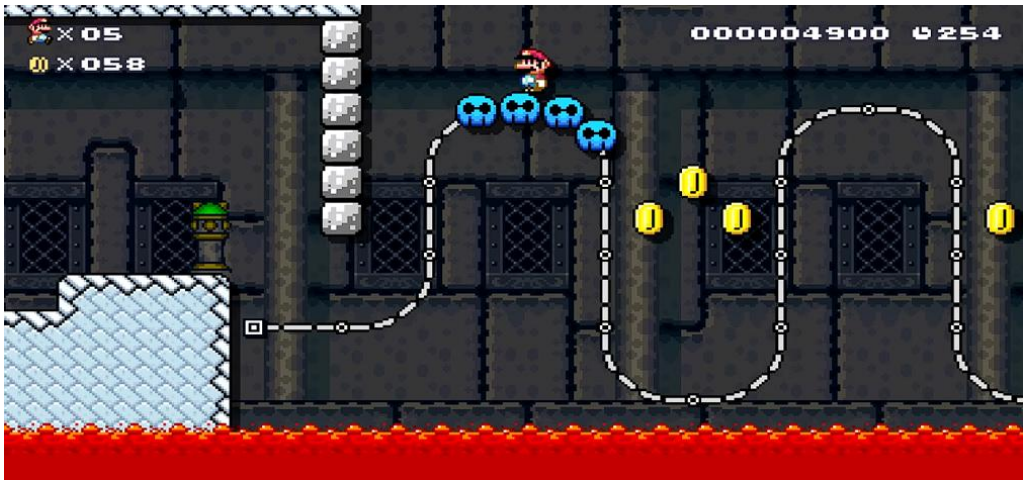
En el diseño de plataformas 2D, las mecánicas de riesgo-recompensa se manifiestan principalmente en la disposición de ítems y en la existencia de rutas alternativas, y Bonilla Carranza et al. señalan que el equilibrio entre riesgo y recompensa constituye una de las buenas prácticas más recurrentes en la literatura sobre diseño de mecánicas, ya que fomenta elecciones significativas y mantiene la motivación del jugador [13]; un ejemplo claro de ello es la colocación de un objeto valioso en una zona de difícil acceso, lo que obliga al jugador a decidir si arriesgar vidas o tiempo para obtener una ventaja futura.

Green et al. proponen la Game Mechanic Alignment Theory, un marco que organiza las mecánicas en función de recompensas ambientales e incentivos intrínsecos del jugador [14] que, bajo esta perspectiva, se observa a las mecánicas de riesgo-recompensa como un alineamiento entre motivaciones internas y externas, lo que se refleja en situaciones donde el jugador debe decidir entre avanzar por un camino seguro, pero menos gratificante, o explorar rutas más peligrosas que ofrecen recompensas varias.

Silver propone un marco cuantitativo que analiza cómo los juegos distribuyen el control entre la destreza del jugador y los elementos aleatorios, destacando que la tensión emerge precisamente cuando las decisiones arriesgadas pueden modificar de manera significativa el resultado [15]. En este tipo de experiencias el jugador se enfrenta a situaciones en las que debe calcular si vale la pena asumir el reto de un obstáculo difícil para obtener una recompensa, aun sabiendo que el factor de suerte puede influir en el desenlace.

Ilustración 3

Mecánica de plataformas móviles en el videojuego Super Mario Maker 2



Nota: Nintendo.

Investigaciones recientes sobre el comportamiento del jugador frente a recompensas intrínsecas y extrínsecas evidencian que la motivación se mantiene cuando dichas recompensas se alinean con las expectativas psicológicas propias del género [16]; en esta línea, San José Martínez analiza cómo distintos tipos de recompensas influyen en la conducta a través de géneros variados y concluye que las recompensas narrativas y simbólicas generan un compromiso más duradero que aquellas de carácter puramente material [16].

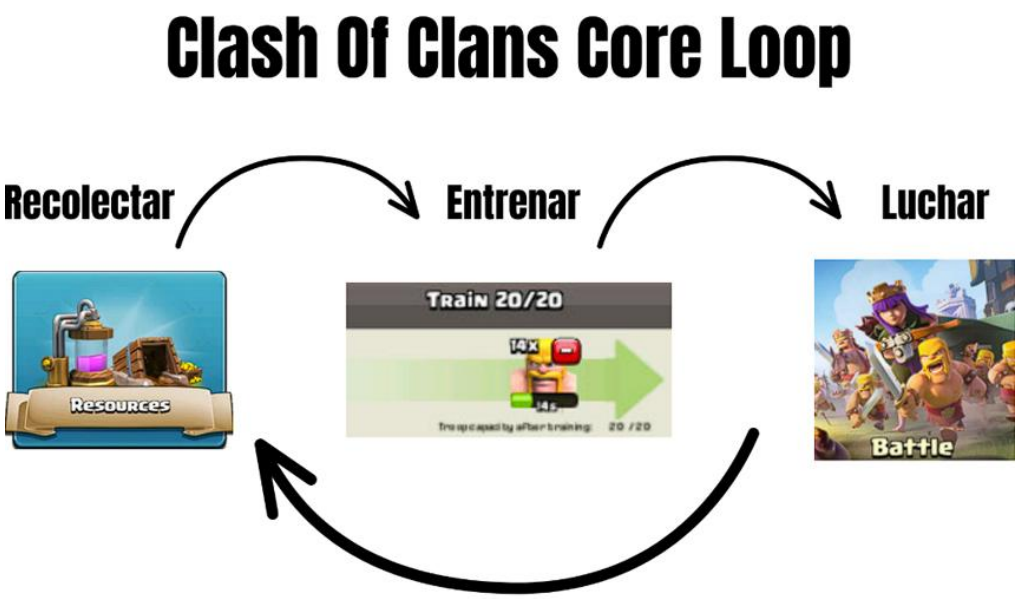
2.1.4. Bucle jugable (Core Loop)

El concepto de core loop se entiende como la secuencia fundamental de acciones que el jugador repite de manera constante durante la partida, y según Weiller este bucle funciona de forma similar a un sistema de condicionamiento operante en el que cada acción genera una respuesta que puede traducirse en recompensa o castigo manteniendo así la motivación y el compromiso [17]; movimientos básicos como correr, saltar y esquivar suelen conformar este ciclo, acompañados de recompensas inmediatas como puntos o ítems.

El autor Chiapello plantea que los bucles jugables deben comprenderse como sistemas dinámicos en equilibrio en los que las acciones del jugador y las respuestas del juego se retroalimentan de manera constante [18]; este enfoque sistémico permite visualizar el core loop como un mecanismo que asegura la coherencia del diseño, garantizando que cada interacción contribuya tanto al progreso como a la inmersión. Para generar una experiencia así sostenida en el tiempo el jugador ejecuta acciones básicas, recibe retroalimentación inmediata y se enfrenta a nuevos retos que refuerzan el ciclo.

Figura 2

Bucle jugable del videojuego Clash of Clans



Nota: Elaboración propia, Supercell.

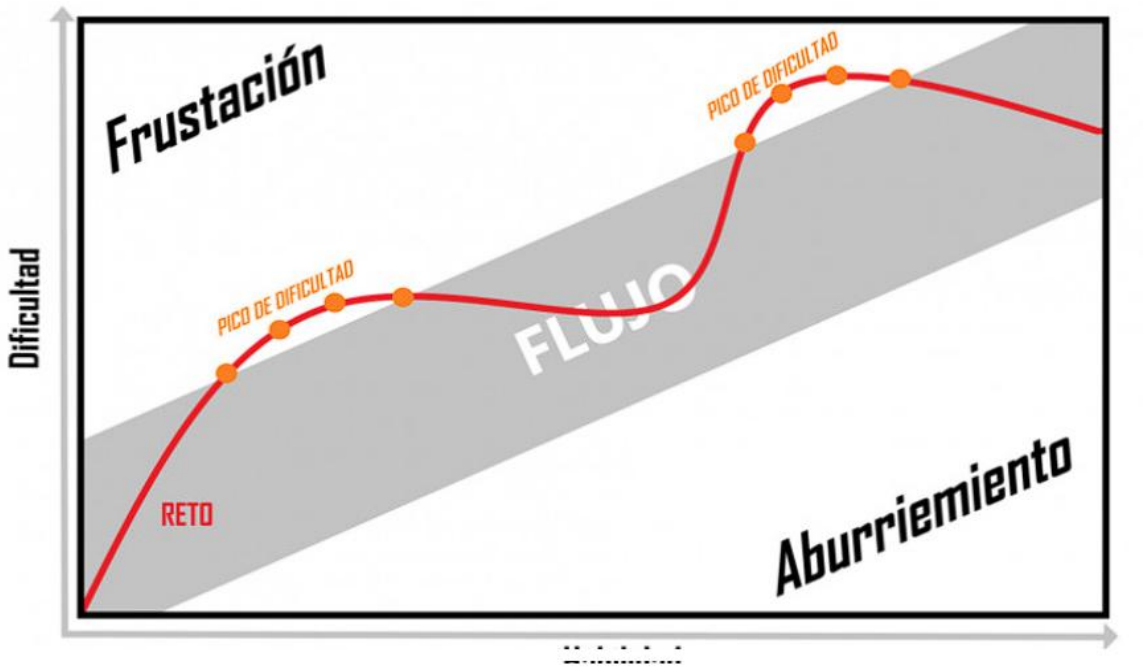
En este sentido, los videojuegos de plataformas 2D logran mantener la vigencia del género al combinar recompensas inmediatas con la construcción de un mundo coherente, donde cada repetición del ciclo contribuye a la inmersión y al significado cultural del juego.

2.1.5. Curva de dificultad

El diseño de la curva de dificultad busca que los desafíos progresen de manera coherente con las habilidades del jugador, y Sarkar y Cooper señalan que una curva bien estructurada evita que este se sienta abrumado o aburrido ya que ajusta el nivel de reto en función de la competencia adquirida [19]; en la práctica, esto se traduce en niveles iniciales que introducen mecánicas básicas y en etapas posteriores que incrementan la complejidad mediante enemigos más agresivos, plataformas móviles o combinaciones de obstáculos.

Figura 3

Grafica de dificultad de un videojuego



Nota: 3D juegos.

Por otro lado, Fatemeh et al. subrayan que los sistemas de ajuste dinámico de dificultad representan una evolución significativa en el diseño de curvas, pues permiten modificar el reto en tiempo real de acuerdo con el desempeño del jugador [20]; este enfoque resulta especialmente valioso en géneros como los de plataformas 2D, donde la precisión y

el tiempo de reacción son determinantes, y el estudio destaca que dichos ajustes no solo incrementan la accesibilidad para quienes se inician en el género, sino que también mantienen el interés de jugadores experimentados al ofrecer un desafío adaptado a sus habilidades.

Un marco analítico ha sido planteado con el fin de evaluar la dificultad de los enemigos mediante árboles de comportamiento automatizados con el fin de ofrecer una herramienta precisa para medir cómo las acciones del adversario inciden en la curva de dificultad [21].

2.2. Antecedentes de investigación

En esta sección se presentan trabajos relacionados con el diseño e implementación de mecánicas de juego en videojuegos de plataformas 2D, con énfasis en la programación de sistemas de recursos y control del jugador, junto a un análisis de los antecedentes el cual permite identificar soluciones técnicas y enfoques de desarrollo que sirven como referencia para el diseño e implementación del prototipo propuesto.

2.2.1. Hollow Knight

Es un videojuego de plataformas 2D desarrollado por el estudio independiente Team Cherry y publicado en 2017, consolidándose como un referente dentro del género gracias a la solidez de su diseño técnico y narrativo, destacando especialmente el uso de la barra de vida como recurso activo [22]. En este sistema la salud del personaje no se limita a ser un indicador de supervivencia, sino que se vincula directamente con un recurso secundario denominado “alma”, el cual permite tanto la curación como la ejecución de habilidades ofensivas [23], generando así una dinámica de riesgo y recompensa que intensifica la experiencia, sumándose un mundo interconectado, un estilo artístico dibujado a mano y una atmósfera inmersiva [24].

Ilustración 4

Estilo artístico y de diseño del videojuego Hollow Knight

Nota: Team Cherry.

Aspectos de jugabilidad destacados:

- Gestión de salud como recurso activo: la barra de vida se complementa con un sistema de curación que requiere consumir un recurso acumulado durante el combate.
- Diseño de riesgo y recompensa: el jugador debe elegir entre invertir su recurso en recuperar salud o utilizarlo para ataques especiales.
- Movimiento y control: el sistema de desplazamiento incluye saltos, esquivas y retroceso al recibir daño.
- Progresión vinculada a la salud: mejoras permanentes y temporales.

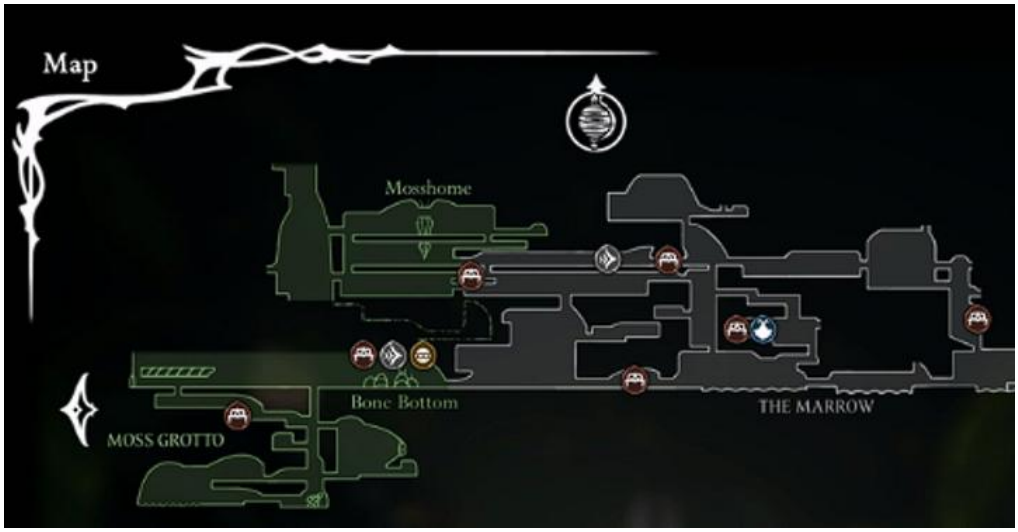
Sistema de salud y mecánicas asociadas:

- Recurso dual: la salud se gestiona en conjunto con el “alma”.
- Curación activa: la acción de recuperar vida requiere tiempo de ejecución y consumo de recurso.

- Feedback programado: efectos visuales, sonoros y temporizadores de invulnerabilidad comunican al jugador el estado de la barra de vida y las consecuencias de sus decisiones.
- Balance paramétrico: variables como cantidad de curación, tiempo de invulnerabilidad y retroceso al recibir daño son parámetros ajustables.

Ilustración 5

Interfaz del mapa de Hollow Knight: Silksong.



Nota: Team Cherry.

2.2.2. Celeste

Celeste es un videojuego de plataformas 2D desarrollado por Maddy Makes Games y publicado en 2018, siendo reconocido por la precisión de su control y por el diseño de niveles, características que la han convertido en un referente moderno del género, a esto se le suma su propuesta técnica se centra en un sistema de movimiento altamente responsivo, capaz de permitir al jugador ejecutar saltos, escaladas y desplazamientos con exactitud milimétrica [25], y este enfoque ha sido decisivo para consolidar la reputación del título como uno de los más influyentes de la última década dentro de las plataformas, sin haber

mencionado que posee una narrativa emocional y un estilo artístico dibujado a mano que enriquecen la experiencia del juego [25].

Ilustración 6
Estilo visual y narrativo de Celeste

Nota: Maddy Makes Games

Aspectos técnicos destacados:

- Movimiento y control preciso: el sistema de programación del avatar incluye mecánicas como el “dash” aéreo y la escalada en paredes.
- Curva de dificultad progresiva: los niveles están diseñados para introducir nuevas mecánicas de manera gradual.
- Feedback inmediato: cada acción del jugador genera una respuesta clara y consistente, lo que implica un sistema de programación optimizado para minimizar latencias y garantizar coherencia en la física del juego.
- Diseño de retos paramétricos: la dificultad se ajusta mediante parámetros programables como la velocidad de desplazamiento, la altura de salto y la duración del “dash”.

Sistema de mecánicas asociadas:

- Dash aéreo: implementado como una acción limitada por recursos internos el cual requiere un gestor de estados que controle disponibilidad, recarga y efectos visuales.
- Escalada en paredes: la programación incluye un sistema de resistencia que limita el tiempo de agarre.
- Checkpoints y progresión: cada nivel integra puntos de control que se gestionan mediante scripts de guardado y restauración de estado.
- Optimización de físicas: la interacción entre gravedad, fricción y colisiones está ajustada para ofrecer una sensación de fluidez.

Ilustración 7
Diseño de nivel y mecánica de dash del videojuego Celeste

Nota: Maddy Makes Games

2.2.3. Super Meat Boy

Super Meat Boy es un videojuego de plataformas 2D desarrollado por Team Meat y lanzado en 2010, el cual se diferencia por su alto nivel de dificultad y por la precisión extrema en el control del personaje, consolidándose como un referente moderno del género; su propuesta técnica se centra en un sistema de movimiento rápido y responsivo, acompañado de un diseño de niveles que exige exactitud y repetición constante [26], y esta combinación ha sido ampliamente reconocida por su capacidad de unir la simplicidad mecánica con una complejidad estratégica que desafía al jugador. Este juego se ha convertido en uno de los más influyentes en la evolución de las plataformas 2D gracias a haber dejado una huella significativa en la manera en que se conciben y desarrollan los juegos del género [27].

Ilustración 8
Diseño de un nivel y estilo visual de Super Meat Boy

Nota: Team Meat

Aspectos de diseño destacados:

- Movimiento acelerado y responsivo: el sistema de control está diseñado para ofrecer una respuesta inmediata a las entradas del jugador.
- Curva de dificultad extrema: los niveles incrementan progresivamente la complejidad.
- Feedback visual y sonoro: cada acción del jugador, desde un salto hasta una muerte, genera una respuesta clara y rápida.
- Diseño de repetición y aprendizaje: la estructura de niveles cortos y desafiantes se apoya en un sistema de reinicio inmediato.

Sistema de mecánicas asociadas:

- Control de velocidad y fricción: el movimiento del personaje está programado para transmitir sensación de rapidez y precisión.
- Colisiones y obstáculos: la programación incluye un sistema de detección y respuesta que permite al jugador interactuar con sierras, plataformas móviles y paredes.
- Checkpoints y reinicio instantáneo: el sistema de reinicio está diseñado para restaurar el estado inicial del nivel en milisegundos.
- Balance paramétrico: variables como la velocidad máxima, la altura de salto y la sensibilidad del control son ajustables.

2.2.4. Cuadro comparativo de trabajos relacionados

Después de haber visto los 3 trabajos relacionas con este proyecto se procederá a la comparación de diferentes aspectos con el objetivo de visualizar de mejor manera lo más destacado de dichos juegos.

Tabla 1
Cuadro comparativo de trabajos relacionados

Criterio	Hollow Knight	Celeste	Super Meat Boy
Combate	Ataques cuerpo a cuerpo con un arma, acompañado de habilidades que consumen recursos.	No se centra en el combate, se centra en la precisión del movimiento y superar obstáculos.	Se deben de superar obstáculos a gran velocidad para completar los niveles
Principales mecánicas	Exploración, gestión de recursos, mejoras, etc.	Movimientos precisos, checkpoints y el dash.	Movimiento responsivo y muy acelerado con niveles desafiantes.
Jugabilidad	Combates estratégicos con diferentes habilidades	Jugabilidad centrada en la precisión y la narrativa.	Jugabilidad repetitiva y rápida, siendo su base en la prueba y error.
Habilidades	Ataques básicos con un clavo, habilidades mágicas, etc.	Escalada de paredes e interacción con elementos, además de su dash.	Control de velocidad y fricción además de su salto.
Dificultad	Dificultad progresiva mediante la incorporación de objetos y habilidades.	Sube gradualmente al introducir mecánicas de movimiento.	Dificultad elevada desde el inicio, requiere de muchos intentos.

Nota: Elaboración propia con base en Hollow Knight, Celeste y Super Meat Boy.

A partir del análisis comparativo realizado, el presente proyecto retoma la precisión y el sistema de dash de Celeste, integrándolos con el movimiento acelerado y altamente

responsivo de Super Meat Boy, cuya jugabilidad se apoya en la lógica del ensayo y error y en la superación progresiva de desafíos, además de Hollow Knight se adopta la idea de vincular el consumo de recursos con la ejecución de habilidades.

2.3. Bases tecnológicas

En esta sección se examinan las herramientas y tecnologías vinculadas con la programación y la organización técnica del proyecto, y la revisión se estructura en dos grandes grupos: motores de desarrollo de videojuegos y editores de código, buscando analizar alternativas y características de los diferentes exponentes.

2.3.1. Motores de desarrollo de videojuegos

Los motores de desarrollo de videojuego constituyen la base sobre la que se construyen experiencias interactivas, por eso es importante que no solo brinden un espacio de desarrollo, sino que también dejen un sitio para la creatividad y el diseño.

En esta sección se investigan Unity, Unreal Engine y Godot Engine, considerando aspectos como su arquitectura de trabajo, lenguajes compatibles, capacidades para proyectos 2D y 3D, requisitos técnicos, licencias, documentación y facilidad de aprendizaje. Finalizando se presenta un cuadro comparativo que permite justificar la selección del motor utilizado en el proyecto.

2.3.2. Unity

Unity es un motor gráfico desarrollado por Unity Technologies y ampliamente utilizado en la creación de videojuegos, aplicaciones interactivas y simulaciones tanto en 2D como en 3D, siendo popular por su capacidad de exportar proyectos a múltiples plataformas, convirtiéndolo así en una herramienta versátil y adaptable a distintos entornos de desarrollo.

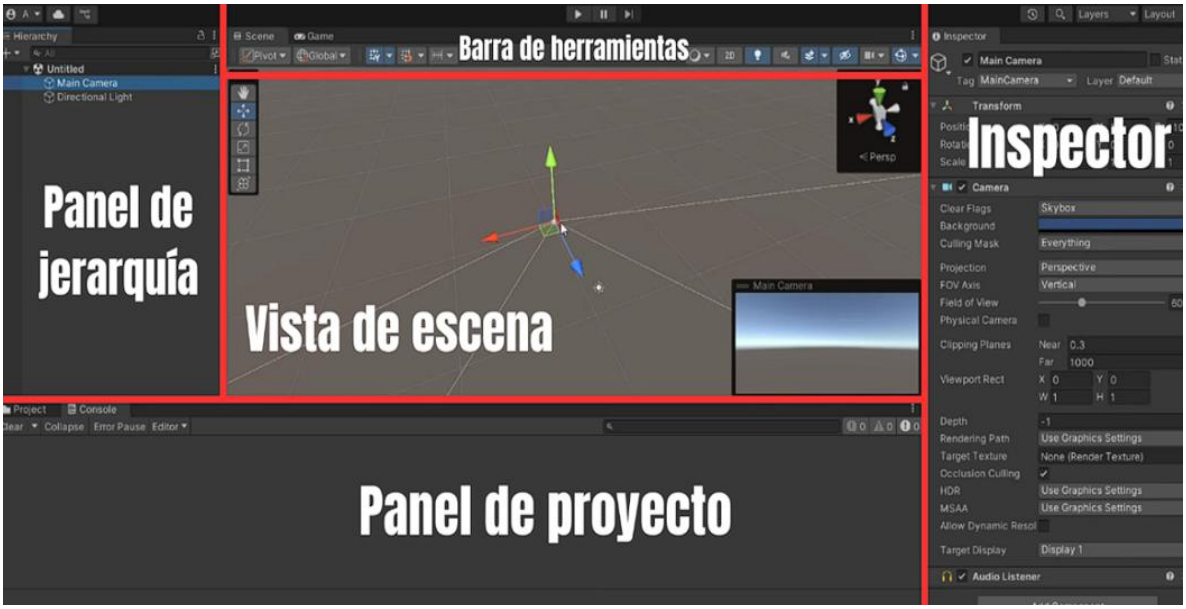
Características principales:

- Lenguaje de programación: al emplear C# como lenguaje principal permite una integración estable y estructurada de scripts [28].
- Assets y recursos: el motor incluye paquetes de assets con modelos, texturas y scripts predefinidos que facilitan la creación de escenas [29].
- Comunidad y soporte: los problemas técnicos son más manejables ya que Unity dispone de una extensa documentación, foros y tutoriales en línea [29].

Unity organiza su entorno de trabajo a través de distintas ventanas que facilitan la construcción y edición de proyectos:

- Panel del proyecto: concentra todos los recursos o Assets disponibles.
- Vista de escena: ofrece una representación visual del espacio en el que se trabaja.
- Panel de jerarquía: muestra la disposición estructural de los objetos presentes en la escena.
- Inspector: despliega las propiedades específicas del objeto seleccionado, otorgando la opción de editarlas y observar sus cambios de manera inmediata.
- Barra de herramientas: reúne accesos directos a funciones y componentes esenciales.

Ilustración 9
Ventanas principales del editor Unity



Nota: Elaboración propia.

Unity se utiliza tanto en el ámbito profesional como académico: en la industria, es reconocido por su papel en el desarrollo de videojuegos indie y comerciales, mientras que en investigación y docencia se emplea para enseñar programación aplicada al diseño de videojuegos y simulaciones [28][30].

2.3.3. Godot

Es un motor de desarrollo de videojuegos de código abierto distribuido bajo la licencia MIT, diseñado para la creación de proyectos tanto en 2D como en 3D [31]; desde su liberación como software libre este ha experimentado un crecimiento sostenido gracias a su naturaleza multiplataforma, la ausencia de costos por licencias o regalías y la posibilidad de exportar proyectos hacia sistemas operativos de escritorio, dispositivos móviles e incluso navegadores web [31][32]. Estas características han favorecido su adopción en proyectos independientes y en entornos educativos y de investigación hasta llegar a su consolidación

como una alternativa relevante frente a motores comerciales y posicionándolo como una herramienta accesible y flexible para distintos perfiles de desarrolladores [33].

Uno de los principales diferenciadores es su modelo de desarrollo abierto, que permite a cualquier usuario acceder al código fuente, modificarlo y contribuir a su evolución bajo los términos de la licencia MIT [31]; este enfoque ha ayudado a la conformación de una comunidad internacional de desarrolladores que participa activamente en la mejora del motor [34]. Todo lo anterior da como resultado que Godot se haya posicionado como una alternativa sólida frente a otros motores, impulsado por un ecosistema colaborativo que ofrece acceso a herramientas profesionales sin restricciones económicas [33][34].

Características principales:

- Multiplataforma: permite exportar proyectos a Windows, Linux, macOS, dispositivos móviles y navegadores web mediante.
- Lenguajes de programación: utiliza principalmente GDScript, inspirado en Python, aunque también soporta C#, C++ y VisualScript.
- Arquitectura basada en nodos: cada elemento del juego se organiza como un nodo dentro de una escena.
- Optimización en 2D: Godot dispone de un motor 2D independiente del 3D, garantizando rendimiento eficiente y herramientas específicas para físicas, animaciones y gestión de sprites.
- Comunidad activa: al ser de código abierto, cuenta con una comunidad que contribuye con tutoriales, módulos, mejoras constantes y documentación extensa.

Ventanas principales de Godot:

- Barra de menús: ubicada en la parte superior de la interfaz, proporciona acceso a las principales opciones de configuración y administración del proyecto, incluyendo herramientas de edición, depuración, configuración del editor y gestión de recursos.
- Espacios de trabajo: permiten alternar entre los diferentes modos de desarrollo del motor, como la edición de escenas en 2D o 3D, la programación mediante scripts y la gestión de recursos.
- Barra de herramientas: reúne accesos rápidos a las funciones más utilizadas durante el desarrollo, como guardar escenas, modificar herramientas de edición y controlar diferentes opciones del editor.
- Botones de ejecución: permiten ejecutar el proyecto completo, la escena actual o iniciar el proceso de depuración.
- Viewport: constituye el espacio principal de trabajo donde se visualizan y editan las escenas del proyecto. En esta área es posible posicionar nodos, modificar objetos y construir visualmente los niveles del videojuego.
- Dock o Panel lateral: agrupa paneles auxiliares del editor que proporcionan herramientas de apoyo para el desarrollo.
- Panel inferior: concentra herramientas complementarias como la consola de salida, el depurador, el administrador de audio y el editor de animaciones.

Ilustración 10
Principales ventanas del editor Godot

Nota: Elaboración propia.

Lo anteriormente descrito muestra que Godot Engine constituye un entorno de desarrollo completo para la creación de videojuegos modernos gracias a su integración en una misma plataforma herramientas orientadas al diseño, la programación, la depuración y la exportación de proyectos [32]; su arquitectura modular, el soporte para múltiples plataformas y el respaldo de una comunidad activa lo convierten en una alternativa adecuada para proyectos académicos y profesionales que requieren flexibilidad y acceso directo al código fuente [31][34], cualidades que han favorecido su creciente presencia en la industria del desarrollo independiente, en donde diversos estudios reconocen el aumento de su adopción en los últimos años [33], justificando su utilización como una opción viable para el desarrollo de videojuegos en diferentes contextos [35].

2.3.4. Cuadro comparativo de motores de videojuegos

Unity y Godot, con el objetivo de contrastar sus fortalezas y particularidades de manera clara y ordenada.

Tabla 2
Cuadro comparativo de motores de videojuegos

Criterio	Unity	Godot Engine
Tipo de licencia	Gratuita hasta alcanzar ciertos ingresos, de ahí se deben adquirir planes de pago.	Código abierto bajo licencia MIT, siendo completamente gratuito.
Arquitectura principal	Basada en escenas y objetos con componentes resultando altamente modular.	Basada en nodos y escenas jerárquicas, resultando modular y escalable.
Lenguajes principales	Soporte C#.	GDScript y C#; permite extensiones mediante C y C++.
Desarrollo 2D	Herramientas profesionales para físicas, animación y UI.	Motor 2D independiente del 3D optimizado para rendimiento de sprites.
Interfaz	Ventana de proyecto, jerarquía, inspector, escenas y barra de herramientas.	Panel de escena, inspector, editor de scripts, recursos y vista 2D y 3D.
Publicación multiplataformas	Exportación directa a consolas, PC, móviles, VR y AR.	Exportación directa a PC, móviles y navegadores.
Ecosistema	Posee una Asset store con muchos recursos y soporte profesional.	La comunidad proporciona una gran cantidad de assets para el motor.
Actividad de comunidad	Muy extensa con soporte y documentación oficial.	Activa y colaborativa con enfoque en proyectos open source.

Nota: Elaboración propia con base en Unity y Godot.

2.3.5. APIs y editores de código

Unity tiene conexión directa con varias APIs y editores de código para que la lógica del juego o aplicación se pueda ejecutar, las más conocidas y usadas son Visual Studio Code, Visual Studio Community y JetBrains Rider, cada una teniendo sus propias ventajas y desventajas, pero compartiendo una correcta integración con Unity.

2.3.6. Visual Studio Code

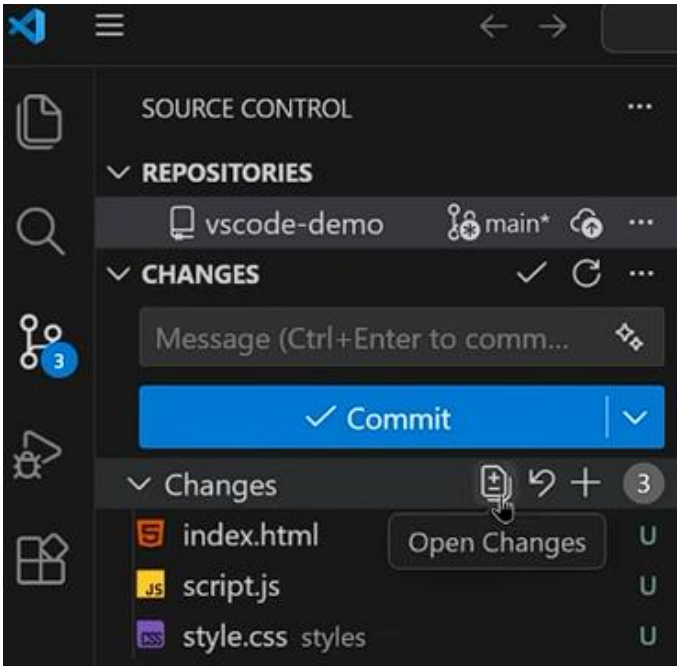
Visual Studio Code (VS Code) se ha consolidado como uno de los editores más utilizados en el desarrollo de software gracias a su arquitectura modular y soporte multiplataforma.

Los autores Tan et al. documentan la implementación de Visual Studio Code en el ámbito de la educación en cursos introductorios de ciencias de la computación y concluyen que su interfaz intuitiva facilita la curva de aprendizaje de estudiantes novatos [36]; este hallazgo resulta relevante porque el editor permite que los alumnos desarrollen hábitos de programación más consistentes y profundicen en la comprensión de conceptos básicos para que de esta manera se reduzca la carga cognitiva en ellos.

Los autores Matthew et al. presentan un marco de visualización en tiempo real integrado en Visual Studio Code, el cual emplea Tree-sitter y WebGPU para representar dinámicamente estructuras de código como funciones, bucles y condicionales [37], colocando de relieve el potencial del editor como herramienta pedagógica, ya que la visualización directa dentro del entorno favorece la comprensión conceptual y fortalece la retención de conocimientos.

Entre las características más relevantes de Visual Studio Code se encuentra su arquitectura extensible la cual posibilita la integración de funcionalidades avanzadas mediante complementos. Tan et al. destacan que sus principales virtudes son: su interfaz ligera y multiplataforma, herramientas como el autocompletado, la depuración integrada y la compatibilidad con Git [36].

Ilustración 11
Integración de Git en VSCode



Nota: Visual Studio Code

Visual Studio Code se presenta como un entorno que logra equilibrar ligereza, flexibilidad y potencia, cualidades que explican su adopción masiva tanto en la industria del software como en la enseñanza de programación. De esto hablan Matthew et al. demuestra que su integración con herramientas de visualización en tiempo real potencia la comprensión conceptual de los estudiantes y refuerza su papel como recurso pedagógico [37]; al mismo tiempo su capacidad de personalización y el aguante de una comunidad activa aseguran un flujo constante de mejoras y extensiones que sostienen su relevancia a largo plazo.

2.3.7. Visual Studio Community

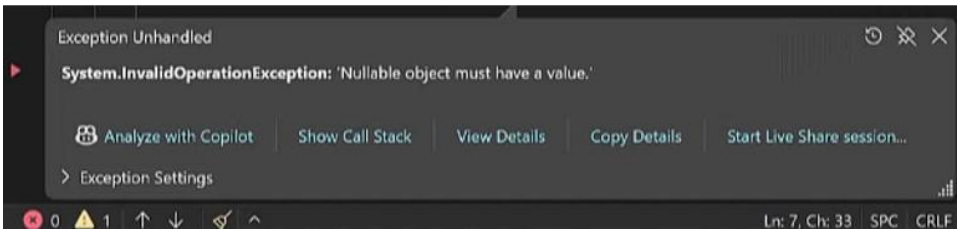
Es un entorno de desarrollo integrado (IDE) gratuito creado por Microsoft y orientado a estudiantes, desarrolladores individuales, instituciones educativas y proyectos de código abierto el cual ofrece la mayoría de las funcionalidades presentes en las versiones Professional y Enterprise, siempre que se trate de escenarios de desarrollo no empresariales,

además de incorporar un sistema de instalación basado en cargas de trabajo, mediante el cual el usuario puede seleccionar únicamente los componentes necesarios para cada tipo de proyecto [38][39][40].

Este IDE reúne un conjunto amplio de herramientas diseñadas para incrementar la productividad durante el desarrollo de software, entre las que destacan IntelliSense por ofrecer autocompletado inteligente de código y un depurador avanzado que permite trabajar con puntos de interrupción y seguimiento de variables, sumándose la compilación integrada, las pruebas unitarias, la administración de soluciones mediante Solution Explorer, la integración nativa con Git y la compatibilidad con miles de extensiones disponibles en Visual Studio Marketplace. Debido a esto es posible gestionar todas las etapas del ciclo de desarrollo dentro de un mismo entorno, desde la escritura del código hasta la compilación. [38][39][40].

Otra de las principales ventajas de Visual Studio Community es su compatibilidad con múltiples lenguajes y plataformas de desarrollo debido a que el entorno ofrece soporte para C#, C++, Visual Basic, F#, Python, JavaScript y TypeScript además de facilitar la creación de aplicaciones mediante .NET, ASP.NET, .NET MAUI, Azure y Unity. En el caso particular del desarrollo de videojuegos se tiene presente que la integración con Unity resulta especialmente valiosa porque permite depurar directamente los scripts en C#, explorar los recursos del proyecto y aprovechar herramientas específicas que optimizan el flujo de trabajo. [38][39][40][41].

Ilustración 12
Opciones avanzadas al depurar en Visual Studio Community



Nota: Visual Studio Code

La utilización de entornos de desarrollo integrados profesionales ha demostrado tener un impacto positivo en los procesos de enseñanza y aprendizaje de la programación, pues Birillo et al. plantean trasladar la formación desde editores básicos hacia IDEs completos y argumentan que, al trabajar con herramientas que integran edición, compilación, depuración y administración de proyectos en un mismo entorno, los estudiantes desarrollan competencias más cercanas a las que demanda el ámbito laboral, evidenciando que el uso de IDEs favorece la adquisición de habilidades prácticas vinculadas con el desarrollo de software moderno y contribuye a reducir la brecha existente entre la formación académica y las necesidades de la industria [42].

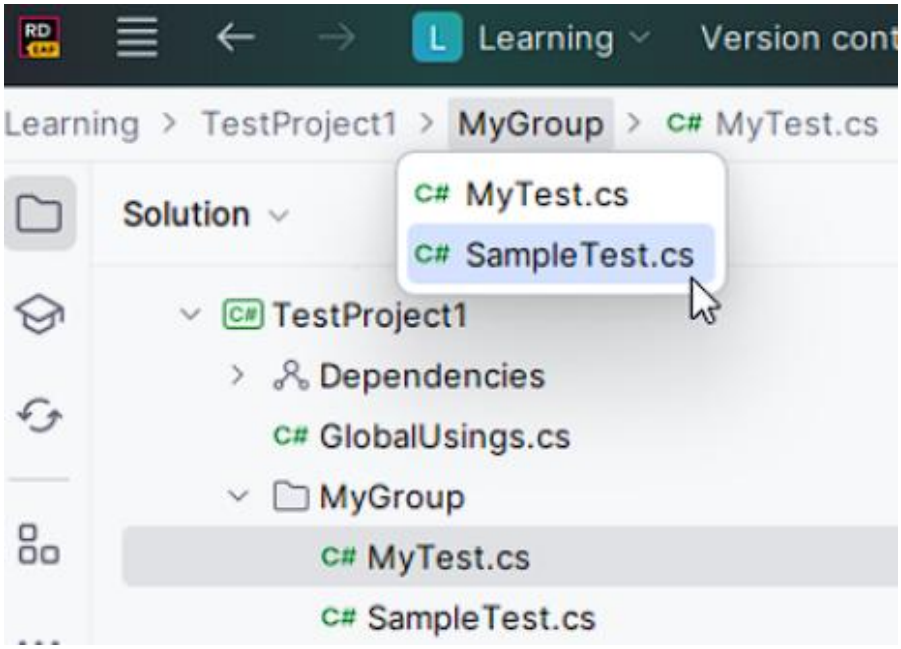
Asimismo, diversos estudios han señalado que las herramientas avanzadas de depuración cumplen un papel esencial en el aprendizaje de la programación, ya que permiten a los estudiantes enfrentarse de manera más estructurada a los problemas que surgen durante el desarrollo. En esta línea, Lin et al. demostraron que quienes aplican estrategias sistemáticas de depuración logran comprender mejor el comportamiento de sus programas y, al mismo tiempo, identifican errores con mayor eficiencia, lo que se traduce en un mejor desempeño en las actividades de programación [43].

2.3.8. JetBrains Rider

JetBrains Rider es un entorno de desarrollo integrado (IDE) creado por JetBrains para aplicaciones basadas en .NET y videojuegos desarrollados con Unity, y se distingue por combinar la plataforma IntelliJ con el motor de análisis de código de ReSharper, lo que le permite ofrecer funciones avanzadas de autocompletado, navegación inteligente, refactorización, inspección estática de código y depuración, sumándole su compatibilidad con Windows, macOS y Linux, que amplía las posibilidades de trabajo al permitir el desarrollo de aplicaciones de escritorio, web, móviles y videojuegos dentro de un mismo entorno [44][45][46].

Una de las principales fortalezas de JetBrains Rider es su integración nativa con Unity debido a que el IDE incorpora herramientas específicas para este motor gráfico que van desde inspecciones de código orientadas a la API de Unity y la generación automática de funciones de eventos hasta la navegación entre escenas, prefabs y recursos. A ello se añaden el análisis de rendimiento en métodos críticos, la depuración de scripts en C#, la ejecución de pruebas unitarias y la visualización de registros del motor directamente dentro del entorno de desarrollo, siendo por todo lo anterior que los desarrolladores pueden reducir errores comunes durante la programación y optimizar su flujo de trabajo, especialmente en proyectos de mediana y gran escala, donde la eficiencia y la organización resultan determinantes [44][45][46].

Ilustración 13
Interfaz y navegación entre diferentes scripts



Nota: JetBrains Rider.

El uso de IDEs, en el ámbito académico, que se basan en la plataforma JetBrains ha demostrado aportar beneficios significativos en la formación de desarrolladores, pues Birillo et al. plantean trasladar el aprendizaje de la programación hacia entornos profesionales mediante cursos integrados directamente en estos IDEs, lo que permite que los estudiantes realicen sus actividades con las mismas herramientas que se emplean en la industria permitiendo que el proceso formativo no solo incorpore teoría y práctica, sino que también integre recursos industriales en un único entorno, favoreciendo la adquisición de competencias técnicas y reduciendo la distancia entre la preparación académica y las exigencias del desarrollo profesional [42]. Lin et al. demostraron que quienes aplican estrategias sistemáticas de depuración logran comprender mejor el comportamiento de sus programas y, al mismo tiempo, detectar errores con mayor eficiencia, lo que se traduce en un desempeño más sólido en las actividades de programación [43], siendo en este contexto que las capacidades de análisis estático, las inspecciones inteligentes, la refactorización

automática y la depuración integrada en JetBrains Rider se convierten en recursos que favorecen tanto la formación académica como el desarrollo profesional de aplicaciones, especialmente en proyectos realizados con Unity y tecnologías .NET [43][44][45][46].

2.3.9. Cuadro comparativo de APIs y editores de código

A continuación, se presenta un cuadro comparativo de los diferentes entornos para la programación y escritura de código.

Tabla 3
Cuadro comparativo de APIs y editores de código.

Criterio	Visual Studio Code	Visual Studio Community	JetBrains Rider
Licencia	Gratuito y de código abierto con soporte oficial de Microsoft y una comunidad activa que desarrolla extensiones.	Gratuito en su versión Community pero con licencia para uso académico.	De pago con licencia anual o mensual. Ofrece periodos de prueba personales.
Lenguajes	Multilenguaje por extensiones, haciéndolo muy flexible.	Soporta de forma nativa c#, c++, phyton, entre otros. Permite integración directa con el ecosistema Microsoft.	Especializado en c# y .NET, aunque soporta otros lenguajes gracias a IntelliJ.
Extensibilidad	Altamente extensible por plugins, además su estructura modular permite personalizar el entorno.	Menos extensible que VS code pero es debido a que ya trae varios componentes y herramientas avanzadas de forma nativa.	Integra ReSharpe y el ecosistema JetBrains, con extensiones centradas en productividad.
Depuración y análisis	Depuración básica, dependiendo de extensiones para análisis avanzado.	Depuración avanzada incluida desde el primer momento, además de perfiles de rendimiento y herramientas de análisis integradas.	Análisis de código avanzado, refactorización automática y sugerencias personalizadas.
Integración con la nube y control de versiones	Compatible con Git y servicios en la nube por medio de extensiones.	Integra directamente a Azure DevOps, Github, etc.	Software integrado con Git y sistemas de control de versiones, sin embargo, es menos orientado a la nube.
Orientación	Ideal para estudiantes y proyectos pequeños.	Orientado a proyectos enormes y complejos, siendo ampliamente utilizado en empresas.	Más orientado a profesionales y equipos especializados en c# y .NET.

Nota: Visual Studio Code

2.4. Bases metodológicas

Se analizarán tres metodologías ágiles ampliamente utilizadas en el desarrollo de software: Scrum, Kanban y Extreme Programming (XP) para reconocer sus principales características y comprender tanto sus ventajas como sus limitaciones en distintos contextos de aplicación.

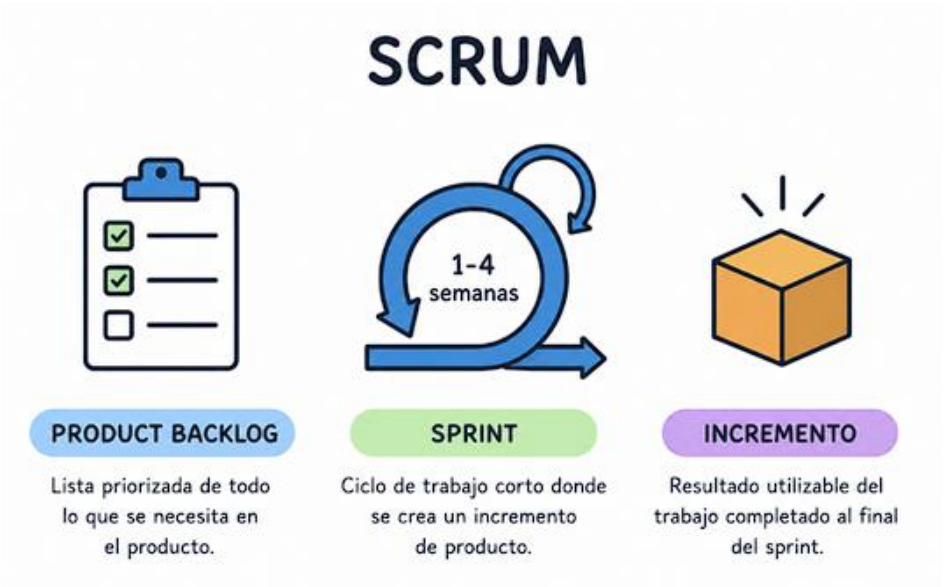
2.4.1. Scrum

Scrum se ha consolidado como el marco ágil más utilizado en la industria del software por a su capacidad para gestionar proyectos complejos mediante iteraciones cortas y entregas incrementales.

Un estudio sistemático de la literatura destaca que Scrum permite mejorar la colaboración entre equipos, incrementar la eficiencia en la entrega y fomentar la adaptación continua frente a la incertidumbre del entorno de desarrollo [47], combinado con su estructura basada en roles definidos, artefactos y eventos regulares asegura transparencia y facilita la inspección y adaptación, principios fundamentales para enfrentar los retos de proyectos dinámicos [48].

Investigaciones recientes han demostrado que factores como la autonomía del equipo, la mejora continua y el apoyo de la gestión son determinantes para el éxito de los proyectos bajo este marco [49], sin embargo, Scrum no se libra de riesgos, ya que se ha evidenciado que su implementación enfrenta desafíos en contextos como el trabajo remoto, donde la falta de interacción física puede afectar la calidad de las ceremonias y la coordinación entre miembros [48].

Ilustración 14
Los tres artefactos de la metodología Scrum



Nota: elaboración propia

Características:

- Iteraciones cortas llamadas sprints: el trabajo se organiza en ciclos de 2 a 4 semanas.
- Roles definidos: se establecen responsabilidades claras entre Product Owner, Scrum Master y Equipo de Desarrollo.
- Eventos estructurados: se realizan ceremonias como la planificación, reuniones diarias, revisión y retrospectiva.
- Entregas incrementales: al final de cada sprint se entrega un incremento funcional del producto.
- Transparencia en el progreso: mediante tableros y artefactos se muestra el estado del proyecto.

2.4.2. Kanban

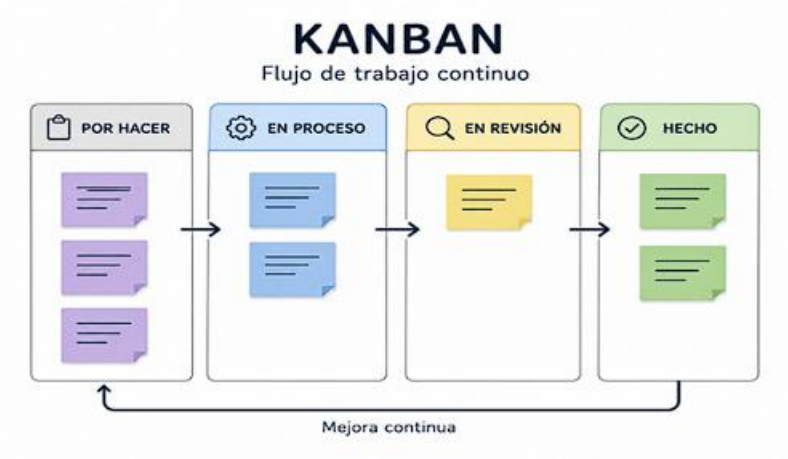
Es una metodología ágil de gestión del trabajo basada en los principios de Lean, cuyo objetivo principal es optimizar el flujo de trabajo mediante la visualización de las tareas, la

limitación del trabajo en progres y la mejora continua de los procesos, diferenciándose de otras metodologías ágiles que trabajan mediante iteraciones de duración fija ya que Kanban promueve un flujo continuo de desarrollo permitiendo que las tareas avancen conforme existe capacidad disponible dentro del equipo al facilitar la adaptación a cambios en los requisitos del proyecto y favorecer una entrega constante de valor al cliente [50][51][52].

Diversas investigaciones han demostrado que la adopción de Kanban aporta beneficios significativos en proyectos de desarrollo de software: Ahmad et al. realizaron una revisión sistemática de la literatura en la que identificaron mejoras relacionadas con la reducción del tiempo de entrega, el incremento de la calidad del software, una mejor comunicación entre los integrantes del equipo y una disminución de los defectos reportados por los clientes [51]. Posteriormente se publicó un estudio de mapeo sistemático donde analizaron la evidencia científica disponible sobre Kanban, concluyendo que la metodología favorece la entrega continua de software, incrementa la visibilidad del proceso de desarrollo y facilita la identificación temprana de cuellos de botella [50].

Oza, Fagerholm y Münch evidenciaron que la utilización de tableros Kanban favorece la coordinación entre los miembros del equipo, mejora la identificación colectiva de tareas pendientes y facilita la visualización del estado del proyecto especialmente durante las primeras etapas de implementación de la metodología [52]. Estos resultados respaldan el uso de Kanban como una metodología adecuada para proyectos de software que requieren flexibilidad, adaptación continua y una gestión eficiente de las actividades de desarrollo [50][51][52].

Ilustración 15
Flujo de trabajo de la metodología Kanban



Nota: elaboración propia.

Características:

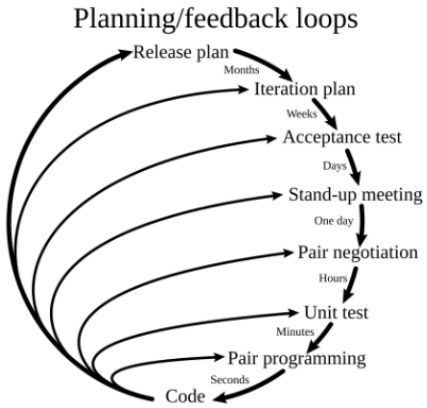
- Visualización del flujo de trabajo: se utilizan tableros y tarjetas para representar las tareas.
- Flujo continuo de tareas: en lugar de trabajar en iteraciones fijas, las tareas avanzan de manera constante según la capacidad del equipo.
- Límites de trabajo en progreso (WIP): se establecen restricciones en la cantidad de tareas simultáneas para evitar sobrecarga.
- Identificación temprana de cuellos de botella: el monitoreo constante del flujo.
- Flexibilidad ante cambios: el sistema se adapta fácilmente a requerimientos nuevos o imprevistos.

2.4.3. Extreme Programming (XP)

Extreme Programming es una metodología ágil que pone énfasis en prácticas técnicas como la programación en parejas, la integración continua y la refactorización frecuente, teniendo como objetivo principal es mejorar la calidad del software y responder rápidamente a cambios en los requisitos, manteniendo ciclos de desarrollo cortos y retroalimentación constante entre clientes y desarrolladores [53][54].

La aplicación de XP ha mostrado beneficios en productividad y satisfacción del cliente, pero también enfrenta retos culturales como puede ser la resistencia a la programación en parejas o la presencia constante del cliente. Estudios señalan que su efectividad depende de la disciplina técnica y del compromiso con la mejora continua [54].

Figura 4
Ciclos de planificación en XP



Nota: Wells, CC BY-SA 3.0.

Características:

- Programación en parejas: consiste en que dos desarrolladores trabajan juntos en una misma estación, uno escribe el código mientras el otro revisa en tiempo real, fomentando la calidad y el aprendizaje compartido.
- Integración continua: se realizan compilaciones y pruebas frecuentes para detectar errores rápidamente, asegurando que el sistema esté siempre en un estado funcional y estable.
- Refactorización frecuente: el código se mejora de manera constante sin alterar su funcionalidad, manteniendo la simplicidad.
- Desarrollo guiado por pruebas (TDD): antes de escribir el código se crean pruebas unitarias garantizando que cada funcionalidad cumpla con los requisitos y se reduzcan defectos.
- Retroalimentación constante del cliente: el cliente participa activamente durante el proceso debido a que proporciona comentarios que permiten ajustar el producto a sus necesidades reales.

2.4.4. Cuadro comparativo de metodologías de desarrollo

Las metodologías ágiles han transformado la forma en que se desarrollan proyectos de software, ofreciendo enfoques flexibles y colaborativos que permiten responder de manera eficiente a los cambios y garantizar entregas continuas de valor siendo las más representativas Scrum, Kanban y Extreme Programming (XP), cada una con características particulares que abarcan desde la gestión de roles y eventos, hasta la optimización del flujo de trabajo y la calidad técnica del código.

El análisis comparativo de estas metodologías resulta fundamental para identificar sus similitudes y diferencias, así como para determinar cuál se adapta mejor a las necesidades

de un proyecto específico. El siguiente cuadro comparativo nos da una visión más cercana de estas 3 metodologías:

Tabla 4

Tabla comparativa de metodologías de desarrollo

Criterio	Scrum	Kanban	XP
Sistema	Iteraciones cortas llamadas sprints.	Flujo continuo y sin mayores interrupciones.	Iteraciones muy cortas.
Roles	Product owner, Scrum Master, Equipo de desarrollo.	No define roles específicos.	Cliente, programadores, Coach.
Eventos	Planificación, reunión diaria, retrospectiva.	No existen eventos fijos.	Reuniones frecuentes con clientes.
Gestión de tareas	Backlog priorizado.	Tablero visual con WIP.	Historia de usuario y pruebas.
Enfoque	Gestión de proyectos, sobre todo de software.	Flujo y eficiencia a lo largo del desarrollo.	Calidad técnica del código.

Nota: Elaboración propia.

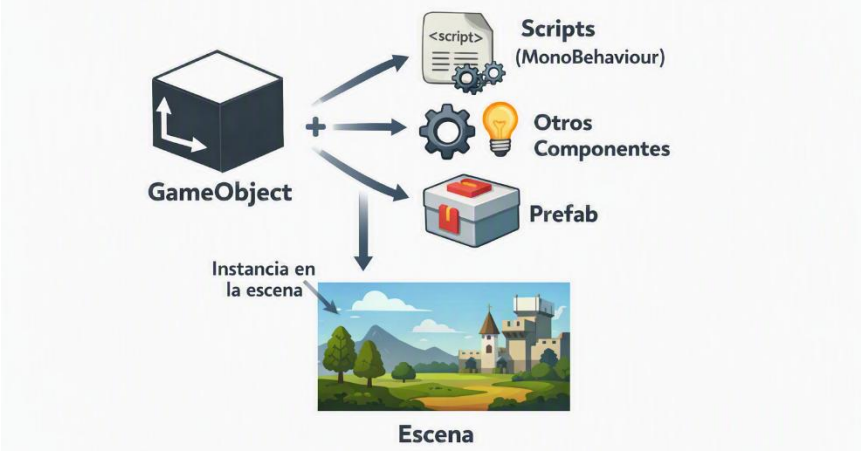
Tomando como referencia las metodologías ágiles analizadas evidencia que cada una aporta beneficios particulares al desarrollo de proyectos. Kanban destaca por su flexibilidad y control del flujo de trabajo, mientras que XP asegura calidad técnica mediante prácticas disciplinadas. Sin embargo, para el contexto del presente proyecto se considera más pertinente la adopción de Scrum, ya que su estructura basada en roles definidos, eventos organizados y entregas incrementales favorece la transparencia, la colaboración y la adaptación continua, siendo características que convierten a Scrum en el marco metodológico más adecuado para garantizar un proceso de desarrollo ordenado, eficiente y capaz de responder a cambios sin perder control ni coherencia en la gestión del equipo.

2.5. Arquitecturas, modelos y estándares

2.5.1. Patrón clásico de Unity

El patrón clásico de Unity se fundamenta en la composición de GameObjects y Componentes mediante scripts derivados de MonoBehaviour, junto con el uso de Prefabs para la reutilización de objetos, constituyendo la arquitectura nativa del motor, pues organiza la lógica y la presentación directamente en la escena, sin necesidad de modelos externos como MVC o MVP, además, por el tamaño del proyecto se ha optado por esta arquitectura. Diversos estudios han señalado que la utilización de componentes y Prefabs incrementa la reutilización y acelera el desarrollo de videojuegos, consolidando este patrón como la práctica más extendida en Unity [55].

Figura 5
Patrón clásico de Unity



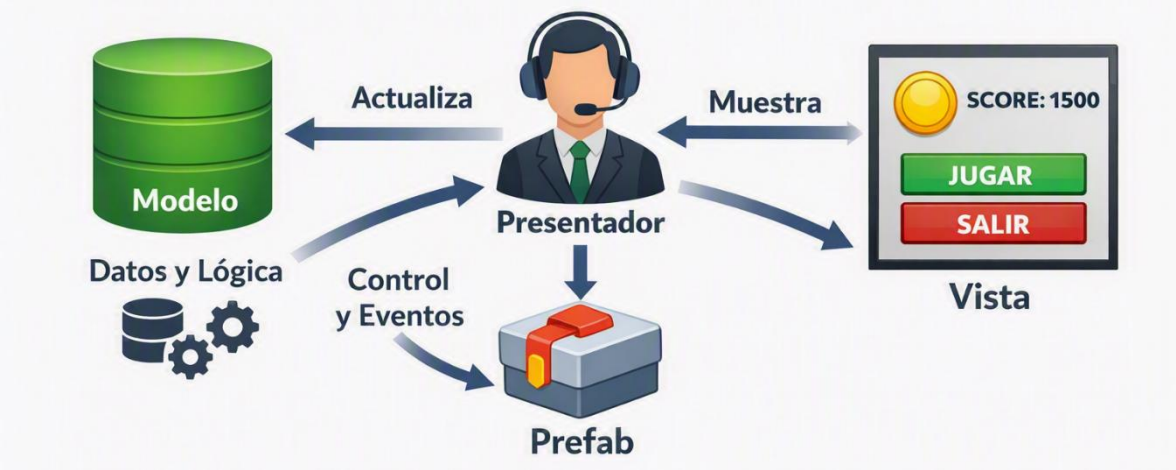
Nota: Elaboración propia.

2.5.2. Modelo MVP

El patrón Model–View–Presenter (MVP) es una arquitectura que busca separar la lógica de negocio de la interfaz de usuario, facilitando la mantenibilidad y la capacidad de prueba del código de la siguiente: Model gestiona los datos del juego, la View representa los elementos gráficos y la Presenter actúa como intermediario, recibiendo eventos de la View,

procesando la lógica en el Model y actualizando la interfaz. Esta separación permite que la lógica pueda probarse de manera independiente, sin necesidad de cargar escenas completas, lo que incrementa la eficiencia en proyectos con interfaces complejas. Investigaciones recientes han demostrado que aplicar patrones como MVP en Unity mejora la reutilización de componentes y acelera el desarrollo de videojuegos [55].

Figura 6
Lógica del modelo MVP



Nota: Elaboración propia.

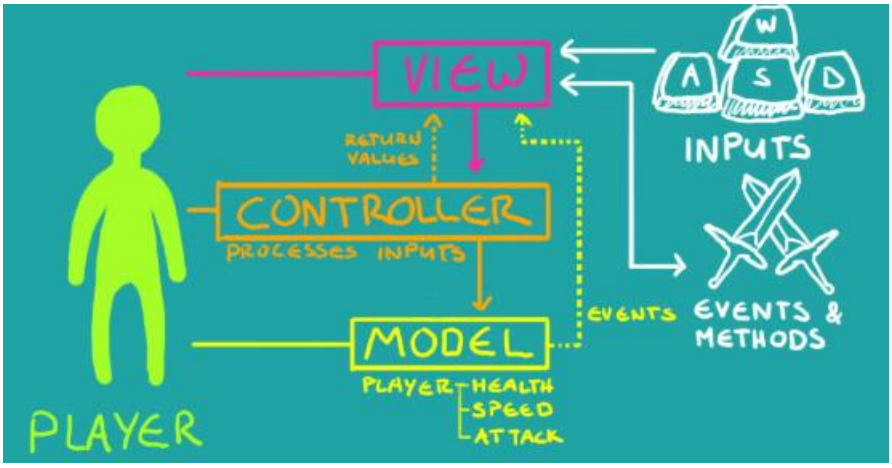
2.5.3. Modelo MVC

El patrón Model–View–Controller (MVC) es una arquitectura de software que separa la lógica de negocio, la interfaz de usuario y el control de la interacción permitiéndose aplicarse para organizar proyectos con interfaces complejas o sistemas interactivos mediante la gestión de los datos y reglas del juego, la View representa la interfaz visual (HUD, menús, elementos gráficos) y el Controller actúa como mediador, procesando las entradas del usuario y actualizando tanto el Model como la View.

Este enfoque permite mantener una clara separación de responsabilidades, lo que facilita la escalabilidad y el mantenimiento del proyecto, especialmente en entornos de

desarrollo de videojuegos, se ha demostrado que aplicar arquitecturas como MVC mejora la modularidad y la capacidad de prueba del código [56].

Figura 7
Lógica del modelo MVC



Nota: Stardust.

2.6. Herramientas de desarrollo

2.6.1. Justificación de uso de Visual Studio Code

Los tres entornos de desarrollo presentados anteriormente responden a enfoques distintos que se reflejan en sus características y usos: Visual Studio Code se distingue por su ligereza y extensibilidad, lo que permite personalizar el entorno mediante una amplia variedad de extensiones; Visual Studio Community ofrece un ecosistema más completo, con herramientas avanzadas de depuración y soporte nativo para múltiples lenguajes y, por su parte, JetBrains Rider se especializa en C# y .NET.

En este proyecto, Visual Studio Code resulta más adecuado porque combina simplicidad con potencia técnica y permitiendo configurar únicamente las herramientas necesarias evitando la sobrecarga de funciones que no aportan valor en un contexto académico.

2.6.2. Justificación de uso de Unity Engine

La elección final y elegida ha sido Unity 2D principalmente por el ecosistema consolidado que ofrece a través de su Asset Store y por su facilidad al momento de la integración de recursos y complementos en cualquier etapa del proyecto; sin olvidar el uso de C# el cual es un lenguaje ampliamente reconocido por su estabilidad y mantenibilidad en proyectos de gran escala. A ello se suma su soporte multiplataforma que incluye consolas y dispositivos de realidad virtual junto con herramientas profesionales integradas para 2D capaces de garantizar un flujo de trabajo eficiente y adaptable a las necesidades específicas del proyecto.

2.7. Marco conceptual

En este apartado se presentan los principales conceptos relacionados con el desarrollo del proyecto, sirviendo como base para comprender la propuesta tecnológica planteada.

Tabla 5

Tabla de conceptos

Término	Definición
Coyote Time	Breve intervalo que permite al jugador saltar después de abandonar una plataforma.
Jump Buffer	Técnica que registra una orden de salto antes de que pueda ejecutarse y la activa después de tocar el suelo.
Vertical Slice	Versión funcional del videojuego que integra las principales mecánicas, elementos visuales y de audio.
Sprint	Periodo de tiempo definido en Scrum durante el cual se planifican, desarrollan y completan un conjunto de tareas o funcionalidades.
Metroidvania	Género de videojuegos caracterizado por la exploración de mapas interconectados, donde el jugador obtiene nuevas habilidades que permiten acceder a zonas previamente inaccesibles.
Tilemaps	Sistema que permite crear escenarios 2D mediante una cuadrícula de pequeños bloques gráficos llamados tiles.
Rebind	Función que permite reasignar los controles o teclas de un videojuego sin modificar el código.
Object pool	Técnica que reutiliza objetos ya creados para mejorar el rendimiento y reducir instanciaciones.

Prefabs	Objeto reutilizable que funciona como una plantilla maestra para crear copias idénticas en tus escenas.
Feedback	Respuesta visual, sonora o táctil que informa al jugador sobre una acción o evento del juego.
FPS	Medida que indica la cantidad de imágenes que se muestran por segundo durante la ejecución del juego.
Inputs	Entradas del usuario, como teclas, botones o movimientos, que permiten interactuar con el videojuego.
Singleton	Patrón de diseño que garantiza la existencia de una única instancia de una clase durante la ejecución del programa. |

Nota: Elaboración propia.

Capitulo III

3. Manual de usuario

3.1. Presentación y perfil del usuario

El producto corresponde a un videojuego 2D de plataformas desarrollado en Unity 6 con renderizado URP el cual fue diseñado para ejecutarse en computadoras. El programa requiere especificaciones técnicas básicas: procesador de gama media, al menos 4 GB de memoria RAM y tarjeta gráfica integrada o dedicada que permita un rendimiento estable en entornos de desarrollo y ejecución de videojuegos. La instalación del programa se realiza de manera convencional mediante archivo ejecutable y no requiere de mayor conocimiento sobre el tema.

El recorrido general por el programa se centra en la interacción directa con el videojuego, el usuario explora escenarios y supera obstáculos mientras pone a prueba la mecánica principal: la barra de vida como recurso activo. Todo el sistema está diseñado y centrado en una tarea, ofrecer una jugabilidad clara y accesible con controles intuitivos que permiten al usuario comprender rápidamente las dinámicas propuestas, siendo realizado con el fin de que el prototipo demuestre cómo una mecánica diferenciada puede influir en la percepción de dificultad y en el nivel de compromiso del jugador.

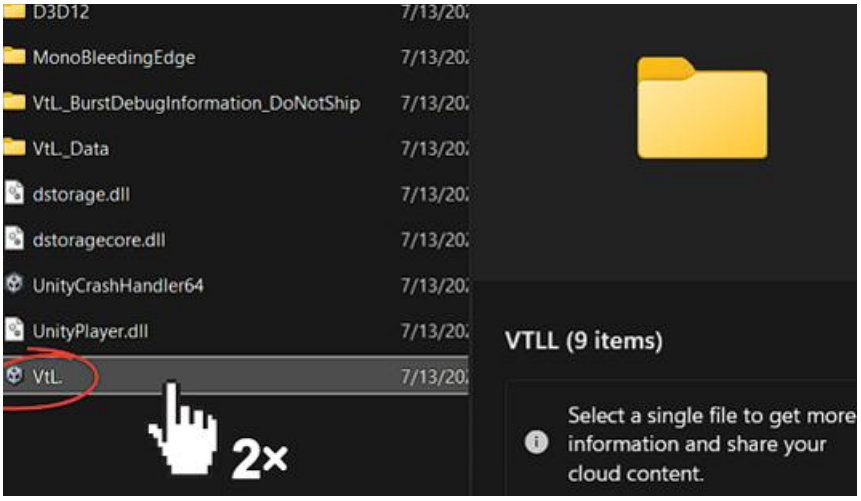
El perfil de los usuarios son jóvenes que busquen experiencias frenéticas y de precisión, que quieran arriesgar todo el recorrido para romper sus propios récords; además el manual de usuario cumple una doble función: servir como guía práctica para quienes ejecutan el programa y como documento de referencia para quienes estudian su aporte dentro de un contexto académico.

3.2. Acceso e inicio

Para ingresar a la experiencia el usuario debe ubicar el archivo ejecutable del programa en su equipo de cómputo, el cual una vez localizado basta con hacer doble clic sobre el archivo para que el sistema inicie automáticamente la carga del videojuego, siendo un proceso rápido e inmediato y no requiere configuraciones adicionales.

Ilustración 16

Ingreso y ejecución del programa desde el ejecutable.



Nota: elaboración propia.

La aplicación despliega la pantalla inicial del videojuego al momento de abrirse, mostrando las opciones básicas de navegación disponibles para que el usuario pueda experimentar: puede seleccionar iniciar juego para acceder directamente a la selección de nivel o explorar otras secciones como ajustes o los créditos.

Al ser un programa de fácil acceso el usuario puede ejecutar el programa sin necesidad de conocimientos avanzados ya que el acceso se limita a una acción simple y familiar dentro del entorno Windows.

3.3. Navegación y controles

Esta se realiza de manera sencilla e intuitiva ya que se usa el teclado y el ratón como principales medios de interacción dando paso al resto del sistema, el cual está diseñado para que el jugador pueda desplazarse y ejecutar acciones básicas sin necesidad de configuraciones avanzadas garantizando una fluides para los usuarios tanto experimentados como para quienes se inician en el género de plataformas.

Ilustración 17

Inputs y acciones del videojuego VtL.

Nota: elaboración propia.

- Movimiento principal: el personaje se controla mediante las teclas W, A, S, D, que permiten desplazarse en las cuatro direcciones básicas.
- Dash: Impulso hacia la dirección en la que mira o hacia la dirección en la que está apuntando con las teclas de movimiento. Se ejecuta el dash con la tecla shift.
- Salto: la tecla Espacio permite realizar saltos, estos se reinician al aterrizar sobre una superficie o suelo, además se usa para escalar paredes.

- Selección: el clic izquierdo del ratón se emplea para interactuar con elementos del menú o confirmar opciones dentro de la interfaz.
- Pausa: Se detiene la partida o se pausa el juego en cualquier momento, congelando el tiempo hasta que el usuario con la misma tecla reanude la partida. Se ejecuta con la tecla espacio

El programa ofrece la posibilidad de cambiar o reasignar las teclas desde el menú de ajustes mediante la opción de rebind de controles asegurando un esquema con una jugabilidad clara y flexible, simplemente centrándose en la mecánica principal del prototipo, esto viene acompañado con la disposición de teclas, la cual responde a controles ampliamente utilizados en videojuegos con el fin de que la adaptación del jugador se buena y que reduzca posibles dificultades en la comprensión de la funcionalidad del sistema.

3.4. Funciones e interacciones disponibles

El usuario cuando ingresa al programa puede acceder al menú principal, el cual presenta cuatro opciones iniciales: Jugar, Tutorial, Opciones y Créditos.

En la opción Jugar, se despliega un submenú que permite seleccionar el nivel específico al que se desea acceder.

La opción Tutorial dirige directamente a un nivel introductorio para que el usuario se familiarice con las mecánicas y los controles.

En Opciones, el usuario puede configurar tres apartados principales:

- Sonido: ajuste del volumen general, música y efectos de sonido.
- Imagen: modificación de la resolución del programa y activación de modos de accesibilidad para daltonismo.

- Controles: posibilidad de realizar rebind de teclas de los movimientos básicos y habilidades.

Finalmente, en Créditos, se muestra un texto en desplazamiento automático que presenta la información de los desarrolladores y colaboradores del proyecto.

Ilustración 18

Menú principal del videojuego VtL.

Nota: elaboración propia.

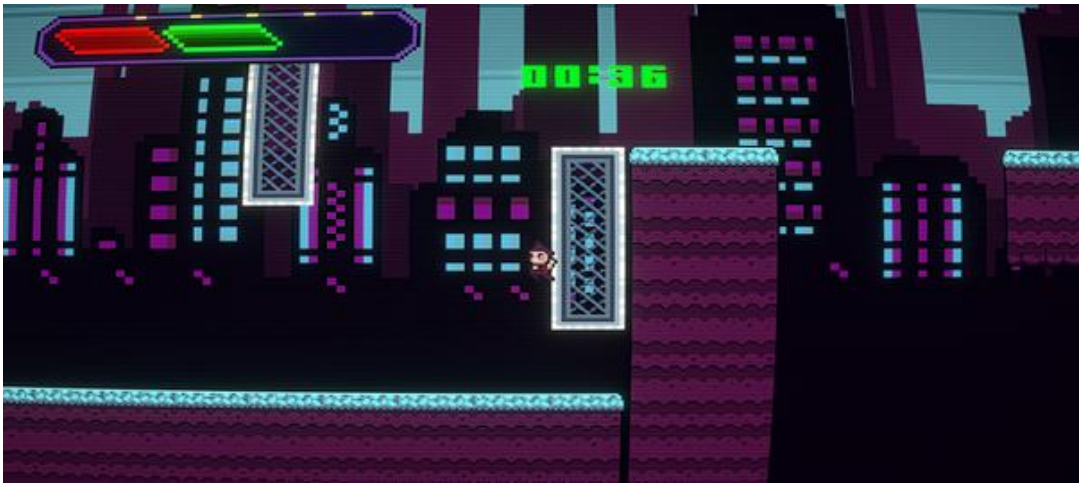
Al seleccionar un nivel se presenta un recorrido de cámara que inicia en la meta, pasa por cada checkpoint y finaliza en el personaje, seguido de una cuenta regresiva de tres segundos; al llegar a cero el nivel inicia y se arranca un cronómetro regresivo de un minuto.

El sistema con el que se desplaza el usuario es con las teclas W, A, S, D, saltar con Espacio y ejecutar dash en la dirección apuntada por el movimiento, sin embargo, este último cumple una doble función: permite atacar enemigos en su punto débil y movilizarse con mayor velocidad, permitiendo a su vez escalar paredes más rápidamente a cambio de perder una barra de vida.

El jugador interactúa a lo largo del recorrido con plataformas para realizar wall jumping y enemigos con distintos patrones de ataque que se activan al entrar en su área de detección, los cuales disparan un proyectil, este reduce una barra de vida del jugador si colisiona con este. Tomar el riesgo de eliminar a un enemigo mediante dash otorga 10 segundos adicionales al cronómetro, además el jugador cuenta con una cantidad de vidas ilimitada, por lo que al morir simplemente aparecerá en el último punto guardado, y para finalizar el nivel se debe de llegar a la meta, momento exacto donde se detiene el tiempo y se convierte en puntuación que se revela en la pantalla.

Ilustración 19

Mecánica de enganche de pared y escalado en el videojuego VtL.



Nota: elaboración propia.

Capitulo IV

4. Manual del programador

4.1. Arquitectura

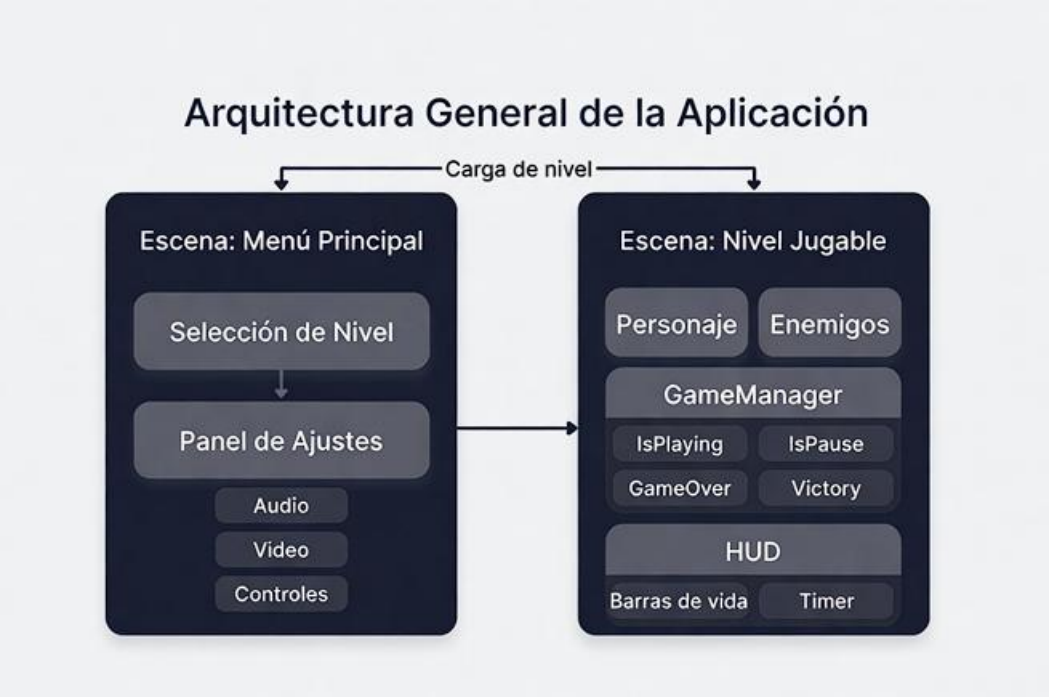
La organización de la arquitectura está basada en 4 componentes principales: el Player, encargado de gestionar las acciones y estados del personaje; el Enemy System, que define el comportamiento y las interacciones de los adversarios; el Controlador central o Game Manager quien se encarga de coordinar la lógica global, cronómetros y condiciones de victoria o derrota; y finalmente el menú principal junto con los managers globales, que administran la navegación, configuraciones y persistencia de datos.

4.1.1. Arquitectura general de la aplicación

Esta arquitectura se fundamenta en un enfoque basado en escenas, el cual permite separar de manera clara y ordenada las responsabilidades de la interfaz de usuario y de la lógica de juego. A continuación, se presenta la estructura general de la aplicación, centrada en sus dos escenas principales:

Figura 8

Relación y arquitectura general del videojuego VtL.



Nota: elaboración propia.

Como se observa en la figura 8, la escena del Menú Principal concentra las funcionalidades relacionadas con la navegación y la configuración del sistema, incluyendo la selección de nivel y el panel de ajustes los cuales agrupan las opciones de Audio, Video y Controles. Por otro lado, existe la escena del Nivel Jugable, que reúne los elementos propios del gameplay: el personaje controlado por el usuario, los enemigos, el GameManager y el HUD, responsable de mostrar las barras de vida y el timer con el tiempo restante para completar el nivel.

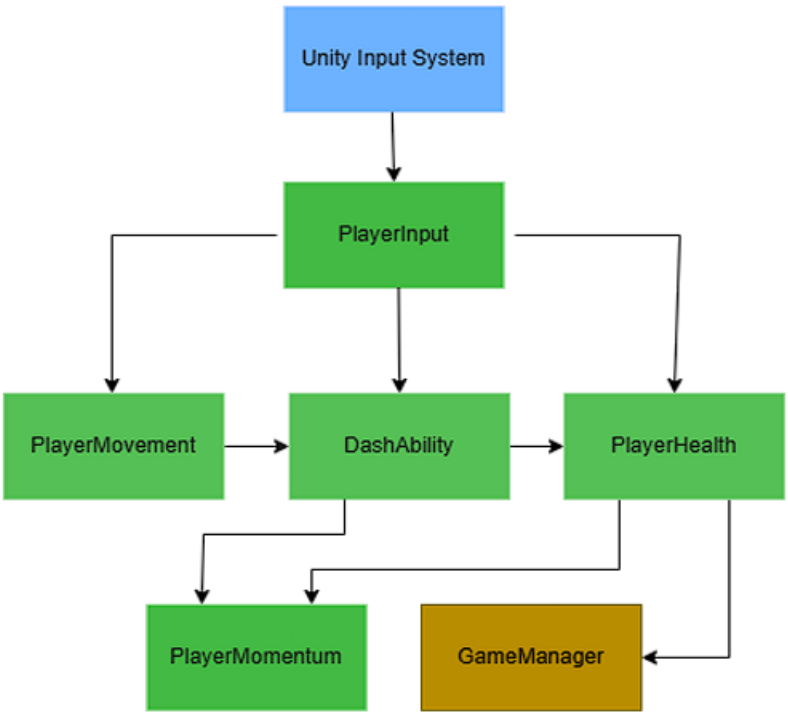
4.1.2. Arquitectura del player

Cada responsabilidad del personaje se encuentra encapsulada en esta clase independiente para que el mantenimiento, la escalabilidad y la comprensión del sistema sean más sencillos.

A continuación, se presenta la arquitectura general de este módulo para visualizar relaciones y el flujo de información entre sus principales componentes.

Figura 9

Diagrama de flujo del módulo player.



Nota: elaboración propia.

Como se observa en la figura 9, el flujo de información inicia en el sistema de entrada de Unity el cual es gestionado por la clase PlayerInput centralizando de esta forma la lectura de las acciones del jugador y las expone al resto del sistema para que PlayerMovement se encargue de la física y el control de movimiento, seguido de DashAbility que implementa la habilidad de desplazamiento rápido consumiendo vida y afectando al sistema de PlayerMomentum. PlayerHealth administra la vida del personaje y notifica al GameManager en caso de muerte para el sistema de respawn y, finalmente, PlayerMomentum evalúa la velocidad del personaje para generar un valor de momentum que permite la regeneración de vida.

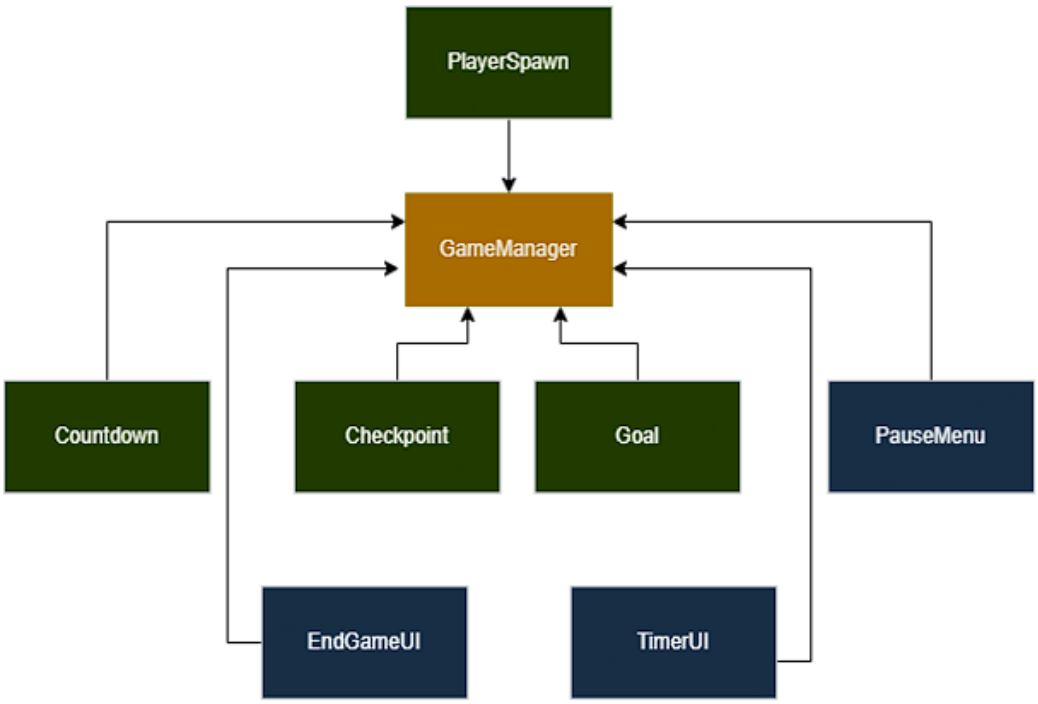
4.1.3. Arquitectura del controlador central

Este módulo gestiona los aspectos globales del nivel: el flujo de inicio, el control del tiempo, los puntos de reaparición, las condiciones de victoria y derrota, la pausa y la interfaz de finalización basando su diseño en un patrón Singleton con el fin de tener una única instancia accesible desde cualquier parte del proyecto y que permita una comunicación desacoplada mediante eventos.

A continuación, se visualiza la arquitectura de este módulo en la que se observan los componentes que interactúan con el núcleo de gestión del juego y el flujo de información entre ellos.

Figura 10

Diagrama de flujo del módulo controlador central



Nota: elaboración propia.

Como se muestra en la figura 10, el GameManager actúa como el nodo central del sistema iniciando con PlayerSpawn quien es el que instancia al jugador, lo registra y establece

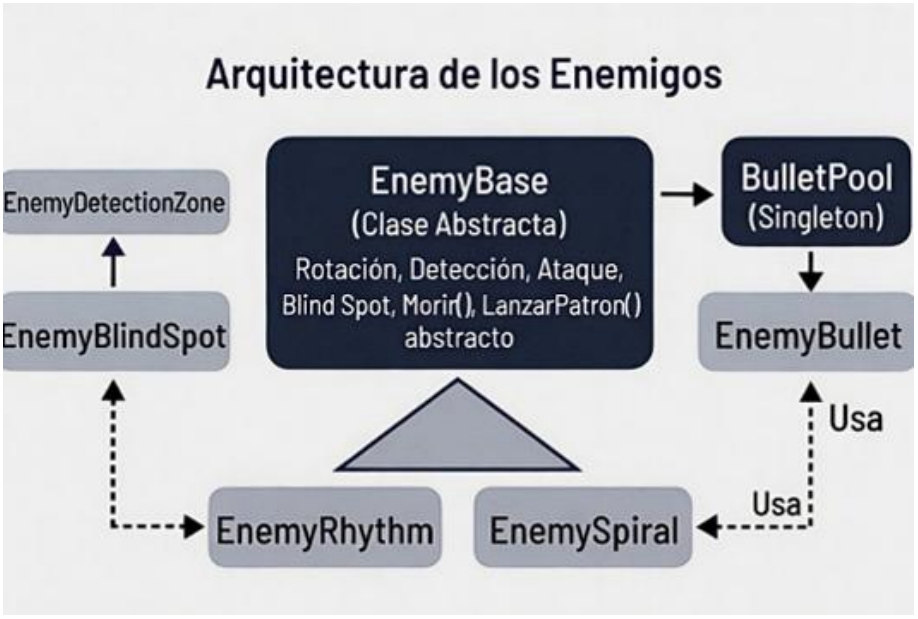
el punto de aparición inicial para que después Countdown ejecute la secuencia de inicio y solicite al GameManager el comienzo oficial de la partida, activando el temporizador y el cronómetro. Los objetos Checkpoint actualizan dinámicamente el punto de respawn, mientras que Goal evalúa la condición de victoria y notifica al sistema central. PauseMenu permite suspender y reanudar la partida sincronizándose con componentes de interfaz TimerUI y EndGameUI.

4.1.4. Arquitectura del enemigo

Para reutilizar el comportamiento común y facilitar la incorporación de nuevos tipos de enemigos se implementa una arquitectura para los mismos, siendo este diseñado bajo un enfoque orientado a la herencia y la composición, con el fin de dar paso al sistema que se centra en una clase abstracta que define la lógica base mientras que los enemigos concretos implementan sus propios patrones de ataque.

Figura 11

Estructura del módulo de enemigo



Nota: elaboración propia.

Como se observa en la figura 11, dentro de todo es la clase abstracta EnemyBase, la cual se encarga de gestionar la rotación, la detección del jugador, el sistema de ataque, la zona ciega y muerte, siendo la clase base que a partir de esta heredan los enemigos concretos EnemyRhythm y EnemySpiral con un patrón de disparo diferente.

La detección del jugador y el manejo de la zona ciega se realizan mediante los componentes auxiliares EnemyDetectionZone y EnemyBlindSpot, mientras que el sistema de proyectiles se gestiona a través de un Object Pool para mejorar el rendimiento del juego reduciendo la recolección de basura, siendo ahí de donde los enemigos obtienen las balas para ser disparadas.

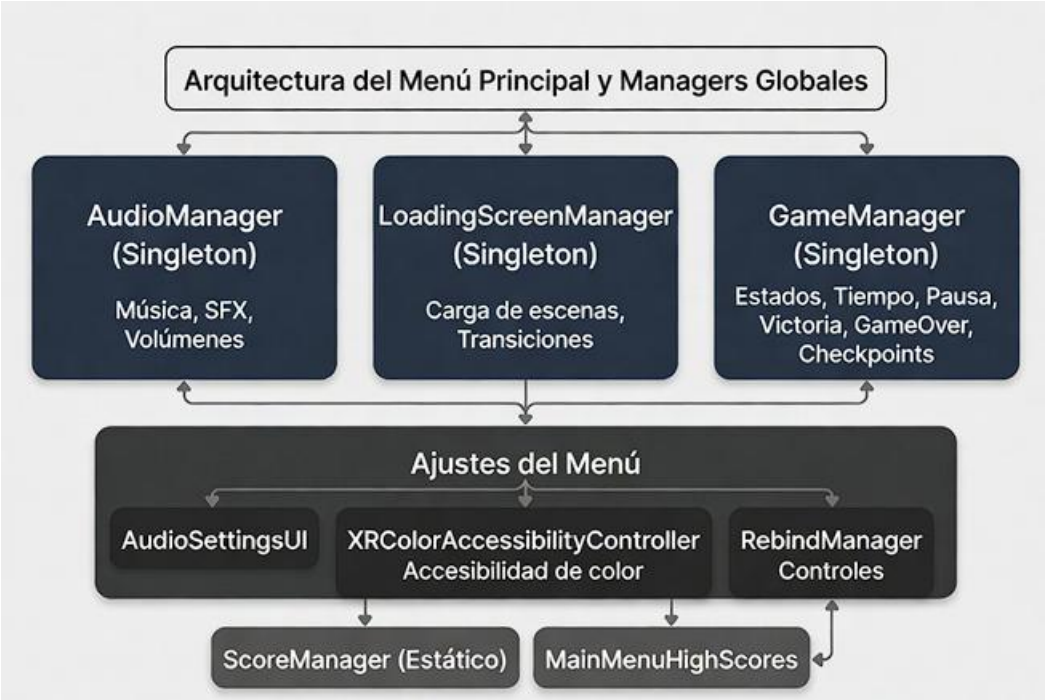
4.1.5. Arquitectura del menú principal y manager globales

La escena de menú principal no solo actúa como punto de entrada de la aplicación, también es el lugar donde se inician los sistemas globales que persisten a lo largo de todo el

juego los cuales son los managers que ya se han hablado con anterioridad, del mismo modo, se implementan principalmente bajo el patrón Singleton, estos se encargan de la gestión de audio, la carga de escenas, el control de estados del juego y el almacenamiento de puntuaciones, relacionándose directamente con el menú que concentra los paneles de configuración que permiten al jugador personalizar el audio, la accesibilidad visual y los controles.

Figura 12

Estructura del módulo de menú principal y manager globales



Nota: elaboración propia.

Como se observa en la figura 12, el AudioManager gestiona la música, los efectos de sonido y los volúmenes; el LoadingScreenManager se encarga de las transiciones entre escenas; y el GameManager administra los estados del juego, el tiempo, la pausa, la victoria, el game over y los checkpoints.

Los ajustes del menú se organizan en tres componentes: AudioSettingsUI para el control de volúmenes, XRColorAccessibilityController para los modos de accesibilidad de color, y RebindManager para la reasignación de controles, terminando con el ScoreManager y el MainMenuHighScores, encargados de mostrar de guardar los puntajes.

4.1.6. Diseño gráfico e interfaces

La propuesta busca mantener un diseño claro, intuitivo y accesible, de manera que el jugador pueda identificar rápidamente la información relevante y emplear cada componente de la interfaz, buscando sobre todo simplicidad visual y coherencia estética para que se conviertan en pilares fundamentales en la inmersión.

Ilustración 20
HUD del juego VtL.

Nota: elaboración propia.

En la Ilustración 20 muestra el HUD principal donde se implementan dos elementos esenciales para la retroalimentación del jugador: la barra de vida, ubicada en la parte superior izquierda, y el timer o cuenta regresiva, en la parte central superior de la pantalla, la primera comunica de forma directa el estado actual del personaje para que el usuario pueda gestionar

sus acciones en función de la resistencia disponible; la segunda es un temporizador que establece un límite de tiempo para completar el nivel.

Ilustración 21
Panel de pausa.

Nota: elaboración propia.

En la Ilustración 21 se puede apreciar cómo este panel de pausa presenta únicamente un texto central que indica que el juego está detenido, acompañado de un botón que permite regresar al menú principal.

Ilustración 22
Panel de victoria

Nota: elaboración propia.

En la Ilustración 22 se observa el panel de victoria o derrota significando el cierre de cada partida y dando de manera inmediata el resultado obtenido, centrándose en el diseño que incluye un texto destacado que anuncia si el jugador ha ganado al alcanzar la meta o perdido al agotarse el tiempo; si el jugador gana se añade el puntaje en la parte inferior derecha.

4.2. Desarrollo

Scrum ha sido la metodología elegida dado que se trata de un videojuego cuyo eje principal es la experiencia del usuario y se requiere un proceso iterativo que permita validar y mejorar continuamente las mecánicas implementadas, de este modo Scrum ofrece un enfoque incremental e iterativo donde cada sprint constituye un incremento funcional del producto conformando un sistema completo.

La organización del desarrollo se estructuró en sprints definidos a partir del Product Backlog con el fin de priorizar las funcionalidades críticas para garantizar un prototipo jugable desde las primeras etapas, viéndose reflejado en que cada sprint se estableció con

entregables claros desde la programación del movimiento y las mecánicas principales hasta la integración de enemigos, niveles y sistemas de retroalimentación.

Mediante un sistema jugable en el que se integran mecánicas de riesgo-recompensa, control de personaje mediante el nuevo Input System de Unity, HUD dinámico y progresión de nivele es que se ha planteado este proyecto siendo a nivel de ingeniería que se resuelve la necesidad de contar con un prototipo funcional que permita validar tanto la jugabilidad como la estabilidad del sistema.

4.2.1 Inventario de recursos

A diferencia de una dependencia exclusiva de bibliotecas externas, el proyecto usa gran parte de Assets desarrollados de manera original, incluyendo el pixel art del personaje principal, los enemigos y sus animaciones, los escenarios, los iconos y la totalidad de los sprites que conforman la experiencia de juego.

No obstante, se incorporaron algunos recursos externos que complementan el desarrollo: la música del videojuego, adquirida de terceros, y el paquete Feel, obtenido desde la Unity Asset Store, utilizado para enriquecer la interacción y retroalimentación en la jugabilidad.

Tabla 6

Tabla de recursos utilizados en el proyecto

Recurso	Nombre	Autoría	Descripción
Efectos visuales	Feel (Asset)	More Mountains	Conjunto de efectos visuales para feedback.
Música	Bingo & Bass	iletzy	Canción chiptune para el nivel tutorial
	Fast Fight	Ville Nousiainen	Canción chiptune para el nivel dos
	A Clockwork	skrjablin	Canción chiptune para el nivel uno
	Gravy	Alex McCulloch	Canción chiptune para el nivel de menú principal
	CG Ram	Chasersgamimg	Canción chiptune para la previsualización del nivel.

Escenarios	VtL. Parallax Background	elaboración (Asesprite)	propia	Fondo para el menú principal, nivel tutorial, nivel 1 (parallax) y nivel 2 (parallax).
Efectos de sonido	VtL. SFX	elaboración (jsfxr)	propia	Efectos de sonido para la selección en el menú principal. Efectos de sonido para el personaje, enemigos y sucesos varios (salto, dash, disparo enemigo, victoria, etc).
Personajes y enemigos (con animaciones)	VtL. Character and Enemies (animated)	elaboración (Asesprite)	propia	Sprites 2D del personaje con animaciones y sprites de los enemigos y el proyectil.
Iconos y sprites	VtL. Icons	elaboración (Asesprite)	propia	Iconos para una mejor navegación en el menú principal y el tutorial, también se lo emplea para el UI.
Tiles	VtL. Tiles	elaboración (Asesprite)	propia	Cuadros para poder editar el nivel mediante su acomodación y la herramienta tilemaps de Unity.

Nota. Elaboración propia

En la Tabla 6 se evidencia los recursos empleados en el proyecto diferenciándose claramente entre los recursos originales y aquellos adquiridos de terceros, organizándose y registrando cómo cada componente contribuye a la construcción de la experiencia de juego desde los elementos gráficos diseñados en Sprites hasta la música chiptune.

También se incorporaron herramientas y paquetes de Unity que resultaron fundamentales para optimizar la jugabilidad y la presentación visual como, por ejemplo: TextMesh, Cinemachine y el New Input System Package para la gestión precisa y configurable de los controles del jugador.

Después de todo lo anterior, se aplicaron configuraciones avanzadas del Universal Render Pipeline (URP) que incluían el uso de volumen y efectos de post-procesado como Bloom con el fin de enriquecer visualmente la experiencia.

4.2.2 Product Backlog

En esta sección se detalla el producto backlog completo y las User Stories (US) que describen desde la perspectiva del jugador, tester o equipo de desarrollo, las necesidades que el sistema debe satisfacer.

Este backlog refleja el proceso iterativo seguido para que se asegure que las características implementadas respondan a los objetivos de diseño y a la experiencia de usuario prevista.

Tabla 7

Product Backlog

PRODUCT BACKLOG				
TÍTULO: DISEÑO, DESARROLLO Y EVALUACIÓN DE UN PROTOTIPO DE PLATAFORMAS 2D CON MECÁNICA DE RIESGO-RECOMPENSA			**FECHA DE PREPARACIÓN:** 10/07/2026	
ID	**DESCRIPCIÓN RESUMIDA**	**PRIORIDA**	**HISTORIA**	**ESTADO**
US 01	Como jugador, necesito mover al personaje lateralmente y saltar para desplazarme correctamente por los niveles.	Alta	Implementación del movimiento, salto y físicas básicas del personaje.	Finalizado
US 02	Como jugador, necesito consumir vida al utilizar habilidades especiales para experimentar la mecánica riesgo-recompensa.	Alta	Desarrollo del sistema de consumo de vida ligado a habilidades.	Finalizado
US 03	Como jugador, necesito utilizar una habilidad de dash o impulso para superar obstáculos y avanzar rápidamente.	Alta	Desarrollo de la habilidad especial de movimiento.	Finalizado
US 04	Como jugador, necesito visualizar mi vida mediante un HUD para conocer mi estado durante la partida.	Alta	Implementación de la barra de vida e interfaz básica.	Finalizado
US 05	Como jugador, necesito recibir retroalimentación visual y sonora cuando uso habilidades o recibo daño para comprender de mejor manera que está ocurriendo.	Media	Integración de efectos visuales y sonoros a lo largo de la experiencia.	Finalizado
US 06	Como jugador, necesito enfrentar enemigos básicos que represente un desafío dentro de los niveles.	Media	Desarrollo de enemigos con comportamiento y sistema de daño definidos.	Finalizado
US 07	Como jugador, necesito un sistema de victoria y derrota, además que gestione el núcleo de la experiencia para que no sea un juego desordenado.	Alta	Diseño e implementación de sistema que sea el núcleo de todo el juego, controlando los diferentes estados de esta.	Finalizado
US 08	Como jugador, necesito checkpoints para reiniciar desde puntos de control sin perder todo mi progreso, además de una meta para finalizar el nivel.	Media	Implementación de checkpoints y meta.	Finalizado
US 09	Como jugador, necesito acceder a un menú principal y opciones de pausa y reinicio para controlar a mi manera la experiencia.	Media	Desarrollo de la interfaz y menús básicos.	Finalizado
US 10	Como tester, necesito que el juego mantenga un rendimiento estable para garantizar una experiencia fluida.	Alta	Optimización de rendimiento y físicas.	Finalizado

US 11	Como jugador, necesito un nivel tutorial para interiorizar las mecánicas del juego.	Alta	Desarrollo del nivel tutorial con guía de mecánicas.	Finalizado
US 12	Como jugador, necesito 2 niveles jugables con diferente nivel de dificultad para tener un escalado progresivo de esta.	Alta	Desarrollo de dos niveles completamente jugables con diseño coherente y enemigos.	Finalizado
US 13	Como equipo de desarrollo, necesitamos realizar un test de Likert para evaluar la experiencia de juego.	Alta	Testing con usuarios para recolección de datos.	Finalizado
US 14	Como equipo de desarrollo, necesitamos documentar y preparar el proyecto para la entrega final.	Media	Documentación y preparación de entrega.	Finalizado

Nota. Elaboración propia.

Como se muestra en la Tabla 7, el Product Backlog presenta evidencias del cumplimiento progresivo de todas las historias de usuario definidas para el proyecto, resultando en que cada funcionalidad fue priorizada, desarrollada y finalizada siguiendo criterios de calidad y coherencia con los objetivos planteados.

El backlog se consolida como un registro que documenta el avance del desarrollo y garantiza la preparación del proyecto para su entrega final

4.2.3 Planificación de sprints

La planificación de sprints constituye una parte esencial dentro de la metodología Scrum al permitir organizar el trabajo en ciclos iterativos y entregar valor de manera continua; todo comienza con un sprint, el cual es un periodo de tiempo fijo, en este caso de aproximadamente un mes, durante el cual el equipo de desarrollo se enfoca en completar un conjunto específico de historias de usuario previamente priorizadas en el Product Backlog.

La siguiente planificación distribuye las historias en cuatro sprints, cada uno orientado a un objetivo concreto: desde la implementación de (i) fundamentos del gameplay, pasando por el (ii) Feedback y desafío inicial, hasta el (iii) diseño de niveles con progresión y la (iv) validación final del proyecto asegurando un avance ordenado.

Tabla 8

Tabla de planificación de sprints

PLANIFICACIÓN DE SPRINTS		
COMPONENTE	**SPRINT**	**HISTORIAS**
Fundamentos del gameplay	Sprint 1	US-01: Movimiento y salto del personaje. US-03: Habilidad especial de movimiento. US-02: Consumo de vida con habilidades. US-04: HUD de vida.
Feedback y desafío inicial	Sprint 2	US-05: Feedback visual y sonoro. US-06: Enemigos básicos. US-10: Optimización de rendimiento.
Niveles y sistema central del juego	Sprint 3	US-07: Sistema de juego base. US-08: Checkpoints y meta. US-12: Diseño de niveles jugables. US-11: Nivel tutorial. US-09: Menú principal y pausa.
Validación y entrega final	Sprint 4	US-13: Testing con usuarios. US-14: Documentación y entrega final.

Nota: elaboración propia.

La planificación de sprints nos permite organizar el desarrollo en ciclos de un mes, siguiendo los principios de Scrum, sin olvidar el llevar un registro claro de todas las actividades realizadas.

4.2.4 Sprint 1

Para llevar a cabo el primer sprint, resulta esencial efectuar una revisión minuciosa de las tareas vinculadas a cada historia de usuario que lo conforma.

Tabla 9

Tabla del Sprint 1

US-01: Movimiento y salto del personaje.	• Implementar la lógica de desplazamiento básico del personaje. • Configurar la mecánica de salto, asegurando su correcta interacción con colisiones y físicas.
US-03: Habilidad especial de movimiento.	• Programar la habilidad especial del desplazamiento. • Conectar con el input de los controles para obtener el objetivo del dash.
US-02: Consumo de vida con habilidades.	• Implementar reducción de vida al ejecutar habilidades especiales de movimiento. • Establecer condiciones y reglas para el consumo del recurso.
US-04: HUD de vida.	• Diseñar e implementar el HUD que muestre la barra de vida del jugador. • Programar la actualización dinámica del HUD.

Nota: elaboración propia.

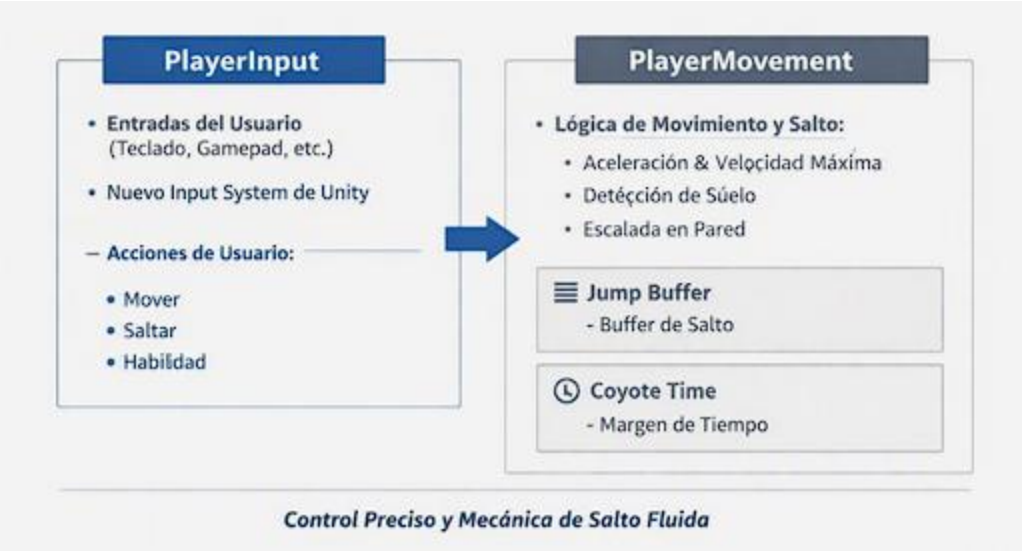
La tabla 9 sintetiza las historias de usuario seleccionadas y las tareas técnicas que las componen, sumado a que la organización evidencia una progresión lógica: desde el movimiento básico del personaje y sus habilidades especiales, hasta la gestión de recursos, la representación visual en el HUD y la configuración del sistema de entradas.

• US-01: Movimiento y salto del personaje.

El sistema de movimiento y salto del personaje se construye a partir de la interacción entre dos componentes principales: PlayerInput y PlayerMovement. El primero actúa como traductor de las acciones del usuario, capturando las entradas de teclado u otros dispositivos mediante el nuevo Input System de Unity y exponiéndolas de forma estructurada con el fin de garantizar que las órdenes de desplazamiento, salto y habilidades lleguen limpias y organizadas al módulo encargado de la lógica física.

Figura 13

Funcionamiento del movimiento del personaje



Nota: elaboración propia.

Como se puede observar en la figura 13, PlayerMovement implementa la lógica completa de desplazamiento y salto, integrando parámetros como aceleración, velocidad

máxima, detección de suelo para gestionar de mejor forma el salto y paredes para la mecánica de escalado, al igual que mecánicas avanzadas como coyote time y jump buffer.

- **US-03: Habilidad especial de movimiento.**

La habilidad de dash permite al jugador realizar un desplazamiento rápido en una dirección determinada, consumiendo una cantidad de vida como costo de activación. Durante su ejecución, el personaje adquiere una velocidad elevada durante un breve periodo de tiempo, acompañado de un efecto visual de estela, después se aplica una desaceleración controlada para suavizar la transición al movimiento normal, incluso contemplando un comportamiento especial cuando el jugador se encuentra adherido a una pared.

Para implementar una nueva habilidad en Unity es recomendable crear un script independiente que contenga toda su lógica y condiciones de activación, para que de esta forma cada habilidad pueda añadirse o quitarse del GameObject sin afectar a las demás. El Player puede tener múltiples scripts de habilidades coexistiendo, cada uno respondiendo a su propio input.

Figura 14

Comunicación con DashAbility

Nota: elaboración propia.

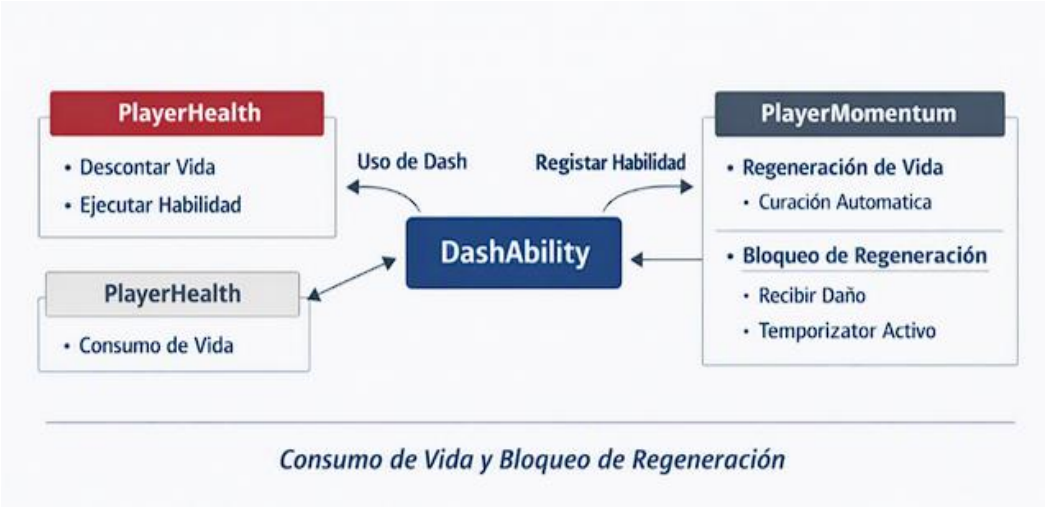
Como se observa en la figura 14, el script DashAbility se comunica con varios componentes del jugador para coordinar su funcionamiento. Obtiene la entrada del usuario a través de PlayerInput, consulta y modifica el estado de movimiento mediante PlayerMovement (incluyendo la detección de paredes, el control de velocidad y la orientación), consume vida a través de PlayerHealth y registra el uso de la habilidad en PlayerMomentum.

- **US-02: Consumo de vida en habilidades.**

El consumo de vida como recurso se gestiona principalmente a través del módulo PlayerHealth, el cual permite descontar puntos de la barra vital cada vez que el jugador ejecuta una habilidad especial. Este diseño asegura que las mecánicas no dependan únicamente de tiempos de enfriamiento, sino también de la administración estratégica de la salud, evitando abusos mediante condiciones que impiden el mal uso de habilidades.

Figura 15

DashAbility y el consumo de vida



Nota: elaboración propia.

Como se puede observar en la figura 15, el sistema PlayerMomentum introduce un factor dinámico que regula la regeneración de vida en función del movimiento del jugador (momentum). Tras recibir daño o utilizar habilidades, se activan temporizadores que bloquean la recuperación automática, obligando al jugador a equilibrar agresividad y supervivencia. En este contexto, el dash se convierte en un ejemplo claro de la interacción entre estos sistemas: al ejecutarlo, se descuenta vida mediante PlayerHealth y se registra el uso de habilidad en PlayerMomentum, lo que impide la regeneración inmediata.

• US-04: HUD de vida.

El sistema de interfaz gráfica para la salud del jugador se implementa mediante el componente PlayerHUD, encargado de representar visualmente el estado vital en pantalla. Cada barra corresponde a un segmento de vida definido, lo que permite una lectura clara y rápida del recurso disponible. El script se suscribe a los eventos de PlayerHealth, actualizando dinámicamente la cantidad de barras activas y asegurando que la información mostrada en la interfaz esté siempre sincronizada con el estado interno del personaje.

Figura 16

Funcionamiento del HUD de vida

Nota: elaboración propia.

Además de la representación básica, el HUD incorpora retroalimentación audiovisual para reforzar la experiencia del jugador. La figura 16 nos permite ver una referencia para un animator, esto con el fin de que cuando la salud del personaje descienda a niveles críticos, se activa una animación de alerta en el fondo, mientras que la recuperación de una barra desencadena efectos de recarga acompañados de sonido.

Pruebas y resultados del Sprint 1:

- Funcionamiento del movimiento y salto del personaje: Se verificó que las mecánicas básicas de desplazamiento responden correctamente a las entradas del jugador, incluyendo la ejecución de saltos con parámetros de física coherentes y consistentes.
- Ejecución de la habilidad especial (dash): Se comprobó que el dash se activa en condiciones normales y desde paredes, respetando la duración y dirección configuradas.

- Consumo de vida en habilidades: Se validó que el uso de habilidades especiales descuenta la cantidad correspondiente de la barra de vida con el fin de evitar ejecuciones suicidas y manteniendo la coherencia con el sistema de regeneración de momentum.
- Interfaz de usuario (HUD de vida): Se constató que la UI refleja en tiempo real los cambios en la salud del jugador para mostrar la reducción o recuperación de barras y activando animaciones de alerta o recarga según corresponda.

4.2.4 Sprint 2

El Sprint 2 se orienta hacia la incorporación de elementos de retroalimentación y desafío iniciales dentro del gameplay, buscando fortalecer la interacción del jugador mediante respuestas visuales y sonoras, así como la introducción de enemigos básicos que representen un primer nivel de dificultad.

Tabla 10

Tabla del Sprint 2

US-05: Feedback visual y sonoro.	• Implementar efectos visuales y sonoros que refuercen las acciones del jugador. • Configurar señales inmediatas que transmitan impacto y retroalimentación.
US-06: Enemigos básicos.	• Programar a los enemigos con un sistema de ataque básico. • Programar la interacción entre el enemigo, la bala y el jugador.
US-10: Optimización y rendimiento	• Revisar y ajustar parámetros y otro tipo de configurables del personaje y el mundo. • Implementar optimización para evitar caídas de FPS o perdida de fluidez.

Nota: elaboración propia.

• US-05: Feedback visual y sonoro.

Se implementa un sistema centralizado de gestión de audio mediante la clase AudioManager para controlar música y efectos de sonido, asegurando que cada acción del jugador tenga una respuesta auditiva inmediata y diferenciada, para esto se configuraron clips

específicos para acciones como el salto, el dash y la recepción de daño, reforzando la inmersión y la claridad de las mecánicas.

Como se puede observar en la figura 17, el uso de parámetros en el AudioMixer garantiza un ajuste dinámico de volúmenes, mientras que la persistencia en PlayerPrefs mantiene las preferencias del usuario entre sesiones.

Figura 17

Conexión con el sistema de sonido.

Nota: elaboración propia.

Además, se integra el asset MoreMountains.Feedbacks (Feel) para añadir retroalimentación perceptible en el personaje mediante la configuración de un MMF Player con efectos como Cinemachine Impulse, que genera una ligera turbulencia en la cámara al recibir daño, y Flicker, que provoca un destello breve en el personaje para señalar vulnerabilidad.

• US-06: Enemigos básicos.

Se emplea una estructura que define el comportamiento de los enemigos a través de la clase EnemyBase y sus derivaciones, gestionando la rotación periódica, la detección del

jugador y la ejecución de ataques mediante corrutinas, además de integrar retroalimentación sonora en acciones como atacar o morir.

El enemigo EnemyRhythm utiliza esta arquitectura para lanzar un patrón radial de disparos, generando múltiples proyectiles en distintas direcciones.

Figura 18

Funcionamiento del sistema de enemigos y proyectiles

Nota: elaboración propia.

El script EnemyBullet se usa para controlar la lógica de cada proyectil disparado por los enemigos. Como se puede ver en la figura 18, cada bala se inicializa con una dirección, se desplaza a velocidad constante y cuenta con un tiempo de vida limitado para evitar acumulación en la escena y al colisionar con el jugador, aplica daño directo a través del componente PlayerHealth.

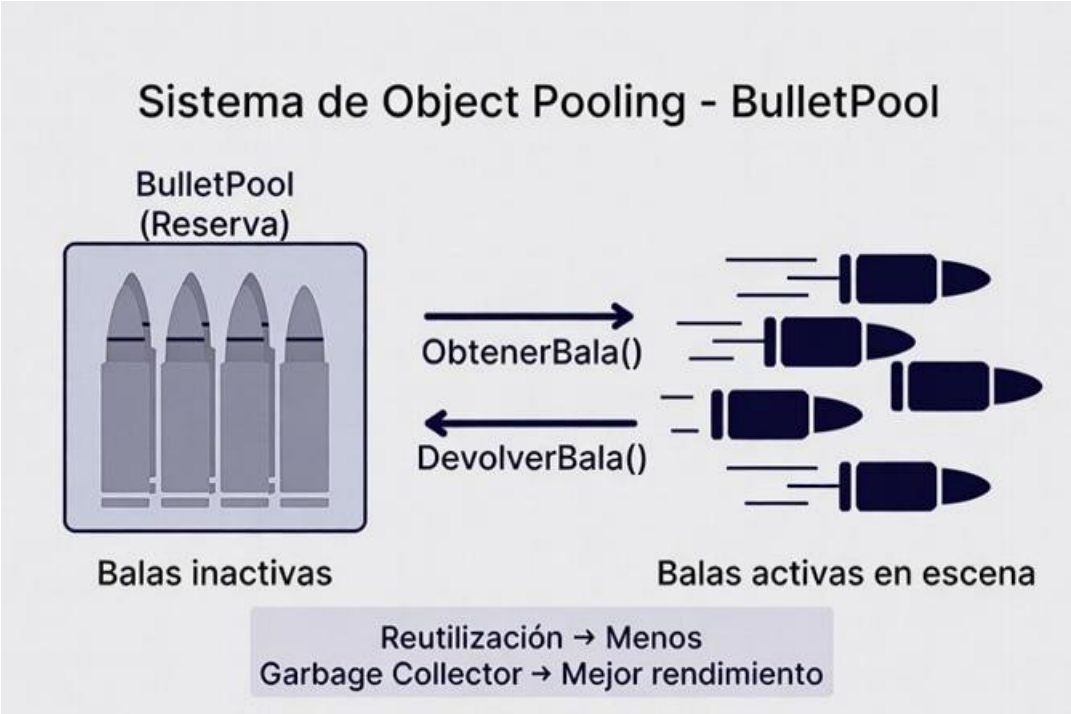
El juego cuenta con dos enemigos con diferente patrón de disparo, pero si se desea en un futuro añadir uno adicional con un patrón diferente solo hace falta que se herede de la clase base EnemyBase y sobreescribir sus métodos.

US-10: Optimización de rendimiento

Para la optimización se ajustan parámetros clave como la velocidad y la gravedad del jugador, además de pruebas de colisión para garantizar que no existan errores de atravesamiento ni inconsistencias en la interacción con el entorno. El uso de componentes como Composite Collider 2D permite que los Tilemaps combinen sus colisionadores en uno solo, reduciendo el costo computacional y mejorando la estabilidad general del sistema de físicas.

Figura 19

Sistema de object pooling



Nota: elaboración propia.

Por otro lado, como se ve en la figura 19, se utiliza un sistema de pooling para los proyectiles, lo que significa que las balas no se crean ni destruyen constantemente, sino que se reutilizan desde una reserva preexistente, disminuyendo la carga en el Garbage Collector

y reduciendo la generación de instancias innecesarias, mejorando el rendimiento en escenas con múltiples disparos simultáneos.

Pruebas y Resultados del Sprint 2

- Feedback visual y sonoro: El sistema de audio responde correctamente a las acciones del jugador, reproduciendo clips específicos para salto, dash y daño; los efectos visuales configurados con MoreMountains.Feedbacks se activan de manera consistente, incluyendo turbulencias de cámara y destellos en el personaje al recibir daño.
- Enemigos básicos: Los enemigos detectan al jugador y ejecutan patrones de ataque definidos, como disparos radiales mediante proyectiles, combinado con efectos sonoros de ataque y destrucción se reproducen adecuadamente, y la interacción con el componente PlayerHealth descuenta la vida del jugador al colisionar.
- Optimización de rendimiento y físicas: Los ajustes en velocidad, gravedad y colisiones eliminan errores de atravesamiento y mejoran la estabilidad del sistema de físicas, combinado con el uso de Composite Collider 2D para optimizar los Tilemaps al reducir la cantidad de colliders activos, mientras que el sistema de pooling de proyectiles mantiene un rendimiento estable en situaciones de combate intenso.

4.2.5 Sprint 3

El Sprint 3 se orienta hacia la construcción de los niveles y la implementación del sistema central de juego para establecer la lógica que regula estados fundamentales como victoria, derrota, pausa y ejecución activa, además de integrar elementos de progresión como checkpoints y un nivel tutorial que facilite la introducción de las mecánicas básicas, sin dejar de lado el desarrollo del menú principal y el sistema de pausa.

Tabla 11

Tabla del Sprint 3

US-07: Sistema de juego.	• Implementar la lógica completa de los niveles, incluyendo estados como victoria, derrota, pausa y ejecución. • Configurar condiciones de finalización y reinicio, asegurando coherencia en la progresión.
US-08: Checkpoints y meta.	• Diseñar puntos de control que permitan la reaparición del jugador. • Establecer una meta final en cada nivel, vinculando su activación con la condición de victoria.
US-12: Dos niveles jugables	• Implementar dos escenarios completos con tilemaps, enemigos, meta y checkpoints. • Configurar la progresión entre niveles, asegurando que el jugador pueda avanzar de manera coherente entre ellos.
US-11: Nivel tutorial	• Desarrollar un nivel introductorio que guíe al usuario para poder dominar las mecánicas básicas • Incorporar carteles y señales para que el camino sea más ameno.
US-09: Menú principal y pausa	• Implementar un menú principal con opciones de jugar, tutorial, configuración y créditos. • Configurar un sistema de pausa sin afectar la lógica del juego.

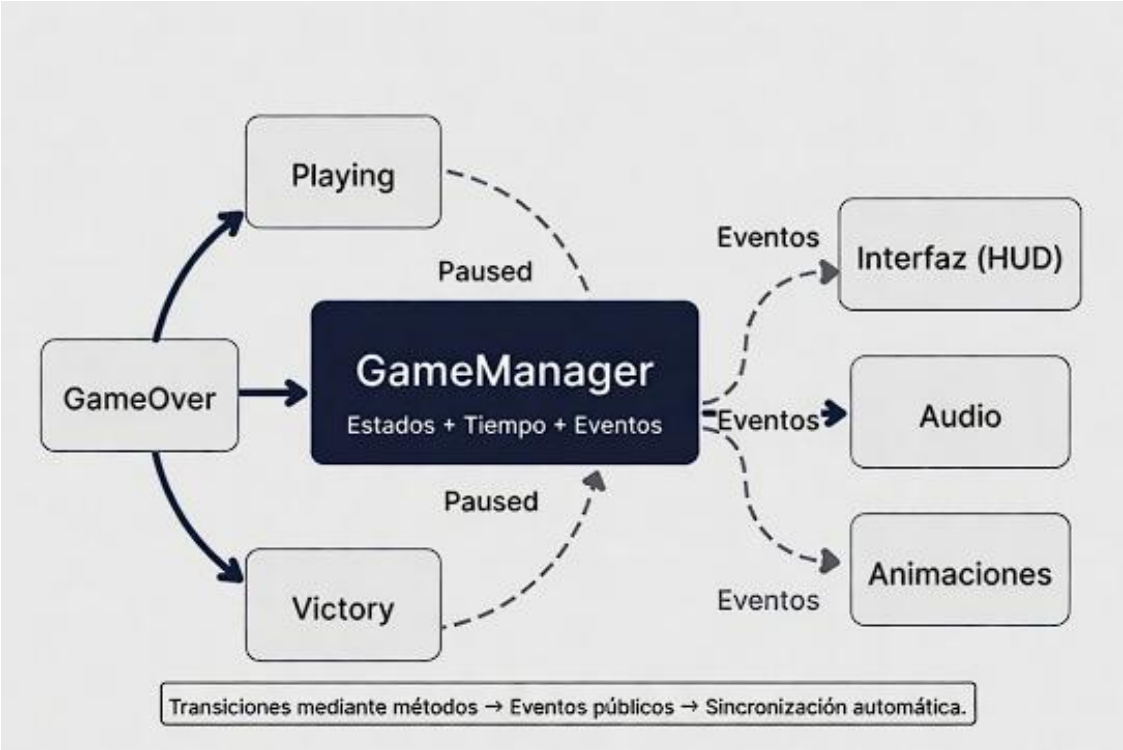
Nota: elaboración propia.

• US-07: Sistema de juego (controlador central).

El núcleo del juego se organiza alrededor de un sistema central que regula los estados de la partida, el tiempo y la interacción con la interfaz. Como se puede observar en la figura 20, el GameManager es el encargado de coordinar esta lógica, definiendo estados como Playing, Paused, Victory y GameOver. Cada transición entre estados se realiza mediante métodos específicos que no solo cambian la condición interna del juego, sino que también disparan eventos públicos que otros componentes escuchan para reaccionar en consecuencia. Esto permite que la interfaz, el audio y las animaciones se sincronicen automáticamente con lo que ocurre en la partida.

Figura 20

Estados de juego y GameManager



Nota: elaboración propia.

Se emplea un cronómetro acumulativo que mide la duración total de la sesión y un temporizador regresivo que define el tiempo restante del nivel. Si el temporizador llega a cero, se activa el estado de GameOver de manera automática. Este tiempo se refleja en pantalla mediante el TimerUI, que traduce los segundos en un formato estándar de minutos y segundos, manteniendo al jugador informado en todo momento, con opciones de agregar en tiempo real unos segundos más.

Figura 21

Flujo de inicio y final de la partida

Nota: elaboración propia.

Como se puede ver en la figura 21, el inicio de cada nivel está gestionado por el Countdown, que bloquea las entradas del jugador, muestra una cuenta regresiva con retroalimentación visual y sonora, y finalmente invoca al GameManager para iniciar la partida. Una vez que el estado cambia a Playing, los cronómetros se activan y la música principal comienza. El cierre de la partida, por su parte, se maneja con el EndGameUI, que despliega la interfaz de victoria o derrota según corresponda, calcula puntajes basados en el tiempo restante y asegura la transición ordenada hacia el menú principal.

El sistema también contempla la pausa mediante los métodos Pausar() y Reanudar(); al pausar se detiene la simulación del tiempo en Unity y se congelan los cronómetros, mientras que al reanudar se restablece el flujo normal. En conjunto, el GameManager y los scripts asociados conforman un núcleo sólido que regula estados, tiempo, inicio y cierre de partidas.

Cada componente cumple una función específica dentro de este flujo: el GameManager coordina la lógica, el TimerUI refleja el tiempo, el Countdown marca el inicio y el EndGameUI gestiona la conclusión.

• US-08: Checkpoints y meta.

Los checkpoints permiten que el jugador registre un punto seguro en el escenario, de manera que si pierde la partida pueda reaparecer en la última posición alcanzada sin necesidad de reiniciar desde el inicio ya que al activarse el sistema actualiza la referencia en el GameManager, reproduce un efecto sonoro y muestra un indicador visual que confirma la activación.

Figura 22

Interacción con la meta

Nota: elaboración propia.

La meta representa la condición de victoria dentro de cada nivel, al alcanzarla se valida la colisión con el jugador como se puede apreciar en la figura 22, se guarda la puntuación en función del tiempo restante y se invoca el estado de victoria en el GameManager, todo esto gestionado por el disparo de los eventos necesarios para mostrar la

interfaz de resultados y reproduce efectos visuales y sonoros que refuerzan la retroalimentación.

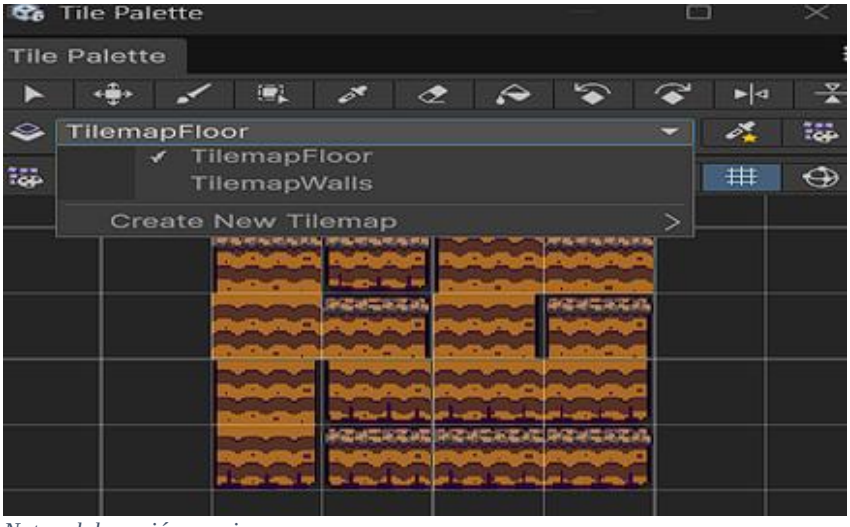
• US-12: Diseño de dos niveles jugables.

En el diseño de los dos niveles jugables se partió de la importación de los tilemaps en forma de tile palette como se puede ver en la Ilustración 23.

Se definieron dos tipos principales de tilemaps: uno de tierra y otro de metal. El tilemap de tierra se asignó al layer floor, mientras que el de metal se colocó en el layer wall. Esta separación fue clave para gestionar el sistema de WallHandle dentro del PlayerMovement, ya que permite que el personaje pueda agarrarse a las paredes o recargar el salto al volver a tocar el suelo. En cuanto a la parte del fondo del videojuego, se incorporaron los fondos de cada nivel divididos en cuatro capas, lo que generó un efecto de parallax por la perspectiva de la cámara.

Ilustración 23

Importación de tilemaps a Unity



Nota: elaboración propia.

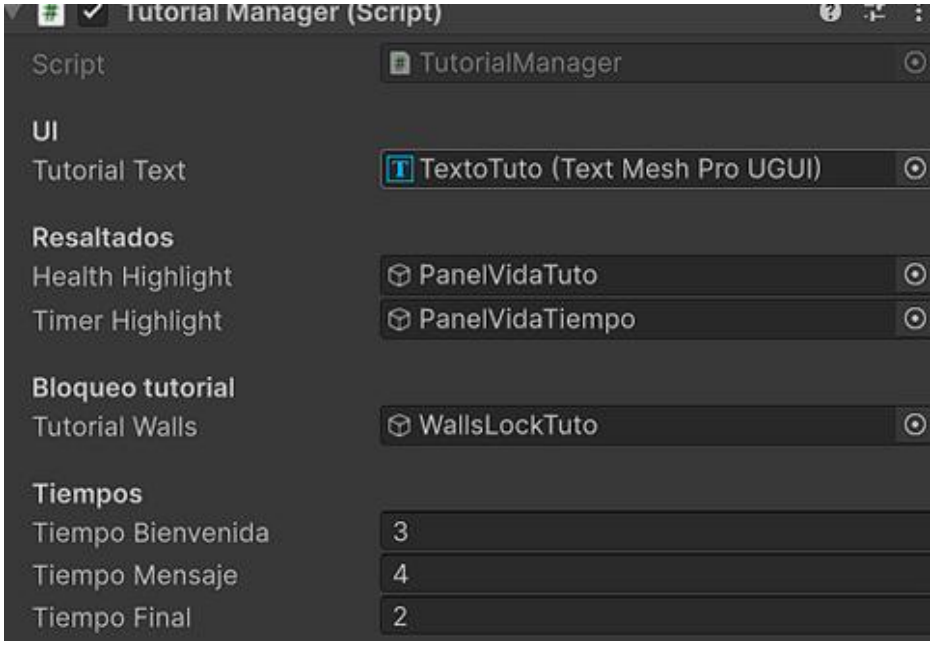
La disposición de los espacios y la ubicación de los enemigos dentro de cada nivel fue pensada de manera que no suponga un grado de frustración en el jugador, dado de la búsqueda de un balance entre desafío y oportunidad de recuperación.

Finalmente, se colocaron los checkpoints en puntos estratégicos para mantener la continuidad del avance y se definió la ubicación de la meta, consolidando la progresión y el cierre de cada escenario.

- **US-11: Nivel Tutorial**

El nivel tutorial se construye como un escenario independiente con tilemaps propios de tierra; tras la cuenta regresiva inicial, el jugador permanece encerrado por las paredes de bloqueo, lo que asegura que no pueda avanzar hasta completar la secuencia de enseñanza. Durante este proceso, el sistema muestra mensajes guiados en pantalla que explican el funcionamiento del HUD, resaltando la barra de vida y energía, así como el temporizador mientras se destaca visualmente los objetos de interfaz que se activan y desactivan en sincronía con los mensajes.

Ilustración 24
Vista desde el inspector de TutorialManager



Nota: elaboración propia.

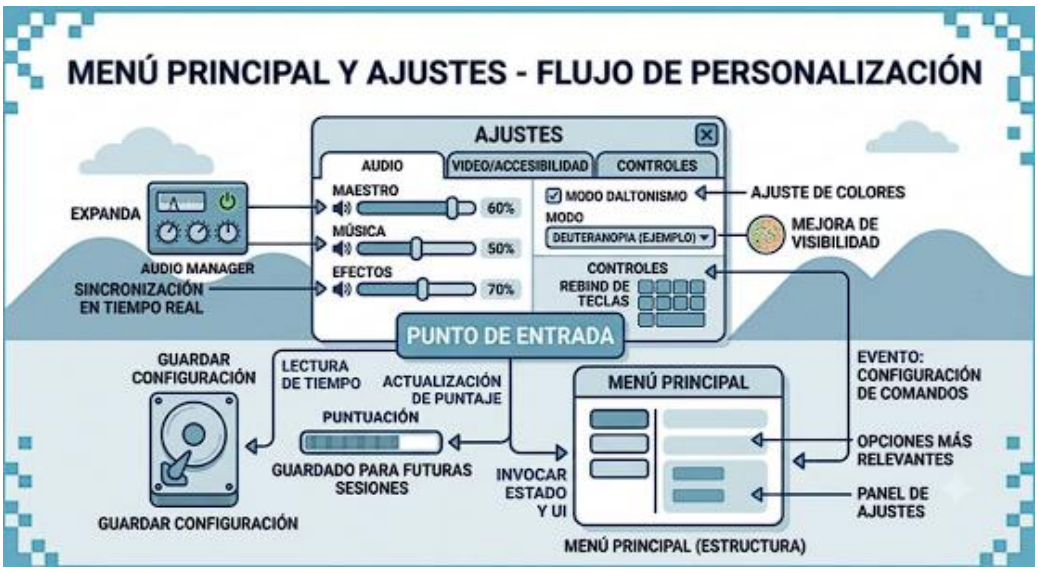
Como se puede ver en la Ilustración 24, el TutorialManager coordina toda la lógica a través de una corrutina que presenta los mensajes en orden, con tiempos definidos para cada explicación. Primero se da la bienvenida, luego se explica cómo funciona la barra de vida y energía, incluyendo la mecánica de recuperación al correr, y finalmente se introduce el temporizador como referencia de límite de tiempo. Al concluir la secuencia, las paredes de bloqueo se destruyen y el jugador queda libre para continuar, habilitando nuevamente la pausa y el flujo normal del juego. De esta manera, el nivel tutorial asegura que el jugador comprenda las mecánicas esenciales antes de enfrentarse a los desafíos de los niveles principales, integrando retroalimentación visual, sonora y lógica central en un entorno seguro y controlado.

• US-09: Menú principal y pausa.

Dentro del menú, el panel de ajustes es el elemento más destacado, ya que permite modificar parámetros esenciales, dando paso a las diferentes secciones configurables: en la sección de audio el jugador puede controlar el volumen maestro, la música y los efectos de sonido mediante sliders sincronizados con el AudioManager, lo que asegura que los cambios se reflejen en tiempo real y puedan guardarse para futuras sesiones como se puede observar en la figura 23. En la sección de video, se incluye la configuración de accesibilidad con el modo daltonismo, que ajusta colores e intensidad para mejorar la visibilidad según las necesidades del usuario. Finalmente, la sección de controles permite realizar rebind de teclas, ofreciendo flexibilidad para que cada jugador configure la disposición de comandos a su manera.

Figura 23

Estructura del menú principal



Nota: elaboración propia.

Al presionar la tecla asignada, el sistema verifica que la partida no esté en estado de victoria ni derrota, y si el jugador está en pleno gameplay, se activa la pausa. Todo esto

mientras se detiene la simulación del tiempo (Time.timeScale = 0), congela los cronómetros y despliega el panel de pausa en la interfaz, en la cual el jugador puede acceder a la opción de salir al menú principal.

Además de pausar y reanudar la acción, el panel incluye la opción de volver al menú principal, lo que se realiza mediante el LoadingScreenManager, encargado de cargar la escena correspondiente con transiciones visuales para ofrecer un espacio seguro y sensación de control del juego.

Pruebas y Resultados del Sprint 3

- Sistema de juego: El núcleo del gameplay funciona de manera estable, con el GameManager gestionando correctamente los estados de la partida (Playing, Paused, Victory, GameOver), los cronómetros acumulativos y regresivos, y la sincronización con la interfaz mediante eventos.
- Checkpoints y meta: Los checkpoints registran la posición del jugador al ser activados, mostrando retroalimentación visual y sonora, y que la meta desencadena el estado de victoria al alcanzarse. El sistema guarda la puntuación en función del tiempo restante y activa los elementos de interfaz y audio correspondientes.
- Dos niveles jugables: Los dos escenarios diseñados con tilemaps de tierra y metal funcionan de manera independiente, integrando el efecto de parallax en los fondos y aplicando correctamente las mecánicas de movimiento, salto y wall handle.
- Nivel tutorial: El tutorial introduce al jugador en las mecánicas básicas mediante mensajes guiados y resaltados en el HUD. El sistema bloquea el avance inicial con paredes, explica la barra de vida, el consumo de energía y el temporizador, y finalmente libera al jugador para continuar.

- Menú principal y pausa: Se verifica que el menú principal permite configurar audio, video y controles, incluyendo el modo daltonismo y el rebind de teclas; por otro lado, el panel de pausa detiene la acción, congela los cronómetros y despliega las opciones de ajustes.

4.2.6 Sprint 4

El Sprint 4 se orienta hacia la validación final del proyecto y la preparación de la entrega completa mediante pruebas de usabilidad con jugadores reales, lo que permite comprobar la estabilidad de las mecánicas, la claridad de las interfaces y la accesibilidad de las configuraciones; siendo el objetivo principal el recopilar observaciones sobre la dificultad, la curva de aprendizaje y la coherencia del sistema.

Tabla 12

Tabla del Sprint 4

US-13: Testing con usuarios	• Realizar pruebas de usabilidad. • Recopilar observaciones sobre claridad de instrucciones y de mecánicas.
US-14: Documentación y entrega final	• Consolidar la información técnica y metodológica en un documento formal. • Preparar la entrega del proyecto con niveles jugables, menú principal y sistema de pausa integrados.

Nota: elaboración propia.

En cambio, la segunda parte del sprint tiene que ver con la documentación y entrega final, aquí se consolida toda la información técnica y metodológica para el informe final, incluyendo evidencias.

• US-013: Testing con usuarios

Se realiza una prueba presencial con 30 participantes pertenecientes al público objetivo del proyecto. La actividad se desarrolla en un centro comercial, donde cada participante interactúa con el videojuego durante un periodo aproximado de cinco a diez minutos.

Figura 24

Road Map del testing con usuarios

Nota: elaboración propia.

Como se puede ver en la figura 24, durante la prueba los usuarios exploran la experiencia completa del prototipo, comenzando desde el menú principal y continuando con opciones de configuración, el tutorial y los niveles disponibles, permitiendo de esta forma que cada participante conozca la navegación, los controles y las mecánicas principales del videojuego antes de completar la evaluación.

Al finalizar la sesión de juego, se aplica una encuesta estructurada con preguntas relacionadas con la experiencia de uso, la comprensión de las mecánicas, la interacción con los sistemas y la percepción general del prototipo. La información recopilada sirve como base para el posterior análisis de la validación con usuarios.

• Resultados

Primero se evalúa la consistencia interna del instrumento mediante el coeficiente alfa de Cronbach únicamente con el fin de determinar el grado de confiabilidad de un cuestionario a partir de la relación existente entre las preguntas que lo conforman; adicionalmente, en la

parte de anexos se encuentran los resultados del test aplicado en forma de gráfica de pastel para su validación.

El cálculo del coeficiente emplea las respuestas obtenidas de los 30 participantes, utilizando la codificación numérica de la escala de Likert de cinco niveles. A partir de estos datos se obtiene un valor de $\alpha = 0.933$, el cual corresponde a un nivel de consistencia interna excelente. Lo anterior únicamente valida que existe una alta coherencia con los ítems expuestos y que su base es confiable para el análisis.

Se calcula además una valoración promedio entre todas las preguntas considerando las respuestas de los 30 participantes, ellos dieron respuestas del 1 al 5 correspondiente al nivel de acuerdo de cada pregunta. Después se obtiene el promedio de todas las respuestas registradas para que estas lancen una valoración grupal de 4.51 sobre 5.

Ahora abordaremos individualmente cada apartado mediante la siguiente tabla:

Tabla 13
Tabla de valoración por pregunta en escala de Likert

Indicador	Promedio 1-5	Porcentaje de Satisfacción (4 o 5)
Atención sostenida durante la partida	4.67	100% (30/30)
Involucramiento en la experiencia del juego	4.67	100% (30/30)
Aumento de tensión usando la vida como recurso	4.67	96.7% (29/30)
La mecánica principal incrementó el entretenimiento	4.60	100% (30/30)
Motivación a seguir jugando después de los primeros 3 minutos	4.57	96.7% (29/30)
Planificación de movimientos con mayor cuidado	4.53	96.7% (29/30)
Uso meditado del dash por consumo de vida	4.53	93.3% (28/30)
Evaluación previa al riesgo de usar habilidades	4.50	90% (27/30)
Diferenciación frente a otros Plataformas 2D	4.33	86.7% (26/30)

Facilidad de comprensión de mecánicas principales	4.10	76.7% (23/30)

Dato: Elaboración propia.

• Conclusiones principales

Alto Compromiso (Engagement): El 100% de los encuestados calificó con 4 o 5 la capacidad del juego para mantener su atención y sentirse involucrados esto se relaciona con la tasa de motivación para continuar tras los tres minutos iniciales alcanzó un 96.7%.

Éxito de la Mecánica Riesgo-Beneficio (Health-as-a-Resource): La decisión de utilizar la vida como recurso para el dash y otras habilidades logró el objetivo táctico proyectado. El 96.7% sintió un incremento en la tensión de juego y más del 90% afirmó sopesar los riesgos y planificar cuidadosamente sus movimientos antes de actuar.

Diferenciación y Propuesta de Valor: Un 86.7% percibió la experiencia como distinta a la de otros títulos de plataformas tradicionales.

Oportunidad de Mejora: Onboarding y Curva de Aprendizaje: Aunque la comprensión general sigue siendo positiva (4.10 / 5), es la puntuación más baja de la encuesta: un 23.3% otorgó una calificación neutral (3/5). Esto sugiere que se puede pulir el tutorial o la retroalimentación visual/auditiva al introducir la mecánica principal para reducir la fricción inicial.

US-014: Documentación y entrega final

Como parte de la etapa final del desarrollo, se consolida la documentación técnica y funcional del proyecto, integrando la información generada durante las diferentes fases de implementación; aquí se reúne la descripción de la propuesta tecnológica, la arquitectura del sistema, las mecánicas implementadas, el diseño de los niveles, la interfaz de usuario y el proceso de validación, permitiendo registrar de manera organizada los componentes que conforman el prototipo, además, se revisan los diferentes documentos elaborados durante el

proyecto con el fin de mantener la coherencia entre el desarrollo realizado y los entregables finales.

De manera adicional se genera una compilación ejecutable del videojuego mediante Unity, destinada a su distribución y evaluación en un entorno independiente del editor de desarrollo, al lado se incluyen los recursos necesarios para su correcta ejecución, así como el manual de usuario y la documentación correspondiente.

4.3 Recomendaciones

Se recomienda continuar el desarrollo del prototipo incorporando nuevos niveles, enemigos y mecánicas que permitan ampliar la experiencia de juego y evaluar el comportamiento de la arquitectura propuesta en escenarios de mayor complejidad.

Asimismo, se sugiere realizar pruebas con un mayor número de usuarios para obtener información sobre la experiencia de juego, la usabilidad y el funcionamiento de la mecánica de riesgo-recompensa.

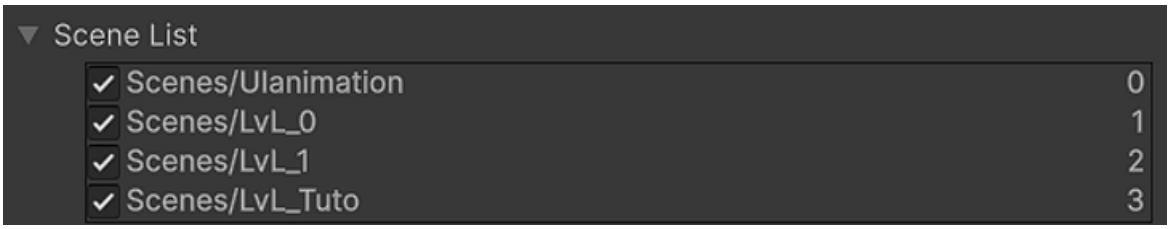
4.4 Organización del proyecto

La siguiente sección abarca la organización del proyecto, explicando cómo se estructuran sus principales componentes y en qué carpetas se encuentran los elementos esenciales como escenas, prefabs y scripts.

4.4.1 Escenas

Ilustración 25

Lista de escenas del juego



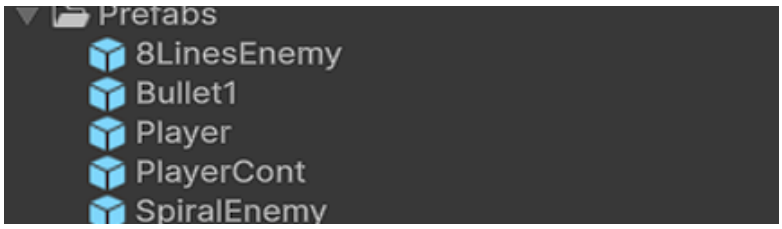
Nota: elaboración propia.

La estructura del proyecto se organiza en cuatro escenas principales: MainMenu, LvL0, LvL1 y LvLtuto. El flujo inicia en el menú principal, desde el cual el usuario puede acceder a cualquiera de las demás escenas. A su vez, cada una de estas escenas incorpora un sistema de pausa que permite regresar en cualquier momento al menú principal.

4.4.2 Prefabs

Ilustración 26

Prefabs utilizados en la carpeta



Nota: elaboración propia.

La organización de los prefabs se centra en un Player con múltiples scripts (input, movimiento, salud, dash y momentum), que se instancia en el PlayerSpawn y cambia según el último checkpoint alcanzado. En La Ilustración 26 existen dos tipos de enemigos: uno que dispara en ocho direcciones y otro que lo hace de forma continua; ambos giran, poseen una DetectionZone para localizar al jugador y un EnemyBlindSpot que los vuelve vulnerables si

el jugador ejecuta un dash en esa dirección, a esto se suma de forma adicional el prefab de bullet, el cual se instancia según el tipo de disparo de los enemigos.

4.4.3 Scripts

Ilustración 27

Lista de carpetas en la que se organizan los scripts

Nota: elaboración propia.

La organización de los scripts se divide en cuatro carpetas principales como se puede ver en la Ilustración 27. En Core se agrupan los componentes núcleo del sistema, como AudioManager, CameraIntro, Checkpoint, Countdown, EndGameUI, GameManager, Goal, LoadingScreenManager, PauseMenu, PlayerSpawn, ScoreManager, TimerUI y TutorialManager. La carpeta Enemy contiene la lógica de los enemigos, incluyendo BulletPool, EnemyBase, EnemyBlindSpot, EnemyBullet, EnemyDetection, EnemyRhythm y EnemySpiral. En MainMenu se encuentran los scripts relacionados con la interfaz inicial, como AudioSettingsUI, MainMenuHighscore, RebindManager y XRColorAccessibilityController. Finalmente, la carpeta Player reúne los componentes del jugador: PlayerHealth, PlayerHUD, PlayerInput, PlayerMomentum, PlayerMovement y DashAbility.

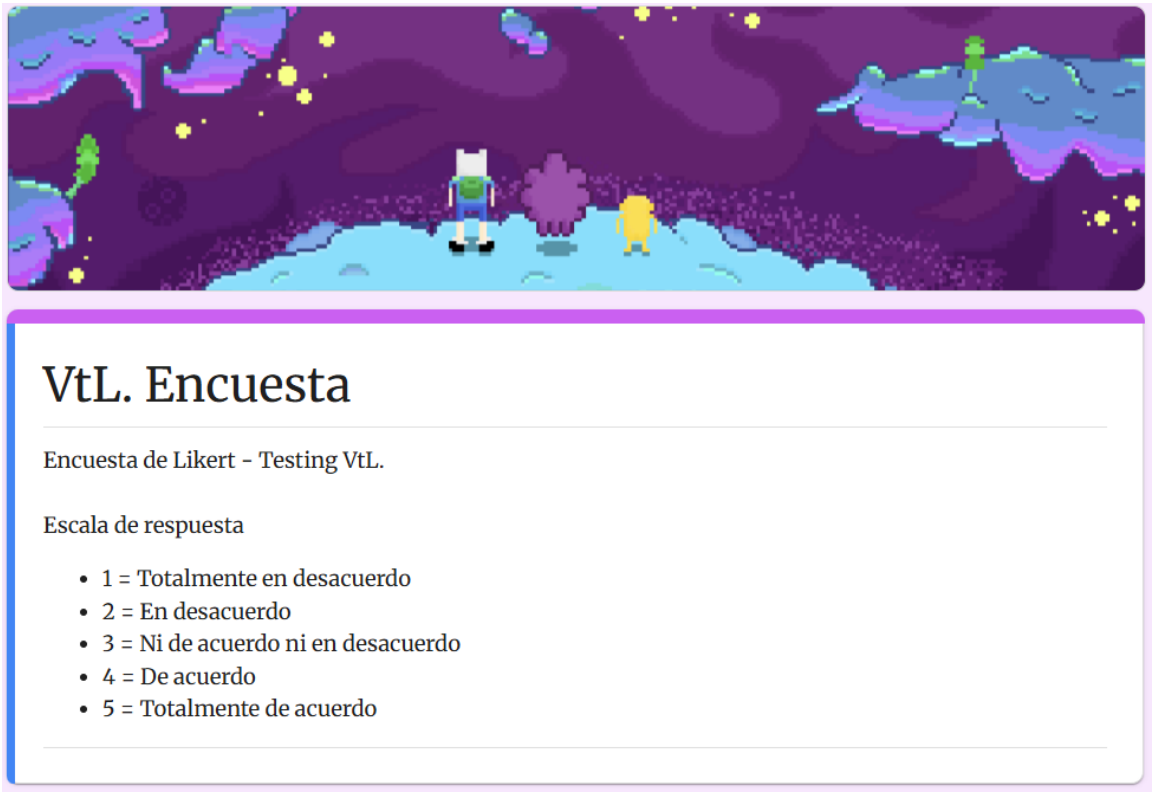
Bibliografía

[1] M. Ann, M. Cuerdo, E. Melcer, E. Carstendottir, and P. F. Biehl, “UNIVERSITY OF CALIFORNIA SANTA CRUZ Fail and Retry: How Challenge Design in Platformer Games Relates to Player Experience and Traits,” 2022.
[2] P. Kaur, M. Jagota, M. Chopra, N. Singhal, and D. Singh, “A Peculiar Review On 2-D Platformer Game Development,” IJARCCE International Journal of Advanced Research in Computer and Communication Engineering Impact Factor 7.39 Vol, vol. 11, 2022, doi: 10.17148/IJARCCE.2022.114135.
[3] J. Wuertz, M. V. Birk, and S. Bateman, “Healthy lies: The effects of misrepresenting player health data on experience, behavior, and performance,” in Conference on Human Factors in Computing Systems - Proceedings, Association for Computing Machinery, May 2019. doi: 10.1145/3290605.3300549.
[4] BJÖRK STAFFAN and HOLOPAINEN JUSSI, PATTERNS IN GAME DESIGN. Hingham, Massachusetts, USA: Charles River Media, 2005.
[5] A. Rashed, S. Shirmohammadi, and M. Hefeeda, “Real-Time Player Engagement Measurement Using Nonintrusive Game Telemetry,” IEEE Open Journal of Instrumentation and Measurement, vol. 4, 2025, doi: 10.1109/OJIM.2025.3555326.
[6] N. Bandeira Romão Tomé, M. Klarkowski, C. Gutwin, C. Phillips, R. L. Mandryk, and A. Cockburn, “Risking Treasure: Testing Loss Aversion in an Adventure Game,” in CHI PLAY 2020 - Proceedings of the Annual Symposium on Computer-Human Interaction in Play, Association for Computing Machinery, Inc, Nov. 2020, pp. 306–320. doi: 10.1145/3410404.3414250.
[7] Peter. McDonald, Run and jump : the meaning of the 2D platformer. The MIT Press, 2024.
[8] M. S. Paul and R. Marques De Albuquerque, “A STUDY OF THE HISTORICAL EVOLUTION OF THE CINEMATIC PLATFORMER SUBGENRE,” Journal of the Interdisciplinary Postgraduate Program in Leisure Studies-UFMG, vol. 28, no. 2, 2025, doi: 10.35699/2447-6218.2025.61135.
[9] G. Smith, U. Santa Cruz Mee Cha, U. Santa Cruz Jim Whitehead, and U. Santa Cruz, “A Framework for Analysis of 2D Platformer Levels.”
[10] R. Richtr and B. Kyseľová, “Swinging Through History: Rope Mechanics in 2D games,” 2026.
[11] C. Chakraborttii and L. Ferreirra, “Towards Generating Surprising Content in 2D Platform Games,” in ACM International Conference Proceeding Series, Association for Computing Machinery, May 2024. doi: 10.1145/3649921.3659848.
[12] A. Johansson, I. Johansson, Z. Nguyen, and J. Ögnelod, “How Procedural Content Generation affects player experience in 2D platformer games Bachelor’s thesis in Computer science and engineering.”
[13] D. Bonilla Carranza et al., “Lecciones Aprendidas de la Literatura: Buenas Prácticas en el Diseño de Mecánicas para Diferentes Géneros de Videojuegos Lessons Learned from the Literature: Best Practices in Designing Mechanics for Different Video Game Genres,” 2025.
[14] M. C. Green, A. Khalifa, P. Bontrager, R. Canaan, and J. Togelius, “Game Mechanic Alignment Theory,” ArXiv, 2021.
[15] D. H. Silver, “Quantifying Skill and Chance: A Unified Framework for the Geometry of Games,” Nov. 2025, [Online]. Available: http://arxiv.org/abs/2511.11611
[16] M. San, J. Martínez, and T. Grau, “Analysis of the behaviour of the player depending on the rewards and the genre of the video game.”
[17] T. Arrias Weiller, “It’s all a Skinner box? Typical Core Loop and Operant Conditioning,” 2026.
[18] L. Chiapello, “Systemic Design and Game Design The equilibrium gameplay loop.”
[19] A. Sarkar and S. Cooper, “Transforming Game Difficulty Curves using Function Composition,” in Conference on Human Factors in Computing Systems - Proceedings, Association for Computing Machinery, May 2019. doi: 10.1145/3290605.3300781.
[20] F. Mortazavi, H. Moradi, and A.-H. Vahabie, “Dynamic difficulty adjustment approaches in video games: a systematic literature review,” Multimed. Tools Appl., vol. 83, no. 35, pp. 83227–83274, 2024, doi: 10.1007/s11042-024-18768-x.
[21] H. Tremblay, D. Ponton, L. Gonzalez, Y. Francillette, and B. Bouchard, “Breaking down the challenge: A comprehensive framework for evaluating enemy difficulty with automated behavior tree analytics,” Entertain. Comput., vol. 57, May 2026, doi: 10.1016/j.entcom.2026.101096.
[22] Team Cherry, “Hollow Knight,” Hollow Knight Official Website.
[23] Jeff Cork, “Hollow Knight Review,” Game Informer.

[24] Tom Marks, “Hollow Knight Review,” PC Gamer.
[25] Maddy Makes Games, “Celeste,” Celeste Official Website.
[26] Team Meat, “Super Meat Boy,” Super Meat Boy Official Website.
[27] Daemon Hatfield, “Super Meat Boy Review,” IGN.
[28] S. Singh and A. Kaur, “Game Development using Unity Game Engine,” in 2022 3rd International Conference on Computing, Analytics and Networks (ICAN), 2022, pp. 1–6. doi: 10.1109/ICAN56228.2022.10007155.
[29] D. Kohli, D. Khurana, B. Singh, A. Kaur, and P. Sachdeva, “Exploring the Capabilities of Unity 3D Gaming Software,” in 2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0, 2024, pp. 1–6. doi: 10.1109/OTCON60325.2024.10687984.
[30] P. R. D. S. Machado, M. A. Souza, and H. C. De Freitas, “Evaluation of Thread Scalability and Compiler Optimization in Parallel Job Execution Using Unity’s Job System,” IEEE Access, vol. 14, pp. 5012–5025, 2026, doi: 10.1109/ACCESS.2026.3651527.
[31] Godot Engine, “License,” Godot Engine.
[32] C. Politowski, F. Petrillo, J. E. Montandon, M. T. Valente, and Y.-G. Guéhéneuc, “Are Game Engines Software Frameworks? A Three-perspective Study,” Sep. 2020, doi: 10.1016/j.jss.2020.110846.
[33] J. Holfeld, “On the relevance of the Godot Engine in the indie game development industry,” Jan. 2024, [Online]. Available: http://arxiv.org/abs/2401.01909
[34] Godot Engine, “Education,” Godot Engine.
[35] B. E. Shelton, J. Scoresby, T. Stowell, M. R. Capell, M. A. Alvarez, and K. C. Coats, “A frankenstein approach to open source: The construction of a 3D game engine as meaningful educational process,” IEEE Transactions on Learning Technologies, vol. 3, no. 2, pp. 85–90, 2010, doi: 10.1109/TLT.2010.3.
[36] J. Tan, Y. Chen, and S. Jiao, Visual Studio Code in Introductory Computer Science Course: An Experience Report. 2024. doi: 10.18260/1-2--48259.
[37] A. Matthew, S. Phadke, and T. Chaudhari, “Real-Time Program Visualization Framework for Visual Studio Code Using Tree-sitter and WebGPU.” [Online]. Available: https://www.ijert.org/
[38] Microsoft, “What is Visual Studio?,” Microsoft Learn.
[39] Microsoft, “Visual Studio IDE Documentation,” Microsoft Learn.
[40] Microsoft, “Visual Studio Community – Free IDE and Developer Tools,” Visual Studio.
[41] Microsoft, “Visual Studio Tools for Unity,” Microsoft Learn.
[42] A. Birillo et al., “Bridging Education and Development: IDEs as Interactive Learning Platforms,” in Proceedings - 2024 1st IDE Workshop, IDE 2024, Association for Computing Machinery, Inc, Aug. 2024, pp. 53–58. doi: 10.1145/3643796.3648454.
[43] Y. T. Lin, C. C. Wu, T. Y. Hou, Y. C. Lin, F. Y. Yang, and C. H. Chang, “Tracking Students’ Cognitive Processes during Program Debugging-An Eye-Movement Approach,” IEEE Transactions on Education, vol. 59, no. 3, pp. 175–186, Aug. 2016, doi: 10.1109/TE.2015.2487341.
[44] JetBrains, “Rider Features,” JetBrains.
[45] JetBrains, “Game Development for Unity – JetBrains Rider Documentation,” JetBrains Documentation.
[46] JetBrains, “Unity Features – JetBrains Rider Documentation,” JetBrains Documentation.
[47] H. L. Fook, M. A. Cagas, A. Aquilino, E. J. Pangan, J. Peralta, and J. R. Clemente, “A Systematic Literature Review on the Application of Scrum in Application Development An Analysis of Scrum in Application Development,” in WSSE 2025 - 2025 The 7th World Symposium on Software Engineering, Association for Computing Machinery, Inc, Apr. 2026, pp. 68–73. doi: 10.1145/3779657.3779668.
[48] G. G. Menekse Dalveren, G. N. Sonmez, and M. Derawi, “Applicability of the Scrum Methodology in Remote Work Environments and the Challenges Encountered,” Electronics (Switzerland), vol. 15, no. 13, Jul. 2026, doi: 10.3390/electronics15132866.
[49] C. Verwijs and D. Russo, “A Theory of Scrum Team Effectiveness,” ACM Transactions on Software Engineering and Methodology, vol. 32, no. 3, Apr. 2023, doi: 10.1145/3571849.
[50] M. O. Ahmad, D. Dennehy, K. Conboy, and M. Oivo, “Kanban in software engineering: A systematic mapping study,” Journal of Systems and Software, vol. 137, pp. 96–113, Mar. 2018, doi: 10.1016/j.jss.2017.11.045.
[51] M. O. Ahmad, J. Markkula, and M. Oivo, “Kanban in software development: A systematic literature review,” IEEE, 2013.

[52] N. Oza, F. Fagerholm, and J. Munch, “How does Kanban impact communication and collaboration in software engineering teams?,” in 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings, 2013, pp. 125–128. doi: 10.1109/CHASE.2013.6614747.
[53] P. Tingling and A. Saeed, Extreme Programming in Action: A Longitudinal Case Study, vol. 4550. 2007. doi: 10.1007/978-3-540-73105-4_27.
[54] A. Mackenzie and S. Monk, “From Cards to Code: How Extreme Programming Re-Embodies Programming as a Collective Practice,” 2004.
[55] A. A. Nugroho, R. Ferdiana, and R. S. Hartanto, “Implementation of Design Patterns on Unity Components to Increase Reusability and Game Speed Development,” 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), IEEE, pp. 177–182, 2020. doi: 10.1109/ICIMCIS51567.2020.9338715.
[56] A. M. Alzahrani and A. Alalwan, “Design Patterns for Developing Maintainable and Reusable Components in Unity Game Engine,” 2021 International Conference on Computer and Information Sciences (ICCIS), IEEE, pp. 1–6, 2021. doi: 10.1109/ICCIS54484.2021.9681762.

Anexos

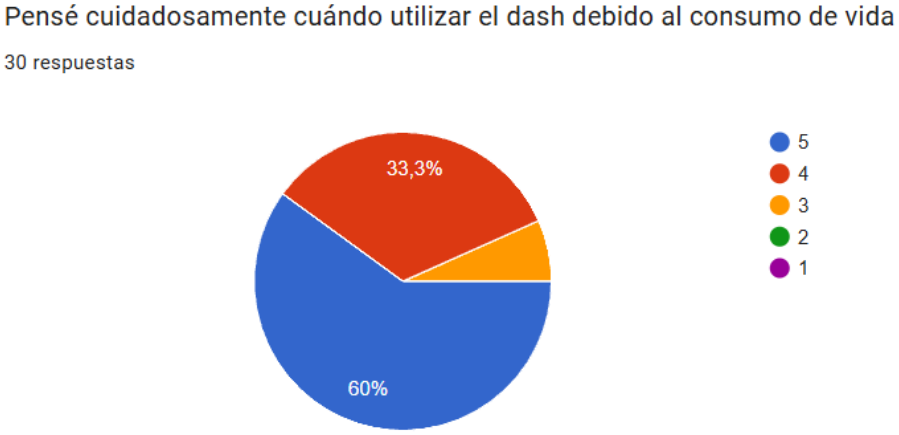


Anexo 1: Inicio de la encuesta de Likert sobre el prototipo.

Anexo 2: Número de respuestas y resultados de la encuesta en la pregunta número 1.

Anexo 3: Resultados de la encuesta en la pregunta número 2.

Anexo 4: Resultados de la encuesta en la pregunta número 3.

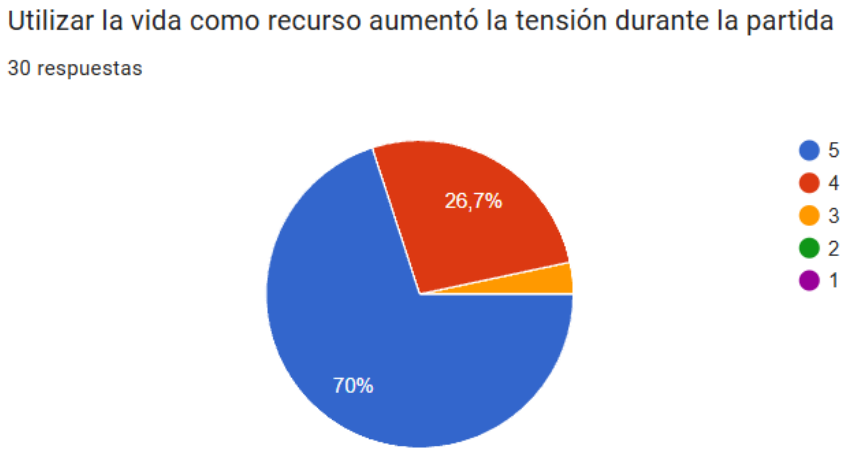


Anexo 5: Resultados de la encuesta en la pregunta número 4.

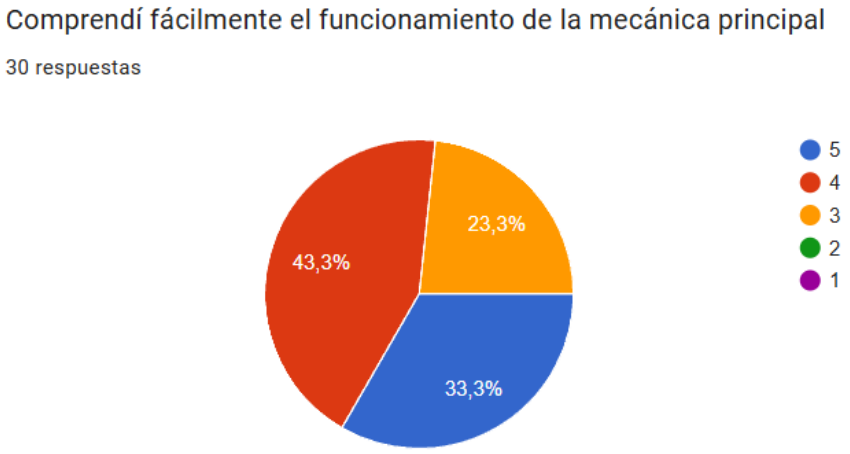
Anexo 6: Resultados de la encuesta en la pregunta número 5.

Anexo 7: Resultados de la encuesta en la pregunta número 6.

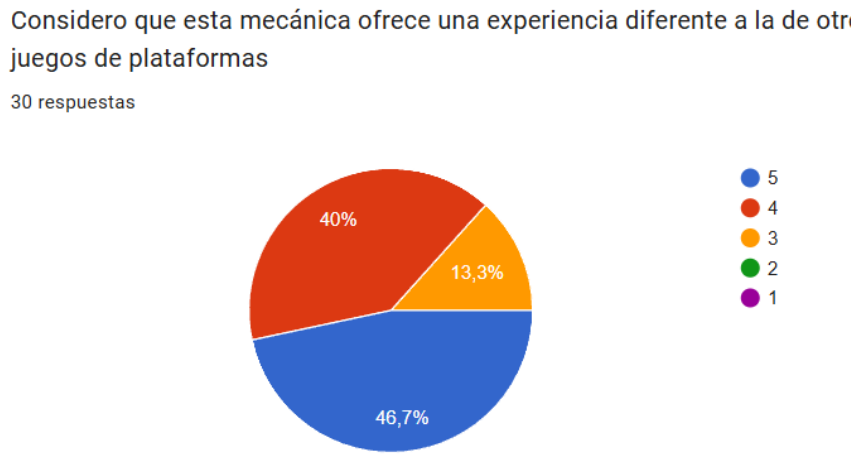
Anexo 8: Resultados de la encuesta en la pregunta número 7.



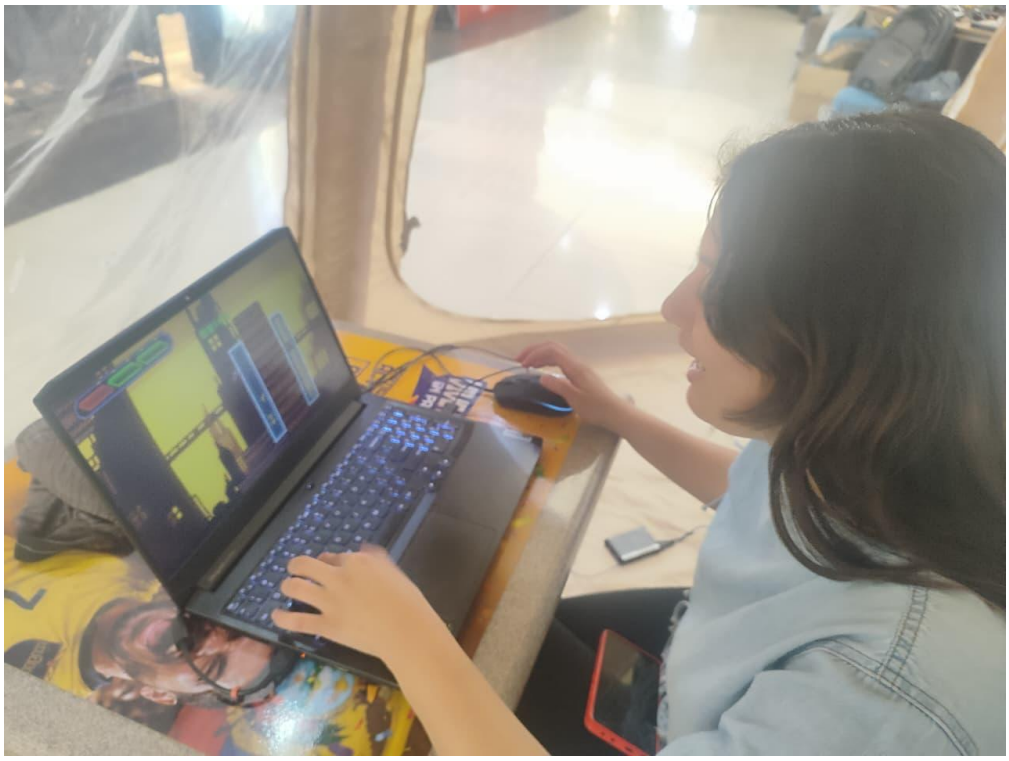
Anexo 9: Resultados de la encuesta en la pregunta número 8.



Anexo 10: Resultados de la encuesta en la pregunta número 9



Anexo 11: Resultados de la encuesta en la pregunta número 10



Anexo 12: Testing del videojuego sobre todos los apartados del videojuego.