UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACION E INNOVACION TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

Desarrollo de inteligencia artificial para personajes no jugables e implementación del control del personaje principal en entornos de videojuego

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

AUTOR: CESAR MATEO RAMÍREZ MALLA
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 COMPUTACION E INNOVACION TECNOLÓGICA

CARRERA DE REALIDAD VIRTUAL Y VIDEOJUEGOS

Desarrollo de inteligencia artificial para personajes no jugables e implementación del control del personaje principal en entornos de videojuego

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

AUTOR: CESAR MATEO RAMÍREZ MALLA

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

Cesar Mateo Ramírez Malla portador de la cédula de ciudadanía Nº **0106857535**. Declaro ser el autor de la obra: **"Desarrollo de inteligencia artificial para personajes no jugables e implementación del control del personaje principal en entornos de videojuego"**, 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, **1 de septiembre de 2026**

F: .....

Cesar Mateo Ramírez Malla

C.I. **0106857535**

Universidad Católica de Cuenca

CERTIFICADO

Certificado

Certifico que el presente Trabajo de Investigación fue desarrollado por **Cesar Mateo Ramírez Malla** portador de la cédula de ciudadanía N.º **0106857535** con el tema **"Desarrollo de inteligencia artificial para personajes no jugables e implementación del control del personaje principal en entornos de videojuego"**, bajo mi supervisión.

F: .....

Ing. Jheyson Steven Gaona Pineda, Mtr.

Docente - Tutor

Dedicatoria

Con todo mi amor y gratitud dedico este logro a todas las personas que han guiado mi camino y me motivaron para alcanzar esta meta.

A mi madre Grace Malla y mis abuelos Cecilia Sánchez y Luis Malla que siempre cuidaron de mí, me criaron, me formaron como persona y me dieron la mano cuando el mundo se me venía encima. Su amor, esfuerzo y apoyo incondicional han sido los pilares fundamentales de mi vida y la fuerza que me permitió lograr esta meta.

Especialmente dedico este triunfo a mi padre Cesar Ramírez que siempre me brindó las oportunidades que él no pudo tener. Gracias, papi, por todas tus enseñanzas, esfuerzo y valores que inculcaste en mí. Aunque siempre dices que he logrado cosas que tú no pudiste alcanzar, la verdad es que nunca dejaré de verte como un ejemplo de lo que quisiera llegar a ser. Este logro también es tuyo papá.

Agradecimiento

Agradezco primeramente a Dios por darme la fuerza y sabiduría en los momentos más difíciles de mi vida y permitirme lograr esta meta.

Agradezco a mi madre por ser un ejemplo de perseverancia y mostrarme que no importa que tan difícil sea el camino siempre habrá una manera de cumplir con mis objetivos. Gracias, mami, porque todo tu amor, esfuerzo y dedicación me han convertido en la persona que soy hoy.

A mi padre, por enseñarme a domar las adversidades y demostrarme que cada día es una oportunidad para crecer y ser mejor, gracias por luchar día a día por nuestra familia.

A mi mami Ceci por criarme, cuidarme y amarme como un hijo más que como a un nieto, agradezco el amor que me has dado en los momentos que más te he necesitado, tu apoyo en todo lo que me he propuesto y acompañarme en cada etapa de mi vida.

A mi abuelo, porque sin su ayuda no habría alcanzado esta meta. Gracias por tu apoyo desinteresado, por cada uno de tus consejos y sabias palabras, las cuales me fortalecieron y me forjaron durante este largo camino.

A mi hermana Paola, gracias ñaña por nunca dejarme solo y acompañarme en cada momento de mi vida. Saber que podía contar con tus consejos y apoyo me dieron la fuerza que muchas veces necesitaba para seguir avanzando.

Por último, un agradecimiento a mi tutor de tesis, Ing**.** Jheyson Gaona, por su orientación, paciencia y apoyo durante el desarrollo de este trabajo.

Resumen

En los videojuegos, los personajes constituyen el eje central de la experiencia interactiva, dividiéndose en jugables, controlados por el usuario, y no jugables, gestionados por el sistema para dar vida y contexto al mundo virtual mediante comportamientos inteligentes basados en reglas. El sistema de combate, por su parte, es una mecánica clave en los juegos de acción-aventura, basada en ataques directos, esquivas y habilidades especiales controladas en tiempo real por el jugador. El presente proyecto implementa el diseño y desarrollo de la inteligencia artificial y las mecánicas del jugador para el videojuego "Ashes of Faith", desarrollando la lógica de programación tanto para el personaje jugable como para los no jugables. Para el personaje jugable se abarcan los sistemas de: (i) movilidad, (ii) combate, (iii) gestión de vida, (iv) habilidad especial e (v) interacción. Para los personajes no jugables se implementan máquinas de estados finitos, utilizando Scriptable Objects como estructura de datos para la configuración y gestión de estados. Para el desarrollo se empleó Unity, utilizando lenguaje de programación C#, bajo la metodología ágil Scrum, permitiendo gestionar de forma iterativa e incremental cada entregable dentro de cada sprint. Los resultados se obtuvieron mediante un cuestionario con escala de Likert, evaluando los criterios de gestión y control de los personajes jugables, no jugables y el sistema de combate. Asimismo, se realizaron pruebas de caja negra en Unity para evaluar la escalabilidad de los enemigos implementados a través de Scriptable Objects, evidenciando un sistema robusto, escalable y coherente con mecánicas jugables.

Palabras clave: *Inteligencia artificial, Máquina de estados finitas, Scriptable Objects, Unity.*

Abstract

In video games, characters constitute the central focus of the interactive experience; they are divided into playable characters, controlled by the user, and non-playable characters, managed by the system to bring life and context to the virtual world through intelligent rule-based behaviors. The combat system, meanwhile, is a key mechanic in action-adventure games, based on direct attacks, dodges, and special abilities controlled in real time by the player. This project involves the design and development of artificial intelligence and player mechanics for the video game "Ashes of Faith," developing the programming logic for both playable and non-playable characters. For the playable character, the following systems are covered: (i) mobility, (ii) combat, (iii) health management, (iv) special abilities, and (v) interaction. For non-playable characters, finite state machines are implemented, using Scriptable Objects as the data structure for state configuration and management. The development process was carried out using Unity and the C# programming language under the Scrum agile methodology, which enabled the iterative and incremental management of each deliverable within each sprint. The results were obtained through a Likert-scale questionnaire that evaluated the management and control criteria of the playable and non-playable characters, as well as the combat system. In addition, black-box testing was conducted in Unity to evaluate the scalability of enemies implemented via Scriptable Objects, demonstrating a robust, scalable system consistent with gameplay mechanics.

Keywords: *Artificial intelligence, Finite-state machine, Scriptable Objects, Unity.*

Índice de contenido

Índice de tablas

Índice de ilustraciones

Introducción

Actualmente los videojuegos se consideran experiencias interactivas donde es fundamental el uso de controles, sistema de combate y comportamientos en general de los personajes para brindar una jugabilidad satisfactoria. En los videojuegos una programación mal implementada puede ocasionar controles incomodos, sistema de ataques repetitivos y personajes directamente no pueden interactuar de forma fluida con el usuario. Estas limitaciones afectan directamente la experiencia del juego, por lo cual es necesario integrar de manera eficiente y organizada las mecánicas del personaje, interacciones e inteligencia artificial.

Por este motivo el presente trabajo aborda el diseño y programación de los sistemas del videojuego “Ashes of Faith”. El desarrollo consta de controles y sistema de combate del personaje principal, sistema de vida y muerte, la habilidad especial, interacción con NPCs (personajes no jugables) y el comportamiento general de los agentes virtuales. Para controlar estos últimos se utiliza una máquina de estados finita capaz de administrar los comportamientos de la IA (inteligencia artificial).

El prototipo se desarrolló en Unity mediante el lenguaje de programación C#, utilizando componentes del motor como Animator, CharacterController, NavMeshAgent y Scriptable Objectss. El proceso se organizó bajo la metodología Scrum y se dividió en cinco sprints para desarrollar, implementar y comprobar progresivamente el funcionamiento del sistema. Finalmente, evaluamos la ejecución de los sistemas mediante una prueba a diferentes usuarios utilizando escala de Likert.

A continuación, una breve descripción de los capítulos que componen este trabajo:

Capítulo I: En este capítulo se desarrolla el marco referencial de la investigación. Se plantea la problemática relacionada al funcionamiento de los personajes tanto jugable como no jugable. Asimismo, se presenta la justificación, objetivo general y específicos, alcance y las limitaciones del trabajo.

Capítulo II: En este capítulo redactamos el marco teórico, es la investigación previa al desarrollo del prototipo. Se abordan temas como IA en videojuegos, máquinas de estado finitas, sistema de combate y jugabilidad. De igual forma se analizan trabajos relacionados, tecnologías y metodologías de trabajo.

Capítulo III: Se presenta el manual de usuario, en el cual se detallan especificaciones, configuración de controles, los elementos de la interfaz y el procedimiento necesario para su instalación y ejecución.

Capítulo IV: Se detalla el desarrollo de los sistemas para “Ashes of Faith”. Se presentan las herramientas empleadas, planificación bajo el Producto Backlog, la ejecución de los cinco sprints y la integración de los sistemas al videojuego. Finalmente se exponen las pruebas y resultados obtenidos.

Capítulo I

1. Marco referencial

1.1. Contextualización del problema

Actualmente en la industria de los videojuegos, el desarrollo de sistemas de IA se ha vuelto un componente fundamental para la edificación de experiencias interactivas y envolventes. Los videojuegos actuales no solo funcionan con gráficos atractivos o mecánicas nuevas, sino también con la implementación de NPCs que contengan comportamientos creíbles y fluidos, la interacción que tiene el jugador con el entorno y el comportamiento del personaje principal dentro del juego. Varios estudios dictan que la IA en videojuegos permite hacer una simulación de decisiones, generan respuestas dinámicas y adaptadas a las acciones del jugador [1], [2].

Dentro del contexto de nuestro videojuego “Ashes Of Faith”, identificamos desafíos específicos con relación a la implementación de IA para lograr regular el comportamiento de los NPCs puesto que la propuesta depende en gran parte del equilibrio que hay entre combate, exploración y el control del personaje. No obstante, sin un sistema adecuado las interacciones pueden volverse monótonas o interrumpir el flujo de juego. De igual manera un control mal ejecutado del personaje principal podría arruinar completamente la inmersión del usuario. Muchos enfoques de diseño sobresaltan la importancia de equilibrar las mecánicas, interacciones y experiencia del jugador [3].

1.2. Planteamiento del problema

A pesar de lo avanzada que está la IA en los videojuegos, aun muchos sistemas tienen limitaciones para generar comportamientos creíbles para los NPCs. En varios

casos los NPCs se limitan únicamente a acciones predefinidas patrones de comportamiento repetitivos y obsoletos que terminan afectando el desafío y jugabilidad. Además, al momento de implementar al personaje principal muchos proyectos no logran adaptarlo al sistema de IA y entornos, generando interacciones poco fluidas. Investigaciones acerca del tema nos sugieren que el uso de árboles de comportamientos o máquinas de estado finitas ayudan a generar respuestas más dinámicas y naturales contribuyendo a una experiencia más inmersiva y favorable para el usuario [4], [2]. No obstante, la implementación de este sistema representa un desafío técnico y de diseño ya que requerimos lograr un equilibrio entre el comportamiento con el rendimiento del sistema y la experiencia del jugador [2].

Factores técnicos y de diseño suele limitar bastante la implementación de IA, una de las principales causas es la dependencia de sistemas simples, como un mal desarrollo de la máquina de estados finita o sistemas que no permiten una adaptabilidad correcta frente a las distintas acciones del jugador. Además, la falta de integración de IA con las mecánicas del personaje principal genera inconsistencias en la experiencia puesto que el jugador no llega a percibir una respuesta adecuada ante las acciones que realiza. Estudios especializados destacan la correcta implementación de estructuras más avanzadas [2].

A consecuencia de estas limitaciones, generamos una experiencia de juego menos inmersiva, donde los NPCs dejan de tener coherencia y dejan de representar un reto para el jugador y este mismo no logra integrarse adecuadamente al sistema del juego. Esto afecta directamente a la calidad, reduciendo el interés y la motivación del usuario para avanzar en la historia. Bajo este concepto se dicta que el

comportamiento inteligente de los agentes virtuales determina la jugabilidad y experiencia interactiva [2].

Para el desarrollo del proyecto nos resulta necesario una elaboración precisa del sistema de IA específicamente con las interacciones del jugador. el problema lo delimitamos a el diseño e implementación de comportamientos inteligentes que manejen de la mejor manera una toma de decisiones, adaptabilidad, coherencia y fluidez con respecto al personaje principal.

La mala implementación de este sistema puede romper completamente las expectativas del jugador en temas de inmersión, desafío y dinamismo. Diversos estudios señalan que la calidad de la IA junto con la correcta implementación del control de jugador, influyen en el realismo y satisfacción del usuario [5]. Por lo tanto, identificar esto problemas permiten crear una base para desarrollar soluciones más eficientes.

1.3. Formulación del problema

¿Cómo diseñar e integrar un sistema de IA basado en máquinas de estado finitas (FSM) para el comportamiento de los NPCs junto con las mecánicas del personaje principal en el videojuego “Ashes of Faith”?

1.4. Justificación

El sistema de IA dentro de un videojuego se ha vuelto fundamental para brindar una experiencia inmersiva y de calidad. Actualmente gran parte del entretenimiento de un videojuego funciona a partir de desafíos, entornos dinámicos y personajes que respondan de manera coherente a todo esto. En este contexto implementar un sistema de IA nos permite simular comportamiento autónomo y

generar interacciones más realistas [1]. Asimismo, esta evidenciado que técnicas modernas de IA mejoran la inmersión al reducir patrones monótonos en los personajes [2].

A pesar de las mejoras en los actuales videojuegos aún es muy común encontrar IA limitada, NPCs con interacciones básicas o enemigos con patrones comunes y predecibles para el jugador esto reduce en gran parte la atención del usuario y la dificultad del juego, bajo esta situación lo mejor es optar por árboles de comportamiento que permiten mejorar la toma de decisiones para generar experiencias más realistas [2]. Aparte el uso de diseño de IA genera sistemas más escalables y con capacidad de adaptarse a situaciones espontaneas dentro del juego[6].

El desarrollo de este proyecto nos da la oportunidad de aplicar e integrar técnicas de IA. Adecuadas al jugador, la correcta implementación de agentes virtuales nos permitirá ofrecer interacciones dinámicas, sistemas de combate estratégicos, difíciles y satisfactorios que son de mucha relevancia en videojuegos de tipo souls o metroidvania. Esto nos indica que el sistema de mecánicas e interactividad mejoran la experiencia del usuario [3].

Finalmente, este proyecto tiene una importancia tanto académica como tecnológica, al abordar el desarrollo de IA aplicada a videojuegos. La implementación de estos sistemas nos permitirá fortalecer las competencias en ingeniería de software y videojuegos y la comprensión de los agentes virtuales y como estos pueden mejorar la experiencia de usuario. Investigaciones aclaran que el estudio y aplicación de IA contribuye a la experimentación y validación de técnicas de comportamiento autónomas [6].

1.5. Objetivo General

Diseñar la arquitectura de programación del videojuego “Ashes of Faith”, mediante el uso de máquinas de estado finitas para el comportamiento de los NPCs y la estructura de las mecánicas del personaje principal, con el propósito de integrar ambos sistemas dentro de un entorno jugable funcional.

1.6. Objetivos Específicos

- Analizar los sistemas de IA basados en FSM aplicados en videojuegos, con énfasis en el comportamiento de NPCs, para identificar técnicas y enfoques adecuados para el desarrollo del proyecto.
- Diseñar la arquitectura y la vinculación del sistema de IA basado en máquinas de estados finitos (FSM), mediante la definición de estados, transiciones y eventos, para establecer los comportamientos e interacciones de los NPCs dentro del videojuego.
- Implementar los sistemas de IA para NPCs, basados en máquinas de estados finitos (FSM), utilizando el motor de desarrollo Unity y herramientas como Scriptable Objects para la gestión de estados y comportamientos, con el fin de permitir respuestas dinámicas y coherentes frente a las acciones del jugador.
- Programar el control, interacción y habilidades del personaje principal, garantizando su integración con el sistema inteligente de los NPCs.
- Evaluar el funcionamiento de los sistemas implementados mediante pruebas de jugabilidad, centradas en el comportamiento del personaje principal y de los personajes no jugables (NPCs), con el fin de validar su desempeño en

función de los comportamientos definidos y las respuestas esperadas dentro del entorno del videojuego.

1.7. Alcance del proyecto

El proyecto está enfocado en el desarrollo e implementación de IA aplicado a NPCs dentro del videojuego “Ashes of Faith”, al igual que las mecánicas de control del personaje principal. Específicamente se aborda los sistemas de comportamiento inteligente que permiten diálogos, sistemas de combate para NPCs hostiles, incluyendo el sistema de vida y muerte de los mismos.

De igual manera se contempla la implementación de las mecánicas fundamentales para el personaje jugable tales como movimiento, habilidades, interacción y sistema de vida y muerte, garantizando que el mismo funcione de una manera fluida y responda correctamente a las acciones que el usuario requiere.

Como resultado se ha planteado la construcción de un entorno de prueba funcional, el cual pueda validar el funcionamiento de todos los sistemas implementados permitiéndonos su correcta funcionalidad entre la IA y el comportamiento del personaje principal.

1.8. Delimitaciones

El proyecto se limita al área de programación únicamente, por esta razón descartamos la creación de modelos 3D, texturas, animaciones, diseño de escenarios, narrativa, diseño de niveles ni recursos audiovisuales, debido a que estos elementos ya corresponden a otras áreas involucradas al desarrollo del prototipo.

De igual manera en el área de programación no se contempla un sistema de IA basado en aprendizaje ni otras técnicas de comportamiento autónomo diferentes a las ya planteadas anteriormente.

1.9. Beneficiarios

Los beneficiarios directos del proyecto son los integrantes del equipo involucrados en la creación y desarrollo del videojuego, debido a que la programación del personaje jugable, personajes no jugables, la IA y la interfaz principal tiene una base funcional para la integración de los componentes del juego restantes. De igual modo, se considera a los estudiantes interesados en aprender sobre el desarrollo de videojuegos como beneficiarios directos.

Los beneficiarios indirectos son los usuarios que interactúen con el prototipo, los mismos que lleguen a experimentar las mecánicas, sistemas de combate y los comportamientos de IA desarrollados. De igual forma la institución académica y la comunidad interesada en desarrollo de videojuegos pueden beneficiarse del proyecto al disponer de un caso práctico evidente.

1.10. Metodología General

El presente proyecto se desarrolló utilizando una metodología de tipo ágil denominada Scrum, esta misma permitió trabajar mediante ciclos interactivos e incrementales. Este enfoque facilito la distribución de tareas, el seguimiento del avance y la adaptación a los cambios que surgieron durante el proceso de desarrollo.

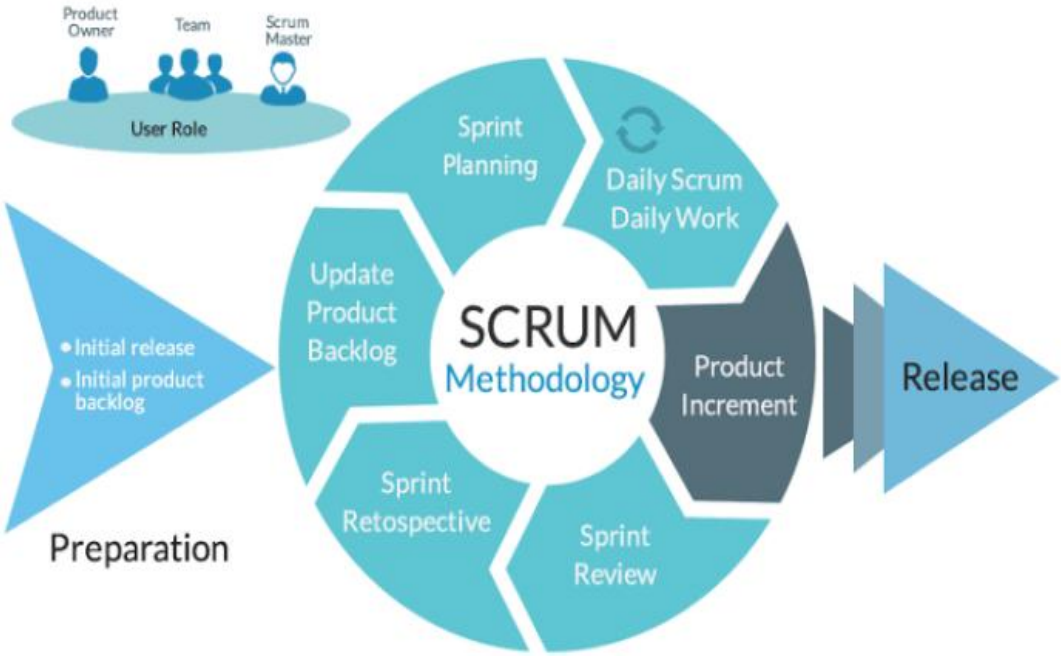
Para la implementación se utiliza el motor Unity y el lenguaje de programación C#. el desarrollo de IA se construye a partir de FSM para controlar los

comportamientos de los NPCs, mientras que las mecánicas del personaje principal se programaron de manera progresiva dentro del prototipo.

Como resultado, se obtendrá un entorno funcional que permite evaluar el comportamiento del personaje principal y NPCs dentro de un escenario controlado, esto nos permite verificar la correcta ejecución de las interacciones entre personaje jugable y personajes no jugables. La ilustración 1 nos muestra el ciclo de trabajo de la metodología scrum

Ilustración 1

Metodología Scrum



Nota: Obtenido de [7]

1.11. Resultados esperados

Los resultados esperados se organizan bajo cinco sprints, definidos de acuerdo a los sistemas principales del proyecto.

- **Sprint 1: Control del personaje principal**

  En este sprint se programó al personaje principal incluyendo su diseño de mecánicas como movimiento, desplazamiento y esquive, implementación de las mismas e interacciones principales.

- **Sprint 2: Sistema de combate del personaje**

  En esta fase se implementaron las mecánicas de combate incluyendo ataques, sistema de combos, habilidad especial, recepción de daño y sistema de vida y muerte.

- **Sprint 3: Sistema de interacción y diálogos**

  Este sprint se centró en la interacción con NPCs no hostiles mediante un sistema de diálogos permitiendo iniciar conversaciones y mostrar la información que corresponde a cada interfaz.

- **Sprint 4: Inteligencia artificial mediante FSM**

  En este sprint se implementa el comportamiento de los enemigos mediante FSM integrando estado de reposo, patrulla, persecución, retorno y otras respuestas asociadas al comportamiento del agente.

- **Sprint 5: Sistema de combate de enemigos**

  Finalmente se integró el combate de los NPCs enemigos incluyendo la detección del jugador, ejecución de ataques, recepción de daño y comunicación con los estados de la FSM.

Capítulo II

2. Marco teórico

2.1. Antecedentes investigativos

En la presente sección se analizan distintos trabajos relacionas para obtener insumos teóricos y técnicos, se revisan proyectos vinculados a la IA, diseño de mecánicas, personajes jugables y personajes no jugables con el objetivo de identificar métodos y herramientas de cada propuesta.

2.1.1. Dark Souls III

Dark Souls III es un videojuego de acción y rol desarrollado por FromSoftware y lanzado en 2016. Este destaca por su sistema de combate estratégico, exploración interconectada y elevado nivel de dificultad. Su diseño integra un sistema de progresión, IA y NPCs, mecánicas de exploración y gestión de recursos. De acuerdo con Guzsvinecz [8], la fórmula de los videojuegos tipo souls es centrarse en la dificultad, puntos de control y la narración ambiental. Gracias a estas características se lo considero como un referente para el desarrollo del presente proyecto.

Seguidamente se enlistarán temas de interés para el desarrollo del proyecto

- **Sistema de combate:** Dark Souls III cuenta con un sistema de combate enfocado en la estrategia, cada acción del jugador requiere ser preciso y conllevar una planificación. Las mecánicas de ataque, esquive, bloqueo y contraataque recompensan la estrategia que se use en combate mientras que la mala ejecución de los mismos penaliza al jugador con perdida considerable de salud. Además, cada enemigo cuenta con patrones diferente de ataque lo

que obliga al jugar a estudiar detenidamente a cada enemigo antes de poder avanzar, este diseño incrementa el nivel de desafío y favorece a la experiencia de usuario.

- **IA de enemigos:** Este título desarrolla una IA que ofrece enfrentamientos dinámicos usando comportamientos adaptativos a las acciones del jugador. Cada enemigo utiliza distintos estados de comportamiento como patrullar, detectar, atacar, entre otros, permitiendo que las acciones tengan relación con el contexto del juego. Este enfoque nos muestra lo importante que es tener un diseño bien estructurado de comportamiento.
- **NPCs e interacciones:** Los NPCs desarrollan un papel importante, no están únicamente de decoración, sino que proporcionan información valiosa y un sistema de comercio que mejoran el equipamiento del jugador, las interacciones con los personajes se realizan mediante un sistema de diálogo que va evolucionando conforme se avance en la historia permitiendo que sus acciones se adapten a la partida. Este sistema fortalece la inmersión del jugador.
- **Exploración y diseño de niveles:** La exploración es un factor central dentro de la experiencia del juego, ya que la misma incentiva al jugador a explorar todos los espacios disponibles en busca de nuevos caminos o mejoras para su aventura, el diseño de nivel incluye atajos o zonas de riesgo con sus respectivas recompensas. Promoviendo una progresión avanzada.
- **Sistema de salud y recursos:** La salud y recursos dentro de *dark souls III* está diseñado para dar emoción a la partida ya que necesitas administrar tus

recursos de forma estratégica gracias a la limitación de los mismos, obligando al jugador a gestionar sus recursos de manera inteligente.

- **Progresión del jugador:** la progresión del jugador se obtiene bajo un sistema denominado obtención de almas, las cuales funcionan como recurso recolectable para mejorar atributos y fortalecer las capacidades preestablecidas. Conforme avanza el jugador obtiene distintos tipos de armas o armadura que se complementan a sus habilidades. Este sistema de mejora se lo llama progresión gradual y motiva al jugador a medida que progresa en la experiencia.
- **Diseño del desafío:** Este videojuego se caracteriza por su sistema alto de dificultad, pero sin llegar a cansar al usuario, El juego funciona con un sistema de dificultad progresiva y exige al jugador a desarrollar habilidades como la observación de patrones, gestionar sus recursos y aprender mediante prueba y error para eliminar a los enemigos.

En la ilustración 2 podemos observar el juego Dark Souls III.

Ilustración 2

Presentación visual del Dark Souls III

Nota: Obtenido de ArsTechnical *[9]*

2.1.2. Hollow Knight

Hollow Knight es un metroidvania donde destaca la exploración, fue desarrollado por Team Cherry y publicado en 2017. Uno de los apartados que más llamó la atención de los usuarios es su sistema de combate basado en la precisión, Además de su sistema de progresión no lineal, una progresión basada en la exploración y adquisición de habilidades para seguir desbloqueando distintos caminos. Asimismo, el videojuego incorpora distintos enemigos y NPCs no hostiles que fortalecen la inmersión y convierten la exploración en un desafío y enriquecen el mundo abierto del mismo. Wang et al. [10] describen a Hollow Knight como una experiencia con narrativa compleja y misteriosa construida mediante la interpretación del jugador a las distintas pistas que se dan a través de la historia. Debido a estas características Hollow Knight proporciona varios apartados que se volvieron referentes en la industria de los videojuegos.

Algunos de sus apartados más importantes son los siguientes.

- **Sistema de combate:** Este se caracteriza por ofrecer enfrentamientos basados en la precisión, velocidad, reflejos y dominio de las habilidades del personaje principal. El jugador utiliza ataques cuerpo a cuerpo, esquiva y desbloquea habilidades especiales que se van desbloqueando a medida que avance en el juego. Cada combate exige reacciones rápidas del jugador lo que lo transforma en un juego frenético, ya no depende de las tácticas al atacar sino de lo rápido que sea el jugador al momento de conectar con los golpes y esquivar los de sus enemigos.
- **IA de enemigos:** Este tipo de enemigos funcionan con distintos tipos de estados, como patrullar la zona, atacar al personaje en cuanto este entre en su campo de visión o realizar ataques distintos según el tipo de enemigo que sea. Este tipo de IA refuerza la utilización de máquinas de estados finitas para obtener un comportamiento coherente y adaptable.
- **NPCs e interacción:** Los NPCs son esenciales en Hollow Knight, al ser un mundo abierto los NPCs ayudan a que los escenarios no se sientan vacíos, Los mismos también contribuyen con el desarrollo narrativo, comercio de objetos o expansión del mapa. Estas interacciones incentivan al jugador a descubrir nuevas zonas y dialogar con los NPCs para obtener más información sobre la historia o llevarse una recompensa extra.
- **Exploración y diseño de niveles:** La exploración es uno de los puntos fuertes de este título, al no tener una progresión lineal el jugador se ve obligado a explorar el mundo en busca de progresar por su cuenta desbloqueando

habilidades o encontrando nuevas rutas para seguir, la exploración se refuerza mediante backtracking bloqueando caminos que posteriormente el jugador podrá desbloquear siempre y cuando haya desbloqueado la habilidad especifica que le permita evitar el obstáculo que le impedía el paso, en resultado a esto el diseño favorece a la exploración que recompensa el descubrimiento.

- **Sistema de salud y recursos:** Recursos y salud se combinan ambos siendo administrados por un recurso llamado alma, el cual se va regenerando a medida que vayamos golpeando a los enemigos. Esta mecánica obliga al jugador a decidir estratégicamente si debería regenerar vida o por el otro lado gastar alma para potenciar su ataque con habilidades especiales. De esta manera el juego introduce una gestión equilibrada de recursos y alienta la planificación del usuario.
- **Progresión del jugador:** *Hollow Knight* basa su progresión en el sistema de habilidades lo que significa que si quieres avanzar en la historia tienes que desbloquear nuevos caminos y para hacerlo primero tienes que obtener habilidades que te permitan hacerlo, Conforme el jugador avance se ira fortaleciendo lo que significa que podrá enfrentar enemigos que no pudo derrotar anteriormente, esto genera una sensación de crecimiento gradual permitiendo que las habilidades no solo tengan impacto en el desbloqueo de rutas sino también en el combate.
- **Diseño de desafío:** El sistema de desafío es progresivo según se vaya avanzando más difícil serán los combates y el desbloqueo de rutas, pero

también será más fuerte el personaje principal, lo cual equilibra la experiencia. La dificultad de los enemigos se basa en un sistema de patrones rápidos que desafían los reflejos del usuario. Esto hace que el jugador requiera de observación, practica y dominio de habilidades para derrotarlo, esto genera satisfacción convirtiendo cada enemigo derrotado en una recompensa dentro de la experiencia del juego.

A continuación la ilustración 3 nos muestra el videojuego Hollow Knight.

Ilustración 3

Presentación visual de Hollow Knight

Nota: Obtenido de Vandal *[11]*

2.1.3. Death's Door

Desarrollado por Acid Nerve y publicado en 2021 Death's Door es un videojuego que combina un sistema de combate en tiempo real con una exploración entre escenarios que se interconectan entre ellos. La propuesta destaco por sus mecánicas de progresión, NPCs y su revolucionario diseño de niveles que incentivan la exploración. Brandse [12] incluye a este título en un análisis de patrones de flujo de niveles y determina que el orden en el que se presentan los desafíos influye en la experiencia del jugador.

A continuación, sus puntos importantes.

- **Sistema de combate:** El combate combina ataques cuerpo a cuerpo, ataques a distancia, esquives y el uso de habilidades, el combate funciona mediante la observación de patrones de ataque del enemigo y utilizar el que mejor le convenga al jugador. Asimismo, el gran número de enemigos y jefes obliga al jugador a cambiar su estrategia y adaptarse a cada entidad hostil que este enfrentando. Este sistema de combate muestra un enfoque de decisiones para utilizar el ataque más oportuno.
- **IA de enemigos:** La IA de *Death's Doors* utiliza comportamientos diferentes según el tipo de enemigo. En el combate, los enemigos realizan acciones como desplazamiento, persecución, ataque y reposicionamiento, en virtud de la ubicación y acciones del jugador. Cada enemigo cuenta con diferentes patrones de ataque y tiempos específicos para ejecutarlos lo que diversifica los enfrentamientos y hace que el juego no se vuelva monótono.
- **NPCs e interacción:** Los NPCs complementan la experiencia del juego apoyando al jugador mediante diálogos, información adicional o desarrollo narrativo. A lo largo de la aventura los NPCs brindan información importante al jugador lo que contribuye directamente a la inmersión del juego y fortalece la conexión entre la jugabilidad y la narrativa.
- **Exploración y diseño de niveles:** Uno de los puntos más destacados del título este funciona por un conjunto de escenarios interconectados que combinan varias rutas principales con secretos o caminos alternativos, este diseño incentivo al jugador a explorar con el objetivo de conseguir mejoras y objetos. Asimismo, estos escenarios cuentan con una distribución equilibrada entre

enemigos, obstáculos y puntos de interés para el jugador esto refuerza el ritmo de juego manteniéndolo fluido y entreteniendo al usuario mientras explora.

- **Sistema de salud y recursos:** este está diseñado para usarse de una manera estratégica durante el combate y la exploración. La salud se recupera mediante elementos que se encuentran en el escenario, mientras que las habilidades consumen recursos que se obtienen de forma planificada para derrotar al enemigo de turno. Esta combinación fortalece la relación entre riesgo, planificación y supervivencia, incentivando al jugador a gestionar de forma adecuada sus recursos.
- **Progresión del jugador:** El jugador puede fortalecer sus atributos relacionados con la salud, fuerza, velocidad y magia mediante un sistema de almas que funcionan como moneda del juego. Asimismo, va desbloqueando nuevas habilidades y equipamiento que mejoran el combate y exploración. Este sistema proporciona una sensación de crecimiento constante.
- **Diseño del desafío:** el sistema de desafío es una combinación entre precisión absoluta y dificultad progresiva, la mayoría de enemigos tienen un sistema de ataques diferentes por lo cual el jugador amerita aprenderse los patrones de cada uno mediante la repetición y la observación. La distribución de enemigos y obstáculos ayudan a mantener el equilibrio dentro del juego al igual que la progresión constante. Este enfoque fomenta el aprendizaje continuo del jugador y genera una sensación de satisfacción dentro del juego.

En la Ilustración 4 se puede observar la presentación visual de Death's Door y su estilo artístico.

Ilustración 4

Presentación visual de Death´s Door

Nota: obtenido de Godisageek [13]

2.1.4. Cuadro Comparativo de trabajos relacionados

En esta sección se toma en cuenta las investigaciones previas y se realiza una comparación de sus puntos fuertes para identificar sus principales diferencias, similitudes y áreas de interés común que se puedan usar para gestionar el presente proyecto investigativo. Por lo tanto, en la tabla 1 se realiza la comparación de los videojuegos: Dark Souls III, Hollow Knight y Death's Door.

Tabla 1

Resumen comparativo de los trabajos relacionados

Aspectos	Dark Souls III	Hollow Knight	Death's Door

Sistema de combate	Combate estratégico basado en ataques, esquivas, bloqueos y lectura de patrones enemigos.	Combate rápido y preciso con ataques cuerpo a cuerpo, esquivas y habilidades especiales.	Combate dinámico en tiempo real con ataques cuerpo a cuerpo, a distancia, esquivas y habilidades mágicas.
IA de enemigos	Enemigos con estados de patrulla, detección, persecución, ataque y retorno, además de patrones de combate diferenciados.	Enemigos con comportamientos variados y patrones específicos según su tipo y entorno.	Enemigos con comportamientos diferenciados de desplazamiento, persecución, ataque y reposicionamiento durante el combate.
NPCs e interacción	NPCs con diálogos, comercio, mejoras y misiones que evolucionan según el progreso del jugador.	NPCs que proporcionan información, comercio y desarrollo narrativo mediante diálogos e interacciones.	NPCs orientados al desarrollo narrativo, orientación del jugador e interacción con el mundo del juego.
Exploración y diseño de niveles	Mundo interconectado con rutas alternativas, atajos y zonas ocultas que recompensan la exploración.	Exploración no lineal basada en regiones conectadas, secretos y habilidades que desbloquean nuevas áreas	Escenarios interconectados con caminos alternativos, secretos y distribución equilibrada entre combate y exploración.

Sistema de salud y recursos	Barra de salud y frascos de estus con uso limitado, fomentando la administración estratégica de recursos	Barra de vida y recurso Alma utilizado tanto para curación como para habilidades especiales.	Barra de salud y recursos para habilidades especiales que requieren una administración estratégica durante el combate.
Progresión del jugador	Mejora de atributos mediante almas, adquisición de equipamiento y fortalecimiento gradual del personaje	Desbloqueo de habilidades, amuletos y mejoras que amplían las capacidades de exploración y combate.	Mejora de atributos mediante almas, desbloqueo de habilidades y fortalecimiento progresivo del personaje.

Nota: Elaboración propia

La comparación presentada en la Tabla 1 destaca las distintas posibilidades y diseños que se pueden utilizar como base en el proyecto, esto no se basa en escoger y replicar las opciones, sino tomarlas como base de desarrollo.

2.2. Bases teóricas

En esta sección se recopila y analiza investigaciones previamente realizadas relacionadas con la IA en videojuegos, comportamiento de personajes no jugables y sistema de control de personaje. El análisis de estos estudios permite conocer enfoques, técnicas, métodos y resultados, obteniendo una base sobre la aplicación y evolución de estas tecnologías.

2.2.1. Inteligencia artificial en videojuegos

La IA en videojuegos funciona mediante técnicas computacionales las cuales se utilizan para generar comportamientos, decisiones o respuestas dentro de los entornos virtuales. El área de IA se compone de varias características como aprendizaje del comportamiento en personajes no jugables (Non-Playable Character) NPCs, búsqueda y planificación, generación procedimental de contenido, narrativa computacional y diseño asistido mediante IA. Por esta razón, no se limita únicamente al control de enemigos, sino comprende distintos procesos en el campo de los videojuegos [14].

Los juegos se han utilizado como un entorno de prueba para la IA poniendo a prueba la capacidad de resolver problemas y seleccionar acciones [15]. Estas investigaciones iniciaron en juegos de mesa sencillos hasta escalar a juegos dinámicos con decisiones en tiempo real [15], [16]. Entre varias técnicas se encuentran la búsqueda heurística, aprendizaje supervisado, aprendizaje por refuerzo, máquinas de estado finitas entre otros. La elección de estas técnicas depende de la cantidad de agentes, información disponible y la complejidad de la experiencia [15].

Las máquinas de estado finitas (FSM) es una técnica utilizada para organizar el comportamiento de los agentes virtuales mediante un sistema de estados y transiciones [17], [18]. Este consiste en mantener al agente siempre en un estado activo ejecutando acciones asociadas con este hasta que se active una condición o evento que lo obligue a cambiar de estado. Los estados pueden representar comportamientos como patrullar, atacar, perseguir o retirarse. La transición de estos estados se pueden producir mediante información del entorno como detección del

jugador , distancia respecto al jugador, perdida de visibilidad o nivel de vida del agente [18].

Las FSM brindan una estructura comprensible y ejecución predecible. Sin embargo, el aumento excesivo de estados y conexiones pueden llevar a que el proyecto tenga un mal mantenimiento y afecta directamente la escalabilidad [17], [18]. Como respuesta a esta limitación, las FSM pueden agrupar varios estados y reutilizar transiciones comunes evitando el exceso de código [18].

2.2.1.1. Árboles de decisiones

Estos son una estructura jerárquica sobre un proceso de elección mediante varios tipos de nodos como por ejemplo el nodo raíz, nodos internos que evalúan condiciones determinadas, ramas que responden a los posibles resultados y nodos finales que establecen una acción. Estos modelos se basan en ejemplos, dividiendo la información para diferenciar entre cada resultado posible [19], [20]. Por ejemplo, el algoritmo ID3 selecciona atributos basados en la ganancia de información, logrando reducir la incertidumbre en cada ramificación del árbol [19].

Los árboles de decisiones dentro de los videojuegos se utilizan para determinar las distintas acciones de los personajes no jugables mediante la evaluación de las condiciones de su entorno. El sistema empieza en un nodo raíz y recorre las ramas según los acontecimientos de su entorno hasta llegar a una acción final en base al estado actual del juego [21]. En una investigación basada en videojuegos de carrera, se utilizó árboles de decisiones basados en los puntos de recorrido para que los NPCs decidieran cuando acelerar, frenar o cambiar de dirección [22].

Una de sus características principales es su representación gráfica esta permite observar que condiciones fueron evaluadas y el recorrido que siguió antes de llegar a la acción final [20]. Esto facilita la revisión de las reglas que utilizo el sistema y que clase de condiciones llegan a provocar una determinada acción [20], [21]. Sin embargo, incorporar demasiadas divisiones puede aumentar la complejidad de su estructura y dificultar la interpretación de la misma. En árboles generados mediante aprendizaje automático, su crecimiento excesivo provoca sobreajuste e inestabilidad [20].

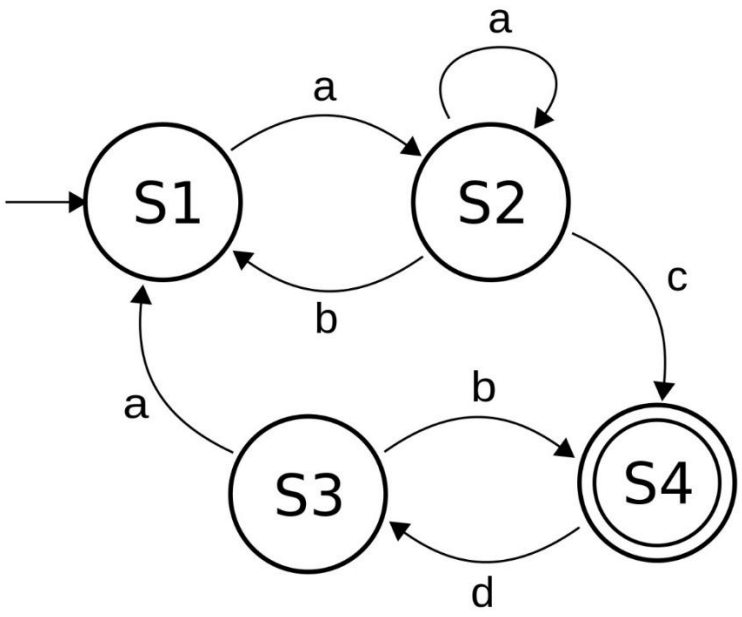
2.2.1.2. Máquinas de estados finitas

Las FSM representan cómo se comporta el sistema mediante un conjunto limitado de estados que se conectan a través de transiciones. Dentro de este sistema solamente permanece activo un estado a la vez, cada transición lleva a un cambio de estado cuando se cumple una condición o se produce algún evento. Esta organización de reglas definidas permite representar gráficamente la relación entre los estados y las condicione para que estos cambien [17].

Dentro de los videojuegos, cada estado agrupa un comportamiento, mientras que las condiciones de transición determinan cuando pasar de un estado a otro. Fathoni et al. [23] implementaron FSM en las cuales los enemigos tienen distintos estados en los cuales persiguen al jugador, lo atacan, y regresan al estado inicial. Esto demostró que la separación de comportamientos permite comprobar individualmente las condiciones y respuestas implementados en los personajes. En la Ilustración 5 se puede observar el flujo de funcionamiento de una máquina de estados finitas (FSM).

Ilustración 5

Flujo de máquinas de estado finitas



Nota: Obtenido de la profesional review [24]

Como muestra la ilustración 5 una máquina de estados finita organiza su comportamiento con un conjunto de estados, los cuales únicamente permanece 1 activo a la vez. Los eventos y condiciones que recibe le permiten cambiar de un estado a otro. Algunos estados permiten regresar a la acción anterior mientras que otros obligan a pasar por otros estados antes de volver al estado inicial. Esta estructura evita la ejecución simultanea de acciones.

2.2.2. Personajes en videojuegos

Los personajes de un videojuego son figuras que tienen un rol dentro del mundo virtual, se representan mediante apariencia, movimientos, acciones, diálogos y decisiones. Dependiendo de la forman en la que se los controla se distinguen entre personajes jugables, los cuales son dirigidos por el usuario y no jugables, cuyas

acciones son administradas por el sistema. Aunque representan 2 formas distintas de interacción ambos son fundamentales para la relación del jugador con el entorno virtual [21], [25].

2.2.2.1. Personajes jugables

Dentro de los videojuegos los personajes se dividen en: el personaje jugable es aquel que se encuentra bajo el control del usuario y su función es ser el medio por el cual este actúa dentro del mundo virtual [26] Esta condición transforma al personaje jugable en un vínculo entre las acciones que realiza y los acontecimientos que representa el sistema. Por consiguiente, el personaje jugable cumple una función interactiva al recibir órdenes del usuario pero de igual forma tiene una función narrativa dependiendo de la apariencia previamente definida por los desarrolladores [26], [27].

La relación que guarda el personaje jugable no se limita únicamente en lo mecánico, sino que también entra en una relación de identificación e interpretación con el papel que representa dentro de la historia [26], [28] Hefner, Klimmt y Vorderder [28] defienden que la identificación con el personaje principal funciona como una modificación temporal de la autopercepción del usuario durante la experiencia de juego. Los autores sostienen que esta interactividad facilita que el jugador realice acciones directamente relacionadas con la personalidad del personaje y experimente con sus capacidades dentro del juego.

La relación que guarda el usuario con el personaje jugable influye al nivel de satisfacción que el jugador obtiene de la experiencia. En un estudio realizado sobre

el videojuego *the last of us part II,* se identificó que modificar el vínculo emoción del personaje y la imagen que se tenía anteriormente del mismo influyen en la valorización del videojuego, Esto demuestra que la percepción que el usuario tiene sobre el personaje puede transformarse en base a los acontecimientos narrativos y a las acciones del usuario [26].

2.2.2.2. Personajes No jugables

El otro tipo de personaje de los videojuegos son: los personajes no jugables (NPCs) en contraste, son controladas por el sistema y no por usuarios reales [27]. En experiencias de tipo narrativas, estos personajes pueden funcionar como gestores de diálogos y están encargados de transmitir información, asumir un papel dentro de la historia o responder intervenciones del usuario [27], [29]. Esta narrativa interactiva permite que el usuario se relacione con los NPCs y que estas interacciones formen parte del desarrollo de la trama [27]. Bajo este contexto el diálogo es el principal medio por el cual el NPC puede interactuar con el jugador [29], [30].

El desarrollo de los diálogos de NPC deben considerar varios aspectos simultáneos como el estado de la trama y propósito comunicativo [30] Rowe, Ha y Lester [30] proponen un sistema de diálogos que usen la personalidad e historia narrativa del personaje para generar naturalidad de respuestas e intervenciones a objetivos específicos. Asimismo, los rasgos de personalidad se pueden comunicar mediante jergas, vocabulario local o la extensión de respuestas [29].

A parte de transmitir información, las interacciones con los NPCs pueden incorporar distintas opciones de respuesta por parte del jugador lo que permite modificar el desarrollo de la interacción, Yin y Xiao [31] señalan que estas

modificaciones pueden producir variantes en los acontecimientos, los personajes y sus diálogos. Por esta razón las respuestas de los NPCs tienen que estar ligadas a la opción que escoja el jugador para conservar la coherencia con la situación narrativa, de manera que el usuario pueda reconocer el efecto de sus decisiones.

2.2.3. Sistemas de Combate

El sistema de combate se basa en un conjunto de mecánicas centrales que regulan las acciones del jugador junto con las respuestas que genera el videojuego [32]. Estas acciones son ejecutadas mediante las condiciones necesarias, restricciones y las consecuencias que produce al estado de juego [32]. Aplicado a combate estas acciones están relacionadas a atacar, esquivar, defenderse, recibir daño o administrar recursos[32]. Por lo tanto, estas dependen del entendimiento y dominio que tenga el jugador sobre ellas[32], [33].

El combate debe permitir al jugador aprender progresivamente y desarrollar las habilidades necesarias para enfrentar los desafíos presentados [33]. Los controles deben responder de una manera precisa y fluida a las acciones del usuario para que los resultados se sientan realistas [33], [34]. Asimismo, las mecánicas deben ser variadas para que el jugador pueda descubrir nuevas estrategias conforme aumente su dominio de las mecánicas[33].

La dificultad del juego está fuertemente vinculada al sistema de combate ya que este determina el nivel de exigencia al que se enfrenta el jugador [33], Un desafío inferior a las habilidades del usuario puede reducir su interés y en contraste un nivel de desafío muy elevado puede causar frustración [33], [35]. Schmierbach et al. [35] comprobaron que una dificultad abusiva puede alterar el equilibrio percibido entre las

habilidades del jugador y el desafío presentado Además, Klimmt et al. [36] demostraron que la dificultad, el rendimiento y las expectativas del jugador trabajan conjuntamente para generar una sensación de satisfacción.

La satisfacción que produce el sistema de combate también está relacionado por la percepción de competencia y que el jugador reconozca el mérito sobre los resultados que obtiene [36], [37]. Ryan, Rigby y Przybylski [37] encontraron el disfrute se asocia con la competencia y autonomía que el desafío le plantee al jugados. Cuando el jugador comprende las mecánicas del juego, puede practicarlas y mejorar sus habilidades esto fortalece la competitividad del juego [37], [36]. Gracias a esto el usuario genera una sensación de dominio cada que logra superar un enfrentamiento lo que le da una sensación de satisfacción si la victoria se percibe como el resultado de un aprendizaje y correcta ejecución de sus habilidades[36], [37].

El equilibrio del combate también es un factor importante este se logra mediante un ajuste correcto de parámetros como el daño generado, el daño recibido, tiempos de recuperación, velocidad de ejecución de acciones y la disponibilidad de recursos, esto mantiene una relación adecuada entre las capacidades del jugador y las exigencias del sistema [33], [34]. Una alternativa también es una dificultad adaptable a las habilidades del jugador, esto se logra analizando el rendimiento del usuario y modificando determinadas condiciones en la partida [34]. En conclusión un sistema de combate equilibrado debe tener controles comprensibles, desafíos alcanzables, oportunidades de aprendizaje, respuestas claras y resultados que dejen evidencia del avance que ha ido obteniendo el jugador [33], [34].

2.3. Bases Metodológicas

En este apartado se investigan dos metodologías empleadas para el desarrollo de proyectos: Scrum y Kanban. De cada metodología se obtendrá información basándose en su enfoque principal, adaptación a cambios y su forma de planificación, con el propósito de comparar sus características principales.

2.3.1. Scrum

Es una metodología ágil se orienta a la gestión de proyectos mediante procesos incrementales los mismos se dividen en periodos denominados Sprints. Estos Sprints permiten establecer objetivos a cumplir en una cantidad delimitada de tiempo, lo que ayuda a organizar actividades y entrega de avances. Esta metodología se basa en la colaboración del equipo y el seguimiento del progreso y revisión de los resultados. Además, Scrum Utiliza herramientas como Product backlog y el sprint backlog para priorizar funcionalidades.

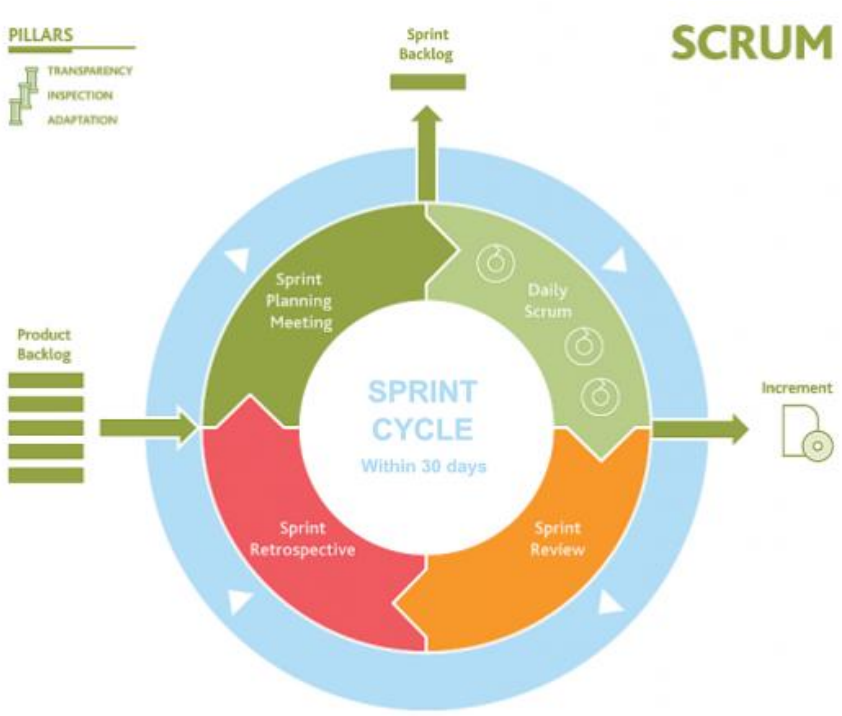
- **¿Qué es Scrum?:** Es un es un marco de trabajo ágil que se utiliza para la organización del desarrollo de productos. Este no define actividades de manera rígida sino que establece ciclos breves de trabajo para observar el avance y ajustar el proceso [38].
- **¿Para qué sirve Scrum?:** este se utiliza para coordinar equipos em trabajos complejos y cuyos requisitos puedan modificarse durante el desarrollo. La aplicación del mismo favorece a la colaboración, transparencia, responsabilidad compartida y la revisión del trabajo frecuente, estos aspectos permiten detectar problemas y mejorar la calidad del producto mediante inspección y adaptación.[39].

- **Flujo de trabajo de Scrum:** este es representado en un tablero compuesto por columnas que marcan el progreso de tareas en pendiente, en proceso y finalizado, los mismos se pueden adaptar a las distintas etapas de cada proceso. Esto permite mantener un flujo continuo de trabajo y detectar acumulaciones o cuellos de botella [40].

En la Ilustración 6 se puede observar el flujo de trabajo utilizado en la metodología Scrum.

Ilustración 6

Flujo de Scrum



Nota: Obtenido de JavierGarzas.com [41]

La ilustración 6 representa el ciclo de trabajo de Scrum, este inicia con la selección de tareas en Product Backlog para posteriormente formar Sprint Backlog. Durante el sprint, se realizan planificaciones, reuniones diarias y retrospectiva, con el objetivo de mejorar la planificación y productividad.

2.3.2. Kanban

Este se enfoca en la gestión visual del flujo de trabajo, utiliza tableros que representan el estado de las actividades. Su principal objetivo es mejorar la organización de las tareas, permitiendo identificar qué actividades se encuentran pendientes, en proceso o ya finalizadas. También, busca optimizar la productividad

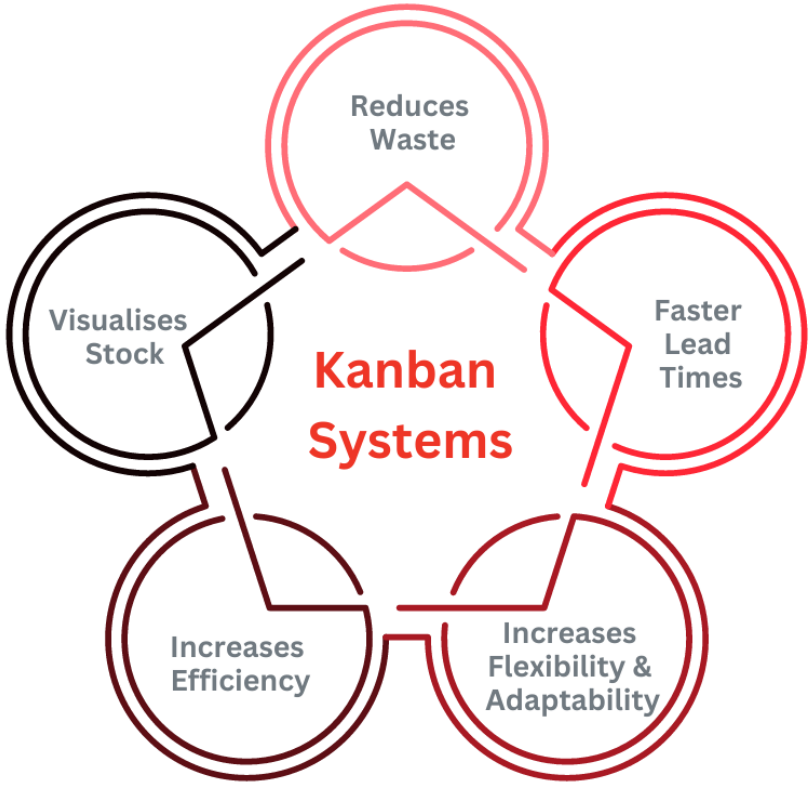
por medio de limitar el trabajo en proceso, evita acumulación de tareas y facilita la detección de posibles retrasos.

- **¿Qué es Kanban?:** Es un método de gestión visual que representa las tareas mediante tarjetas distribuidas en columnas correspondientes al estado del proceso. Tiene como principios visualizar el trabajo, limitar la cantidad de tareas y administrar el desplazamiento continuo [42].
- **¿Para qué sirve Kanban?:** Se utiliza para visualizar la carga de trabajo, controlar las tareas e identificar bloqueos en el proceso. Entre sus beneficios esta una mejora en la comunicación, reducción de trabajo en proceso y mayor fluidez de desarrollo [43].
- **Flujo de trabajo de Kanban:** Se representa en un tablero marcado por columnas como pendiente, en proceso y finalizado, aunque esto se adapta a la fase en la que se encuentra el proyecto. Las tarjetas avanzan de una columna a otra conforme se cambia de estado y únicamente se incorpora nuevo trabajo si existe la capacidad disponible [40].

En la Ilustración 7 se puede observar la organización de las actividades mediante la metodología Kanban.

Ilustración 7

Metodología Kanban



Nota: Obtenido de TwinBin [44]

La ilustración 7 representa el flujo de la metodología Kanban, esta permite visualizar el trabajo pendiente, reducir desperdicios y agilizar los tiempos de entrega. También favorece a la adaptación ante cambios, mediante la continua gestión del flujo de trabajo.

2.3.3. Cuadro comparativo de Metodologías

En esta sección se compara las características que tiene cada metodología para destacar sus principales ventajas y desventajas. La Tabla 2 presenta una comparación entre las características principales de Scrum y Kanban.

Tabla 2

Comparación de metodología

Aspecto	Scrum	Kanban
Enfoque principal	Organiza el trabajo de manera iterativa e incremental mediante periodos definidos llamados *sprints*. Cada ciclo tiene objetivos específicos y finaliza con la entrega y revisión de un incremento funcional.	Se enfoca en mantener un flujo continuo de trabajo mediante un tablero visual. Las tareas avanzan entre diferentes estados y se limita el trabajo en progreso para evitar acumulaciones.
Planificación	La planificación se realiza al comienzo de cada sprint. El equipo selecciona las tareas prioritarias del *Product Backlog* y establece el trabajo que deberá completar durante el ciclo.	La planificación es continua. Se incorporan nuevas tareas cuando existe capacidad disponible, sin necesidad de esperar a que finalice un periodo de trabajo previamente establecido.
Adaptación al cambio	Los cambios se analizan durante las revisiones y retrospectivas. Generalmente, los nuevos requerimientos se priorizan para incorporarlos en el siguiente sprint sin alterar el objetivo del ciclo actual.	Permite introducir y reorganizar tareas de manera flexible conforme cambian las prioridades. Esta adaptación se realiza sin interrumpir el flujo de las actividades que ya se encuentran en ejecución.

Nota: Elaboración propia

Después de comparar las dos metodologías investigadas se seleccionó Scrum ya que este permite organizar el desarrollo en periodos cortos de trabajo y distribuir funciones de manera progresiva. Dividir el proyecto de esta manera facilita la implementación de cambios. Además, las revisiones después de cada ciclo permiten detectar errores y reorganizar tareas pendientes.

2.4. Bases tecnológicas

En esta sección se analiza los motores gráficos para desarrollo de videojuegos, en busca de la mejor opción para desarrollar el presente proyecto. Se busca apartados que faciliten el uso de IA en enemigos y el desarrollo del control para el jugador principal, se investiga los puntos fuertes de cada uno como el lenguaje de programación que utiliza, la barra de herramientas, jerarquía de acciones, comodidad de uso, vista de la escena, entre otros.

2.4.1. Unity

Este es un motor de desarrollo multiplataforma que permite crear proyectos bidimensionales y tridimensionales para diferentes dispositivos [45], [46]. Este software gano reconocimiento en el mundo del desarrollo gracias a su disponibilidad de herramientas integradas y a su adaptabilidad con múltiples plataformas [45], [46]. La lógica de programación emplea el lenguaje C# para desarrollar interacciones, mecánicas, entornos e IA [47]. Estos proyectos son organizados mediante escenas , GameObjects, prefabs y componentes, mientras que su programación orientada a objetos nos permite reutilizar código y designar distintas responsabilidades entre varios scripts [48]. Además, este mismo incorpora distintas herramientas para la implementación de físicas, colisiones, cámara, interfaces y escenas[48], así mismo, facilita el uso de agentes virtuales mediante el uso de mallas de navegación [49]. Estas características convierten a Unity en una opción flexible y adaptativa, ahora enlistaremos sus características principales para posteriormente compararlas con otro software.

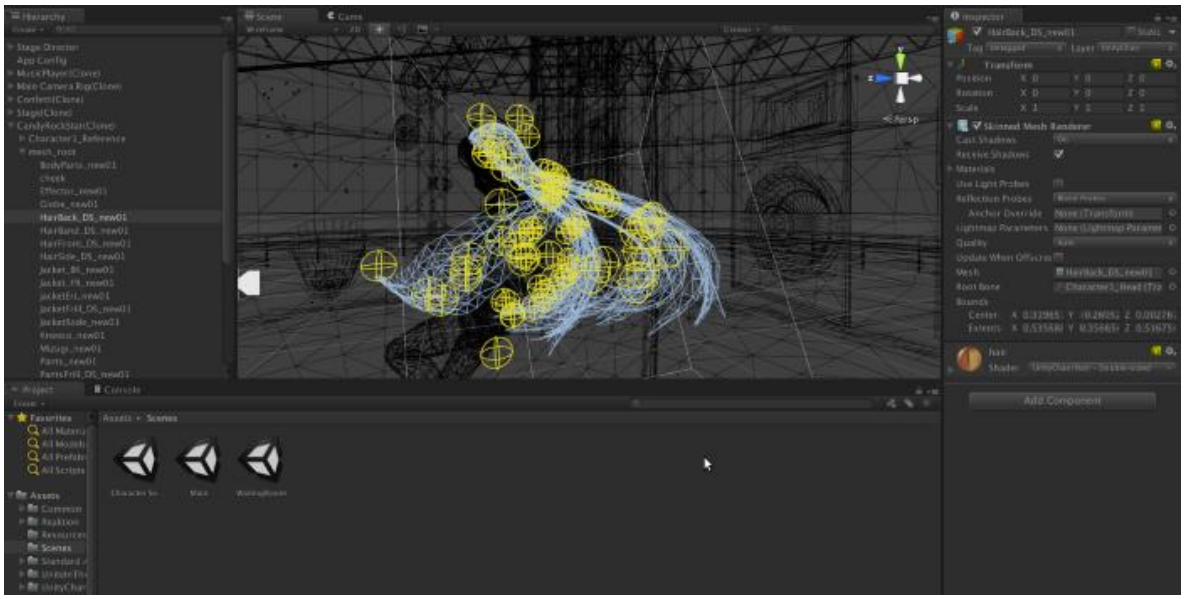
- **Lenguaje de programación:** Unity trabaja con el lenguaje C# como principal herramienta de desarrollo de código. Este es un lenguaje orientado a objetos que permite la implementación de mecánicas, gestionar eventos y controlar interacciones mediante scripts reutilizables. C# facilita la organización del código y el mantenimiento del proyecto a la hora de implementar funciones complejas.
- **Ventana del proyecto:** Este es el espacio donde Unity organiza los recursos que se usan durante el desarrollo. Esta sección permite almacenar distintos archivos como modelos 3D, texturas, materiales, scripts, animaciones, sonidos, escenas, permitiéndonos organizarlos a nuestra conveniencia y de forma personalizada.
- **Vista de la escena:** La vista de escena nos brinda un entorno donde se puede construir y editar visualmente el escenario de los juegos. Dentro de esta ventana se puede reposicionar objetos, ajustar la iluminación, organizar elementos, entre otras cosas. Además, permite ver la distribución de los componentes que vamos colocando en la escena.
- **Ventana de jerarquía:** La ventana de jerarquía acumula todos los objetos que están dentro de una escena mediante una estructura personalizable y organizada en forma de ramas. Este diseño facilita identificar la relación entre cada objeto principal y secundario.
- **Componentes para el desarrollo de agentes virtuales:** Unity tiene incorporado herramientas para el uso de agentes virtuales. Entre ellas destacan NavMesh, este permite implementar un sistema de navegación automática

sobre el escenario las cuales se pueden modificar para gestionar donde si pueden o no navegar los NPCs; el animator, se encarga de gestionar las animaciones y los parámetros para cambiar de animación; el Input System, utilizado para gestionar las acciones del jugador a través de distintos dispositivos de entrada; y Cinemachine, herramienta que facilita la modificación del sistema de cámaras en el juego. La integración de estas herramientas facilita el uso de máquinas de estado entre otros métodos para comportamiento de IA.

En la Ilustración 8 se puede observar la interfaz principal del motor de desarrollo Unity.

Ilustración 8

Visualización de interfaz de Unity



Nota: obtenido de Unity[50]

2.4.2. Godot

Godot es un motor de desarrollo de videojuegos gratuito, este destaco por su código abierto, orientado a la creación de proyectos en dos y tres dimensiones [51], [52]. Su arquitectura es organizada mediante nodos que se agrupan para formar las distintas escenas las cuales pueden ser reutilizadas e incorporadas dentro de otras escenas [51], [52]. Asimismo, el motor tiene incorporado herramientas para la integración de interfaces, físicas, animaciones y navegación [51], [52]. La lógica de programación utiliza un lenguaje integrado específicamente para Godot denominado GDScript, aparte admite también lenguaje como C# entre otras opciones [51], [52].

- **Lenguaje de programación:** Este motor utiliza principalmente GDScipt, un lenguaje diseñado específicamente para este motor con una sintaxis que se asemeja a Phyton. Adicionalmente, es compatible con C# Y C++ mediante

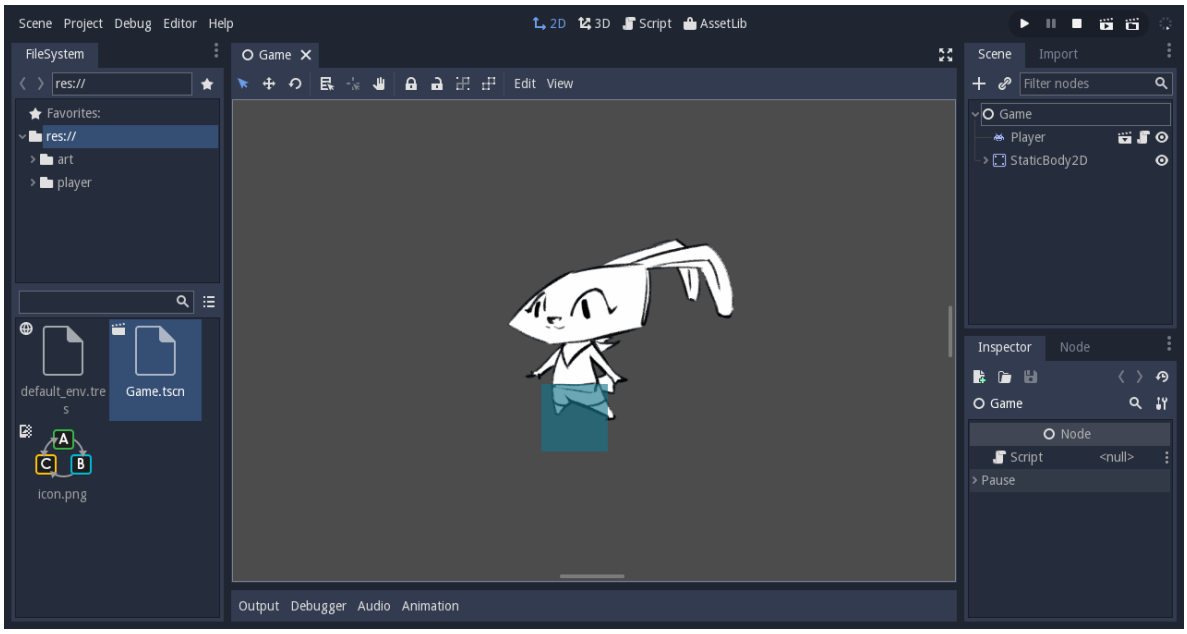
GDExtension. Esta flexibilidad permite optar por el lenguaje necesario para el tipo de proyecto que se realice.

- **Ventana del proyecto:** esta organiza todos los recursos que se usaran durante el desarrollo, incluyendo escenas, scripts, modelos, texturas y más. Los recursos se pueden clasificar en carpetas para una mejor organización y reutilización.
- **Vista de la escena:** la vista de escena permite editar el nivel del videojuego mediante la incorporación y manipulación de los nodos. Aquí es posible posicionar objetos, modificar la cámara, iluminación y otros componentes. Esta herramienta optimiza el proceso de diseño.
- **Ventana de jerarquía:** esta presenta la estructura de cada escena mediante un sistema de nodos en forma de árbol. Los nodos representan un elemento especifico de la escena como los personajes, cámaras, interfaces o elementos extras del escenario. Esta organización facilita la administración de distintos componentes.
- **Componentes para el desarrollo de agentes virtuales:** Godot cuenta con distintas herramientas para el desarrollo de agentes virtuales. Entre ellas destacan NavigationServer, que permite implementar navegación por mallas; AnimationTree, utilizado para la gestión de transiciones entre animaciones; AnimationPlayer, este controla secuencia de animaciones y facilita la interacción entre personajes y entornos.

En la Ilustración 9 se puede observar la interfaz principal del motor de desarrollo Godot.

Ilustración 9

Visualización de interfaz de Godot



Nota: obtenido de godot *[53]*

2.5. Arquitectura, modelos y estándares

` La arquitectura nos permite establecer una organización, defendiendo sus componentes, responsabilidades y la manera en la que estos se relacionan. Esta depende de las características del sistema, su complejidad y la necesidad de organizar y mantener el código [54]. Para determinar una estructura adecuada se analizaron tres alternativas: modelo vista-controlador, Arquitectura en Capas y Arquitectura en Capas y Arquitectura Orientada a Eventos.

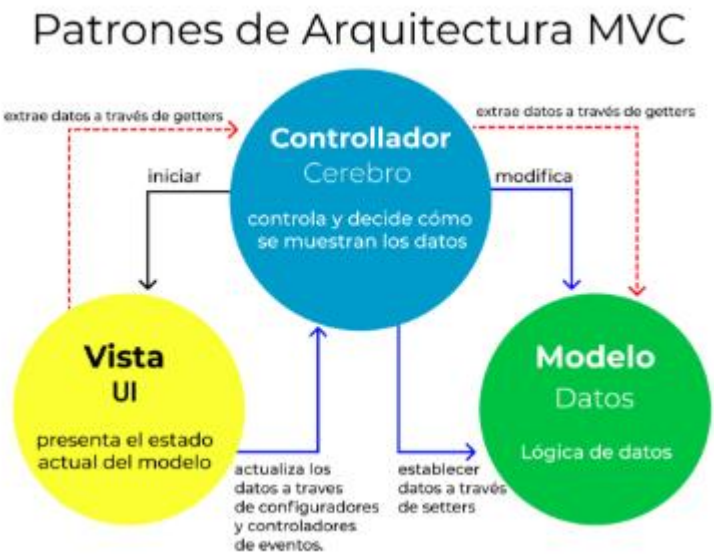
2.5.1. Modelo vista-controlador (MVC)

El modelo vista-controlador se divide en tres principios. El modelo administra los datos y la información utilizada por la aplicación; la vista representa la información al usuario; y el Controlador procesa las acciones e interacciones que

modifican el estado del sistema [55]. Esta separación permite distribuir las responsabilidades y evitar concentrar toda la lógica de un mismo componente. En la Ilustración 10 se puede observar la estructura de la arquitectura Modelo-Vista-Controlador (MVC).

Ilustración 10

Arquitectura MVC



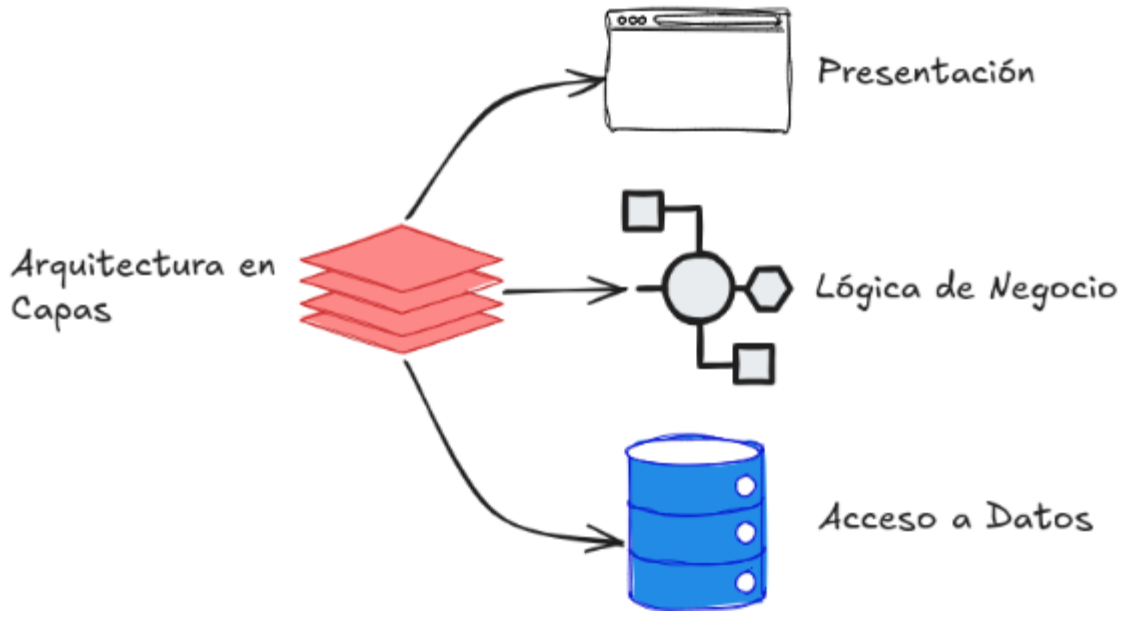
Nota: Obtenido de [56]

2.5.2. Arquitectura en capas

La arquitectura en capas usa organización por diferentes niveles, cada uno mantiene responsabilidades específicas y se relacionan con los demás utilizando interfaces o dependencias definidas [54]. Esta organización permite separar elementos relacionados con la presentación, lógica, datos y otros subsistemas. En la Ilustración 11 se puede observar la distribución de los componentes mediante una arquitectura en capas.

Ilustración 11

Arquitectura en capas



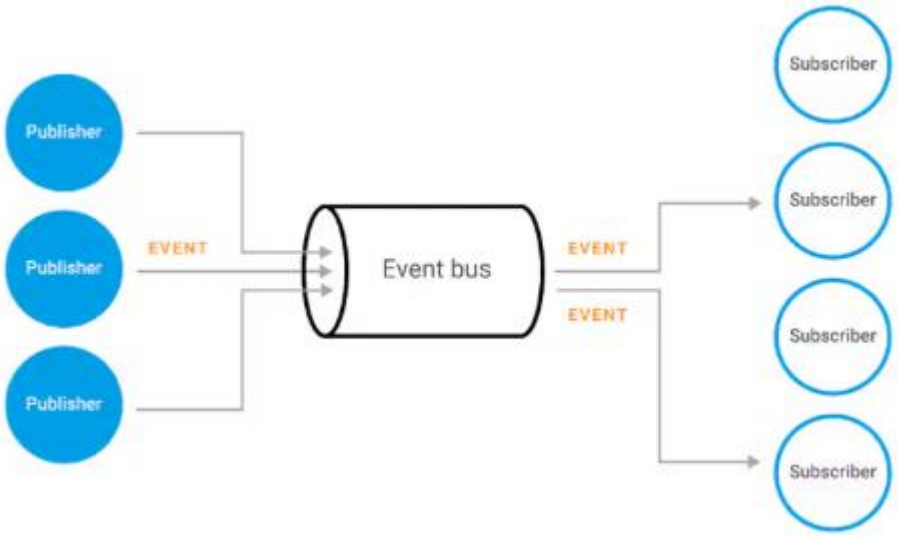
Nota: obtenido de [57]

2.5.3. Arquitectura Orientada a eventos

La arquitectura orientada a eventos establece la comunicación entre los componentes a partir de eventos cuando ocurre una acción o algo cambia dentro del sistema [54]. Esto permite reducir la dependencia directa entre diferentes componentes. En la Ilustración 12 se puede observar la estructura de una arquitectura orientada a eventos.

Ilustración 12

Arquitectura orientada a eventos



Nota: Obtenido de [58]

A continuación, la tabla 3, se presenta comparando las distintas arquitecturas presentadas.

Tabla 3

Comparación de arquitecturas

Aspecto	MVC	Arquitectura en Capas	Orientada a Eventos
Organización	Modelo, Vista y Controlador	División mediante diferentes capas	Componentes comunicados mediante eventos
Separación de responsabilidades	Alta	Alta	Alta
Comunicación	El Controlador relaciona Modelo y Vista	Comunicación entre capas	Emisión y recepción de eventos
Facilidad de seguimiento	Alta	Alta	Media

Adaptación al proyecto	Alta	Media-Alta	Media
Principal aplicación	Separación de datos, presentación y lógica	Organización de grandes subsistemas	Comunicación desacoplada entre sistemas

Nota: Elaboración propia

2.6. Herramientas de desarrollo

En esta sección se detallan las principales características de las bases tecnológicas descritas en la sección 2.4. Esta comparación considera lenguaje de programación, la ventana del proyecto, la vista de escena y herramientas para el desarrollo de agentes virtuales. La Tabla 4 presenta una comparación de las principales características de Unity y Godot.

Tabla 4

Comparativa de herramientas tecnológicas

Aspecto	Unity	Godot
Lenguaje de programación	Utiliza principalmente **C#**, un lenguaje orientado a objetos ampliamente utilizado en el desarrollo de software, que facilita la creación de scripts modulares, escalables y reutilizables.	Emplea principalmente GDScript, un lenguaje propio con sintaxis similar a Python, además de ofrecer compatibilidad con C#, C++ y otros lenguajes mediante extensiones.
Ventana del proyecto	Organiza todos los recursos del videojuego, como escenas, modelos, scripts, materiales, animaciones y archivos multimedia, permitiendo una administración estructurada del proyecto.	Administra los recursos del proyecto mediante un explorador de archivos que organiza escenas, scripts, texturas, modelos y demás elementos necesarios para el desarrollo del videojuego.
Vista de la escena	Permite diseñar y editar visualmente escenarios	Proporciona un entorno visual para construir las

	tridimensionales y bidimensionales, facilitando la ubicación y configuración de los objetos antes de ejecutar el proyecto.	escenas mediante un sistema basado en nodos, permitiendo editar la distribución y configuración de los elementos del juego.
Ventana de jerarquía	Presenta los objetos de la escena en una estructura jerárquica, facilitando la organización de personajes, cámaras, interfaces y demás componentes del videojuego.	Organiza los elementos de la escena mediante una jerarquía de nodos que representa las relaciones entre los distintos componentes del proyecto.
Componentes para el desarrollo de agentes virtuales	Incorpora herramientas como NavMesh, Animator, Input System y Cinemachine, que facilitan la implementación de navegación, animaciones, control del jugador y sistemas de IA como las Máquinas de Estado Finitas (FSM).	Incluye herramientas como NavigationServer, AnimationTree, AnimationPlayer y un sistema de físicas integrado, permitiendo desarrollar navegación, animaciones y comportamientos de IA mediante una arquitectura modular.

Nota: Elaboración propia

El análisis comparativo muestra que tanto Unity como Godot ofrecen herramientas adecuadas para el desarrollo del videojuego. Sin embargo, para el desarrollo del presente proyecto se optó por utilizar Unity debido a que ofrece mayor soporte para proyectos tridimensionales, facilita la integración de IA y cuanta con amplia documentación y una comunidad activa de desarrolladores.

2.7. Marco conceptual

El marco conceptual reúne los principios empleados durante el desarrollo del prototipo con el objetico de establecer una clara comprensión de los elementos

técnicos relacionados con la programación del proyecto. La Tabla 5 presenta los principales conceptos relacionados con el desarrollo del proyecto.

Tabla 5

Marco conceptual

Concepto	Definición aplicada al proyecto
IA	Sistema encargado de administrar los comportamientos y respuestas de los NPCs dentro del videojuego.
Personaje jugable	Personaje controlado directamente por el usuario mediante las mecánicas de movimiento, combate e interacción.
NPC	Personaje administrado por el sistema que puede presentar comportamientos hostiles o interacciones mediante diálogos.
Máquina de estados finita (FSM)	Estructura que organiza el comportamiento de los enemigos mediante estados y condiciones de transición.
Scriptable Objects	Recurso de Unity utilizado para almacenar y configurar datos de los estados y otros sistemas sin incorporarlos directamente en la lógica.
NavMeshAgent	Componente utilizado para controlar el desplazamiento y navegación autónoma de los enemigos.
Hitbox	Área de colisión utilizada durante los ataques para detectar impactos y aplicar daño.
Animation Event	Evento incorporado en una animación para ejecutar una acción en un momento determinado, como activar un hitbox o generar partículas.
Sistema de combate	Conjunto de mecánicas que administra ataques, daño, vida, muerte y habilidades de los personajes.

Nota: Elaboración propia

2.8. Relación entre la teoría y la propuesta tecnológica

La relación entre la propuesta tecnológica está establecida mediante los fundamentos estudiados acerca de IA, personajes, sistema de combate, tecnologías y arquitecturas de software aplicadas al desarrollo de videojuegos. Estos conocimientos sustentan la elaboración de un sistema de IA basado en máquinas de estado finitas. La

programación de personajes jugables administra las acciones como movimiento, esquive, combate, vida y muerte mientras que los NPCs y la IA generan interacciones y comportamientos que responden a las acciones del jugador.

Capítulo III

3. Manual de usuario

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

3.1. Especificaciones de uso

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

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

3.2. Configuración de Controles

La ejecución de acciones se realiza utilizando un teclado alfanumérico para el desplazamiento espacial del personaje jugable y la orientación del ángulo visual de la cámara, junto a los botones del ratón que sirven para las acciones de combate. La Tabla 6 presenta el esquema de controles utilizado para ejecutar las acciones del personaje.

Tabla 6

Esquema de controles e interfaz de entrada

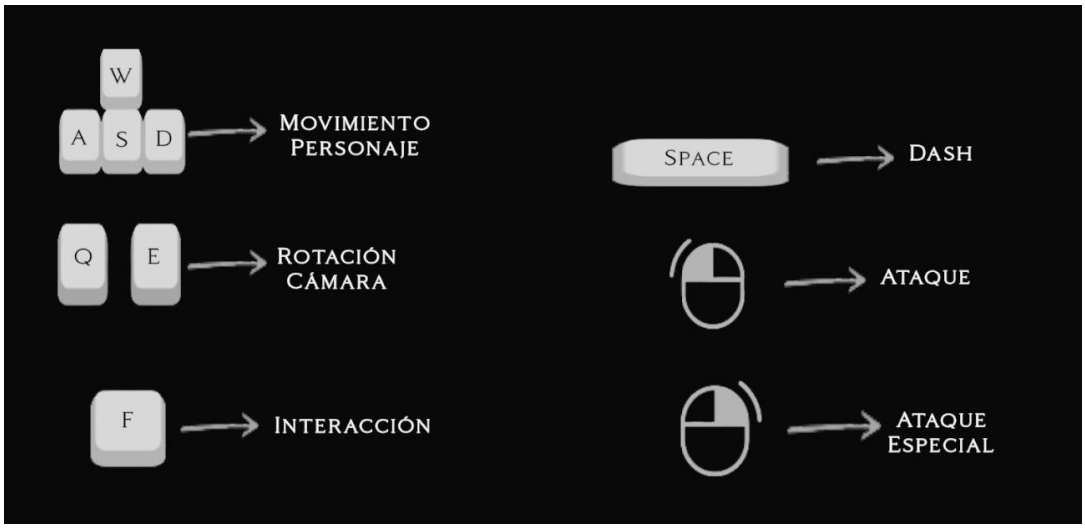
Acción Mecánica	Periférico / Tecla	Descripción Funcional dentro del Sistema

Movimiento de Personaje	W, A, S, D	Controla el movimiento del personaje jugable en el plano tridimensional (X, Z).
Rotación de Cámara	Q, E	Ajusta la perspectiva visual y la orientación angular de la cámara.
Interacción	F	Activa el diálogo con personajes no jugables (NPCs) e interacción con objetos del escenario.
Esquive / Impulso (Dash)	SPACE	Ejecuta un desplazamiento rápido para evadir ataques enemigos.
Ataque Básico	Clic Izquierdo	Activa la secuencia de ataque principal con el arma equipada.
Ataque Especial	Clic Derecho	Despliega la técnica especial sujeta al tiempo de recarga (Cooldown).

Nota: Elaboración propia

Ilustración 13

Esquema y distribución de los botones



Nota: Elaboración propia

3.3. Interfaz de Usuario (UI) y HUD

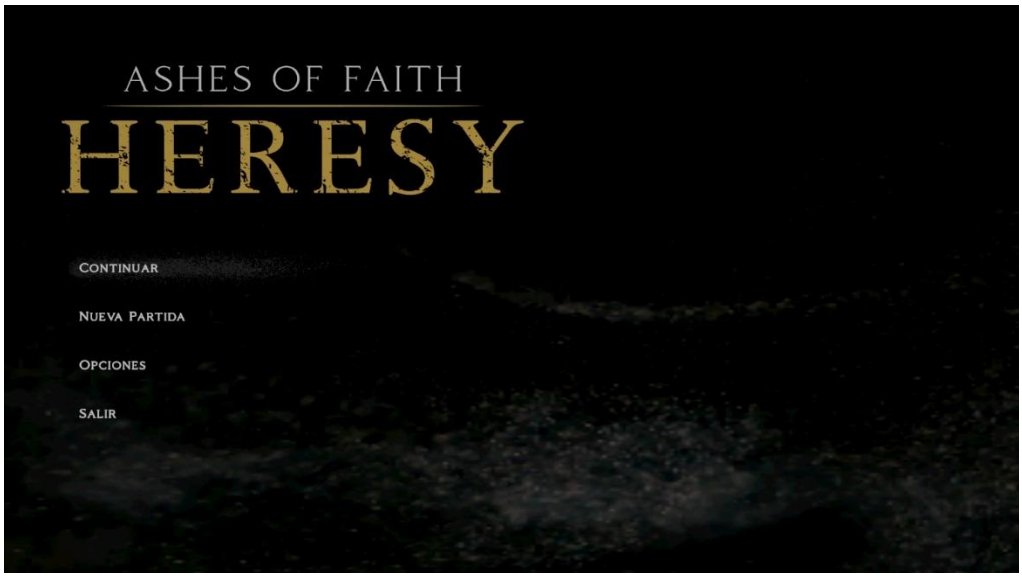
La interfaz gráfica presenta la información de las pantallas del juego desde la pantalla de inicio, los elementos permanentes en pantalla durante el gameplay (Heads-Up Display - HUD) y las pantallas de pausa para la gestión de información (menús).

3.3.1. Menú de inicio del juego

- **Continuar:** Permite reanudar la sesión de juego en el último punto de guardado donde se quedó el jugador la última vez que juego.
- **Nueva Partida:** Empieza la historia del juego desde cero, comienza una nueva aventura.
- **Opciones:** Muestra un panel de ajustes técnicos configurables como sonido, video y controles.
- **Salir:** Cierra el videojuego de forma segura y vuelve al escritorio de Windows.

Ilustración 14

Interfaz gráfica del menú de inicio



Nota: Elaboración propia

3.3.2. Indicadores del HUD

- **Barra de Vida (Salud):** Se encuentra ubicada en la parte inferior izquierda de la pantalla. Muestra la salud del jugador dentro de un marco alargado oscuro y metálico.

- **Indicador de Poder (Cooldown):** Ubicado por encima de la barra de vida. Indica si el ataque especial está disponible para usarse y el tiempo de espera para volver a usarlo.

Ilustración 15

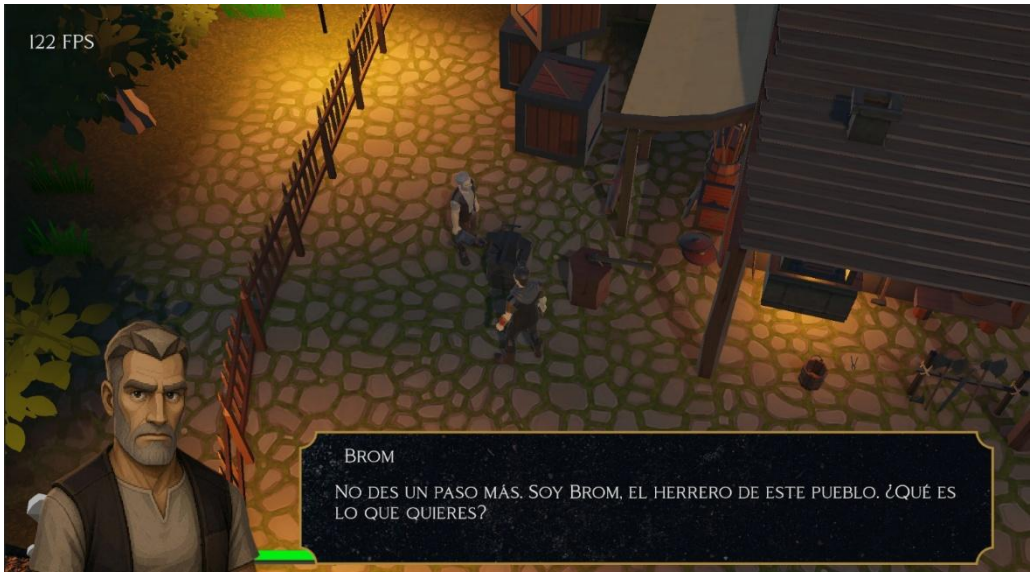
Vista de la perspectiva de cámara e indicadores del HUD (salud y cooldown)

Nota: Elaboración propia

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

Ilustración 16

Ejecución del sistema de diálogos con el NPC Brom



Nota: Elaboración propia

3.3.3. Sistema de Menú de Pausa

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

Ilustración 17

Interfaz de pausa

Nota: Elaboración propia

3.4. Procedimiento de Instalación

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

- Seleccionar la casilla Crear acceso directo en el Escritorio para crear un icono de acceso rápido en el escritorio y facilitar el ingreso posterior al videojuego.
- Hacer clic en el botón Instalar para iniciar el proceso de instalación de datos, copia de librerías DLL y registro de recursos del motor.

- **Finalización de la Instalación y Puesta en Marcha:**
  - Una vez completado el proceso de instalación, la interfaz del asistente mostrará la pantalla de finalización.
  - Marcar la casilla Ejecutar Ashes of Faith: Heresy y hacer clic en Finalizar.
  - Entonces el aplicativo estará listo para su ejecución.

Capítulo IV

4. Manual del programador

En el presente capítulo se documenta la construcción técnica implementada para el videojuego “Ashes of Faith”. Este capítulo describe la arquitectura general, las tecnologías empleadas, organización de recursos y la estructura de las clases y componentes desarrollados. También se aborda la programación de mecánicas para el personaje principal, sistema de combate, la IA de los NPCs hostiles y la interacción mediante diálogos de los NPCs no hostiles.

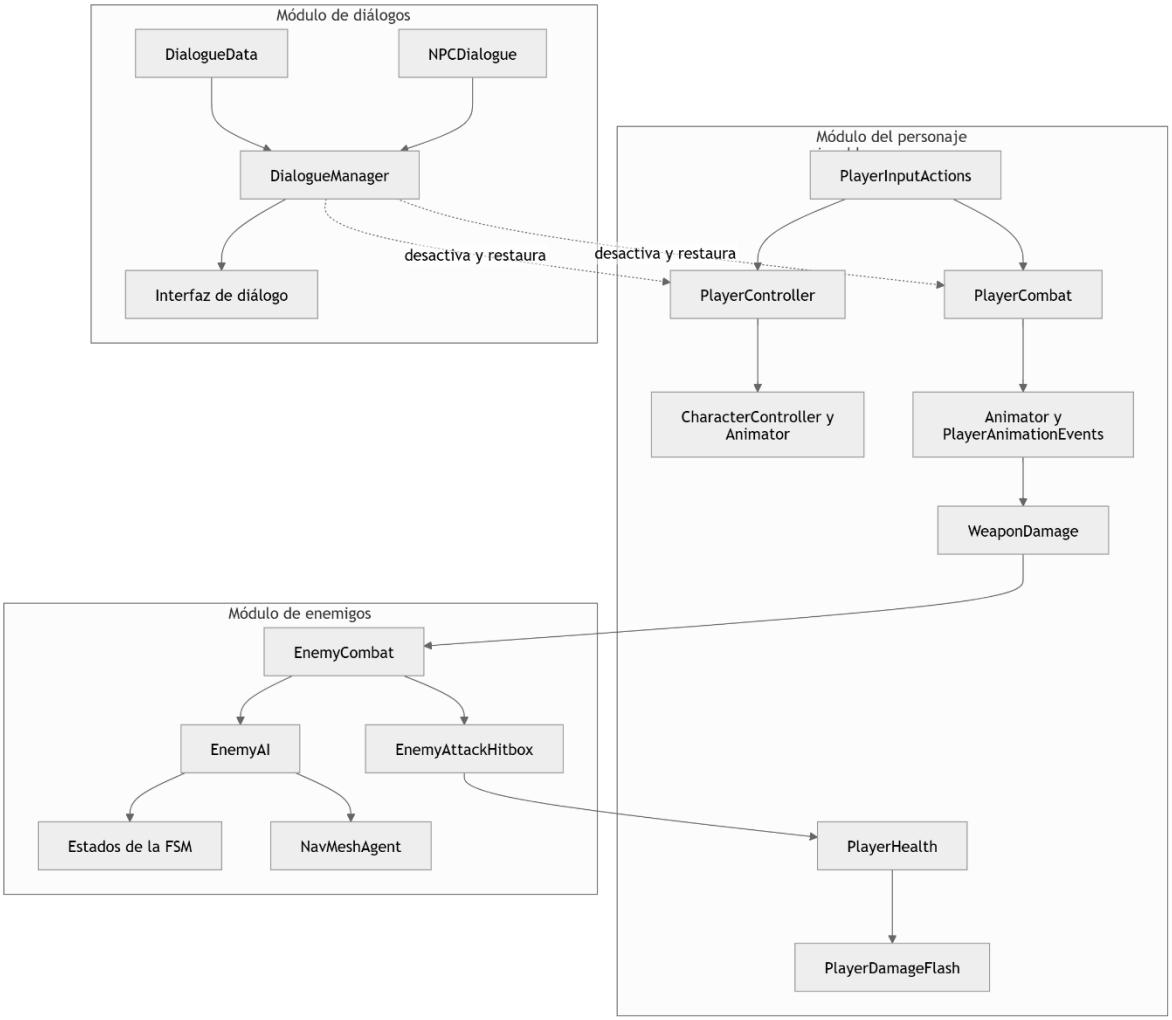
Asimismo, detalla la integración de estos sistemas junto con los recursos visuales como animaciones, modelos e interfaces. Finalmente se presentan pruebas técnicas, medidas de optimización y procesos generados para mantener la escalabilidad y mantenimiento del código. Esta información se complementa mediante diagramas de flujo, capturas y fragmentos del código generado.

4.1. Arquitectura

La implementación del software está organizada de manera modular. Específicamente la programación está compuesta de tres módulos: control del personaje jugable, IA de los enemigos y diálogos de personajes no jugables. Cada módulo contiene clases con distintas responsabilidades que se comunican mediante los componentes de Unity como CharacterController, Animator, NavMeshAgent, colliders e interfaces de usuario. La relación que guarda cada módulo se representa en la siguiente ilustración. En la Ilustración 18 se puede observar la arquitectura general del sistema desarrollado para el proyecto.

Ilustración 18

Arquitectura del sistema



Nota: Elaboración propia

Dentro del módulo de personaje jugable sus acciones se establecen mediante PlayerInputActions. La clase llamada PlayerControler se encarga de gestionar movimiento, rotación, sprint, la gravedad y el esquive haciendo uso de los componentes CharacterController y Animator. Por otro lado, la clase PlayerCombat se encarga de activar las animaciones de ataque. Mientras que PlayerAnimationEvents habilita y deshabilita el área de impacto controlada por

WeaponDamage en el momento especifico de la animación de ataque para que esta misma pueda causar daño.

WeaponDamage comprueba que el área de impacto alcanzó la capa especifica asignada a los enemigos y busca el componente EnemyCombat dentro de los mismos. Cuando el ataque valida estas dos condiciones, se aplica el daño y queda registrado dentro del enemigo. Además, este componente permite comprobar y aumentar el daño que genera el arma. El módulo también cuenta con la Clase PlayerHealth, la cual se encarga de controlar la vida, la invulnerabilidad temporal, la reacción de impacto y la muerte.

El siguiente módulo es el de IA que se centra en EnemyAI, esta clase se encarga de controlar las transiciones de la FSM y manejar la navegación mediante NavMeshAgent. En las máquinas de estados finitas se implementó comportamientos de reposo, patrulla, persecución, espera de combate, ataque básico, retorno y muerte. Cada estado hereda de la clase abstracta AIState, la cual implementa los métodos de inicialización, entrada, ejecución y salida. Las propiedades de estos estados están almacenadas en Scriptable Objects, estos permiten modificar sus valores como velocidad, el daño, el tiempo entre ataques y el límite de persecución sin necesidad de cambiar el código.

Enemy combat hereda directamente de la clase Combat e implementa los métodos definidos en ICombat para la administración de vida, efecto de daño, ataques y la muerte del enemigo. Cuando la cantidad de vida es <= a 0 este componente empieza un evento que EnemyAI utiliza para pasar al estado de muerte AIStateDie. Por otro lado, EnemyAttackHitbox activa el área de impacto en determinados

fotogramas de la animación de ataque y se comunica con PlayerHealth para la recepción de daño del jugador.

Por último, tenemos el módulo de diálogo que se compone de tres scripts: DialogueData, DialogueManager y NPCDialogue. DialogueData se encarga de almacenar la información de cada diálogo por ejemplo nombres de participantes, retratos y líneas de conversación. NPCDialogue se encarga de detectar si el jugador entro en el área de interacción e informa al administrador de diálogos. DialogueManager muestra el aviso de integración, controla la velocidad en la que se muestra el texto, actualiza el nombre y retrato dependiendo de quién está hablando en ese momento, pausa el juego, y desactiva temporalmente el control del personaje. Al finalizar la interacción, el sistema se restaura y habilita nuevamente los controles del personaje.

4.1.1. Tecnologías, motores y herramientas utilizadas

En este apartado están presentadas las plataformas, herramientas y paquetes que se emplearon para el desarrollo del proyecto. Estos recursos permitieron un mejor manejo de la lógica del juego, configuración de los componentes dentro del motor e integrar controles del personaje, IA y diálogos.

- **Unity 6.3 LTS (VERSION 6000.3.14F1):** La elección de Unity se establece a partir de la investigación previamente realizada donde comparamos motores para desarrollo de videojuegos, se consideraron aspectos tales como capacidad para desarrollar proyectos tridimensionales, su arquitectura basada en componentes y las herramientas que dispone para la integración de distintos mecanismos. La versión escogida corresponde a una versión LTS,

para mantener un entorno de trabajo estable durante la integración final del proyecto. La Tabla 7 presenta los requisitos técnicos para el uso de esta versión de Unity.

Tabla 7

Requisitos técnicos para el uso de Unity

Componente	Requisito técnico
Sistema operativo	Windows 10 versión 21H1 o posterior, en arquitectura de 64 bits. También es compatible con Windows 11.
Procesador	Arquitectura x64 con soporte para el conjunto de instrucciones SSE2 o arquitectura Arm64.
Memoria RAM	Un mínimo recomendado de 8 GB para ejecutar el editor. La cantidad necesaria puede aumentar según la complejidad del proyecto.
Tarjeta gráfica	Unidad gráfica compatible con DirectX 10, DirectX 11, DirectX 12 o Vulkan y controladores admitidos por el fabricante.
Almacenamiento	Se recomienda una unidad con una velocidad alta de lectura y escritura. El espacio requerido depende de los módulos instalados y del tamaño del proyecto.

Nota: Elaboración propia

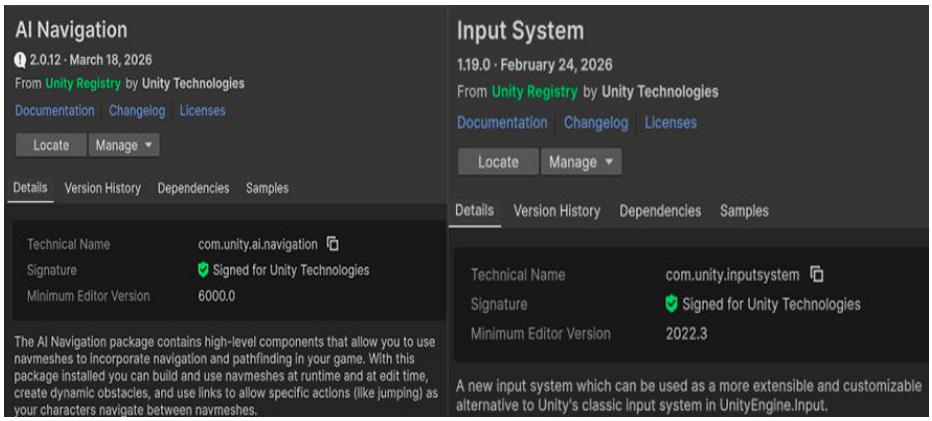
- **Visual Studio:** Se utilizo para el desarrollo de Scripts mediante el lenguaje C#. Esta herramienta brindó una mejor organización de clases, permitió acceso a las bibliotecas de Unity y programas todos los sistemas implementados. Todos los scripts generados fueron posteriormente asociados a objetos dentro de las escenas del juego.
- **C#:** Este fue el lenguaje empleado para programar las mecánicas del personaje jugable, sistema de combate, las FSM, navegación de agentes virtuales y diálogos entre personajes. Este lenguaje nos permitió acceder mediante código a los distintos componentes del proyecto como animator, CharacterController, Collider, NavMeshAgent y elementos de la interfaz.
- **Input System 1.19.0:** Utilizamos este paquete de Unity para recibir y configurar las acciones realizadas por el jugador mediante un mapa de inputs.

Esto nos ermitio gestionar las entradas correspondientes para el movimiento, sprint, esquive y ataque del personaje, manteniendo separada la configuración de los controles con la lógica de programación.

- **AI Navigation 2.0.12:** Se empleo para definir las superficies navegables dentro del escenario y controlar el desplazamiento de los enemigos. Su componente NavMeshAgent se integró dentro de la clase EnemyAI para la ejecución de estados como patrulla, persecución y retorno.
- **Unity UI y TextMeshPro:** Esta herramienta nos permitió presentar información dentro de la interfaz del videojuego, su implementación logro hacer visual las barras de vida. Avisos de interacción, retratos de personajes y líneas de texto. En la Ilustración 19 se pueden observar los principales componentes utilizados dentro del motor Unity.

Ilustración 19

Componentes de Unity



Nota: Elaboración propia

4.1.2. Organización técnica del proyecto

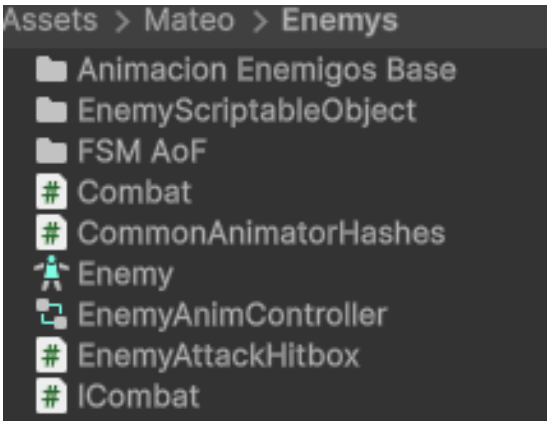
La organización se estableció bajo el uso de carpeta con el objetivo de separar recursos y scripts dependiendo del sistema al que pertenecen. Esta distribución

permite mantener diferenciada la programación del personaje principal, IA, los diálogos y animaciones. Esto también facilita localizar los componentes con mayor facilidad.

Toda la programación de los enemigos se almaceno dentro de la carpeta Enemys. Dentro de esta carpeta se encuentran las animaciones de los enemigos, el controlador de las animaciones, los scripts Combat, Icombat, y los componentes relacionados con el área de impacto. Dentro de la carpeta se creó “FSM AoF”, una carpeta que guarda toda la programación de las máquinas de estado finitas. En la Ilustración 20 se puede observar la organización de los archivos correspondientes a los enemigos del proyecto.

Ilustración 20

Carpeta de enemigos



Nota: Elaboración Propia

La carpeta “FSM AoF”, almacena EnemyAI y EnemyCombat, además de las clases correspondientes a los estados de reposo, patrulla, persecución, espera de combate, ataque, retorno y muerte. Dentro de esta estructura también está la carpeta “Templates”, la cual guarda las clases utilizadas para almacenar la configuración de

los comportamientos. Esta separación nos permite diferenciar la ejecución de cada estado y sus modificables. En la Ilustración 21 se puede observar la organización de los archivos correspondientes al sistema FSM.

Ilustración 21

Carpeta de FSM

Nota: Elaboración propia

Los componentes relacionados con el personaje jugable se almacenaron en "Player Scripts". Esta carpeta contiene las clases responsables del movimiento, combate, eventos de animación, vida, respuesta visual, daño y área de impacto del arma. Esta organización nos ayuda a separar los scripts del personaje controlable de los scripts de IA. En la Ilustración 22 se puede observar la organización de los archivos relacionados con el personaje jugable.

Ilustración 22

Carpeta de personaje jugable

Nota: Elaboración propia

Los diálogos están estructurados en DialogueData, DialogueManager y NPCDialogue dentro de la carpeta NPCsDiálogos, cada script separa el almacenamiento de la conversación, la administración de la interfaz y la detección del jugador para iniciar una interacción. Asimismo, las animaciones se mantuvieron separadas en carpetas independientes para evitar mezclar funcionalidades.

El proyecto dispone de 3 escenas, una escena de pruebas y 2 del recorrido general del juego. La escena de prueba nos permitió probar las mecánicas programadas dentro de un ambiente aislado. Las otras dos escenas nos dejaron comprobar el sistema ya integrado al videojuego final.

4.1.3. Arquitectura de clases y componentes

La arquitectura general del proyecto se basa en programación orientada a objetos junto con el sistema de componentes de Unity. Cada funcionalidad está programada con responsabilidades específicas que posteriormente se incorporan a los

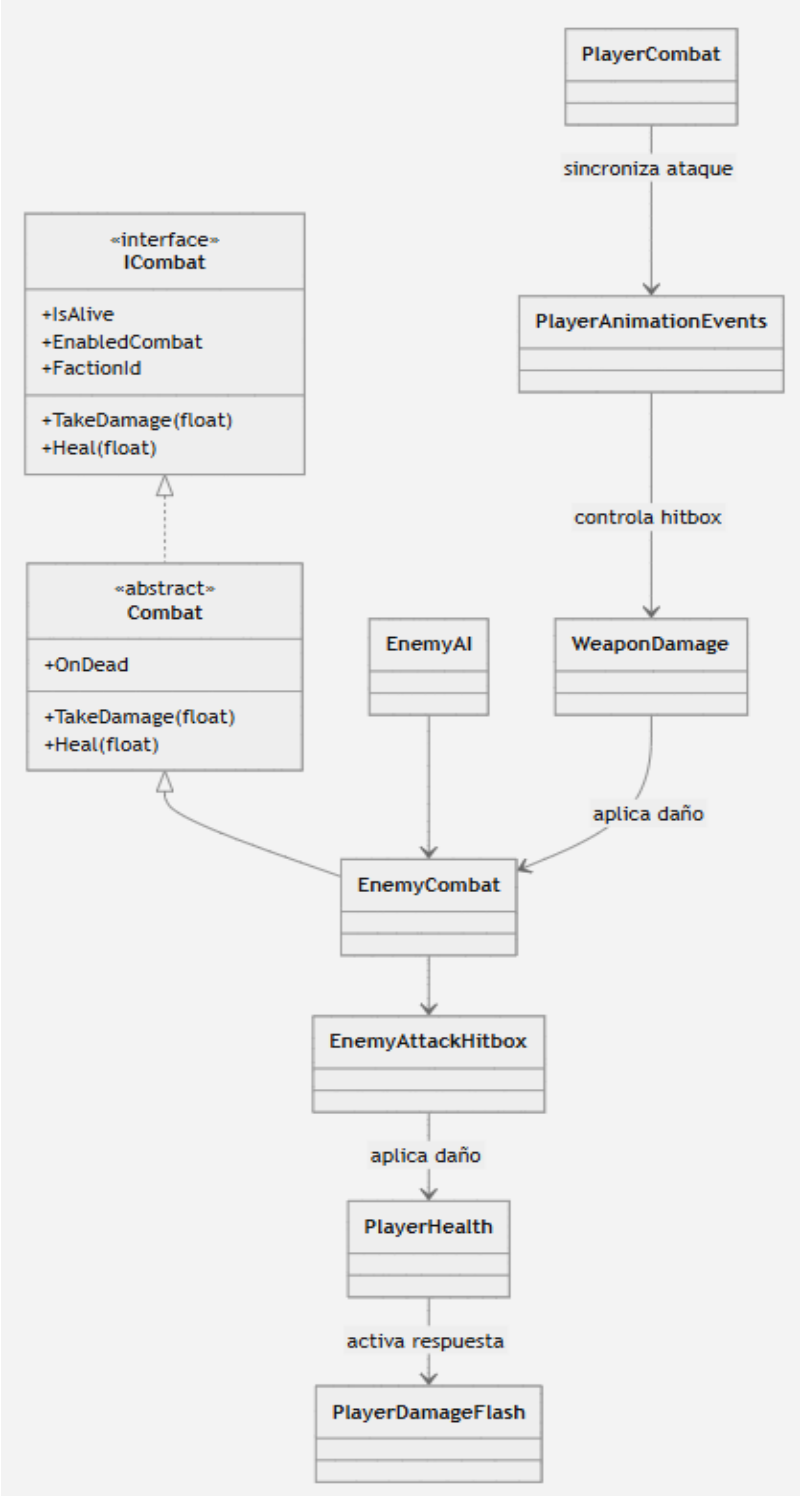
objetos de las escenas. Esto permite separar la lógica de programación para evitar que una sola clase se encargara de administrar todo.

Dentro del sistema de enemigos. El combate parte de la interfaz ICombat, esta define las operaciones y propiedades como la vida, la recepción de daño, la recuperación. La clase abstracta Combat implementa esta interfaz y la aplica a la lógica general que utilizan los enemigos, como puntos de vida, el estado de muerte y el evento OnDead. EnemyCombat hereda directamente de Combat y amplia las funciones mediante la ejecución de ataques, actualiza la barra de vida, activa y desactiva el área de impacto y la reproduce las animaciones de daño y muerte.

En el personaje jugable, PlayerController se encarga del movimiento mediante CharacterController y actualiza los parámetros del animator permitiendo transicionar entre animaciones. PlayerCombat recibe las acciones del usuario activando el ataque y sistema de combos, mientras que PlayerAnimationEvents, se encarga de sincronizar las animaciones con WeaponDamage. Este último componente comunica el daño con EnemyCombat siempre y cuando este haya detectado que la capa que colisiono corresponde a un enemigo. Los enemigos cuentan con el mismo sistema utilizando EnemyAttackHitbox el cual detecta colisiones con Player y aplica daño mediante la clase PlayerHealth. La respuesta visual del golpe se administra en PlayerDamageFlash. En la Ilustración 23 se puede observar el diagrama de las clases que conforman el sistema de combate.

Ilustración 23

Diagrama de las clases de combate



Nota: Elaboración propia

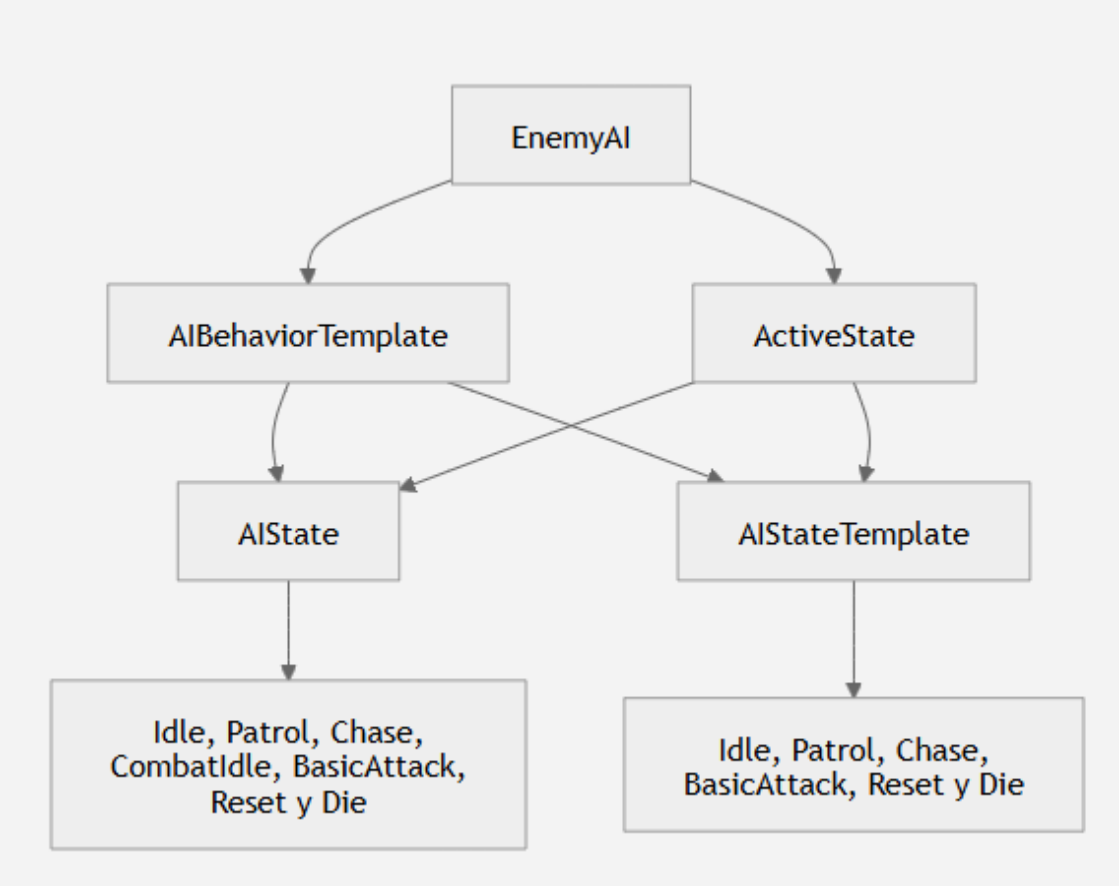
La IA se aplica mediante FSM. AIState es una clase abstracta que contiene los métodos Initialize, Enter, Execute y Exit. A partir de esta estructura se desarrollan los estados en este caso AIStateIdle, AIStatePatrol, AIStateChase, AIStateCombatIdle, AIStateBasicAttack, AIStateReset y AIStateDie. Todas estas clases funcionan con un comportamiento especifico y son delegadas a EnemyAI.

EnemyAI se encarga de controlar la máquina de estados finita. Al iniciar el enemigo registra que estados tiene disponibles, genera una instancia independiente para cada uno y los almacena en ActivateState. Posteriormente activa el estado actual y administra las transiciones bajo los parámetros correspondientes como detección de jugador, la distancia de ataque, el límite de persecución y la vida del enemigo. Esto nos evita compartir datos de ejecución entre diferentes tipos de enemigos.

Las configuraciones de la FSM están separadas mediante Scriptable Objects. AIStateTemplate funciona como una plantilla de cada estado mientras que AIBehaviorTemplate agrupa las configuraciones y estados que posteriormente colocaremos al enemigo. Las plantillas nos permiten configurar ciertas variables como la velocidad, la distancia de detección, el daño, el tiempo entre ataques y el límite de persecución. De esta manera podemos generar distintos tipos de enemigos sin necesidad de cambiar el código. En la Ilustración 24 se puede observar el flujo de funcionamiento de la FSM implementada en los enemigos.

Ilustración 24

Flujo de la FSM en enemigos



Nota: Elaboración propia

Por último, el sistema de diálogos también funciona con lógica separada en diferentes responsabilidades, DialogueData almacena la información obtenida mediante Scriptable Objectss y contiene listas de DialogueCharacter y DilogueLine para guardar nombres, retratos y textos. NPCDialogue se le asigna la referencia del diálogo y detecta la presencia del jugador mediante un collider. DialogueManager recibe esta información, administra la interfaz, presenta las líneas de texto y controla la pausa del juego.

Este tipo de organización modular nos facilita modificar las configuraciones de los enemigos, reutilizar comportamientos y mantener cada sistema sin alterar los demás componentes del videojuego.

4.2. Desarrollo

El desarrollo fue organizado mediante Scrum dentro de un periodo estimado de cuatro meses. El trabajo se dividió en cinco Sprints para la implementación del control del personaje, el combate, diálogos, IA y enemigos. Para establecer funcionalidades se elaboró un Product Backlog compuesto con historias de usuario relacionadas al aporte de programación. La tabla 8 representa la información general de la bitácora utilizada para registrar y organizar el desarrollo del proyecto.

Tabla 8

Información general de la bitácora

Campo	Información
Nombre	Mateo Ramírez
Título	Bitácora del proyecto
Descripción	Desarrollar e integrar los controles y el combate del personaje jugable, los diálogos de NPCs no hostiles y la IA de los enemigos mediante una máquina de estados finita.
Fecha de preparación	07/05/2026
Estimación	Cuatro meses

Nota: Elaboración propia

El Product Backlog organiza las funcionalidades con su respectivo grado de importancia dentro del prototipo. Las historias que se relacionan con los controles, el combate y la IA tienen prioridad, debido a que representan sistemas indispensables para el videojuego. Las interacciones con NPCs son de prioridad media puesto que funcionan como un complemento narrativo y no afectan directamente a la jugabilidad.

Como se muestra en la tabla 9, el Product Backlog organiza las funcionalidades con su respectivo grado de importancia dentro del prototipo.

Tabla 9

Product Backlog de los sistemas de programación

PRODUCT BACKLOG				
ID	**Descripción resumida**	**propiedad**	**Historia**	**Estado**
US-01	Como jugador necesito controlar al personaje principal para moverme libremente por el escenario.	Alta	Sistema de movimiento	Completo
US-02	Como jugador, necesito ejecutar un esquive para evitar los ataques realizados por los enemigos.	Alta	Sistema de esquive	Completo
US-03	Como jugador, necesito realizar una secuencia de ataques cuerpo a cuerpo para enfrentar a los enemigos.	Alta	Sistema de combate	Completo
US-04	Como jugador, necesito utilizar una habilidad especial durante el combate para disponer de una alternativa de ataque.	Media	Habilidad especial	Completo
US-05	Como jugador, necesito un sistema de vida, recepción de daño y muerte que represente el estado del personaje.	Alta	Vida y muerte del jugador	Completo
US-06	Como jugador, necesito visualizar mi vida y la de los	Alta	Interfaz de vida	Completo

	enemigos para conocer el estado de cada participante durante el combate.			
US-07	Como jugador, necesito interactuar con los NPCs no hostiles al acercarme a ellos para iniciar una conversación.	Media	Interacción con NPCs	Completo
US-08	Como jugador, necesito visualizar los nombres, retratos y líneas de los participantes para comprender las conversaciones.	Alta	Sistema de diálogos	Completo
US-09	Como jugador, necesito que los enemigos patrullen, detecten mi presencia, me persigan y regresen a su posición inicial cuando me aleje.	Alta	FSM de enemigos	Completo
US-10	Como jugador, necesito que los enemigos puedan atacar, causar daño, recibir impactos y morir para desarrollar los enfrentamientos.	Alta	Combate de enemigos	Completo

Nota: Elaboración propia

Se distribuyeron las historias de usuario en cinco sprints según el sistema correspondiente. Los controles de personaje se desarrollaron primero ya que son la base para probar las demás mecánicas integradas. En la tabla 10 podemos ver la distribución de los controles.

Tabla 10

Planificación de los sprints

Componente	Sprint	Historias
Controles del personaje	Sprint 1	US-01: Sistema de movimiento; US-02: Sistema de esquive.
Combate del personaje	Sprint 2	US-03: Sistema de combate; US-04: Habilidad especial; US-05: Vida y muerte; US-06: Interfaz de vida.
NPCs no hostiles	Sprint 3	US-07: Interacción con NPCs; US-08: Sistema de diálogos.
IA	Sprint 4	US-09: FSM de enemigos.
Combate de enemigos	Sprint 5	US-10: Ataques, daño y muerte de enemigos.

Nota: Elaboración propia

4.2.1. Sprint 1

Dentro del primer sprint se desarrolla las funciones básicas para el control del personaje jugable. Se Estableció esta etapa como un punto inicial dentro de la implementación, puesto que el control del jugador es necesario para posteriormente comprobar otras funciones como el combate o el diálogo. Este sprint se divide en dos historias dos historias de usuario. La primera está relacionada con el desplazamiento y orientación del personaje, mientras que la segunda se asocia con la ejecución de un esquive. La Tabla 11 presenta las historias de usuario y tareas correspondientes al Sprint 1.

Tabla 11

Tabla de sprint 1

Historia de usuario	**Tareas desarrolladas**	**Entregable**
US-01: Sistema de movimiento. Como jugador, necesito controlar el movimiento y la	Configurar las acciones Move y Sprint en Input System; programar PlayerController;	Personaje capaz de caminar, correr, orientarse y desplazarse por el escenario.

orientación del personaje para desplazarme por el escenario.	integrar CharacterController; administrar movimiento, rotación y gravedad; conectar las acciones con Animator.	
US-02: Sistema de esquive. Como jugador, necesito ejecutar un esquive para evitar los ataques realizados por los enemigos.	Configurar la acción Dash; validar la disponibilidad del personaje; activar la animación RollForward; bloquear temporalmente el movimiento convencional; integrar el desplazamiento de la animación mediante Root Motion.	Personaje capaz de ejecutar un esquive hacia delante mediante la tecla Espacio.

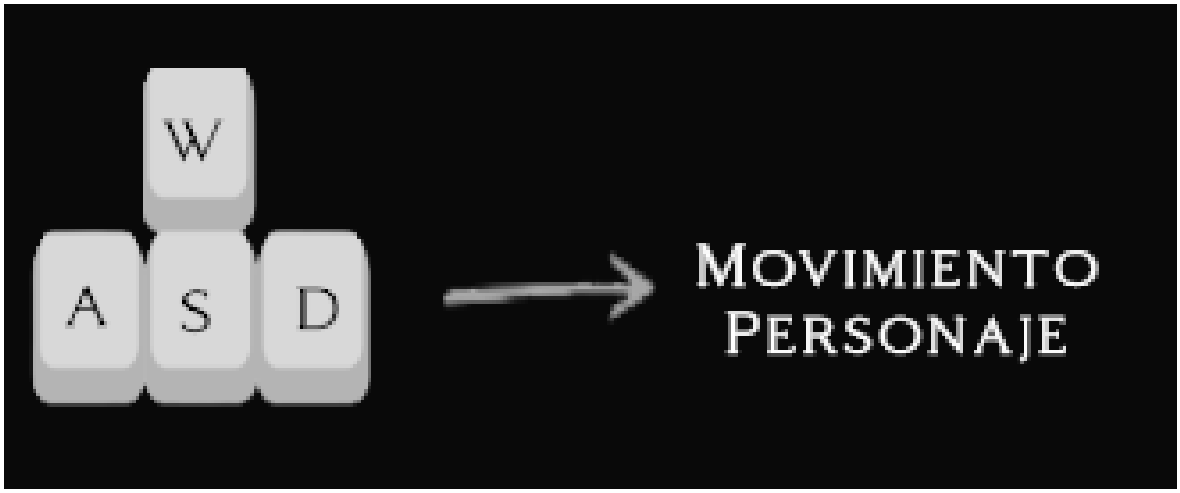
Nota: Elaboración propia

- **US-01: Sistema de movimiento y orientación**

El primer paso a realizar fue configurar las entradas para el control del personaje, para ello se utilizó InputAction para mapear las acciones y asignarles una tecla para su respectiva ejecución. Dentro del mapeo se configuró la acción “Move” con un valor bidimensional asociado a las teclas W, A, S, D, también se vinculó la tecla shift para la acción de Sprint. En la Ilustración 25 se puede observar la implementación del sistema de control de movimiento del personaje.

Ilustración 25

Control de movimiento



Nota: Elaboración propia

Para recibir estas entradas y convertirlas en desplazamiento del personaje se desarrolló el script PlayerController. La información obtenida por la acción “Move” permite calcular la dirección del movimiento, mientras que CharacterController ejecuta el desplazamiento y gestiona las colisiones. Se selecciónó este componente ya que evita que dependamos de las físicas que imparte Rigidbody.

La orientación del jugador se calcula igualmente por la acción “Move” cuando hay una entrada de movimiento el personaje rota progresivamente hacia la dirección que avanza. Para hacer que este efecto sea más natural se asignó un valor a la velocidad con la que rota, evitando que el cambio de dirección sea muy brusco.

El sistema de desplazamiento funciona bajo dos variables que definen la velocidad con la que se muevo el personaje. La primera es WalkSpeed esta es la velocidad con la cual el jugador avanza caminado y se estableció en 5 unidades. La segunda es SprintSpeed establecida en 7 unidades esta es la velocidad en la que el personaje va a correr cuando se mantiene presionado la entrada de sprint.

También se incorporó un sistema de gravedad para aplicar una caída cuando el personaje no se encuentra encima de una superficie para esto se utiliza la variable Gravity puesta en -20 unidades. Asimismo, se agrega la variable GroundedGravity em -2 unidades con la finalidad de estabilizar al personaje cuando esta encima de una superficie. La Tabla 12 presenta las funciones y valores principales configurados en PlayerController.

Tabla 12

Parámetros principales de PlayerController

Parámetro	Valor	Función
Walk Speed	5	Velocidad de desplazamiento normal
Sprint Speed	7	Velocidad aplicada al apretar shift
Rotation Speed	10	Suavizado de la orientación del personaje
Gravity	-20	Fuerza aplicada cuando el personaje no está en el duelo
Grounded Gravity	-2	Mantiene al personaje en contacto con el suelo

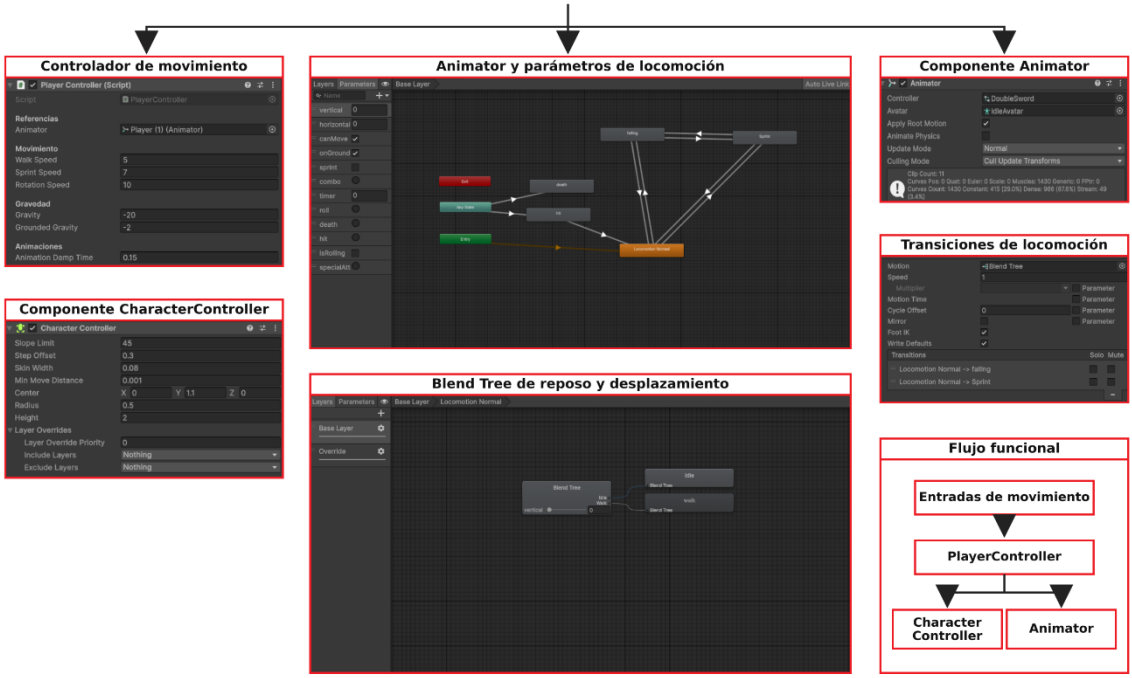
Nota: Elaboración propia

La visualización de movimiento se logra gracias a un animator conectado con el script PlayerController. Las variables de movimiento cambian parámetros dentro del controlador de animaciones lo que permite que el personaje alterne entre reposo, caminata y carrera. Para intercambiar entre reposo y caminata se utilizó un Blend Tree el cual calcula la velocidad de movimiento, si la velocidad es cero este se mantendrá en reposo si la velocidad es igual o más que 0.1 pasara al estado de caminata. Por otro lado, el sprint se logra bajo un parámetro de tipo bool el cual se

activa y desactiva mediante la tecla shift. En la Ilustración 26 se puede observar la implementación del control del personaje principal.

Ilustración 26

Control del personaje principal



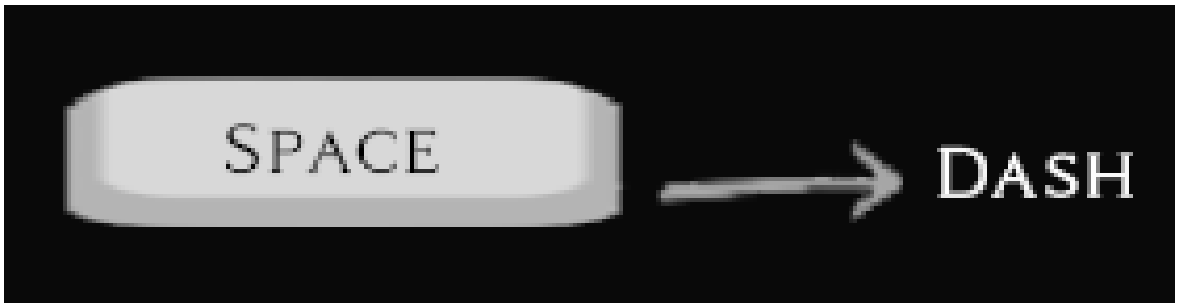
Nota: Elaboración propia

- **US-02: Sistema de esquive**

En la segunda historia se enfoca en la acción de esquive la cual permite al jugador cambiar rápidamente de posición. En el mapeo de controles esta acción se le asigno a la tecla espacio. Cuando PlayerController detecta esta entrada se asegura de verificar que el jugador este en la condición adecuada antes de ejecutar la acción.

Ilustración 27

Control de esquive



Nota: Elaboración propia

Esta verificación evita que el personaje pueda usar varios esquives a la vez o de activar el esquive mientras el jugador está realizando una acción incompatible. Si se cumplen estas condiciones el sistema activa el parámetro correspondiente en el animator para reproducir la animación.

A diferencia del movimiento convencional, el desplazamiento y velocidad del esquive funciona directamente con la animación mediante Root Motion el cual se encarga de desplazar al personaje, Durante la ejecución de la animación se restringe temporalmente el control de movimiento para evitar que se altere la dirección del esquive.

Se mantuvo separadas las responsabilidades del sistema mediante el uso de PlayerController y el animator. El script se encarga de iniciar el esquive, mientras que la animación le da visualización y desplazamiento.

- **Resultados del Sprint 1**

Dentro de la escena de prueba se pudo verificar el correcto funcionamiento de los sistemas desarrollados en este sprint. Se comprobó que el personaje respondiera a las entradas de movimiento, modificara su velocidad dependiendo de si camina o

corre, mantuviera contacto con el terreno y que el esquive se ejecute de forma correcta bloqueando movimiento temporalmente.

4.2.2. Sprint 2

El segundo sprint se orienta a las mecánicas de combate, para ello se integraron entradas de usuario, animator, detector de colisiones, la administración de vida y la interfaz. Este sprint está compuesto por cuatro historias de usuario que se relacionan con la habilidad especial, vida y muerte, y la interfaz de vida. La Tabla 13 presenta las historias de usuario y tareas desarrolladas durante el Sprint 2.

Tabla 13

Tabla de sprint 2

Historia de usuario	Tareas desarrolladas	Entregable
US-03: Sistema de combate. Como jugador, necesito realizar una secuencia de ataques cuerpo a cuerpo para enfrentar a los enemigos.	Configurar ComboAttack; programar PlayerCombat; organizar tres animaciones consecutivas; sincronizar el collider del arma; detectar impactos mediante WeaponDamage.	Combo de tres ataques capaz de causar daño a los enemigos.
US-04: Habilidad especial. Como jugador, necesito utilizar una habilidad especial durante el combate para disponer de una alternativa de ataque.	Configurar la entrada con el clic derecho; validar las condiciones de ejecución; implementar el cooldown; detectar enemigos en un área; integrar partículas y animación.	Ataque especial con daño en área y tiempo de reutilización.
US-05: Vida y muerte. Como jugador, necesito un	Programar PlayerHealth; administrar vida	Sistema capaz de recibir daño, representar impactos

sistema de vida, recepción de daño y muerte que represente el estado del personaje.	actual y máxima; incorporar respuesta al daño, aturdimiento e invulnerabilidad; activar la muerte del personaje.	y controlar la muerte.
US-06: Interfaz de vida. Como jugador, necesito visualizar mi vida y la de los enemigos para conocer su estado durante el combate.	Conectar la vida del jugador con un Slider; actualizar las barras de los enemigos; utilizar un gradiente para representar su estado.	Interfaz de vida funcional para el jugador y los enemigos.

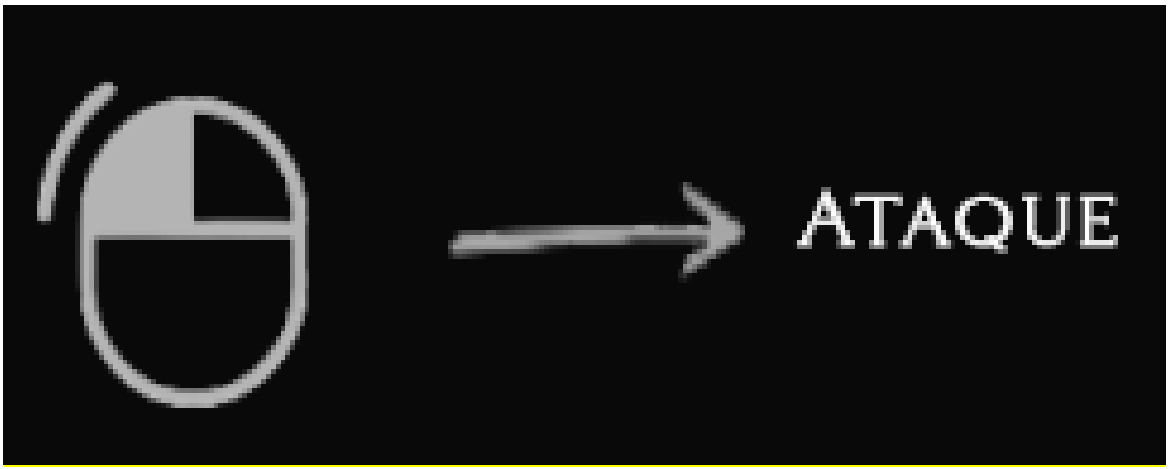
Nota: Elaboración propia

- **US-03: sistema de combate**

Dentro del mapa de acciones se implementaron los inputs necesarios para el combate. Esta acción la asociamos al clic izquierdo del ratón. El script PlayerCombat se encarga de implementar esta entrada y verifica que el personaje este en el suelo y no este ejecutando un esquive para poder dar inicio al ataque. En la Ilustración 28 se puede observar la implementación del sistema de ataque del personaje.

Ilustración 28

Control de ataque

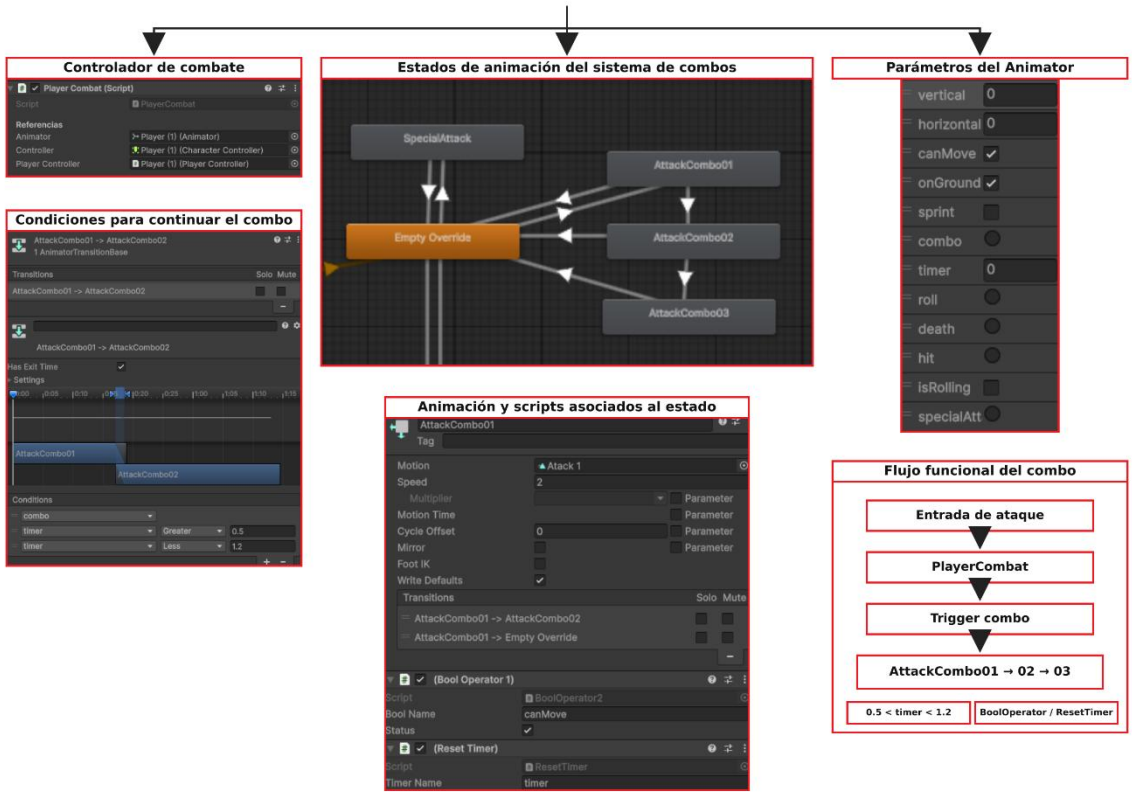


Nota: Elaboración propia

Cuando estas condiciones se cumplen se activa el parámetro combo dentro del animator. El controlador de animaciones cuanta con los estados AttackCombo01, AttackCombo02 y AttackCombo03 los cuales representan 3 golpes distintos que se encadenan el uno al otro. Las transiciones permiten continuar la secuencia del combo mediante el script ResetTime el cual coloca un contador si el usuario vuelve a presionar el botón dentro del intervalo de tiempo adecuado se concatena el siguiente ataque. Si no se registra ninguna entrada vuelve al estado idle y se reinicia el combo. En la Ilustración 29 se puede observar el funcionamiento del sistema de combos implementado para el personaje.

Ilustración 29

Sistema de combos



Nota: Elaboración propia

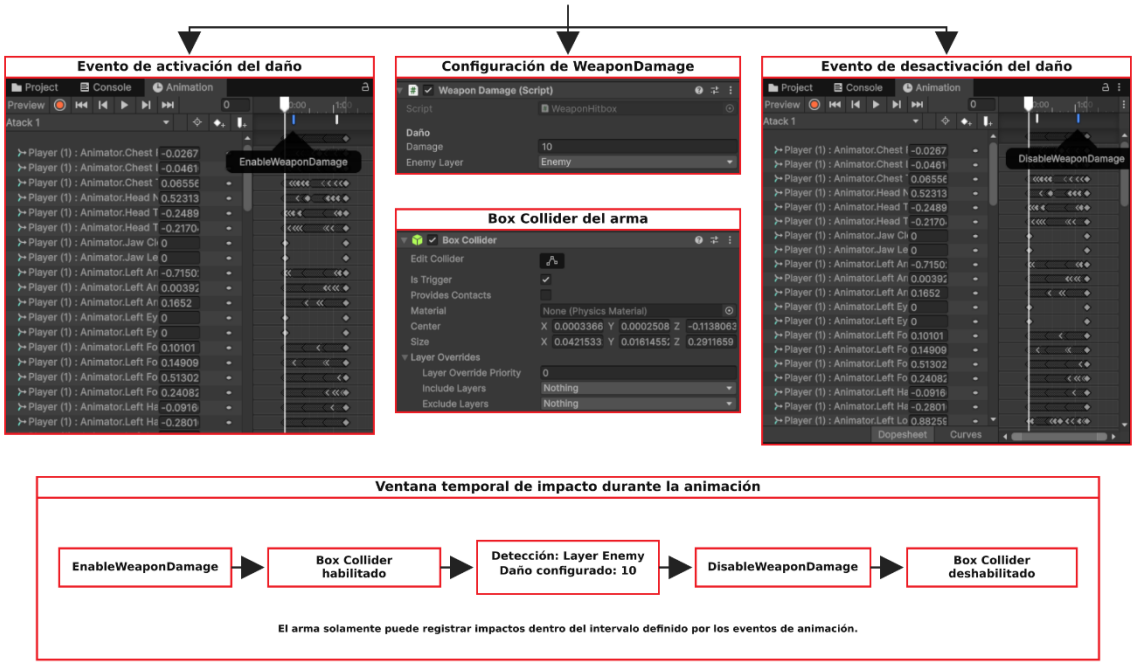
Los ataques tienen prioridad sobre ciertos ataques por lo tanto mientras se esté ejecutando el combo se bloquea temporalmente el movimiento, esquive o la habilidad especial. Al terminar el ataque el Animator vuelve a restablecer la disponibilidad de estos parámetros.

Se agrega el script PlayerAnimationEvent el cual permite sincronizar el daño del arma con el movimiento de la espada dentro de la animación. Este funciona mediante dos eventos que se colocan en el frame indicado dentro de la animación el primer evento se encarga de activar la colisión del arma mientras que el segundo lo desactiva con el objetivo de solo hacer daño dentro de la animación de ataque.

Mientras que los eventos se encargan de activar las colisiones, WeaponDamage se encarga de leer los impactos de las mismas, este script filtra mediante LayerMask las capas de los objetos, si detecta que un objeto tiene la capa Enemy entonces este se comunica con el gestor de vida del enemigo para registrar el daño. En la Ilustración 30 se puede observar el funcionamiento del sistema WeaponDamage durante los ataques.

Ilustración 30

Funcionamiento de WeaponDamage



Nota: Elaboración propia

- **US-04: habilidad especial**

Asignamos la entrada de la habilidad especial al clic derecho del ratón y se administró mediante PlayerCombat. Antes de ejecutar la acción el sistema comprueba que el jugador esté en el piso, no realice ningún esquive en ese momento, tenga permitido moverse y que haya finalizado el tiempo de reutilización.

Cuando se cumplen estas condiciones PlayerCombat activa este ataque permitiendo que se reproduzca la animación specialAttack, cancela activaciones pendientes de combo y registra el momento en el cual puede volver a usar la habilidad especial. En la configuración final se estableció un tiempo de 10 segundos antes de poder volver a utilizar esta habilidad.

El script SpecialAttackArea se encarga de detectar el impacto, este se vincula a un punto denominado AttackCenter el cual se utiliza como el centro del ataque. En el momento indicado de la animación SpecialAttackImpact se encarga de activar las partículas y aplicar daño.

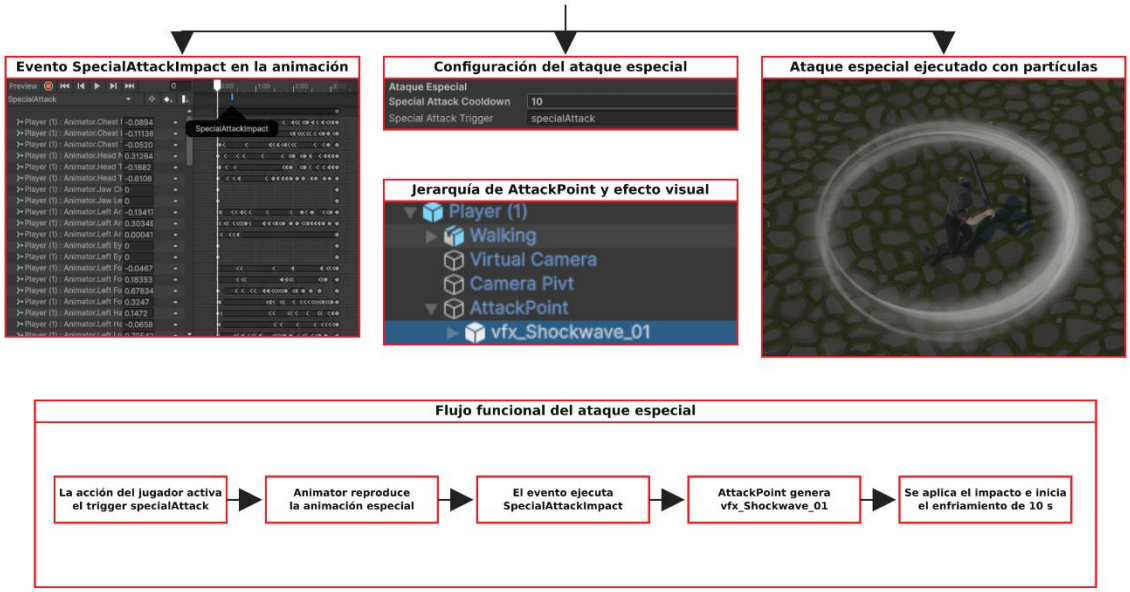
Se utiliza Physics.OverlapSphere para localizar a los enemigos utilizamos un radio de impacto de 7.55 unidades y esto permite que se ejecute el daño en todos los objetos con la capa enemy que se encuentren dentro de este radio.

SpecialAttackArea utiliza un HashSet de EnemyCombat para impedir que un enemigo con varios colliders reciba daño más de una vez. Los enemigos que se encuentran muerto son ignorados completamente por este script.

El sistema de partículas es posicionado mediante AttackCenter, Una corrutina se encarga de la duración de las partículas y desactiva este efecto cuando termina la animación, evitando que estas permanezcan activas después del ataque. En la Ilustración 31 se puede observar el funcionamiento del sistema de ataque especial del personaje.

Ilustración 31

Sistema de ataque especial



Nota: Elaboración propia

- **US-05: sistema de vida, daño y muerte**

PlayerHealth es el encargado de administrar la vida del personaje. Se configuró un valor máximo de 100 puntos de vida y, al iniciar el juego el jugador toma este valor de vida actual. Al impactar un ataque con daño valido, el sistema descuenta los puntos de vida hasta llegar a un límite de cero.

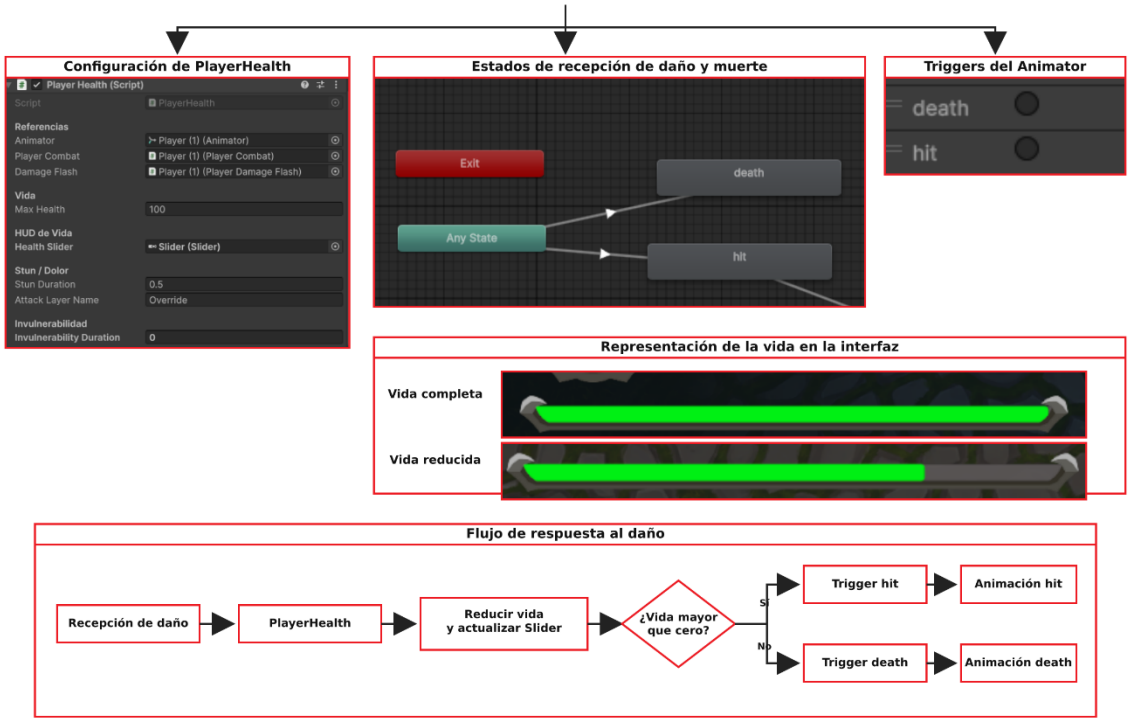
Para evitar que varios enemigos produzcan daño a la vez se utiliza un periodo de invulnerabilidad. Durante este periodo todos los impactos son ignorados durante 1.2 segundos. De igual forma se implementó un aturdimiento de 0.5 segundos a la hora de recibir daño de un impacto para tener un feedback visual de que el jugador sufrió de un ataque.

PlayerHealth está conectado con el Animator para activar las animaciones de daño y muerte. Si la vida es menor a uno se establece el estado de muerte. Este estado

impide nuevas acciones de movimiento o combate y activa la animación de muerte. Esta validación evita que el jugador reciba daño o ejecute mecánicas después de su muerte. En la Ilustración 32 se puede observar el funcionamiento del sistema PlayerHealth para gestionar la salud del personaje.

Ilustración 32

Funcionamiento de PlayerHealth



Nota: Elaboración propia

- **Interfaz de vida**

La vida del personaje se representa visualmente en la interfaz del juego para esto utilizamos un Slider de Unity UI. PlayerHealth obtiene una referencia de este mismo y actualiza su valor en base de los puntos de vida que tiene y se actualiza cada que el jugador recibe daño o se cura.

Utilizamos un relleno verde para representar la vida del personaje y un fondo de color rojo para identificar el daño que ha recibido. Esto nos permite identificar

directamente el estado en el que se encuentra el personaje sin tener que recurrir a valores numéricos. Este efecto visual da a entender un estado de peligro para que el jugador tome las medidas adecuadas.

A diferencia del personaje jugable la vida de los enemigos se visualiza mediante un círculo en sus pies. Este representa su porcentaje de vida mediante el color, empieza en un todo verde fuerte y se va degradando llegando a un amarillo y terminando en rojo que representa su pérdida total de vida. Este se actualiza mediante el script EnemyCombat que guarda el valor de vida actual de los enemigos. La Tabla 14 presenta los principales parámetros configurados para los sistemas desarrollados durante el Sprint 2.

Tabla 14

Configuraciones del sistema para el Sprint 2

Parámetro	Valor final	Componente
Daño del ataque normal	10	WeaponDamage
Cooldown del ataque especial	10 segundos	PlayerCombat
Daño del ataque especial	50	SpecialAttackArea
Radio del ataque especial	7.55	SpecialAttackArea
Vida máxima del jugador	100	PlayerHealth
Duración del aturdimiento	0.5 segundos	PlayerHealth

Nota: Elaboración propia

- **Resultados del Sprint 2**

Dentro de la escena de pruebas se realizaron enfrentamientos controlados para poder verificar que funcionen las animaciones tanto en el combo como en el sistema

de eventos para activar colliders y partículas, también medimos el correcto funcionamiento de la interfaz, recibimiento de daño y el estado de muerte.

Para la habilidad especial se colocó múltiples enemigos dentro y fuera del radio de ataque para comprobar que únicamente recibían daño los que se encontraban centro de esta, el sistema de reutilización de la habilidad se redujo a dos y a cinco segundos para comprobar el funcionamiento del mismo y finalmente se validó la reducción de vida y la activación de la animación.

Como resultado final se obtuvo un sistema de combate funcional compuesto por combos y un ataque especial, recepción de daño, muerte y visualización de estado en la interfaz. Esto establece una base para la posterior integración de enemigos.

4.2.3. Sprint 3

Este sprint se orientó en las interacciones con NPCs no hostiles. Su implementación permite que el jugador detecte áreas de diálogo, iniciar una interacción y conversar con NPCs mediante una visualización progresiva de texto acompañada por el nombre e imagen del personaje que esté hablando en ese momento.

El sistema se dividió en tres componentes principales. NPCDialogue se encarga de administrar las colisiones del jugador para detectar si entro a una zona de diálogo, DialogueData almacena la información de las conversaciones y DialogueManager se encarga de controlar la interfaz, el avance progresivo del texto y pausar el juego durante el diálogo. La Tabla 15 presenta la planificación general correspondiente al Sprint 3.

Tabla 15

Planificación de sprint 3

Historia de usuario	Tareas desarrolladas	Entregable
US-07: Interacción con NPCs. Como jugador, necesito interactuar con los NPCs no hostiles al acercarme a ellos para iniciar una conversación.	Configurar un collider de interacción; detectar al objeto con la etiqueta Player; mostrar el mensaje de interacción; recibir la entrada de la tecla F; iniciar y finalizar la conversación.	Sistema capaz de detectar al jugador e iniciar un diálogo con el NPC seleccionado.
US-08: Sistema de diálogos. Como jugador, necesito visualizar los nombres, retratos y líneas de los participantes para comprender las conversaciones.	Crear DialogueData; almacenar participantes y líneas; programar DialogueManager; implementar texto progresivo; mostrar nombres y retratos; suspender y restablecer los controles.	Sistema de conversaciones progresivas con múltiples participantes e interfaz visual.

Nota: Elaboración propia

- **Interacción con NPCs no hostiles**

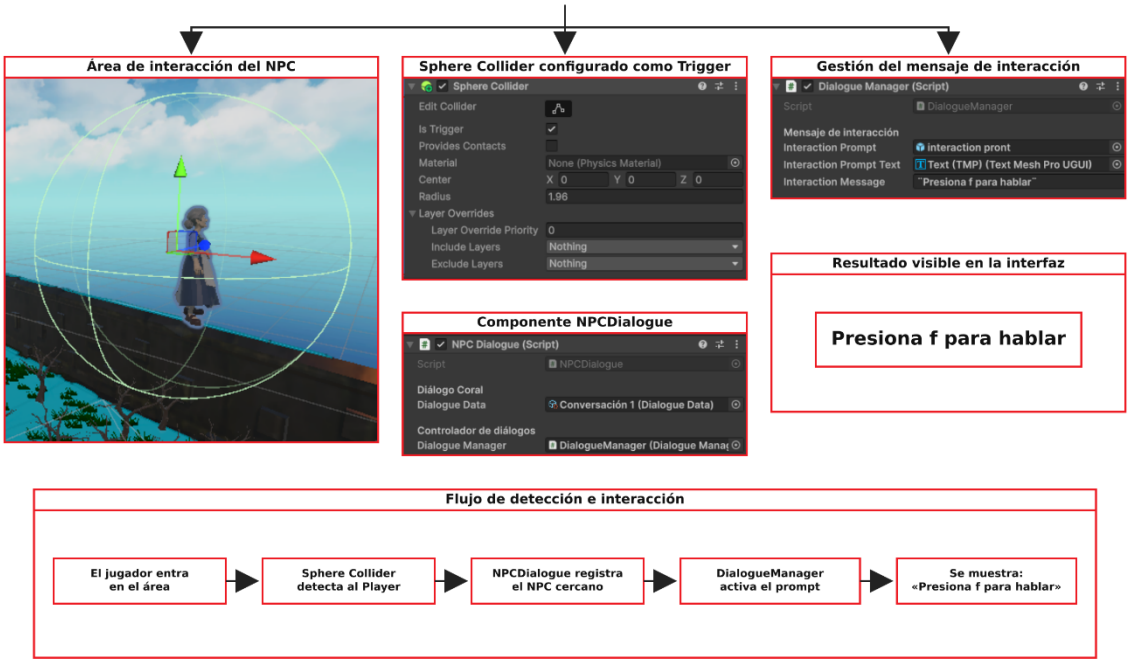
Se incorpora el script NPCDialogue en cada personaje no hostil. Este componente pide obligatoriamente un Collider, el cual configuramos como trigger para establecer un área de diálogo alrededor del NPC. El área funciona como un detector no produce una colisión física con el jugador.

Cuando un objeto entra en este Collider, OnTriggerEnter verifica que el objeto tenga la etiqueta “Player”. Si este es el caso NPCDialogue se comunica con DialogueManager para permitir una interacción. Esto evita que los scripts se activen con otro tipo de colisiones como la de los enemigos o la de otros NPCs.

Cuando DialogueManager es informado de la presencia del jugador este activa un texto en la interfaz para avisarle al jugador que tiene oportunidad de interactuar en este caso el texto es “Presione F para hablar” si el usuario se aleja del área asignada para interactuar OnTriggerExit se encarga de eliminar la referencia por lo cual se oculta el mensaje de aviso. En la Ilustración 33 se puede observar el funcionamiento del sistema de interacción del videojuego.

Ilustración 33

Sistema de interacción



Nota: Elaboración propia

NPCDialogue utiliza una referencia al archivo DialogueData correspondiente a cada personaje. Esto permite reutilizar el sistema de detección con diferentes NPCs, puesto que cada uno tiene asignada una conversación diferente, pero utilizando el mismo Script. También mantiene una referencia a DialogueManager el cual es el mismo para toda la escena.

La entrada para la interacción la administra DialogueManager con el uso de keyboard de inputSystem. Cuando el jugador presiona la tecla asignada, en este caso la F, se ejecuta el método StartDialogue. Antes de iniciar la conversación el sistema comprueba que NPC tenga guardada una conversación en DialogueData.

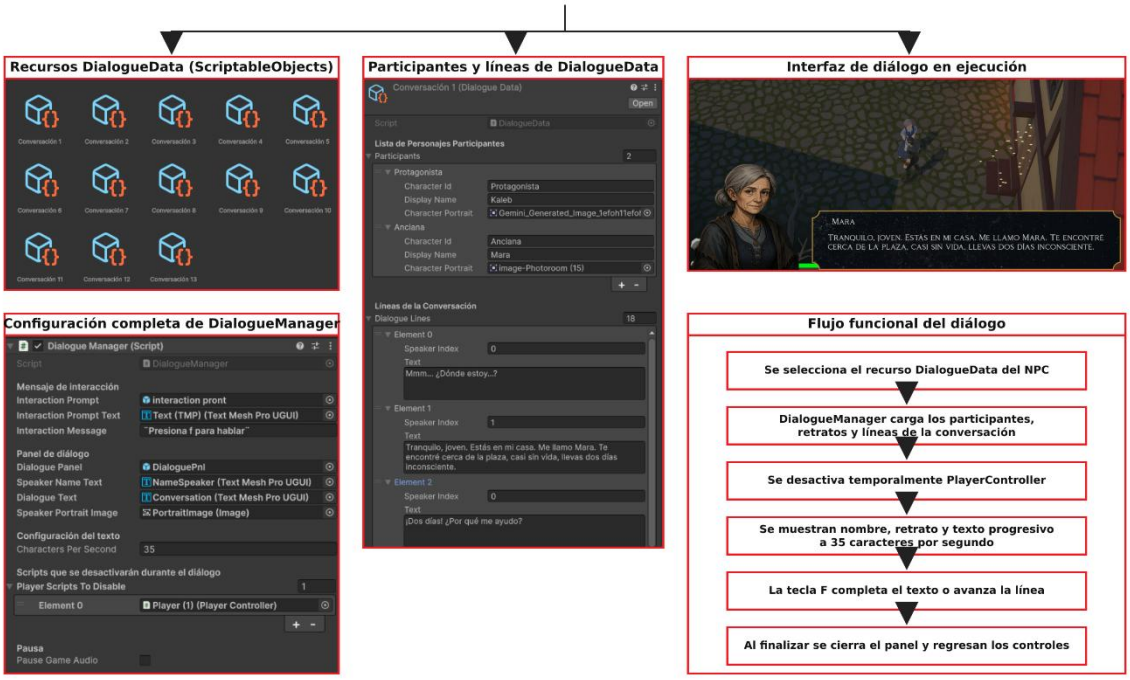
- **US-08: Sistema de diálogos**

Para el funcionamiento de DialogueData se utilizó Scriptable Objectss como sistema de almacenamiento de información para poder separar la lógica de programación de los textos narrativos. Cada archivo puede crearse desde el menú de Unity y se modifican desde el inspector sin cambiar el código.

El archivo creado contiene una lista de participantes y otra de líneas de texto. Cada participante se representa mediante DialogueCharacter, estructura que permite almacenar un identificador interno, el nombre y la imagen del personaje. Por otro lado, DialogueLine contiene el texto que se mostrara a forma de diálogo durante el juego. En la Ilustración 34 se puede observar la estructura y funcionamiento del sistema de diálogos implementado.

Ilustración 34

Sistema de diálogos



Nota: Elaboración propia

Se desarrolla la interacción con NPCs no hostiles. NPCDialogue se encarga de detectar al jugador, DialogueData almacena la información de los personajes que participan en la conversación y DialogueManager administra la interfaz y el avance del texto. Como resultado se obtuvo un sistema de conversación que permite mostrar el diálogo en texto, nombres y retratos de los personajes.

DialogueManager se encarga de leer la información guardad. Al momento de presentar una línea, primero obtiene su speakerIndex y comprueba que el participante del diálogo sea válido. Luego actualiza el nombre, el retrato y el texto dentro de la interfaz. Si el personaje no cuenta con una imagen este apartado se oculta para no mostrar un recuadro vacío.

El sistema comprueba errores de validación. Si una línea de código no identifica al participante o este no fue asignado se muestra los signos de interrogación: "???" para dar a entender que no se validó una referencia y se genera una advertencia en la consola. Esto facilita identificar errores durante la configuración de los diálogos.

Se implementó un sistema de escritura progresivo. Cada línea esta almacena desde el principio del diálogo, pero se va mostrando de forma progresiva y con una limitada cantidad de caracteres. El valor inicial se estableció en 35 caracteres por segundo pero este valor es modificable para pasar el texto más rápido o más lento, además se agregó CompleteCurrentLine que omite la progresión lenta de texto y muestra toda la línea al presionar la tecla F.

Time.unscaledDeltaTime permitió que el juego se pause a excepción del texto. Este valor permite que aún se puedan actualizar las líneas de diálogo, aunque Time,timeScale se encuentre en un valor de cero. De esta manera el diálogo puede pausar las mecánicas del juego sin afectar a este mismo.

Para prevenir bugs al momento de pausar el juego OnDisable comprueba que exista un diálogo activo, si por algún motivo DialogueManager se desactiva de forma inesperada el juego se reactivara automáticamente. La Tabla 16 presenta los principales parámetros configurados para el funcionamiento del sistema de diálogos.

Tabla 16

Configuración principal del diálogo

Parámetro	Configuración	Función
Tecla de interacción	f	Inicia, completa o avanza la conversación.
Mensaje de interacción	Press F to speak	Informa que existe un NPC disponible.

Velocidad del texto	35 caracteres por segundo	Controla el efecto de escritura progresiva.
Pausa de la escena	Time.timeScale = 0	Suspende las mecánicas durante el diálogo.
Datos narrativos	DialogueData	Almacena participantes, retratos y líneas.
Detección	Collider configurado como trigger	Reconoce la entrada y salida del jugador.

Nota: Elaboración propia

- **Resultados del Sprint**

Las pruebas se realizaron integrando varios NPCs en el mapa cada uno con diferentes diálogos, nombres e imágenes, se comprobó que el área de diálogo funcione correctamente al momento de entrar o salir de la colisión, que la tecla f iniciara correctamente la interacción y que el texto apareciera de manera progresiva.

Como resultado del Sprint 3 se obtuvo un sistema de interacciones reutilizable. Estos están separados entre detección, almacenamiento de datos y administración de interfaz posibilitando el generar nuevas conversaciones sin limitación de personajes y sin la necesidad de cambiar el código.

4.2.4. Sprint 4

En este sprint se implementó el comportamiento de IA mediante una máquina de estados finita. Esta etapa se enfocó en los agentes virtuales y sus diferentes estados como reposo, patrullaje, perseguir y regresar a su posición inicial.

Para desarrollar la IA es importante primero tener listo el control del personaje y el sistema de combate, debido a que estas acciones son necesarias para comprobar el funcionamiento adecuado de la FSM dependiendo de cada situación dentro del juego.

Dividimos la historia de usuario US-09 en diferentes tareas técnicas relacionadas con la arquitectura de estados que necesitamos, la configuración de los comportamientos, la navegación y las condiciones utilizadas para cambiar entre acciones. La Tabla 17 presenta la planificación de las tareas correspondientes al Sprint 4.

Tabla 17

Planificación de Sprint 4

Id de tarea	Tarea desarrollada	Entregable
T-09.1	Crear la clase base AIState y los estados utilizados por los enemigos.	Arquitectura modular de estados.
T-09.2	Crear AIStateTemplate y AIBehaviorTemplate para configurar los comportamientos desde el Inspector.	Plantillas reutilizables mediante Scriptable Objects.
T-09.3	Integrar EnemyAI con NavMeshAgent y configurar puntos de patrulla.	Navegación autónoma por el escenario.
T-09.4	Implementar las condiciones de detección, persecución y pérdida del objetivo.	Cambio controlado entre los estados de la FSM.
T-09.5	Programar el retorno del enemigo y comprobar su comportamiento dentro de la escena.	Enemigo capaz de regresar a su posición inicial.

Nota: Elaboración propia

- **US-09: Máquina de estados finita de los enemigos**

EnemyAI es el componente principal para el funcionamiento de la máquina de estados finita, este script almacena el estado en el que se encuentra actualmente el

enemigo, el objetivo, los puntos de patrulla, el Animator y la referencia al comportamiento que se configuró. Se encarga de administrar la FSM y permite que el personaje ejecute una sola acción a la vez.

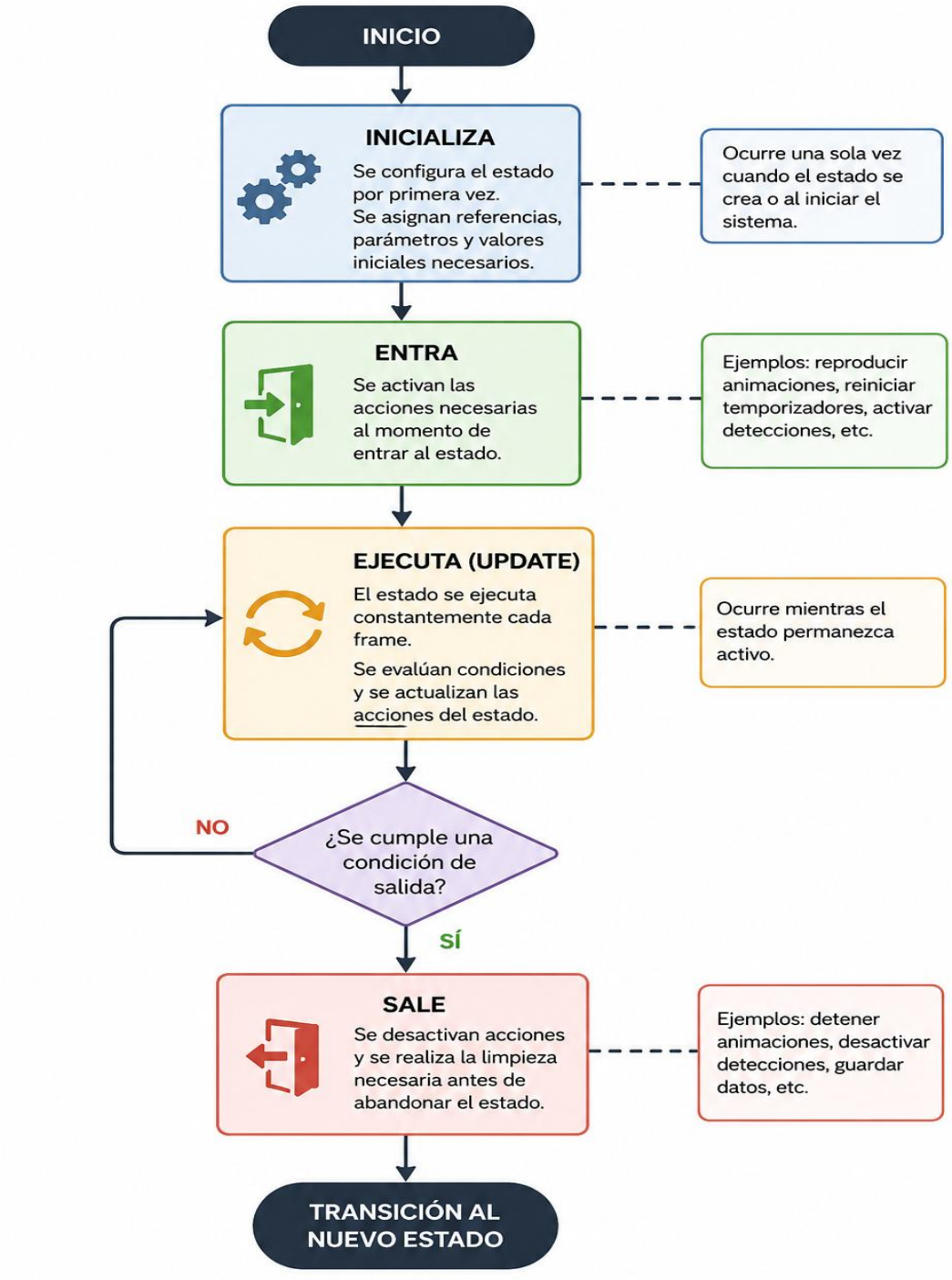
La clase abstracta AIState se utiliza como base para los estados. Esta contiene los métodos Initialize, Enter, Execute, Exit. Cada estado hereda de esta clase, pero implementando su propia lógica.

Cada uno de estos métodos tiene las funciones necesarias para el cambio de estados. Initialize prepara el estado mediante las referencias necesarias que recibe del juego, Enter se ejecuta cuando comienza un nuevo comportamiento, Execute contiene la lógica del estado mientras permanece activo y Exit finaliza el estado.

Cuando se requiere un cambio de estado EnemyAI primero finaliza su comportamiento actual mediante Exit, luego establece un nuevo comportamiento en Enter y a partir de ese momento Execute permanece activo hasta que se cumpla otro cambio. En la Ilustración 35 se puede observar el flujo de ejecución de los principales métodos utilizados en la FSM.

Ilustración 35

Flujo de métodos FSM



Nota: elaboración propia

- **Configuración mediante plantillas**

AIStateTemplate permite separar los datos de la lógica implementada, esta clase hace la función de ser una base para la plantilla específicas de cada estado y almacena los parámetros que se pueden modificar como la velocidad, distancia y tiempos de espera.

Las plantillas se crearon mediante Scriptable Objectss, por lo cual se modifican desde el inspector y se pueden reutilizar en distintos enemigos. De esta manera podemos generar más de un tipo de enemigos que utilicen la misma clase, pero con diferentes valores sin tener que duplicar el código.

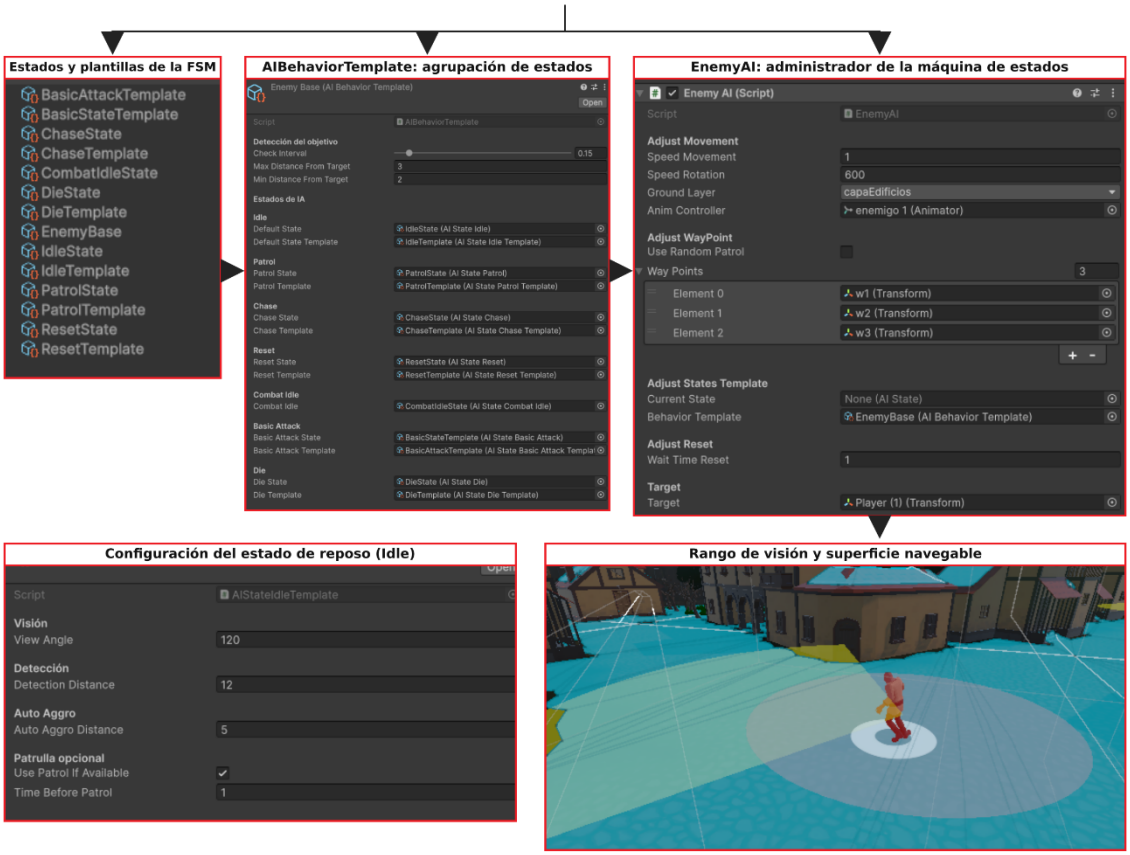
AIBehaviorTemplate es donde se agrupan todos los estados que componen el comportamiento de un enemigo. Almacena condiciones generales de detección. EnemyAI necesita una referencia de este componente y utiliza almacenados para administrar los estados guardados.

Este tipo de organización permite crear variaciones en enemigos. Su comportamiento se puede alterar sin necesidad de tocar el código o modificar EnemyAI. Únicamente configuramos las variables de las plantillas.

Esta estructura permite ampliar el sistema de enemigos con facilidad, debido a que un nuevo agente puede utilizar las mismas clases y estados ya implementados, modificando únicamente los valores almacenados en las plantillas. De esta forma se puede generar más variantes de enemigos con diferentes velocidades, distancia de detección y parámetros de comportamiento sin duplicar la lógica de programación ni modificar la clase EnemyAI. En la Ilustración 36 se puede observar el funcionamiento general de la FSM implementada para los enemigos.

Ilustración 36

Funcionamiento de FSM



Nota: Elaboración propia

- **Navegación y puntos de patrulla**

Para el desplazamiento de los agentes utilizamos el componente de Unity NavMeshAgent, este calcula las rutas en las que se permite navegar dentro del mapa. Previamente configuramos el mapa mediante AI Navegation para reconocer el terreno transitable y evitar los obstáculos.

EnemyAI guarda una lista de distintos puntos dentro del mapa llamados waypoints. En la configuración podemos asignar estos puntos y definir si queremos

que el enemigo los siga en orden o de forma aleatoria. El agente selecciona estos puntos y NavMeshAgent busca la forma más corta de llegar a ellos.

NavMeshAgent se configuró con la velocidad en 2.5, aceleración de 8 y velocidad angular de 900. La distancia de frenado en 0,5 para impedir que el enemigo colisione directamente con objetos del destino. La Tabla 18 presenta los principales parámetros configurados en EnemyAI para el comportamiento de los enemigos.

Tabla 18

Configuración de EnemyAI

Parámetro	Valor	Función
Speed	2,5	Controla la velocidad del agente.
Angular Speed	900	Determina la rapidez con la que puede girar.
Acceleration	8	Controla el incremento de velocidad.
Stopping Distance	0,5	Establece la distancia aceptada para detenerse.
Radius	0,5	Representa el espacio ocupado por el agente.
Height	2	Define la altura utilizada durante la navegación.
Waypoints	Indefinidos	Establece los destinos de la patrulla.
Wait Time Reset	1 segundo	Tiempo aplicado al completar el retorno.

Nota: Elaboración propia

- **Estado de reposo**

Cuando el enemigo no tiene ningún objetivo ni debe desplazarse a ningún waypoint se activa el estado de reposo. Este estado únicamente tiene activa las

comprobaciones necesarias para cambiar de estado y cuenta con una animación de Idle.

Este estado funciona como comportamiento inicial dentro del juego. Si el jugador entra en el campo de visión o el rango establecido de ataque, EnemyAI manda a cambiar de estado.

- **Estado de patrulla**

El estado de patrulla utiliza los waypoint que se asignaron a EnemyAI y administra el recorrido entre ellos. NavMeshAgent recibe los destinos y calcula la ruta de llegada. Cuando el agente alcanza su destino pasa al siguiente waypoint y mantiene este comportamiento en bucle.

Mientras está en el estado de patrulla se comprueba periódicamente la distancia del jugador. esta evaluación se puede configurar en AIBehaivorTemplate, evitando ejecutar la detección todo el tiempo. Esto con la finalidad de optimizar el videojuego.

- **Estado de persecución**

Cuando el jugador entra en las condiciones de detección, se convierte en el objetivo del enemigo entonces comienza el estado de persecución. Durante este estado NavMeshAgent desplaza al enemigo hacia el jugador calculando la ruta más rápida.

Mientras el objetivo permanezca dentro del límite de distancia permitido este estado permanecerá activo, si el enemigo logra alcanzar el rango de ataque, se detiene la persecución y da paso al comportamiento de esperar o realizar un ataque.

Si el jugador se aleja lo suficiente para rebasar el límite de persecución el enemigo cambia al estado de retorno. Esto evita que el jugador sea perseguido indefinidamente por todo el mapa.

- **Estado de retorno**

El enemigo guarda su posición inicial, cuando el enemigo pierde al jugador este regresa al punto inicial. Para esto se usa la referencia guardada y NavMeshAgent se encarga de desplazarlo hasta ese destino.

Cuando el agente retorna a su posición inicial, espera durante el tiempo que se configuró y luego regresa a su estado inicial de reposo o patrulla dependiendo de cómo este asignado. Esta función evita que los enemigos pierdan su área y se queden deambulando por el mapa sin ningún objetivo.

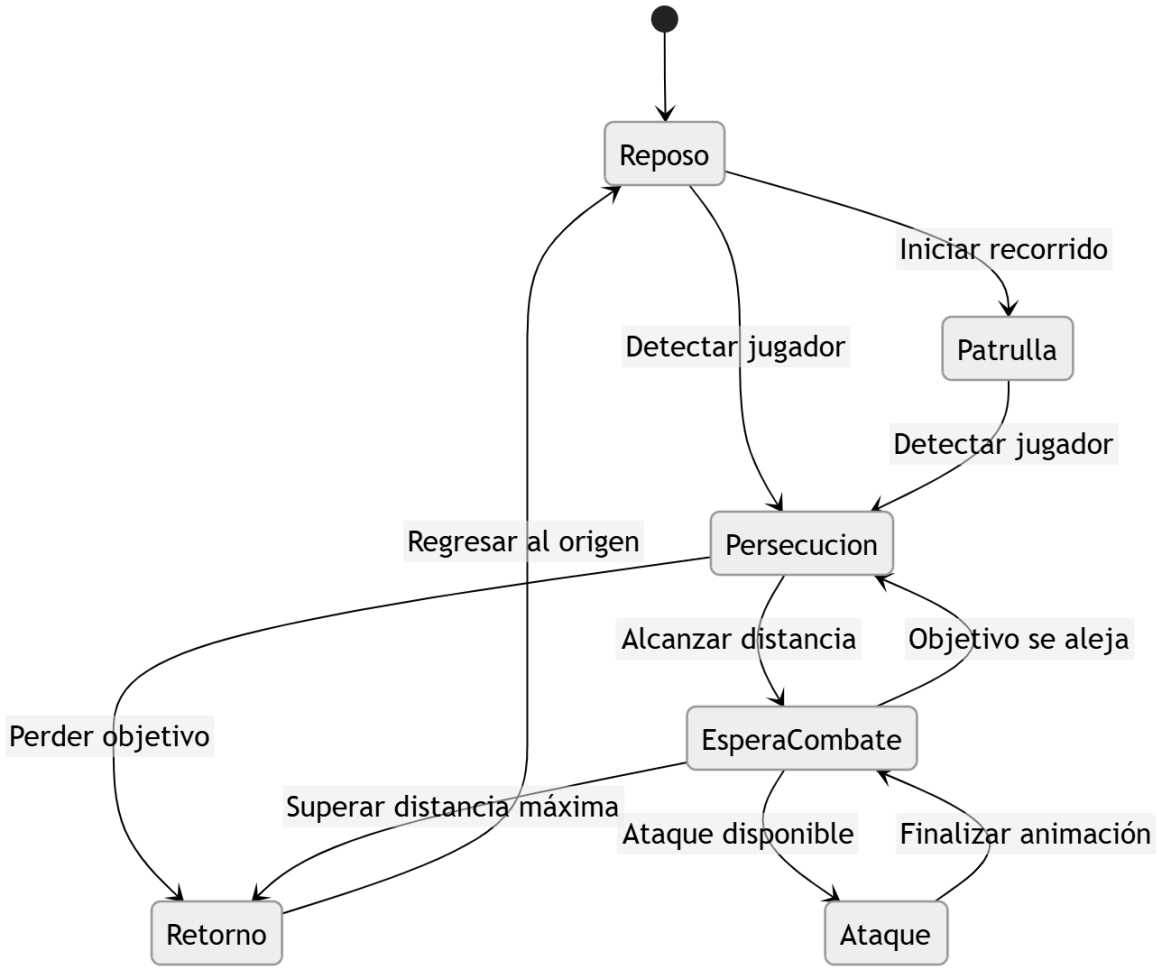
- **Flujo de la máquina de estados finita**

La transición entre los estados depende de la detección del jugador, su estado de vida, la distancia del jugador y la ubicación del enemigo. Estos parámetros permiten que el estado del enemigo cambie de forma controlada y que cada agente responda a las acciones que suceden en su entorno.

En la Ilustración 37 se puede observar el flujo de estados que conforman la FSM utilizada en el proyecto.

Ilustración ***37***

Flujo de la FSM del proyecto



Nota: Elaboración propia

- **Resultados del Sprint 4**

Se prueba la IA dentro de un entorno controlado y se utiliza diferentes posiciones y waypoints para comprobar el estado de patrulla y reposo del enemigo, se verificó que este recorra el mapa para llegar a los distintos puntos asignados y que detectara al jugador cuando entre en rango.

También se comprueba si NavMeshAgent calcula las rutas más cortas para llegar al objetivo, al igual si evita los obstáculos y se desplaza solamente por el terreno

asignado. La persecución actualiza el destino del enemigo y el estado de retornar lo hace regresar a su posición inicial.

Como resultado se obtuvo que un enemigo es capaz de administrar sus comportamientos mediante la FSM configurable. Esta estructura nos permitió tener separada la lógica de programación de la configuración de variables para la máquina de estados finita.

4.2.5. Sprint 5

EL último sprint se orientó a incorporar el sistema de combate para los enemigos y conecta este sistema con la máquina de estados. Esta etapa permitió que los agentes puedan ejecutar ataques, causar daño al personaje principal, recibir impactos y morir. La Tabla 19 presenta la planificación de las tareas desarrolladas durante el Sprint 5.

Tabla 19

Planificación del sprint 5

ID de tarea	Tarea desarrollada	Entregable
T-10.1	Implementar la vida, recepción de daño y muerte mediante EnemyCombat.	Enemigo capaz de recibir impactos y morir.
T-10.2	Integrar los ataques con Animator y EnemyAttackHitbox.	Ataques sincronizados con las animaciones.
T-10.3	Conectar EnemyCombat con PlayerHealth.	Intercambio de daño entre el enemigo y el jugador.
T-10.4	Integrar el combate con los estados de la FSM.	Enemigo capaz de perseguir, atacar y detenerse al morir.

Nota: Elaboración propia

- **US-10: sistema de combate de los enemigos**

Se implementa el script EnemyCombat el cual se encarga de administrar el sistema de vida, ataques, recepción de daño y muerte del enemigo. Este componente hereda directamente de Combat, donde se generan las propiedades comunes del sistema para tener un sistema reutilizable.

Se configuró la vida máxima del enemigo en 100 puntos. Cuando WeaponDamage o SpecialAttackArea detectan un golpe valido, buscan el componente EnemyCombat y este ejecuta el método TakeDamage. El mismo se encarga de reducir la vida y actualizar la interfaz dentro del juego.

La interfaz se representa con una imagen para mostrar el nivel de vida usamos un Gradient de color para pasar de verde a rojo conforme disminuye o aumenta su vida. Esto nos permite identificar visualmente el estado del enemigo.

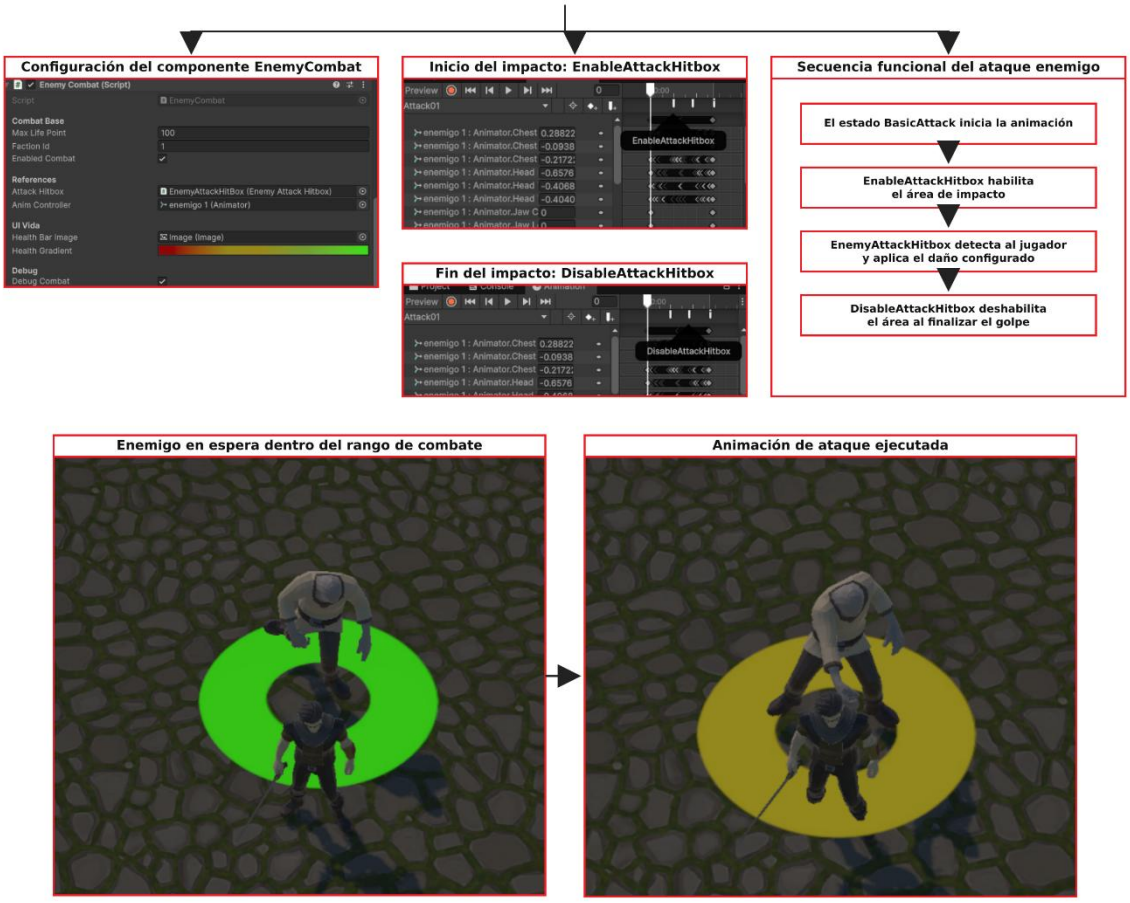
Para poder causar daño al jugador se implementó EnemyAttackHitbox. Al igual que el jugador este funciona mediante eventos en las animaciones activando y desactivando las colisiones dentro del golpe en los frames de inicio y final.

Cuando la colisión impacta la etiqueta “Player” el sistema conecta con PlayerHealth para aplicar el daño que género. La activación temporal de colisiones permite que el enemigo solamente produzca daño dentro de la animación de ataque.

El combate se conecta con la FSM. Cuando el jugador está dentro del rango de combate del enemigo, este pasa de estado de persecución para ingresar al estado de espera de combate. Si el ataque está disponible este se activa reproduciendo una animación de golpe. Después de terminar el golpe el enemigo vuelve al estado de espera para evaluar si continúa golpeando o tiene que regresar al estado de persecución o retorno. En la Ilustración 38 se puede observar el funcionamiento del sistema de combate implementado para los enemigos.

Ilustración 38

Sistema de combate en enemigos



Nota: Elaboración propia

- **Resultado del Sprint 5**

Dentro de las pruebas se evalúa que los ataques generen daño al jugador únicamente durante la animación de ataque, la recepción de impacto por parte del jugador y que el enemigo reciba daño gracias a los ataques normales y especiales y la actualización de la barra de vida.

Con este Sprint se completa el flujo principal de exploración, persecución y combate del juego.

4.3. Pruebas y resultados

Durante el desarrollo los sprints se realizaron pruebas técnicas orientadas a verificar el correcto funcionamiento de los sistemas programados, incluyendo las transiciones de estados de la FSM, detección de colisiones, ejecución de ataques, recepción de daño y mecánicas del personaje principal. Estas pruebas se realizaron directamente dentro del entorno de Unity observando la respuesta de los sistemas ante las acciones y eventos configurados. Posteriormente se realizó una evaluación con usuarios.

Una vez integrados los sistemas principales del prototipo, se realizó una evaluación con usuarios para conocer su opinión acerca del funcionamiento del videojuego. A diferencia de las pruebas técnicas realizadas anteriormente, esta evaluación se centró en recopilar el feedback de los participantes con respecto a los controles, sistema de combate y la dificultad de los enemigos.

4.3.1. Procedimiento de la prueba

La evaluación se realizó en un campamento tecnológico organizado por la Universidad Católica De Cuenca. Durante la presentación se dispuso un puesto equipado con una computadora donde los asistentes pudieron probar voluntariamente un ejecutable del videojuego. Al finalizar la experiencia, se invito a los participantes a responder una encuesta sobre distintos aspectos del prototipo. A partir de las respuestas obtenidas se conto un total de 20 usuarios con edades comprendidas entre los 12 y 30 años.

La participación fue de manera voluntaria, por lo cual los participantes no fueron seleccionados mediante algún criterio en específico, sino que correspondieron

a las personas que se acercaron al puesto y aceptaron probar el videojuego. La selección de este rango de edad respondió al publico objetivo del videojuego orientado a adolescentes y jóvenes adultos, permitiendo recopilar información de diferentes usuarios con distintos niveles de experiencia en videojuegos.

4.4. Conclusiones

La implementación de la IA mediante FSM permite estructurar el comportamiento de los enemigos bajo el uso de estados. Las pruebas durante el desarrollo permiten comprobar las transiciones entre estados y la respuesta de los enemigos frente a los comportamientos del personaje principal, cumpliendo el objetivo de implementar IA Utilizando FSM.

La programación de las mecánicas del personaje principal permitió integrar controles para el movimiento, esquive, combate, habilidad especial, sistema de vida y muerte e interacciones con NPCs dentro del prototipo, en la evaluación que se realizó, los controles tuvieron resultados de 4,60 sobre 5 y el sistema de combate 4,20 sobre 5, lo que evidencia una valoración que favorece estos campos por parte de los usuarios.

Como resultado del objetivo general se obtuvo un entorno jugable en el que se integran las mecánicas del personaje principal, los NPCs y el sistema de IA, la dificultad de los agentes virtuales obtuvo un promedio de 3,90 sobre 5, inferior a los resultados obtenidos en controles y combate, esto identifica al sistema de dificultad como uno de los principales aspectos que pueden mejorarse en futuras versiones.

4.5. Recomendaciones

Como línea de mejora futura, resulta conveniente ampliar de forma progresiva los sistemas desarrollados mediante nuevas mecánicas para el personaje, interacciones adicionales con NPCs, mejoras en la información de la interfaz y nuevos estados de comportamiento para los enemigos. La estructura modular implementada permite fáciles modificaciones dentro del proyecto.

Otra opción de mejora consiste en ampliar las mecánicas de combate del personaje jugable, debido a que el prototipo actual presenta un sistema de complejidad intermedia basado en ataques, esquive y habilidad especial. En futuras etapas se puede incorporar más combinaciones de ataques, opciones defensivas, contraataques y una mayor variedad de patrones ofensivos en los enemigos, con el propósito de generar enfrentamientos más variados y estratégicos.

Bibliografía

[1] I. Millington, *Artificial intelligence for games*. en The Morgan Kaufmann series in interactive 3D technology. Amsterdam Heidelberg: Elsevier Morgan Kaufmann, 2006.
[2] Y. A. Sekhavat, «Behavior Trees for Computer Games», *Int. J. Artif. Intell. Tools*, vol. 26, n.º 02, p. 1730001, abr. 2017, doi: 10.1142/S0218213017300010.
[3] J. Schell, *The art of game design: a book of lenses*, Reprinted. Amsterdam Heidelberg: Elsevier, 2010.
[4] S. Rabin, Ed., *Game AI pro 2: collected wisdom of game AI professionals*. Boca Raton: CRC Press, 2015. doi: 10.1201/b18373.
[5] K. Karpouzis y G. Tsatiris, «AI in (and for) Games», 2021, *arXiv*. doi: 10.48550/ARXIV.2105.03123.
[6] Z. Hu *et al.*, «Deep learning applications in games: a survey from a data perspective», *Appl Intell*, vol. 53, n.º 24, pp. 31129-31164, dic. 2023, doi: 10.1007/s10489-023-05094-2.
[7] «Scrum Metodologia». [En línea]. Disponible en: https://cristophervegacom.wordpress.com/metodologias-agiles-scrum-y-kanban/
[8] T. Guzsvinecz, «The correlation between positive reviews, playtime, design and game mechanics in souls-like role-playing video games», *Multimed Tools Appl*, vol. 82, n.º 3, pp. 4641-4670, ene. 2023, doi: 10.1007/s11042-022-12308-1.
[9] «Dark Souls III», arsTechnica.
[10] X. Wang *et al.*, «ClueCart: Supporting Game Story Interpretation and Narrative Inference from Fragmented Clues», en *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, Yokohama Japan: ACM, abr. 2025, pp. 1-26. doi: 10.1145/3706598.3713381.
[11] C. Leiva, «Vandal», Cómo conseguir todos los finales en Hollow Knight.
[12] M. Brandse, «Level Flow Patterns in Game Design», en *HCI in Games*, vol. 14046, X. Fang, Ed., en Lecture Notes in Computer Science, vol. 14046. , Cham: Springer Nature Switzerland, 2023, pp. 51-65. doi: 10.1007/978-3-031-35930-9_4.
[13] C. White, «Death's Door».
[14] G. N. Yannakakis y J. Togelius, «A Panorama of Artificial and Computational Intelligence in Games», *IEEE Trans. Comput. Intell. AI Games*, vol. 7, n.º 4, pp. 317-335, dic. 2015, doi: 10.1109/TCIAIG.2014.2339221.
[15] Y. Lu y W. Li, «Techniques and Paradigms in Modern Game AI Systems», *Algorithms*, vol. 15, n.º 8, p. 282, ago. 2022, doi: 10.3390/a15080282.
[16] S. Risi y M. Preuss, «From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI», *Künstl Intell*, vol. 34, n.º 1, pp. 7-17, mar. 2020, doi: 10.1007/s13218-020-00647-w.
[17] D. Harel, «Statecharts: a visual formalism for complex systems», *Science of Computer Programming*, vol. 8, n.º 3, pp. 231-274, jun. 1987, doi: 10.1016/0167-6423(87)90035-9.
[18] I. Millington y J. Funge, *Artificial Intelligence for Games*, 2.ª ed. CRC Press, 2018. doi: 10.1201/9781315375229.
[19] J. R. Quinlan, «Induction of decision trees», *Mach Learn*, vol. 1, n.º 1, pp. 81-106, mar. 1986, doi: 10.1007/BF00116251.
[20] I. D. Mienye y N. Jere, «A Survey of Decision Trees: Concepts, Algorithms, and Applications», *IEEE Access*, vol. 12, pp. 86716-86727, 2024, doi: 10.1109/ACCESS.2024.3416838.
[21] M. Ç. Uludağlı y K. Oğuz, «Non-player character decision-making in computer games», *Artif Intell Rev*, vol. 56, n.º 12, pp. 14159-14191, dic. 2023, doi: 10.1007/s10462-023-10491-7.
[22] N. A. Mas'udi, E. M. A. Jonemaro, M. A. Akbar, y T. Afirianto, «Development of Non-Player Character for 3D Kart Racing Game Using Decision Tree», *FIJ*, vol. 6, n.º 2, p. 51, jun. 2021, doi: 10.21111/fij.v6i2.4678.
[23] K. Fathoni, R. Y. Hakkun, y H. A. T. Nurhadi, «Finite State Machines for Building Believable Non-Playable Character in the Game of Khalid ibn Al-Walid», *J. Phys.: Conf. Ser.*, vol. 1577, n.º 1, p. 012018, jul. 2020, doi: 10.1088/1742-6596/1577/1/012018.
[24] Isaac, «Maquinas de estado finitas», Máquinas de estados finitos: la clave oculta de la Inteligencia artificial.

[25] K. Christoph, H. Dorothée, y V. Peter, «The Video Game Experience as “True” Identification: A Theory of Enjoyable Alterations of Players’ Self-Perception», *Communication Theory*, vol. 19, n.º 4, pp. 351-373, nov. 2009, doi: 10.1111/j.1468-2885.2009.01347.x.
[26] V. Erb, S. Lee, y Y. Y. Doh, «Player-Character Relationship and Game Satisfaction in Narrative Game: Focus on Player Experience of Character Switch in The Last of Us Part II», *Front. Psychol.*, vol. 12, p. 709926, sep. 2021, doi: 10.3389/fpsyg.2021.709926.
[27] M. O. Riedl y V. Bulitko, «Interactive Narrative: An Intelligent Systems Approach», *AI Magazine*, vol. 34, n.º 1, pp. 67-77, mar. 2013, doi: 10.1609/aimag.v34i1.2449.
[28] D. Hefner, C. Klimmt, y P. Vorderer, «Identification with the Player Character as Determinant of Video Game Enjoyment», en *Entertainment Computing – ICEC 2007*, vol. 4740, L. Ma, M. Rauterberg, y R. Nakatsu, Eds., en Lecture Notes in Computer Science, vol. 4740. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 39-48. doi: 10.1007/978-3-540-74873-1_6.
[29] R. Poivet, M. Lopez Malet, C. Pelachaud, y M. Auvray, «The influence of conversational agents’ role and communication style on user experience», *Front. Psychol.*, vol. 14, p. 1266186, dic. 2023, doi: 10.3389/fpsyg.2023.1266186.
[30] J. P. Rowe, E. Y. Ha, y J. C. Lester, «Archetype-Driven Character Dialogue Generation for Interactive Narrative», en *Intelligent Virtual Agents*, vol. 5208, H. Prendinger, J. Lester, y M. Ishizuka, Eds., en Lecture Notes in Computer Science, vol. 5208. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 45-58. doi: 10.1007/978-3-540-85483-8_5.
[31] M. Yin y R. Xiao, «How Should I Respond to “Good Morning?”: Understanding Choice in Narrative-Rich Games», en *Designing Interactive Systems Conference*, Virtual Event Australia: ACM, jun. 2022, pp. 726-744. doi: 10.1145/3532106.3533459.
[32] K. Sedig, P. Parsons, y R. Haworth, «Player–Game Interaction and Cognitive Gameplay: A Taxonomic Framework for the Core Mechanic of Videogames», *Informatics*, vol. 4, n.º 1, p. 4, ene. 2017, doi: 10.3390/informatics4010004.
[33] P. Sweetser y P. Wyeth, «GameFlow: a model for evaluating player enjoyment in games», *Comput. Entertain.*, vol. 3, n.º 3, pp. 3-3, jul. 2005, doi: 10.1145/1077246.1077253.
[34] R. Hunicke, «The case for dynamic difficulty adjustment in games», en *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, Valencia Spain: ACM, jun. 2005, pp. 429-433. doi: 10.1145/1178477.1178573.
[35] M. Schmierbach, M.-Y. Chung, M. Wu, y K. Kim, «No One Likes to Lose: The Effect of Game Difficulty on Competency, Flow, and Enjoyment», *Journal of Media Psychology*, vol. 26, n.º 3, pp. 105-110, ene. 2014, doi: 10.1027/1864-1105/a000120.
[36] C. Klimmt, C. Blake, D. Hefner, P. Vorderer, y C. Roth, «Player Performance, Satisfaction, and Video Game Enjoyment», en *Entertainment Computing – ICEC 2009*, vol. 5709, S. Natkin y J. Dupire, Eds., en Lecture Notes in Computer Science, vol. 5709. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1-12. doi: 10.1007/978-3-642-04052-8_1.
[37] R. M. Ryan, C. S. Rigby, y A. Przybylski, «The Motivational Pull of Video Games: A Self-Determination Theory Approach», *Motiv Emot*, vol. 30, n.º 4, pp. 344-360, dic. 2006, doi: 10.1007/s11031-006-9051-8.
[38] K. Schwaber, «SCRUM Development Process», en *Business Object Design and Implementation*, J. Sutherland, C. Casanave, J. Miller, P. Patel, y G. Hollowell, Eds., London: Springer London, 1997, pp. 117-134. doi: 10.1007/978-1-4471-0947-1_11.
[39] A. Alami y O. Krancher, «How Scrum adds value to achieving software quality?», *Empir Software Eng*, vol. 27, n.º 7, p. 165, dic. 2022, doi: 10.1007/s10664-022-10208-4.
[40] D. Dennehy y K. Conboy, «Going with the flow: An activity theory analysis of flow techniques in software development», *Journal of Systems and Software*, vol. 133, pp. 160-173, nov. 2017, doi: 10.1016/j.jss.2016.10.003.
[41] J. Garzas, «Scrum», La evolución del diagrama Scrum.
[42] M. O. Ahmad, J. Markkula, y M. Oivo, «Kanban in software development: A systematic literature review», en *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander: IEEE, sep. 2013, pp. 9-16. doi: 10.1109/SEAA.2013.28.
[43] M. O. Ahmad, J. Markkula, y M. Oivo, «Insights into the Perceived Benefits of Kanban in Software Companies: Practitioners’ Views», en *Agile Processes, in Software Engineering, and Extreme Programming*, vol. 251, H. Sharp y T. Hall, Eds., en Lecture Notes in Business Information Processing, vol. 251. , Cham: Springer International Publishing, 2016, pp. 156-168. doi: 10.1007/978-3-319-33515-5_13.

[44] H. Janeczko, «kanban», Revolucione su flujo de trabajo: el poder de los sistemas Kanban en la gestión de inventario.
[45] N. Lee, «Unity, a 2D and 3D Game Engine», en *Encyclopedia of Computer Graphics and Games*, N. Lee, Ed., Cham: Springer International Publishing, 2024, pp. 1942-1944. doi: 10.1007/978-3-031-23161-2_536.
[46] E. Coronado, S. Itadera, y I. G. Ramirez-Alpizar, «Integrating Virtual, Mixed, and Augmented Reality to Human–Robot Interaction Applications Using Game Engines: A Brief Review of Accessible Software Tools and Frameworks», *Applied Sciences*, vol. 13, n.º 3, p. 1292, ene. 2023, doi: 10.3390/app13031292.
[47] D. Laksono y T. Aditya, «Utilizing A Game Engine for Interactive 3D Topographic Data Visualization», *IJGI*, vol. 8, n.º 8, p. 361, ago. 2019, doi: 10.3390/ijgi8080361.
[48] C. Hardman, *Game Programming with Unity and C#: A Complete Beginner's Guide*. Berkeley, CA: Apress, 2020. doi: 10.1007/978-1-4842-5656-5.
[49] S. M. Cossu, *Beginning Game AI with Unity: Programming Artificial Intelligence with C#*. Berkeley, CA: Apress, 2021. doi: 10.1007/978-1-4842-6355-6.
[50] Capterra, «Unity», Unity - Opiniones, precios y características.
[51] M. Dhule, *Beginning Game Development with Godot: Learn to Create and Publish Your First 2D Platform Game*. Berkeley, CA: Apress, 2022. doi: 10.1007/978-1-4842-7455-2.
[52] A. Thorn, *Moving from Unity to Godot: An In-Depth Handbook to Godot for Unity Users*. Berkeley, CA: Apress, 2020. doi: 10.1007/978-1-4842-5908-5.
[53] Godot Engine, «Godot», Design interfaces with the Control nodes.
[54] F. Horozal, P. Reimer, y S. Scholze, «Tool Support for Architectural Pattern Selection and Application in Cloud-Centric Service-Oriented IDEs», en *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*, Ludwigsburg Germany: ACM, oct. 2023, pp. 53-61. doi: 10.1145/3624486.3624494.
[55] A. Leff y J. T. Rayfield, «Web-application development using the Model/View/Controller design pattern», en *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Seattle, WA, USA: IEEE Comput. Soc, 2001, pp. 118-127. doi: 10.1109/EDOC.2001.950428.
[56] «Arquitectura MVC». [En línea]. Disponible en: https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/
[57] «Arquitectura por capas». [En línea]. Disponible en: https://blog.walthercuro.com/programacion-por-capas/
[58] «arquitectura por eventos». [En línea]. Disponible en: https://www.akamai.com/es/glossary/what-is-event-driven-architecture

Anexos

Instrumento de evaluación

Para elaborar el cuestionario se recurrió a la escala de Likert ya que esta permite visualizar de mejor manera el grado de satisfacción que tuvieron en cada apartado. Cada usuario eligió una respuesta de acuerdo a su nivel de satisfacción respecto a las preguntas presentadas. La Tabla 20 presenta la escala de Likert utilizada para estructurar el instrumento de evaluación.

Tabla 20

Estructura de la encuesta

Valor	Interpretación
1	Totalmente en desacuerdo
2	En desacuerdo
3	Neutral
4	De acuerdo
5	Totalmente de acuerdo

Nota: Elaboración propia

Las preguntas se organizaron en distintos apartados técnicos vinculados a la programación del personaje principal y NPCs. La tabla 21 presenta las preguntas planteadas a los usuarios.

Tabla 21

preguntas a usuarios

Preguntas a usuarios
¿Los controles respondieron correctamente?
¿El sistema de combate fue fácil de comprender?
¿Los enemigos presentaron un nivel de dificultad adecuado?

Nota : Elaboración propia

Resultados de la escala de Likert

Se obtuvo los resultados del videojuego en general se desarrollaron 9 preguntas acerca de la programación, los modelos 3D, experiencia de usuario y narrativa por lo cual únicamente se toma en cuenta el apartado de programación. En la Ilustración 39 se pueden observar los resultados obtenidos durante la evaluación del proyecto.

Ilustración 39

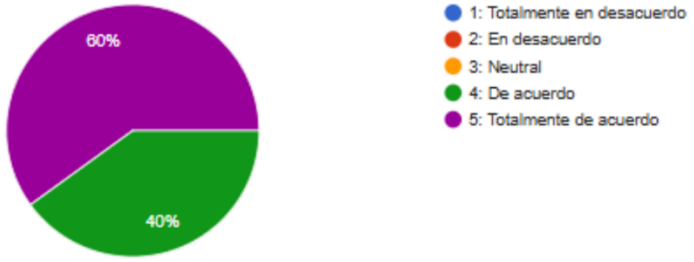
Resultados de evaluación

a) Respuesta de los controles del personaje

Pregunta 1: ¿Los controles respondieron correctamente?

Copiar gráfico

20 respuestas

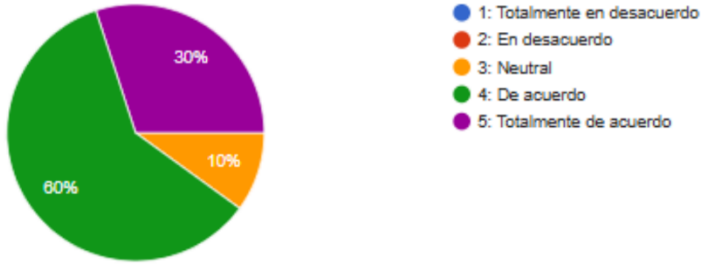


b) Comprensión del sistema de combate

Pregunta 2: ¿El sistema de combate fue facil de comprender?

Copiar gráfico

20 respuestas

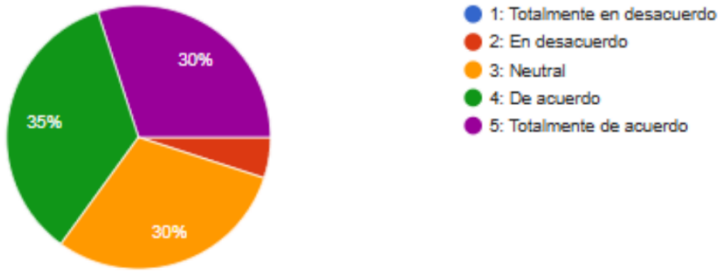


c) Dificultad presentada por los enemigos

Pregunta 3: ¿Los enemigos presentaron un nivel de dificultad adecuado?

Copiar gráfico

20 respuestas



Nota: Elaboración propia

Como se muestra en la ilustración 39 se obtuvo que la primera pregunta, “¿los controles respondieron correctamente?” se relaciona con los controles del juego donde, el 60% de los participantes indico estar totalmente de acuerdo y 40% de acuerdo. El promedio obtenido es de 4.60 sobre 5. Esto representa que los usuarios percibieron una respuesta adecuada al control de movimiento y acciones del personaje.

Respecto al sistema de combate en la pregunta “¿El sistema de combate fue fácil de comprender?” los resultados son, 30% totalmente de acuerdo, 60% de acuerdo y 10% neutral. El promedio es 4.20 sobre 5 por lo tanto el 90% de los participantes tuvo una valoración positiva respecto al combate del juego.

La dificultad de los enemigos con la pregunta “¿los enemigos presentaron un nivel de dificultad adecuado?” tuvo resultados del 30% totalmente de acuerdo, 35% de acuerdo, 30% neutral y el 5% en desacuerdo. El promedio es de 3.90 sobre 5. Las respuestas neutrales y en desacuerdo indican que el apartado de dificultad presenta oportunidades de mejora en cuanto a su equilibrio.

Los resultados muestran que los sistemas de control y combate fueron valorados favorablemente por la mayoría de los participantes. Por otro lado, el sistema de dificultad presentó respuestas más variadas, por lo que se identifican oportunidades de mejora en cuanto a su equilibrio.