Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Ingeniería Inversa de la Jugabilidad: Análisis y Recreación de Mecánicas Innovadoras en Videojuegos

Memoria del Trabajo Fin de Grado

en Diseño y Desarrollo de Videojuegos

Autor:

Alejandro Campbell Legarreta

Tutor: David María Arribas

Julio 2024

«El conocimiento no tiene ningún poder si no puede ser compartido»
José A. Pallavicini

Agradecimientos

En primer lugar, quiero agradecer a mi tutor David María Arribas toda la ayuda y apoyo prestado durante el desarrollo de este trabajo y durante los 2 años de carrera en los cuales he tenido la suerte de contar con su presencia, no solo ha sido un profesor, sino también un gran mentor y una gran persona.

A mis padres, Inma y Alex, por su apoyo incondicional, tanto económico como moral, en la persecución de mi sueño de llegar a ser desarrollador de videojuegos, aguantando mis continuas ausencias debido a estar trabajando en mi sueño. Me han dado la confianza en mí mismo que me ha ayudado a llegar a donde estoy ahora y me ayudará a conseguir lo que me proponga en el futuro. También quiero dar gracias a mi 'yaya' Clementa, que ha sido y sigue siendo mi ángel de la guarda, apoyándome y dándome todo el amor que una persona puede dar. Me ha brindado los valores que me hacen ser la persona que soy hoy en día y a ella le dedico todos mis logros. Siguiendo con mi familia, quiero agradecer a mi tío Javi y a mi tía Isa el cariño, soporte y apoyo brindado durante toda mi vida, ayudándome de la mejor manera posible para que pueda alcanzar mis objetivos y ser feliz incluso en mis peores momentos. También a mi difunto abuelo Miguel, que, aunque no siempre estuvo presente, me apoyó y se alegró por mis logros en los últimos años que pude disfrutar de su presencia. Para concluir con la familia, no quiero dejar sin mencionar a mi tío político Rubén, que desde que lo conocí me ha aconsejado y ayudado en todo lo que ha podido, haciéndome más fácil el camino.

A continuación, a Paula, mi principal apoyo, quiero agradecerle por aguantar todas las noches y momentos en que me he ausentado, y por toda la ayuda que me ha brindado continuamente sin esperar nada a cambio. Te quiero.

A mi gran e insoportable amiga Alicia, por acompañarme y hacer muchísimo más divertido todo este viaje que ha sido completar este grado. Desde participar en jams con excursión a Colonia, hasta realizar unas prácticas fenomenales juntos y colaborar en este Trabajo Fin de Grado, gracias por darle mucho color a un viaje que pudo haber sido gris.

Continuando con mis colaboradores, quiero agradecer a mi gran descubrimiento de este último año de carrera, Manuel Alejandro, su gran trabajo y colaboración en este proyecto por pura vocación. También quiero agradecer a todos mis compañeros como Javier, Anatoli, Santy, Dani etc. haber compartido este camino conmigo. Trabajar con gente que ama lo que hace es maravilloso.

Por último, pero no menos importante, a André Cardoso, por motivarme con sus videos y mostrarme que una sola persona, si se lo propone, puede lograr grandes cosas, e inspirarme para realizar este Trabajo de Fin de Grado.

Resumen

Las mecánicas en los videojuegos constituyen el núcleo de su atractivo, por lo que es esencial conocer cómo diseñarlas y prototiparlas para destacar desarrollador en la industria del desarrollo de videojuegos.

Sin embargo, la industria del videojuego es muy hermética, lo que dificulta aprender cómo se han hecho las mecánicas en videojuegos profesionales.

El presente trabajo tiene como principal objetivo analizar y recrear mecánicas de videojuegos profesionales para conocer cómo funcionan y acercar este conocimiento al público general.

Para ello, se han analizado y recreado 7 mecánicas de videojuegos profesionales utilizando un proceso de ingeniería inversa, pasando por un proceso de análisis, diseño, implementación y pulido.

Una vez se tuvieron los prototipos de las mecánicas, se integraron modelos y arte acorde a la dirección artística de los videojuegos originales para conseguir un acabado más profesional y una mayor fidelidad al videojuego original.

Durante el desarrollo de este proyecto se han utilizado metodologías de desarrollo ágil, en concreto Scrumban, además de buenas prácticas de ingeniería de software como patrones de diseño y trazabilidad para tener bien localizadas las partes del código donde está implementado cada requisito funcional.

El código fuente del presente proyecto se puede encontrar en el siguiente enlace de GitHub y una demo jugable en su página de Itch.io.

Conceptos clave:

- Mecánica de juego
- Ingeniería inversa
- Interactividad
- Scrumban
- Trazabilidad
- Patrones de diseño

Abstract

In video games, mechanics are the core of their fun, making it crucial to understand how to design and prototype them to become a proficient developer in the video game industry.

However, the video game industry is very secretive, making it difficult to learn how mechanics are created in professional video games.

This project aims to analyze and recreate mechanics from professional video games to understand how they work and to bring this knowledge to the general public.

To achieve this, seven mechanics from professional video games were analyzed and recreated using a reverse engineering process, which included analysis, design, implementation, and refinement.

Once the prototypes of the mechanics were developed, models and art were integrated in accordance with the original video games' artistic direction to achieve a more professional finish and greater fidelity to the original games.

During the development of this project, agile development methodologies have been used, specifically Scrumban, in addition to good software engineering practices such as design patterns and traceability to clearly locate the parts of the code where each functional requirement is implemented.

The source code for this project can be found at the following GitHub link, and a playable demo is available on its Itch.io page.

Key concepts:

- Game mechanic
- Reverse engineering
- Interactivity
- Scrumban
- Traceability
- Design patterns

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En la actualidad, los videojuegos han trascendido su papel original de simples herramientas de entretenimiento para convertirse en una referencia cultural. Esto ha incrementado el reconocimiento y valor otorgado a los diseñadores y desarrolladores de videojuegos.

Uno de los pilares principales de los videojuegos son las mecánicas de juego, las cuales son el núcleo de la interactividad y la diversión. Este Trabajo de Fin de Grado (TFG) se centra en la recreación de siete mecánicas fundamentales de videojuegos, con el objetivo de desvelar los principios y técnicas que subyacen en su desarrollo. A través de este proyecto, se busca acercar al público general al conocimiento técnico y creativo involucrado en la creación de estos sistemas interactivos.

1.1. Motivación

La industria del videojuego es una de las más grandes y lucrativas del mundo, caracterizada por su constante innovación y creatividad. Sin embargo, esta misma innovación conlleva una hermeticidad significativa, donde los estudios rara vez revelan los entresijos de cómo se desarrollan ciertas mecánicas de juego. Esta falta de transparencia presenta un desafío considerable para aquellos que llevan a cabo su incursión en la industria, dificultando su capacidad para comprender y aprender cómo funcionan realmente estas mecánicas o sistemas.

El presente proyecto se propone abordar esta brecha de conocimiento mediante un enfoque de ingeniería inversa para analizar y replicar diversas mecánicas de videojuegos. El objetivo es obtener resultados que se asemejen lo más posible a las mecánicas originales, proporcionando así una comprensión profunda de cómo se desarrollan y ejecutan estas mecánicas en entornos profesionales. Además, este proceso no solo busca generar conocimiento, sino también servir como una herramienta efectiva de aprendizaje para aquellos que desean iniciarse en el campo del desarrollo de videojuegos.

La motivación personal del autor para este trabajo de fin de grado surge de sus propias experiencias en los primeros años de su carrera universitaria. Enfrentando dificultades y desafíos que estuvieron a punto de llevarlo al abandono, el autor encontró inspiración

en recursos educativos accesibles, como los videos de André Cardoso. Estos recursos le demostraron que replicar mecánicas de videojuegos profesionales no solo es posible, sino también alcanzable para aquellos sin una vasta experiencia o grandes equipos. Este proyecto, por lo tanto, busca no solo transmitir conocimientos técnicos, sino también resaltar la importancia de hacer accesible el desarrollo de videojuegos a una amplia gama de personas.

1.2. Objetivos

Este trabajo tiene como objetivo general analizar y replicar diferentes mecánicas de videojuegos profesionales. Este objetivo pretende hacer accesible el conocimiento sobre cómo funcionan estas mecánicas al público en general. También utilizar la ingeniería inversa como método de aprendizaje. Este objetivo general se puede desglosar en los siguientes objetivos específicos:

- **Objetivo 1**: Analizar distintas mecánicas estudiando sus diferentes componentes para extraer cómo están diseñadas.
- **Objetivo 2**: Implementar las mecánicas de manera fiel a las originales, pero con un toque distintivo propio.
- **Objetivo 3**: Pasar a las mecánicas implementadas por un proceso de pulido para lograr un acabado más profesional.
- **Objetivo 4**: Desarrollar las siguientes mecánicas:
  - Hacha de Leviathan (God of War)
  - Perspectiva forzada (Superliminal)
  - Fotografía mágica (Viewfinder)
  - Tiempo bala (Superhot)
  - Lanza de Draupnir (God of War: Ragnarok)
  - *Speed Booster* y *Shinespark* (Metroid Dread)
  - Selector de personajes (Super Smash Bros. Ultimate.)
- **Objetivo 5**: Facilitar la comprensión del funcionamiento interno de estas mecánicas para el público en general.
- **Objetivo 6**: Adquirir conocimientos sobre el prototipado de mecánicas y seguir una metodología para alcanzar resultados óptimos.
- **Objetivo 7**: Desarrollo de las mecánicas con buenas prácticas de ingeniería de software como trazabilidad y uso de patrones de diseño.
- **Objetivo 8**: Establecer una metodología para el aprendizaje con este tipo de técnica utilizando ingeniería inversa y definiendo las diferentes etapas por las que pasa este proceso de análisis y recreación de mecánicas.

1.3. Estado del arte

Las mecánicas son uno de los elementos más relevantes y diferenciadores de los videojuegos, es de suma importancia para crecer como desarrollador en la industria del videojuego saber diseñarlas, prototiparlas y desarrollarlas correctamente. Esto refuerza aún más la importancia de compartir información sobre el desarrollo y funcionamiento de las mecánicas de juego a nivel profesional con el público en general.

Como se ha mencionado anteriormente, la industria del desarrollo de videojuegos es muy hermética, por lo que es difícil encontrar casos en los que estudios hayan compartido cómo se han diseñado las mecánicas de sus juegos o alguien haya revelado los entresijos de estas. Sin embargo, tenemos varios casos en los que esto ha ocurrido.

1.3.1. Liberación de código fuente de videojuegos comerciales

El movimiento *open source* [1], o de código abierto, ha tenido un impacto significativo en el desarrollo software desde sus inicios en la década de 1980. Se basa en la premisa de que el código fuente de un programa de software debería estar disponible para todos los usuarios. Esto permite no solo el uso libre del software, sino también su estudio, modificación y redistribución. Este movimiento llevó al desarrollo de la Licencia Pública General de GNU (o GNU General Public License) (GPL) [2], una de las licencias más populares del software libre.

Sin embargo, en el sector de los videojuegos este movimiento no se ha extendido tanto como en otras industrias del software, y aún le falta mucho por avanzar en este campo. No obstante, ha habido varias ocasiones en las que el movimiento *open source* ha tenido presencia en la industria de los videojuegos, lo que ha permitido a la comunidad de desarrolladores y entusiastas estudiar, modificar y crear derivados de estos juegos, prolongando su vida útil y aumentando su accesibilidad. A continuación, se presentan algunos de estos casos en los que se ha producido el movimiento *open source* en la industria del videojuego.

El 23 de diciembre de 1997 marca un hito en la historia de los videojuegos: ***DOOM*** [3] se convierte en el primer juego comercial cuyo código fuente se libera al público [4]. Inicialmente, *DOOM* se lanzó bajo una licencia sin ánimo de lucro. Sin embargo, el 3 de octubre de 1999, se le otorgó permiso para volver a ser liberado bajo la *GNU General Public License* [2], y actualmente se encuentra bajo la versión 2 de esta misma licencia. La liberación del código fuente de *DOOM* no solo fue un acontecimiento sin precedentes, sino que también tuvo un impacto significativo en la comunidad de jugadores y desarrolladores. Permitió la creación de una amplia variedad de mods y adaptaciones para prácticamente cualquier plataforma. Esto llevó a *DOOM* a ejecutarse en lugares inimaginables, desde cajeros automáticos hasta bacterias intestinales [5], destacando su versatilidad y alcance en la cultura popular y tecnológica. Todo esto demuestra que la publicación del código abierto de ciertos juegos o sus mecánicas también puede beneficiar a los estudios y no solo al público general.

Otro videojuego comercial cuyo código fuente fue liberado junto al de su propio motor el 14 de febrero de 2013 fue ***Half-Life 1*** [6] por parte de Valve Corporation bajo una licencia

propia llamada *Half Life 1 SDK LICENSE* [7] bajo la que permitían la modificación del código y desarrollo de juegos modificados en el motor de *Half-Life 1* siempre que estos se lanzaran de manera gratuita en su plataforma de distribución digital de videojuegos *Steam*.

Noel Berry, uno de los desarrolladores del videojuego ***Celeste*** [8], compartió el código de movimiento de la versión de Pico-8 [9] del juego en su repositorio de GitHub [10]. Este repositorio ofrece una visión detallada del funcionamiento y la programación detrás del movimiento en el juego, el cual fue muy aplaudido a nivel de jugabilidad, lo que lo convierte en una valiosa fuente de aprendizaje en lo que a implementación se refiere.

1.3.2. Conferencias de desarrollo en las *Game Developer Conference* de San Francisco

En la *Game Developers Conference* de 2024 John Johanas, director del videojuego ***Hi-Fi Rush*** [11], impartió una conferencia [12] en la que explicó cómo se llevó a cabo el diseño e implementación de la mecánica principal de este videojuego relacionada con el ritmo de la música. Johanas abordó tanto el nivel conceptual de la idea, mostrando el prototipo inicial, como el proceso de desarrollo, incluyendo la cantidad de personas necesarias y el tiempo requerido. Además, explicó cómo se planteó la producción y diseño de los demás elementos del juego en torno a esta mecánica, ya que es fundamental en la propuesta de valor del videojuego. Este recurso aporta un valor muy diferente respecto a los otros recursos mencionados anteriormente, ya que no se proporciona código fuente, pero sí información sobre el análisis, planteamiento y diseño de una mecánica, y cómo llevarla a su máximo exponente haciéndola presente en todas las partes del videojuego, desde la interfaz de usuario hasta el movimiento de los elementos del mundo.

En la misma edición de la *Game Developer Conference*, Shiro Mouri y Takashi Tezuka, desarrolladores del videojuego de Nintendo ***Super Mario Bros. Wonder*** [13], realizaron una conferencia [14] donde explicaron las decisiones de diseño del videojuego y cómo las plantearon. También mostraron ideas y prototipos de mecánicas que no llegaron a incluirse en el juego final. Al igual que la conferencia de John Johanas mencionada anteriormente, aunque no proporciona código fuente, sí ofrece valiosa información sobre decisiones de diseño, cómo se llevaron a cabo y con qué intenciones.

1.3.3. Otras fuentes de aprendizaje

André Cardoso en su canal de Youtube 'Mix and Jam' [15] realizó una serie de videos orientados a replicar mecánicas de sus videojuegos favoritos para desarrollarse como 'gameplay programmer'. Además, publicó todos los proyectos desarrollados de código abierto en GitHub para la comunidad, fomentando así el aprendizaje de una manera atractiva y mediante el análisis del funcionamiento de mecánicas famosas.

El día 26 de julio de 2023 se hicieron públicas en Espacenet [16], varias patentes del videojuego de Nintendo ***The Legend of Zelda: Tears of the Kingdom*** [17], como la referente al movimiento dinámico del jugador en montado sobre varios objetos [18],

en estos registros explican en que consiste el funcionamiento de las distintas mecánicas registradas en las patentes aportando información visual y documentos de diseño. Sin embargo, se aporta la información justa para poder registrar la patente y no se aporta ningún tipo de explicación sobre la implementación o el proceso seguido para llevarla a cabo por lo que sigue siendo una fuente de aprendizaje limitada.

Todas estas fuentes, aunque son de gran utilidad, siguen siendo insuficientes o incompletas, ya que ninguna (excepto los videos de André Cardoso) concibe una mecánica, idea o implementación de principio a fin, o realiza un análisis con un proceso diferenciado. Esto refuerza aún más la necesidad de este proyecto.

1.4. Marco Teórico

Las **mecánicas de juego** [19] en los videojuegos constituyen el núcleo esencial de su experiencia, definiendo las diversas formas en que los jugadores pueden interactuar con el mundo virtual y moldear su progresión. Esta interacción, fundamental en el medio, se materializa a través de la característica distintiva del videojuego: la **interactividad**. Como define Daniel Blanco Aza en su Trabajo de Fin de Grado [20, p. 7, párr 3, línea 6-7] *"la interactividad es la característica que genera en el usuario la sensación de libertad, fundamental para el proceso de la inmersión del usuario"*. Esta es la principal importancia de la interactividad, ya que por sí sola, de forma aislada, la interactividad en los videojuegos difícilmente se diferencia de la de otros medios.

Relacionado con la interactividad, es crucial definir el concepto de **inmersión**, ya que desempeña un papel fundamental en los videojuegos y en la función de las mecánicas. Según el mismo autor, la inmersión en videojuegos es la capacidad de un videojuego para hacer creer al jugador que forma parte del mundo virtual que representa. Esto se sustenta en la suspensión voluntaria de la incredulidad [20].

De este modo, las mecánicas permiten al jugador sumergirse en el universo del juego, explorar sus posibilidades y alterar su curso mediante acciones específicas. El propósito fundamental es generar dinámicas de juego que cautiven, entretengan y mantengan el interés del jugador, incentivándolo a continuar explorando y disfrutando de la experiencia ofrecida por el videojuego.

El **Pensamiento Computacional** [21] es el proceso de pensamiento que permite formular o resolver problemas del mundo que nos rodea haciendo uso de habilidades y técnicas, como las secuencias e instrucciones ordenadas (algoritmos), para llegar a la solución. La metodología que se pretende usar para el presente proyecto cuenta con el valor añadido de entrenar o utilizar las destrezas y/o habilidades que se asocian con el Pensamiento Computacional tales como las siguientes:

- **Reconocimiento de patrones:** Analizando las mecánicas y los problemas a los que hay que enfrentarse para comprender el funcionamiento de estas, es posible identificar similitudes o características compartidas. Esto permite desarrollar estrategias que aborden dichos problemas comunes sin realizar más trabajo del necesario ni repetir tareas o soluciones previamente realizadas.

- **Descomposición:** Mediante el análisis de las diferentes mecánicas que se pretenden replicar, es necesario descomponerlas en pasos, tareas y problemas más pequeños para ser resueltos individualmente. Esto las hace más manejables y sencillas. Además, fomenta la capacidad de **deducción** ya que, a través de la observación, se debe deducir cómo se puede llevar a cabo cada parte del proceso.
- **Abstracción:** Mediante la identificación de **elementos clave** en un problema se puede llevar a cabo un **sistema de representación** de estos, permitiendo seleccionar la información relevante, filtrando la esencial e ignorando detalles no relacionados o sin importancia.

Estas competencias están muy presentes en la fase de análisis de la mecánica y en la fase de diseño de requisitos de cada prototipo, ya que es esencial identificar los elementos clave y descomponerlos para establecer cada requisito por separado y determinar lo que necesita.

Las mecánicas seleccionadas para este trabajo se han elegido siguiendo un criterio basado en la experiencia personal del autor, considerándolas de alto nivel y desafiantes, lo que les otorga un valor emocional significativo. Además, se han tenido en cuenta su reconocimiento y originalidad en la industria del videojuego, ya que rompen con los esquemas tradicionales y elevan el estándar del diseño. Por tanto, estas mecánicas son ideales para un análisis detallado y su replicación, ya que pueden proporcionar conocimientos valiosos y de gran relevancia para el autor y para los lectores del presente trabajo.

Capítulo 2

Análisis y diseño de las mecánicas

2.1. Hacha de Leviathan. God of War

Esta mecánica 1 pertenece al aclamado videojuego de Santa Mónica Studio [22] ***God Of War*** (2018). Durante el juego, el jugador puede lanzar el hacha en una dirección, la cual destruirá o dañará todo lo que encuentre en su camino. Una vez el hacha ha sido lanzada, el jugador tiene la posibilidad de atraerla hacia la mano del personaje nuevamente pulsando el botón correspondiente, creando un efecto similar al del martillo Mjolnir del dios nórdico Thor. Todo esto proporciona al jugador una sensación de poder y combate únicos.

Figura 1: Kratos apuntando con su hacha en *God of War: Ragnarok* (2022)

2.1.1. Requisitos del prototipo

Los requisitos funcionales y no funcionales para la implementación de esta mecánica son los siguientes:

Requisitos funcionales

- **RF1.01 - Movimiento del personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando con cámara en tercera persona por detrás del hombro del personaje.
- **RF1.02 - Animaciones de las acciones del personaje:** El personaje debe realizar animaciones acordes a las acciones que está realizando el jugador consiguiendo una mejor sensación y retroalimentación para el jugador.
- **RF1.03 - Sistema de apuntado:** El prototipo debe contar con un sistema permita al jugador apuntar con precisión a los objetivos.
  - **RF1.03a - Indicador visual de apuntado:** El sistema debe mostrar un indicador visual indique a qué objetivo está apuntando el jugador.
  - **RF1.03b - Corrección de dirección del proyectil:** El sistema debe corregir la dirección del proyectil basándose en el sistema de apuntado, asegurando que el proyectil se dirija al objetivo seleccionado, ya que en caso contrario habría un desfase al no ser lanzado desde la cámara.
- **RF1.04 - Lanzamiento del hacha:** El jugador debe ser capaz de lanzar el Hacha de Leviathan correctamente al pulsar el *input* correspondiente. El hacha debe moverse en la dirección lanzada y rotar sobre sí misma en esa dirección. No podrá ser lanzada de nuevo hasta que el jugador la recupere.
- **RF1.05 - Colisiones del hacha:** El hacha debe colisionar con los diferentes objetos de la escena reaccionando diferente en función del tipo de objeto con el que se ha colisionado.
- **RF1.06 - Atracción del hacha:** El prototipo debe contar con un sistema de atracción del hacha que permita al jugador atraer de nuevo el hacha hacia la mano del personaje cuando el jugador pulse el *input* correspondiente.
- **RF1.07 - Objetos destructibles:** El prototipo debe contar con objetos destructibles en la escena que se rompan y destrocen cuando el hacha colisiona con estos.

Requisitos no funcionales

- **RNF1.01 - Feedback al capturar el hacha con la mano:** El prototipo debe proporcionar feedback como un camera shake, animación y/o sonido cuando el personaje capture el hacha para reforzar la sensación de fuerza y captura del personaje.

- **RNF1.02 - Trail visual en hacha de Leviathan:** El Hacha de Leviathan debe contar con un *trail renderer* acorde con el estilo frío del entorno a modo de retroalimentación visual. Debe activarse únicamente cuando el arma está moviéndose en el aire para que el jugador pueda visualizar con más facilidad el recorrido del arma.
- **RNF1.03 - Atmósfera fría:** El escenario debe contar con elementos como la calidez de la luz y partículas de nieve o polvo que transmitan la sensación de estar en un entorno frío y al aire libre.

2.2. Perspectiva forzada. Superliminal

Una mecánica muy particular y original es la que se encuentra en el videojuego independiente ***Superliminal*** [23] (2019) la cual involucra el uso de **ilusiones ópticas** y técnicas de **perspectiva forzada** [24] para alterar el tamaño o escala de los objetos de manera que siempre mantienen el mismo tamaño aparente para el jugador mientras los está desplazando o utilizando, independientemente de la perspectiva o distancia dentro del escenario, es decir, un objeto manipulado por el jugador conservará su tamaño percibido según la visualización del jugador, sin obedecer necesariamente las leyes tradicionales de la perspectiva.

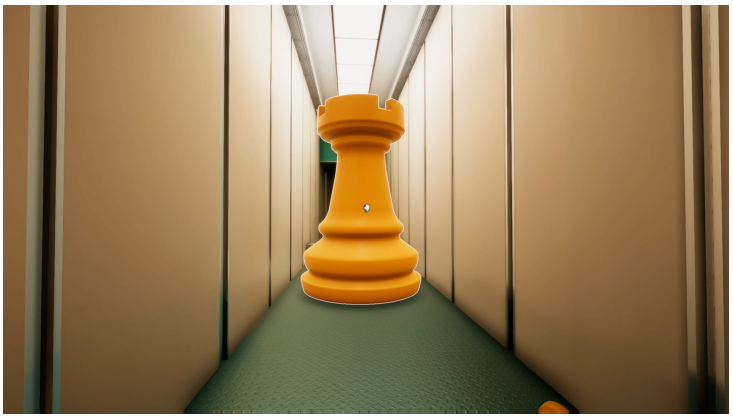

Figura 2: Pieza de ajedrez gigante. Superliminal (2019)

Por ejemplo, en cierto momento del juego, al jugador se le presenta el camino bloqueado por una pieza de ajedrez gigante, como se observa en la Figura 2. Cuando el jugador agarra esta pieza y gira la cámara hacia el suelo, su tamaño aparenta ser más pequeño en relación con la distancia al fondo, como se muestra en la Figura 3. Además, debido a que no se siguen las leyes tradicionales de la perspectiva, no solo parece ser más pequeño, sino que el tamaño también se modifica. Sin embargo, el jugador no percibe este cambio de tamaño hasta que suelta el objeto, logrando así el efecto tan sorprendente y característico del juego.

Figura 3: Pieza de ajedrez después de ser escalada. Superliminal (2019)

2.2.1. Requisitos del prototipo

Los requisitos funcionales y no funcionales para la implementación de esta mecánica son los siguientes:

Requisitos funcionales

- **RF2.01 - Movimiento del personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando relativo a la cámara.
- **RF2.02 - Cámara en primera persona:** El prototipo debe contar con una cámara en primera persona al ser un recurso vital para el correcto uso de la mecánica principal de la perspectiva.
- **RF2.03 - Sistema de apuntado:** El prototipo debe contar con un sistema de apuntado que permita al jugador visualizar el objeto con el que va a interactuar.
- **RF2.04 - Objetos interactuables:** El jugador debe poder interactuar con diferentes tipos de objetos y cada uno realizar la lógica correspondiente (agarrar, pulsar, soltar...) cuando el jugador interactúe con ellos.
- **RF2.05 - Deformación de la escala de objetos ignorando la perspectiva:** El prototipo debe contar con un sistema mediante el cual la escala de los objetos se deforme manteniendo el tamaño que el jugador ve que tiene ignorando las leyes de la perspectiva con el entorno.

Requisitos no funcionales

- **RNF2.01 - Feedback en objetos interactuables:** El jugador debe recibir información visual de cuando puede interactuar con un objeto o cuándo está agarrándolo realizando un cambio de cursor y un contorno alrededor del objeto.
- **RNF2.02 - Feedback adaptativo según el tamaño del objeto:** El prototipo debe contar con un sistema de feedback en los objetos que produzcan un feedback

diferente en función del tamaño del objeto. Sonidos más graves y contundentes y con un camera shake cuando se suelte un objeto muy grande o sonidos más suaves cuando se suelte un objeto pequeño, partículas en relación etc.

2.3. Fotografía mágica. Viewfinder

El videojuego independiente ***Viewfinder*** [25] (2023) introduce una mecánica innovadora que redefine la experiencia del jugador. En esta mecánica, los jugadores tienen la capacidad única de tomar fotografías del entorno (como se ve en la Figura 4) y luego instanciar la escena capturada en cualquier ubicación dentro del mundo del juego. Lo notable de esta mecánica es que solo se instancian las partes visibles de los objetos en la fotografía, lo que implica un procesamiento en tiempo real que incluye el corte de mallas, aumentando así la dificultad y el realismo de la experiencia de juego. Esta característica no solo ofrece un desafío adicional para los jugadores, sino que también mejora significativamente la inmersión en el mundo del juego, permitiendo una interacción más dinámica y personalizada con el entorno virtual.

Figura 4: Fotografía del entorno saliendo de la cámara. Viewfinder (2023)

2.3.1. Requisitos del prototipo

Los requisitos funcionales y no funcionales para la implementación de esta mecánica son los siguientes:

Requisitos funcionales

- **RF3.01 - Movimiento del personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando relativo a la cámara.
- **RF3.02 - Cámara en primera persona:** El prototipo debe contar con una cámara en primera persona al ser un recurso vital para el correcto uso de la mecánica principal de la colocación de objetos de la fotografía desde el punto de vista del jugador o la cámara.
- **RF3.03 - Sistema de apuntado:** El prototipo debe contar con un sistema de apuntado que permita al jugador visualizar objetos con los que puede interactuar.
- **RF3.04 - Objetos interactuables:** El jugador debe poder interactuar con diferentes tipos de objetos y cada uno realizar la lógica correspondiente (agarrar, pulsar, soltar...) cuando el jugador interactúe con ellos.
- **RF3.05 - Captura de los objetos en el frustrum de la cámara:** El prototipo debe contar con un sistema mediante el cual el jugador pueda realizar una fotografía y en esta se almacenen todos los objetos que captura la lente de la cámara.
- **RF3.06 - Instanciación de objetos presentes en fotografías:** El prototipo debe contar con un sistema que instancie los objetos almacenados en una fotografía cuando el jugador lo desee. Además, este cambio entre fotografía y objetos instanciados no debe ser perceptible hasta que el jugador se mueva y por tanto cambie su perspectiva.
- **RF3.07 - Corte de geometría en tiempo real:** Se deben almacenar solo las partes de los objetos que se ven en el *frustum* cámara y descartar las que no, lo cual conlleva un sistema de corte de mallas en tiempo real.

Requisitos no funcionales

- **RNF3.01 - Muestra de la foto capturada:** La foto capturada debe salir con una animación simulando la impresión de una foto en una cámara Polaroid con respectivo movimiento saliendo de la ranura de la cámara
- **RNF3.02 - Movimiento de los ítems entre posiciones:** Los ítems deben poder moverse por la pantalla de manera suave entre sus diferentes posiciones posibles para mejorar la sensación del prototipo.
- **RNF3.03 - Diferenciación entre objetos copiados y originales y representación en foto:** Los objetos del mundo que sean copiados por la cámara deben tener una distinción visual como un shader o color diferente para poder diferenciarse de los objetos originales y esta diferencia también debe mostrarse en la foto que imprime la cámara Polaroid.
- **RNF3.04 - Feedback al apuntar y realizar fotografía:** Al apuntar con la cámara en la interfaz de esta debe mostrarse una mirilla típica de cámaras digitales y al realizar una fotografía debe realizarse una animación que de retroalimentación al jugador de que se ha realizado la fotografía.

2.4. Tiempo bala. Superhot

De manera recurrente, en la industria del videojuego se dan casos de videojuegos que tienen gran éxito gracias a mecánicas muy creativas, cuyo atractivo radica en su simplicidad. Estas mecánicas son fáciles de entender desde el punto de vista del jugador y sencillas de prototipar o implementar desde el punto de vista del desarrollador.

Figura 5: Enfrentamiento con enemigo. Superhot (2016)

Un ejemplo de esto es el videojuego ***Superhot*** [26] (2016), cuya mecánica principal es el uso del "tiempo bala"[27], un concepto visto por primera vez en la película de 1999 titulada *Matrix* [28]. En este videojuego, el jugador se enfrenta a diferentes enemigos que intentan matarlo disparándole o yendo a por él. El tiempo solo transcurre a velocidad normal mientras el jugador se mueve; cuando el jugador está quieto, el tiempo pasa mucho más lento. Esto proporciona al jugador la capacidad de tomar decisiones con tiempo para meditar en un entorno hostil y dinámico. Además, su estética minimalista, en la que los enemigos y armas tienen colores mucho más llamativos o complementarios a los del escenario (ejemplo visible en la Figura 5), consigue que los jugadores identifiquen sus objetivos de manera instantánea, brindando así una jugabilidad única y una experiencia sobresaliente y poco común.

2.4.1. Requisitos del prototipo

Una vez analizada la mecánica y desglosados sus componentes clave se pueden definir los siguientes requisitos funcionales y no funcionales:

Requisitos funcionales

- **RF4.01 - Movimiento del personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando relativo a la cámara.
- **RF4.02 - Cámara en primera persona:** El prototipo debe contar con una cámara en primera persona al ser un recurso vital para un shooter en primera persona y lograr una mayor inmersión por parte del jugador.
- **RF4.03 - Mirilla dinámica:** El prototipo debe contar con un sistema de mirilla dinámica que permita al jugador visualizar el punto hacia el que va a disparar y el objeto con el que puede interactuar en cada momento.
- **RF4.04 - Manipulación del tiempo acorde a las acciones del jugador:** El prototipo debe contar con un sistema que modifique la velocidad del tiempo en función de las acciones del jugador como moverse, disparar, lanzar un objeto etc.
- **RF4.05 - Disparo del arma:** El jugador debe poder disparar proyectiles con el arma que lleva equipada cuando este lo desee presionando el botón o tecla correspondiente.
- **RF4.06 - Enemigos vulnerables:** El prototipo debe contar con enemigos que detecten cuando son alcanzados por un objeto letal (ya sea del jugador o de otro enemigo) o golpeados por el jugador y mueran en consecuencia.
- **RF4.07 - Sistema de cambio armas:** El jugador debe poder lanzar o soltar el arma que lleva equipada y equipar cualquier arma que se encuentre en el mundo o proveniente de otros enemigos.

Requisitos no funcionales

- **RNF4.01 - Feedback en armas o enemigos interactuables:** El jugador debe recibir información visual de cuándo puede interactuar con un arma realizando un cambio de cursor y feedback visual sobre el arma o enemigo.
- **RNF4.02 - Feedback visual en muerte de enemigos:** El prototipo debe contar con enemigos que al morir otorguen feedback visual sobre ello como animaciones de *ragdoll*, perdida o explosiones de partes del cuerpo etc.
- **RNF4.03 - Feedback visual en recarga de arma:** El prototipo debe contar con un sistema que haga girar al cursor, modificar su tamaño y reproducir un sonido cuando el arma equipada pueda ser disparada de nuevo.

2.5. Lanza de Draupnir. God of War: Ragnarök

En el videojuego ***God of War: Ragnarok*** (2022), secuela de God Of War (2018) se añadió una nueva arma para Kratos: la **Lanza de Draupnir**. Esta arma, un anillo que

invoca lanzas espartanas, tiene un gran significado en el personaje, ya que la lanza era la primera arma que aprendían a dominar los soldados espartanos en la antigua Esparta.

Figura 6: Kratos apuntando con su lanza en *God of War: Ragnarok* (2022)

La mecánica de la Lanza de Draupnir (Figura 6) consiste en el lanzamiento de lanzas que dañan todo lo que encuentran a su paso. Estas lanzas se quedan clavadas en el entorno y en los enemigos. Cada vez que una lanza es disparada, otra se invoca en la mano del personaje, permitiendo al jugador tener hasta cinco lanzas a la vez clavadas en el mundo (este número se puede ampliar posteriormente). Estas lanzas se marcan con indicadores en forma de punta de lanza debajo de la mirilla de apuntado, como se muestra en la Figura 7. Las puntas de color blanco indican el número de proyectiles (lanzas) presentes sin explotar, mientras que las de color oscuro muestran los proyectiles disponibles para ser arrojados antes de que se empiece a destruir el más antiguo. Al pulsar el botón correspondiente, el jugador puede hacer explotar todos los proyectiles presentes en el mundo, causando daño y destruyendo objetos en el radio de explosión de cada uno.

Figura 7: Indicador en interfaz de usuario sobre las lanzas disparadas sin explotar

2.5.1. Requisitos del prototipo

Una vez analizada la mecánica y desglosados sus componentes, podemos determinar que los requisitos funcionales y no funcionales para la implementación de esta mecánica son los siguientes:

Requisitos funcionales

- **RF5.01 - Movimiento del personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando con cámara en tercera persona por detrás del hombro del personaje.
- **RF5.02 - Animaciones de las acciones del personaje:** El personaje debe realizar animaciones acordes a las acciones que está realizando el jugador consiguiendo una mejor sensación y retroalimentación para el jugador.
- **RF5.03 - Sistema de apuntado:** El prototipo debe contar con un sistema que permita al jugador apuntar con precisión a los objetivos.
  - **RF5.03a - Indicador visual de apuntado:** El sistema debe mostrar un indicador visual que indique a qué objetivo está apuntando el jugador.
  - **RF5.03b - Corrección de dirección del proyectil:** El sistema debe corregir la dirección del proyectil basándose en el sistema de apuntado, asegurando que el proyectil se dirija al objetivo seleccionado, ya que en caso contrario habría un desfase al no ser lanzado desde la cámara.
- **RF5.04 - Lanzamiento del arma:** El jugador debe ser capaz de disparar la Lanza de Draupnir correctamente al pulsar el *input* correspondiente. La lanza debe moverse en la dirección lanzada y rotar en su eje z.
- **RF5.05 - Colisiones del arma:** El arma debe colisionar con los diferentes objetos de la escena reaccionando diferente en función del tipo de objeto con el que se ha colisionado y quedarse clavada en los objetos resistentes o enemigos.
- **RF5.06 - Aparición de nuevas lanzas:** El prototipo debe contar con un sistema de instanciación de una nueva lanza cuando el jugador dispara la que tiene actualmente, imitando el comportamiento del anillo que multiplica las lanzas.
- **RF5.07 - Objetos destructibles:** El prototipo debe contar con objetos destructibles en la escena que se rompan y destrocen cuando el hacha colisiona con estos.
- **RF5.08 - Explosión de lanzas:** El jugador debe poder explotar las lanzas disparadas que se encuentran clavadas en algún elemento del escenario al pulsar el *input* correspondiente. Reseteando así el contador de lanzas de la interfaz.
- **RF5.09 - Cambio de armas:** El jugador debe poder cambiar el arma equipada actualmente entre las disponibles al pulsar el *input* correspondiente.

Requisitos no funcionales

- **RNF5.01 - Feedback al explotar las lanzas:** El prototipo debe proporcionar feedback como un camera shake, animación y/o sonido, además de partículas tanto en las lanzas al explotar como en el golpe al suelo del personaje para reforzar la acción y la sensación de poder.
- **RNF5.02 - Trail visual en Lanza de Draupnir:** La Lanza de Draupnir debe contar con un trail de viento o similar acorde con el estilo del arma a modo de retroalimentación visual de su movimiento. Debe activarse únicamente cuando el arma está moviéndose en el aire para que el jugador pueda visualizar con más facilidad el recorrido del arma.
- **RNF5.03 - Atmósfera fría:** El escenario debe contar con elementos como la calidez de la luz y partículas de nieve o polvo que transmitan la sensación de estar en un entorno frío y al aire libre.
- **RNF5.04 - Movimiento de cámara:** La cámara debe alejarse levemente y ampliar su campo de visión cuando el jugador explote las lanzas golpeando el suelo para enfatizar la acción.

2.6. Speed Booster and Shinespark. Metroid Dread

El videojuego español ganador del premio GOTY a mejor videojuego de acción/aventura en 2021 [29], ***Metroid Dread*** (2021), desarrollado por MercurySteam Entertainment, cuenta con numerosas mecánicas destacables y divertidas que encajan perfectamente con el diseño de niveles y la progresión del juego, una característica fundamental del subgénero *metroidvania* [30]. Entre ellas se encuentran el "acelerón" y la "técnica cometa" (*Speed Booster* y *Shinespark* en inglés), que son los objetivos principales a replicar en este prototipo. Además, también se han implementado todas las mecánicas de un videojuego de plataformas básico, como el movimiento, deslizamiento, salto en pared, detección de pendientes etc. ya que son necesarios para el correcto funcionamiento de la mecánica principal que se quiere replicar y su correcta demostración.

Para entender correctamente el funcionamiento de esta mecánica, es importante dividirlo en dos partes: el *Speed Booster* y el *Shinespark*. Mientras el jugador se mueve, puede presionar un determinado *input* para comenzar a cargar el *Speed Booster*. Una vez ha transcurrido cierto tiempo, el jugador se desplaza a gran velocidad en la dirección en la que se movía. Durante este estado de alta velocidad, al realizar el *input* correspondiente (en consola, mover el joystick izquierdo hacia abajo), se activa la posibilidad de usar la habilidad de *Shinespark* por tiempo limitado. El *Speed Booster* se perderá si el jugador salta , se detiene o cae al vacío.

Cuando se utiliza la técnica *Shinespark* 8, pulsando el *input* correspondiente y orientando el joystick o tecla de movimiento en alguna dirección, el personaje se desplaza rodeado por un aura en forma de cometa de color vino hacia la dirección más cercana (entre las 8 posibles entre puntos cardinales y ordinales) según la orientación del *input* de movimiento del jugador. Durante este desplazamiento, el personaje destruye todos los enemigos y

Figura 8: Samus Aran realizando la técnica *Shinespark* en Metroid Dread (2021)

bloques a su paso indefinidamente hasta que choca con una pared, techo o suelo. Si al usar esta habilidad no se orienta el joystick en ninguna dirección, o se apunta hacia el suelo o la pared con la que estamos en contacto, la habilidad se cancela.

2.6.1. Requisitos del prototipo

Una vez analizada la mecánica y desglosados sus componentes, podemos determinar que los requisitos funcionales y no funcionales para la implementación de esta mecánica son los siguientes:

Requisitos funcionales

- **RF6.01 - Control de plataformas básico:** El prototipo debe contar con un sistema de control de movimiento con funcionalidades propias de un movimiento de plataformas.
    - **RF6.01a - Movimiento de personaje:** El prototipo debe contar con un sistema de movimiento por teclado o mando con cámara lateral.
    - **RF6.01b - Mecánica de salto:** El personaje debe saltar cuando el jugador pulse el *input* correspondiente.
    - **RF6.01c - Mecánica de deslizamiento:** El personaje debe deslizarse por el suelo cuando el jugador pulse el *input* correspondiente, reduciendo la altura de su área de colisión para poder pasar por zonas más bajas y estrechas y no terminando de deslizar hasta que se termine de detectar un objeto cercano por encima del jugador.

- **RF6.01d - Mecánica y estado de freno:** El prototipo debe contar con un sistema que, al cambiar la dirección del personaje mientras está en movimiento, disminuya su velocidad gradualmente y no permita su movimiento durante ese periodo. Todo esto debe ser manejado por un estado de freno en su máquina de estados.
- **RF6.01e - Mecánica de salto en pared:** El personaje debe saltar al lado opuesto de la pared con la que está en contacto cuando el jugador pulse el *input* correspondiente, manteniendo la dirección y animación del salto y rebotando perpendicularmente al vector de velocidad con el que se ha contactado la pared.
- **RF6.01f - Detección de superficie:** El prototipo debe contar con un sistema que detecte la superficie sobre la cual está en contacto el personaje y obtenga datos importantes, como el tipo de superficie para sonidos de pasos o el ángulo con la normal para calcular la inclinación de la pendiente en caso de estar sobre una.

- **RF6.02 - Animaciones de las acciones del personaje:** El personaje debe realizar animaciones acordes a las acciones que está realizando el jugador consiguiendo una mejor sensación y retroalimentación para el jugador.
- **RF6.03 - Sistema de control de *Speed Booster*:** El prototipo debe contar con un sistema que controle la lógica y variables del *Speed Booster*.
  - **RF6.03a - Carga de *Speed Booster*:** El prototipo debe contar con un sistema que comience a cargar el *Speed Booster* en función de un determinado tiempo mínimo cuando el jugador pulse el *input* correspondiente, aumentando la velocidad máxima actual del personaje cuando el *Speed Booster* esté cargado.
  - **RF6.03b - Control de estados de aceleración:** El prototipo debe contar con un sistema que controle los diferentes estados de la aceleración, tales como: apagado, cargando, cargado y activado *Shinespark*.
  - **RF6.03c - Tiempo de almacenamiento de *Shinespark*:** El prototipo debe contar con un sistema que controle el tiempo que puede estar el jugador con *Shinespark* cargado, y que lo descargue completamente cuando este tiempo se agote.
- **RF6.04 - Sistema de control de *Shinespark*:** El prototipo debe contar con un sistema que controle la lógica y variables de la técnica *Shinespark*.
  - **RF6.04a - Congelación de movimiento:** Cuando el jugador pulse el *input* correspondiente, el personaje debe congelar su movimiento por unos instantes, permitiendo al jugador elegir una dirección en la que dirigir al personaje durante la técnica *Shinespark*.
  - **RF6.04b - Cálculo de la dirección a la que moverse:** El prototipo debe contar con un sistema que calcule el punto cardinal u ordinal más próximo a la dirección en la que se ha recibido el *input* de movimiento del jugador y mover al personaje en dicha dirección calculada.
  - **RF6.04c - Detección de colisiones en *Shinespark*:** El prototipo debe contar con un sistema que controle las colisiones del personaje durante el movimiento de la técnica y actúe en consecuencia dependiendo del objeto con el

que se colisiona (destrozando bloques y enemigos, finalizando la técnica al chocar con una pared o superficie estática, o reactivando el *Speed Booster* cuando el ángulo de colisión es diagonal).

- **RF6.05 - Estados para cada situación:** El personaje debe tener diferentes estados en su máquina de estados, correspondientes a cada situación de control dentro del juego, para un mejor funcionamiento.
- **RF6.06 - Bloques destructibles:** El prototipo debe incluir bloques destructibles en la escena, que se rompan cuando el jugador colisione con ellos utilizando el *Shinespark* o *Speed Booster*.

Requisitos no funcionales

- **RNF6.01 - Efectos visuales en *Speed Booster*:** El prototipo debe reproducir efectos visuales cuando se está cargando o usando el *Speed Booster*.
  - **RNF6.01a - Efecto Fresnel:** El personaje debe tener un efecto Fresnel [31] en toda su malla, parpadeando entre dos colores a velocidades diferentes en función del estado de carga del *Speed Booster*. Si está cargando va de azul a transparente y si está ya cargado de azul a blanco.
  - **RF6.01b - Efecto de distorsión:** Cuando el personaje está corriendo, debe aparecer a su alrededor un efecto de distorsión del entorno con forma de semi esfera puntiaguda con partículas de aire a su alrededor, para dar la sensación estar moviéndose a gran velocidad.
  - **RF6.01c - Efecto de emisión circular:** Cuando el personaje está cargando el *Speed Booster*, debe aparecer un efecto en forma de círculo reduciéndose hacia el personaje para dar la sensación de cargar u obtener energía.
  - **RF6.01d - Efecto fuego propulsor:** El personaje debe tener fuego a modo de propulsor saliendo de su espalda para reforzar la sensación de impulso.
  - **RF6.01e - Efectos de propulsión fuerte:** Cuando se termine de cargar el *Speed Booster*, deben emitirse efectos simulando una explosión o una fuerte propulsión, como una explosión circular en la dirección contraria del movimiento simulando la ruptura de la barrera del sonido, emisión de humo por la propulsión y emisión de chispas eléctricas.
- **RNF6.02 - Efectos visuales en *Shinespark*:** El prototipo debe reproducir efectos visuales cuando se carga y se usa la energía de la técnica *Shinespark*.
  - **RNF6.02a - Efecto Fresnel:** El personaje debe tener un efecto Fresnel [31] en toda su malla de color vino o similar, estático, cuando la energía de la técnica es preparada para ser usada.
  - **RF6.02b - Emisiones almacenamiento de energía:** Cuando la energía de la técnica esté preparada para ser usada, deben emitirse partículas simulando chispas eléctricas del color del aura del cometa y una emisión circular hacia afuera del color del aura del *Speed Booster*, para simular la transición de energía del *Speed Booster* a la de la técnica *Shinespark*.

- **RF6.02c - Efecto de carga:** Cuando el personaje se paraliza en el aire para que el jugador elija la dirección de la técnica, deben aparecer chispas o rastros de energía desplazándose hacia arriba para simular la energía saliente a punto de explotar.
- **RF6.02d - Partículas del cometa:** Cuando se use la técnica, deben emitirse partículas como humo alrededor de una capsula semicircular puntiaguda alrededor del jugador en la dirección contraria al movimiento, y partículas de chispas eléctricas del color del aura del cometa.
- **RF6.02e - Remolino de emisión giratoria:** Cuando se use la técnica, debe aparecer un remolino de emisión giratoria que se cruza entre sí y se desplaza a lo largo del tiempo en la dirección contraria al movimiento.

- **RNF6.03 - Atmósfera espacial:** El escenario debe contar con elementos acordes a una estética de ciencia ficción en el espacio, transmitiendo el aura del videojuego original.
- **RNF6.04 - Estética de bloques:** La estética de los bloques debe transmitir que solo se pueden destruir mediante la técnica *Shinespark* o con *Speed Booster*, utilizando algún símbolo identificativo.
- **RNF6.05 - Movimiento de cámara:** La primera vez que se use la técnica *Shinespark*, la cámara debe colocarse en una posición diferente para un plano más espectacular.

2.7. Selector de personajes. Super Smash Bros. Ultimate

El aclamado videojuego de lucha ***Super Smash Bros. Ultimate*** [32] (2018) cuenta con un selector de personajes, mostrado en la Figura 9, en el que cada jugador tiene un cursor con forma de mano y una ficha con el color propio del jugador. Cada jugador puede coger su ficha y arrastrarla por las diferentes celdas del selector, cada una correspondiente a un personaje. A medida que el jugador mueve la ficha sobre estas celdas, la casilla que contiene la información del personaje seleccionado por el jugador cambia de manera muy suave, mostrando la imagen, el logotipo del videojuego o saga al que pertenece y el nombre del personaje actual tal y como se ve en la Figura 10. Además, las distintas celdas que contienen a los personajes adaptan su posición y tamaño en función del número de personajes disponibles en ese momento, ya que se van desbloqueando a medida que avanzas en el juego. Este selector proporciona una experiencia muy satisfactoria e introduce al jugador en una atmósfera de combate y epicidad simplemente mediante el uso del selector de personajes. Esto se consigue gracias a las transiciones suaves, la retroalimentación con cada acción (como vibración o sonido al seleccionar y cambiar de personaje) y los cambios visuales en el menú acorde a las acciones del jugador.

El funcionamiento de esta pantalla demuestra que incluso la selección de personajes puede ser divertida, al mismo tiempo que transmite de manera efectiva la atmósfera del juego

Figura 9: Selector de personajes con ningún personaje seleccionado

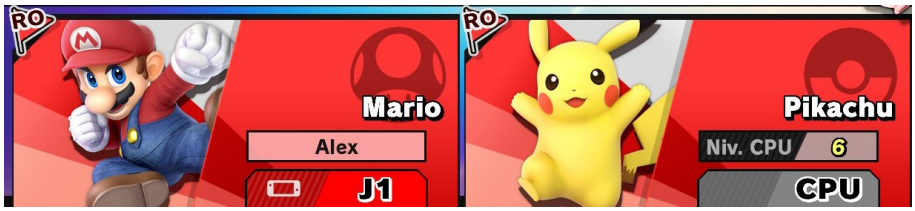

Figura 10: Casillas de información sobre el personaje seleccionado para cada jugador

y atrapa al jugador antes de que comience una partida. Por este motivo, esta mecánica o sistema ha sido seleccionada para ser recreada en el presente proyecto.

No obstante, para una mejor integración en el proyecto, esta recreación será utilizada como selector entre las distintas mecánicas recreadas, permitiendo al usuario elegir qué mecánica desea probar. Por lo tanto, contará con un mayor número de elementos diferentes o reinterpretados para este fin.

2.7.1. Requisitos del prototipo

Una vez analizado el funcionamiento de este sistema y desglosados sus componentes clave se pueden definir los siguientes requisitos funcionales y no funcionales:

Requisitos funcionales

- **RF7.01 - Celdas de personaje:** El prototipo debe incluir celdas que contengan a los diferentes personajes o elementos seleccionables, así como sus datos y la lógica

asociada.

- **RF7.02 - Cuadrícula dinámica:** El prototipo debe contar con una cuadrícula que contenga a las celdas de los personajes o elementos seleccionables, ajustándose dinámicamente en tamaño y disposición en función del número de elementos en la cuadrícula.
- **RF7.03 - Cursor y ficha para selección:** El prototipo debe incluir un cursor que permita al usuario desplazarse por la pantalla para seleccionar diferentes opciones. Además, el usuario debe poder recoger y soltar la ficha correspondiente al personaje elegido utilizando este cursor.
- **RF7.04 - Selección de personaje u elemento:** Cuando se suelte la ficha sobre la celda que contiene un personaje o elemento seleccionable, la información de este debe guardarse para su uso posterior.
- **RF7.05 - Confirmación de selección:** Una vez que el jugador haya seleccionado su personaje, debe aparecer un botón que permita avanzar y cargar la escena correspondiente.
- **RF7.06 - Cursor dinámico:** El cursor debe cambiar de estado según su interacción con la ficha, elementos interactuables y otras condiciones. Los estados incluyen el agarre de la ficha, cuando está sobre un elemento interactuable y cuando no se cumple ninguna de las condiciones anteriores.
- **RF7.07 - Ranura de jugador:** Cada jugador debe tener una ranura en la que se muestre información sobre el personaje detectado por la ficha del jugador, incluyendo imagen y logotipo del videojuego al que pertenece, con actualización en tiempo real. Además, cada ranura debe ser distintiva para cada jugador, asegurando así una clara identificación de la información mostrada.
- **RF7.08 - Vuelta al menú principal:** Esta pantalla de selección debe incluir un botón que permita volver atrás y cargar la última escena en la que se ha estado, en este caso, el menú principal.

Requisitos no funcionales

- **RNF7.01 - Feedback en celdas:** Cuando la ficha es agarrada por el jugador y se desplaza sobre las distintas celdas, estas deben ofrecer retroalimentación visual para que el usuario pueda identificar con certeza con qué celda está interactuando en ese momento.
- **RNF7.02 - Cursor y ficha identificativos:** Tanto el cursor como la ficha de selección deben ser identificativos con el número de jugador (J1, J2, J3, J4) y el color correspondiente (rojo, azul, amarillo, verde) para garantizar un mejor reconocimiento por parte del usuario.
- **RNF7.03 - Ranuras de jugador identificativas:** Cada ranura de selección de jugador debe ser identificativa, mostrando el número de jugador (J1, J2, J3, J4) y teniendo el color correspondiente a ese jugador (rojo, azul, amarillo, verde), para garantizar un reconocimiento claro por parte del usuario.

- **RNF7.04 - Transiciones suaves en ranuras de jugador:** Cuando se cambie el personaje sobre el que se encuentra la ficha del jugador, el personaje mostrado en la ranura de este debe desplazarse suavemente hacia la izquierda, mientras que el nuevo personaje seleccionado entra por la derecha. Consiguiendo así transiciones más agradables para el usuario.
- **RNF7.05 - Feedback en selección de elemento:** Cuando se seleccione un elemento o personaje soltando la ficha, se debe proporcionar un feedback visual en la ranura del jugador para reforzar el efecto de selección y mejorar la retroalimentación hacia el jugador.
- **RNF7.06 - Cambio visual de cursor en función de estado:** El cursor debe cambiar su elemento visual según el estado en el que se encuentre, proporcionando así una mayor retroalimentación al jugador sobre lo que puede o no puede hacer en cada momento.

Capítulo 3

Metodología utilizada

3.1. Herramientas utilizadas

3.1.1. Unity

Para llevar a cabo la recreación de las mecánicas propuestas se ha utilizado el motor de desarrollo de videojuegos ***Unity*** [33]. Lanzado por primera vez en junio de 2005 durante la Conferencia Mundial de Desarrolladores de Apple como motor de juegos para MAC OS X, actualmente está disponible para Windows, Mac y Linux.

Algunos de los motivos para la elección de este motor son los siguientes:

- **Comunidad**: Es uno de los motores más utilizados de la industria del videojuego, por lo que cuenta con una amplia comunidad que dispone de un gran número de tutoriales, recursos de código abierto, *assets* en su tienda propia y numerosos foros con respuestas a problemas comunes, los cuales se encuentran con gran facilidad.
- **Usabilidad**: Al ser un motor de propósito general, Unity destina gran cantidad de recursos a la usabilidad y accesibilidad del motor para facilitar su uso por parte de desarrolladores, lo cual agiliza el flujo de trabajo y el desarrollo de entornos interactivos en este motor.
- **Familiaridad**: El autor del proyecto tiene una amplia experiencia utilizando Unity durante años, lo que facilita el proceso de desarrollo. Esta experiencia permite aprovechar al máximo las capacidades del motor, asegurando una implementación eficiente y efectiva de las mecánicas del juego.
- **Documentación**: Unity cuenta con una extensa documentación que está estructurada de manera que es muy intuitiva para el usuario y que explica de forma bastante clara el funcionamiento de sus distintas clases, componentes, paquetes etc. que además se va a actualizando con regularidad.
- **Facilidad de exportación**: Unity permite a los desarrolladores cambiar la plataforma de destino para la compilación del proyecto con un solo clic. Esto facilita la

exportación de ejecutables en varias plataformas como Windows o WebGL, para las cuales el autor desea que el proyecto esté disponible.

3.1.2. Rider

Para la programación del código del proyecto se ha utilizado el IDE ***JetBrains Rider*** [34], que es de pago, pero cuenta con una licencia gratuita para estudiantes, la cual se ha aprovechado para el desarrollo de este proyecto. Es uno de los principales IDE multiplataforma para desarrolladores de juegos. El motivo principal para la elección de este IDE es su gran integración con *Unity* y las herramientas que proporciona para trabajar con este motor. Permite crear scripts de diferentes tipos del motor (ScriptableObject, MonoBehaviour) o clases comunes en estos entornos de desarrollo (interface, enum, struct) directamente desde el editor de código, sin tener que crear clases vacías y escribir la clase de la que hereda manualmente. Además, destaca por su rapidez y potencia.

3.1.3. C#

Para la implementación del código del proyecto se ha utilizado ***C#***, ya que es el lenguaje de programación con el que mejor está integrado Unity.

3.1.4. GitHub

Con el objetivo de mantener la integridad del código y los recursos del proyecto, se ha utilizado el control de versiones ***Git*** [35] a través de un repositorio en ***GitHub*** y su aplicación de escritorio, ***GitHub Desktop***.

3.1.5. DOTween

DOTween [36] es un motor de animación orientado a objetos para Unity, optimizado para usuarios de C#, gratuito y de código abierto, con una amplia gama de funciones avanzadas. Se ha utilizado debido a que resulta de gran ayuda a la hora de realizar tareas como mejorar la sensación general del juego con movimientos fluidos, interpolaciones y modificaciones de numerosas propiedades utilizando curvas de animación. Además, permite la creación de secuencias de acciones y mucho más.

3.1.6. Photoshop

Ante la necesidad de elementos gráficos 2D, principalmente para la interfaz de usuario y *Head-Up Display* (HUD) de las diferentes mecánicas, se ha utilizado el creador de gráficos y editor de fotografías ***Photoshop*** [37]. Este programa fue elegido por la facilidad que ofrece para crear los recursos necesarios y por la familiaridad del autor con su uso.

3.1.7. Trello

Para el manejo de la producción del proyecto y una mejor organización personal con metodologías ágiles, se ha utilizado el software de administración de proyectos Trello [38]. Este software ha facilitado una mejor organización de las tareas y la división del proyecto en sprints, agilizando el proceso y permitiendo conocer en cada momento qué tareas quedaban por hacer.

3.2. Método de desarrollo

Para llevar a cabo el desarrollo de las diferentes mecánicas se ha llevado a cabo un proceso propio que se ha repetido en cada mecánica desarrollada. Este método consta de tres partes: Análisis y Diseño, Implementación y Pulido.

- **Análisis y diseño**: En primer lugar, mediante la observación de videos extraídos por el autor directamente de los videojuegos originales, se realiza una fase de análisis del funcionamiento de las mecánicas, desglosándolas en problemas más pequeños y extrayendo a partir de estos los requisitos funcionales y no funcionales de cada mecánica. Esto permite una mejor organización de las tareas y un mayor entendimiento del funcionamiento de cada componente por separado. Además, gracias a este enfoque, se refuerzan competencias del Pensamiento Computacional como la descomposición de problemas.
- **Implementación**: Una vez definidos los requisitos funcionales y no funcionales, se procede a la implementación de cada uno de los requisitos para lograr el funcionamiento básico de la mecánica.
- **Pulido**: En esta fase se desarrollan los requisitos no funcionales que no se hayan completado en la fase de implementación, se corrigen pequeños errores y se mejora la retroalimentación de las acciones del personaje al jugador para obtener un acabado más profesional y cercano al de las mecánicas originales.

Este método permite desglosar la gran cantidad de trabajo que conlleva desarrollar cada una de estas mecánicas en problemas más pequeños, abordándolos con un enfoque más específico y concentrado. Esto facilita que el desarrollador se centre en las pequeñas partes que, en conjunto, logran el resultado final, evitando abrumarse al contemplar constantemente el problema en su totalidad.

3.3. Metodologías ágiles

Para la organización de las tareas y el control del proceso de realización de estas se han utilizado metodologías ágiles, en concreto una combinación de Scrum y Kanban, conocida como Scrumban [39]. Se ha escogido esta metodología por los siguientes motivos:

- **Uso de *sprints* con entrega continua**: Al combinar el uso de *sprints* propio de Scrum con el flujo de trabajo continuo de Kanban, permite que se puedan entregar funciones a medida que se completan sin tener que esperar a que termine el sprint.
- **Flexibilidad**: Scrumban ofrece un trabajo incremental con cada sprint, permitiendo cambios en el proyecto incluso a mitad del proceso, mientras se avanza hacia la finalización del mismo.
- **Adaptación al proyecto**: Al ser desarrollado principalmente por una persona, algunas de las desventajas de Scrumban como el menor control de la producción debido a la ausencia de roles definidos, desaparecen, manteniendo todas sus ventajas.
- **Visibilidad de tareas**: Con el tablero de Kanban y las tarjetas relativas a las tareas del proyecto, es muy fácil y rápido visualizar las tareas pendientes, las completadas, el transcurso del sprint, etc.
- **Familiaridad**: El autor del proyecto ha trabajado previamente con metodologías ágiles como Scrum y Kanban, por lo que está habituado a trabajar con este flujo de trabajo.

En las Figuras 11 y 12 se pueden observar las tablas de Kanban relativas al inicio de los sprint 7 y 8 respectivamente. Cada *sprint* tenía una duración media de 10 días, pero al usar Scrumban, si se terminaban todas las tareas en menos tiempo se podía pasar al siguiente sprint.

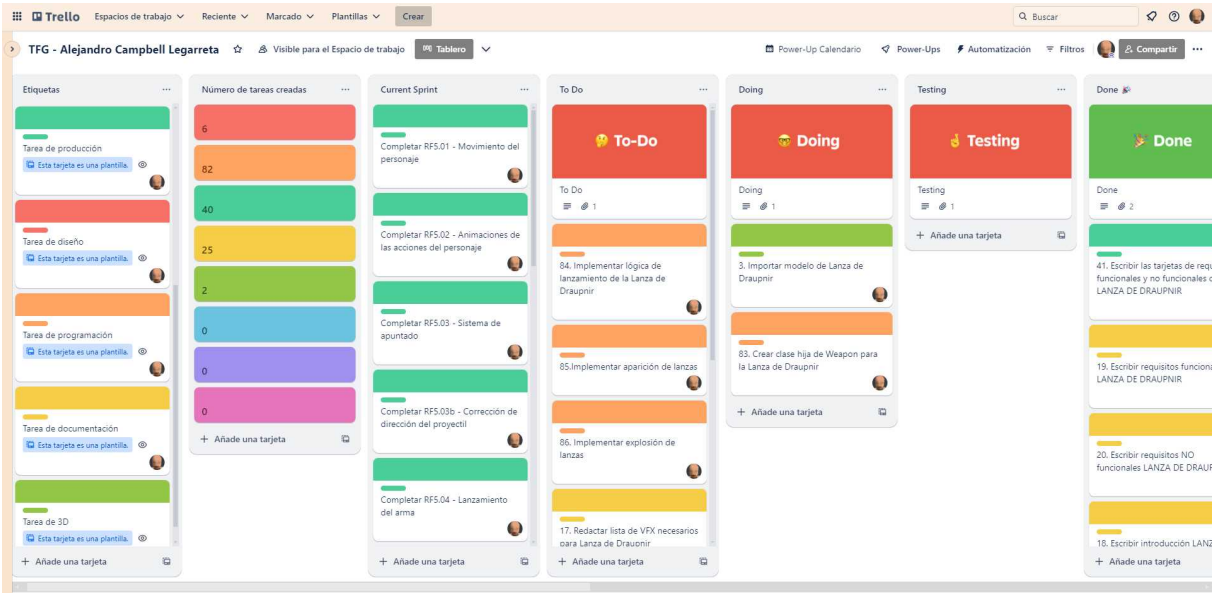

Figura 11: Tablas de Kanban relativas al Sprint 7 en Trello

3.3.1. Sprints realizados en el proyecto

En la tabla 1 se pueden observar los distintos *sprints* realizados a lo largo del proyecto con las tareas y requisitos completados en cada *sprint*.

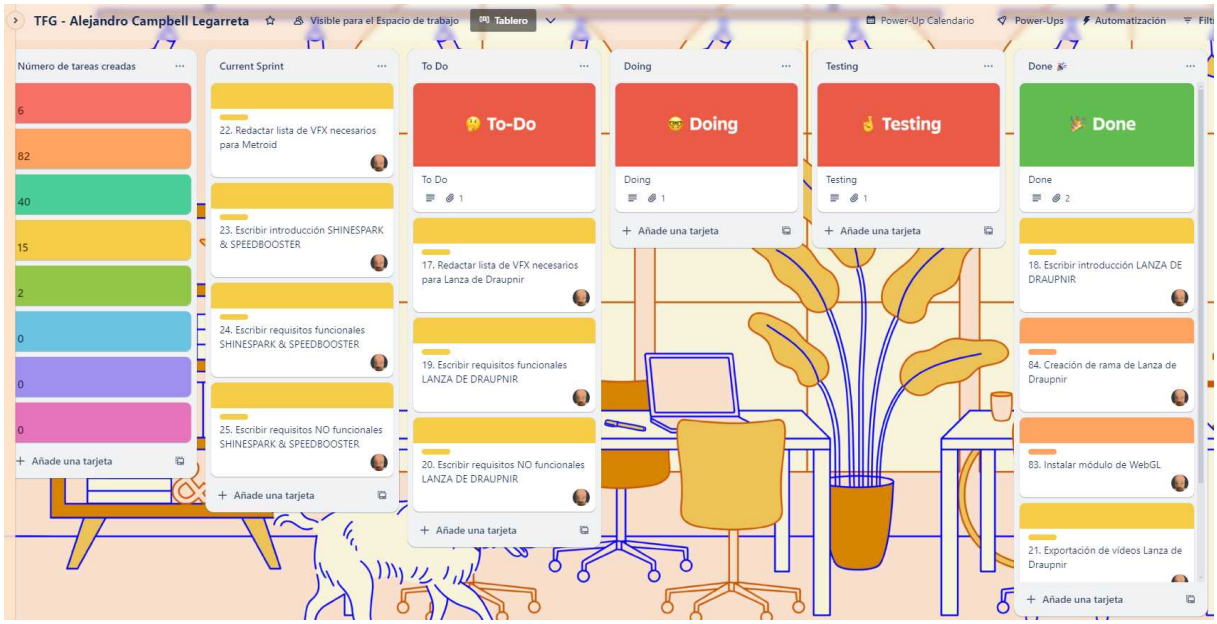

Figura 12: Tablas de Kanban relativas al Sprint 8 en Trello

Sprint	**Requisitos/Tareas afrontadas**
Sprint 1	Creación del proyecto en motor Programación de estructura general de entidades Programación de códigos comunes a todas las mecánicas Instalación de paquetes necesarios Creación de repositorio de GitHub
Sprint 2	RF1.01 - Movimiento del personaje RF1.02 - Animaciones de las acciones del personaje RF1.03 - Sistema de apuntado RF1.04 - Lanzamiento del hacha RF1.05 - Colisiones del hacha RF1.06 - Atracción del hacha RF1.07 - Objetos destructibles RNF1.01 - Feedback al capturar el hacha con la mano
Sprint 3	RF2.01 - Movimiento del personaje RF2.02 - Cámara en primera persona RF2.03 - Sistema de apuntado/detección de objetos RF2.04 - Objetos interactuables RF2.05 - Deformación de la escala de objetos ignorando la perspectiva RNF2.01 - Feedback en objetos interactuables
Sprint 4	RF3.01 - Movimiento del personaje RF3.02 - Cámara en primera persona RF3.03 - Sistema de apuntado RF3.04 - Objetos interactuables RF3.05 - Captura de los objetos en el frustrum de la cámara RF3.06 - Instanciación de objetos presentes en fotografías RF3.07 - Corte de geometría en tiempo real RNF3.01 - Muestra de la foto capturada RNF3.02 - Movimiento de los ítems entre posiciones

Sprint	**Requisitos/Tareas afrontadas**
Sprint 5	RF4.01 - Movimiento del personaje RF4.02 - Cámara en primera persona RF4.03 - Mirilla dinámica RF4.04 - Manipulación del tiempo acorde a las acciones del jugador RF4.05 - Disparo del arma RF4.06 - Enemigos vulnerables RF4.07 - Sistema de cambio armas RNF4.03 - Feedback visual en recarga de arma
Sprint 6	RF7.01 - Celdas de personaje RF7.02 - Cuadrícula dinámica RF7.03 - Cursor y ficha para selección RF7.04 - Selección de personaje u elemento RF7.05 - Confirmación de selección RF7.06 - Cursor dinámico RF7.07 - Ranura de jugador RF7.08 - Vuelta al menú principal RNF7.01 - Feedback en celdas RNF7.02 - Cursor y ficha identificativos RNF7.03 - Ranuras de jugador identificativas RNF7.04 - Transiciones suaves en ranuras de jugador RNF7.05 - Feedback en selección de elemento
Sprint 7	RF5.03 - Sistema de apuntado RF5.04 - Lanzamiento del arma RF5.05 - Colisiones del arma RF5.06 - Aparición de nuevas lanzas RF5.07 - Objetos destructibles RF5.08 - Explosión de lanzas RF5.09 - Cambio de armas

Sprint	**Requisitos/Tareas afrontadas**
Sprint 8	RF6.01 - Control de plataformas básico RF6.02 - Animaciones de las acciones del personaje RF6.03 - Sistema de control de *Speed Booster* RF6.04 - Sistema de control de *Shinespark* RF6.05 - Estados para cada situación
Sprint 9	RNF1.02 - Trail visual en hacha de Leviathan RNF1.03 y RF5.03 - Atmósfera fría RNF3.04 - Feedback al apuntar y realizar fotografía RNF4.01 - Feedback en armas o enemigos interactuables RNF4.02 - Feedback visual en muerte de enemigos RNF5.01 - Feedback al explotar las lanzas RNF5.02 - Trail visual en Lanza de Draupnir RNF6.01 - Efectos visuales en *Speed Booster* RNF6.02 - Efectos visuales en *Shinespark* RNF6.03 - Atmósfera espacial RNF6.04 - Estética de bloques

Tabla 1: Sprints realizados en el proyecto.

Capítulo 4

Implementación de las mecánicas

Antes de explicar los elementos clave e implementación específica de cada una de las mecánicas, es importante mencionar los sistemas implementados que son comunes a todas ellas o a gran parte de ellas. Estos sistemas son los siguientes:

- **Sistema de movimiento relativo a la cámara:** Para todas las mecánicas en las que el jugador puede moverse libremente por la escena de juego, se ha implementado un sistema de movimiento relativo a la cámara. Esto permite que el sistema de movimiento se utilice tanto para juegos en primera persona como para juegos en tercera persona, ya que solo es necesario modificar la posición de la cámara respecto al personaje.
- **Acciones basadas en datos:** Se ha utilizado una clase que almacena los datos con variables estáticas relativas a parámetros como velocidad, aceleración, fuerza de salto, etc. Esto permite que se pueda utilizar una misma clase con diferentes Scriptable Objects, cada una con sus propias estadísticas. El código común de movimiento sabe a qué variables acceder y, al llamarse igual en todos los personajes que se pueden controlar, se facilita la modificación del comportamiento de cada personaje por parte de los diseñadores.
- **Sistema de pausa y cambio de escenas:** El controlador presente en todos los niveles escucha al evento del *input* de jugador correspondiente a la pausa y, cuando es lanzado, pausa el juego. El sistema de cambio de niveles también es común a todas las mecánicas, al estar controlado por el controlador de juego y el controlador de niveles.

4.1. Hacha de Leviathan. God of War

4.1.1. Elementos clave

Entre otras características y retos afrontados en la presente mecánica 13, el más relevante ha sido el cálculo del movimiento y rotación en la trayectoria de regreso del hacha.

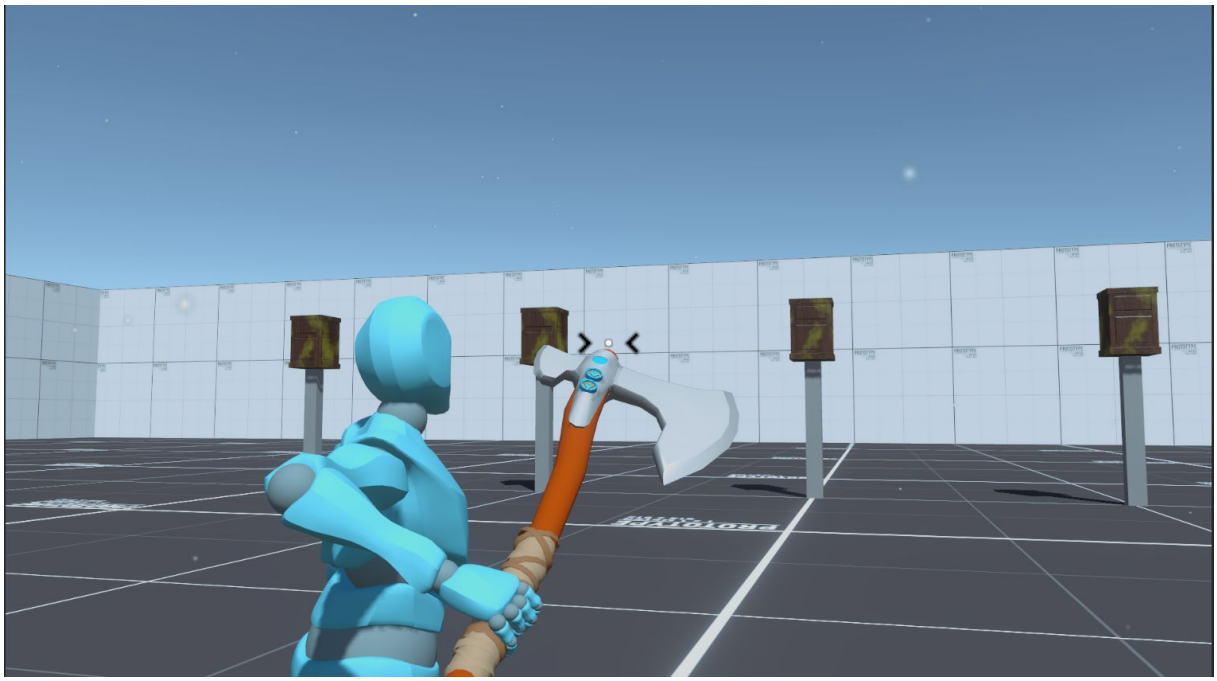

Figura 13: Recreación propia de la mecánica del Hacha de Leviathan

Para la implementación de esta, se ha utilizado una curva de Bezier cuadrática [40] para trazar la trayectoria realizando un arco. Esto se debe a que tenemos un punto inicial P_0 (la posición del hacha antes de volver) y un punto final P_2 (la posición de la mano del personaje). Para evitar una trayectoria recta, que resultaría poco estética, necesitamos un punto adicional P_1, que corresponde al punto intermedio al que se intenta aproximar la trayectoria para definir la curva.

La ecuación utilizada 1 define una curva cuadrática de Bézier con puntos de control P_0, P_1 y P_2.

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{1}$$

Consiguiendo con esto un efecto mucho más estético, parecido al de la mecánica original y agradable para el jugador.

4.1.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 2 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica del Hacha de Leviathan.

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF1.01 - Movimiento del personaje	Entity.cs EntityController.cs WalkPlayerState.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF1.02 - Animaciones de las acciones del personaje	ArmedPlayerAnimator.cs	Observer
RF1.03 - Sistema de apuntado	ArmedPlayer.cs PlayerInputManager.cs PlayerEvents.cs PlayerAimController.cs AimPlayerState.cs	Observer State Flyweight
RF1.04 - Lanzamiento del hacha	ArmedPlayer.cs PlayerInputManager.cs PlayerStatsManager AxeWeapon.cs	Observer Flyweight
RF1.05 - Colisiones del hacha	AWeapon.cs	DirtyFlag
RF1.06 - Atracción del hacha	Player.cs PlayerInputManager.cs AxeWeapon.cs	Observer State DirtyFlag
RF1.07 - Objetos destructibles	IBreakable.cs BreakableBox.cs	Prototype

Tabla 2: Trazabilidad de la mecánica Hacha de Leviathan.

4.2. Perspectiva forzada. Superliminal

4.2.1. Elementos clave

En esta particular mecánica 14, entre los diferentes retos afrontados, el más relevante ha sido conseguir escalar los objetos rompiendo las reglas tradicionales de la perspectiva, haciendo que el jugador no note el cambio de tamaño de los objetos hasta que los suelta.

En primer lugar, para conseguir este efecto, cuando el jugador agarra un objeto, este se desplaza continuamente en la dirección en la que mira el jugador hasta que colisiona con una superficie, momento en el que deja de moverse. Con esta acción, el objeto se verá más pequeño desde la perspectiva del jugador. Aquí es donde entra en juego nuestro principal

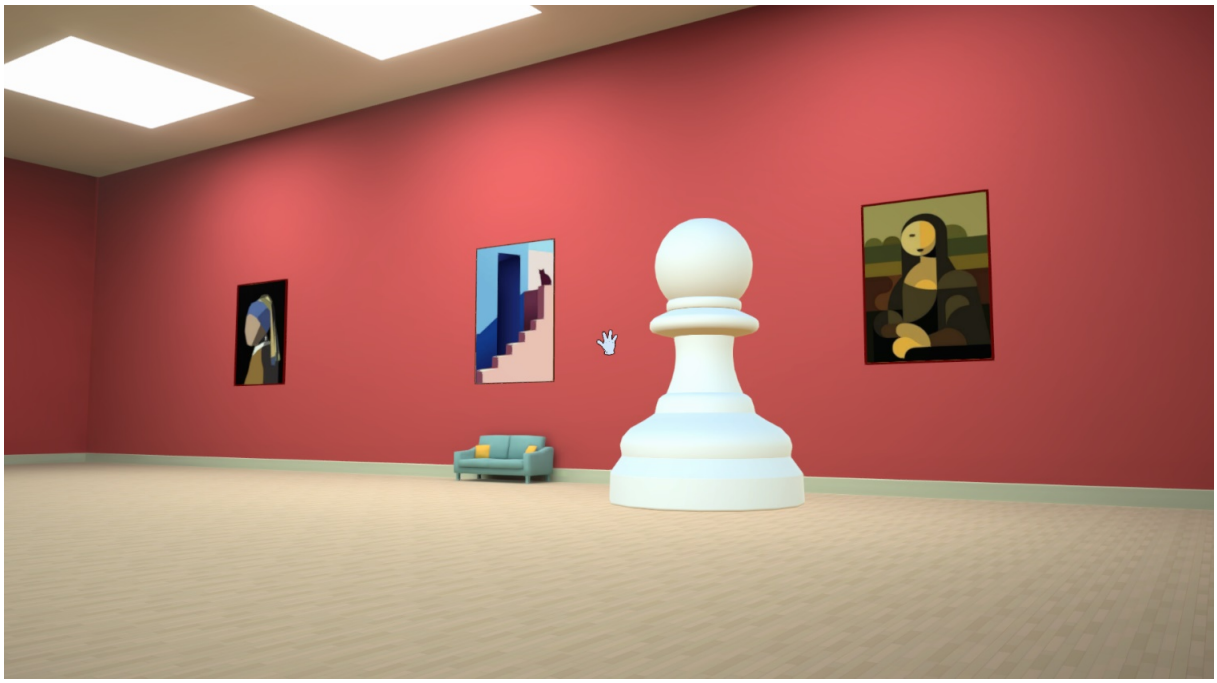

Figura 14: Recreación propia de la mecánica de perspectiva forzada

elemento clave: forzar la perspectiva compensando la diferencia de tamaño modificando su escala en función de la distancia a la que se encuentra.

Para la implementación de esta técnica, se ha utilizado la ecuación 2. En ella, se divide la distancia actual desde la posición del objeto hasta la posición del jugador D_c entre la distancia original cuando el jugador agarró el objeto D_o. Este resultado se multiplica por la escala original del objeto cuando es agarrado por el jugador S_o, y así obtenemos la nueva escala del objeto para mantener el tamaño desde el punto de vista del jugador. Consiguiendo este efecto tan impactante para el público.

$$S_n = S_o * \frac{D_c}{D_o} \tag{2}$$

4.2.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 3 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica de perspectiva forzada.

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF2.01 - Movimiento del personaje	Entity.cs EntityController.cs WalkPlayerState.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF2.02 - Cámara en primera persona		Component
RF2.03 - Sistema de apuntado	FirstPersonPlayer.cs AimSightChanger.cs	Observer Singleton
RF2.04 - Objetos interactuables	APickable.cs IInteractable.cs	Observer Prototype
RF2.05 - Deformación de la escala de objetos ignorando la perspectiva	SuperliminalPlayer.cs	DirtyFlag

Tabla 3: Trazabilidad de la mecánica Perspectiva forzada.

4.3. Fotografía mágica. Viewfinder

4.3.1. Elementos clave

Al afrontar los diferentes retos que supone recrear esta mecánica 15, es importante destacar dos de ellos: la detección de los objetos que se están viendo en la cámara Polaroid en el momento en que se realiza la foto y la eliminación de la geometría parcial de objetos que se ven parcialmente pero no de manera completa.

Para detectar los objetos que están en la vista de la cámara Polaroid cuando se toma la foto, se ha utilizado el campo de visión (*frustum*) [41] de dicha cámara, no el de la cámara del jugador, para instanciar un plano por cada uno de los límites del *frustum* de la cámara que capturará las fotografías. Además de los planos visibles en la Figura 16 también se crean los contrarios, es decir, los relativos a *left*, *bottom* y *far* (izquierda, inferior y lejano). Una vez definidos estos planos, al tomar la fotografía, se revisa qué objetos están dentro del volumen formado por estos planos y se guarda una copia de cada uno de ellos. De esta manera, se obtienen todos los objetos capturados por la cámara, ya sea de manera parcial o total, en el momento de la toma.

Una vez tenemos los objetos que están presentes en el *frustum* de la cámara, es el momento de eliminar las partes de los objetos que no están presentes, lo cual se consigue cortando la geometría. Para llevar a cabo este proceso, es necesario detectar si cada objeto interseca con uno de los planos del *frustum* y, en tal caso, considerar cuatro posibles situaciones

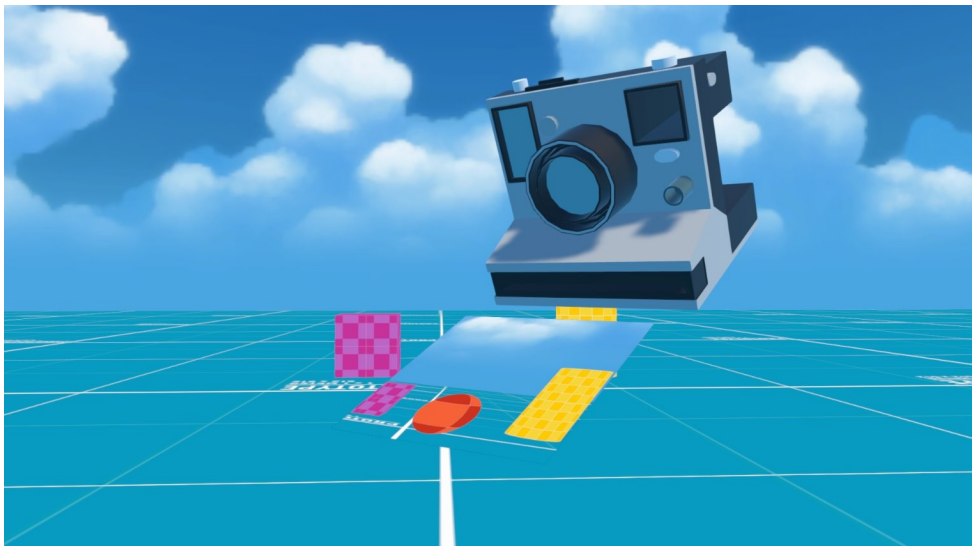

Figura 15: Recreación propia de la mecánica de fotografía mágica

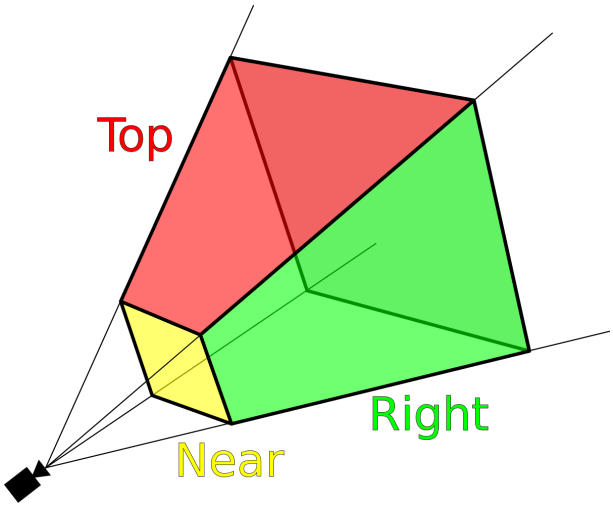

Figura 16: *Frustum* de una cámara [42]

por cada vértice de cada triángulo del objeto en función de cuántos vértices están en el lado positivo del plano (dentro del volumen del *frustum* a partir de ese plano):

- **Ningún vértice en el lado positivo**: El triángulo está completamente fuera del *frustum* y se elimina.
- **Un vértice en el lado positivo**: Se genera un nuevo triángulo con el vértice dentro del *frustum* y los puntos de intersección del triángulo original con el plano.
- **Dos vértices en el lado positivo**: Se generan dos nuevos triángulos utilizando los vértices dentro del *frustum* y los puntos de intersección del triángulo original con el plano.
- **Todos los vértices en el lado positivo**: El triángulo está completamente dentro del *frustum* y se conserva tal cual.

Se puede apreciar un ejemplo visual del segundo caso en la parte izquierda de la Figura 17 y del tercer caso en la parte derecha de la misma. En estas representaciones, la línea azul indica el plano que corta la geometría; los puntos rojos son los vértices en el lado negativo del plano; los puntos verdes son los vértices en el lado positivo; y las líneas amarillas son los rayos que se lanzan para detectar la distancia desde los vértices positivos hasta la intersección con el plano.

Este método asegura que solo las partes visibles de los objetos se mantengan en la fotografía, proporcionando una representación precisa y limpia de la escena capturada por la cámara Polaroid.

Finalmente, una vez tenemos todo lo necesario, lo guardamos en un objeto que se oculta. Cuando el jugador apunta con la fotografía previamente sacada y la utiliza, el objeto con todos los elementos se vuelve visible justo detrás de la fotografía, y esta se oculta, consiguiendo así este impresionante efecto. Todo este proceso se lleva a cabo de una manera similar a un truco de magia, creando una experiencia visualmente atractiva y fluida para el jugador que no sabe cómo se está produciendo el efecto.

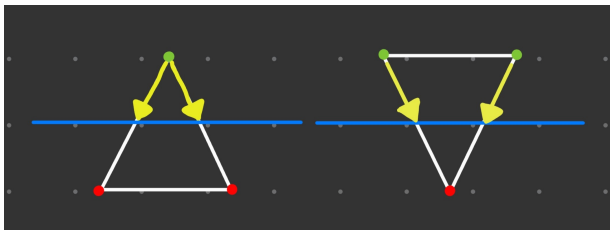

Figura 17: Casos donde hay intersección entre los vértices

4.3.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 4 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica de fotografía mágica.

4.4. Tiempo bala. Superhot

4.4.1. Elementos clave

El elemento clave de la presente mecánica 18 ha sido uno de los más sencillos de desarrollar debido a su simplicidad. Para hacer que el tiempo pase más lento cuando el jugador está quieto, se ha aprovechado la máquina de estados utilizada en todas las mecánicas, creando tres estados nuevos:

- **IdleSuperhotPlayerState**: Correspondiente a cuando el jugador está quieto. Al entrar en este estado, se asignan valores correspondientes al paso de tiempo ralentizado a las variables que manejan la escala de tiempo y el tiempo que se tarda en transicionar a la nueva escala.

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF3.01 - Movimiento del personaje	Entity.cs EntityController.cs WalkPlayerState.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF3.02 - Cámara en primera persona		Component
RF3.03 - Sistema de apuntado	FirstPersonPlayer.cs AimSightChanger.cs	Observer Singleton
RF3.04 - Objetos interactuables	APickable.cs IInteractable.cs CameraItemsController.cs	Observer Prototype
RF3.05 - Captura de los objetos en el frustrum de la cámara	PolaroidCamera.cs	Observer DirtyFlag
RF3.06 - Instanciación de objetos presentes en fotografías	CameraItemsController.cs Photo.cs	Observer DirtyFlag
RF3.07 - Corte de geometría en tiempo real	Photo.cs MeshUtils.cs	Observer DirtyFlag

Tabla 4: Trazabilidad de la mecánica Fotografía mágica.

- **WalkSuperhotPlayerState**: Correspondiente a cuando el jugador está andando. Al entrar en este estado, se asignan valores correspondientes a una escala de tiempo normal (valor 1) a las variables que manejan la escala de tiempo y el tiempo que se tarda en transicionar a la nueva escala.
- **FallSuperhotPlayerState**: Correspondiente a cuando el jugador está en el aire, ya sea saltando o cayendo. Al entrar en este estado, se asignan valores correspondientes a una escala de tiempo normal (valor 1) a las variables que manejan la escala de tiempo y el tiempo que se tarda en transicionar a la nueva escala.

Para evitar cambios bruscos, los valores de la escala de tiempo no se cambian instantáneamente, sino que se utiliza una interpolación que transiciona entre el valor actual y el nuevo valor en un tiempo establecido, manejando toda la lógica por datos. Además, cuando el jugador realiza una acción como coger o lanzar un arma, disparar, etc., mientras está quieto, el tiempo se acelera a escala normal durante un breve momento para dar una mejor sensación de respuesta a esas acciones y representar el movimiento que realiza el personaje al realizarlas.

Figura 18: Recreación propia de la mecánica de tiempo bala

4.4.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 5 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica de tiempo bala.

4.5. Lanza de Draupnir. God of War: Ragnarok

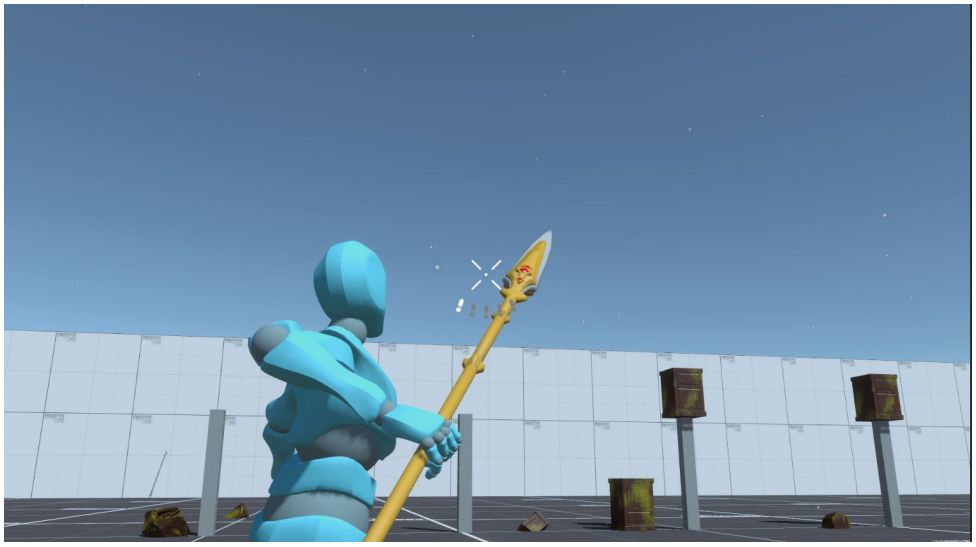

Figura 19: Recreación propia de la Lanza del Draupnir

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF4.01 - Movimiento del personaje	Entity.cs EntityController.cs WalkSuperhotPlayerState.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF4.02 - Cámara en primera persona		Component
RF4.03 - Mirilla dinámica	FirstPersonArmedPlayer.cs CrosshairManager.cs	Observer
RF4.04 - Manipulación del tiempo acorde a las acciones del jugador	PlayerStateManager.cs IdleSuperHotPlayerState.cs FallSuperHotPlayerState.cs WalkSuperHotPlayerState.cs FirstPersonArmedPlayer.cs	Observer State
RF4.05 - Disparo del arma	PlayerWeaponsManager.cs WeaponController.cs	Observer
RF4.06 - Enemigos vulnerables	EnemyPortotype.cs BodyRagdollPart.cs	Observer Component
RF4.07 - Sistema de cambio de armas	PlayerWeaponsManager.cs APickable.cs PickableWeapon.cs	Observer Prototype

Tabla 5: Trazabilidad de la mecánica tiempo bala.

4.5.1. Elementos clave

La parte más importante en la implementación de esta mecánica 19 ha sido el manejo de las diferentes lanzas en la escena, marcando las lanzas ya disparadas y presentes en la escena en la mirilla presente en la interfaz y la explosión de la más antigua cuando se dispara otra lanza una vez superado el límite o la explosión de todas cuando el jugador pulsa el *input* correspondiente. Para la correcta implementación de este sistema, se ha utilizado el patrón *Object Pooling*. Este patrón evita la continua instanciación y destrucción de geometría, lo cual podría afectar negativamente el rendimiento. Mediante *Object Pooling*, se reutilizan los objetos, mejorando así la eficiencia y la fluidez del juego.

Cada vez que el jugador dispara una lanza, se llama al *pool* de objetos para activar una nueva en la mano del jugador y se lanza el evento correspondiente para actualizar la interfaz, la cual indica el número de lanzas presentes en la escena. Tal y como se muestra en la Figura 20, se puede observar que hay dos lanzas presentes en la escena. Una vez que hay tantas lanzas en la escena como el máximo permitido, que en este caso es cinco, la lanza más antigua se destruye para dar paso a una nueva. Si el jugador explota todas las lanzas, la mirilla se actualiza en consecuencia, mostrando que no hay lanzas presentes en la escena. Es importante mencionar que, para la lógica del sistema, no se tiene en cuenta la lanza que tiene el jugador actualmente en la mano.

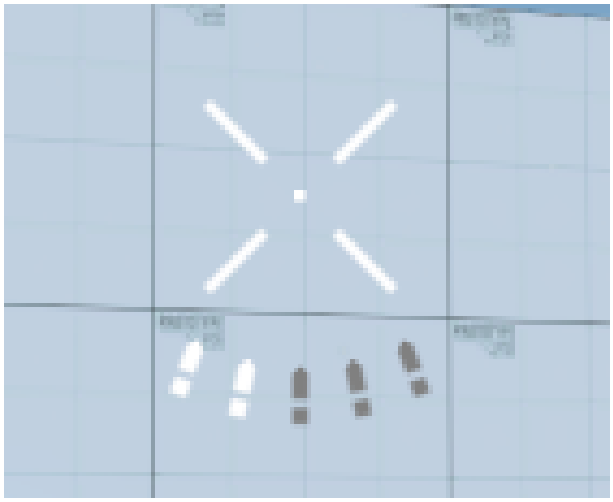

Figura 20: Mirilla e indicador de lanzas presentes en la escena

Además, el uso de buenas prácticas ha permitido la reutilización del código del Hacha de Leviathan para el lanzamiento de la lanza, el giro de esta mientras está en el aire y la lógica de colisiones con elementos del escenario. Esta reutilización de código no solo ha ahorrado tiempo de desarrollo, sino que también ha garantizado la consistencia y robustez en el comportamiento de las armas en el juego.

4.5.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 6 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica de la Lanza de Draupnir.

4.6. Speed Booster and Shinespark. Metroid Dread

4.6.1. Elementos clave

La mayoría de los retos afrontados en esta mecánica 21 están relacionados con el control del tiempo y el movimiento, así como con las mecánicas básicas de un personaje en un videojuego de plataformas, por lo que su explicación no es relevante al ser un problema

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF5.01 - Movimiento del personaje	Entity.cs EntityController.cs WalkPlayerState.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF5.02 - Animaciones de las acciones del personaje	ArmedPlayerAnimator.cs	Observer
RF5.03 - Sistema de apuntado	ArmedPlayer.cs PlayerInputManager.cs PlayerEvents.cs PlayerAimController.cs AimPlayerState.cs CrosshairManager.cs	Observer State Flyweight
RF5.04 - Lanzamiento del arma	ArmedPlayer.cs PlayerInputManager.cs PlayerStatsManager AxeWeapon.cs	Observer Flyweight
RF5.05 - Colisiones del arma	AWeapon.cs	DirtyFlag
RF5.06 - Aparición de nuevas lanzas	Player.cs PlayerInputManager.cs LanceManager.cs LanceObjectPool.cs	Observer Object Pool
RF5.07 - Objetos destructibles	IBreakable.cs BreakableBox.cs	Prototype
RF5.08 - Explosión de lanzas	LanceManager.cs LanceWeapon.cs PlayerInputManager.cs	Observer DirtyFlag
RF5.09 - Cambio de armas	ArmedPlayer.cs WeaponManager.cs PlayerInputManager.cs	Observer

Tabla 6: Trazabilidad de la mecánica Lanza de Draupnir.

más general. Sin embargo, un reto más específico de esta mecánica que sí merece la pena abordar es el cálculo de dirección cuando se usa el *Shinespark*.

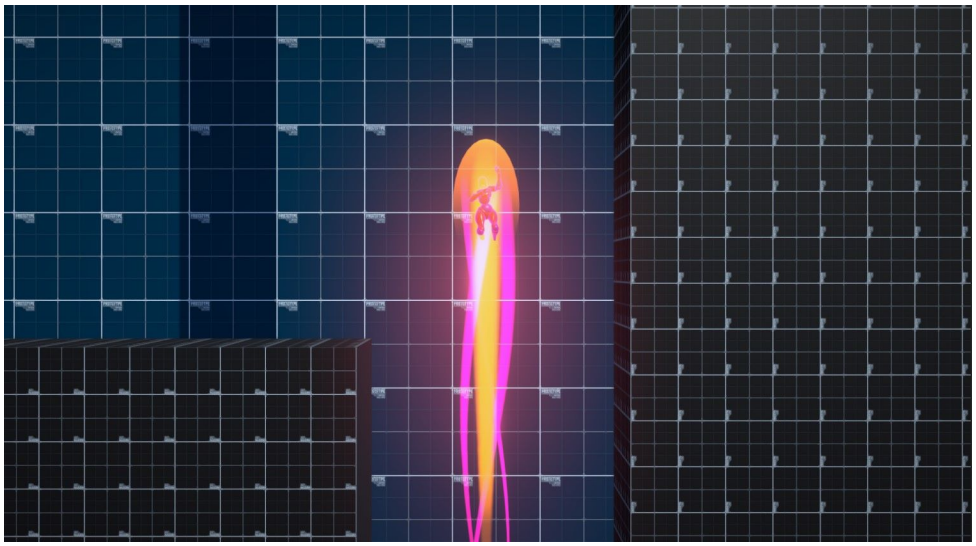

Figura 21: Recreación propia de la técnica *Shinespark* de Metroid Dread

Cuando el jugador usa el *Shinespark*, puede mover el *joystick* de movimiento para seleccionar la dirección en la que moverse. El reto surge al intentar aproximar esta dirección hacia el punto cardinal u ordinal más cercano entre los puntos cardinales y ordinales. Las ecuaciones 3 y 4 muestran el cálculo utilizado para lograr esta aproximación:

Sea $\vec{d} = (x, z)$ la dirección del movimiento del jugador.

$$\theta = \text{atan2}(x, z) \times \frac{180}{\pi} \tag{3}$$

$$\theta_{\text{ajustado}} = \left\lfloor \frac{\theta}{45} + 0{,}5 \right\rfloor \times 45 \tag{4}$$

En la ecuación 3 se calcula el ángulo entre la dirección en el eje x y en el eje z (equivalente al eje y en el *joystick*) y se convierte a grados. De esta manera se obtiene el valor en grados de la dirección que ha seleccionado el jugador. Luego, para aproximar este ángulo al eje cardinal u ordinal más cercano, se utiliza la ecuación 4. En esta parte, el ángulo obtenido se divide por 45 para reducir el rango de ángulos de 0 a 360 grados a un rango de 0 a 8 (ya que $360/45 = 8$). Posteriormente, se multiplica por 45 para devolver el ángulo a su escala original, pero redondeado al múltiplo más cercano de 45 grados. Esto resulta en un ángulo que siempre corresponderá a uno de los ejes cardinales u ordinales.

Este cálculo garantiza que la dirección seleccionada se aproxime al punto cardinal u ordinal más cercano, considerando la disposición circular de las direcciones en un sistema de coordenadas cartesianas.

4.6.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 7 muestra la trazabilidad y patrones de diseño usados en el desarrollo de la mecánica *Shinespark* y *Speed Booster*.

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF6.01 - Control de plataformas básico	Entity.cs EntityController.cs PlatformerPlayer.cs PlayerInputManager.cs PlayerStatsManager.cs	Observer State Prototype Flyweight
RF6.02 - Animaciones de las acciones del personaje	PlatformerPlayerAnimator.cs	Observer
RF6.03 - Sistema de control de *Speed Booster*	PlatformerPlayer.cs	State Flyweight
RF6.04 - Sistema de control de *Shinespark*	PlatformerPlayer.cs	Flyweight
RF6.05 - Estados para cada situación	PlatformerPlayer.cs PlaterStateManager.cs FallPlayerState.cs FloatingPlayerState.cs ShinesparkPlayerState.cs SlidingPlayerState.cs WallDragPlayerState.cs	State Observer
RF6.06 - Bloques destructibles	IBreakable.cs BreakableBox.cs	Prototype

Tabla 7: Trazabilidad de la mecánica *Shinespark* y *Speed Booster*.

4.7. Selector de personajes. Super Smash Bros

4.7.1. Elementos clave

La presente mecánica 22 está desarrollada completamente en un lienzo de interfaz bidimensional, lo que reduce la complejidad general. Sin embargo, uno de los retos afrontados merece ser mencionado, ya que requirió bastante esfuerzo y tiempo abordarlo. Este desafío se relaciona con la colocación del arte de los personajes dentro de las celdas. Dado que los datos de los personajes están contenidos en *Scriptable Objects* de Unity, no es práctico colocar manualmente cada uno de ellos en las celdas ni crear cada celda individualmente, ya que esto sería poco escalable. Aquí es donde surge la idea de utilizar el pivote de cada *sprite*, editándolo mediante el *sprite editor* de Unity, de manera que se coloque en función de su pivote en la celda correspondiente. De esta manera, solo necesitamos ajustar

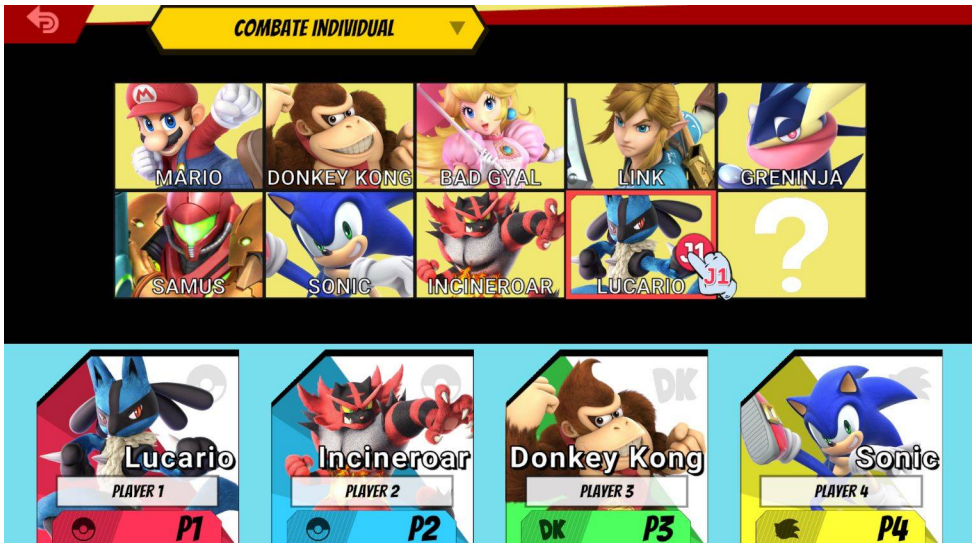

Figura 22: Recreación propia del selector de personajes de Super Smash Bros. Ultimate

el pivote para cada nuevo personaje a fin de establecer su posición central dentro de la celda, lo cual es crucial para la presentación visual deseada. Sin embargo, al modificar el pivote, la imagen apenas se desplazaba dentro de la celda. Después de una extensa investigación, se descubrió que esto se debe a una diferencia fundamental entre el pivote de una textura y el pivote en la interfaz de usuario de Unity. Mientras que la posición del pivote en una textura se expresa en píxeles, en la interfaz de usuario está normalizada entre 0 y 1. Para resolver este problema, se convierten los valores del pivote en píxeles a las unidades normalizadas de la interfaz. Esto se muestra en la ecuación 5, donde el pivote del elemento de interfaz del personaje P_n es igual a la posición del pivote en píxeles P_p dividido por el tamaño en píxeles del *sprite* P_s.

$$P_n = (P_p.x/P_s.x, P_p.y/P_s.y) \tag{5}$$

Esta conversión asegura que el pivote se ajuste correctamente dentro de la interfaz de usuario, permitiendo una colocación precisa y coherente de los personajes en las celdas definidas.

Con esto implementado, la mayoría de los personajes se veían bien. Sin embargo, algunos con *sprites* más pequeños daban la sensación de estar muy lejos. Para solucionar esto, se añadió un parámetro ”zoom” al *Scriptable Object* de cada personaje, que permite modificar el tamaño del elemento de interfaz del personaje. Este tamaño se multiplica por el valor del ”zoom”, simulando así el efecto de que el personaje se vea más cerca.

4.7.2. Buenas prácticas: Trazabilidad y patrones de diseño

La tabla 8 muestra la trazabilidad y patrones de diseño usados en el desarrollo del selector de personajes de Super Smash Bros.

Requisito Funcional	**Archivos de implementación**	**Patrones de diseño usados**
RF7.01 - Celdas de personaje	CellLogic.cs CellData.cs	Prototype Flyweight
RF7.02 - Cuadrícula dinámica	GridElementsManager.cs	Component Prototype Flyweight
RF7.03 - Cursor y ficha para selección	CursorLogic.cs CursorInputManager.cs Token.cs	Observer Flyweight DirtyFlag
RF7.04 - Selección de personaje u elemento	CursorDetection.cs	Component
RF7.05 - Confirmación de selección		Observer
RF7.06 - Cursor dinámico	CursorLogic.cs	Observer Component
RF7.07 - Ranura de jugador	SlotManager.cs CursorDetection.cs	Observer Prototype
RF7.08 - Vuelta al menú principal	GameSceneLoader.cs GameController.cs	Observer Prototype Component Singleton

Tabla 8: Trazabilidad del selector de personajes.

Capítulo 5

Resultados

En este capítulo se presentan los resultados del proyecto desarrollado, con métricas cuantitativas y cualitativas, que pretenden demostrar la calidad del proyecto desarrollado y mostrar tanto el rendimiento técnico como la experiencia del usuario.

5.1. Métricas cuantitativas

Para obtener datos objetivos es necesario usar métricas cuantitativas, las cuales, al estar basadas en números y datos, son imparciales y no contienen subjetividad. La tabla 9 muestra las métricas cuantitativas utilizadas para analizar los resultados de este proyecto.

Métrica	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**Total / Promedio**
Horas de desarrollo esperadas	30	25	40	30	20	40	25	210 / 30
Horas de desarrollo	35	23	41	34	21	32	19	205 / 29.3
FPS Promedio	240	220	200	180	240	260	280	217.14 fps
Bugs identificados	4	0	5	4	2	6	0	21 / 3
Bugs solucionados	4	0	4	4	2	5	0	19 / 3

Tabla 9: Metricas Cuantitativas.

A partir de los datos de la tabla 9 se puede extraer la siguiente información:

- **Tiempo empleado:** Se estimaron inicialmente 210 horas para el desarrollo total de las mecánicas, con un promedio esperado de 30 horas por mecánica. Sin embargo, el tiempo real empleado fue de 205 horas, con un promedio ligeramente menor de 29.3 horas por mecánica. Esto sugiere una planificación eficiente del tiempo y una ejecución acorde a las expectativas iniciales.

- **Rendimiento:** Las mecánicas tienen un rango de rendimiento aceptable, con un tramo de 180 a 280 FPS. Las mecánicas M1, M5, M6, y M7 muestran un rendimiento muy alto (240, 240, 260 y 280 FPS, respectivamente), mientras que M4 tiene el rendimiento más bajo (180 FPS). La variabilidad en el rendimiento sugiere que algunas mecánicas son más demandantes en recursos que otras, y podrían beneficiarse de optimizaciones adicionales. Aun así, un número de fps tan alto en cada una de ellas sugiere una implementación bastante óptima teniendo en cuenta que cada mecánica es un prototipo. Por otro lado, las diferencias en el rendimiento de FPS entre mecánicas indican áreas potenciales para futuras optimizaciones. Es recomendable realizar análisis detallados para identificar posibles cuellos de botella y aplicar técnicas de optimización adecuadas en trabajos futuros.
- **Resolución de errores:** La mayoría de los bugs identificados fueron solucionados (19 de 21), lo cual indica una buena capacidad de depuración y solución de problemas, buenas prácticas como patrones de diseño y trazabilidad pueden haber ayudado a esta rápida y eficiente solución de problemas. Algunas mecánicas (M2 y M7) no tuvieron bugs identificados, lo que podría sugerir una implementación más robusta o menos complejidad en esas áreas.

5.2. Métricas cualitativas

Para obtener unos datos más cercanos a las sensaciones, que es lo que se busca recreando estas mecánicas, hacer sentir lo mismo que con las originales, se utilizaron a 7 sujetos de prueba que probaron las mecánicas originales y, posteriormente, las recreadas en este proyecto. Además, se mostraron videos de las mecánicas recreadas a otras 5 personas para que ofrecieran su opinión y aportaran retroalimentación sobre cada una de ellas. A partir de esas pruebas, se obtuvieron los siguientes datos:

- **Calidad de la mecánica:** La mayoría de los sujetos (5 de 7) consideran que la mayoría de los prototipos tienen en su estado actual calidad suficiente para utilizarlas en diferentes niveles y sacar un juego a mercado. Además, 4 de ellos consideran que las mecánicas recreadas se encuentran al mismo nivel de calidad que las mecánicas originales.
- **Intuitividad de los controles:** Los controles fueron descritos como intuitivos, aunque algunos usuarios sugirieron una mejor aclaración de los controles en las mecánicas 6 y 7.
- **Nivel de diversión:** Los usuarios calificaron las mecánicas con un promedio de 7/10 en términos de diversión, una nota bastante aceptable para tratarse de prototipos. Además, todos ellos consideraron la mecánica 4 como la más divertida.
- **Feedback de los usuarios:** Los usuarios encontraron la mecánica 3 particularmente innovadora y sorprendente, pero la mecánica 6 fue considerada demasiado confusa.
- **Curiosidad de los sujetos:** Las 12 personas que participaron en esta fase sintieron gran curiosidad por cómo funcionaban las mecánicas 2 y 3. Además, se sorprendieron

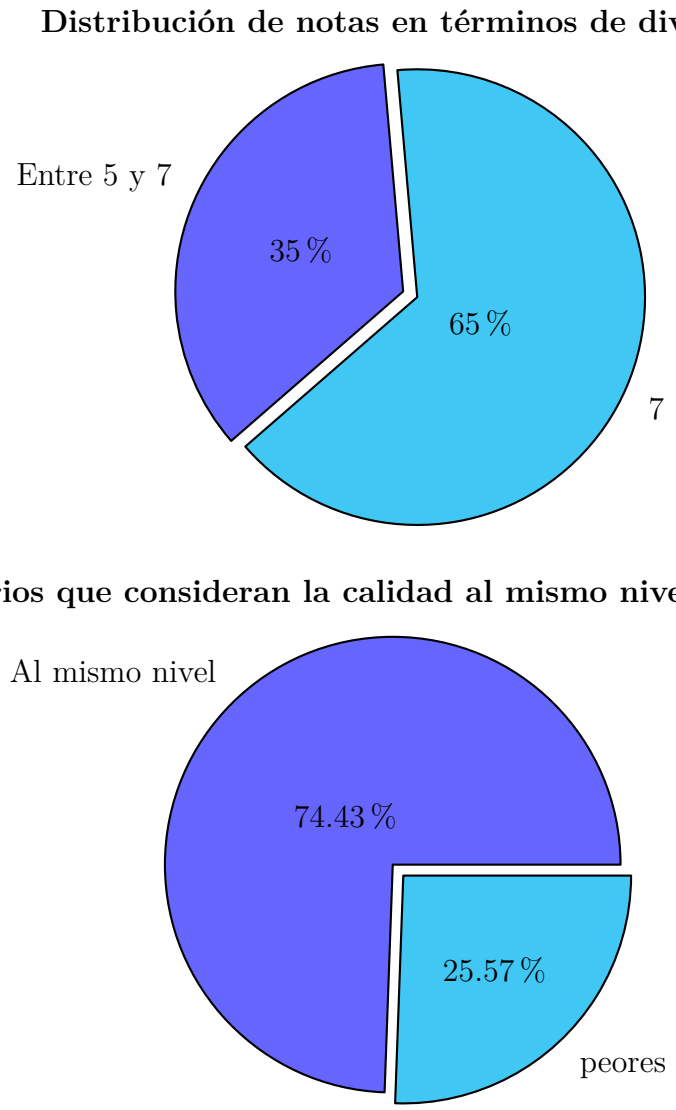

Figura 23: Distribución de diferentes datos cualitativos

de que una sola persona pudiera llevar a cabo su desarrollo, lo cual refuerza la idea planteada en el apartado 1.1 de que generalmente se percibe como algo muy lejano el conseguir diseñar e implementar este tipo de mecánicas.

Capítulo 6

Conclusiones

En el presente capítulo se presentan las conclusiones a las que se ha llegado durante el desarrollo de este proyecto. Describiendo los objetivos conseguidos, conclusiones personales del autor y posibles trabajos futuros en torno al proyecto desarrollado.

6.1. Objetivos conseguidos

Como fruto del desarrollo de este proyecto, se han alcanzado los siguientes objetivos:

- **Objetivo 1**: Se ha aprendido sobre el funcionamiento de diversas mecánicas de valor para el autor, estudiando sus diferentes componentes y extrayendo cómo están diseñadas e implementadas.
- **Objetivo 2**: Se han conseguido resultados en el funcionamiento y visualización de las mecánicas, logrando una fidelidad a las originales, pero con un toque distintivo propio.
- **Objetivo 3**: Se ha logrado un acabado más profesional al pasar todas las mecánicas por un proceso de pulido, añadiendo efectos visuales, sonido, movimientos suaves, efectos de cámara, etc.
- **Objetivo 5**: Se ha comprendido cómo funcionan las mecánicas recreadas por dentro, facilitando esta comprensión al público general, para ello se ha facilitado el código fuente en un repositorio de GitHub y se ha explicado la metodología empleada y la lógica detrás de cada mecánica.
- **Objetivo 6**: Se han adquirido conocimientos sobre el prototipado de mecánicas, siguiendo una metodología para alcanzar resultados óptimos.
- **Objetivo 7**: Se han desarrollado mecánicas siguiendo buenas prácticas de ingeniería de software, como trazabilidad y uso de patrones de diseño, lo que ha permitido una gestión efectiva de la corrección de errores y un código escalable y reutilizable para diferentes proyectos.

- **Objetivo 8**: Se ha conseguido establecer una metodología para el aprendizaje con este tipo de técnica utilizando ingeniería inversa y definiendo las diferentes etapas por las que pasa este proceso de análisis y recreación de mecánicas acercándola al público en este documento para que la gente sea capaz de utilizar este sistema para aprender.
- Se han reforzado conocimientos muy útiles en el desarrollo de videojuegos, abarcando trigonometría, funcionamiento de cámaras, curvas de Bezier, y cómo, en ocasiones, es suficiente con simular una acción y hacer que el jugador crea que es real en lugar de crear todo un proceso fiel a la realidad, lo cual lleva mucho más trabajo.
- Se ha creado un proyecto unificado en el que funcionan todas las mecánicas, permitiendo que cualquier persona las pueda probar fácilmente.

6.2. Trabajos futuros

Uno de los posibles trabajos futuros es el desarrollo de una web donde se incluya el resultado y todo el código, permitiendo que la gente aporte mejoras a las mecánicas implementadas, comparta sus propias mecánicas recreadas o sugiera nuevas mecánicas para implementar. Esto fomentaría una comunidad que aprenda y se nutra del conocimiento de numerosos profesionales y estudiantes. Además, se podrían crear una serie de videos en plataformas como Youtube explicando el proceso de diseño y desarrollo de cada mecánica de manera más visual, para facilitar su aprendizaje y llegar a un público más amplio.

Por otro lado, se podrían implementar otras mecánicas novedosas que el autor hubiera deseado incluir en el presente proyecto y seguir mejorando las actuales.

Finalmente, un posible trabajo futuro que, aunque llevaría bastante tiempo, sería interesante de desarrollar, es la creación de editores de niveles. Esto permitiría a la comunidad desarrollar niveles que funcionen con estas mecánicas, observando así cómo la gente explota las mecánicas y fluye la creatividad.

6.3. Conclusiones personales

Con el desarrollo del presente proyecto el autor ha llegado a la conclusión de que gracias a este método de aprendizaje utilizando ingeniería inversa se puede lograr una gran cantidad de conocimiento de manera autodidacta, gracias a unos objetivos marcados y el conocimiento previo del resultado al que se quiere llegar. Este método de aprendizaje puede ayudar a mucha gente a encontrar la motivación para seguir aprendiendo y lograr resultados atractivos.

Bibliografía

[1] Inc. Red Hat. «¿Qué es el software open source?» En: *Red Hat, Inc.* (2022) (vid. pág. 3).

[2] Free Software Foundation. *GNU-GPL*. Página de información sobre GNU GPL. 1989. URL: https://en.wikipedia.org/wiki/GNU_General_Public_License (vid. pág. 3).

[3] id Software. *DOOM*. Página de información sobre DOOM. 1993. URL: https://es.wikipedia.org/wiki/Doom_(videojuego_de_1993) (vid. pág. 3).

[4] John Carmack. *DOOM Sources Repository*. Repositorio con el código fuente del DOOM original. 2012. URL: https://github.com/id-Software/DOOM (vid. pág. 3).

[5] Sara Borondo. «Una estudiante del MIT logra reproducir Doom en bacterias intestinales, pero llevará 600 años completar el juego». En: *Vandal* (2024) (vid. pág. 3).

[6] Valve Software. *Half-Life 1 game*. Página de información sobre Half-Life 1. 1998. URL: https://es.wikipedia.org/wiki/Half-Life (vid. pág. 3).

[7] Valve Software. *Half-Life 1 Sources Repository*. Repositorio con el código fuente de Half-Life 1 y su motor. 2013. URL: https://github.com/ValveSoftware/halflife?tab=readme-ov-file (vid. pág. 4).

[8] Maddy Makes Games. *Celeste*. Página web oficial del videojuego 'Celeste'. 2018. URL: https://www.celestegame.com/ (vid. pág. 4).

[9] Lexaloffle Games. *Pico-8*. URL: https://www.lexaloffle.com/pico-8.php (vid. pág. 4).

[10] Noel Berry. *Celeste Movement Code Repository*. Repositorio con el código de movimiento de Celeste en la versión de Pico-8. 2021. URL: https://github.com/NoelFB/Celeste/blob/master/Source/Player/Player.cs (vid. pág. 4).

[11] Bethesda Softworks. *Hi-Fi Rush official website*. Página web oficial del videojuego Hi-Fi Rush. 2023. URL: https://bethesda.net/es-ES/game/hifirush (vid. pág. 4).

[12] John Johanas. *Developing 'Hi-Fi RUSH' Backwards and Finding Our Positive Gameplay Loop*. Conferencia de John Johanas sobre cómo se hizo el videojuego Hi-Fi Rush. 2024. URL: https://www.youtube.com/watch?v=pG4UxqRMNX0 (vid. pág. 4).

[13] Nintendo. *Super Mario Wonder official website*. Página web oficial del videojuego Super Mario Wonder. 2023. URL: https://www.nintendo.com/es-es/Juegos/Juegos-de-Nintendo-Switch/Super-Mario-Bros-Wonder-2404150.html (vid. pág. 4).

[14] Shiro Mouri y Takashi Tezuka. *2D and Tomorrow: How the Devs of 'Super Mario Bros. Wonder' Find Joy in Making Side-Scrolling Games*. Conferencia de Shiro Mouri y Takashi Tezuka sobre cómo se hizo el videojuego e ideas descartadas de Super Mario Wonder. 2024. URL: https://www.youtube.com/watch?v=DLyW2rmVukk (vid. pág. 4).

[15] Andrés Cardoso. *Mix and Jam youtube channel*. Canal de youtube de Mix And Jam. 2019. URL: https://www.youtube.com/@mixandjam (vid. pág. 4).

[16] European Patent Office. *Espacenet official website*. Página web oficial de Espacenet, la oficina europea de patentes. 1998. URL: https://worldwide.espacenet.com/patent/ (vid. pág. 4).

[17] Monolith Soft. *The Legend of Zelda: Tears of the Kingdom official website*. Página web oficial del videojuego The Legend of Zelda: Tears of the Kingdom. 2023. URL: https://zelda.nintendo.com/tears-of-the-kingdom/es/ (vid. pág. 4).

[18] Sato Haruki y col. *Patente de movimiento dinámico del jugador en el videojuego 'The Legend of Zelda: Tears of the Kingdom'*. Página web con el registro de la patente de movimiento del jugador sobre objetos del videojuego 'The Legend of Zelda: Tears of the Kingdom'. 2023. URL: https://worldwide.espacenet.com/patent/search/family/087377622/publication/JP2023103273A?q=nftxt%20%3D%20%22nintendo%22%20AND%20nftxt%20%3D%20%22zelda%22 (vid. pág. 4).

[19] La nigromante. «¿QUÉ SON LAS MECÁNICAS DE JUEGO? UNA APROXIMACIÓN AL CONCEPTO». En: *Todas Gamers* (2017) (vid. pág. 5).

[20] Daniel Blanco Aza. *Interactividad: el videojuego como medio de creación artística*. Universidad de Salamanca, 2018. URL: https://gredos.usal.es/bitstream/handle/10366/139126/Blanco%20Aza%2c%20Daniel_Interactividad%20%28GRADO%20HISTORIA%20DEL%20ARTE%29.pdf?sequence=1&isAllowed=y (vid. pág. 5).

[21] Área de tecnología educativa del Gobierno de Canarias. *Página web sobre el pensamiento computacional y sus técnicas y habilidades principales*. Página web sobre el pensamiento computacional y sus técnicas y habilidades principales. s.f. URL: https://www3.gobiernodecanarias.org/medusa/ecoescuela/pedagotic/pensamiento-computacional/ (vid. pág. 5).

[22] *Santa Monica Studio Web Page*. Página web oficial de Santa Mónica Studio. 1999. URL: https://sms.playstation.com/ (vid. pág. 7).

[23] Pillow Castle Games. *Superliminal Web Page*. Página web oficial se Superliminal y del estudio que lo desarrolló. 2019. URL: https://www.pillowcastlegames.com/ (vid. pág. 9).

[24] Wikipedia. *Forced Perspective*. Información sobre la perspectiva forzada. s.f. URL: https://en.wikipedia.org/wiki/Forced_perspective (vid. pág. 9).

[25] Sad Owl Studios. *Viewfinder Steam Page*. Página de steam oficial del videojuego Viewfinder. 2023. URL: https://store.steampowered.com/app/1382070/Viewfinder/ (vid. pág. 11).

[26] Monolith Soft. *Superhot official website*. Página oficial del videojuego Superhot y del estudio desarrollador. 2016. URL: https://superhotgame.com/ (vid. pág. 13).

[27] Wikipedia. *Bullet Time Wikipedia Page*. Información sobre el tiempo bala. s.f. URL: https://es.wikipedia.org/wiki/Tiempo_bala (vid. pág. 13).

[28] Wikipedia. *Matrix Wikipedia Page*. Página de wikipedia de la película Matrix. s.f. URL: https://es.wikipedia.org/wiki/The_Matrix (vid. pág. 13).

[29] Wikipedia. *The Game Awards Winners 2021*. Página de wikipedia de los ganadores de The Game Awards 2021. s.f. URL: https://es.wikipedia.org/wiki/The_Game_Awards_2021 (vid. pág. 17).

[30] Wikipedia. *Metroidvania genre Wikipedia Page*. Página de wikipedia con información sobre el subgénero Metroidvania. s.f. URL: https://es.wikipedia.org/wiki/Metroidvania (vid. pág. 17).

[31] Unity. *Fresnel Effect Unity information web page*. Página de información de Unity sobre el efecto Fresnel. s.f. URL: https://docs.unity3d.com/es/2018.4/Manual/StandardShaderFresnel.html (vid. pág. 20).

[32] Sora Ltd, Bandai Namco y Nintendo. *Super Smash Bros. Ultimate official website*. Página oficial del videojuego Super Smash Bros. Ultimate. 2018. URL: https://www.smashbros.com/es_ES/ (vid. pág. 21).

[33] Unity. *Unity official website*. Página oficial del motor de desarrollo de videojuegos Unity. s.f. URL: https://unity.com/es (vid. pág. 25).

[34] JetBrains. *Rider official website*. Página oficial del IDE para desarrollo de videojuegos y .NET Rider. s.f. URL: https://www.jetbrains.com/es-es/rider/ (vid. pág. 26).

[35] Linus Torvalds. *Git official website*. Página oficial de Git. s.f. URL: https://git-scm.com/ (vid. pág. 26).

[36] Demigiant. *DOTween official website*. Página oficial de DOTween. s.f. URL: https://dotween.demigiant.com/index.php (vid. pág. 26).

[37] Adobe Inc. *Photoshop official website*. Página oficial del editor de fotografía y gráficos Photoshop. s.f. URL: https://www.adobe.com/es/products/photoshop/landpa.html (vid. pág. 26).

[38] Trello Inc. *Trello official website*. Página oficial de Trello. s.f. URL: https://trello.com/es (vid. pág. 27).

[39] Atlassian. «Scrumban: domina dos metodologías ágiles». En: *Atlassian* (2023) (vid. pág. 27).

[40] *Bezier Curves Wikipedia Page*. Página de Wikipedia con información sobre las curvas de Bezier. s.f. URL: https://en.wikipedia.org/wiki/B%C3%A9zier_curve (vid. pág. 34).

[41] Unity Technologies. *Entendiendo el View Frustum*. Página explicativa de la documentación de Unity del funcionamiento del frustum de la cámara. 2018. URL: https://docs.unity3d.com/es/2018.4/Manual/UnderstandingFrustum.html (vid. pág. 37).

[42] Brakeza. *Frustum*. Página explicativa del funcionamiento del frustum de la cámara en un motor. 2019. URL: https://brakeza.com/frustum/ (vid. pág. 38).

Glosario

Input Botón, tecla o dispositivo de entrada que puede utilizar el jugador para realizar ciertas acciones en consecuencia.. 8, 16–19, 33, 42

Open Source Modelo de desarrollo de software en el cual el código fuente se encuentra accesible públicamente para que cualquier persona pueda examinarlo, modificarlo y distribuirlo. Este enfoque fomenta la colaboración abierta y la mejora continua del software por parte de una comunidad global de desarrolladores.. 3

Shinespark Mecánica de técnica cometa presente en Metroid Dread. Consiste en un movimiento a gran velocidad que destruye todo a su paso hasta que choca con una pared.. 2, 17–21, 32, 44–46, 58, 62

Speed Booster Mecánica de acelerón presente en Metroid Dread. Consiste en la carga de un acelerón que permite al jugador desplazarse a gran velocidad y cargar el *Shinespark*.. 2, 17, 19–21, 32, 45, 46

Sprint Un sprint es un periodo de tiempo fijo, generalmente de una a cuatro semanas, durante el cual un equipo de desarrollo ágil trabaja para completar un conjunto específico de tareas del backlog y objetivos previamente planificados.. 28

Apéndices

Apéndice A

Guía de controles y uso del prototipo

En esta guía se describen y muestran los controles generales del prototipo y controles específicos de cada mecánica.

En la escena de cada mecánica, se puede pulsar la tecla 'P' en todo momento para pausar el juego, desde el menú de pausa, se podrá acceder a los controles específicos de cada juego.

A.1. Controles God of War

Además de lo mostrado en la Figura 24 también es posible apuntar con la tecla 'Q', disparar con la tecla 'E' y usar la habilidad especial (explosión de lanzas o atracción de hacha) con la tecla 'R'.

A.2. Controles Superliminal

En la Figura 25 se muestran los controles del prototipo de Superliminal

A.3. Controles Viewfinder

En la Figura 26 se muestran los controles del prototipo de Viewfinder. Además de lo mostrado, también se puede apuntar con la tecla 'Q' y disparar con la tecla 'E'.

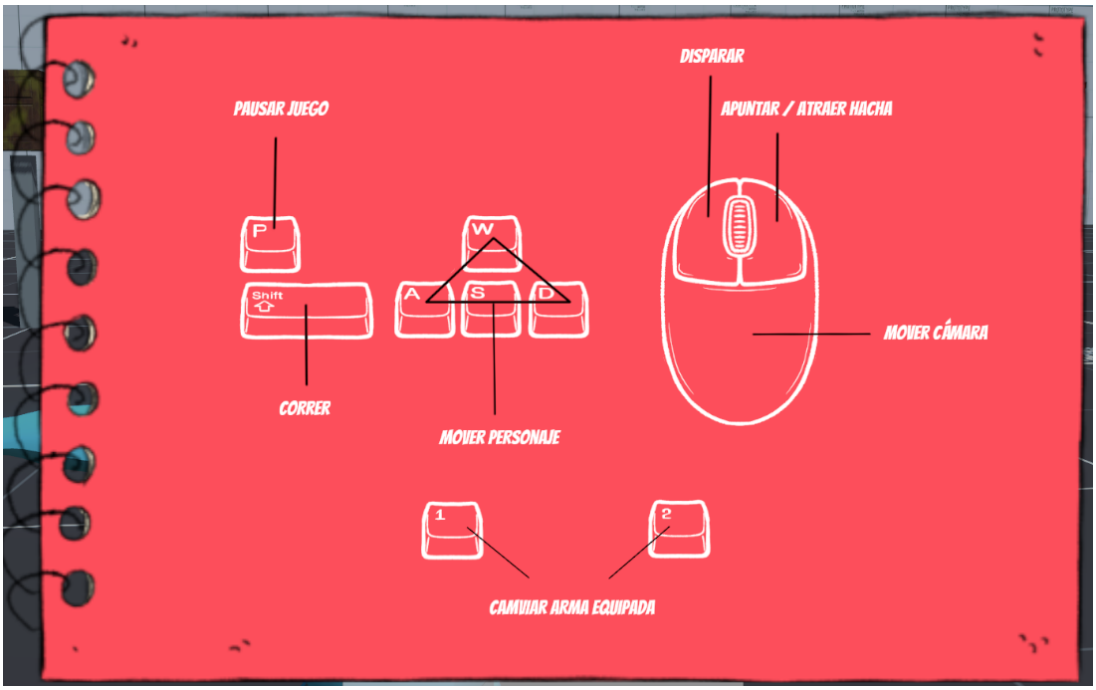

Figura 24: Controles del prototipo de God of War

A.4. Controles Superhot

En la Figura 27 se muestran los controles del prototipo de Superhot.

A.5. Controles Metroid Dread

En la Figura 28 se muestran los controles del prototipo de Metroid Dread. Además de los mostrados, se puede pulsar la tecla 'X' para usar el *Shinespark* una vez cargado.

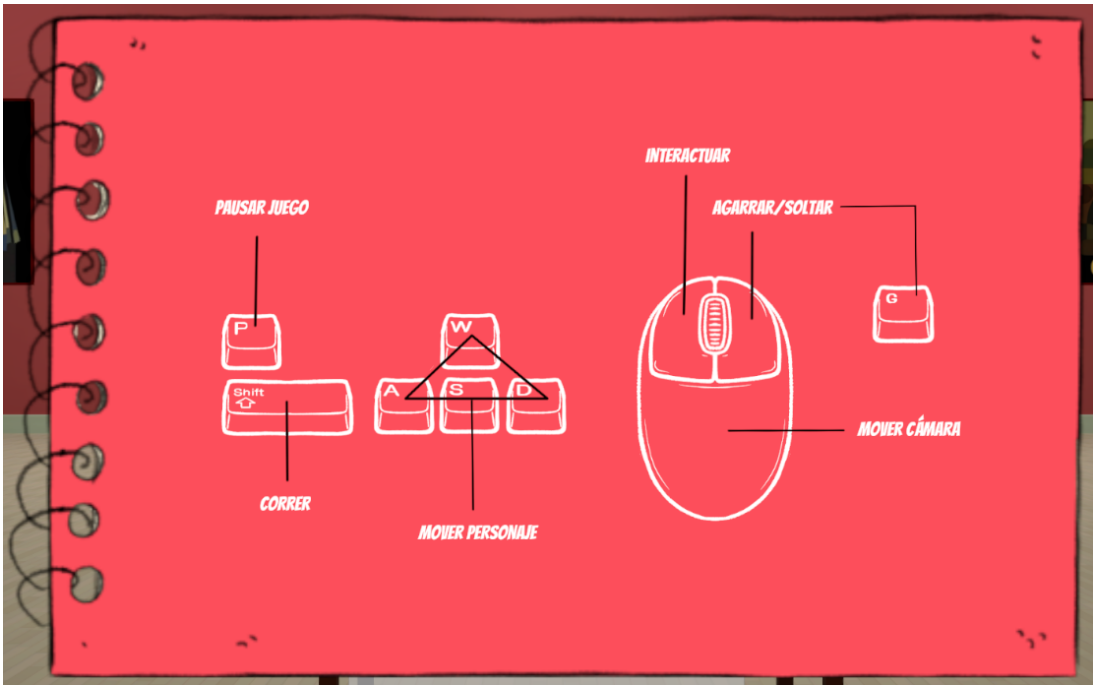

Figura 25: Controles del prototipo de Superliminal

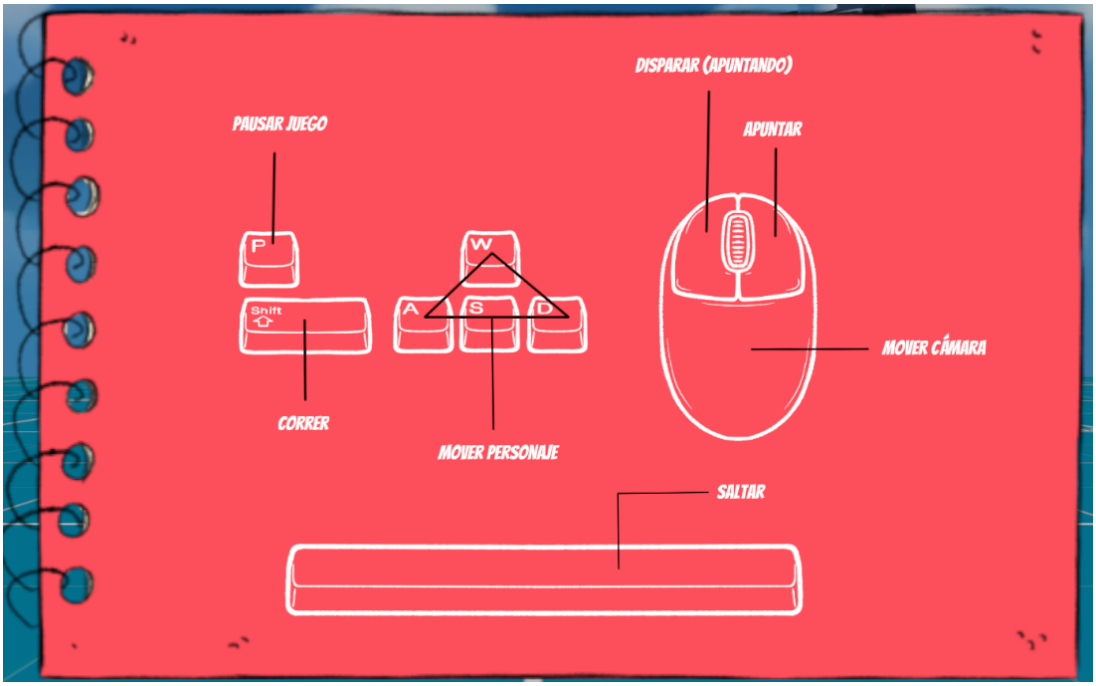

Figura 26: Controles del prototipo de Viewfinder

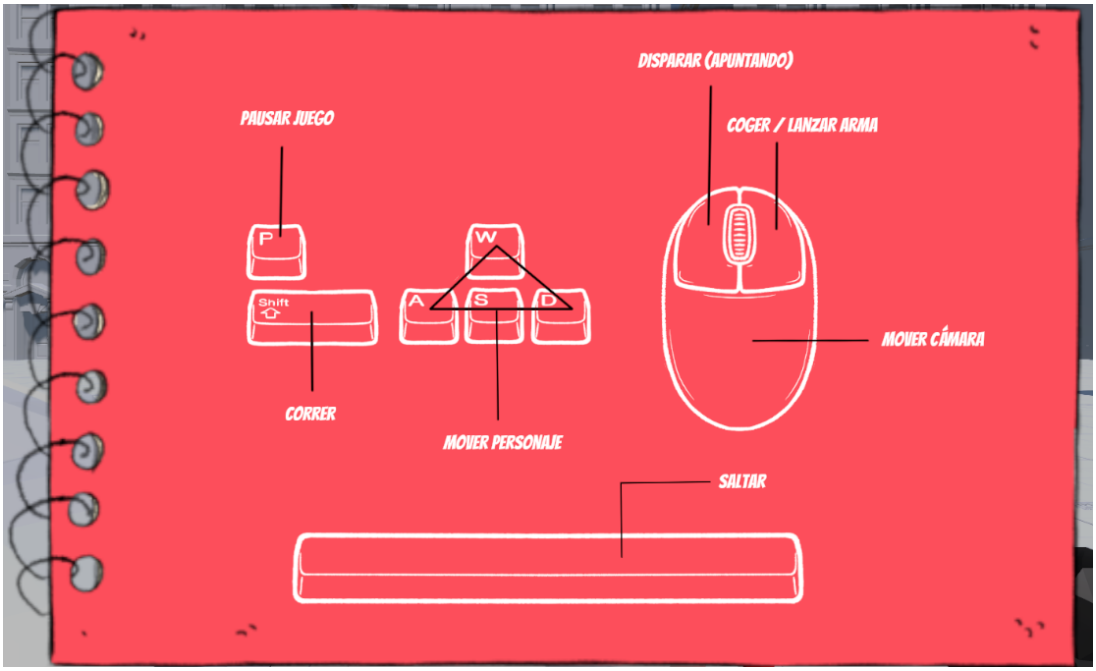

Figura 27: Controles del prototipo de Superhot

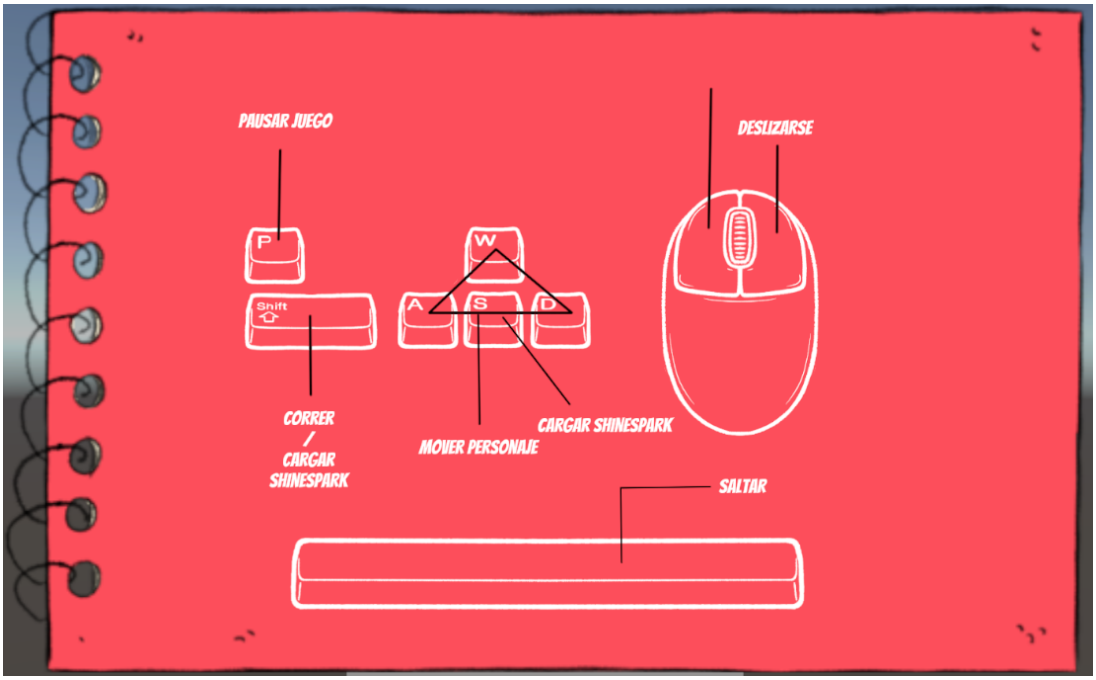

Figura 28: Controles del prototipo de Metroid Dread

Apéndice B

Créditos externos

Efectos visuales y elementos visuales de menú de controles realizados por Alicia Enríquez Martínez.

Modelos 3D y animaciones de enemigos y bala de Superhot, armas y cajas de God of War, camara de Viewfinder y modelos interactuables de Superliminal realizados por Manuel Alejandro Villalba Cruz.

Cursor de Super Smash, Superliminal y arte de menús por Daniel Caldés García.