



Diseño y desarrollo de un videojuego *Platform Fighter*

Alex Gesti Fernández

Director: Lasse Löpfe

Grado: Videojocs

Curso: 2023-24

Universidad: CITM-UPC

Índice

Resumen.....	4
Palabras clave.....	5
Enlaces.....	5
Índice de tablas.....	6
Índice de figuras.....	7
Glosario.....	11
1. Introducción.....	14
1.1. Motivación.....	14
1.2. Formulación del problema.....	14
1.3. Objetivos generales del TFG.....	15
1.4. Objetivos específicos del TFG.....	15
1.5. Alcance del proyecto.....	16
2. Marco teórico.....	17
2.1. Estilo visual.....	17
2.1.1. Diseño de personajes.....	17
2.1.2. Animaciones.....	19
2.1.3. Efectos.....	22
2.2. Diseño de interfaz.....	25
2.3. Gameplay.....	28
2.4. Movimientos de los personajes.....	35
2.5. Controles de mando.....	45
2.6. Diseño de escenarios.....	48
2.7. Recepción.....	52
2.8. Comparación.....	53
3. Gestión del proyecto.....	58
3.1. DAFO.....	61
3.2. Riesgos y plan de contingencias.....	62
3.3. Herramientas del proyecto.....	63
3.4. Análisis inicial de costes.....	64
4. Metodología.....	65
4.1. Diseño Centrado en el Usuario.....	65
4.1.1 Fases de desarrollo de un DCU.....	66
4.1.2 Investigación y requisitos de usuario.....	67
4.1.3 Diseño hacia el usuario.....	68
4.1.4 Prototipado.....	68
4.1.4 Tests con usuarios.....	69

5. Desarrollo del proyecto.....	71
5.1. Documento de diseño.....	71
5.1.2. Concepto.....	71
5.1.3. Jugabilidad.....	75
5.2. Desarrollo del prototipado.....	111
5.2.1. Objetivos del prototipo.....	111
5.2.2. Proceso de creación.....	112
5.3. Gestión de comunidad y retroalimentación.....	149
6. Validación del proyecto.....	152
7. Conclusiones.....	155
7.1. Evaluación del proyecto.....	155
7.1.1. Desarrollo técnico.....	155
7.1.2. Iteración basada en feedback.....	156
7.1.3. Logros clave.....	157
7.1.4. Conclusiones generales.....	158
7.2. Líneas de futuro.....	159
7.2.1. Mejoras potenciales.....	159
7.2.2. Nuevas funcionalidades.....	159
7.2.3. Planes cara a futuro.....	160
8. Bibliografía.....	161
8.1. Artículos.....	161
8.2. Videos.....	171
9. Anexos.....	174

Resumen

El género de los *platformer fighters* es un subgénero de los juegos de lucha que combina elementos del género *plataformas* y del género de juegos de lucha, en donde los personajes se enfrentan en escenarios que incluyen elementos de salto, mientras disponen de un arsenal de movimientos y habilidades para combatir. Este género suele tener un ritmo rápido de juego, que requiere habilidad del jugador tanto de juegos de plataformas como de lucha.

Este género de juegos presenta una inconsistencia en su popularidad y recepción por parte de los jugadores. Mientras algunos títulos destacan por sus personajes reconocibles o mecánicas complejas, otros acaban en el olvido, teniendo que cancelar el proyecto aun estando en medio de desarrollo. Este trabajo de investigación se centra en abordar esta inconsistencia y en encontrar formas de hacer que el género sea atractivo tanto para jugadores dentro de la comunidad como para aquellos fuera de ella, centrándose en las mecánicas y la jugabilidad en lugar de la identidad de los personajes.

El objetivo de este proyecto es desarrollar un prototipo que sea atractivo para una amplia audiencia, manteniendo un equilibrio entre mecánicas accesibles y profundas. Además, se busca proporcionar una guía para los desarrolladores interesados en este género, así como ofrecer a los jugadores un proyecto que satisfaga sus necesidades y expectativas. También se pretende que este proyecto permita adquirir experiencia en el desarrollo de juegos de este subgénero, siempre manteniendo al jugador como el eje central del proyecto.

Con ello, la propuesta se basa en el desarrollo iterativo del juego, comenzando con un prototipo inicial que se someterá a pruebas con un grupo reducido de jugadores, llamados *playtesters*, los cuales serán tanto jugadores casuales como jugadores ya acostumbrados a este género. Posteriormente, se realizarán ajustes y mejoras en versiones posteriores, con la participación tanto de los playtesters originales como de nuevos jugadores externos. Así, con esta metodología, se permitirá obtener una retroalimentación continua y garantizar que el resultado final satisfaga las necesidades y preferencias de la audiencia objetivo.

Palabras clave

Investigación, conocimiento, desarrollo, entretenimiento, visual, reto.

Enlaces

- Repositorio de Github: <https://github.com/alexgesti/Platform-Fighter-TFG>
- Descarga del proyecto:
<https://github.com/alexgesti/Platform-Fighter-TFG/releases/tag/v0.20.15092024>
- Video presentación: <https://youtu.be/CWbi0DsT31M>

Índice de tablas

Tabla 1: Comparativa - Datos básicos.....	Pág. 55
Tabla 2: Comparativa - Modos y mecánicas	Pág. 57
Tabla 3: Fechas de proyecto.....	Pág. 60
Tabla 4: Análisis DAFO.....	Pág. 63
Tabla 5: Análisis de riesgos y soluciones.....	Pág. 64
Tabla 6: Análisis de costes.....	Pág. 66

Índice de figuras

Figura 1: A Fight Between Live Action and Animation [Graphics].....	Pág. 17
Figura 2: A Fight Between Live Action and Animation [Graphics].....	Pág. 18
Figura 3: A Fight Between Live Action and Animation [Graphics].....	Pág. 19
Figura 4: Meteos (Nintendo DS).....	Pág. 27
Figura 5: Make Important Elements Bigger [UI].....	Pág. 28
Figura 6: Knockback.....Pág. 28	
Figura 7: Button Settings [UI].....	Pág. 48
Figura 8: Los escenarios se crean como “arte” y “juego”.....	Pág. 51
Figura 9: 3.1. Los métodos del diseño centrado en el usuario.....	Pág. 68
Figura 10: Movimientos de un Platformer Fighter - Neutral Attack.....	Pág. 83
Figura 11: Movimientos de un Platformer Fighter - Dash Attack.....	Pág. 84
Figura 12: Movimientos de un Platformer Fighter - Forward Attack.....	Pág. 85
Figura 13: Movimientos de un Platformer Fighter - Up Attack.....	Pág. 86
Figura 14: Movimientos de un Platformer Fighter - Down Attack.....	Pág. 86
Figura 15: Movimientos de un Platformer Fighter - Neutral Aerial Attack.....	Pág. 87
Figura 16: Movimientos de un Platformer Fighter - Forward Aerial Attack.....	Pág. 88
Figura 17: Movimientos de un Platformer Fighter - Back Aerial Attack.....	Pág. 88
Figura 18: Movimientos de un Platformer Fighter - Up Aerial Attack.....	Pág. 89
Figura 19: Movimientos de un Platformer Fighter - Down Aerial Attack.....	Pág. 89
Figura 20: Movimientos de un Platformer Fighter - Forward Smash Attack.....	Pág. 90
Figura 21: Movimientos de un Platformer Fighter - Up Smash Attack.....	Pág. 91
Figura 22: Movimientos de un Platformer Fighter - Down Smash Attack.....	Pág. 91
Figura 23: Movimientos de un Platformer Fighter - Neutral Special Attack.....	Pág. 92

- Figura 24: Movimientos de un Platformer Fighter - Side Special Attack.....Pág. 92
- Figura 25: Movimientos de un Platformer Fighter - Up Special Attack.....Pág. 93
- Figura 26: Movimientos de un Platformer Fighter - Down Special Attack.....Pág. 94
- Figura 27: Movimientos de un Platformer Fighter - Pummel.....Pág. 95
- Figura 28: Movimientos de un Platformer Fighter - Forward Throw.....Pág. 95
- Figura 29: Movimientos de un Platformer Fighter - Back Throw.....Pág. 96
- Figura 30: Movimientos de un Platformer Fighter - Up Throw.....Pág. 97
- Figura 31: Movimientos de un Platformer Fighter - Down Throw.....Pág. 97
- Figura 32: Movimientos de un Platformer Fighter - Escudo.....Pág. 98
- Figura 33: Movimientos de un Platformer Fighter - Voltereta delantera.....Pág. 99
- Figura 34: Movimientos de un Platformer Fighter - Voltereta trasera.....Pág. 99
- Figura 35: Diseño de nivel - Partes de un escenario.....Pág. 103
- Figura 36: Diseño de nivel - 3 plataformas flotantes.....Pág. 104
- Figura 37: Diseño de nivel - 2 plataformas flotantes variante A.....Pág. 105
- Figura 38: Diseño de nivel - 2 plataformas flotantes variante B.....Pág. 106
- Figura 39: Diseño de nivel - escenario flotante.....Pág. 106
- Figura 40: Diseño de nivel - Circuito 1.....Pág. 108

Figura 41: Controles - Asignación de controles de teclado.....	Pág. 109	de	controles	de
Figura 42: Controles - Asignación de controles de Xbox.....	Pág. 110	de	controles	de
Figura 43: Controles - Asignación de controles de PS4/PS5.....	Pág. 110	de	controles	de
Figura 44: Controles - Asignación de controles de Pro Controller de Nintendo Switch.....	Pág. 111			
Figura 45: Controles - Asignación de controles de GameCube.....	Pág. 111	de	controles	de
Figura 46: Proceso de creación - Adobe Photoshop.....	Pág. 113	-		
Figura 47: Proceso de creación - Modelo base de personaje.....	Pág. 114			
Figura 48: Proceso de creación - Actions.....	Pág. 116	-		
Figura 49: Proceso de creación - Asignación de eventos de acciones.....	Pág. 118			
Figura 50: Proceso de creación - Asignación de eventos de acciones.....	Pág. 119			
Figura 51: Proceso de creación - Tapping.....	Pág. 121	-		
Figura 52: Proceso de creación - Movimiento lateral.....	Pág. 123	-		
Figura 53: Proceso de creación - Movimiento lateral.....	Pág. 124	-		
Figura 54: Proceso de creación - Full hop y Short hop / Doble salto.....	Pág. 126			
Figura 55: Proceso de creación - Full hop y Short hop / Doble salto.....	Pág. 127			
Figura 56: Proceso de creación - Velocidad aérea.....	Pág. 128	-		

Figura 57: Proceso de creación - Gravedad.....	Pág. 128
Figura 58: Proceso de creación - Fast fall (Caída rápida).....	Pág. 129
Figura 59: Proceso de creación - Caída desde plataformas.....	Pág. 130
Figura 60: Proceso de creación - Creación de colisión para detección de físicas..Pág. 131	
Figura 61: Proceso de creación - OnCollisionEnter.....	Pág. 132
Figura 62: Proceso de creación - OnCollisionStay.....	Pág. 133
Figura 63: Proceso de creación - OnCollisionExit.....	Pág. 133
Figura 64: Proceso de creación - Ignorar colisiones.....	Pág. 134
Figura 65: Proceso de creación - Activar o desactivar colisiones.....Pág. 134	
Figura 66: Proceso de creación - Ignorar colisión en ciertas situaciones.....Pág. 135	
Figura 67: Proceso de creación - Raycast de detección de suelo.....Pág. 136	
Figura 68: Proceso de creación - Raycast de detección de pared.....Pág. 137	
Figura 69: Proceso de creación - Cálculo de los límites.....	Pág. 137
Figura 70: Proceso de creación - Cálculo del área visible.....	Pág. 138
Figura 71: Proceso de creación - Posición de la cámara dentro de los límites.....Pág. 138	
Figura 72: Proceso de creación - Recopilación de posiciones.....Pág. 139	
Figura 73: Proceso de creación - Cálculo del área que abarca a todos los jugadores.....Pág. 139	

Figura 74: Proceso de creación general.....	Pág. 140	Cálculo
Figura 75: Proceso de creación - Creación de colliders.....	Pág. 142	
Figura 76: Proceso de creación - Asignación de animaciones.....	Pág. 143	
Figura 77: Proceso de creación - Asignación de valores de cada movimiento.....	Pág. 144	
Figura 78: Proceso de creación - Asignación de valores al enemigo.....	Pág. 145	
Figura 79: Proceso de creación - Cálculo del ángulo y la dirección del Knockback.....	Pág. 146	
Figura 80: Proceso de creación - Aplicación del daño y ajuste del porcentaje.....	Pág. 147	
Figura 81: Proceso de creación - Cálculo de la velocidad de Knockback / Aplicación de fuerzas y gestión del Knockback.....	Pág. 148	
Figura 82: Proceso de creación - KO.....	Pág. 148	

Glosario

- **Plataformas/Platformer:** género de videojuegos en el cual se centra principalmente en llegar a la otra punta del mapa en el cual deberá superar obstáculos en tiempo real con un personaje que usa varias habilidades, como escalar o saltar, y objetos, como armas.
- **Fighting game/Juego de lucha:** género de videojuegos en el cual dos personajes en pantalla se enfrentan en combate. Estos personajes suelen estar desarmados y suelen usar estilos de lucha como karate, taekwondo, boxeo o capoeira, entre muchos otros, aunque también pueden usar estilos de lucha con armas de corto y largo alcance.
- **Playtest:** es un método de control de calidad el cual se desarrolla en diferentes momentos durante el desarrollo del videojuego, en el cual un grupo de usuarios, estos llamados playtesters, juegan versiones inacabadas del juego para ver cómo funcionan diferentes mecánicas, diseño de niveles y elementos varios del juego, con tal de encontrar y que se puedan resolver diferentes errores de este.
- **Melee:** acrónimo del juego Super Smash Bros. Melee, el segundo juego de la saga Super Smash Bros., disponible en la videoconsola Nintendo Gamecube. Este juego, pese que ser un juego de 2002 y a diferencia de otras versiones, corre a 60 fotogramas por segundo y dispone de una esquiva aérea, la cual combinada con la tracción del suelo de los personajes, permite que se deslicen por el suelo de manera rápida y con largas distancias, haciendo que se convirtiera en la versión más competitiva de la saga.
- **Mods:** acrónimo de la palabra modificación, en este caso relacionado a las modificaciones en los videojuegos. Pueden ser añadidos al juego, revisiones del juego, cambios en la interfaz, una conversión total del juego original o simples modificaciones artísticas.
- **Smash 64:** acrónimo del juego Super Smash Bros., el primer juego de la saga Super Smash Bros., disponible para la Nintendo 64. En este juego 12 personajes de Nintendo se enfrentan entre ellos en unos escenarios de lucha con movimiento 2D pero con perspectiva 3D y obstáculos repartidos por toda el área. Entre 2 y 4 jugadores se enfrentan de forma simultánea y en la misma pantalla, hasta dejar K.O. al oponente tantas veces como sea posible en un límite de tiempo.
- **Game design:** el diseño de juegos es el proceso de crear la estructura, reglas, mecánicas y experiencias que conforman un videojuego, con el objetivo de lograr una experiencia de juego divertida, desafiante y satisfactoria.
- **Hardcore:** adjetivo referido a una forma fuerte de realizar una actividad.
- **Escenario/Stage:** nombre adquirido a aquellas localizaciones en donde se enfrentan los personajes o completan objetivos en un Platform Fighter. Estos también pueden adquirir el nombre de nivel, mapa, arena o estadio.
- **ESports:** actividad que consta de competir en juegos en línea contra otra gente, normalmente por dinero, y normalmente vistos por otra gente usando el

internet, aunque algunas veces se organizan eventos especiales para poder verlos en presencialmente.

- **Stream:** es cualquier contenido de medios, ya sea en vivo o grabado, que se puede disfrutar a través de computadoras y dispositivos móviles con internet y en tiempo real. Podcasts, webcasts, películas o programas de TV son varios ejemplos de tipo de contenido de streaming.
- **Streamer:** son jugadores profesionales o especializados en temáticas de videojuegos. No solo juega, gran parte de la transmisión también comentan. Su trabajo consiste en pasar bastantes horas delante del ordenador, jugando y comentando contenido para el público. Para ello usan algunas plataformas donde difunden sus contenidos como Twitch, YouTube, Facebook live y Mixer.
- **Stick:** también conocido como la palanca de mando o joystick, es un periférico de entrada que consiste en una palanca que gira unos 360º sobre una base e informa su ángulo o dirección al dispositivo que está controlando.
- **Counter:** terminología conocida en los juegos de lucha, que consiste en detener el ataque del rival mientras lo está realizando con otro ataque.
- **Frame por segundo/FPS/Frames:** son la cantidad de imágenes consecutivas que se muestran en pantalla por segundo. También se podría decir que es una secuencia de varios fotogramas que pasan a gran velocidad.
- **Frames de ventaja:** se describe como “el primero que se recupera después de que un movimiento haya hecho contacto o sea bloqueado”, otra manera de describirlo el tiempo que tienes después de realizar un golpe y volver a tener el control, en comparación a tu rival, que aún está en un estado de aturdido o de recibir el golpe, antes de que él también vuelva tener el control del personaje.
- **Mob:** es una criatura controlada por la máquina, el cual suele ser un enemigo que te permite obtener puntos de experiencia, dinero, objetos o completar misiones. A veces estos mobs no necesariamente se les debe de derrotar, en algunos casos hasta deben ser protegidos.
- **Cartoon:** es un tipo de personaje de *dibujo animado*, que suele ser surrealista y cómico, con frecuencia nervioso e histérico, y capaz de desafiar las leyes de la física o de la lógica tal y como se conocen.
- **Frames de cancelación:** período de tiempo dentro de la animación donde se puede realizar una *cancelación* del movimiento exitosa.
- **Meteos:** es un juego de puzzles lanzado para la **Nintendo DS**, donde los jugadores deben evitar que un planeta se convierta en una nova al alinear y alinear bloques de colores al espacio. La jugabilidad se centra en mover los bloques vertical u horizontalmente para crear combos y evitar acumulaciones, con variaciones en la mecánica según el planeta en el que se juegue.
- **Ultimate:** acrónimo del juego Super Smash Bros. Ultimate, el último juego de la saga Super Smash Bros., disponible en la videoconsola Nintendo Switch. Este juego consta con todos los personajes de la saga que salieron anteriormente y con casi todos los escenarios de la saga que salieron con anterioridad, convirtiéndose en el juego más completo de la saga.

- **Wiimote:** conocido realmente como Wii Remote, es el mando principal de la consola Wii de Nintendo. Este funciona principalmente con sensores de movimiento y se puede usar para apuntar e interactuar con los elementos en pantalla, como si de un mando se tratase. Además, dispone de una ranura en la parte baja que le permite conectar una gran cantidad de dispositivos para mejorar el gameplay del juego.
- **Nunchuck:** accesorio para el mando principal de la consola Wii de Nintendo que permite controlar de distintas formas los juegos. Este dispone de una stick adicional para el mando para aquellos juegos que lo requieran.
- **Wii Classic Controller/Wii Pro Controller:** mando que se debe conectar al mando principal de la consola Wii de Nintendo, el cual funciona como un mando clásico de videojuegos. Este dispone de un diseño que retoma elementos de los mandos de las antiguas consolas de Nintendo, aunque la versión *Pro* dispone de una ergonomía más cómoda y asequible.
- **Gamepad de Wii U:** mando principal de la consola Wii U de Nintendo, que tiene forma de mando común de consolas, pero dispone de una pantalla táctil en medio de esta, además de giroscopio, para aquellos juegos que necesiten controles de movimiento.
- **Nintendo 3DS:** consola portátil de Nintendo, la cual puede ejecutar gráficos en 3D estereoscópico sin el uso de gafas 3D o accesorios adicionales. La consola es sucesora de la Nintendo DS, ofreciendo retrocompatibilidad con juegos de esta.
- **Joy-Cons:** mandos principales para la consola Nintendo Switch. Estos mandos funcionan como un solo mando cuando están conectados a la consola, pero se pueden separar y usarlos, como un solo mando o como dos mandos por separado. Estos mandos disponen de una sujeción para poderlos poner juntos cuando no están conectados a la consola, además de controles de movimiento parecidos a su antecesores de Wii.
- **Power-ups:** se trata de un objeto común en los videojuegos, el cual al ser utilizado u obtenido proporciona al personaje características especiales o mejoras en sus habilidades que lo hacen más poderoso.
- **Modo VS:** también llamado *versus*, es un término, en los juegos de lucha, que se refiere a un enfrentamiento entre luchadores.
- **Indie:** se refiere a *videojuego independiente*, siendo, en este caso, un juego desarrollado por una desarrolladora pequeña sin soporte directo, de manera financial o de manera técnica por grandes publicadores de juegos.

1. Introducción

1.1. Motivación

Durante años he sido jugador de la saga de juegos *Super Smash Bros*, empezando desde el segundo juego, *Melee*, con ello he seguido sin descanso toda información relevante sobre su creador, **Masahiro Sakurai**, y estudiado su la filosofía de desarrollo, seguido los productos de esta saga, leído toda clase de artículos, testeado y observado los juegos de este subgénero, participado activamente en la comunidad de este género y hasta participado en el desarrollo de *mods* para *Smash 64*.

Con este último me he dado cuenta de algo, que todo desarrollador, sea ser de *mods* o de proyectos parecidos a este, nunca han sabido cómo equilibrar el factor diversión y competitividad, haciendo que entre diferentes proyectos se probarán diferentes cosas, algunas con unos resultados positivos y otras con resultados realmente negativos.

Con lo cual, planteé esta posibilidad: ¿Realmente estos juegos están hechos para cualquier consumidor o son productos solamente para los fans? Ya que, a excepción del primer juego, *Smash 64*, todos los proyectos de este género de juegos han tenido unos planteamientos muy diferentes entre ellos, pero ninguno se ha planteado lo que él primero buscaba, **hacer un juego para divertir a la gente sin importar la experiencia de este**.

1.2. Formulación del problema

El género de *Platform Fighter* no está generalizado entre la gente, siendo este que normalmente se reconoce por el videojuego *Super Smash Bros*, el cual fue el precursor de este subgénero de juegos de lucha.

Con lo cual, el público, en un inicio, era reducido a los fans de los personajes de esas sagas de juegos que estaban dentro del videojuego *Super Smash Bros*, pero con el paso de los años, ha ido aumentando gracias a que han ido saliendo nuevas versiones de este. Con este incremento de jugadores de este género, el resto de empresas recrearon las bases de este juego, pero sin conseguirlo, sea por los personajes que no sean reconocibles, o por las mecánicas de este que sean muy incómodas.

Con esto se encontró que este género de juegos se veía reducido a uno o dos juegos representativos, haciendo que el resto se acabaran descartando, además de cerrando las puertas a nuevos jugadores que estuvieran interesados a probarlo, pero no pudieran por la complejidad de las mecánicas o por la propia comunidad.

El problema que ha acabado surgiendo ha sido que las desarrolladoras que han intentado trabajar en este género de juegos sin ninguna guía ni nada parecido, han tenido que acabar descartando el proyecto por no acabar de funcionar en comparación a sus competidores y no ser juegos **hechos para todo el mundo**, sino **hecho para el jugador** de este subgénero de juegos de lucha.

Aun así, el desarrollador de la saga *Super Smash Bros*, **Masahiro Sakurai**, está realizando actualmente una serie de vídeos de *game design* los cuales aportan información de todo tipo, aunque generalmente se centra en la saga ya nombrada. Pese a eso, esta información que él aporta es muy leve, sin llegar nunca a entrar en la parte importante de esta.

1.3. Objetivos generales del TFG

Los objetivos de este proyecto empiezan por la realización de una guía a libre interpretación para los desarrolladores que estén interesados por este género, ya que este no está muy abarcado en el mercado en comparación a otros géneros y estos suelen presentar unas incongruencias en cuanto a mecánicas y jugabilidad.

Además, también otro objetivo es la búsqueda de un juego que encuentre el equilibrio entre la diversión de este en un ambiente casual y uno *hardcore*, sin que ninguno de los bandos llegue aburrirse o le parezca demasiado difícil a la hora de jugarlo.

1.4. Objetivos específicos del TFG

Los objetivos más específicos de este proyecto empiezan por desarrollar un juego de este género el cual la comunidad haya tenido contribución en que se podría cambiar, mejorar o añadir de nuevo.

Otro objetivo es aportar alguna nueva mecánica o jugabilidad nueva, la cual no se haya usado o usado en un nivel muy inferior a la idea desarrollada.

El siguiente objetivo sería conseguir romper la idea de que estos juegos necesitan disponer de unos personajes que sean de otros juegos, haciendo que con unos nuevos personajes, este juego acabe gustando y tenga el mismo reconocimiento que tienen otros de este género.

Por último, siendo este objetivo más personal, me gustaría desarrollar un proyecto de este calibre, el cual sea reconocido por la comunidad y sirva como proyecto principal, con lo que se podrá ir trabajando y acabar lanzando al mercado.

1.5. Alcance del proyecto

El proyecto se enfocará en el desarrollo de un juego del género *Platform Fighter* de manera limitada. Este se centrará en la creación de un prototipo funcional que demuestre las mecánicas de este género con alguna aportación, como nuevas maneras de jugar a este género o nuevas clases de personajes, manteniendo todo esto en un estándar acorde al nivel del jugador.

Con lo cual, este proyecto va dirigido a jugadores tanto casuales como experimentados en este género de juegos, principalmente, los que busquen una experiencia de juego emocionante, accesible y bien diseñada.

Además de este grupo, los desarrolladores interesados en este género de juegos, pueden utilizar este trabajo como guía o referencia para informarse y mejorar sus propios proyectos, teniendo así una mejoría de calidad y accesibilidad en ellos.

2. Marco teórico

En el mundo diverso de los videojuegos, hay pocas sagas que se comparan con Super Smash Bros. Con cada juego nuevo, esta serie ha superado las expectativas, atrapando a jugadores de todas las edades y convirtiéndose en una sensación cultural.

En este trabajo, nos metemos de lleno en el increíble mundo de Super Smash Bros. Ultimate, el punto más alto de una saga que ha dejado huella en la historia de los videojuegos. Analizamos en detalle cómo se desarrolló este juego, desde sus ideas desarrolladas por su creador, **Masahiro Sakurai**, explorando todas las cosas complicadas que hicieron que se convirtiera en algo tan especial.

2.1. Estilo visual

2.1.1. Diseño de personajes

En esta saga de juegos 3D, los personajes ya existentes en otras sagas de videojuegos, dicho de otra manera, este juego es una mezcla entre diferentes personajes de videojuegos para que luchen entre ellos.

De esta manera cada personaje tiene su propio estilo en su saga original, por ejemplo, tenemos a *Wario* de la saga *Wario Ware*, que tiene un estilo más *cartoon* y tenemos a *Ganondorf* de la saga *The Legend of Zelda*, que tiene un estilo más realista. El problema con esto, es que si se ponen en el mismo estilo artístico que en el material original de dicho videojuego, estos se verían raro y no podrían sentirse fuera de lugar juntos en el mismo mundo.



A Fight Between Live Action and Animation [Graphics]

Fuente: <https://www.youtube.com/watch?v=luB-5GqW55Q&list=PLgKCjZ2WsVLSO4Zq-JuJzGvhTe1QqRSZ5&index=8>

Por ello, el creador de la saga, **Masahiro Sakurai**, mantiene el modelo o idea base del personaje, pero se le hacen algunos cambios, los cuales serían:

- **Bajar la saturación.** Ya que algunos de los personajes originalmente tienen colores *grandes* y *atrevidos*, pero en *Smash* los representan un poco más pálidos y se ven afectados más fácilmente por las fuentes de luz.

De esta manera los colores más chillones no chocarán tanto con personajes con paletas más tenues y se mantendrá un equilibrio.

- **Mantener la texturas similarmente detalladas.** Ya que personajes más *cartoon* usan colores más planos o simples, mientras que personajes más realistas tienen texturas más detalladas, a la hora de transportarlo a *Smash*, incluye más detalle a las texturas a los personajes más *cartoon*, mientras que simplifica los personajes más detallados.
- **Modificar las proporciones de los cuerpos.** Ya que algunos personajes resultan más altos que otros, suelen alterar el tamaño de estos personajes para que todos estén al mismo tamaño, pese que en su contenido original midan un par de centímetros o hasta bastantes metros de altura, aunque en otros casos simplemente cambian un par de proporciones del cuerpo para mantener el estilo de *Smash*.
- **Realizar los refinamientos necesarios.** Algunos personajes, al ser tan simples necesitan tener un poquito más de detalle en sus diseños para así unificar un estilo el cual sea el mismo para todos los personajes del juego.

Un ejemplo podría ser en el caso del personaje *Olimar* del juego *Pikmin*, el cual en su diseño original tiene un traje de astronauta increíblemente simple, mientras que en *Smash* lo detallan un poco más, o en el caso de los *Ice Climbers* del juego *Ice Climbers* de la *Nintendo Entertainment System*, o también llamada *NES*, los cuales disponen de unos pinchos en sus botas, ya que son escaladores.



A Fight Between Live Action and Animation [Graphics]

Fuente: <https://www.youtube.com/watch?v=luB-5GqW55Q&list=PLqKCjZ2WsVLSO4Zq-JuJzGvhTe1QqRSZ5&index=8>

- **Expresar personajes de manera más realista.** Hay personajes los cuales, en sus juegos originales, tienen expresiones más al puro estilo cómic, como ocurre con *Donkey Kong* y *Yoshi* de los juegos de *Super Mario*, los cuales a la hora de pestañear no tienen párpados, pero para unificar un mismo arte en el juego, les ha creado párpados para que estos los cierren siempre que sea necesario.



A Fight Between Live Action and Animation [Graphics]

Fuente: <https://www.youtube.com/watch?v=luB-5GqW55Q&list=PLgKCjZ2WsVLSO4Zq-JuJzGvhTe1QqRSZ5&index=8>

2.1.2. Animaciones

En el desarrollo de videojuegos, las animaciones desempeñan un papel crucial en la experiencia del jugador, especialmente en juegos de lucha como **Super Smash Bros.** Cada movimiento, cada golpe y cada interacción visualiza la intensidad y la emoción del combate, y es responsabilidad del equipo de desarrollo transmitir estas ideas de manera efectiva a través de las animaciones.

Cuando se diseñan luchadores, es esencial considerar cómo se animarán sus acciones y cómo estas animaciones se integrarán con el sistema de juego en general. Para lograr esto, el proceso comienza con la creación de una **lista detallada de animaciones** que enumera todos los movimientos posibles de un personaje. Esta lista no solo describe los movimientos textualmente, sino que también **especifica la duración de cada animación** y cuándo ocurren los ataques.

Para comunicar ideas más complejas y poses importantes, como la inactividad, el movimiento y los ataques, se utilizan **figuras posables**. Estas figuras se pueden **posicionar en diferentes poses y fotografiar desde varios ángulos** para capturar cada detalle. Las imágenes resultantes se incorporan a las especificaciones del juego, proporcionando al equipo de animación una referencia visual clara y detallada de lo que se espera de cada animación.

Una vez que se proporciona este material detallado al equipo de animación, comienza el proceso de convertir estas ideas en animaciones efectivas. Cada animación se construye a partir de cuatro elementos clave:

- **Espera (Standby):** Esta es la pose base desde la cual se derivan todas las demás. Se establece la postura inicial del personaje cuando no está realizando ninguna acción. Las animaciones de espera son cruciales ya que definen la personalidad y el estilo de lucha de cada personaje.
- **Preparación (Lead-In):** También conocida como anticipación o preparación para el ataque. Esta fase comienza con el movimiento de un ataque, como levantar un arma para golpear. En Super Smash Bros., estas animaciones a menudo comienzan con un movimiento grande y repentino en el primer fotograma para indicar al jugador que su entrada ha sido reconocida. La transición rápida entre la espera y la preparación ayuda a proporcionar retroalimentación inmediata al jugador y puede incentivar a su oponente a esquivar el ataque.
- **Ataque (Attack):** Esta es la fase clave del ataque, donde se registra el golpe. Se presenta una pose firme y distintiva en la que el ataque puede hacer contacto con el oponente. En algunos casos, se utiliza una técnica llamada *hit stop* para detener la animación cuando el ataque aterriza, lo que resalta el momento crucial del impacto y hace que sea más fácil de leer para el jugador. La efectividad de un ataque a menudo depende de la claridad y contundencia de esta pose.

- **Seguimiento (Follow-through):** Esta fase ocurre después de que el ataque ha terminado y el personaje regresa a la pose de espera. La pose se mantiene durante un momento adicional antes de volver a la espera, lo que permite al jugador absorber el resultado del ataque. En Super Smash Bros., se asignan *frames de cancelación* al seguimiento, lo que permite a los jugadores recuperar el control sin volver completamente a la espera. Esto proporciona un poco de flexibilidad y fluidez en el juego.

Por otro lado, cuando un jugador carga un **ataque Smash**, el personaje entra en una pose de preparación y comienza a temblar en su lugar. Esta técnica se implementa para **proporcionar retroalimentación inmediata** al jugador en el momento en que se presiona el botón de ataque. En lugar de interpolar desde la pose de espera, el juego **prioriza la respuesta instantánea** al jugador. Esto ayuda a reducir la latencia percibida y a garantizar que el jugador sepa que su entrada ha sido reconocida. Aunque esta técnica puede no eliminar completamente el retraso causado por problemas de red o de televisión, ayuda a minimizar la incertidumbre del jugador y a mantener una experiencia de juego más fluida y receptiva.

Las animaciones de ataque son vitales para la jugabilidad, ya que **afectan drásticamente cómo se siente** el juego. Uno de los momentos más importantes es cuando el ataque conecta, lo que llaman las *poses de golpe*. Aquí hay algunos puntos clave sobre cómo crear estas poses:

- **Exageración:** Es importante exagerar los movimientos para que sean claros incluso desde lejos.
- **Centralización:** Mantén los ataques centrados para que se registren correctamente.
- **Asimetría:** Evita hacer poses completamente simétricas, a menos que sea por efecto cómico o intencional.
- **Torso en torsión:** Evita simplemente doblar el torso hacia un lado; una torsión lo hará más interesante.
- **Momento del golpe:** Para los ataques con espada, el momento del golpe debe ocurrir después de que la espada haya avanzado un poco, para no perder la sensación de impacto.
- **Perspectivas:** Asegúrate de que las poses se vean bien desde diferentes perspectivas, especialmente en el lado izquierdo y derecho de la pantalla.
- **Silueta distintiva:** Es crucial que las poses tengan una silueta clara y distintiva, con todos los miembros claramente visibles.

Después del ataque, viene la **fase de seguimiento**, donde el personaje queda momentáneamente vulnerable. Esto es crucial para equilibrar la jugabilidad, ya que los ataques fuertes **deben dejar al personaje vulnerable** proporcionalmente. La fase de

seguimiento ocupa **la mayor parte del tiempo** después del ataque, y su duración debe ser proporcional al tiempo de acción del ataque. Es esencial que la animación de seguimiento ilustre claramente el peso y la fuerza del ataque, y que las cancelaciones de *frames* se utilicen para permitir que el jugador recupere el control de manera oportuna.

Con esto, en el diseño de las animaciones de cuando se **recibe daño** suelen tener una cosas a tener en cuenta que no suelen cumplir el resto de animaciones de esta lista:

- **Adoptar una pose al recibir el impacto.** En el caso de *Super Smash Bros.*, en el momento en que un personaje recibe daño, **cambian rápidamente** a una pose que refleja dolor. Este primer fotograma de la animación de daño tiene aplicado un efecto de pausa, por lo que es fácil de ver. Por lo tanto, es vital que esta pose realmente exprese el dolor.
- **Impacto grande con poses de daño.** La *pose de daño* y la *pose de espera* pueden ser bastante similares en algunos juegos, pero les gusta optar por algo más extremo que no se parezca en nada a la pose de espera. De esta manera, se puede distinguir claramente cuando el personaje ha sido golpeado, incluso desde lejos.
- **Interpolación.** Sin embargo, no se pasa instantáneamente a la animación de daño, como en otros casos. Utilizando interpolación, consiguen una transición desde la pose en el momento del impacto al primer fotograma de la animación de daño.
- **Problemas de alcance del hitbox.** Los personajes siempre entran en *poses de daño* al ser golpeados, sin importar la situación, por lo que si el *hitbox* del oponente **se alarga**, podrían entrar en esa *pose de daño* a una distancia considerable. Si bien hay méritos en aplicar un efecto de pausa en el fotograma extendido del *hitbox*, es preferible enfatizar *el sentido del impacto*.
- **Tipos de reacciones.** Las reacciones al daño en *Super Smash Bros.* pueden dividirse en **alto, medio y bajo**, y se eligen según la proximidad al punto de impacto. Además, la reacción en el suelo puede tener tres niveles de intensidad: **pequeño, medio y grande**. Multiplicándose todo, terminamos con nueve tipos de reacciones posibles.
- **Tipos de cuerpo únicos.** En comparación con otros juegos de lucha que presentan principalmente personajes humanoides, *Super Smash Bros.* tiene una amplia variedad de luchadores con formas únicas. Incluso los personajes humanoides pueden ser portadores de espadas, escudos u otras armas, por lo que hay mucha variación.
- **Facilitar la lectura.** Para todas las animaciones de *daño*, se enfatiza romper la simetría del personaje y hacer que su silueta sea claramente legible.

Por último, hay una peculiaridad que *Super Smash Bros. Ultimate* añade en esta entrega en cuanto a las animaciones, las **animaciones invertidas**, un concepto heredado de juegos como *Street Fighter*, los cuales reusan sus animaciones, como si de un reflejo se tratase, para así que el usuario pueda ver lo que el personaje está haciendo y no lo tenga de espaldas. En un inicio, esto no lo plantearon ya que requería potencia de la consola y no sabían cómo crearlo para luchadores con espadas.

2.1.3. Efectos

Se describen los elementos visuales que acompañan los ataques a los oponentes como *marcas de golpe*. En los juegos de **Sakurai**, estas marcas suelen ser **llamativas**, con un aspecto general de **dibujos animados**. Reconoce que este enfoque puede resultar extraño al trabajar con temas realistas, por lo que intenta encontrar un equilibrio adecuado para cada título representado.

Además de las *marcas de golpe*, a veces se emplean otros efectos, como salpicaduras de sangre. *Super Smash Bros. Ultimate* presenta una amplia variedad de estas marcas, incluyendo movimientos de llamas, perforaciones de espadas, impactos de tipo trueno y movimientos de congelación, entre otros. **Sakurai** también sugiere que los efectos de guardia se pueden considerar como un tipo de marca de golpe.

Dispone de una lista de criterios que considera importantes para que una marca de golpe sea efectiva:

1. **La llamativa apariencia debería coincidir con su impacto.** El propósito es proporcionar una retroalimentación visual clara y divertida para el jugador, por lo que generalmente es mejor hacer que tus marcas de golpe sean llamativas.
2. **Visuales equilibrados.** Solo usar mezcla aditiva o contrastes débiles podría dejarle con un juego que se siente insatisfactorio.
3. **Posicionamiento preciso del golpe en sí mismo.** Un golpe ocurre cuando hay *colisión de daño*, en otras palabras, cuando las cajas de *hitbox* y el objetivo se encuentran, pero si esto no se verifica correctamente, la marca de golpe puede aparecer en una posición extraña. La cámara en *Smash Bros.* se encuentra alejada, por lo que las cajas de colisión son más grandes, haciendo que esto a veces puede resultar desafiante al posicionar estos efectos.
4. **Variedad de elementos.** En lugar de simplemente arrojar un solo efecto en la pantalla, también se puede agregar una *onda de choque* detrás del personaje, así como un rocío ascendente. Los oponentes mecánicos también reciben algunas chispas.

No se quiere que las cosas se vuelvan demasiado *ruidosas*, pero algunas adiciones de color bien sincronizadas harán que los efectos funcionen en una variedad de fondos.

5. **Los personajes no están eclipsados.** Esto es bastante complicado en la práctica, pero es mejor si las *marcas de golpe* no oscurecen demasiado a los personajes mismos. Esto se puede lograr bajando la prioridad de visualización de las *marcas de golpe*, pero esto no siempre funciona tan bien, ya que es posible que solo se vea el terreno.

6. **Duración apropiada.** Los efectos para golpes rápidos y repetidos deben ser cortos, mientras que los de otros ataques deberían ser más largos. Sin embargo, dependiendo del contenido del juego, se sugiere que incluso esos efectos "largos" sean rápidos.
7. **Niveles de brillo contrastantes.** Idealmente, los efectos de las marcas de golpe deberían ser bastante claros contra fondos claros y oscuros. Los efectos exclusivamente brillantes no serán lo suficientemente visibles en un fondo claro, así que es bueno incluir algunos colores oscuros también.

Estas son excelentes técnicas para incluir en cualquier juego que use *hit stop* para marcar impactos. En la mayoría de los casos, las *marcas de golpe* **pueden continuar sus animaciones** incluso durante la detención del golpe, así que deberías aprovechar ese tiempo para incluir una **presentación aún más detallada**.

- **Sacudida de pantalla**

La sacudida de pantalla, o *screen shake*, es una técnica comúnmente utilizada en el diseño de videojuegos para aumentar el impacto y la inmersión de la experiencia del jugador. Consiste en mover la pantalla de manera dinámica y temporal en respuesta a ciertos eventos del juego, como ataques, explosiones o impactos.

Según como describe **Masahiro Sakurai**, existen dos métodos principales para implementar la sacudida de pantalla:

1. **Movimiento de la Cámara:** Este método implica mover la propia cámara en el espacio 3D del juego. Al hacerlo, la sacudida afecta a todos los elementos de la escena, pero los objetos más distantes pueden no ser tan afectados como los más cercanos. Es importante considerar cómo afectará este movimiento a la percepción visual del jugador y al diseño general del juego.
2. **Movimiento de la Imagen Renderizada:** Aquí, en lugar de mover la cámara, se mueve toda la imagen después de ser renderizada. Esto garantiza que toda la pantalla, incluida la interfaz de usuario (UI), se sacuda uniformemente. Sin embargo, puede revelar espacio vacío en los bordes de la pantalla, lo que debe tenerse en cuenta al implementarlo.

Es fundamental encontrar un equilibrio en la intensidad y frecuencia de la sacudida de pantalla. Demasiada sacudida puede resultar molesta o distraer a los jugadores, mientras que muy poca puede no tener el impacto deseado en la experiencia del juego.

Para una implementación efectiva de la sacudida de pantalla, se recomienda:

- **Ajustar la intensidad de la sacudida** según la distancia de la cámara para mantener la coherencia visual.

- **Utilizar diferentes patrones de sacudida** según la situación en el juego, como sacudidas grandes para eventos importantes y sacudidas más sutiles para acciones más pequeñas.
- **Considerar cómo la sacudida afectará a la interfaz de usuario (UI)** y a otros elementos visuales en pantalla para mantener una experiencia fluida y cohesiva.

2.2. Diseño de interfaz

- Menús

El diseño de los menús de esta saga se inspira en principios innovadores, como lo explica **Masahiro Sakurai**, director de la serie. Desde su experiencia previa, **Sakurai** ha implementado un enfoque distintivo para mejorar la accesibilidad y la usabilidad de los menús.

En particular, destaca la importancia de los iconos de **diferentes tamaños** para indicar la **prioridad de los modos y funciones**, una técnica que adoptó por primera vez en **Meteos**. Este enfoque jerárquico facilita la **identificación rápida** de las opciones más utilizadas y mejora la experiencia del usuario al proporcionar una guía visual clara.



Meteos (Nintendo DS)

Fuente: https://tcrf.net/Meteos_%28Nintendo_DS%29

Además, enfatiza el **equilibrio entre claridad y estilo** en el diseño de los menús. Aunque la estética es importante, la legibilidad y organización de la información tienen prioridad. Este equilibrio se refleja en la elección de **colores, tipografías y disposición** de los elementos en pantalla, asegurando que la interfaz sea atractiva y fácil de entender para los jugadores.

Un aspecto destacado del diseño de los menús en **Super Smash Bros. Ultimate** es el uso de colores temáticos para diferenciar las secciones principales, como "Smash", en color rojo, y "Spirits", en color verde. Esta técnica, inspirada en la disposición de los botones en el control de **GameCube**, ayuda a los jugadores a identificar rápidamente las áreas de interés y facilita la navegación dentro de los menús.



Make Important Elements Bigger [UI]

Fuente: https://www.youtube.com/watch?v=va8wt2yGviY&list=PLgKCjZ2WsVLTcEHQFioJF263lp_yoxen&index=5

- **En batalla**

La interfaz de usuario durante una partida desempeña un papel crucial en la experiencia del jugador al proporcionar información vital de manera clara y concisa. Uno de los elementos más destacados es **el porcentaje** que se muestra encima de cada personaje, que indica su nivel de **daño acumulado**. Este porcentaje no solo sirve como una representación visual del **estado de salud** del personaje, sino que también determina su **vulnerabilidad a ser lanzado** fuera del escenario por los ataques de sus oponentes.

Los fans, que se han dedicado a investigar internamente el código fuente de los juegos, han ofrecido una explicación detallada del **significado detrás de este porcentaje**, revelando **cómo se calcula** y su importancia en el juego estratégico. Este enfoque en la transparencia y comprensión del sistema de juego ayuda a los jugadores a tomar decisiones informadas y a desarrollar estrategias efectivas durante las peleas.

La forma de calcularlo sería la siguiente:

$$\left(\left(\left(\left(\left(\frac{p}{10} + \frac{p \times d}{20} \right) \times \frac{200}{w + 100} \times 1.4 \right) + 18 \right) \times s \right) + b \right) \times r$$

Knockback

Fuente: <https://www.ssbwiki.com/Knockback>

Teniendo que:

- **p** es el **porcentaje del objetivo**, contado después de que se suma el daño del ataque.
- **d** es el **daño** que infinge el ataque.
- **w** es el **peso** del objetivo. Se establece en 100 si el ataque es independiente del peso.
- **s** es la **escala** de empuje del ataque dividida por 100.
- **b** es el **empuje base** del ataque.
- **r** es una **serie de razones** basadas en una serie de factores, como la **dificultad, la tasa de lanzamiento, efectividad del tipo, cambios de tamaño, rabia, etc.**

Además del porcentaje, la interfaz de usuario también incluye una variedad de **elementos visuales** que proporcionan información adicional durante la partida. Por ejemplo, se pueden mostrar **medidores de habilidades especiales** para ciertos personajes, como el medidor de Rebelión de *Joker*, protagonista de *Persona 5*, o el indicador de tinta de *Inkling*, mascota de la saga *Splatoon*. Estos elementos no solo agregan profundidad táctica al juego, sino que también **reflejan las características únicas** de cada personaje y su jugabilidad distintiva.

Otro aspecto destacado es la evolución de la interfaz de usuario a lo largo de las diferentes entregas de la serie. Con *Super Smash Bros Ultimate*, se ha logrado una **mayor claridad y legibilidad** en la pantalla de juego, con una **distribución más equilibrada** de elementos y una presentación más intuitiva de la información. Esto mejora la accesibilidad y la experiencia del jugador, especialmente para aquellos nuevos en la franquicia o menos familiarizados con sus mecánicas de juego.

2.3. Gameplay

- **Cómo se juega**

El objetivo principal es **lanzar a los oponentes fuera del escenario** para eliminarlos de la partida. Esto se logra **infiriendo daño** a los oponentes mediante una variedad de movimientos y ataques. Cada vez que un personaje recibe un golpe, **su porcentaje de daño aumenta**, lo que hace que sea más fácil lanzarlo fuera del escenario con golpes posteriores. Por lo tanto, los jugadores deben equilibrar el ataque y la defensa mientras intentan evitar ser lanzados ellos mismos.

La mecánica del juego es accesible para jugadores de todos los niveles de habilidad. Los controles son simples de aprender, con movimientos básicos como **golpear, saltar y bloquear**. Sin embargo, hay una profundidad considerable en el juego, con **técnicas más avanzadas** como **esquivar, realizar ataques especiales y ejecutar combos** de movimientos que permiten a los jugadores desplegar estrategias más elaboradas y dominar el juego.

Además de los movimientos estándar, cada personaje tiene acceso a movimientos especiales únicos que pueden ser clave para cambiar el curso de la batalla. Estos movimientos especiales varían desde poderosos ataques de largo alcance hasta habilidades defensivas que pueden proteger al jugador de los ataques enemigos.

- **Reglas**

Este juego, tiene diferentes reglas de juego mientras se juega en el modo *VS*, el principal modo de juego en el cual se enfrentan los personajes. Las más importantes son:

- **Tiempo:** En este modo hay que derrotar al mayor número de rivales dentro de un límite de tiempo. Cada vez que se noquee a un rival, se obtendrá un punto, en caso de que se noquee al jugador, éste perderá un punto.

Al finalizar el tiempo, se compará quien tiene más puntos, el cual será el ganador. En caso de que más de un jugador tenga los mismos puntos, estos se enfrentarán en **muerte súbita**, un combate en el todos los participantes iniciarán con un **daño de 300%**. Si en ese tiempo ninguno ha muerto, los límites del escenario se irán haciendo más pequeños y empezarán a caer bombas del cielo hasta que uno de los jugadores muera.

- **Vidas:** Parecido a las reglas de “*tiempo*”, hay que noquear a los rivales, pero en este caso los jugadores contarán con un número de vidas, el

cual se irán perdiendo cada vez que el jugador muera. El ganador será el jugador que disponga con el mayor número de vidas.

De normal, este modo no dispone de un contador de tiempo, pero se puede combinar con el tiempo para que la partida tenga un tiempo límite y finalice la partida cuando no quede más tiempo.

- **Smash especial:** en este modo, se dispone de varias reglas para participar en un combate. Estas reglas pueden hacer que haya combates por energía, más parecido a como un juego del género *fighter* funciona, equipar a los luchadores con accesorios o hacerlos moverse rápidamente, entre otras opciones.

● Objetos

Los objetos son **artilugios que aparecen durante la partida y pueden ser recogidos y utilizados** por los personajes. Algunos pueden ser sujetados de manera normal, mientras que otros aplican un efecto, y casi todos pueden ser lanzados a los oponentes. Estos son una parte crucial del juego, especialmente para el público casual, donde la diversión caótica se prioriza sobre la competición seria.

Aún así, existen algunos objetos los cuales se usan a nivel competitivo **por ser parte de los movimientos del personaje**, como los *nabos* de *Peach*, de la saga *Super Mario Bros.*, o las *bombas* de los *Links*, de la saga *The Legend of Zelda*.

Existen varias formas de que los objetos aparezcan durante una partida. La forma más normal es aparecer aleatoriamente en medio de la partida. Pese a ello, **ciertos escenarios también generan automáticamente objetos** específicos durante eventos del escenario, al igual que pasa con los personajes.

Los personajes solo **pueden sostener un objeto a la vez**, siempre que sea de un solo uso. Si un objeto se deja solo, eventualmente desaparecerá, parpadeando brevemente antes. Los objetos recogidos por los personajes pueden ser soltados o lanzados con el botón de agarre, y pueden ser soltados si el personaje recibe daño. El daño causado por un objeto lanzado depende de qué tan rápido se esté moviendo. La mayoría de los objetos también se pueden tragar, incluso pueden curar a estos personajes cuando lo hacen, a menos que el objeto sea explosivo, lo que infle un daño menor.

Si otro jugador logra adquirir un objeto lanzado o soltado antes de que se active, la propiedad cambia a ese jugador. Algunos objetos también pueden cambiar de propiedad después de activarse si son atacados por un jugador.

Los objetos se pueden agrupar en varias categorías según sus atributos y efectos:

- Los **objetos contenedor** liberan otros objetos una vez que se rompen, lo que se puede hacer atacándolos o recogiéndolos y lanzándolos.
- Los **objetos espada** son empuñados por los personajes una vez recogidos, convirtiendo sus ataques en un golpe de objeto.
- Los **objetos tipo arma** disparan proyectiles cuando se usan. Todos tienen munición limitada y simplemente dispararán hasta que se agote.
- Los **objetos lanzables** solo se pueden lanzar a los oponentes. Sostener un objeto lanzable significa que un personaje no puede usar la mayoría de sus ataques, ya que al presionar el botón de ataque simplemente lanzará el objeto.
- Los **objetos explosivos** causan una explosión cuando se activan, luego desaparecen del campo. La mayoría de los objetos explosivos también son objetos lanzables, pero hay excepciones.
- Los **objetos de estado** aplican algún tipo de efecto al usuario, a menudo beneficioso pero no necesariamente.
- Los **objetos de recuperación** disminuyen el daño del usuario.
- Los **objetos de invocación** liberan un aliado temporal que lucha junto a quien activa el objeto.
- Los **objetos colecciónables** no afectan al partido, sino que se añaden a la colección del jugador una vez recogidos. Se pueden recoger incluso si el personaje ya está sosteniendo un objeto.

- **Mecánicas internas básicas**

- **Hitboxes**

Las *hitboxes*, o también conocidas como *burbujas de ataque*, es la estructura principal de cómo funcionan los ataques en la mayoría de los juegos de lucha. Estas *burbujas* suelen solaparse con el área de contacto del personaje que, aunque también entran dentro de la misma propiedad que las *hitboxes*, se les conoce como *hurtboxes* o *burbujas de daño*, siendo estas las colisiones del daño recibido por el ataque, las cuales suelen ser representadas en el cuerpo del personaje.

Las dos formas más comunes de *hitboxes* y *hurtboxes* son cubos y esferas, aunque ninguna de estas es mejor que la otra, los cubos pueden formar hitboxes más fácilmente, y las esferas son más fáciles de calcular en la detección de colisiones.

Todos los movimientos constan de múltiples *hitboxes* siempre, cada una teniendo diferentes valores, aunque siempre hay una que está y es la

más importante, llamada **sweet spot**, la cual es la *hitbox* que más efecto de *knockback* suele tener, dando así que es la más fuerte de todas ellas.

Si múltiples *hitboxes* de un solo movimiento se conectan con el oponente, solo una de ellas contará. Esto normalmente se hace de manera que la *hitbox* más cercana a la *hurtbox* del rival sea la que se ejecute y el resto se anulan.

Las *hitboxes* que están separadas de las *hurtboxes* de un personaje a menudo se les llama **disjoints**, lo cual suele ser una ventaja, ya que pueden conectarse desde una mayor distancia y al mismo tiempo mantener al personaje más seguro de recibir el golpe del rival.

Como se sabe este juego es en 3D, pese jugarse en una vista 2.5D, por ello las *hitboxes* y *hurtboxes* de este proyecto son también 3D. Lo curioso, es que pese a que por ello que tienen un comportamiento “inesperado” dentro de lo que suelen ser los juegos de lucha:

- Si la animación del personaje se inclina hacia los lados, pueden hacer que los ataques no entren en contacto a donde debería de haber habido contacto.
- Los ataques que giran en un arco horizontal pueden golpear a personajes, aunque en verdad no debería de haber recibido el golpe.
- **Tipos**

- **Hitbox**

- **Ofensivo:** son las *hitboxes* estándar, las cuales actúan como colisiones de ataque. Cada una tiene propiedades únicas, algunas como qué ángulo y potencia para el *knockback* tiene el movimiento realizado o estados especiales que se ejecutan al dañar al rival.
 - **Coger:** *hitboxes* que actúan para agarrar al rival, ignoran todo tipo de *hitboxes* de defensa y permiten efectuar la acción siempre y cuando entren en contacto con la *hurtbox* principal del rival.
 - **Escudo:** *hitbox* defensiva que actúa cuando se activa el escudo. Estas suelen ser inútiles ante agarres.

- **Hurtbox**

- **Dañable:** son las *hurtboxes* estándar, las cuales actúan como colisiones de daño. Estas colisiones suelen variar dependiendo del personaje y suelen estar activas casi siempre.
- **Invencible:** *hurtbox* la cual permite al personaje recibir los ataques, pero no recibe ningún tipo de daño o *knockback*.
- **Intangible:** *hurtbox* la cual el personaje no recibe ningún tipo de ataque que esté a su alcance. Normalmente se ejecuta este tipo de colisión cuando el personaje está haciendo una esquiva, un *tech* y un *edge grab*.

A veces el personaje dispone de algunos ataques que vuelven solamente algunas *hurtbox* de su cuerpo se vuelven *intangibles*.

- **Damage**

El **damage** o **daño** es la medida básica de **cuán vulnerable** es un personaje al **knockback** o **empuje** de los ataques. A bajos niveles de daño, el personaje no puede ser empujado muy lejos y solo puede ser lanzado por los ataques fuertes, estos infligen considerablemente más *knockback*, aumentando la probabilidad de que los personajes sean eliminados.

El **damage** se representa con un **porcentaje numérico** que comienza en 0% y puede aumentar hasta 999%. A pesar de este formato, llegar al 100% no significa que un personaje vaya a morir fácilmente. Este valor no es un valor único, cuando el personaje muera, este se **resetea** a 0% de nuevo.

Cada ataque infinge una **cantidad fija de daño**, que luego se modifica por cosas como la *negación de movimientos gastados* antes de que el objetivo sea lanzado. Un ataque que no infinge daño no hará que los objetivos retrocedan ni produzcan un sonido de golpe regular, aunque aún causará *knockback*. El daño que infinge un ataque es un factor significativo en cuánto *knockback* causa.

Los siguientes aspectos se ven afectados por el daño:

- **Knockback:** a medida que un personaje recibe más daño, vuela más lejos cuando es golpeado. Ciertos ataques tienen un empuje fijo y no aplican el daño a la fórmula del *knockback*.

- **Tiempo de agarre:** los personajes con un alto daño pueden ser agarrados durante períodos más largos.
- **Tiempo de estado:** generalmente, los personajes con más daño son vulnerables durante más tiempo a un efecto de estado. La excepción es ser aturrido por una **ruptura de escudo**: cuánto más daño tenga un personaje aturrido, más rápido podrá recuperarse.
- **Tiempo de nado:** los personajes con más daño tienen menos tiempo para nadar antes de entrar en pánico, con personajes con un 96% o más de daño teniendo el tiempo de nado más corto.
- **KOs instantáneos:** Muchos ataques que hacen KOs instantáneos no lograrán eliminar a un personaje con menos de una cierta cantidad de daño y en su lugar causarán daño y *knockback*.

- **Knockback**

En *Super Smash Bros.*, los ataques no simplemente infligen daño a un oponente o reducen su salud basándose en la fuerza del ataque infligido. El ***knockback* o empuje** está en el corazón de su esencia de juego, así que tenemos que hacer algunas consideraciones únicas.

En este juego, cuanto **más daño** infinge un ataque y cuanto **mayor sea el porcentaje** de daño acumulado por el objetivo, **mayor será el empuje**.

Por ello cabe destacar algunos parámetros en los cuales se componen para calcular el *knockback* final:

- Los **vectores** determinan la dirección del empuje. Esto existe con una direccionalidad más limitada que en algunos otros juegos de lucha, pero en *Super Smash Bros.*, se puede hacer que el empuje ocurra en cualquier dirección. Y al aplicar un modificador especial, se puede hacer que el empuje reaccione a la dirección en la que viaja el atacante. Con combos aéreos, por ejemplo, esto ayuda a que más golpes del atacante aterricen.

Además, el arco del empuje, incluso con todas las demás condiciones iguales, cambiará según valores como la velocidad de caída y el peso del luchador objetivo.

- La **escalabilidad del *knockback*** es un multiplicador que permite ajustar el empuje infligido cuando un ataque aterriza. Se puede hacer que los ataques fuertes causen sólo un ligero empuje, o que los ataques débiles alejen bastante a los luchadores.

- El **knockback base** es la cantidad mínima de empuje que un ataque infinge, sin importar qué. Por ejemplo, un oponente que no ha recibido mucho daño no será empujado mucho en absoluto por ataques destinados a lanzarlos hacia arriba, pero al menos debería haber alguna reacción, y este parámetro nos permite garantizar una cantidad mínima de empuje establecida.

Sin embargo, simplemente agregar este valor haría que los empujes fueran demasiado grandes, así que se usa la escalabilidad del empuje para mantener todo bajo control.

- El **knockback fijo** permite establecer un empuje exacto independientemente de otras condiciones. No importa cuánto daño haya recibido un luchador, siempre reacciona exactamente de la misma manera.

2.4. Movimientos de los personajes

Los personajes de esta saga son su misma esencia, cada uno con su propio estilo de lucha y una amplia variedad de movimientos distintivos. Con ello, existen diferentes tipos de movimientos que se pueden ejecutar, los cuales se clasifican en:

- **Movimiento lateral:**

- **Andar:** es una forma de movimiento terrestre más lenta que correr, la cual se realiza moviendo el *stick izquierdo* levemente hacia el lado mientras se está en el suelo. En *Smash*, es posible caminar inclinando el *stick* diagonalmente hacia arriba en la dirección en la que los jugadores desean caminar.

Los personajes tienen tres velocidades de caminar diferentes según lo lejos que se incline el joystick, siendo la más lenta de estas animaciones a menudo una especie de andar de puntillas. Sin embargo, en *Super Smash Bros. Brawl*, si un personaje es controlado con un mando de *Wii* de lado, siempre caminarán a su velocidad máxima.

- **Correr:** es una forma de movimiento terrestre más rápida que caminar. Se realiza tocando hacia los lados en el *stick izquierdo*. El personaje del jugador entra en su animación de *dash* inicial, en la que cambian de dirección con muy poco retraso, y luego proceden a su animación de correr.

En *Super Smash Bros. Brawl*, un personaje que intenta hacer un dash, al azar, puede tropezar en su lugar, a este hecho se le conoce como un *tripping*. En *Super Smash Bros. 4* y *Super Smash Bros. Ultimate*, hacer un dash desde una plataforma hará que el personaje realice una corta animación de salto hacia adelante, y al aterrizar en otra plataforma mientras se mantiene presionado el *stick* en esa dirección le permitirá continuar el dash.

Después de realizar un *dash*, el jugador puede ejecutar un **pivote**, el cual le permite volver a la posición neutral. Para ello, el jugador tiene que mover el mismo *stick* en la dirección opuesta y dejarlo en pose neutral.

- **Gatear:** es una técnica introducida en *Super Smash Bros. Brawl*, la cual solo puede realizar un pequeño grupo de personajes, mientras el jugador esté agachado, puede moverse lateralmente, lo que le permite evitar algunos ataques o proyectiles.

La utilidad del arrastre depende principalmente de cuán bajo sea el agachado del personaje. Por ejemplo, *Snake*, de la saga de juegos *Metal Gear Solid*, se agacha y gatea de manera muy baja, haciendo que

reduzca significativamente su altura, lo que hace que sea efectivo para esquivar proyectiles y ataques.

- **Agacharse:** es una posición en el suelo que se logra manteniendo presionado hacia abajo en el *stick izquierdo* cuando se está de pie en el suelo. Esto acorta la zona de golpe del personaje, lo que le permite esquivar algunos ataques y proyectiles.
- **Salto:** es una acción que mueve a un personaje desde el suelo al aire. Todos los personajes también pueden saltar por segunda vez en el aire, siempre que no estén atacando, esquivando el aire, indefensos, aturdidos o ya lo hayan usado. Se realiza moviendo hacia arriba el *stick izquierdo*, o presionando el botón de salto, que es por defecto.

Por otro lado, existen dos tipos de salto que se pueden efectuar cuando se está en el suelo, el primer tipo y el más común es el salto completo o *full hop*, en el que el jugador mantiene presionado el botón de salto para que el luchador pueda saltar a la altura máxima. El segundo tipo es el salto corto o el *short hop*, que implica que el jugador toque rápidamente el botón de salto para que el luchador solo realice un ligero salto, casi sin altura alguna.

Algunos personajes pueden saltar más de una vez en el aire, los cuales les permite tener más maniobrabilidad aérea y poder realizar más combos.

Super Smash Bros. Brawl agregó una mecánica de salto conocida como *footstools*. Esto permite a los personajes saltar de las cabezas de otros personajes. Se puede hacer incluso cuando ya se hayan utilizado todos los saltos en el aire.

- **Ataques:** movimientos que se realizan para hacer daño al rival y que salga disparados hacia la *blast zone*, o zona de límite, y así conseguir un *KO*.

Existen muchos tipos de ataques diferentes para *Smash Bros.*, de los cuales se pueden mencionar:

- **Ataques normales:** movimiento que para realizarlo hay que pulsar el botón de ataque. Estos ataques normalmente son rápidos y se pueden ejecutar tanto en el suelo, el cual se le llama un **neutral attack**, o también llamado *jab*, como en el aire, el cual se le llama un **neutral aerial**. Son ataques usados para sacar al rival de encima de una manera rápida y efectiva.

Existe otro tipo de ataque solamente realizado cuando el jugador corre en el suelo, llamado **dash attack**. Este tipo de ataques suele ser rápido y su principal opción es de acercamiento al rival, pero suelen tardar mucho en terminarse y por ello no permite que el jugador pueda realizar otra acción hasta que termine.

- **Ataques fuertes:** movimiento que para realizarlo hay que inclinar la palanca de control en alguna dirección antes de pulsar el botón de ataque, de esta manera efectuando un ataque más fuerte orientado hacia esa dirección. Además estos suelen tener más alcance que el normal.

Como los ataques normales, estos tienen dos tipos y cada uno dispone de un nombre y función:

■ **Terrenales:**

- **Forward tilt:** para realizarlo hay que sostener el *stick izquierdo* de manera horizontal y presionar el botón de ataque. Este tipo de movimiento suelen ser golpes simples, rápidos y débiles hacia afuera para el combate a corta distancia.

Algunos de este tipo pueden ser angulados hacia arriba o abajo, teniendo que ser más fuertes cuando se inclinan hacia arriba y más débiles cuando se inclinan hacia abajo.

- **Down tilt:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia abajo y presionar el botón de ataque. Este tipo de movimiento tienden a ser movimientos muy rápidos con buen alcance, lo que los hace útiles para crear espacio.

Algunos de estos suelen estar diseñados para golpear a los oponentes que están en el suelo, además de estar diseñados como iniciadores de *combos* o ataque de KO, debido al ángulo que tienen, que no permite a los rivales regresar al escenario.

- **Up tilt:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia arriba y presionar el botón de ataque. Puede ser difícil de ejecutar rápidamente ya que al mover el *stick izquierdo* hacia arriba puede saltar si se realiza el movimiento demasiado rápido.

Normalmente estos golpes se usan para realizar *malabares*, dicho de otra manera, para que el rival se mantenga en el aire desprotegido mientras que el jugador desde el suelo le sigue golpeando sin preocuparse.

Este tipo de movimiento puede caer en dos categorías: golpes rápidos con un rango estrecho directamente sobre

la cabeza, golpes más lentos que cubren un arco amplio sobre el personaje.

■ Aéreos:

- **Forward aerial:** para realizarlo hay que sostener el *stick izquierdo* de manera horizontal hacia la dirección que se está mirando y presionar el botón de ataque. Estos ataques suelen ser débiles, pero rápidos y tienen mucho alcance.
- **Back aerial:** para realizarlo hay que sostener el *stick izquierdo* de manera horizontal hacia la dirección contraria que se está mirando y presionar el botón de ataque. La mayoría de veces suele ser una patada o un combo de ellas, y tienden a ser considerablemente más fuertes y/o más rápidos que otros ataques aéreos, además de disponer de un buen alcance.

No solamente son buenos ataques fuertes y rápidos, sino que también son buenas opciones defensivas, ya que pueden atrapar a un oponente que intenta atacar por detrás.

- **Down aerial:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia abajo y presionar el botón de ataque, mientras está en el aire. Muchos de estos movimientos son *golpes meteoro*s, golpes diseñados principalmente para noquear o debilitar a los oponentes que intentan recuperarse del ataque anterior.

Típicamente son ataques de uno o varios golpes realizados en la parte baja del personaje o movimientos donde el personaje se suspende brevemente en el aire mientras ataca y luego sigue cayendo.

- **Up aerial:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia arriba y presionar el botón de ataque, mientras está en el aire. Muchos de estos ataques tienen una amplia cobertura sobre el usuario, como patadas en el aire, o pueden ser golpes *estrechos* pero poderosos. También tienden a lanzar al oponente en un ángulo ascendente, lo que los hace cruciales para el *malabarismo* del enemigo.

- **Ataques Smash:** movimiento realizado al mover el *stick izquierdo* de manera agresiva hacia cualquier dirección mientras se pulsa a la vez el

botón de ataque mientras se está en el suelo. Son ataques extremadamente poderosos que pueden infligir un gran daño y enviar a los oponentes volando por los aires.

Estos ataques, a diferencia del resto, tienen un mayor *knockback* en comparación al resto de ataques de los personajes. Además estos ataques se pueden *cargar* durante unos segundos para que el daño sea mayor.

Estos ataques se pueden ejecutar en cuatro direcciones:

- **Forward Smash:** para realizarlo hay que sostener el *stick izquierdo* de manera horizontal y presionar el botón de ataque a la misma vez. Este tipo de ataques son *golpes fuertes* con una gran *hitbox* en frente del personaje.

Algunos de estos ataques contienen una *super armor*, lo que le permite aguantar algunos ataques de los rivales. Además de ello, algunos de estos ataques se pueden angular ligeramente hacia arriba o hacia abajo.

- **Down Smash:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia abajo y presionar el botón de ataque a la misma vez. Este tipo de ataques son más rápidos y débiles que el resto de ataques *Smash*, y golpean a ambos lados del personaje, haciendo que sean realmente útiles para cubrir un espacio general por si algún rival se te acerca o no tienes manera de huir de él.
- **Up Smash:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia arriba y presionar el botón de ataque a la misma vez. Estos ataques suelen cubrir la parte superior del personaje, además de ser un gran ataque para hacer un KO en la mayoría de los escenarios del juego.

- **Movimientos especiales:** estos son los movimientos que definen la *personalidad* de cada personaje, las cuales se pueden realizar hasta cuatro movimientos diferentes con solo presionar el botón de *especial* y diferentes inclinaciones en el *stick izquierdo*.

Como sugiere el nombre, estos movimientos son ataques únicos que pueden tener funciones diferentes, en muchos casos proporcionando ventajas tácticas en comparación con los ataques normales y causan más daño que los ataques *Smash*, aunque esto difiere entre cada movimiento especial.

Estos movimientos se suelen clasificar en:

- **Neutral Special:** movimiento que para realizarlo hay que pulsar el botón *especial*. La mayoría de los especiales neutrales son un proyectil o ataque de un solo golpe, estos pueden ser cargados, como el disparo cargado de *Samus*, personaje de la saga *Metroid*, o puede ser un ataque rápido, como la *bola de fuego* de *Mario*. Estos movimientos tienden a no mover al personaje, aunque algunos obligan al personaje a moverse o le dan la opción de hacerlo.
- **Side Special:** para realizarlo hay que sostener el *stick izquierdo* de manera horizontal y presionar el botón *especial*. Muchos movimientos de este tipo suelen atribuir un movimiento lateral extra del personaje, por ello, muchos de estos pueden ser usados para volver al escenario y así no morir.

Aun así, algunos tienen diferentes *propiedades* dependiendo de cómo se activa o *continua*, ya que algunos al mover el *stick izquierdo* rápidamente hacia el lado puede activar un golpe más potente o permiten variar el ángulo de ataque haciendo que se transforme en un ataque completamente diferente, por ejemplo, el *Dancing Blade* de *Marth*, personaje de la saga de juegos *Fire Emblem*, el cual se debe de pulsar el botón *especial* conjuntamente con un ángulo diferente tanto hacia arriba como hacia abajo para tener variantes del mismo *combo* de especial, cada uno con un ataque final de dicho especial diferente.

- **Down Special:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia abajo y presionar el botón *especial*. Este tipo de movimientos suelen ser los más variados de todos los movimientos especiales y a menudo son los ataques más “llamativos” del personaje.

Estos pueden llegar a ser: movimientos que reflejan proyectiles, movimiento que absorben proyectiles como fuego, rayos o agua, movimientos que crean objetos, transformaciones de personajes, ataques cargados o movimientos increíblemente únicos, como puede ser el especial de *Hero*, personaje de la saga de juegos *Dragon Quest*, el cual saca un menú de cuatro opciones elegidas al azar que le permiten hacer todo tipo de movimientos, desde cambios de estado hacia él o los rivales, como movimientos tipo proyectil o ataques cuerpo a cuerpo devastadores.

- **Up Special:** para realizarlo el jugador debe de sostener el *stick izquierdo* hacia arriba y presionar el botón *especial*. Estos movimientos suelen tener la función principal de que el personaje se pueda recuperar y volver al escenario, aunque al usarlos, los personajes no puedan realizar ningún movimiento más hasta que vuelvan a tocar el suelo o sean golpeados.

- **Defensa:** es una burbuja de energía que rodea al usuario para poder protegerse de la mayoría de los ataques, más concretamente para todos aquellos golpes que no sean agarres o *imbloqueables*. Para realizarlo el jugador simplemente debe mantener presionado el botón de *escudo*.

Aún así, el escudo solo puede activarse en el suelo y no se puede usar indefinidamente, ya que se reduce con el tiempo y con cada golpe recibido, cuanto más fuerte sea dicho golpe, más se reducirá el tamaño del escudo. Con el tiempo, el escudo se puede acabar rompiendo, lo que produce que el personaje entre en estado de **aturdimiento**, cosa que provoca que sea vulnerable a cualquier golpe que reciba.

Igualmente, no es la única manera defenderse mientras se tiene activo el **escudo**:

- **Esquiva en el sitio/Spot dodge:** es una maniobra de evasión en el sitio, que se ejecuta pulsando el *stick izquierdo* hacia abajo, haciendo que el personaje sea *intangible* durante unos frames. Suele ser realmente útil para esquivar proyectiles, objetos que sean lanzados y evitar agarres, ya que resulta ser el esquive más corto y rápido de todos los existentes.
- **Voltereta/Roll:** es una maniobra de evasión que se ejecuta pulsando el *stick izquierdo* hacia cualquiera de los ejes horizontales. Al realizar esto los jugadores darán una voltereta hacia delante o hacia atrás obteniendo *intangibilidad* durante unos frames. Después de rodar, los personajes siempre terminarán mirando en la dirección de la que vinieron.

Un personaje con una voltereta rápida y con larga distancia es bueno, ya que son más difíciles de *castigar*, a diferencia de las más lentas y con corta distancia.

Desde *Super Smash Bros. Ultimate*, al rodar varias veces seguidas en un corto periodo de tiempo, hará que cada voltereta que se intente realizar será más lenta y se tendrá menos *intangibilidad*, teniendo así que sea más fácil de golpear al personaje.

- **Esquiva aérea/Air dodge:** es una maniobra de evasión, introducida a partir de *Super Smash Bros. Melee*, que se puede ejecutar pulsando el botón de escudo cuando el personaje está en el aire. Al realizarlo, otorga al jugador unos *frames* de *intangibilidad*.

Esta esquiva ha cambiado durante los años, llegando a tener dos variantes parecidas entre ellas:

- **Esquiva aérea estática:** esta variante introducida en *Super Smash Bros. Brawl* y continuada en *Super Smash Bros. for 3DS / Wii U* es

una esquiva que solo permite esquivar en el lugar en el cual se está, manteniendo la gravedad y *dirección* en la cual el personaje está bajando. Después de esta acción los jugadores pueden atacar, saltar o esquivar de nuevo.

- **Esquiva aérea angulada:** esta variante introducida en *Super Smash Bros. Melee* y continuada en *Super Smash Bros. Ultimate* es una esquiva que permite esquivar en cualquier dirección del mando, forzando al personaje a tomar una distancia dada por defecto en la dirección que se le haya otorgado. Esta acción para momentáneamente la caída del jugador, haciendo que después de esto continúe cayendo.

Si a esta acción se realiza junto al suelo, el personaje realizará una mecánica que te permite deslizarte por el suelo y recorrer bastante distancia con ella además de cancelar cualquier animación que debiera hacer, lo cual le permite realizar otra acción al mismo momento de tocar el suelo. Esta mecánica fue nombrada por el desarrollador como “*backdash*”, aunque no le dio importancia ya que “no le veía uso alguno”, la comunidad pronto la empezó a utilizar en torneos y fue llamado **wavedash**.

Esta mecánica tuvo mucho uso en *Melee*, ya que los personajes tenían poca tracción y podían moverse más rápidamente. Pero años después, en el lanzamiento de *Ultimate*, pese que es cierto que se puede realizar, se encontró que ya no hay tanta tracción en el juego y los personajes realizan la animación de aterrizaje, convirtiéndolos más lentos en comparación a su predecesor, haciendo que esta mecánica se use muy pocas veces en este juego.

- **Agarre y golpe rápido:** un **agarre** es un ataque que hace mantener a un oponente en frente del enemigo, con este agarrandolo, en preparación de lanzarlo o atacar. Estos son *imbloqueables* y, por lo tanto, son una forma fácil de abrir un hueco en la defensa del rival. Para realizarlo hay que estar delante del rival y presionar el botón de agarre.

El agarre se puede realizar mientras el jugador está **estacionario en el lugar**, el cual tiene un inicio más rápido pero más corto, **está corriendo**, el cual tiene un alcance más largo pero es más lento de realizar, o **agarre después de un giro**, el cual tiene el mayor alcance de todos pero tarda más en dejar al jugador realizar otro movimiento.

Existen algunos agarres que usan cables o ganchos, los cuales se pueden agarrar al escenario y ser usados como otra clase de recuperación, estos se llaman **tether recovery**.

Mientras se tiene agarrado al rival, el jugador puede realizar un **golpe rápido** o **pummel**, llamado en inglés. Este es un ataque que se puede realizar con cualquier botón de ataque mientras el rival está agarrado. Este ataque es extremadamente flojo, pero se puede usar repetidamente a una velocidad bastante rápida hasta que el oponente se libere o sea lanzado.

- **Lanzamientos:** es un movimiento que se puede realizar después de agarrar a un rival, estos se efectúan con el *stick izquierdo*, en cual dirección. Los rivales no pueden escapar de los lanzamientos, además estos otorgan un breve período de *invencibilidad* al inicio de hacer la acción de lanzar. Este tipo de movimiento no usa *hitboxes* normales al soltar a los rivales, en su lugar, usa un tipo de *hitbox especial* el cual solo afecta al objetivo del lanzamiento.

Los lanzamientos disponen de cuatro variantes de ellas mismas:

- **Forward throw:** para realizarlo el jugador debe de mover el *stick izquierdo* hacia la dirección que está mirando. Estos suelen usarse para lanzar al rival lejos, y algunos pueden usarse para comenzar combos.
- **Back throw:** para realizarlo el jugador debe de mover el *stick izquierdo* hacia sus espaldas. Estos raramente son buenos para empezar combos, aunque tienden a ser buenos para sacar al rival del escenario o para realizar *KO*.
- **Down throw:** para realizarlo el jugador debe de mover el *stick izquierdo* hacia abajo. Generalmente, muestran al usuario golpeando el suelo, haciéndolo rebotar en ángulos verticales, haciendo que tenga poco *knockback* y así permitiendo que el jugador pueda iniciar combos de manera sencilla.
- **Up throw:** para realizarlo el jugador debe de mover el *stick izquierdo* hacia arriba. Debido al increíble *knockback* que tienen, son usualmente usados para realizar *KO*, o incluso, en porcentajes bajos, suele ser usado para empezar combos aéreos.
- **Burla:** este movimiento está diseñado para provocar al oponente. Este movimiento se puede realizar hasta con tres botones diferentes, ya que existen tres variantes de esta misma, cada una diferente de la otra.

La mayoría de las burlas no tienen otra función que la ya nombrada anteriormente, pero existen algunas excepciones. Estas excepciones pueden

disponer de una *hitbox*, la cual dañará al rival si está cerca del jugador, aunque no le realizará casi daño.

- **Tech:** esta mecánica *pasiva* de los juegos es una acción que se puede realizar cuando el jugador es golpeado y cae en el suelo, pared o techo, la cual le permite anular la mayor parte de *frames* de recuperación y evitar así que le puedan realizar un combo. Para realizarlo el jugador debe de pulsar el botón de escudo poco antes de tocar la superficie.

Con ello cabe destacar que existen diferentes tipos de *tech*:

- **Standard tech:** es el *tech* neutral el cual se realiza al presionar el botón de escudo. El personaje se quedará en el sitio realizando el *tech* sin ninguna peculiaridad extra.
- **Rolling tech:** este *tech* se puede realizar al realizar un *Standard tech* en combinación de un movimiento del *stick izquierdo* en horizontal, el cual le permitirá rodar instantáneamente después de tocar el suelo. Esto permite a los personajes alejarse del punto de impacto original y crear espacio entre él y el rival.
- **Wall tech:** este *tech* se realiza cuando el personaje se estrella en la pared, siguiendo las mismas bases que el *Standard tech*, este se quedará en el sitio y seguidamente caerá.

Este *tech* dispone de una variante que solo tienen pocos personajes que es un **Wall jump tech**, el cual les permite hacer un salto desde la pared sin necesidad de parar en el sitio y luego realizar otra acción.

- **Ceiling tech:** este *tech* se realiza cuando el personaje se estrella en el techo, siguiendo las mismas bases que el *Standard tech*, este se quedará en el sitio y seguidamente caerá.

2.5. Controles de mando

En la saga *Smash Bros.*, ha deleitado a incontables jugadores a lo largo de los años, la precisión y la respuesta táctil de los controles son aspectos fundamentales para disfrutar de una experiencia de juego inmersiva y satisfactoria.

Desde su lanzamiento inicial, los jugadores han expresado una preferencia destacada por utilizar el mando de GameCube, venerado por su **diseño ergonómico** y su **respuesta rápida**, lo que lo convierte en el periférico preferido para muchas personas en el mundo competitivo.

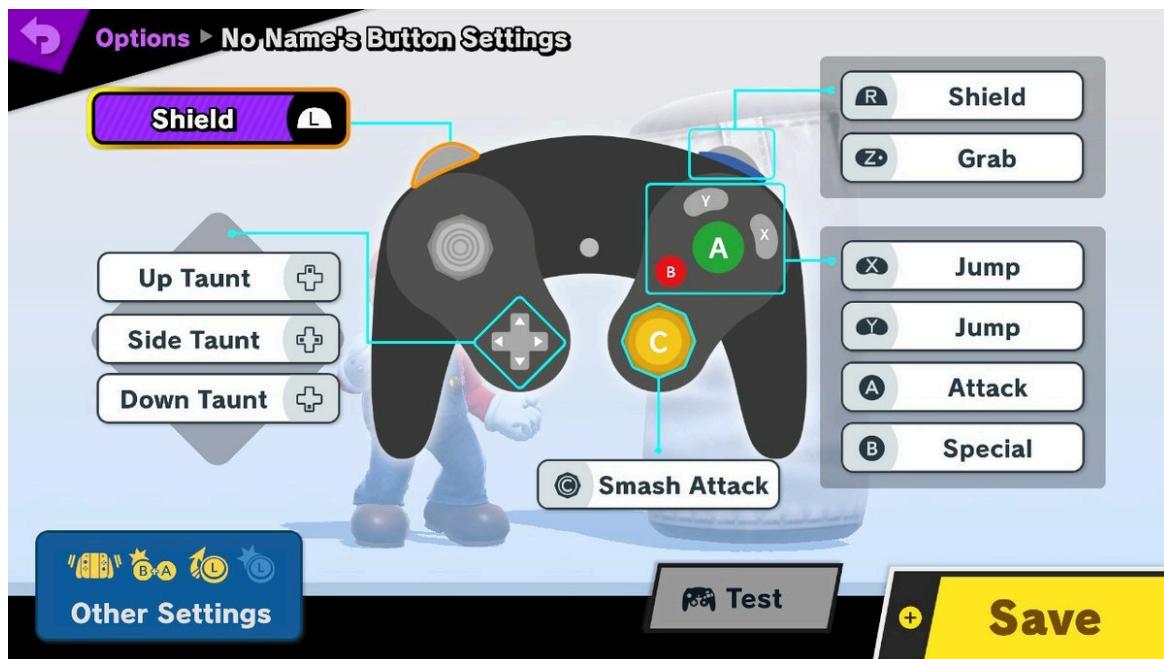
- **Controles Principales**

- **Stick Analógico Izquierdo:** este componente, conocido por su sensibilidad y durabilidad, permite al jugador moverse con precisión por el escenario, realizando movimientos fluidos y cambios de dirección con facilidad. Desde desplazamientos suaves hasta maniobras evasivas rápidas, es la herramienta principal para controlar el movimiento del personaje.
- **Stick Analógico Derecho:** proporciona al jugador la capacidad realizar ataques fuertes al instante para poder atacar rápidamente al rival.
- **Botones de Dirección (D-Pad):** estos botones, dispuestos de manera intuitiva para facilitar su acceso, permiten al jugador realizar una variedad de burlas o *taunts*, en la jerga inglesa, que pueden ser tanto simples animaciones como tener mecánicas *intrínsecas* solo conocidas por los jugadores de aquellos personajes.
- **Botones de Acción (A, B, X, Y):** estos botones activan una amplia gama de acciones, desde ataques básicos hasta poderosos movimientos especiales y saltos estratégicos, que son cruciales para el éxito en el campo de batalla.

En el caso de los botones de salto, que son la X y la Y, dependiendo de la cantidad de tiempo que los mantengan pulsados, hacen variar entre dos tipos de saltos, llamados por la comunidad *Short hop* o salto corto, un salto que no llega a mucha altura, y *Full hop* o salto, es el salto completo y llega a la altura máxima.

- **Gatillos (L, R):** al presionar estos botones, ofrecen una versatilidad excepcional en combate, permitiendo al jugador usar el escudo, así pudiendo defenderse de los ataques enemigos y, en combinación al *stick analógico izquierdo*, ejecutar maniobras ofensivas con precisión y determinación.

- **Botones Z:** este botón activa la función de *agarre* para poder sujetar al rival y darle una serie de golpes o lanzarlo hacia cualquier dirección que se quiera.
- **Botón Start:** situado en el medio del mando, este botón ofrece al jugador la posibilidad de pausar el juego en cualquier momento y acceder al menú de pausa, donde se puede seleccionar para activar la cámara de fotos, resetear la partida, mirar la lista de movimientos del personaje, conectar mandos o salir de la partida.



Button Settings [UI]

Fuente: https://www.youtube.com/watch?v=8nfMt4oG6hq&list=PLgKCjZ2WsVLTvcEHQFioJF263lp_yoxen

- **Configuración del mando**

Sakurai cree que todos los juegos, sin importar lo simples que sean, deberían permitirte **cambiar las asignaciones de botones** a través de una configuración de botones.

Las pantallas de configuración de botones son una **característica bastante pesada** en diseño. Pedirle a cada desarrollador que cree la suya propia conduce a **mucho más trabajo durante el desarrollo**. Por lo tanto, según su opinión, sería genial si las consolas mismas permitieran personalizar los controles de forma individual para cada juego, pero eso realmente no ha sucedido aún.

Para los juegos que ha dirigido, el equipo de desarrollo ha tenido que admitir muchos mandos diferentes desde que llegó el Wii, algunos de los cuales son el *Wiimote de lado*, *Wiimote con Nunchuck*, el *Wii Classic* y *Pro Controller*, en el caso de la Wii y Wii U, el *Gamepad de Wii U*, la *Nintendo 3DS*, el *mando Pro de*

Wii U, en el caso de la *Wii U*, los *Joy-Cons* juntos, los *Joy-Cons* de lado, el *mando Pro Controller* de *Nintendo Switch*, en el caso de la *Nintendo Switch* y por último el mando de *Gamecube*, el cual es compatible con todas las plataformas. Y *Smash Bros.* está destinado a jugarse juntos, así que necesitaban una forma para que los jugadores invitados puedan configurar fácilmente sus controles.

Con todo eso en mente, aquí hay algunas consideraciones que tuvieron que hacer durante el desarrollo:

- **Los controladores se representan todos gráficamente.** Muchos juegos configuran sus configuraciones de teclas como una lista vertical, que ciertamente es más fácil de hacer, pero **puede ser difícil para el jugador llevar un seguimiento de qué botón hace qué**.
- **Seleccionar una acción para asignar ese botón.** Algunos juegos te obligan a presionar el propio botón para asignarlo a una acción, pero optaron por no hacerlo e hicieron que usaras el cursor para elegir un botón y luego la acción deseada.
- **Permitir que varios botones compartieran la misma acción.** Por ejemplo, hay dos botones de salto en *Smash Bros.*, puedes asignar uno a una acción diferente si lo deseas.
- **Configuraciones de botones se pueden guardar en un nombre de perfil.** Al seleccionar ese nombre de perfil antes de una pelea, **cambiarás a esos controles preestablecidos**, lo que puede ayudar si tienes muchos jugadores invitados. Cuando sea posible, incluso han permitido que los jugadores **almacenen configuraciones de botones en el propio mando** para llevarlo consigo mismo.

2.6. Diseño de escenarios

Los escenarios de *Super Smash Bros.* son una amalgama de **arte y funcionalidad**, donde cada detalle es cuidadosamente diseñado para ofrecer una experiencia visualmente atractiva y al mismo tiempo garantizar una jugabilidad fluida y accesible. En este juego, los escenarios no son simplemente fondos estáticos, sino que desempeñan un **papel activo en el desarrollo de las peleas**, proporcionando **plataformas, obstáculos y elementos interactivos** que influyen en el curso de la acción.

Para comprender cómo se crean estos escenarios, es necesario analizar tanto los aspectos artísticos como los aspectos técnicos involucrados en su diseño. En primer lugar, los desarrolladores se centran en la **estética del escenario**, buscando crear un entorno visualmente atractivo que sea **coherente con el estilo y la temática del juego**. Esto implica seleccionar cuidadosamente los elementos de diseño, como la **paleta de colores, la iluminación y los efectos visuales**, para crear una atmósfera que sea tanto inmersiva como emocionante para el jugador.

Sin embargo, la estética no es el único factor a considerar al diseñar escenarios. También es crucial tener en cuenta la **jugabilidad y la funcionalidad** del entorno de juego. En *Super Smash Bros.*, caerse del escenario puede resultar en una penalización para el jugador, por lo que es fundamental diseñar escenarios que permitan **una fácil navegación** y una comprensión clara de los **límites del terreno**. Esto se logra dividiendo los escenarios en capas según la **profundidad**, lo que facilita a los jugadores determinar dónde pueden moverse y qué áreas deben evitar.

Además, se presta especial atención a la **visibilidad del escenario**, asegurándose de que los jugadores puedan **distinguir claramente los elementos importantes**, como las plataformas y los bordes del escenario, incluso en medio del caos de la batalla. Para lograr esto, se utilizan técnicas como el diseño de niveles en tres capas de profundidad: "**vista lejana**", "**vista intermedia**" y "**vista cercana**", lo que permite resaltar los puntos de apoyo y garantizar una navegación intuitiva.



Los escenarios se crean como “arte” y “juego”

Fuente: https://www-4gamer-net.translate.gooq/games/412/G041234/20190906163/?x_tr_sl=ja&x_tr_tl=es&x_tr_hl=es&x_tr_pto=wapp

Otro aspecto importante del diseño de escenarios es la **coherencia** y la **integración temática**. Los escenarios se basan en diferentes universos de juegos, películas y otros medios, por lo que es crucial mantener la cohesión estilística y narrativa dentro de cada escenario. Esto se logra mediante la selección de objetos, elementos y ambientaciones que se ajusten al mundo del juego al que pertenecen, creando así una experiencia inmersiva y auténtica para el jugador.

Además de la estética y la funcionalidad, también se tienen en cuenta consideraciones técnicas al diseñar escenarios. Los desarrolladores deben **optimizar el rendimiento del juego** para garantizar una experiencia de **juego fluida y sin problemas** en una variedad de plataformas y dispositivos. Esto puede implicar la reducción de la complejidad geométrica de los escenarios, la optimización de texturas y efectos visuales, y la implementación de técnicas de renderizado eficientes.

- **Partes de un escenario**

Como ya se ha nombrado anteriormente, los escenarios de *Super Smash Bros.* se componen de elementos fundamentales que convergen para crear una experiencia de juego envolvente y dinámica. Estas partes, meticulosamente diseñadas, combinan tanto aspectos artísticos como funcionales para brindar un entorno que no solo sea estéticamente atractivo, sino también profundamente interactivo y desafiante para los jugadores.

Con esto, se puede identificar distintos componentes que contribuyen al conjunto armonioso de cada escenario, cada uno cumpliendo un papel esencial:

○ **Plataforma:**

- **Plataforma principal:** Es la zona central y más prominente en el escenario, donde la mayor parte de la acción de juego tiene lugar. Es la superficie principal sobre la cual los personajes luchan y se desplazan durante la partida.
- **Plataformas estáticas:** Son plataformas fijas y sin movimiento alguno dentro del escenario, las cuales se pueden atravesar tanto desde abajo como desde arriba. Proporcionan puntos de apoyo adicionales para los personajes y afectan la dinámica de la lucha al crear diferentes alturas y niveles de acceso en el campo de batalla.
- **Plataformas móviles:** Estas son plataformas dentro del escenario que tienen la capacidad de desplazarse o cambiar de posición durante la partida. A menudo, estas plataformas móviles añaden un elemento de imprevisibilidad y estrategia, ya que los jugadores deben adaptarse a su movimiento para mantenerse seguros o ganar ventaja sobre sus oponentes.

○ **Elementos de fondo y ambientación:**

- **Elementos interactivos:** Estos suelen ser diferentes tipos de *objetos* o *características* del escenario que pueden ser activados o utilizados por los jugadores durante la partida para influir en la dinámica del juego.

Estos elementos pueden incluir *interruptores*, *trampas*, *power-ups* o propios elementos del mapa como el propio agua.

Algunos de los cuales pueden ser como los *Flying Man*, un personaje, que aparece a veces en un escenario específico, que al activarlo se une a tu equipo y empieza a pegar a los rivales, o en otro escenario, donde tienes frutas que van apareciendo de unos árboles con el paso del tiempo y el jugador puede cogerlos para curarse o para lanzarlo al rival.

- **Elementos no-interactivos:** Estas son características del escenario que no cambian ni se activan durante la partida.

Estos pueden incluir objetos decorativos, estructuras o detalles temáticos que añaden profundidad visual al escenario pero no afectan directamente al juego. Algun ejemplo de estos sería personajes en el fondo animando a los luchadores en el escenario *Smashville*, mapa inspirado en *Animal Crossing*.

○ **Límites del escenario o *blast zone*:**

- **Límites estáticos:** Son los bordes del escenario que delimitan el área de juego y evitan que los personajes se salgan de la pantalla. Estos límites están fijos y no cambian durante la partida.
- **Límites especiales:** Son características del escenario que pueden influir en la jugabilidad de manera única. Pueden incluir elementos como paredes rompibles, como en el caso de *Mishima Dojo*, mapa inspirado en *Tekken*, áreas peligrosas fuera de la plataforma principal o zonas que causan daño o efectos especiales a los personajes que las atraviesan. Estos límites añaden una capa adicional de estrategia y riesgo-recompensa al juego.

2.7. Recepción

Super Smash Bros. ha sido aclamado como un enfoque fresco y emocionante dentro del género de los juegos de lucha. Aunque no se ajusta exactamente a la plantilla de un juego de lucha tradicional, ha logrado capturar la atención de los jugadores con su enfoque único y su mezcla de elementos de *plataformas* y combate. La prensa describe cómo el juego se aparta de la norma al no depender de complejas combinaciones de botones o una gran cantidad de movimientos para cobrar vida. En lugar de eso, ofrece una experiencia de juego accesible y divertida que se basa en la habilidad de expulsar a los oponentes de la plataforma para obtener puntos.

Una de las principales **fortalezas** del juego es la **inclusión de una amplia variedad de personajes** emblemáticos de Nintendo, cada uno con sus propias habilidades y movimientos característicos. Desde *Mario* y *Link*, progratognistas de las saga *Super Mario* y *The Legend of Zelda*, hasta *Samus*, protagonista de la saga *Metroid*, y *Pikachu*, mascota de las saga *Pokémon*, los jugadores tienen la oportunidad de controlar a sus personajes favoritos mientras luchan en escenarios inspirados en sus respectivas franquicias. Este enfoque ha sido elogiado por su **originalidad** y su capacidad para **evocar nostalgia** en los jugadores.

El **modo multijugador** es el punto culminante del juego, ofreciendo **frenéticas batallas** de hasta **cuatro jugadores** que son adictivas y emocionantes. Los combates son **rápidos y satisfactorios**, lo que convierte a *Super Smash Bros.* en una opción popular para las reuniones con amigos.

2.8. Comparación

Seguidamente del análisis del desarrollo y diseño detrás de la saga Super Smash Bros., a continuación se analizarán varios juegos que comparten similitudes notables con esta franquicia. A través de esta comparación, se observará la esencia de la lucha entre personajes mientras agregan su propio giro único al género.

Juego	Plataformas	Mandos compatibles	Estilo artístico	Nº de personajes	Personalización de personajes	Nº de ventas/descargas
Super Smash Bros. Ultimate	Nintendo Switch	Joy-cons, Nintendo Switch Pro Controller, GameCube Controller, Nintendo 64 Controller	3D, Recreación del contenido original	89	Los personajes disponen de trajes que pueden ser colores alternativos o trajes nuevos, usados alguna vez en sus sagas. Existe un personaje el cual se puede crear desde cero con unos movimientos a gusto del usuario, se dispone hasta 3 perfiles de personajes (Luchador, Espadachín, Tirador) con diferentes trajes de diferentes personajes.	33,67 millones de unidades
Rivals of Aether	PC, Nintendo Switch	Xbox One Controller, PS4/PS5 Controller, Arcade Stick, Joy-cons, Nintendo Switch Pro Controller, GameCube Controller, Nintendo 64 Controller	Pixel Art, Contenido original	18	Los personajes disponen de trajes que pueden ser colores alternativos o trajes nuevos, usados alguna vez en sus juegos. Existe la posibilidad de usar un traje hecho por la comunidad o un personaje nuevo.	1,2 millones de unidades

Nickelodeon All-Star Brawl 2	PlayStation, PC, Nintendo Switch, Xbox One	Xbox One Controller, PS4/PS5 Controller, Arcade Stick, Nintendo Switch Pro Controller, GameCube Controller	3D, Recreación del contenido original	29	Los personajes disponen de trajes que pueden ser colores alternativos o trajes nuevos, usados alguna vez en sus series animadas.	41.370 unidades
Multiversus	PlayStation, PC, Xbox One	Xbox One Controller, PS4/PS5 Controller, Arcade Stick, Nintendo Switch Pro Controller, GameCube Controller	3D, algunos personajes igual al contenido original, otros son recreaciones <i>cartoon</i> de dicho personaje	24	Los personajes disponen de trajes que pueden ser colores alternativos o trajes nuevos, usados alguna vez en sus series animadas.	2,8 millones de unidades
Brawlhalla	PlayStation, Xbox One, Nintendo Switch, PC, Android, iOS	Xbox One Controller, PS4/PS5 Controller, Arcade Stick, Nintendo Switch Pro Controller, GameCube Controller	2D, Contenido original con recreaciones <i>cartoon</i> de dicho personaje	133	Los personajes disponen de trajes que pueden ser de colores alternativos. Aunque también disponen de trajes nuevos, los cuales son personajes de otras saga de animación, videojuegos, películas o hasta personas reales.	26 millones de unidades

T. 1. Comparativa - Datos Básicos

Juego	Modos de juego	Mecánicas únicas
Super Smash Bros. Ultimate	<ul style="list-style-type: none"> - <i>Smash</i> (modo VS) - <i>Squad Strike</i> (modo VS, se escoge entre 3 o 5 personajes para una sola partida, los cuales se irán turnando cada vez que muera que se esté usando. El ganador es el que quede con más personajes.) - <i>Torneo</i> (modo VS clasificatorio en el cual se realizarán varios combates seguidos para conseguir un ganador) - <i>Smash Especial</i> (modo VS con reglas especiales) - <i>Modo Clásico</i> (peleas modo VS para un jugador. Cada personaje una ruta de peleas diferente) - <i>Mob Smash</i> (modo VS donde salen rivales extremadamente débiles los cuales hay que derrotar. La meta final varía dependiendo del modo) - <i>Home-Run Contest</i> (modo en el cual hay que golpear un saco de boxeo y lanzarlo lo más lejos posible) - <i>Mii Fighter</i> (modo de creación de personajes) - <i>Creador de escenarios</i> (modo de creación de escenarios) - <i>Online</i> (modo VS en línea. Existe la posibilidad de crear una sala privada para jugar con amigos) - <i>Desafíos</i> (lugar donde muestra unos <i>desafíos</i> que propone el juego para completar) - <i>Spiritus</i> (en este se dispone del <i>Modo aventura</i>, que son una serie de combates con temática y cinemáticas específicas de este modo y del modo <i>Spirit Board</i>, el cual permite realizar combates modo VS para conseguir un <i>espíritu</i>, estos siendo fotos colecciónables de diferentes personajes de videojuegos) 	<ul style="list-style-type: none"> - Ataques definitivos que hacen daño a todo aquel que esté cerca (<i>Final Smash</i>). - <i>Burlas especiales</i> las cuales pueden ser usadas de ataque. - Uso y existencia de <i>objetos</i>. - Existe un personaje con la habilidad de copiar una habilidad de los rivales. - Existen varios personajes que se pueden intercambiar entre sí en medio de batalla (<i>Entrenador Pokémon</i> dispone de 3 personajes jugables, <i>Squirtle</i>, <i>Ivysaur</i> y <i>Charizard</i>, los cuales se pueden cambiar en medio de combate). - Existe un personaje con la posibilidad de mejorar sus estadísticas en medio de combate, volviéndose más fuerte, más rápido, etc. - Posibilidad de creación de un personaje con el arquetipo "<i>Luchador</i>", "<i>Espadachín</i>" o "<i>Tirador</i>". - Existe un personaje con <i>inputs especiales</i> (referencia a personajes de juegos de lucha, los cuales deben de mover el <i>stick izquierdo</i> de una manera concreta para realizar ataque mejorado o especial). - Existe un personaje con la mecánica de elegir un "<i>hechizo</i>" de los juegos RPG, los cuales tienen funciones especiales tipo envenenamiento al rival, mejora de ataque o movimiento que puede hacer matar a alguien al momento. - Existe un personaje con la mecánica especial de construir <i>terreno</i> en donde quiera del escenario, siempre y cuando tenga recursos disponibles. - Mejora del personaje en cuanto a su fuerza siempre y cuando hayan recibido mucho daño. - Escenarios cambiantes en medio de la batalla. - Escenarios móviles en medio de la batalla. - Escenarios con mecánicas especiales en medio de batalla (posibilidad de un compañero más, posibilidad de un <i>mob</i> el cual hace daño a todo personaje que se encuentre en su rango, recolección de monedas para volverse de oro al superar las 100 monedas, etc).

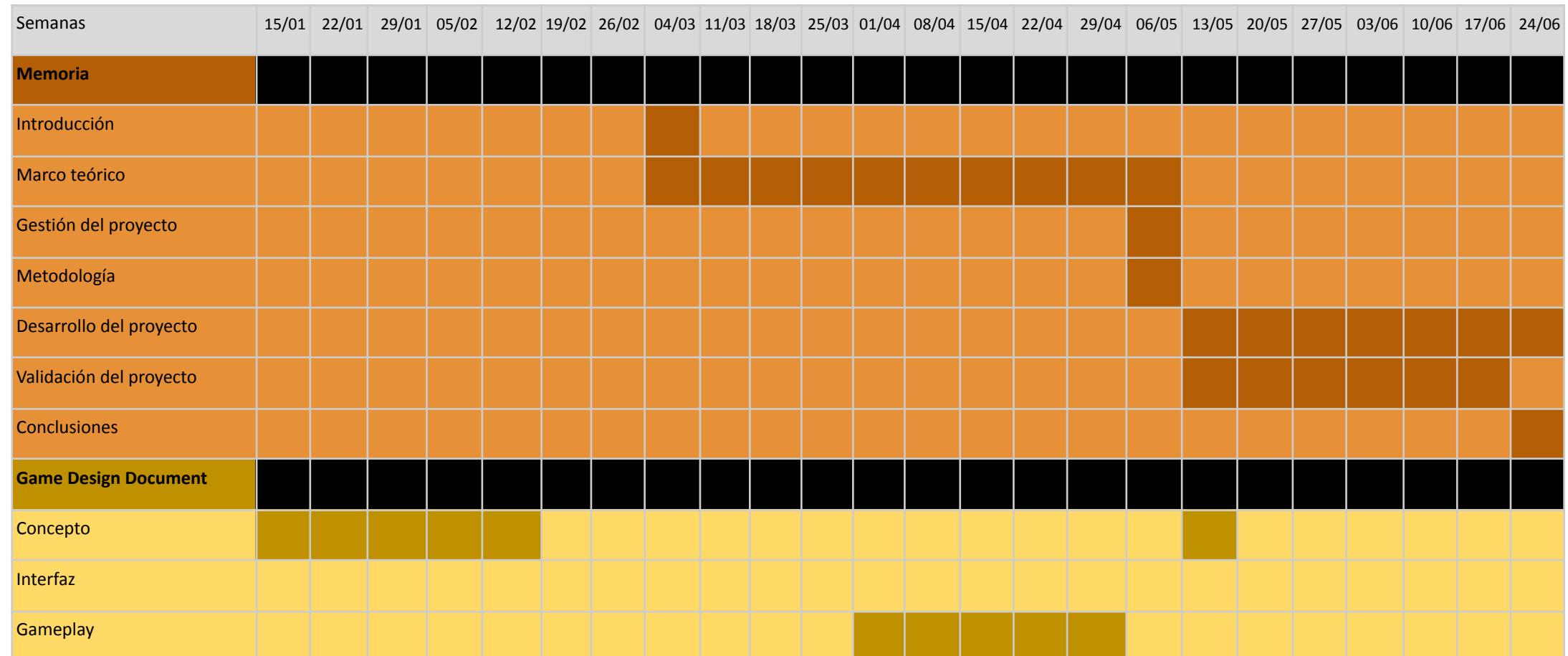
Rivals of Aether	<ul style="list-style-type: none"> - <i>Versus mode</i> (modo VS) - <i>Story Mode</i> (modo historia que consta de combates VS) - <i>Rivals of Tether</i> (un modo único en el cual los jugadores combaten en un juego americano de recreo llamado <i>tetherball</i>) - <i>Tourney</i> (modo VS clasificatorio en el cual se realizarán varios combates seguidos para conseguir un ganador) - <i>Online</i> (modo VS en línea clasificatorio o no clasificatorio) - <i>Abyss Endless Mode</i> (modo VS de un jugador en el cual aparecerán rivales, el cual deberá de derrotarlo. Este va en función de "olas" de enemigos y deberá sobrevivir hasta el final) 	<ul style="list-style-type: none"> - Los personajes no usan escudo, hacen uso de un <i>parry</i>, el cual, si lo usan al momento de recibir un golpe, dejarán al rival <i>aturdido</i> durante unos segundos. - Los personajes tienen la capacidad de realizar un <i>wavedash</i>, movimiento que al usar un salto y una esquiva direccional a la vez, ganan un movimiento terrenal más rápido y cancelan toda animación de aterrizaje y pueden moverse instantáneamente sin problemas. - Los personajes no tienen la capacidad de agarrarse de los bordes de los escenarios para recuperarse de una caída. - Existe un personaje el cual, al atacar a rivales, hace que suelten gemas, las cuales puede gastar luego en una tienda de nuevas habilidades que aparece al usar el botón de burla. - Existen personajes y escenarios hechos por la comunidad los cuales se pueden usar de forma legítima en el juego. - Escenarios con mecánicas especiales en medio de batalla (como puede ser personajes de fondo que ataquen o movilidad de elementos del escenario).
Nickelodeon All-Star Brawl 2	<ul style="list-style-type: none"> - <i>Battle</i> (modo VS) - <i>Online</i> (modo VS en línea clasificatorio o no clasificatorio) - <i>Campaign</i> (peleas modo VS para un jugador. Este dispone de una historia) - <i>Arcade</i> (peleas modo VS para un jugador) - <i>Boss Rush</i> (peleas seguidas contra los <i>jefes</i> que aparecen en el modo <i>Campaign</i>) - <i>Pop the Slime Balloons</i> (modo de golpear todos de globos antes de que se acabe el tiempo) - <i>Whack-A-Bot</i> (modo de golpear a los bots antes de que desaparezcan para conseguir el mayor número de puntos) 	<ul style="list-style-type: none"> - Uso y existencia de objetos. - Los personajes disponen de un ataque Smash "neutral" (referido a que pueden cargar un ataque Smash que se usa sin mover el stick izquierdo hacia ningún sitio). - Los ataques Smash se pueden ejecutar en el aire también, siendo estos unos ataques completamente nuevos. - Existe la mecánica de Slime Bar, la cual al atacar va incrementando. Esta barra se divide en tres secciones, las cuales al realizar según qué acciones haga. - Al tener una barra de Slime, el jugador puede potenciar un ataque Smash, un ataque especial, hacer una esquiva aérea o cancelar cualquier movimiento que se realice. - Al tener dos barras de Slime, el jugador puede cancelar todo el knockback que haya recibido de un golpe. - Al tener tres barras de Slime, el jugador puede realizar una ataque definitivo. - Los personajes tienen la capacidad de realizar un <i>wavedash</i>.
Multiversus	<p>Todos los modos disponibles son modo VS, aunque pueden ser jugados <i>Online</i> o <i>Local</i>. Estos modos VS son:</p> <ul style="list-style-type: none"> - <i>Team</i> (combates de 2 vs 2) - <i>Co-Op Vs AI</i> (combates de 2 vs 2 IAs) - <i>1 vs 1</i> (combates de 1 vs 1) - <i>FFA</i> (combates de todos contra todos, estos siendo combates de 4 personas como máximo) 	<ul style="list-style-type: none"> - Los personajes disponen de un ataque Smash "neutral" (referido a que pueden cargar un ataque Smash que se usa sin mover el stick izquierdo hacia ningún sitio). - Los ataques Smash se pueden ejecutar en el aire también, siendo estos unos ataques completamente nuevos. - El daño recibido se muestra debajo del jugador todo el rato (Esta opción se puede cambiar en el menú de configuración). - Las habilidades no se pueden ejecutar repetidamente ya que disponen de un <i>tiempo de recarga</i>. - Al realizar un ataque repetidamente en un <i>combo</i>, este se volverá más flojo con cada repetición. - Cada personaje tiene un rol en las partidas <i>cooperativas</i>, siendo estos roles: <i>Matón, Asesino, Tanque, Mago/Larga Distancia, Apoyo</i>. - Los personajes tienen la capacidad de realizar un <i>wavedash</i>.

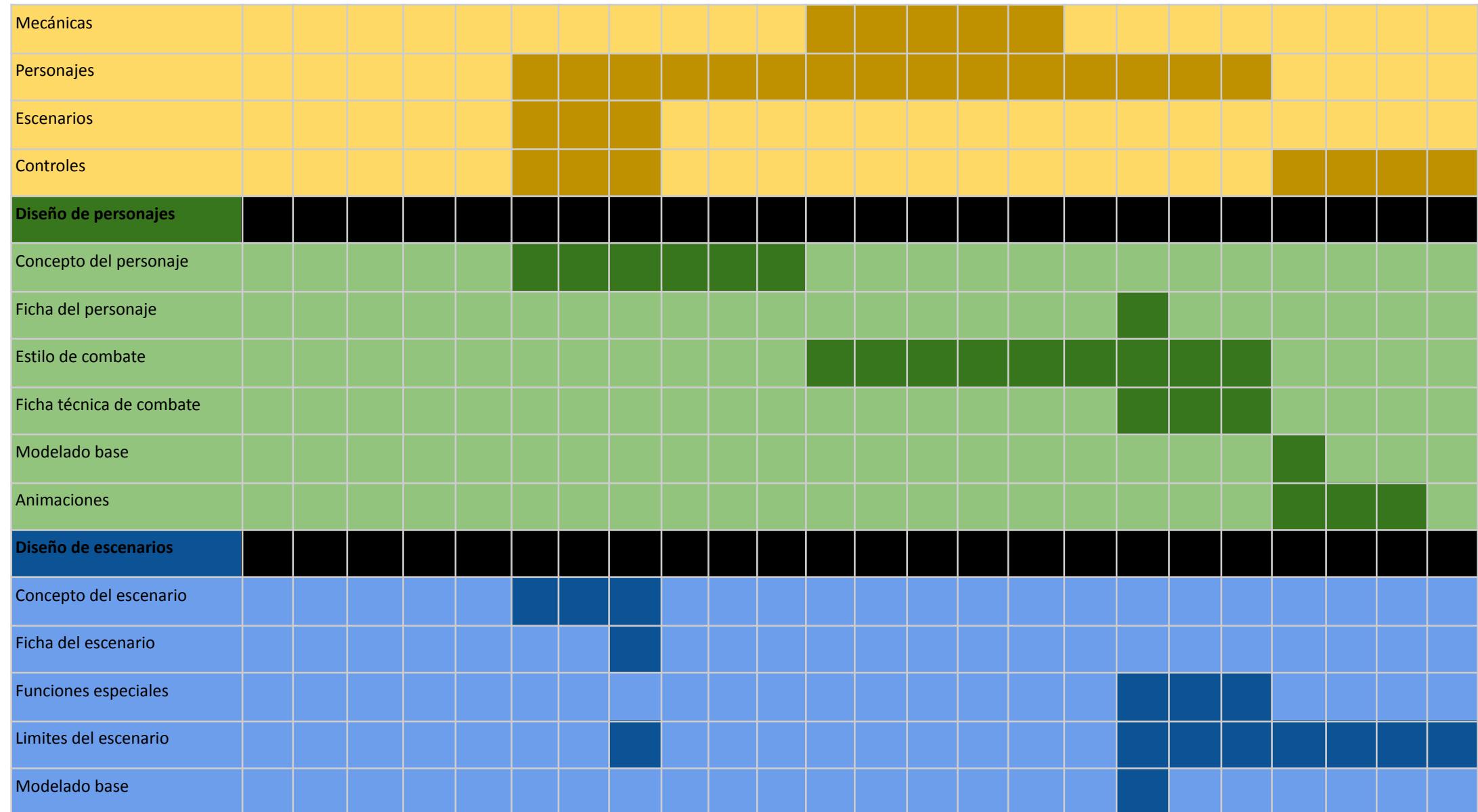
Brawlhalla	<ul style="list-style-type: none">- <i>Casual Matchmaking</i> (modo VS no clasificatorio <i>Online</i>)- <i>Ranked</i> (modo VS clasificatorio <i>Online</i>)- <i>Custom Online</i> (modo VS <i>Online</i> con modos de juegos personalizados)- <i>Brawl of the Week</i> (modo VS <i>Online</i> con modos de juegos personalizados semanales)- <i>Offline Play</i> (modo VS <i>Local</i>)	<ul style="list-style-type: none">- Uso y existencia de objetos. Estos objetos son armas las cuales pueden utilizar los personajes.- Cada personaje tiene un <i>set</i> de armas, las cuales están disponibles al coger un <i>objeto</i>.- Existe un <i>dash</i> moverse momentáneamente rápido, a diferencia de la acción de <i>correr</i> del resto de juegos.- Se dispone de tres saltos en vez de dos saltos como en el resto de juegos.- Es posible <i>cancelar la gravedad</i> al realizar un ataque aéreo con una esquiva aérea.- Los personajes no tienen la capacidad de agarrarse de los bordes de los escenarios para recuperarse de una caída.
-------------------	---	---

T. 2. Comparativa - Modos y Mecánicas

3. Gestión del proyecto

La clave para el éxito de dicho proyecto es la organización y planificación. Este segmento mostrará el plan de tiempo para cada tarea, así como el calendario y las tareas generales del proyecto.





Prototipado del proyecto																				
Controles básicos - Plataformas																				
Controles básicos - Lucha																				
Implementación de animaciones																				
Implementación de colisiones																				
Implementación de sonidos																				
Implementación de jugadores																				
Arreglos y cambios																				
Playtest																				
Creación de servidor de proyecto																				
Documentación de playtesters																				
Encuestas de playtesters																				
Análisis de encuestas																				

T. 3. Fechas de proyecto

3.1. DAFO

	Positivos	Negativos
Origen Interno	<p>Fortalezas</p> <p>Oportunidad de desarrollo de un proyecto.</p> <p>Experiencia previa en desarrollo de animaciones de <i>Platform Fighter</i>.</p> <p>Jugador del género de juego.</p> <p>Conocimiento sobre las mecánicas internas tanto de jugabilidad como de función de <i>Super Smash Bros</i>.</p> <p>Conocimiento sobre modelado, animación y programación en C#.</p> <p>Contacto con la comunidad de los <i>Platform Fighters</i>.</p> <p>Contacto con desarrolladores de juegos de lucha.</p>	<p>Debilidades</p> <p>Poco tiempo de desarrollo.</p> <p>Mucha carga para una sola persona.</p> <p>Inexperto en desarrollo de programación de juegos de lucha.</p>
Origen Externo	<p>Oportunidades</p> <p>Extensa cantidad de información técnica de <i>Super Smash Bros</i>. por parte de las comunidades.</p> <p>No existe una gran cantidad de <i>Platform Fighters</i> los cuales documenten su desarrollo.</p> <p>Gran interés de la comunidad por parte de nuevos proyectos estilo <i>Platform Fighter</i>.</p> <p>Es un proyecto de estudiante que da pie a errar y ganar experiencia.</p>	<p>Amenazas</p> <p>Público objetivo muy reducido.</p> <p>Probabilidad de no ser interesante para la comunidad.</p> <p>Aparición reciente de nuevos proyectos estilo <i>Platform Fighter</i>.</p>

T. 4. Análisis DAFO

3.2. Riesgos y plan de contingencias

Riesgo	Solución
No acabar de desarrollar el prototipo.	Optimizar el tiempo previamente descartando aquellos temas o mecánicas menos importantes y usando ese tiempo para acabar el prototipo.
No acabar de disponer de referencias para las animaciones del personaje.	Al ser un artista marcial el primer personaje, se puede optar a coger referencia de otras artes marciales o interpretaciones de personajes de otros juegos de lucha.
No disponer de suficiente tiempo para animar a todo el personaje.	Al encontrarse en este caso, se debería de optar por el reuso de algunas animaciones ya hechas anteriormente o incluso de usar webs que dispongan de animaciones gratuitas para usar en proyectos.
No ser lo suficientemente interesante.	En caso de no ser atractivo, se podría optar a implementar una nueva mecánica interesante que lo aleje del resto de <i>Platform Fighters</i> o tener un estilo visual que atraiga más al público.
Errores de compilación del prototipo.	En caso de ser errores leves, se puede optar a solucionarlo, camuflarlo o a desechar aquella mecánica que de error. En caso de ser errores graves, se debe de buscar el origen del error, siempre y cuando no lleve mucho tiempo. En caso de cubrir mucho tiempo, se deberá o rehacer el código que de error o <i>bloquearlo</i> para que no pueda ser ejecutado en el prototipo.

T. 5. Análisis de riesgos y soluciones

3.3. Herramientas del proyecto

Para gestionar tareas a lo largo del proyecto y desarrollar el prototipo a tiempo, estoy utilizando:

- **Github y Github Desktop:** para poder trasladar el proyecto de un lado a otro y gestionar las versiones de este.
- **Drive:** para poder guardar la documentación o *concepts* los cuales quiera usar más adelante. También me sirve para la creación de encuestas para los playtesters.
- **Discord:** principal canal de comunicación con los playtesters.
- **Unity:** programa principal de desarrollo del prototipo. Al tener experiencia previa en ello me resulta más cómodo todo el proceso de prototipado.
- **Adobe Photoshop:** programa para desarrollar los *concepts* de niveles, personajes, etc. También le doy uso para desarrollar *texturas* para la interfaz.
- **Autodesk Maya:** principal programa para el desarrollo de modelos, esqueletos y animaciones.

3.4. Análisis inicial de costes

En este análisis de costes se ha planteado desde la perspectiva de un desarrollo realista del proyecto como un estudio *indie*, en la cual se tendría un grupo de 5 personas trabajando, los cuales estarían un artista 2D, artista 3D, programador y músico. Estos podrían llegar a cobrar un total de 1.200€ al mes, teniendo que la hora saldría a 7,5 euro/hora, en el contexto de esta ser una indie para abarcar un proyecto para lanzar al público. Todo esto en un período de tiempo que podría abarcar 2 años perfectamente. En cuanto al uso de *software*, algunos serían gratuitos (*Visual Studio, GitHub, Blender*), pero otros habría que pagarlos (*Unity, Adobe Photoshop, Adobe Audition, Pro Tools - Music Software*). Por último se ha tenido en cuenta que las instalaciones primarias serán en Barcelona, el cual tiene unos costes indirectos de agua, electricidad, el alquiler del local, permisos y licencias.

Tipo	Tema	Coste	Type	Cantidad	Coste mensual	Amortización (anual)	Precio total
Personal	Salario	1.200,00 €	Mensual	5	6.000,00 €		144.000,00 €
Equipo	Ordenador	1.200,00 €	Amortización	5	250,00 €	2	6.000,00 €
	Pantalla	120,00 €	Amortización	10	50,00 €	2	1.200,00 €
	Teclado	30,00 €	Amortización	5	6,25 €	2	150,00 €
	Ratón	15,00 €	Amortización	5	3,13 €	2	75,00 €
	Mesa	150,00 €	Amortización	5	31,25 €	2	750,00 €
	Silla	100,00 €	Amortización	5	20,83 €	2	500,00 €
	Nevera	250,00 €	Amortización	1	10,42 €	2	250,00 €
	Microondas	60,00 €	Amortización	1	2,50 €	2	60,00 €
	Vitrocerámica	250,00 €	Amortización	1	10,42 €	2	250,00 €
	Tableta gráfica	60,00 €	Amortización	2	5,00 €	2	120,00 €
	Teclado MIDI	100,00 €	Amortización	1	4,17 €	2	100,00 €
Software	Pack Adobe Creative Cloud	67,57 €	Mensual	1	67,57 €	2	1.621,68 €
	Unity	185,00 €	Mensual	1	185,00 €	2	4.440,00 €
	Pro Tools - Music Software	34,00 €	Mensual	1	34,00 €	2	816,00 €
Costes indirectos	Local	1.000,00 €	Mensual		1.000,00 €		24.000,00 €
	Agua	150,00 €	Mensual		150,00 €		3.600,00 €
	Electricidad	250,00 €	Mensual		250,00 €		6.000,00 €
	Seguridad social	600,00 €	Mensual	5	3.000,00 €		72.000,00 €
	Permisos y licencias	1.000,00 €	Único pago	1	1.000,00 €		1.000,00 €
TOTAL							266.932,68 €

T. 6. Análisis de costes

4. Metodología

Para este proyecto, al desarrollar una aplicación, prefiero seguir el enfoque de **Diseño Centrado en el Usuario (DCU)**. En lugar de adoptar un método específico como **SCRUM**, opto por el **DCU** para asegurar que el producto final satisfaga las necesidades y expectativas de los usuarios.

Este enfoque implica entender profundamente a los usuarios finales, definir claramente sus requisitos y diseñar soluciones centradas en sus necesidades. Con esta metodología, busco crear un juego intuitivo, atractivo y hecho para los usuarios.

4.1. Diseño Centrado en el Usuario

El **DCU** es mucho más que una simple metodología de desarrollo, es una **filosofía que coloca al usuario en el corazón de todo el proceso** de diseño de productos y aplicaciones. Este enfoque reconoce que para garantizar el éxito de un producto, es fundamental **comprender las necesidades, deseos y limitaciones de los usuarios finales** desde el principio hasta el final del proceso de desarrollo.

Aunque el paradigma del **DCU** es aplicable a cualquier tipo de producto, ha adquirido una especial relevancia en el desarrollo de productos con un fuerte componente tecnológico, ya sean *hardware o software*. En estos productos, es común que se prioricen las prestaciones técnicas en detrimento de aspectos relacionados con la facilidad de uso para los usuarios finales. El **DCU** busca cambiar esta perspectiva al involucrar al usuario en todas las fases del desarrollo, desde la concepción hasta la evaluación.

Se puede comprender mejor al considerar ciertos elementos que están fuera del propio concepto del DCU:

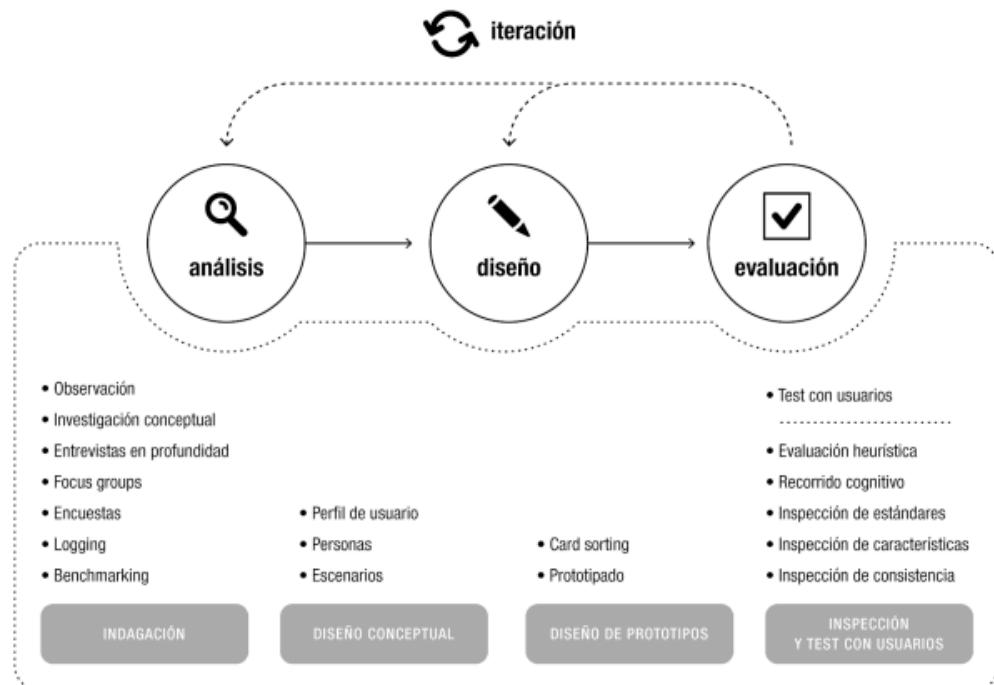
- Es crucial entender que **el usuario no eres tú, ni tu jefe, ni el cliente**. El usuario de un producto son **aquellas personas que lo van a utilizar y para quienes lo diseñamos**. A menudo se hace referencia al usuario final para enfatizar que se trata de la persona que realmente interactuará con el producto, no los intermediarios que facilitarán su uso.
- **No es "agnóstico"**, en cualquier proceso se tienen en cuenta **los requisitos tanto del usuario como del cliente o empresa, así como del producto mismo**. Estas tres fuentes de requisitos alimentan el proceso de diseño de manera integral.
- **No es ingeniería del software** y está considerablemente **alejado del enfoque de desarrollo en cascada de aplicaciones**. Se diferencia por su **enfoque centrado en el usuario y su énfasis en la iteración y la mejora continua**.

- **No es marketing ni estudios de mercado.** Aunque estas disciplinas pueden y deben interactuar y complementarse entre sí, su **enfoque hacia el usuario es diferente, al igual que sus objetivos y métodos.**
- Aunque a menudo revela necesidades de los usuarios que no se han abordado previamente, **no debe confundirse con el desarrollo de nuevos productos propio del marketing.** Si bien ambas disciplinas pueden compartir ciertos aspectos, se centra principalmente en **diseñar productos y aplicaciones que sean útiles, utilizables y satisfactorios** para los usuarios finales.

4.1.1 Fases de desarrollo de un DCU

El diseño centrado en el usuario se apoya en una variedad de enfoques y técnicas que ponen al usuario como el punto focal de todas las etapas del proceso de diseño. El siguiente esquema muestra un resumen de los principales métodos utilizados en cada fase.

En esta representación visual, los distintos métodos empleados en el diseño centrado en el usuario se agrupan en diferentes categorías. Es relevante tener en cuenta que esta clasificación no es estricta y que la elección de los métodos debe considerar las particularidades y objetivos de cada proyecto y fase del mismo.



3. 1. Los métodos del diseño centrado en el usuario

Fuente: https://openaccess.uoc.edu/bitstream/10609/76105/6/Introducción%20a%20la%20interacción%20persona%20ordenador_Módulo%203_Diseño%20centrado%20en%20el%20usuario.pdf

A continuación, se presentan de manera concisa los principales métodos organizados de acuerdo con el proceso estándar del DCU: investigación y requisitos de usuario para definir el producto e informar del diseño, la creación y el desarrollo de la arquitectura de información y los prototipos, y la evaluación de los elementos diseñados.

4.1.2 Investigación y requisitos de usuario

Los métodos de investigación se aplican en las etapas de comprensión del entorno de uso y de los requisitos. Se centran en la participación activa de los usuarios en diversas actividades con el fin de recopilar información para definir el producto o servicio. Entender a fondo a los usuarios, su entorno de uso, sus necesidades, objetivos y actitudes es fundamental para un diseño centrado en el usuario y para crear aplicaciones y entornos que sean fáciles de usar.

Las claves para recoger la información se podrían clasificar en:

- **Observación e investigación contextual:** implica observar a los usuarios en su entorno habitual para comprender sus acciones y condiciones. Proporciona información valiosa sobre el comportamiento y la usabilidad de los usuarios.
- **Entrevistas en profundidad:** se realizan con muestras pequeñas y guiones abiertos para comprender las necesidades y experiencias de los usuarios, no buscan representatividad sino comprensión en profundidad. No se enfocan en cuantificar respuestas, sino en explorar procesos y significados.
- **Dinámicas de grupo:** implican entrevistas con seis a ocho personas, donde la moderación es esencial para obtener información de calidad y fomentar la participación. El moderador sigue un guión para dirigir la conversación hacia los temas de interés y la investigación.
- **Encuestas:** es una técnica cuantitativa que requiere una muestra representativa de usuarios y utiliza formularios estructurados con preguntas que cubren todas las posibles alternativas de respuesta. Se diferencia de los métodos anteriores por su enfoque en la recopilación de datos *cuantitativos*.
- **Logging:** las *técnicas de registro informático* implican monitorear la actividad de los usuarios para recopilar y analizar datos de su interacción con un sistema o sitio web. Utilizan sistemas automatizados para la recopilación y procesamiento de datos, a menudo aprovechando los archivos de registro de actividad del sistema, conocidos como "logs".
- **Análisis competitivo:** implica examinar productos similares o competidores del sistema interactivo que se está diseñando. Los objetivos incluyen comprender las expectativas de los usuarios, entender las tendencias del mercado, aprender de errores y éxitos, y estudiar características comunes, interfaces, entre otros aspectos.

4.1.3 Diseño hacia el usuario

Las técnicas de perfiles de usuario, personas y escenarios nos permiten comprender a los usuarios, sus motivaciones y cómo utilizan los sistemas interactivos. Estas técnicas se basan en la recopilación y análisis de datos de entrevistas, observaciones, encuestas, entre otros, para crear modelos que describen a los usuarios. Es crucial haber realizado previamente una investigación de requisitos y análisis de usuarios para desarrollar estos perfiles y escenarios.

- **Perfil de usuario:** son agrupaciones de usuarios según su *persona*. Normalmente, son el resultado de un estudio que permite definir porcentajes de cada perfil y agruparlos. Las características de cada persona pueden ser aspectos socio demográficos, actitudinales, de expectativas, etc.
- **Personas:** son descripciones de usuarios arquetípicos que guían el proceso de diseño, construidas a partir de datos de investigación para representar grupos de usuarios y recordar a los diseñadores para quién están diseñando.
- **Escenarios:** es la narrativa de un usuario enfrentándose a un sistema o producto con objetivos específicos, detallando el contexto y la secuencia de acciones. Estas situaciones son esenciales para explorar ideas y considerar aspectos del diseño.

4.1.4 Prototipado

Gracias a la información de *los perfiles de usuario, la persona y los escenarios*, se puede obtener resultados y la definición de sus requisitos. Esta información sirve para iniciar a realizar el prototipo del proyecto.

- **Card sorting:** se trata de una actividad en la que los usuarios organizan y clasifican conceptos representados en tarjetas. Al ordenar estas tarjetas, se logra una categorización que facilita la comprensión y la estructuración natural de la información para los usuarios.
- **Prototipado:** consiste en crear modelos o prototipos del sistema que se está diseñando. Estos modelos, que pueden ser parciales o simulaciones completas, se utilizan para realizar pruebas antes de que el proyecto esté completamente terminado. No intentan replicar todo el sistema, sino que se centran en partes específicas que son más complejas.

Los modelos que se utilizan con más frecuencia son el prototipado de baja fidelidad, alta fidelidad, vertical y horizontal.

- **Baja fidelidad:** son esquemáticos y representan aspectos generales del sistema o diseño de interfaz, sin detalles. Se construyen con lápiz y papel y tienen como objetivo proporcionar una idea inicial de la disposición y visibilidad de los elementos de la interfaz. No incluyen

diseño gráfico ni funcionalidades avanzadas. A menudo se utilizan para obtener retroalimentación de los usuarios mediante diseño participativo.

- **Alta fidelidad:** busca crear un modelo muy cercano al producto final. Se emplea para evaluar aspectos funcionales y de usabilidad, mediante pruebas con expertos o usuarios.
- **Vertical:** implica crear un modelo detallado de una sección específica del producto final, con su apariencia y funcionalidad completas. Esto posibilita realizar evaluaciones de usabilidad sobre partes específicas o funciones determinadas del producto.
- **Horizontal:** implica la creación de un modelo que representa todas o la mayoría de las funcionalidades, espacios y menús del producto o sitio web, pero sin que estas estén completamente implementadas o funcionen. Este tipo de prototipo se utiliza principalmente para evaluar el alcance del sistema, la navegación y la arquitectura de la información, en lugar de centrarse en la interacción real del usuario.

4.1.4 Tests con usuarios

Los métodos de evaluación de usabilidad, conocidos como tests con usuarios, son fundamentales para evaluar la efectividad de los diseños y mejorar su calidad. Estos métodos implican la realización de experimentos y pruebas con los usuarios reales del sistema o producto interactivo. A través de ellos, se pueden identificar obstáculos y dificultades que los usuarios enfrentan al interactuar con la interfaz, así como evaluar la eficacia de los procesos diseñados.

Estos tests proporcionan información cualitativa sobre la experiencia del usuario, centrándose en la interpretación de los resultados más que en métricas cuantitativas. Se analiza si las interfaces y procesos están adecuadamente diseñados y si los usuarios pueden alcanzar sus objetivos de manera eficiente. La complejidad de los tests puede variar desde simples tareas con pocos usuarios hasta evaluaciones más elaboradas con una amplia muestra de usuarios y diferentes perfiles.

Existen diferentes modalidades de tests con usuarios, cada una adaptada a los objetivos y características del proyecto. Por ejemplo, el protocolo del pensamiento en voz alta permite obtener información adicional al solicitar a los usuarios que expresen sus pensamientos mientras realizan las tareas. Otros métodos implican la formulación de preguntas relacionadas con la experiencia del usuario durante la interacción.

Normalmente, estos tests se llevan a cabo en un laboratorio de usabilidad, donde se puede observar la interacción del usuario con la interfaz y registrar sus expresiones faciales y comentarios verbales. Esta información complementaria ayuda a comprender mejor la experiencia del usuario y a identificar áreas de mejora en el diseño.

Sin embargo, los tests con usuarios también tienen desventajas, como el costo asociado al tiempo y los desplazamientos de los usuarios y expertos involucrados. Además, la presencia en un laboratorio puede influir en el comportamiento de los usuarios, generando resultados no satisfactorios. Una solución a este problema es el test remoto, que permite realizar pruebas a distancia, evitando así el sesgo del laboratorio y simplificando la captación de usuarios.

A pesar de las limitaciones, los tests con usuarios son una herramienta invaluable para evaluar la usabilidad de una interfaz y descubrir áreas de mejora. Permiten identificar problemas que de otro modo podrían pasar desapercibidos y garantizan que el diseño final se adapte de manera efectiva a las necesidades y expectativas de los usuarios.

5. Desarrollo del proyecto

5.1. Documento de diseño

5.1.2. Concepto

- **Título:** Platform Fighter.
- **Diseñador:** Alex Gestí Fernández.
- **Plataforma:** PC, con soporte de mandos de XBOX, Playstation 4 y 5, Nintendo Switch y Gamecube. Está desarrollado en PC, ya que es una plataforma que todo el mundo dispone en su casa y es fácil de poder jugar, principalmente, ya que se puede conectar todo tipo de controles, haciendo así que se pueda tener una amplia gama de controles sin necesidad de tener una plataforma concreta para poderlo jugar con ese mando. Además, este al no ser un juego que requiera mucha potencia gráfica, se podría llevar a otras plataformas si es requerido.
- **Versión del GDD:** 0.1.5.16032024.
- **Sinopsis de Jugabilidad:** Videojuego en el cual una serie de personajes saltan de plataforma a plataforma y se pegan entre ellos para intentar sacarse fuera del escenario.
- **Categoría:** A continuación se comparará con un par de juegos del mismo género, los cuales han triunfado en el sector, *Super Smash Bros Ultimate* y *Rivals of Aether*, los cuales son muy parecidos:
 - *Super Smash Bros Ultimate* se centra en personajes icónicos de franquicias de videojuegos, mientras que el caso del proyecto presenta personajes originales y únicos.
 - La jugabilidad de estos dos juegos tiende a ser más accesible para jugadores veteranos, mientras que el caso del proyecto busca un equilibrio entre estos, un perfil de jugadores veteranos y uno casual.
 - Las mecánicas de estos dos juegos están establecidas desde **1999**, teniendo a *Rivals of Aether* con algún cambio un tanto diferente, pero manteniendo las bases de los modos, por ello casi nunca ha aportado muchas ideas nuevas, a diferencia de este proyecto que buscará inventar alguna mecánica nueva y algún modo diferente del ya establecido.
 - *Super Smash Bros Ultimate* ha tenido mínima interacción con la comunidad, a diferencia del proyecto que está en contacto continuo con

la comunidad en todo momento para tener la mejor experiencia posible para los jugadores.

- **Mecánica:** Este juego tiene componentes de dos géneros de videojuegos:
 - **Plataformas**, ya que dispone de mecánicas de este género como el salto, el salto en la pared, andar, correr, etc y toda su base es estar *dentro del escenario*.
 - **Fighter/Lucha**, ya que la mecánica principalmente que hay es pegarse para poder ganar, sea con movimientos flojos y rápidos, con movimientos fuertes o incluso especiales, bastante típicos de este género de juegos.

Con ello, en este juego los jugadores deben de pegarse entre ellos, de todas las maneras posibles, incluso lanzándose objetos o agarrándose para lanzarlos fuera del área de “vivir” o que se caigan del *escenario*.

El último que le queden vidas es el ganador de la partida, o, en caso de ser un modo por tiempo, la persona que haya echado más veces del *escenario* a los rivales es el ganador.

- **Tecnología:** Para este juego se necesitará una serie de programas y *hardware*:
 - **Programas:**
 - Unity 2022.3.30f1 (C#)
 - Adobe Photoshop 2022
 - Maya 2022
 - Substance Painter
 - Adobe Audition 2020
 - Github
 - Discord
 - Google Docs
 - **Hardware:**
 - PC y sus periféricos (teclado, ratón, pantalla/s)
 - Red Wifi
 - Mando de Xbox One/Series X
 - Mando de PS4/PS5

- Mando Pro de Nintendo Switch
- Adaptador Oficial de Gamecube
- Mando de Gamecube
- **Público:** Este producto va dedicado a tanto a jugadores casuales como jugadores veteranos que busquen nuevas experiencias en el mundo de los videojuegos.

Por ponerlos en contexto, estos dos tipos de jugadores cumplen un perfil parecido a este:

○ **Jugadores casuales:**

- **Edad:** 12 a 35 años.
- **Intereses:** Socializar, cultura japonesa, juegos de *plataformas*, juegos de *aventuras*, juegos *RPG*, literatura, películas de animación, películas de acción.
- **Motivación:** Busca una experiencia de juego que sea divertida y entretenida, que pueda jugar tanto solo como con amigos de manera despreocupada.
- **Comportamiento:** Busca pasar el rato de manera entretenida, leyendo, estando con los amigos o incluso jugando, aunque este último no dispone mucho tiempo para hacerlo, ya sea por los estudios o por el trabajo.

Aun así su nivel de apego hacia esto es pequeño, ya que es un simple hobby para pasar el rato, con lo cual lo mantiene siempre aparte del resto de su vida.

- **Otros juegos que juega:** Mario Kart 8 Deluxe, The Legend of Zelda: Tears of the Kingdom, Animal Crossing New Horizons.
- **Plataformas que usa:** Nintendo Switch, PC, Playstation 5.
- **Otras variables:** Son del tipo de jugador que cuando ven un juego interesante se lo compran sin pensarlo y luego lo critican. Normalmente, son de la gente que se dejan guiar por las recomendaciones del resto.

En muchas ocasiones suelen jugarlos poco rato y dejarlos abandonados.

○ **Jugadores veteranos:**

- **Edad:** 18 a 45 años.

- **Intereses:** videojuegos en toda su totalidad, eSports, socializar con la comunidad, hacer/ver *streams*, navegar por las redes sociales, participar en eventos, realizar *speedruns*.
- **Motivación:** llegar a ser el mejor en aquel juego que juegan, sea mediante torneos, la velocidad en la cual se lo pasan o parecido.
Además, quieren que ese juego sobreviva al paso del tiempo, haciendo así que la gente lo recuerde.
- **Comportamiento:** Suelen ser jugadores que lo dan todo por el juego que les gusta, hasta el punto de llegar a vivir de ello, sea participando en torneos, convirtiéndose en *streamers*, siendo organizadores de torneos...
Además de ello, suelen analizar el juego muy en profundidad, tanto sus mecánicas, como se mueven, como es el dispositivo en el cual lo juegan, como es el código de ese juego para entenderlo mejor...
- **Otros juegos que juega:** Super Smash Bros Melee, Rivals of Aether, The Legend of Zelda: Ocarina of Time, Super Mario 64.
- **Plataformas que usa:** Nintendo Switch, Nintendo Wii, Nintendo Gamecube, PC, PS5, Xbox Series X.
- **Otras variables:** Son jugadores los cuales no tienen problemas en gastarse el dinero para disfrutar mejor en aquellos juegos que juegan, sean comprando nuevos mandos, figuras que se puedan utilizar en aquella consola o simplemente comprar el último producto lanzado por la empresa que les gusta sin ningún problema, hasta llegar al punto de colecionar cada cosa que tenga relación con aquella empresa, por muy mínima la relación sea.

Además, estos jugadores suelen estar atascados en el pasado, o dicho de otra manera, son jugadores que tienen mucha *nostalgia* hacia aquellos juegos antiguos que jugaban cuando eran unos niños o adolescentes. Este factor es importante, ya que cualquier producto actual que no se parezca en nada al nuevo puede que lo rechacen totalmente.

Por otra parte, estos jugadores, cuando están por un juego, están al 100% por ello, olvidándose de todo el resto hasta que lo completan totalmente o creen que lo han finalizado. Aun así, siguen jugando aunque lo hayan finalizado o se lo vuelven a jugar desde 0 múltiples veces.

5.1.3. Jugabilidad

5.1.3.1. Mecánicas

● Movement

- **Walk:** Es un tipo de movimiento lateral lento, el cual se puede realizar al mover el stick un poco en el eje X. Este tipo de movimiento retiene su uso para poder realizar algunos ataques mientras camina, dando así un uso técnico a esta mecánica. Además, este movimiento varía su velocidad dependiendo de cuanto se mueva el stick.

Los personajes tienen diferentes velocidades de andar, haciendo así que sea un valor más dentro de los valores de cada personaje, independientemente del resto.

- **Run:** Es un tipo de movimiento lateral rápido, el cual se puede realizar al mover el stick rápidamente hacia una de las puntas en el eje X, teniendo que pasar del punto 0, 0 al punto 1, 0 o -1, 0 dependiendo hacia qué dirección se quiera ir, este tipo de acción con el stick se le suele conocer como **tap** o **tapping**. Este tipo de movimiento retiene su uso para así poder realizar ataques mientras se corre, dando así un uso técnico a esta mecánica.

Además, al realizar este movimiento, se inicia con un **dash inicial** como animación y fuerza inicial antes de empezar a correr. Esta animación inicial permite al jugador cambiar rápidamente de dirección antes de empezar a correr si así lo desea.

Por último, al parar de correr o cambiar de dirección justo después de correr, se realiza una **frenada**, la cual dependiendo de si el jugador cambia de dirección se realizará un **pivoteo**. Cada frenada será diferente dependiendo de la cantidad de **tracción** que tenga.

Los personajes tienen diferentes velocidades de correr, haciendo así que sea un valor más dentro de los valores de cada personaje, independientemente del resto.

- **Jump:** acción que el jugador puede realizar para *saltar*, el cual le permite moverse verticalmente por un periodo de tiempo hasta que la gravedad afecte y caiga hacia abajo. Este tipo de movimiento se puede realizar tanto por la pulsación de un botón. La altura del salto variará en otro tipo de salto si el jugador mantiene pulsado muy momentáneamente el botón de salto a uno más bajo, el cual se le conoce como **short hop**.

Además, el jugador tendrá disponible otro salto adicional, conocido como **doble salto**, siempre y cuando este no haya realizado todos los saltos posibles, haya hecho una esquiva aérea o esté en la animación de haber sido golpeado.

Por otro lado, estos movimientos permiten al jugador realizar una serie de movimientos que solo están disponibles si el personaje está en el aire.

Los personajes tienen diferentes velocidades de salto, doble salto y de *short hop*, además de diferentes velocidades de movimiento aéreo, haciendo así que sean unos valores más dentro de los valores de cada personaje, independientemente del resto.

- **Fast fall:** esta acción solo se puede realizar realizando un **tap hacia abajo** y cuando el jugador **está en el aire**, haciendo así que caiga más rápido de lo normal, siempre y cuando no, no esté en una animación de haber sido golpeado. Al hacer esto, la gravedad se ignorará y caerá con una velocidad ajustada por defecto sin ninguna aceleración.

Por este motivo, cada personaje dispone de un valor de velocidad de caída rápida independiente de cada personaje.

- **Edge grab:** acción en la cual el personaje se agarra de la punta del escenario para no caerse, que se puede realizar únicamente cuando se está en el aire y en una de las puntas del escenario, las cuales deben de tener un ángulo igual o menor de los 90º, y no son plataformas pequeñas o traspasables.

Al realizar esta acción, el personaje puede volver al escenario realizando únicamente una acción que el jugador seleccione: subiendo normalmente y mantenerse cerca de la punta del escenario, dando una boletera cuando está subiendo para tener más distancia, saltar o dar un golpe delante del personaje.

- **Attacks:** acción principal del juego, en el cual golpeas a tu oponente para hacerle daño. Estos ataques están clasificados en diferentes tipos de golpes:

- **Neutral attacks:** ataques realizados solamente pulsando el botón de ataque sin que el personaje se mueva con el stick. Pueden ser realizados en el aire o en el suelo.

Estos suelen ser tres ataques seguidos siempre y cuando el jugador vuelva a pulsar el botón de ataque mientras se está realizando el *neutral attack*. Siendo el último de ellos el más fuerte de los tres.

- **Tilt attacks:** ataques realizados en el suelo moviendo suavemente el stick hacia arriba, abajo, derecha o izquierda y pulsando el botón de ataque o manteniéndolo pulsado.
 - **Forward tilt attack:** ataque realizado hacia delante para, como los jugadores denominan, “*sacarse al rival de encima*”. Normalmente no son ataques muy fuertes, pero permiten alejar al rival a cierta distancia.
 - **Down tilt attack:** ataque rápido en el suelo, el cual solo se puede realizar si el personaje está agachado. Es bastante rápido, lo que permite realizarlo varias veces seguidas, pese a eso, no es muy fuerte.
 - **Up tilt attack:** ataque realizado hacia arriba para mantener a los rivales en el aire, esta acción también se denomina coloquialmente como “***hacer malabarismos***”.
- **Smash attacks:** ataques cargados y fuertes realizados en el suelo al hacer *tapping* hacia una dirección seguido de pulsar el botón de ataque.
 - **Forward smash:** a diferencia del “*forward tilt*”, este ataque fuerte se usa para realizar el mayor daño posible al rival, hasta el punto de echarlos del escenario.
 - **Down smash:** ataque permite tomar un poco de distancia de los rivales. Raramente se usa para rematar, ya que, aunque sea el más veloz de todos los *smash attack*, es el más débil de todos.
 - **Up smash:** ataque fuerte que pone al rival en el aire y, si el rival está herido, es capaz de echar al rival a la parte superior del escenario.
- **Aerial attacks:** ataques realizados en el aire moviendo suavemente el stick hacia arriba, abajo, derecha o izquierda y pulsando el botón de ataque.

Cada uno de estos tienen un patrón diferente entre ellos, con lo cual no se suelen clasificar de una manera concreta pese tener los nombres de ***Forward Air***, ***Back Air***, ***Up Air*** y ***Down Air***. Aún así, uno de estos siempre permite al jugador realizar un ***smash meteórico***, el cual tiene un gran impulso y un ángulo de 270º y hace caer al rival rápidamente hacia abajo.
- **Dash attacks:** ataques realizados al correr y pulsando el botón de ataque.

- **Throws:** ataques donde el personaje debe coger, con el botón de agarrar, y mover el stick hacia arriba, abajo, derecha o izquierda, para así lanzarlo hacia una de esas direcciones.

Además, mientras el personaje tiene agarrado a alguien, puede empezar a golpearle rápidamente, con el botón de ataque, antes de lanzarlo o de que se escape.

- **Get-up attacks:** ataque único, ejecutable con el botón de ataque, que solamente se puede realizar cuando el personaje ha sido golpeado y está estirado en el suelo por culpa del golpe.
- **Special moves:** ataques especiales, ejecutable con el botón de especiales juntamente moviendo el stick hacia arriba, abajo, derecha o izquierda, o sin moverlo, el cual realiza un movimiento el cual es puede hacer funciones diferentes, como reflejar un ataque, lanzar un objeto, un ataque más fuerte, realizar un *counter*... puede variar dependiendo del personaje.

Aun así, siempre se cuenta con un ataque especial hacia arriba el cual da un impulso hacia arriba, como si de un tercer salto se tratara. Después de ejecutar este movimiento el jugador se queda sin la oportunidad de hacer más movimientos, quedando así en una caída libre donde la única acción que puede hacer es moverse de en el aire de manera horizontal para tocar el suelo o una zona agarrable del escenario.

- **Shield:** también conocido como guardia, es una burbuja de energía que rodea al personaje, que lo protege de la mayoría de los ataques, específicamente, cualquiera que no requiere coger a alguien o que no sea un ataque imbloqueable. Esta acción se suele realizar con el botón designado para el escudo.

Además, al realizar el escudo, se puede realizar un **dodge** o esquiva, la cual se puede realizar en el sitio, teniendo así el **spot dodge**, o realizarlo dando una voltereta hacia adelante o hacia atrás, estos llamados **rolls**, respectivamente, **forward roll** y **backwards roll**.

Este tipo de esquiva también se puede realizar en el aire, tanto con el stick neutral como moviendo el stick para ir hacia el lado que se quiera ir.

- **Taunt:** acción para provocar al rival, la cual se pueden llegar hasta ser tres burlas diferentes con sus botones designados.

Este tipo de movimiento no suelen tener acciones algunas, pero algunas veces pueden tener interacciones especiales, desde ser un movimiento

que haga un mínimo de daño, hasta uno que haga más fuerte al personaje temporalmente.

- **Cancels:** es una acción interna en el juego, la cual no se nombra normalmente, que permite al jugador cancelar la acción que esté haciendo para poder realizar otra de manera seguida, saltándose así unos *frames* y tener *frames de ventaja*. Normalmente, a los movimientos lentos y fuertes no se les suele atribuir esta mecánica.
- **Tech:** también referenciado como *freno de la caída*, es una acción que se puede ejecutar cuando el jugador presiona el botón de escudo poco antes de que el personaje entre en contacto con el suelo. Esta acción permite al jugador frenar todo el golpe al caer al suelo y volver a la acción al mismo momento de haber tocado el suelo. Si no se realiza esta acción en el momento justo, no se podrá realizar hasta haber sido golpeado de nuevo.

● Game Physics

- **Traction:** medida la cual hace que el personaje resbale al pararse cuando se está moviendo por el suelo. Personajes con tracción menor suelen resbalar mucho más lejos que otros personajes con mayor tracción. Además, con este valor y al estar en la animación de “resbalar”, el personaje no puede realizar ninguna acción hasta que no haya terminado con esta, lo cual le deja vulnerable a golpes.
- **Weight:** medida la cual muestra cuánto pesa un personaje y resiste de manera mayor o de manera menor al *knockback* que se le aplique por el golpe recibido.

Los personajes más pesados tendrán más resistencia al *knockback*, mientras que los más ligeros tendrán menos.

- **Gravity:** medida dedicada a que tan rápido un personaje que cae alcanza su aceleración máxima de caída. Los personajes disponen de diferentes medidas de máxima velocidad de caída, ya que un personaje con alta gravedad no debe de por qué caer a una velocidad superior que la de otro personaje, al igual que este valor influye en la misma altura máxima de salto, teniendo que un personaje con mucha gravedad puede saltar más bajo que otro con poca.
- **Knockback:** medida la cual calcula cuánta potencia, distancia y ángulo sale volando el personaje al recibir un golpe del rival.

Esta medida va mano en mano con el *damage*, haciendo así que cuanto más daño tenga, más potencia y ángulo tendrá, y más distancia recorrerá.

Esta medida varía dependiendo del peso, la gravedad y tracción que tenga el personaje, teniendo que personajes pesados y con mucha gravedad serán más difíciles de que se muevan del sitio, mientras que personajes menos pesados serán mucho más fáciles de mover del sitio.

- **Damage:** medida básica de unidad de cuán vulnerable es un personaje al *knockback* del ataque dado por los rivales. Cuanto más bajo es este valor, menos vulnerable es, cuanto más alto es, más vulnerable, llegando hasta poder realizar un *KO*.

El daño es representado en un porcentaje que va desde el 0% al 999%, siendo este el límite, de manera que el personaje empezará con 0% y se irá incrementando su daño hasta que reciba una muerte y se resetee a 0% de nuevo.

Por otro lado, cada ataque recibido tiene una cantidad de daño determinada, con lo cual, el daño recibido de un movimiento siempre será el mismo para cada tipo de ataque que se ejecute y no habrá valores al azar.

- **Hitstun:** periodo de tiempo después de recibir un golpe en el cual el personaje es incapaz de efectuar cualquier acción, aparte de moverse hacia qué dirección quiere ir o realizar un *tech* al entrar en contacto con el suelo. Además, esta variable es muy dependiente del *knockback* recibido.
- **KO:** normalmente conocido como la muerte en la partida, suele ocurrir al haber sido lanzado al límite del mapa, haciendo así que el porcentaje del personaje, es decir, su daño, vuelva a ser cero y pierda una vida o un punto, dependiendo del modo de juego en el cual se esté.

5.1.3.2. Diseño de movimiento

A continuación, se describen los movimientos del personaje en detalle, cubriendo tanto sus fundamentos como las habilidades específicas.

El objetivo es ofrecer una visión más detallada de cómo cada movimiento contribuye al estilo de juego del personaje y cómo encaja en el diseño general del juego.

Se analizarán los principios detrás del desarrollo de cada movimiento, teniendo en cuenta factores como la jugabilidad, el balance y la coherencia con las mecánicas generales del juego.

- **Descripción general del personaje:**

El primer personaje y el que va a servir de base neutral para demostrar las mecánicas fundamentales del juego, sin características únicas ni un estilo de lucha definido. Su propósito es ofrecer a los jugadores una experiencia directa y accesible, permitiéndoles familiarizarse con los controles y las mecánicas principales sin la complejidad adicional de un estilo de combate especializado.

Utiliza una combinación de puñetazos, patadas y ataques con la cabeza para ejecutar movimientos simples pero efectivos. Algunos de estos movimientos hacen referencia a otros personajes de juegos *platform fighter*, creando una conexión reconocible para aquellos familiarizados con el género.

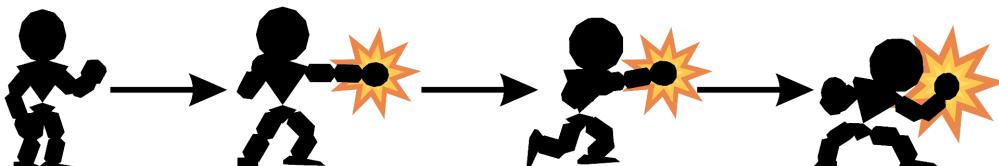
A pesar de no tener un estilo propio, el personaje permite una transición fluida entre ataques, asegurando que los personajes puedan experimentar y aprender las mecánicas sin sentirse abrumados.

A continuación se explicará la idea detrás de los ataques con sus estadísticas de la *hitbox* más grande, ya que las pequeñas son un 50% más flojas que la mayor:

- **Movimientos de ataque básicos**

- **Neutral Attack:**

La idea detrás de este movimiento es ejecutar una serie de golpes rápidos sin desplazarse, donde los dos primeros ataques son más ligeros y el tercero, más poderoso, sirve para mantener al oponente a raya. El objetivo es transmitir fluidez y velocidad en los primeros golpes, seguido de un impacto contundente en el último para cerrar la secuencia de manera efectiva.



Movimientos de un Platformer Fighter - Neutral Attack

➤ **Stats base:**

● **Golpe 1:**

- **Escala:** x1,5 de su tamaño original
- **Launch Angle:** 5 grados
- **Launch Speed:** 5 unidades
- **Damage:** 3 unidades

● **Golpe 2:**

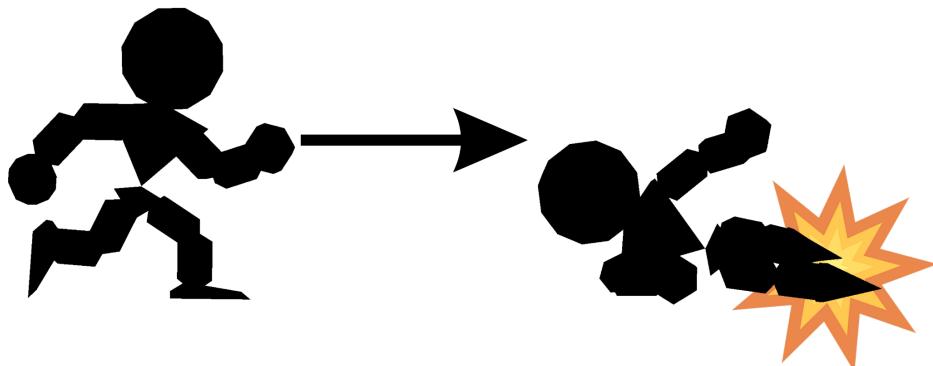
- **Escala:** x1,5 de su tamaño original
- **Launch Angle:** 5 grados
- **Launch Speed:** 5 unidades
- **Damage:** 3 unidades

● **Golpe 3:**

- **Escala:** x3,5 de su tamaño original
- **Launch Angle:** 25 grados
- **Launch Speed:** 20 unidades
- **Damage:** 6 unidades

■ **Dash Attack:**

Este movimiento está diseñado para ejecutarse mientras el personaje corre, permitiendo un ataque rápido y efectivo. Su principal función es levantar al oponente del suelo, abriendo la posibilidad de continuar la ofensiva con ataques aéreos, y facilitando así transiciones fluidas en combate.



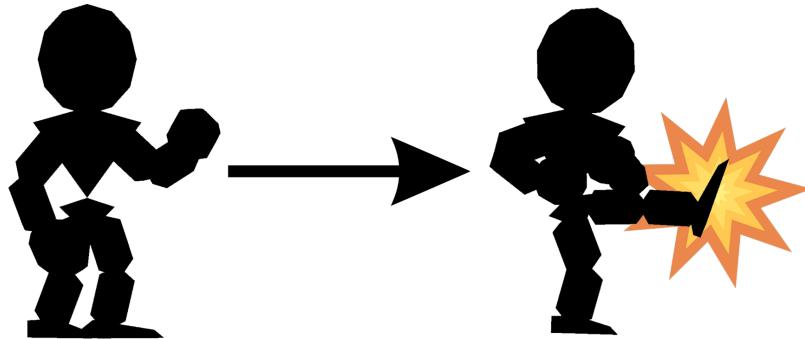
Movimientos de un Platformer Fighter - Dash Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 45 grados
- **Launch Speed:** 35 unidades
- **Damage:** 6 unidades

■ **Forward Tilt Attack:**

Este ataque está diseñado para crear distancia entre los jugadores, sin ser excesivamente fuerte. Su objetivo principal es empujar al oponente lejos, sin tener el potencial de causar un KO directo, pero proporcionando un espacio seguro para el jugador.



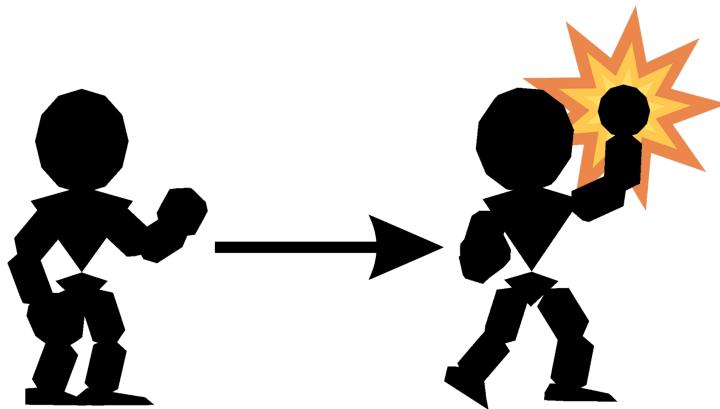
Movimientos de un Platformer Fighter - Forward Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 20 grados
- **Launch Speed:** 40 unidades
- **Damage:** 6 unidades

■ **Up Tilt Attack:**

El movimiento busca mantener al oponente en el aire, impidiendo que toque el suelo. La idea es que el jugador pueda utilizar este ataque para hacer "malabares" con el rival, manteniéndolo en una posición vulnerable.



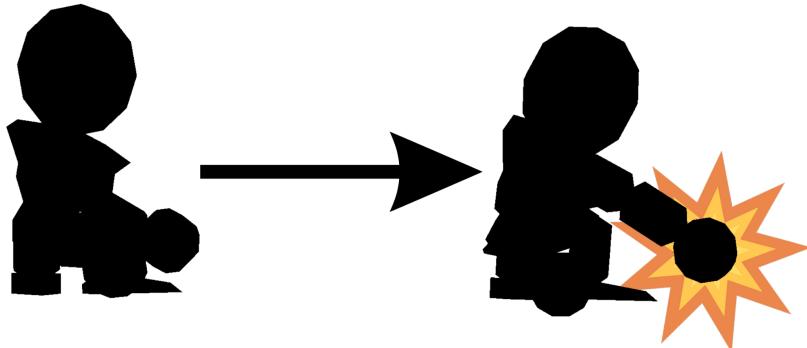
Movimientos de un Platformer Fighter - Up Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 80 grados
- **Launch Speed:** 20 unidades
- **Damage:** 4 unidades

■ **Down Tilt Attack:**

Este es un movimiento rápido y de baja potencia que tiene como objetivo levantar al oponente ligeramente del suelo. Se concibe como un ataque que se pueda repetir rápidamente, ideal para interrumpir al oponente y preparar combos.



Movimientos de un Platformer Fighter - Down Attack

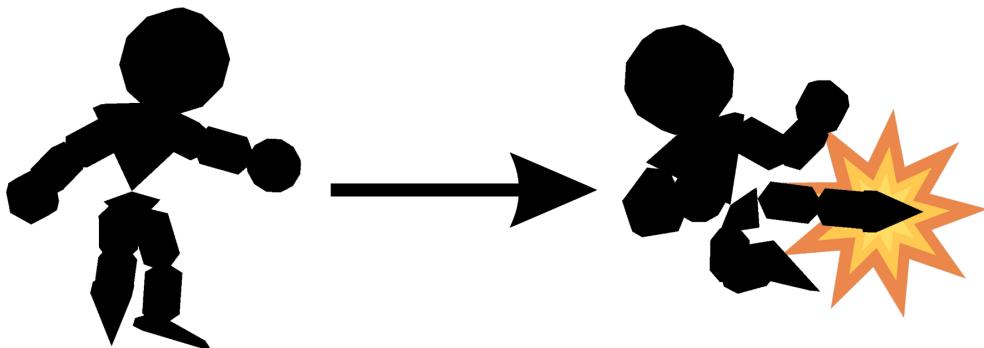
➤ **Stats base:**

- **Escala:** x3,5 de su tamaño original
- **Launch Angle:** 25 grados
- **Launch Speed:** 15 unidades

- **Damage:** 4 unidades
- **Movimientos de ataque aéreos**

■ **Neutral Aerial Attack:**

El propósito de este ataque aéreo es crear un espacio de seguridad alrededor del jugador, permitiéndole golpear al oponente mientras mantiene una zona segura. Aunque no es un movimiento muy fuerte, su utilidad radica en el control del espacio.



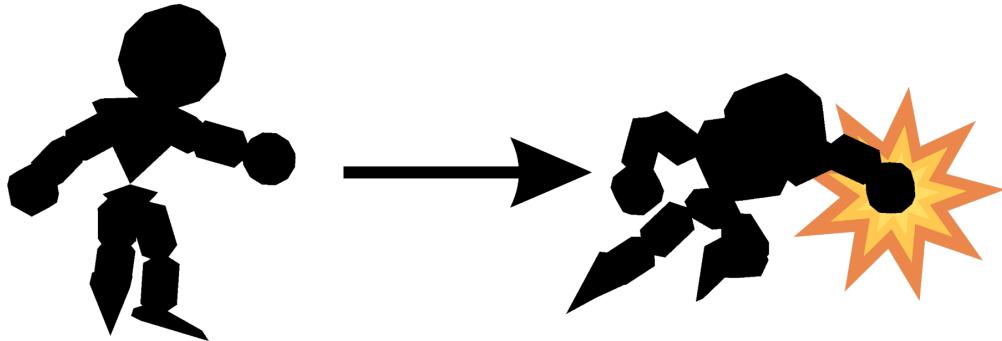
Movimientos de un Platformer Fighter - Neutral Aerial Attack

➤ **Stats base:**

- **Escala:** x2,5 de su tamaño original
- **Launch Angle:** 45 grados
- **Launch Speed:** 15 unidades
- **Damage:** 7 unidades

■ **Forward Aerial Attack:**

Un ataque aéreo que se asemeja a un movimiento *smash*, pero sin la potencia suficiente para un KO directo. Es más lento y medianamente poderoso, diseñado para ser un golpe deliberado en el aire.



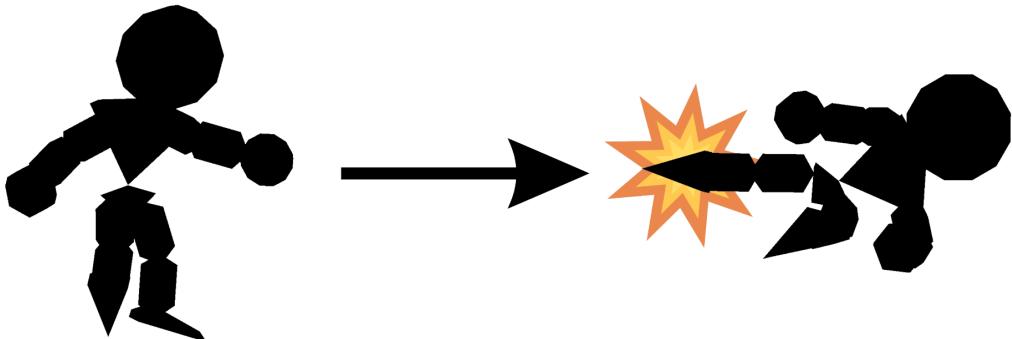
Movimientos de un Platformer Fighter - Forward Aerial Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 35 grados
- **Launch Speed:** 35 unidades
- **Damage:** 10 unidades

■ **Back Aerial Attack:**

Similar al *Forward Aerial*, pero con un enfoque en la velocidad y ligereza. Este ataque es más rápido y menos poderoso, ideal para mantener la presión mientras el jugador está en el aire.



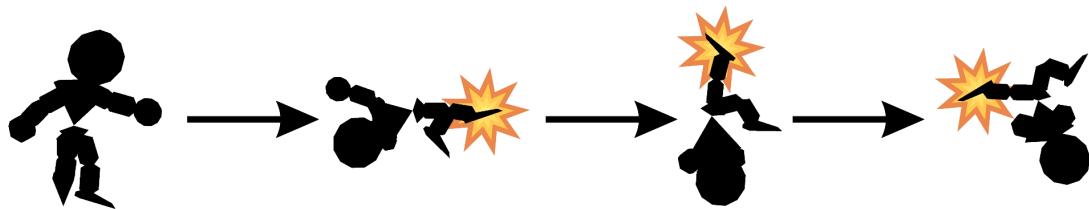
Movimientos de un Platformer Fighter - Back Aerial Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 150 grados
- **Launch Speed:** 11 unidades
- **Damage:** 7 unidades

■ **Up Aerial Attack:**

Este es un movimiento rápido y repetitivo que busca mantener al oponente en el aire, permitiendo al jugador realizar combos aéreos constantes. Aunque no es muy fuerte, su velocidad lo convierte en una herramienta eficaz para "*malabares*" aéreos.



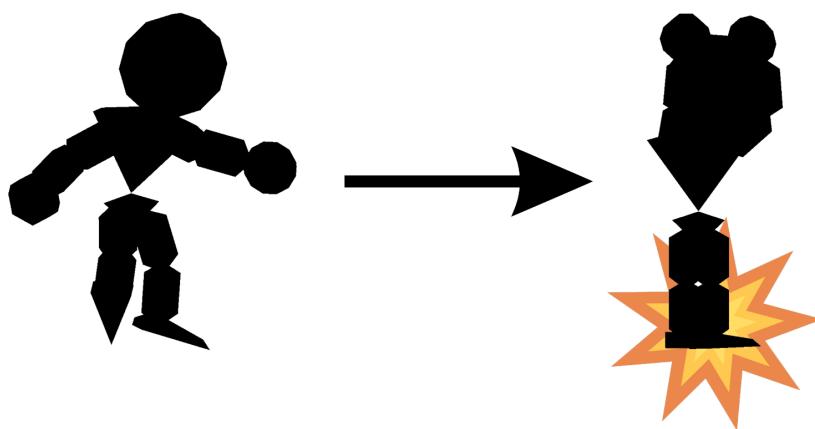
Movimientos de un Platformer Fighter - Up Aerial Attack

➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 90 grados
- **Launch Speed:** 10 unidades
- **Damage:** 8 unidades

■ **Down Aerial Attack:**

Un movimiento relativamente poderoso, diseñado para enviar al oponente hacia abajo con fuerza, creando una oportunidad de KO directo. Es un ataque que castiga fuertemente a los rivales que se encuentran debajo del personaje.



Movimientos de un Platformer Fighter - Down Aerial Attack

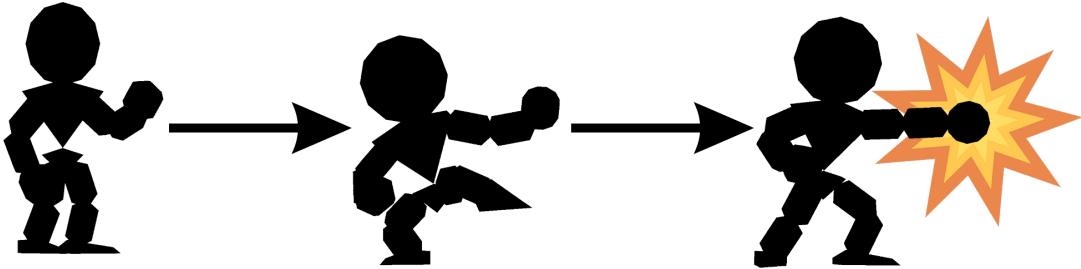
➤ **Stats base:**

- **Escala:** x4,5 de su tamaño original
- **Launch Angle:** 270 grados
- **Launch Speed:** 50 unidades
- **Damage:** 8 unidades

○ **Movimientos de ataque fuertes**

■ **Forward Smash Attack:**

Un ataque cargado lento pero extremadamente poderoso, diseñado para golpear con fuerza hacia adelante y buscar un KO. Su lentitud lo hace arriesgado, pero el daño causado es considerable.



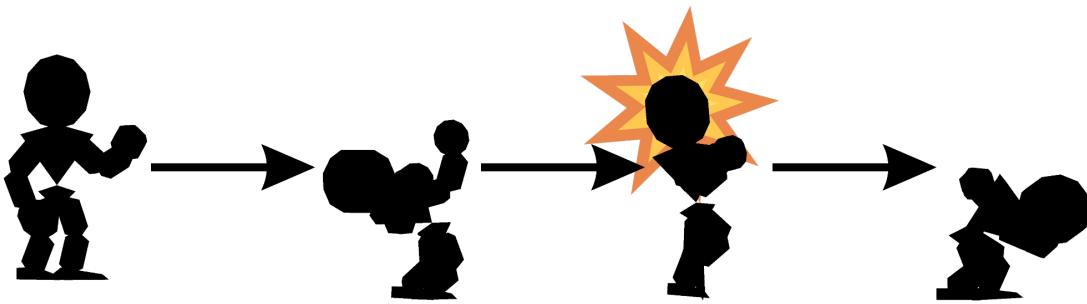
Movimientos de un Platformer Fighter - Forward Smash Attack

➤ **Stats base:**

- **Escala:** x6,5 de su tamaño original
- **Launch Angle:** 35 grados
- **Launch Speed:** 50 unidades
- **Damage:** 13 unidades

■ **Up Smash Attack:**

Este es un ataque cargado que se ejecuta rápidamente y con fuerza, con el objetivo de elevar a los rivales. Aunque tiene la capacidad de KO, su principal función es levantar al oponente y preparar posibles combinaciones posteriores.



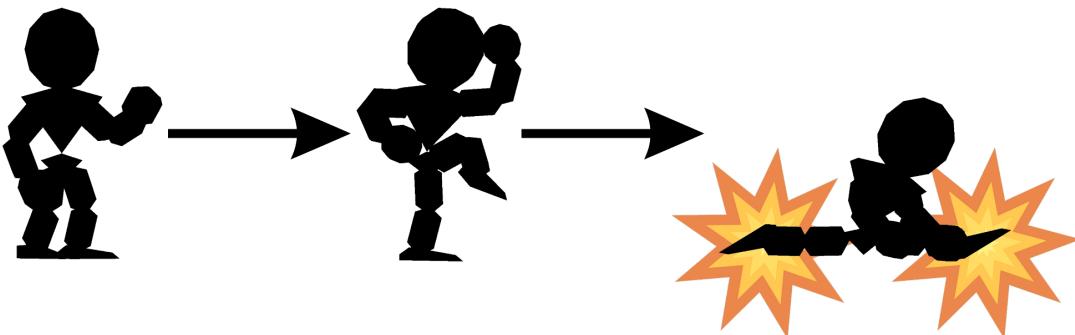
Movimientos de un Platformer Fighter - Up Smash Attack

➤ **Stats base:**

- **Escala:** x6,5 de su tamaño original
- **Launch Angle:** 90 grados
- **Launch Speed:** 45 unidades
- **Damage:** 14 unidades

■ **Down Smash Attack:**

Un movimiento cargado y fuerte que cubre un amplio espacio a su alrededor, ideal para mantener a los rivales alejados y ejercer control sobre la zona inmediata del personaje.



Movimientos de un Platformer Fighter - Down Smash Attack

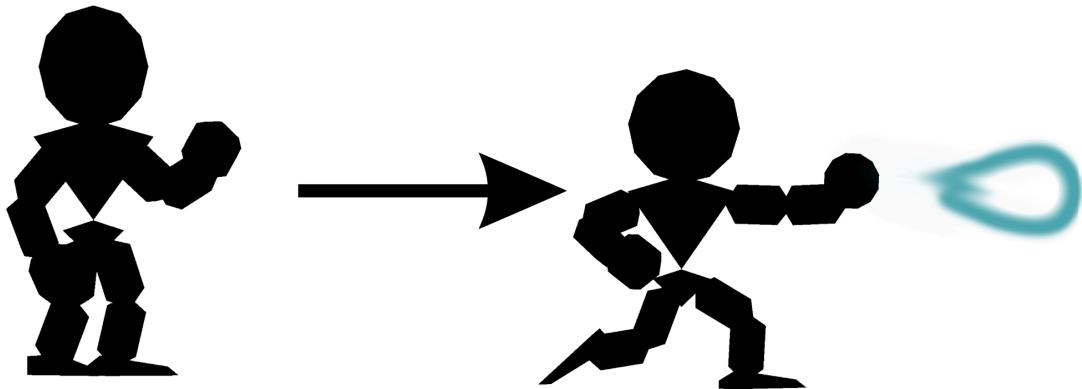
➤ **Stats base:**

- **Escala:** x5,5 de su tamaño original
- **Launch Angle:** 135 grados
- **Launch Speed:** 40 unidades
- **Damage:** 12 unidades

○ **Movimientos de ataque especiales**

■ **Neutral Special Attack:**

Este movimiento especial está diseñado para atacar al rival a distancia, causando un pequeño daño acumulativo sin lanzar al oponente muy lejos. Su propósito principal es distraer o presionar al rival, manteniéndolo en alerta y aumentando su daño poco a poco.



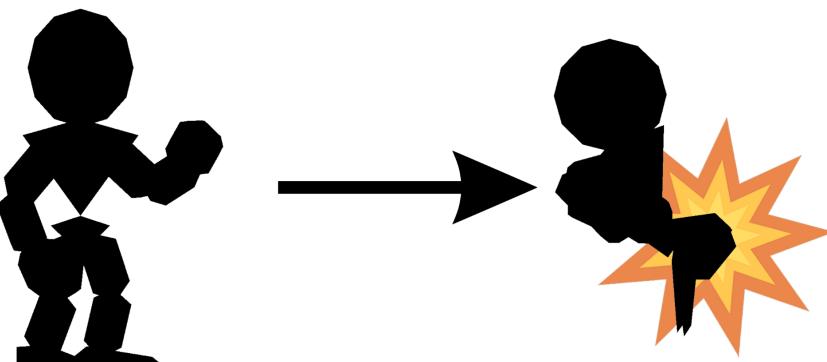
Movimientos de un Platformer Fighter - Neutral Special Attack

➤ **Stats base:**

- **Launch Angle:** 5 grados
- **Launch Speed:** 2 unidades
- **Damage:** 3 unidades

■ **Side Special Attack:**

Un ataque rápido en el que el personaje se lanza hacia el oponente con un rodillazo potente. Este movimiento cubre una buena distancia, lo que lo hace útil tanto para atacar como para recuperarse en situaciones de desventaja.



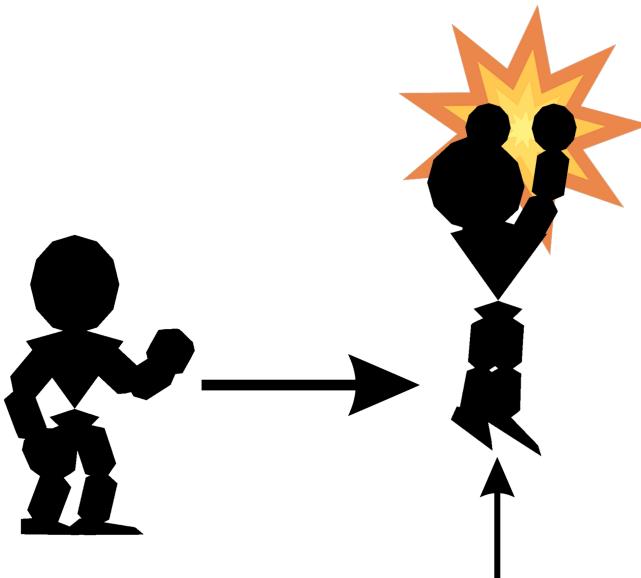
Movimientos de un Platformer Fighter - Side Special Attack

➤ **Stats base:**

- **Escala:** x5,5 de su tamaño original
- **Launch Angle:** 45 grados
- **Launch Speed:** 35 unidades
- **Damage:** 8 unidades

■ **Up Special Attack:**

Principalmente utilizado como una herramienta de recuperación cuando el personaje está cayendo o agotado de otros ataques. Sin embargo, si un oponente se encuentra debajo, el movimiento puede asegurar un golpe poderoso.



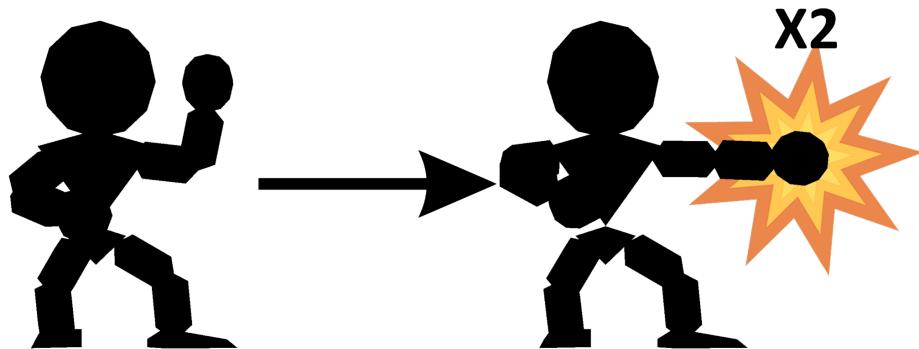
Movimientos de un Platformer Fighter - Up Special Attack

➤ **Stats base:**

- **Escala:** x3,5 de su tamaño original
- **Launch Angle:** 90 grados
- **Launch Speed:** 30 unidades
- **Damage:** 10 unidades

■ **Down Special Attack:**

Este es un movimiento de "counter" que solo se ejecuta completamente si el personaje es atacado. Cuando es activado, el personaje responde al ataque del rival, devolviéndole el golpe con el doble de daño.



Movimientos de un Platformer Fighter - Down Special Attack

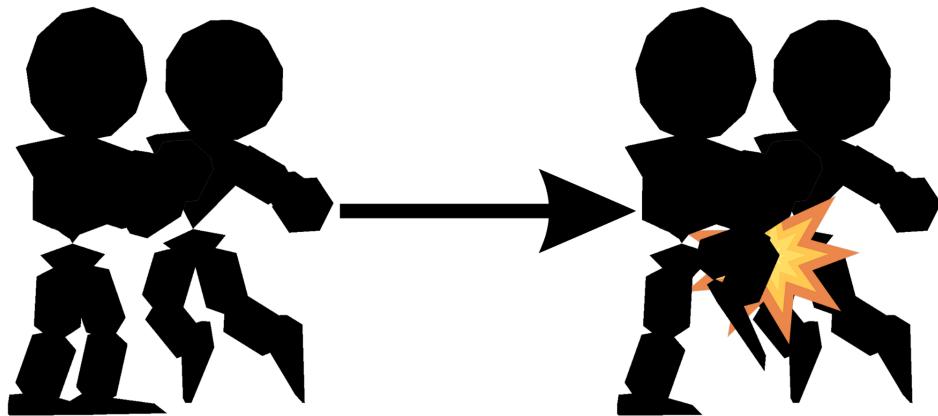
➤ **Stats base:**

- **Escala:** x6 de su tamaño original
- **Launch Angle:** 45 grados
- **Launch Speed:** 20 unidades * el *launch speed* del ataque rival * 0,1
- **Damage:** 4 unidades * el *damage* del ataque rival * 0,1

○ **Movimientos de agarre**

■ **Pummel:**

Un rápido rodillazo que acumula daño en el oponente sin ningún efecto adicional. La idea detrás de este movimiento es su velocidad, permitiendo que el daño se acumule rápidamente.



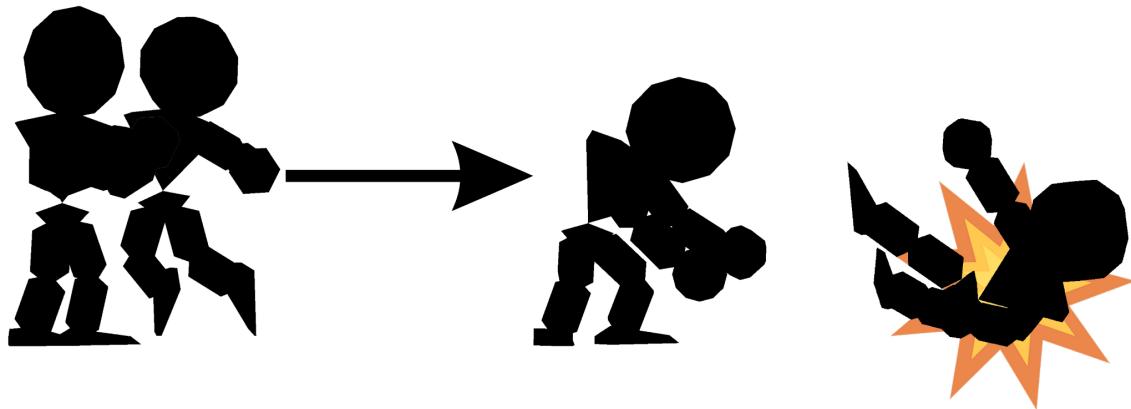
Movimientos de un Platformer Fighter - Pummel

➤ **Stats base:**

- **Escala:** x1,1 de su tamaño original
- **Launch Angle:** 0 grados
- **Launch Speed:** 0 unidades
- **Damage:** 2 unidades

■ **Forward Throw:**

Un lanzamiento hacia adelante usando una técnica de judo para cargar al oponente sobre su espalda y lanzarlo al suelo. Este movimiento está diseñado para crear espacio entre los personajes a la vez de que el rival salga volando.



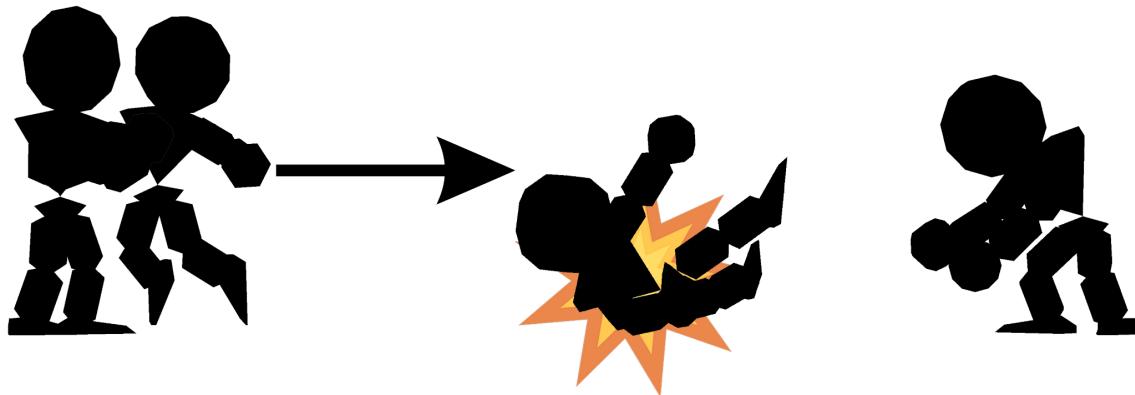
Movimientos de un Platformer Fighter - Forward Throw

➤ **Stats base:**

- **Launch Angle:** 65 grados
- **Launch Speed:** 30 unidades
- **Damage:** 7 unidades

■ **Back Throw:**

Un agarre seguido de una voltereta hacia atrás que termina con una patada poderosa, lanzando al oponente hacia atrás. Este movimiento también busca cubrir espacio y puede ser lo suficientemente fuerte como para hacer KO.



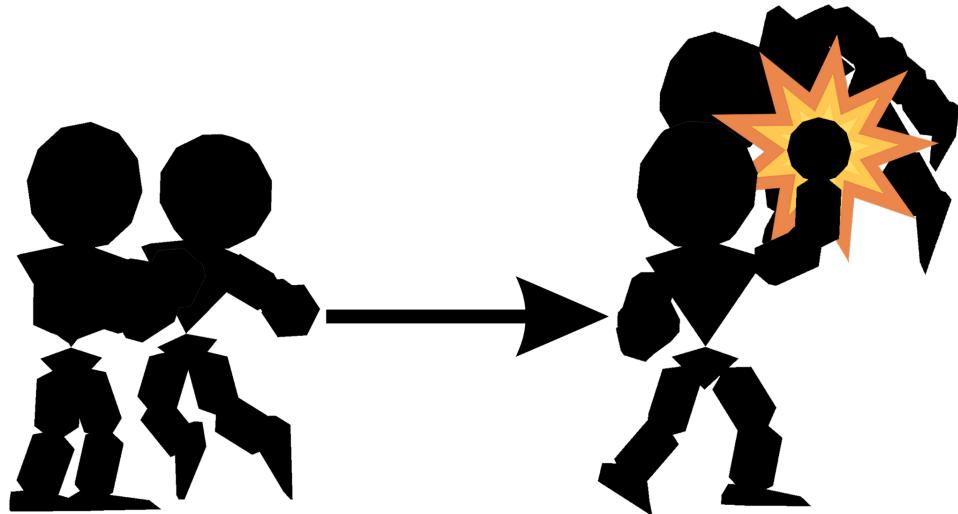
Movimientos de un Platformer Fighter - Back Throw

➤ **Stats base:**

- **Launch Angle:** 65 grados
- **Launch Speed:** 35 unidades
- **Damage:** 8 unidades

■ **Up Throw:**

Un puñetazo en la barriga del oponente que lo lanza hacia arriba. Este ataque es ideal para poner al rival en una posición desventajosa en el aire y tiene el potencial de causar un KO.



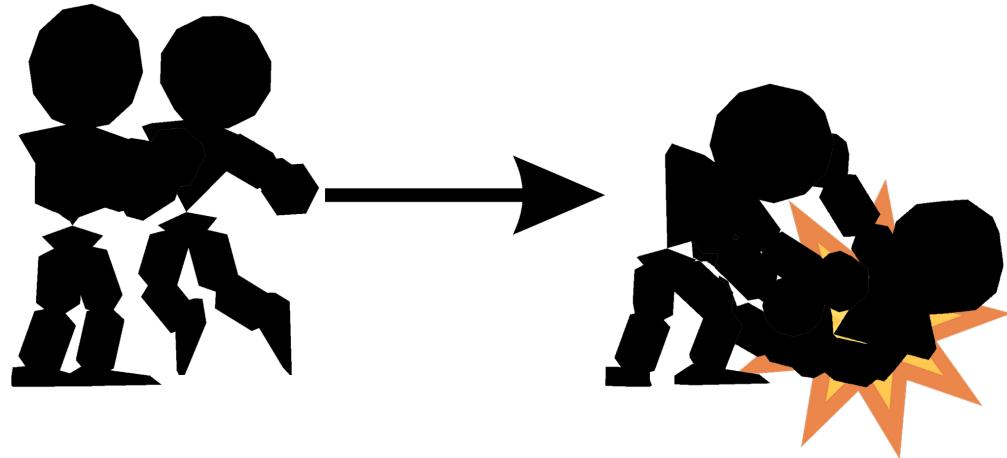
Movimientos de un Platformer Fighter - Up Throw

➤ **Stats base:**

- **Launch Angle:** 90 grados
- **Launch Speed:** 35 unidades
- **Damage:** 7 unidades

■ **Down Throw:**

Un lanzamiento en el sitio al estilo judo que hace que el oponente rebote ligeramente hacia arriba. Es un movimiento ideal para iniciar combos aéreos.



Movimientos de un Platformer Fighter - Down Throw

➤ **Stats base:**

- **Launch Angle:** 75 grados
- **Launch Speed:** 20 unidades
- **Damage:** 6 unidades

○ **Movimientos de defensivos**

■ **Escudo:**

Una postura de protección básica que permite resistir múltiples golpes. Es fundamental para la defensa en situaciones de presión.



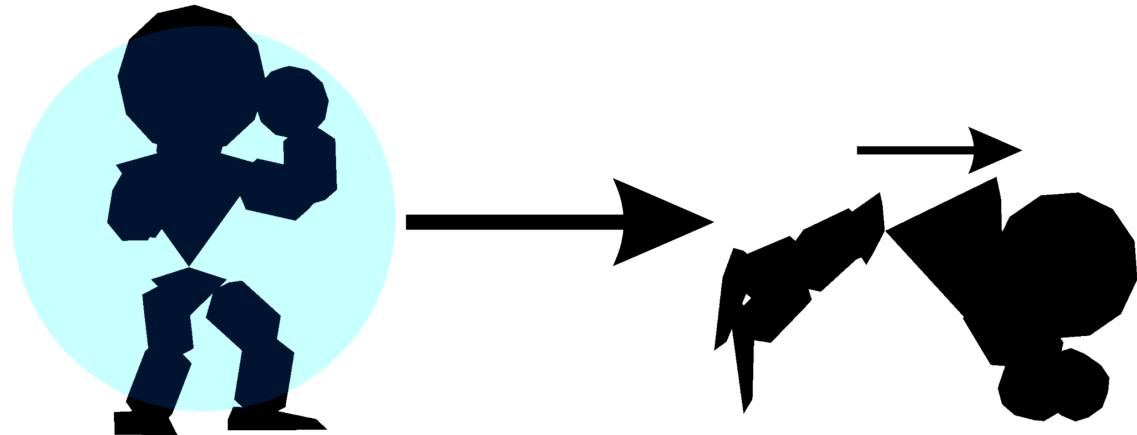
Movimientos de un Platformer Fighter - Escudo

➤ **Stats base:**

- **Resistencia:** 120 unidades de *damage*

■ **Voltereta delantera:**

Una voltereta rápida hacia adelante que cubre espacio, una técnica común en artes marciales para posicionarse rápidamente.



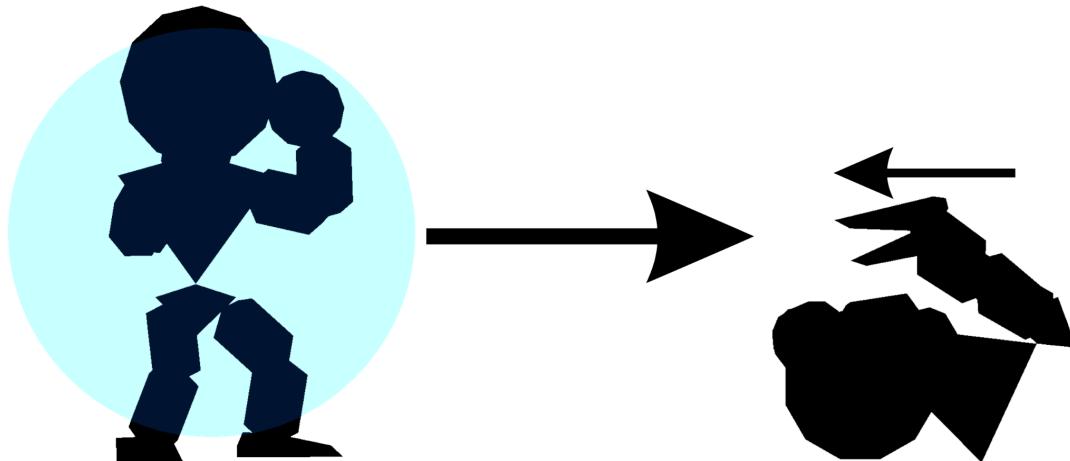
Movimientos de un Platformer Fighter - Voltereta delantera

➤ **Stats base:**

- **Distancia recorrida:** 3 unidades de espacio

■ **Voltereta trasera:**

Una voltereta hacia atrás que también sirve para cubrir espacio, aunque es más difícil de ejecutar debido a la necesidad de girar completamente de espaldas y colocarse rápidamente.



Movimientos de un Platformer Fighter - Voltereta trasera

➤ **Stats base:**

- **Distancia recorrida:** 3 unidades de espacio

5.1.3.3. Modos

● Para un jugador

- **Modo aventura:** Modo de juego de un jugador en el cual se selecciona un personaje para ir recorriendo unos niveles estilo plataformas, pero con las mecánicas propias del juego, donde enemigos le vendrán de frente para pararle los pies.

Para superar el nivel, el personaje deberá llegar hasta un punto B establecido en el mapa. Durante el camino, el personaje se deberá parar y completar algunas hordas de enemigos que aparecerán de vez en cuando para derrotarlo y pararle los pies.

En algunos niveles se le presentarán *mini jefes*, versiones más fuertes que los rivales normales, los cuales deberá derrotar para seguir.

Para finalizar el modo, el personaje deberá derrotar al jefe final del modo.

Este modo da al jugador una cantidad de 3 a 5 vidas y con diferentes niveles de dificultad, los cuales harán que los enemigos sean más fuertes a la hora de atacar y más difíciles de derrotar.

El tiempo de juego de este modo puede ser de entre 10 a 20 minutos.

- **Home-run contest:** Modo de juego de un jugador en el cual el personaje deberá golpear a un *mob estático* durante unos cuantos segundos, para finalizar dándole un golpe fuerte antes de que se acabe el tiempo.

Al dar el último golpe fuerte, este *mob* saldrá volando y el juego irá mostrando la distancia que está recorriendo este hasta aterrizar y pararse del todo, así teniendo una puntuación final con ese personaje.

Dependiendo de la distancia que haya recorrido el *mob*, el personaje obtendrá una mejor o peor clasificación en el cual se le clasificará de 0 a 5 estrellas.

- **Maestría de personaje:** Modo de juego donde el jugador tendrá tres escenarios por personaje, los cuales retará a romper unas tablas de madera repartidas por dicho escenario y recoger la máxima cantidad de un objeto que se le sumarán a la puntuación final en el menor tiempo posible.

Dependiendo de todo lo anterior, el personaje obtendrá una mejor o peor clasificación en el cual se le clasificará de 0 a 5 estrellas.

En este modo el jugador solo tendrá 1 vida y un tiempo máximo para poder realizar la prueba.

● Multijugador

- **Modo VS:** En este modo, de entre 2 y 4 jugadores, se enfrentarán entre ellos o en equipo para derrotar al rival y así ganar la partida.

Para derrotar a los rivales, los jugadores deberán atacar a su oponente, haciendo así que incremente su *daño* y obtenga un mayor *knockback*, haciendo así que cuando salga del escenario y alcance el límite del mapa y reciba un KO, este muera. El jugador que haya realizado el último golpe antes de que el rival tocara el suelo y de matarlo, recibirá un punto, mientras que el rival muerto recibirá un punto negativo.

En este modo, los jugadores pueden elegir el escenario en el cual se van a pelear, además de las reglas, las cuales son:

- **Vidas:** como su nombre indica, permite al jugador elegir cuántas vidas tendrán todos los jugadores durante la partida. Si este valor se pone en *infinito*, el ganador se elegirá por la cantidad de puntos acumulados al final de la partida.
 - **Tiempo:** como su nombre indica, permite al jugador saber cuánto tiempo durará la partida. Si este valor se pone en *infinito*, el ganador será el que aún conserve vidas.
- En caso de tener este valor definido, el ganador será elegido por la cantidad de vidas que tenga cada uno. Si los jugadores tienen la misma cantidad de vidas, se elegirá el ganador por el *damage* más bajo.
- **Fuego amigo:** permite cambiar si los compañeros de equipo podrán golpearse entre ellos o no.
 - **Daño de inicio:** permite seleccionar cuánto daño dispondrá al inicio de la partida y después de perder una vida cada jugador por separado, esto solamente se podrá seleccionar en la pantalla de *selección de personaje*.

5.1.3.4. Diseño de nivel

- **Multijugador**

- **Diseño básico de escenario:**

En este juego todos los escenarios son diferentes entre ellos, pero todos comparten unos elementos en común los cuales no se diferenciarán entre ellos:

- **Plataforma central:** Es la **zona central** representada con el **color azul**, donde la mayor del tiempo los jugadores se la pasan peleando. Es la parte más sólida del escenario la cual aquí pueden ocurrir interacciones entre los jugadores mientras se golpean en el suelo, rebotar con el suelo o las paredes después de recibir un golpe o restaurar el uso de algunas habilidades, normalmente siendo estas los saltos y el especial hacia arriba.

En la plataforma central o plataformas solidas suele haber una **zona agarrible**, representada con el **color amarillo**, la cual permite al jugador cogerse de ella para prevenir una caída inminente y así darle otra oportunidad para pelear, además de restaurarle el uso del salto doble, el resto de movimientos aereos y especiales.

En esta zona, el jugador, al haberse agarrado al inicio, dispone de una **invencibilidad** donde cualquier golpe que reciba será anulado automáticamente, además de poder volver al escenario de diferentes maneras, entre ellas las cuales se encuentran:

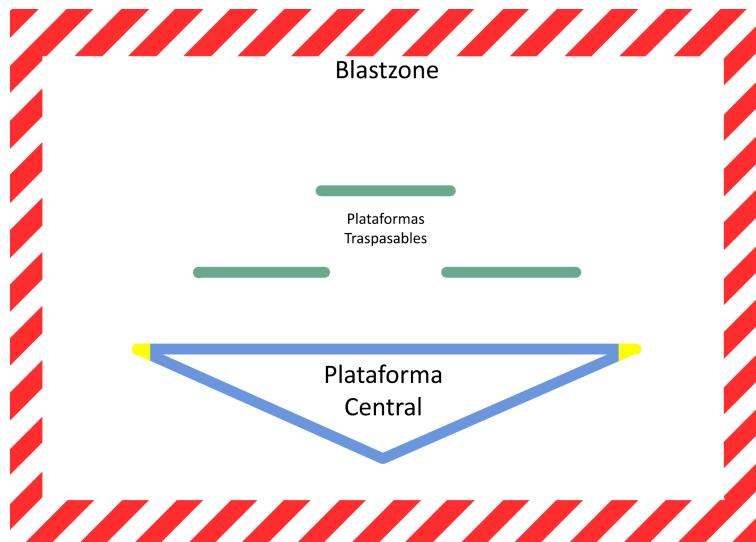
- **Subir en el sitio:** en este caso el jugador recibirá unos *frames* extras en los cuales será **intangible** para así que no pueda recibir el golpe.
 - **Dar una voltereta hacia delante:** parecido a subir en el sitio, pero con más distancia y más frames los cuales será **tangible** y deberá de esperar para poder hacer el siguiente movimiento.
 - **Volver a saltar:** no recibe intangibilidad alguna y simplemente usa el salto extra que tenía para poder elegir su siguiente movimiento.
 - **Dejarse caer:** se puede dejar caer para cubrir espacio y, si el enemigo sale del escenario, golpearle a él en vez de subir directamente.

- **Pegar antes de subirse:** en este caso, el más parecido a subir en el sitio, pero con el añadido de poder hacer un pequeño golpe para cubrir espacio y así que los rivales no le golpeen. Al igual que *la voltereta hacia delante*, este tiene más *frames* los cuales será **tangible** y le podrán golpear.
- **Plataformas traspasables:** Estas plataformas marcadas en **color verde** suelen cubrir una parte del escenario la cual suelen ser de ayuda, para cubrir más zona aérea y que los jugadores puedan restaurar sus habilidades o para tomar distancia entre ellos.

Tienen la peculiaridad que son traspasables tanto por arriba como por abajo, permitiendo atravesarlas en cualquier momento al igual que hacer algunos ataques siempre que no se aterricen en ellas.

En algunos casos estas plataformas pueden ser móviles o **pueden ser el escenario entero**, estas teniendo **zonas agarrables**.
- **Blastzone o Zona de muerte:** Son los bordes del escenario marcados en **color rojo** que delimitan el área de pelea, haciendo que si estos superan o tocan este límite mueran y pierdan una vida o un punto, dependiendo del modo en el cual se esté jugando.

Cuando un jugador llega al inicio de esta zona, la cámara no le seguirá y marcará la zona en la cual está personaje. En caso de que el jugador llegue al final de esta zona, será automáticamente *ko*, es decir, **morirá**.



Diseño de nivel - Partes de un escenario

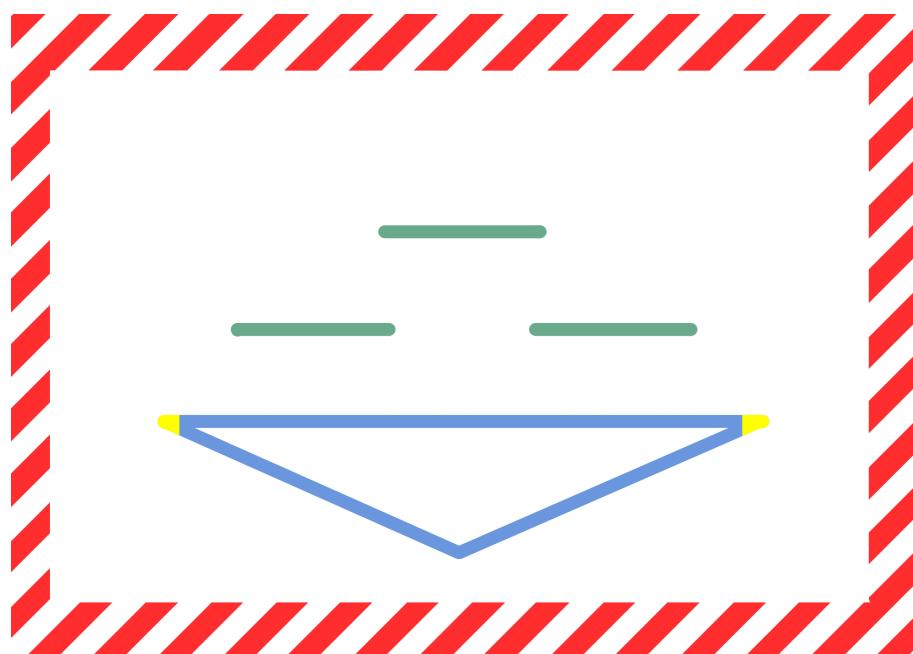
○ **Tipos de escenarios:**

En este género es común encontrar ciertos tipos de escenarios que tienden a representarse de manera similar, algunos de ellos parecidos entre ellos, dado que esta forma ha demostrado ser la más funcional y efectiva en términos de jugabilidad, en este juego también se representarán de la misma forma:

- **Tipo de escenario:** 3 plataformas flotantes.
- **Descripción del nivel:** este escenario presenta una disposición de una plataforma central sólida y tres plataformas traspasables estacionarias que flotan sobre ella. Estas plataformas proporcionan puntos estratégicos para que los jugadores se coloquen y ejecuten ataques desde diferentes alturas.

La parte de debajo del escenario, en este caso, tiene forma rectangular que da más espacio a los jugadores para poder acceder por debajo de ella o al otro lado del escenario si ellos quieren.

Además, la disposición del escenario está diseñada para fomentar la acción intensa y estratégica, con amplio espacio para el movimiento y el combate mientras se mantiene un equilibrio entre la accesibilidad y la complejidad del entorno de juego.

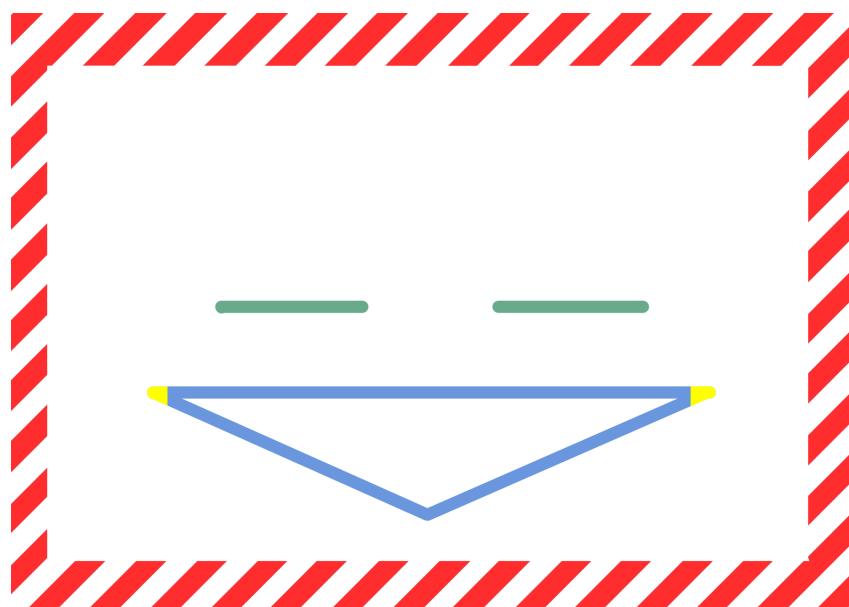


Diseño de nivel - 3 plataformas flotantes

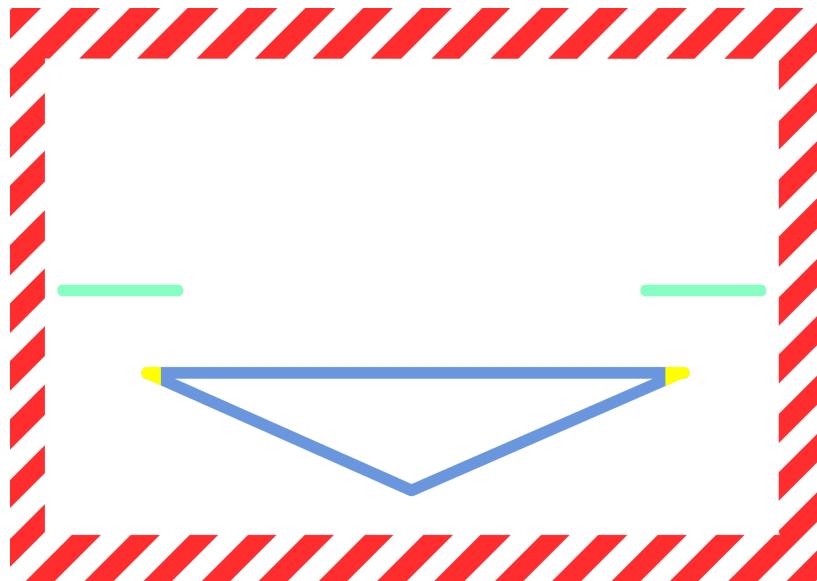
- **Tipo de escenario:** 2 plataformas flotantes.
- **Descripción del nivel:** este escenario presenta una disposición de una plataforma central sólida y dos plataformas traspasables estacionarias que flotan sobre ella. Estas plataformas proporcionan puntos estratégicos para que los jugadores se coloquen y ejecuten ataques seguidos aéreos para que los rivales lo tengan difícil de volver al escenario.

La disposición de este escenario está diseñado para que los jugadores tengan la sensación de disponer *espacio* para poder moverse con libertad.

Por otro lado, este tipo de escenario suele tener dos tipos de disposiciones para las plataformas, uno el cual está más centrado, para un tipo de pelea más parecido al de 3 plataformas y otro en el cual las plataformas están más cerca de la *blastzone*, las cuales, a personaje que lo tengan más difícil de realizar un *ko* lo tendrán más fácil para hacerlo hacia los lados.



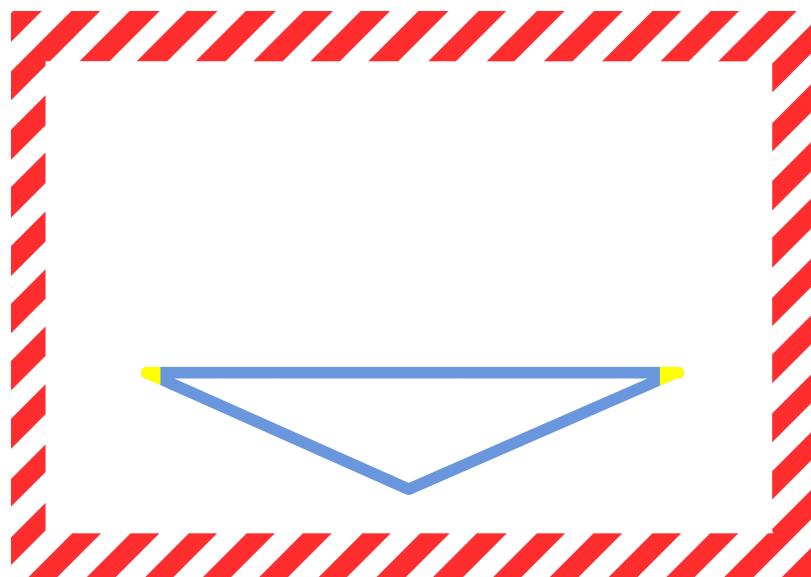
Diseño de nivel - 2 plataformas flotantes variante A



Diseño de nivel - 2 plataformas flotantes variante B

- **Tipo de escenario:** Escenario plano.
- **Descripción del nivel:** este escenario presenta una disposición de una plataforma central sólida sin disponer de ninguna plataforma flotante.

La disposición de este escenario está diseñado para que los jugadores tengan una batalla cara a cara, en la cual ninguno de los dos personajes intente huir del rival constantemente. Además, este escenario suele tener batallas muy intensas entre jugadores ya que la lejanía entre ellos en casi todo momento será realmente nula.



Diseño de nivel - escenario plano

- **Un solo jugador**

- **Maestría de personajes:**

- **Círcito 1:** Este circuito está diseñado para que los jugadores practiquen y perfeccionen las habilidades básicas del personajes, guiándolos a través de diferentes *micro*-desafíos. El objetivo es destruir todas las dianas en el menor tiempo posible, utilizando las distintas habilidades del personaje en escenarios específicos.

El circuito se divide en varias zonas, cada una enfocada en una habilidad particular:

1. Zona de aparición - Prueba de saltos y altura

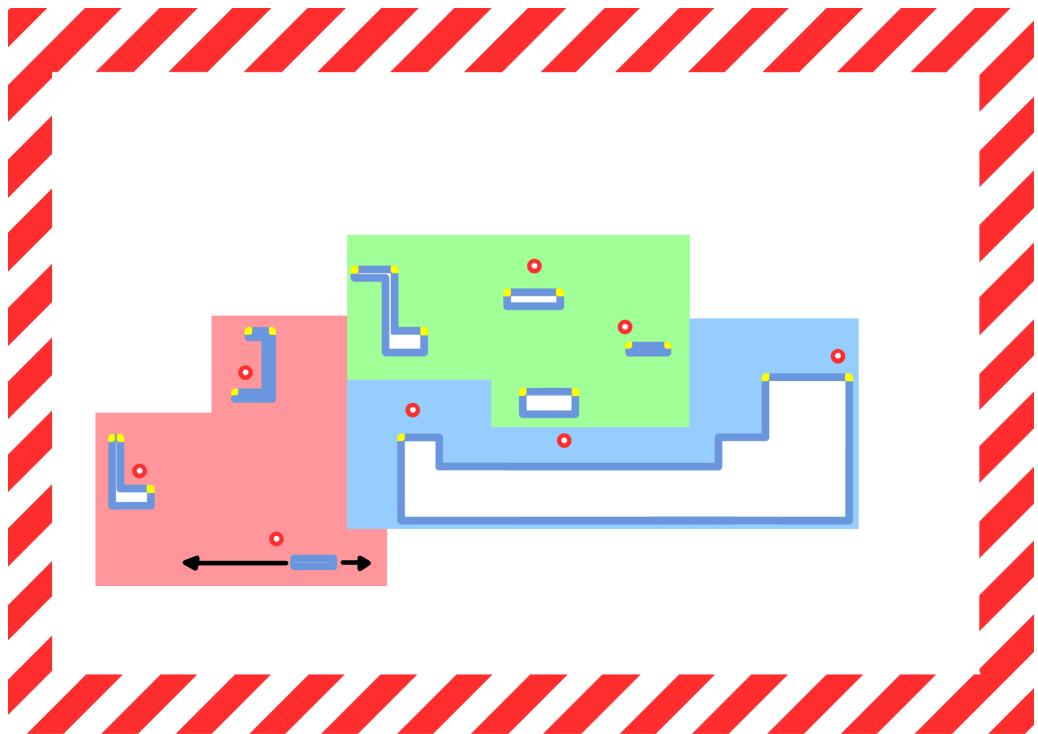
En esta primera sección, marcada con un color **verde**, el jugador comienza en un conjunto de plataformas. Aquí, el objetivo es permitir que el jugador se familiarice con la altura máxima del salto del personaje y cómo puede moverse entre diferentes niveles de plataformas sin tener peligro alguno de morir. Esta zona está diseñada para que el jugador entienda la relación entre el salto y la gravedad, ajustando sus movimientos para alcanzar cada plataforma de la manera más cómoda posible.

2. Zona inferior - Prueba de Velocidad

En esta parte del circuito, marcada con un color **azul**, las dianas están situadas a nivel del suelo. La disposición de las dianas exige que el jugador se desplace rápidamente para obtener el menor tiempo posible. El diseño de esta zona pone a prueba la velocidad y la capacidad de respuesta del personaje, ayudando al jugador a evaluar cuán rápido puede moverse por el escenario.

3. Zona de plataformas alejadas - Control de saltos y gravedad

La última sección del circuito, marcada con un color **rojo**, presenta un desafío más técnico, con plataformas distantes que requieren saltos precisos. Aquí, el jugador debe combinar lo aprendido en las secciones anteriores para alcanzar las dianas sin caer. Esta zona es clave para que el jugador domine el control de los saltos largos y entienda cómo la gravedad afecta los movimientos del personaje.



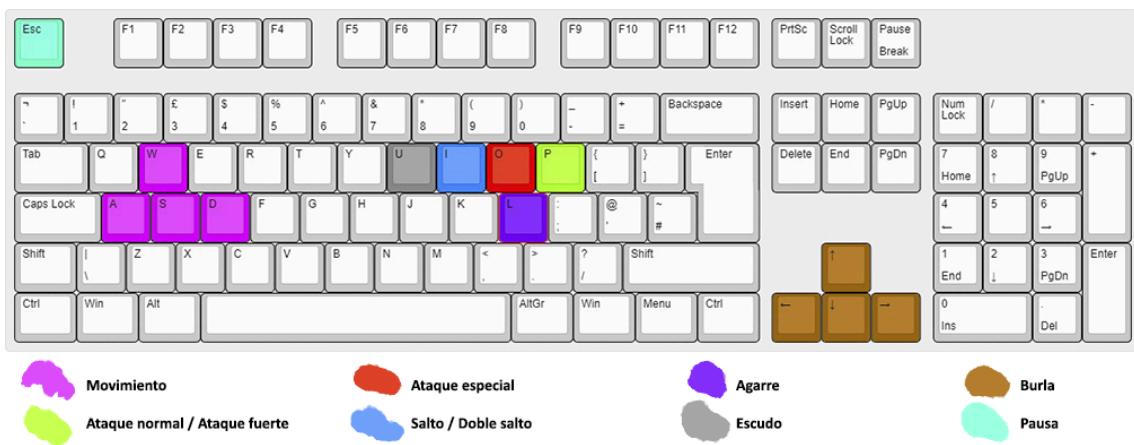
Diseño de nivel - Circuito 1

5.1.3.5. Controles

A continuación se presenta como se configura el sistema de entrada para el juego. Pese a ello hay que tener en cuenta los siguientes puntos:

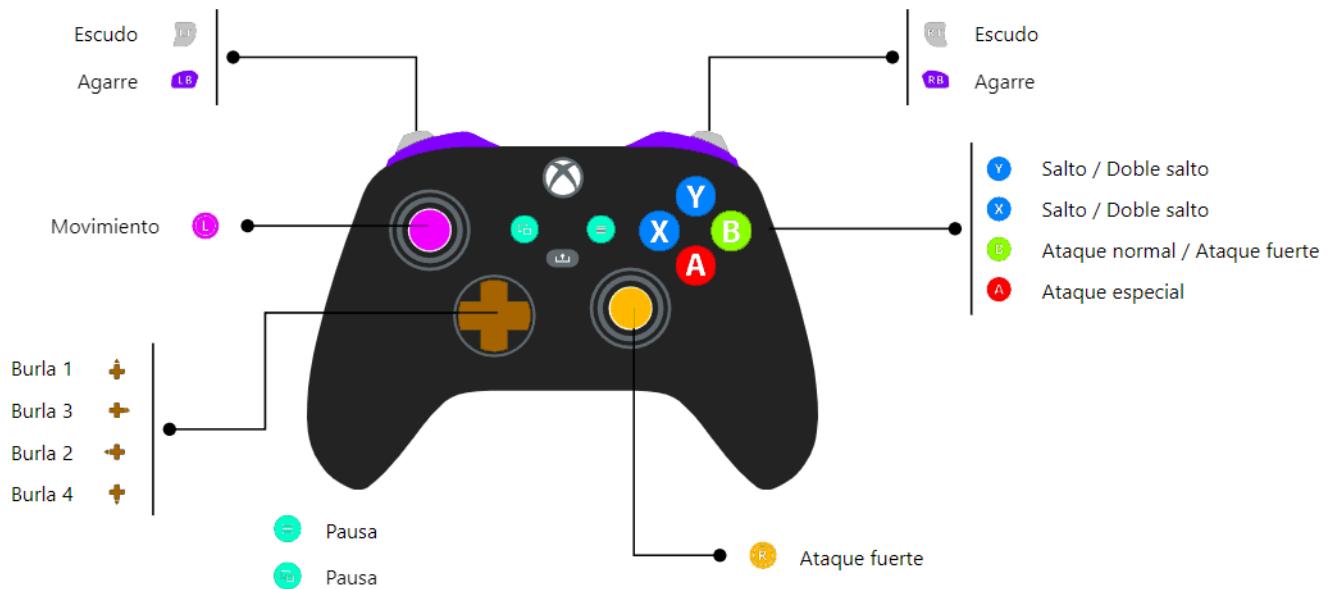
- Aunque es posible jugar con teclado, el proyecto se ha diseñado preferentemente para ser jugado con mando, debido a que algunas mecánicas no se podrán ejecutar de manera óptima utilizando solo el teclado.
- Para cubrir el mayor público posible, el proyecto es compatible con una variedad de mandos, incluyendo el de Xbox, PS4, PS5, el Pro Controller de Nintendo Switch y el mando de GameCube mediante un adaptador específico para PC.
- La distribución de botones entre los mandos de Xbox, Pro Controller de Nintendo Switch, PS4 y PS5 es la misma, facilitando la experiencia consistente para los jugadores que cambien de un dispositivo a otro.

Asignación de controles de teclado



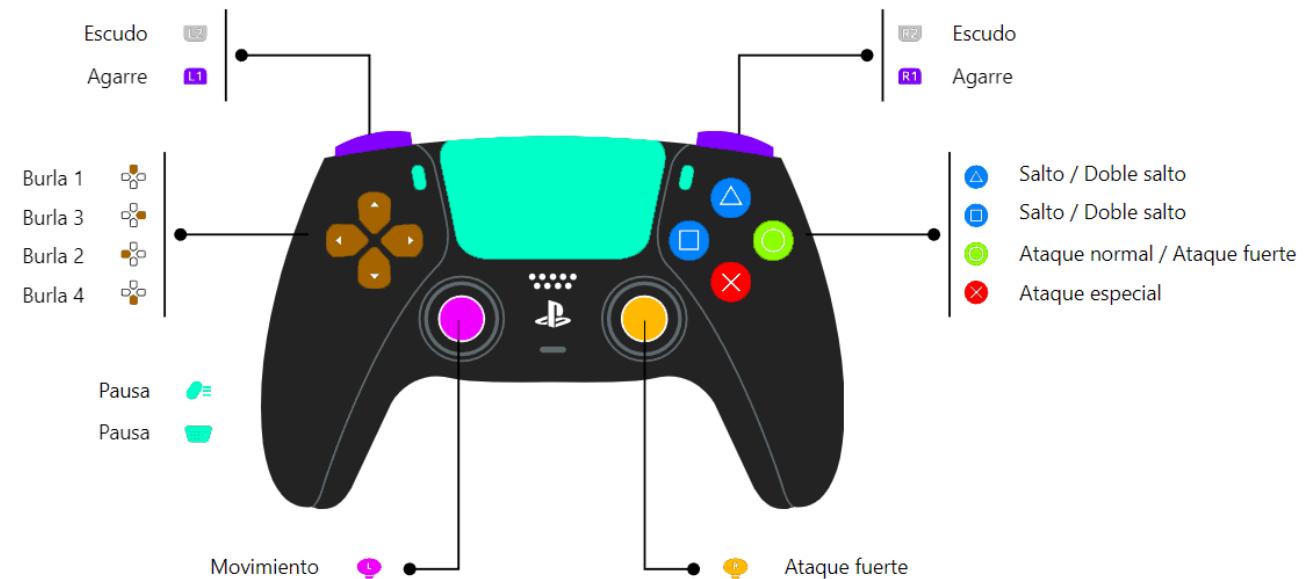
Controles - Asignación de controles de teclado

Asignación de controles de Xbox



Controles - Asignación de controles de Xbox

Asignación de controles de PS4/PS5



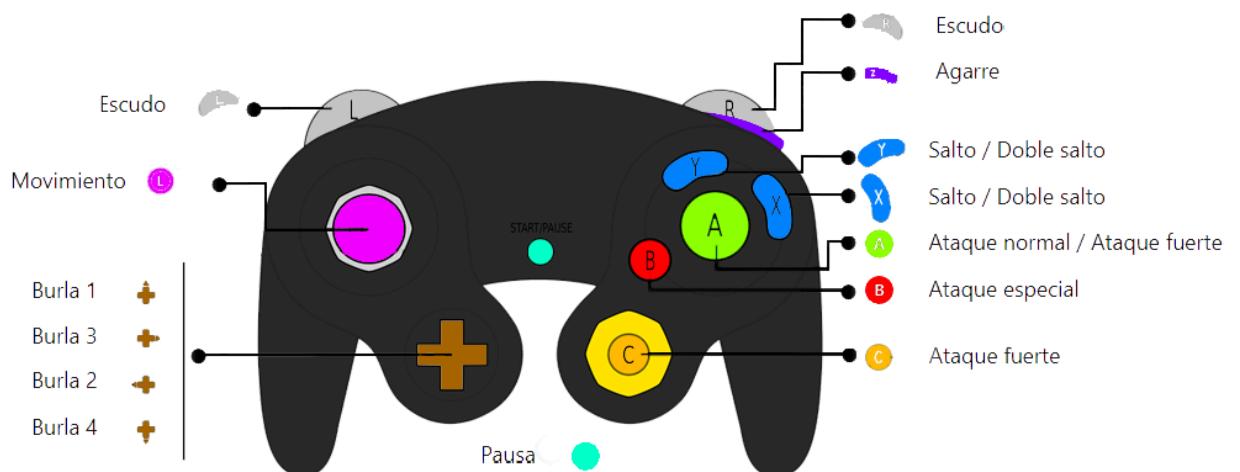
Controles - Asignación de controles de PS4/PS5

Asignación de controles de Pro Controller de Nintendo Switch



Controles - Asignación de controles de Pro Controller de Nintendo Switch

Asignación de controles de GameCube



Controles - Asignación de controles de GameCube

5.2. Desarrollo del prototipado

5.2.1. Objetivos del prototipo

1. Validar las mecánicas básicas del combate y su coherencia con el diseño

El prototipo debe confirmar que las mecánicas fundamentales del juego, como los ataques, movimientos y colisiones, funcionan correctamente y se alinean con las ideas del diseño. Es crucial comprobar que las hitboxes respondan con precisión, que los ataques tengan el impacto deseado y que los diferentes movimientos del personaje, como neutral attacks y smash attacks, se integren de manera fluida y coherente con las físicas del juego.

2. Comprobar la fluidez en la jugabilidad y la responsividad de los controles

No solo se busca verificar que los controles respondan de manera precisa, sino también que el personaje se sienta ágil y fluido durante la partida. El jugador debe tener una experiencia intuitiva, con movimientos que respondan inmediatamente a las órdenes del mando, ya sea de Xbox, PS4/PS5 o Nintendo Switch. La sensación de peso, la gravedad, y la inercia del personaje al moverse y saltar deben ser consistentes y agradables, permitiendo una jugabilidad rápida y reactiva.

3. Garantizar la compatibilidad con diferentes controladores y ajustar la experiencia para el teclado

Un objetivo clave es probar la versatilidad del juego con diferentes mandos, como los de Xbox, PS4/PS5, Pro Controller de Nintendo Switch y, con un adaptador, el mando de GameCube. Además, se deben identificar las limitaciones del teclado para garantizar que los jugadores que no dispongan de un mando puedan ejecutar las mecánicas esenciales del juego, aunque se recomienda su uso preferentemente con mando.

4. Detectar problemas y áreas de mejora en las mecánicas y controles

A través de iteraciones del prototipo, se buscará identificar errores o áreas que requieran ajustes en las mecánicas de combate, físicas, o controles. La identificación temprana de estos problemas permitirá refinar el diseño y optimizar la experiencia de juego para futuras pruebas con usuarios externos.

5.2.2. Proceso de creación

Para el desarrollo de este proyecto se ha hecho uso de diversas herramientas y programas para llevarlo a cabo. A continuación, se detallarán las principales herramientas empleadas, junto con una explicación paso a paso de las tareas realizadas más relevantes en cada una, acompañadas de imágenes que ilustran el resultado.

- **Adobe Photoshop:** usado para la edición y creación de sprites destinados a la interfaz del juego. Los principales sprites en el cual se han usado ha sido para los *portraits* de los personajes.

Como manera temporal, se usó como referencia los *portraits* en batalla de Super Smash Bros. Ultimate, en el cual consta de un recuadro con el personaje, esté sobresaliendo de dicho recuadro.

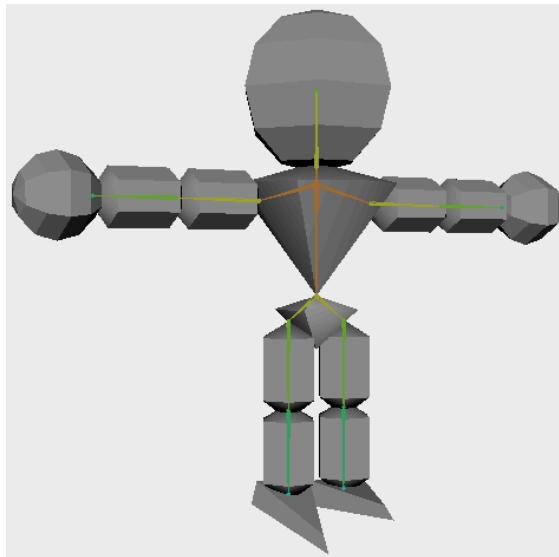


Proceso de creación - Adobe Photoshop

- **Autodesk Maya:** usado para la creación y animación de modelos destinados a usarse en el proyecto. Explicado paso por paso de lo que se ha realizado tendríamos:

- **Modelo base de personaje:** para poder hacer ataques y mecánicas más específicas necesitaba de un personaje, con lo cual, cree un modelo básico el cual fuera lo suficientemente funcional.

Por la parte del modelo se creó un modelo con un formato *low poly*, de unos 300 polígonos, inspirado en antiguos modelos de la Nintendo 64 y Playstation 1, el cual fuera sencillo de añadirle un esqueleto y animarlo.



Proceso de creación - Modelo base de personaje

En cuanto a las animaciones del personaje, se han seguido las pautas previamente mencionadas, con movimientos que hacen referencia a otros personajes de “*Platform Fighters*”. Este personaje ha sido diseñado principalmente para probar y ajustar las mecánicas básicas del juegos como los saltos y ataques.

Ya que este es un personaje para *testear* mecánicas, las animaciones que se han realizado no hacen mucho uso de algunos de los **12 principios de la animación**, por eso, algunas animaciones resultan ser más *rígidas* que otras.

En el siguiente enlace se puede apreciar dos animaciones de ejemplo, una más *dinámica* y otra más *rígida*: [Forward Air Attack](#), [Forward Attack](#).

- **Efectos especiales del personaje:**

- **Doble salto:** para realizar este modelo y animación tuve inspiración del hipotético caso de que el personaje comprime el aire y toca de nuevo el suelo para volver a saltar, así dando la sensación de que el doble salto no es algo mágico, sino algo más real.

Por ello se ha usado dos anillos, uno más pequeño que el otro, que haría referencia a ser una especie de “*plataforma comprimida por el aire*”, además de esta manera no sería intrusivo para el usuario.

Originariamente esta plataforma tenía una animación la cual el aro pequeño actuaba como una especie de trampolín, seguido de un incremento de tamaño y de una desaparición, pero al

pasarlo al Unity se observó que la animación era muy larga e intrusiva, así que se transformó a un formato más corto descartando la parte del aro que se usaba como trampolín pero manteniendo la animación de incremento de tamaño y desaparición de estas, dando resultado a una animación entendible y poco intrusiva.

En el siguiente enlace se puede apreciar el resultado final de dicha animación: [Double Jump](#).

- **Efecto de caída rápida:** en el momento de la caída rápida, me di cuenta que el jugador no sabía cuando estaba activa y cuando no, por ello decidí hacer algún tipo de señal.

Al ser una mecánica la cual solamente se puede apreciar si caes más rápido, pensé que algún tipo de señal que fuera dinámica y veloz funcionaría bien para el proyecto.

Por ello, realicé un pequeño modelo con forma de estrella, parecido a una chispa, la cual aparece momentáneamente cuando el jugador usa la caída rápida.

En el siguiente enlace se puede apreciar el resultado final de dicha animación: [Efecto de caída rápida](#).

- **Adobe Audition:** para el modo *Maestría de personaje*, grabe y edite unas líneas de voz propias diciendo “Ready”, “Go”, “Failure”, “Success”.

Dichas líneas de voz se les aplicó un “*Desfasador de tono*” personalizado, para que la voz de dichos audios fuera más grave.

En los siguientes enlaces se puede apreciar el resultado final de dichas grabaciones: [Ready](#), [Go](#), [Failure](#), [Success](#).

- **Unity:** el objetivo principal del proyecto fue recrear el *Platform Fighter* que estaba diseñando, por lo que se utilizó Unity para implementar la idea central. El desarrollo se dividió en dos grandes ramas principales para optimizar el proceso y cubrir distintos aspectos del diseño y las mecánicas del juego.

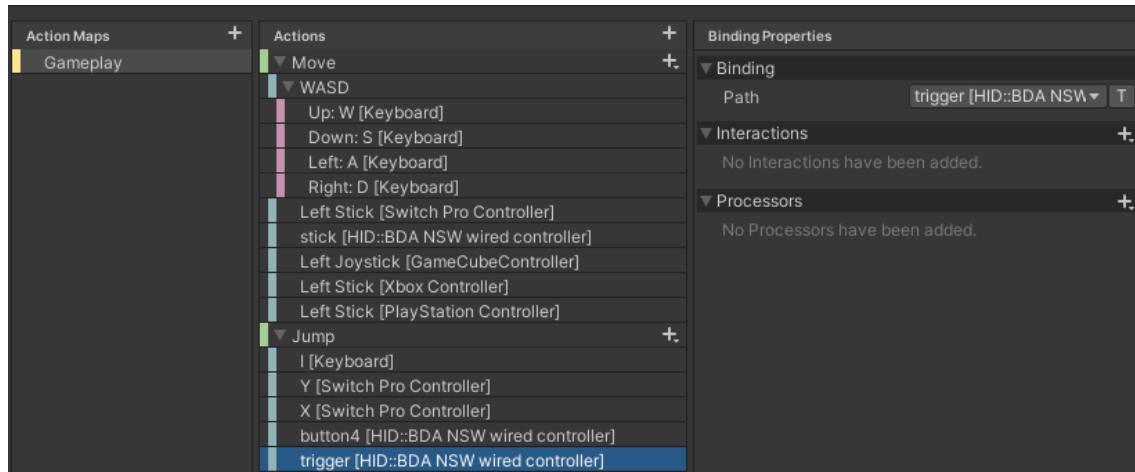
- **Rama platformer:** Esta *rama* del proyecto se enfoca principalmente en el movimiento del personaje, tanto en el aire como en el suelo, así como en las interacciones y colisiones con los distintos objetos del escenario. A continuación, se clasifica en los siguientes apartados:

- **Entradas de Inputs:** pese a que Unity dispone de un sistema de entrada de inputs por defecto, tuve que implementar un **nuevo input system** el cual tenía soporte con nuevos mandos como el mando de PS5 o el Pro Controller de Nintendo Switch.

Este tipo de *input system* requiere de un ***Input Actions***, el cual debía ser configurado que entrada de mando o teclado se quiere y para que se va a usar, para poder funcionar. En este Input Actions la manera en que se clasifica es en el siguiente orden:

- **Action Maps:** esta es la primera “capa” de configuración. Este necesita que el usuario cree un nuevo *mapa* de controles para con un nombre. Este se le asignará al cual el usuario lo haya enlazado.
- **Actions:** esta capa es donde se distribuyen la acción y el input a usar. De esta manera, el usuario primariamente crea una acción con un nombre, el cual luego le tiene que asociar el tipo de *acción*, es decir, si es un **valor**, considerado como un float, o es un **botón**, el cual un entero, todo esto yendo de 0 a 1 o a -1.

Seguidamente el usuario le debe de asociar el ***binding***, es decir, el input de teclado o mando al cual se le asociará para que haga la función designada.



Proceso de creación - Actions

Aún teniendo todo esto, los inputs no se leerán tal cual. Se debe crear un script el cual obtenga todos los mandos que estén conectados al ordenador, distribuirlos y hacer la lectura de cada *inputs* por individual.

En mi caso creé un script llamado *InputReader*, el cual no debía ser asignado como componente a ningún *Player* pero sí que debía de ponerlo como componente del script principal de movimiento.

A continuación explicaré que hace este código de manera detallada:

Detección y asignación de mando

➤ Inicialización de los mandos y dispositivos:

Cuando se activa el componente *InputReader*, se crean dos instancias de la clase ***GamelInput: _gameInput1*** para el Jugador 1 y ***_gameInput2*** para el Jugador 2. Estos objetos gestionan la entrada de los jugadores y permiten habilitar el esquema de control del juego a través del método ***SetGameplay()***.

➤ Detección y asignación de dispositivos de entrada

El método ***AssignPlayerDevice(InputDevice device)*** es el encargado de asignar el dispositivo (teclado, mando, etc.) a cada jugador. Si un dispositivo no está asignado, se asocia al Jugador 1 o al Jugador 2. Si ambos dispositivos ya están asignados, se informa que no hay espacio para más jugadores.

Asignación de eventos de acción

➤ Gestión unificada de las acciones de los jugadores

El código utiliza los métodos ***OnMove(), OnJump() y OnAttack()*** para gestionar las entradas de movimiento, salto y ataque de los jugadores. Aunque cada acción tiene su propio método, todas ellas siguen el mismo proceso general de asignación de dispositivo y mapeo de eventos.

➤ Detección de dispositivo activo

Cada método primero llama a ***AssignPlayerDevice(context.control.device)*** para determinar qué dispositivo está realizando la acción. Una vez asignado el dispositivo (ya sea para Jugador 1 o Jugador 2), se pasa la entrada correspondiente a un evento asociado.

➤ Disparadores de eventos para las acciones

Para **moverse**, el método lee el **vector2D** de movimiento y lo pasa al evento ***MoveEvent***, diferenciando entre jugadores.

Para saltar, los eventos ***JumpStartEvent***, ***JumpEvent*** y ***JumpCancelEvent*** son activados en función de las fases de la acción de salto (inició, realización, cancelación).

Para atacar, se invocan los eventos ***AttackStartEvent*** y ***AttackCancelEvent***, que manejan el inicio y la finalización de la acción de ataque.

```
public class InputReader : ScriptableObject, GameInput.IGameplayActions
{
    GameInput _gameInput1, _gameInput2;

    InputDevice _player1Device, _player2Device;

    ④ Mensaje de Unity | 0 referencias
    private void OnEnable()
    {
        if (_gameInput1 == null && _gameInput2 == null)
        {
            _gameInput1 = new GameInput();
            _gameInput2 = new GameInput();

            _gameInput1.Gameplay.SetCallbacks(this);
            _gameInput2.Gameplay.SetCallbacks(this);

            SetGameplay();
        }
    }

    1 referencia
    public void SetGameplay()
    {
        _gameInput1.Gameplay.Enable();
        _gameInput2.Gameplay.Enable();
    }

    3 referencias
    private void AssignPlayerDevice(InputDevice device)
    {
        if (_player1Device == null)
        {
            _player1Device = device;
            _gameInput1.devices = new[] { _player1Device };
            Debug.Log("Asignado Jugador 1: " + _player1Device.name);
        }
        else if (_player2Device == null && device != _player1Device)
        {
            _player2Device = device;
            _gameInput2.devices = new[] { _player2Device };
            Debug.Log("Asignado Jugador 2: " + _player2Device.name);
        }
        else if (device != _player1Device && device != _player2Device)
        {
            Debug.Log("No hay espacio para más jugadores");
        }
    }
}
```

Proceso de creación - Asignación de eventos de acciones

```
public event Action<Vector2, int> MoveEvent;

public event Action<int> JumpStartEvent;
public event Action<int> JumpEvent;
public event Action<int> JumpCancelEvent;

public event Action<int> AttackStartEvent;
public event Action<int> AttackCancelEvent;

7 referencias
public void OnJump(InputAction.CallbackContext context)
{
    AssignPlayerDevice(context.control.device);

    if (context.control.device == _player1Device)
        HandleJumpEvents(context, 1);

    if (context.control.device == _player2Device)
        HandleJumpEvents(context, 2);
}

2 referencias
void HandleJumpEvents(InputAction.CallbackContext context, int playerID)
{
    if (context.phase == InputActionPhase.Started) JumpStartEvent?.Invoke(playerID);
    if (context.phase == InputActionPhase.Performed) JumpEvent?.Invoke(playerID);
    if (context.phase == InputActionPhase.Canceled) JumpCancelEvent?.Invoke(playerID);
}

7 referencias
public void OnMove(InputAction.CallbackContext context)
{
    AssignPlayerDevice(context.control.device);

    if (context.control.device == _player1Device)
        MoveEvent?.Invoke(context.ReadValue<Vector2>(), 1);

    if (context.control.device == _player2Device)
        MoveEvent?.Invoke(context.ReadValue<Vector2>(), 2);
}

7 referencias
public void OnAttack(InputAction.CallbackContext context)
{
    AssignPlayerDevice(context.control.device);

    if (context.control.device == _player1Device)
        HandleAttackEvents(context, 1);

    if (context.control.device == _player2Device)
        HandleAttackEvents(context, 2);
}

2 referencias
private void HandleAttackEvents(InputAction.CallbackContext context, int playerId)
{
    if (context.phase == InputActionPhase.Started) AttackStartEvent?.Invoke(playerId);
    if (context.phase == InputActionPhase.Canceled) AttackCancelEvent?.Invoke(playerId);
}
```

Proceso de creación - Asignación de eventos de acciones

- **Movimiento del personaje:** este resulta ser uno de los scripts más extensos, en el cual tiene toda la lógica de movimiento del personaje. Detrás de esta lógica, se puede apreciar un par de grupos que engloban varias “*sub lógicas*”.

Tapping

El sistema de **tapping** se refiere a cómo el juego interpreta y maneja los movimientos del jugador cuando este utiliza la palanca de control. Específicamente, en este código, tiene un manejo detallado sobre cómo el jugador puede moverse dependiendo de la intensidad con la que se mueve la palanca.

- **Umbrales de la palanca:** Hay dos umbrales definidos, *joystickThresholdMin* y *joystickThresholdMax*. Estos valores determinan si el input de la palanca es lo suficientemente fuerte para registrar un movimiento y si el movimiento es lo suficientemente intenso como para desencadenar una acción específica, como correr o ejecutar un dash.
 - **Threshold mínimo:** Si la palanca se mueve más allá del *joystickThresholdMin*, o **umbral mínimo**, el juego reconoce que el jugador está moviendo al personaje, pero el personaje podría caminar en lugar de correr.
 - **Threshold máximo:** Si el jugador empuja la palanca más allá del *joystickThresholdMax*, o **umbral máximo**, el personaje cambia su comportamiento de caminar a correr o realizar un dash, lo que activa un movimiento más rápido.

- **Contador de frames (*framesHeld*):** Para diferenciar entre un toque rápido y uno prolongado, el código utiliza un contador de frames que registra cuánto tiempo se mantiene la palanca en una dirección fuera del **umbral mínimo**. Dependiendo de cuánto tiempo pase desde que se ha salido del **umbral mínimo** y se haya llegado al **umbral máximo**, el personaje puede caminar o correr. Además, cuando se cambia de dirección rápidamente, este contador se resetea, lo que permite al juego manejar las transiciones suaves entre movimientos y cambios de dirección.

```
[Header("Tapping Register")]
public float joystickThresholdMin;
public float joystickThresholdMax;

[Header("Tapping Horizontal")]
public float requiredStickTimeX;
float framesHeld, prevThreshold;

if (Mathf.Abs(Axis.x) > joystickThresholdMin)
{
    if (!isRunning) framesHeld++;

    if (Mathf.Abs(Axis.x) >= joystickThresholdMax)
    {
        if (framesHeld <= requiredStickTimeX)
            isRunning = true;
        else isRunning = false;
    }
}
```

Proceso de creación - Tapping

Movimiento lateral

El movimiento lateral en el juego está influenciado por la **velocidad del personaje**, el **dash**, la **fricción o tracción**, y cómo el personaje cambia entre caminar y correr. Aquí se puede ver un sistema detallado que evalúa la rapidez con la que el jugador mueve la palanca y ajusta la respuesta del personaje en consecuencia.

- **Dash Inicial**

El personaje tiene la capacidad de ejecutar un dash cuando el jugador empuja la palanca rápidamente y por encima del umbral máximo (**joystickThresholdMax**).

El dash se realiza al principio del movimiento y es una acción rápida, con una duración limitada definida por ***dashDuration*** y una velocidad especificada por ***dashSpeed***.

Como detalle, si el jugador **cambia la dirección mientras está haciendo un dash**, el sistema lo registra y ajusta la dirección del dash. Este detalle permite que el dash sea dinámico y responsivo, evitando que el personaje se vea atrapado en una animación o secuencia no deseada.

➤ Selección de velocidad mediante *Tapping*

Dependiendo de cuánto tiempo se mantenga la palanca en una dirección, el personaje puede caminar o correr. El sistema diferencia entre un toque corto y prolongado para determinar si el jugador desea moverse lentamente o más rápido.

Si el ***framesHeld*** es menor que el tiempo requerido para un dash (***requiredStickTimeX***), el personaje caminará. Sin embargo, si el jugador mantiene la palanca en esa dirección más allá de ese tiempo, el personaje empezará a correr, lo que le otorga una mayor velocidad.

➤ Frenada y Tracción

Cuando el personaje deja de moverse o cambia de dirección, la tracción (o fricción) entra en juego. La tracción reduce gradualmente la velocidad del personaje (***tractionValue***), permitiendo una transición más suave hacia la detención completa o un cambio de dirección.

Si el jugador vuelve a empujar la palanca mientras el personaje está frenando, este volverá a correr o caminar dependiendo de la intensidad del movimiento.

Si el jugador cambia la dirección de la palanca bruscamente, el juego activa la tracción para permitir que el personaje frene antes de cambiar de dirección, lo que se ajusta de acuerdo con la física del juego para mantener la fluidez del movimiento.

```
[Header("Movement Horizontal")]
[Header("Speed")]
public float walkSpeed;
public float runSpeed;
[HideInInspector] public float speed;
[HideInInspector] public float platformMoving;
[HideInInspector] public float finalSpeed;
bool isRunning;

[Header("Dash")]
public float dashSpeed;
public float dashDuration;
[HideInInspector] public bool isDashing;
float dashTimeLeft, dashDirection;

[Header("Traction")]
public float tractionValue;
[HideInInspector] public bool tractionBool, isChangingDirTraction;
float tractionAxis, tractionFrames;

void HandleMove(Vector2 dir, int eventPlayerID)
{
    if (eventPlayerID == playerID) Axis = dir;
}

void HorizontalMovement()
{
    if (Mathf.Abs(Axis.x) > joystickThresholdMin && !tractionBool)
    {
        if (!isRunning) framesHeld++;

        if (Mathf.Abs(Axis.x) >= joystickThresholdMax)
        {
            if (framesHeld <= requiredStickTimeX)
            {
                if (!isDashing) // Start Dash
                {
                    isDashing = true;
                    dashTimeLeft = dashDuration;
                    dashDirection = Axis.x;
                }
            }
            else if (isDashing && Mathf.Sign(Axis.x) != Mathf.Sign(dashDirection)) // Change direction dash
            {
                isDashing = true;
                dashTimeLeft = dashDuration;
                dashDirection = Axis.x;
            }
        }
    }
}
```

Proceso de creación - Movimiento lateral

```
        if (isRunning) speed = runSpeed;
        else speed = walkSpeed;
    }
    else speed = walkSpeed;
}
else // Traction + Stop
{
    if (isRunning) tractionBool = true;
    else if (speed == walkSpeed || isDashing)
    {
        speed = 0;
        isDashing = false;
    }

    if (speed > 0 && tractionBool) speed -= tractionValue;
    else if (speed <= 0 && tractionBool)
    {
        speed = 0;
        tractionFrames++;

        if (tractionFrames >= 3)
        {
            tractionBool = false;
            isChangingDirTraction = false;
            tractionFrames = 0;
        }
    }

    isRunning = false;
    framesHeld = 0;
}

if ((prevThreshold > joystickThresholdMin && Axis.x < -joystickThresholdMin ||
     prevThreshold < -joystickThresholdMin && Axis.x > joystickThresholdMin)
    && !isDashing)
{
    framesHeld = 0;
    if (isRunning)
    {
        isChangingDirTraction = true;
        tractionBool = true;
    }
}

if (Mathf.Abs(Axis.x) > joystickThresholdMin) prevThreshold = Axis.x;
else prevThreshold = 0;
```

Proceso de creación - Movimiento lateral

Salto y Gravedad

El sistema de salto y gravedad es uno de los aspectos más importantes del movimiento vertical en este juego, donde se maneja de manera detallada el comportamiento del personaje al saltar y caer.

➤ Full Hop y Short Hop

El juego cuenta con dos tipos de salto: **Full Hop (salto completo)** y **Short Hop (salto corto)**, que se diferencian por la cantidad de tiempo que el botón de salto se mantiene presionado.

Full Hop: se activa cuando el botón de salto se mantiene presionado más tiempo que el valor de ***maxFramesSwitchJump***, lo que provoca un salto más alto. En este caso, el valor de ***sTop*** (velocidad inicial del salto) se iguala a ***speedFullHop***.

Short Hop: ocurre cuando el botón se libera rápidamente, antes de alcanzar ***maxFramesSwitchJump***. En este caso, el personaje ejecuta un salto más bajo, utilizando ***speedShortHop*** como la velocidad de salto.

➤ Doble salto

El personaje puede realizar un doble salto si ha saltado una vez y aún está en el aire. El código utiliza una variable llamada ***canDJump*** que se activa cuando el personaje está en el aire, permitiendo el doble salto cuando el jugador presiona el botón de salto nuevamente. Al ejecutar el doble salto, la velocidad vertical se resetea a 0, lo que permite que el personaje alcance nuevas alturas.

Al igual que el primer salto, el doble salto también utiliza la misma lógica de gravedad y velocidad aérea, con la diferencia que la velocidad del doble salto siempre será la de ***speedFullHop***.

```
[Header("Aerial Speed")]
public float airFastSpeed;
public float airSlowSpeed;
bool speedAirOneTime;

[Header("Movement Vertical")]
[Header("Force")]
public float speedShortHop;
public float speedFullHop;
[HideInInspector] public float verticalSpeed;
float sTop;
[Header("Time")]
public float maxFramesSwitchJump;
public float maxFramesJump;
float counterSwitchJump, counterJump;
[HideInInspector] public bool isJumping;
bool jOneTime, isDJumping, canDJump;
[HideInInspector] public bool djOneTime;

[Header("Gravity")]
public float weight;
float gravity;
public float maxGravity;

[Header("Fast Fall and Fall from platform")]
public float requiredStickTimeY;
float framesHeldY;
[HideInInspector] public bool isFastFall;
[HideInInspector] public bool canFallPlatform;
bool isDownAxisY;

void HandleJump(int eventPlayerID)
{
    if (eventPlayerID == playerID)
    {
        if (cb.isGrounded && !isJumping)
        {
            isJumping = true;

            if (!hit.isD && !hit.isDSmash) hit.CancelAttackForFallAnim();
        }

        if (canDJump) isDJumping = true;
    }
}
```

Proceso de creación - Full hop y Short hop / Doble salto

```
void Jump()
{
    if (!cb.isGrounded)
    {
        canDJump = true;
    }

    if (!canDJump)
    {
        if (isJumping && cb.isGrounded) counterSwitchJump++;

        if (counterSwitchJump > 0) counterJump++;

        if (counterJump >= maxFramesJump)
        {
            if (counterSwitchJump >= maxFramesSwitchJump && cb.isGrounded && verticalSpeed == 0) sTop = speedFullHop;
            else if (counterSwitchJump < maxFramesSwitchJump && cb.isGrounded && verticalSpeed == 0) sTop = speedShortHop;

            if (!jOneTime)
            {
                jOneTime = true;
                audio.Jump();
            }
        }
    }
    else if (canDJump)
    {
        if (isDJumping && !djOneTime)
        {
            verticalSpeed = 0;
            gravity = 0;
            sTop = speedFullHop;
            djOneTime = true;
            speedAirOneTime = false;

            isFastFall = false;
            framesHeldY = 0;
        }
    }

    if (cb.isGrounded && verticalSpeed < 0)
    {
        sTop = 0;
        verticalSpeed = 0;
        canDJump = false;
        isDJumping = false;
        djOneTime = false;

        if (counterJump >= maxFramesJump)
        {
            counterJump = 0;
            counterSwitchJump = 0;
        }

        if (isJumping) isJumping = false;
    }
}
```

Proceso de creación - Full hop y Short hop / Doble salto

➤ Velocidad aérea

Una vez en el aire, la velocidad aérea del personaje está controlada por si el jugador continúa moviendo la palanca. Si el jugador mantiene el control antes de saltar, la velocidad aérea será ***airFastSpeed***, en caso contrario, será ***airSlowSpeed***, lo que permite un control dinámico del personaje mientras se encuentra en el aire.

```
void HorizontalMovement()
{
    if (Mathf.Abs(Axis.x) > joystickThresholdMin && !speedAirOneTime)
    {
        if ((speed == walkSpeed || speed == runSpeed || isDashing) && isJumping && !isDJumping) speed = airFastSpeed;
        else speed = airSlowSpeed;

        speedAirOneTime = true;
    }
}
```

Proceso de creación - Velocidad aérea

➤ Gravedad

La gravedad es esencial para la física del personaje en el aire. El valor de ***gravity*** aumenta progresivamente durante el tiempo en que el personaje está en el aire, simulando el efecto de caída natural.

Máxima gravedad: Se establece un límite en la velocidad de caída mediante ***maxGravity***, asegurando que el personaje no caiga más rápido de lo que debería.

La gravedad se aplica tanto al caer desde un salto como al caer de una plataforma, proporcionando coherencia en el comportamiento de la física vertical en todas las situaciones.

```
void Gravity()
{
    if (!cb.isGrounded)
    {
        gravity += weight * 9.81f * Time.deltaTime;

        if (!hit.canAir) hit.canAir = true;

        if (hit.isF || hit.isU || hit.isD ||
            !hit.isFSmash || !hit.isUSmash || !hit.isDSmash) hit.CancelAttackForFallAnim();
    }
    else
    {
        gravity = 0;

        if (hit.canAir) hit.canAir = false;
    }
}
```

Proceso de creación - Gravedad

➤ Fast fall (Caída rápida)

Cuando el jugador empuja la palanca hacia abajo mientras el personaje está cayendo, puede activar el **fast fall**, lo que aumenta la velocidad de caída del personaje por encima del valor de **maxGravity**.

La caída rápida solo se activa si el personaje ya está descendiendo, y su velocidad de caída se incrementa considerablemente, permitiendo al jugador alcanzar el suelo más rápido.

```
void FastFall()
{
    if (!cb.isGrounded && verticalSpeed < 0)
    {
        if (Axis.y < -joystickThresholdMin)
        {
            framesHeldY++;

            if (Axis.y <= -joystickThresholdMax)
            {
                if (framesHeldY <= requiredStickTimeY && !isDownAxisY)
                {
                    isFastFall = true;
                }
            }
        }
        else if (!isFastFall) framesHeldY = 0;
    }
}
```

Proceso de creación - Fast fall (Caída rápida)

➤ Caída desde plataformas

Además de la caída normal, el personaje puede atravesar plataformas si se mueve hacia abajo mientras está sobre una plataforma. El código registra si el jugador mueve la palanca hacia abajo lo suficiente (*joystickThresholdMin* y *joystickThresholdMax*) y si el personaje está tocando una plataforma (*cb.raycastHitPlatform*). Si ambas condiciones se cumplen, el personaje atraviesa la plataforma y empieza a caer.

```
void FallPlatform()
{
    if (Axis.y < -joystickThresholdMin)
    {
        if (Axis.y <= -joystickThresholdMax)
        {
            if (cb.raycastHitPlatform)
            {
                if ((!cb.isGrounded && verticalSpeed < 0) || cb.isGrounded)
                {
                    if (cb.isGrounded) isDownAxisY = true;

                    canFallPlatform = true;
                    cb.isGrounded = false;
                    cb.raycastHitPlatform = false;
                }
            }
        }
    }
    else
    {
        isDownAxisY = false;

        if (!cb.isTouchingPlatform)
            canFallPlatform = false;
    }
}
```

Proceso de creación - Caída desde plataformas

- **Collider del personaje:** para manejar las colisiones del personaje, opté por no utilizar el componente *PlayerController*, ya que la forma del personaje requería un control más preciso y flexible de las colisiones. En su lugar, se utilizó un ***Rigidbody*** para asegurar que el movimiento fuera más "**físicamente correcto**".

➤ Creación de colisión para detección de físicas

En este apartado, el método ***CreatingCollision()*** se encarga de configurar la malla de colisión del personaje.

- **Puntos clave de colisión:** Se crea un arreglo de puntos (***layer***) que define los vértices de la malla en función del tamaño del personaje (***transform.localScale***). Esto asegura que la colisión sea precisa y ajustada al modelo del personaje.
- **Generación de la malla:** Estos puntos se utilizan para generar una malla que será asignada al ***MeshCollider*** del personaje. Luego, la malla se triangula, lo cual es necesario para que las colisiones funcionen correctamente en Unity.

```
void CreatingCollision()
{
    if (layer != null && layer.Length >= 3)
    {
        // Creación de los puntos y mesh.
        layer[0] = new Vector3(0, transform.localScale.y, 0);
        layer[1] = new Vector3(transform.localScale.x / 2, 0, 0);
        layer[2] = new Vector3(0, 0, transform.localScale.z / 2);
        layer[3] = new Vector3(-transform.localScale.x / 2, 0, 0);
        layer[4] = new Vector3(0, -transform.localScale.y, 0);
        layer[5] = new Vector3(0, 0, -transform.localScale.z / 2);
        layer[6] = new Vector3(transform.localScale.x / 3, transform.localScale.y / 1.5f, 0);
        layer[7] = new Vector3(-transform.localScale.x / 3, transform.localScale.y / 1.5f, 0);
        layer[8] = new Vector3(0, transform.localScale.y / 1.5f, transform.localScale.z / 3);
        layer[9] = new Vector3(0, transform.localScale.y / 1.5f, -transform.localScale.z / 3);

        Mesh mesh = new Mesh();
        mesh.vertices = layer;

        // Triangular la Mesh (necesario para el MeshCollider)
        int[] triangles = new int[3 * (layer.Length - 2)];
        for (int i = 0, j = 1, k = 2; k < layer.Length; i++, j++, k++)
        {
            triangles[3 * i] = 0;
            triangles[3 * i + 1] = j;
            triangles[3 * i + 2] = k;
        }
        mesh.triangles = triangles;

        // Asignar el Mesh al MeshFilter
        GetComponent<MeshFilter>().mesh = mesh;

        // Configurar el MeshCollider
        meshCollider = GetComponent<MeshCollider>();
        meshCollider.sharedMesh = mesh;
        meshCollider.convex = true;
    }
}
```

Proceso de creación - Creación de colisión para detección de físicas

➤ Uso de la colisión

El manejo de colisiones físicas con el entorno se realiza principalmente a través de los métodos ***OnCollisionEnter***, ***OnCollisionStay*** y ***OnCollisionExit***.

- ***OnCollisionEnter***: Este método se ejecuta cuando el personaje toca el suelo o una plataforma. Se comprueba el tag del objeto colisionado (usando ***other.gameObject.tag***) para determinar si es una plataforma o el suelo, y se actualiza el estado del personaje (***isGrounded***, ***isTouchingPlatform***).

```
private void OnCollisionEnter(Collision other)
{
    if (other.gameObject.tag == "Floor" && raycastHitFloor
        && other.collider == raycastHitCollider)
    {
        isGrounded = true;
        if (player.isHitted)
        {
            player.damagedOneTime = false;
            player.isHitted = false;
            player.knockbackSpeed = Vector3.zero;
        }
        player.AfterLanding();
        player.audio.IsGround();
    }

    if (other.gameObject.tag == "PlatformF" && raycastHitPlatform
        && player.verticalSpeed <= 0 && !player.canFallPlatform
        && other.collider == raycastHitCollider)
    {
        isTouchingPlatform = true;
        isGrounded = true;
        if (player.isHitted)
        {
            player.damagedOneTime = false;
            player.isHitted = false;
            player.knockbackSpeed = Vector3.zero;
        }
        player.AfterLanding();
        player.audio.IsGround();
    }
}
```

Proceso de creación - *OnCollisionEnter*

- **OnCollisionStay:** Similar a **OnCollisionEnter**, pero se ejecuta mientras el personaje sigue en contacto con una superficie. Aquí también se comprueba si el personaje está realizando una "fast fall" (**caída rápida**) y se ejecutan las acciones correspondientes al aterrizar.

```
private void OnCollisionStay(Collision other)
{
    if (other.gameObject.tag == "Floor" && raycastHitFloor
        && other.collider == raycastHitCollider)
    {
        isGrounded = true;

        if (player.isFastFall)
        {
            player.AfterLanding();
            player.audio.IsGround();
        }
    }

    if (other.gameObject.tag == "PlatformF" && raycastHitPlatform
        && player.verticalSpeed <= 0 && !player.canFallPlatform
        && other.collider == raycastHitCollider)
    {
        isTouchingPlatform = true;
        isGrounded = true;
    }
}
```

Proceso de creación - *OnCollisionStay*

- **OnCollisionExit:** Cuando el personaje deja de tocar el suelo o una plataforma, se actualiza el estado para reflejar que ya no está en tierra.

```
private void OnCollisionExit(Collision other)
{
    if (other.gameObject.tag == "Floor")
    {
        isGrounded = false;

        if (player.isTouchingWall) player.isTouchingWall = false;
    }

    if (other.gameObject.tag == "PlatformF")
    {
        isTouchingPlatform = false;

        isGrounded = false;
    }
}
```

Proceso de creación - *OnCollisionExit*

➤ Traspasar plataformas

Este apartado se gestiona con ***OnTriggerEnter***, ***OnTriggerStay*** y ***OnTriggerExit***, que controlan cómo el personaje puede pasar a través de plataformas.

- **Ignorar colisiones:** Cuando el personaje está en modo de caída o se le permite atravesar plataformas (***player.canFallPlatform***), las colisiones con las plataformas se ignoran temporalmente. Esto se logra con el método ***Physics.IgnoreCollision***.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "PlatformF" && player.canFallPlatform) isFalling = true;
    else
    {
        if (other.gameObject.tag == "PlatformF") Physics.IgnoreCollision(meshCollider, other, true);
    }
}
```

Proceso de creación - Ignorar colisiones

- **Activar o desactivar colisión:** Dependiendo del estado del personaje (si está cayendo o no), se activa o desactiva la colisión con la plataforma. Este comportamiento se reinicia al salir de la plataforma con ***OnTriggerExit***.

```
private void OnTriggerExit(Collider other)
{
    if (other.gameObject.tag == "PlatformF" || other.gameObject.tag == "Floor")
    {
        if (isFalling)
        {
            meshCollider.enabled = true;

            isTouchingPlatform = false;
            isFalling = false;
        }
    }
}
```

Proceso de creación - Activar o desactivar colisiones

➤ Detección de contacto entre jugadores

El juego también maneja colisiones entre jugadores usando el mismo enfoque de ***Physics.IgnoreCollision***, dependiendo de si el personaje está en el suelo o si está siendo golpeado.

- **Ignorar colisión en ciertas situaciones:** Si el personaje no está en el suelo o está en un estado de daño (***isHitted***), se ignoran las colisiones con otros personajes.

```
if (other.gameObject.tag == "Player" && (!isGrounded || player.isHitted))
    Physics.IgnoreCollision(meshCollider, other, true);
else if (other.gameObject.tag == "Player" && isGrounded)
    Physics.IgnoreCollision(meshCollider, other, false);
```

Proceso de creación - Ignorar colisión en ciertas situaciones

Este comportamiento se asegura de que los jugadores no interfieran físicamente entre sí en momentos donde las colisiones no son necesarias o podrían complicar el gameplay.

➤ Raycast de detección de suelo

Este sistema utiliza el método ***RaycastGround()*** para detectar si hay una superficie (piso o plataforma) debajo del personaje. Se lanza un raycast desde la parte inferior del personaje y se comprueba qué tipo de superficie se encuentra.

- **Configuración del raycast:** se lanza desde una posición debajo del personaje (***transform.position - new Vector3(0, transform.localScale.y / 2, 0)***) y en la dirección definida en ***directionC***.

- **Detección de superficies:** Dependiendo de lo que se detecta (ya sea una plataforma o el suelo), se ajustan las variables *raycastHitFloor* y *raycastHitPlatform*, que luego se usan para determinar el estado de si el personaje está en tierra o no.

```
void RaycastGround()
{
    RaycastHit hit;

    maxDistance = Mathf.Infinity;

    if (Physics.Raycast(transform.position - new Vector3(0, transform.localScale.y / 2, 0), directionC, out hit, maxDistance))
    {
        if (hit.collider.gameObject != gameObject && hit.collider.gameObject.layer != LayerMask.NameToLayer("HitBoxes")
            && hit.collider.gameObject.tag != "Player")
        {
            float distance = Vector3.Distance(transform.position - new Vector3(0, transform.localScale.y / 2, 0), hit.point);
            if (distance < maxDistance) maxDistance = distance;

            raycastHitCollider = hit.collider;

            if (hit.collider.gameObject.tag == "Floor")
            {
                raycastHitFloor = true;
                raycastHitPlatform = false;
            }

            if (hit.collider.gameObject.tag == "PlatformF")
            {
                raycastHitPlatform = true;
                raycastHitFloor = false;
            }

            if (hit.collider.gameObject.tag != "PlatformF" && hit.collider.gameObject.tag != "Floor")
            {
                raycastHitPlatform = false;
                raycastHitFloor = false;
            }
        }
    }
    else
    {
        raycastHitFloor = false;
        raycastHitPlatform = false;
        raycastHitCollider = null;
    }
}
```

Proceso de creación - Raycast de detección de suelo

➤ Raycast de detección de pared

El método *RaycastHorizontal()* se usa para detectar si el personaje está cerca de una pared u obstáculo lateral.

- **Lanzamiento de raycast:** Los raycasts se lanzan desde varias posiciones (*originRaycasts[]*) en los lados del personaje y en la dirección del movimiento (*Mathf.Sign(player.Axis.x)*).

- **Detección de paredes:** Si se detecta una pared, el estado *isTouchingWall* del personaje se actualiza, lo que puede afectar su capacidad para seguir moviéndose o realizar acciones como un "wall jump".

```
void RaycastHorizontal()
{
    player.isTouchingWall = false;

    foreach (Vector3 origin in originRaycasts)
    {
        RaycastHit hitW;

        if (Physics.Raycast(origin + transform.position, new Vector3(Mathf.Sign(player.Axis.x) *
            Mathf.Ceil(Mathf.Abs(player.Axis.x)), 0, 0), out hitW, maxDistanceW))
        {
            if (hitW.collider.gameObject != gameObject && hitW.collider.gameObject.layer != LayerMask.NameToLayer("HitBoxes"))
            {
                if (hitW.collider.gameObject.tag == "Floor")
                {
                    player.isTouchingWall = true;
                }
            }
        }
    }
}
```

Proceso de creación - Raycast de detección de pared

- **Seguimiento de la cámara:** controla la cámara en el juego multijugador, ajustando su posición y campo de visión (FOV) para que todos los personajes se mantengan visibles dentro de los límites del escenario.
 - **Zona de visión:** Este apartado se refiere a cómo la cámara se ajusta para mantener a todos los personajes visibles, limitando el movimiento de la cámara dentro de los límites del escenario.

- **Cálculo de los límites:** En el método *UpdateLimits()*, se definen los límites de la cámara a partir de la posición de los objetos *leftLimit*, *rightLimit*, *topLimit* y *bottomLimit*, que representan las fronteras del escenario. Se ajustan mediante las escalas locales de esos objetos y se añade un *limitPaddingX* y *limitPaddingY* para evitar que la cámara se acerque demasiado a los bordes.

```
1 referencia
void UpdateLimits()
{
    leftLimitX = leftLimit.position.x + leftLimit.localScale.x + limitPaddingX;
    rightLimitX = rightLimit.position.x - rightLimit.localScale.x - limitPaddingX;
    topLimitY = topLimit.position.y - topLimit.localScale.y - limitPaddingY;
    bottomLimitY = bottomLimit.position.y + bottomLimit.localScale.y + limitPaddingY;
}
```

Proceso de creación - Cálculo de los límites

- **Cálculo del área visible:** En el método ***CalculatePlayersBox()***, se crea un rectángulo que contiene las posiciones de todos los jugadores. Se calcula el punto mínimo y máximo en los ejes X e Y que abarcan a todos los personajes. Luego, a ese área se le añade un ***padding*** para asegurar que los personajes no queden pegados al borde de la pantalla.

```
return Rect.MinMaxRect(min.x - padding, max.y + padding, max.x + padding, min.y - padding);
```

Proceso de creación - Cálculo del área visible

- **Posición de la cámara dentro de los límites:** En ***CalculateCamPos()***, la cámara se coloca en el centro del área definida por el rectángulo que contiene a los jugadores, pero su posición se ajusta para que no sobrepase los límites del escenario. Se usan las funciones ***Mathf.Clamp*** para asegurar que la cámara no salga de los bordes.

```
newPos.x = Mathf.Clamp(newPos.x, leftLimitX + halfHorizontalSize, rightLimitX - halfHorizontalSize);
newPos.y = Mathf.Clamp(newPos.y, bottomLimitY + halfVerticalSize, topLimitY - halfVerticalSize);
```

Proceso de creación - Posición de la cámara dentro de los límites

- **Proporción de la cámara:** Para ajustar la vista de la cámara, se calculan los tamaños vertical y horizontal de la caja visible mediante las proporciones de la cámara (el aspecto y el FOV) y la distancia del punto de vista.

➤ **Posición de los personajes:** Este apartado gestiona cómo la cámara sigue a los personajes y ajusta su posición en función de dónde están ubicados en el escenario.

- **Recopilación de posiciones:** En el método *Start()*, el script encuentra todos los objetos etiquetados como "*Player*" y almacena sus transformaciones (posiciones) en el arreglo *players*. Este arreglo se utiliza durante todo el script para conocer en tiempo real dónde están los personajes.

```
void Start()
{
    cam = GetComponent<Camera>();
    cam.orthographic = false;

    GameObject[] playersObjects = GameObject.FindGameObjectsWithTag("Player");
    players = new Transform[playersObjects.Length];
    for (int i = 0; i < playersObjects.Length; i++)
    {
        players[i] = playersObjects[i].transform;
    }
}
```

Proceso de creación - Recopilación de posiciones

- **Cálculo del área que abarca a todos los jugadores:** En *CalculatePlayersBox()*, se recorren todas las posiciones de los personajes (*players*) y se determina el área mínima y máxima que ocupan. Esto se hace mediante las funciones *Mathf.Min* y *Mathf.Max* para calcular las posiciones extremas en los ejes X e Y.

```
foreach (Transform player in players)
{
    min.x = Mathf.Min(min.x, player.position.x);
    min.y = Mathf.Min(min.y, player.position.y);
    max.x = Mathf.Max(max.x, player.position.x);
    max.y = Mathf.Max(max.y, player.position.y);
}
```

Proceso de creación - Cálculo del área que abarca a todos los jugadores

Este rectángulo que abarca a todos los personajes se usa para posicionar la cámara de modo que todos los jugadores permanezcan visibles en la pantalla, aunque se muevan por el escenario.

➤ **Cálculo general:** Este apartado maneja el cálculo del campo de visión (FOV, por sus siglas en inglés) y cómo se ajusta dinámicamente para asegurarse de que todos los jugadores permanezcan dentro del marco visible de la cámara.

- **Cálculo del FOV:** En el método ***CalculateFOVSize()***, se determina el tamaño del campo de visión que necesita la cámara para que todos los personajes sean visibles. Se calcula tanto el FOV vertical como el horizontal, basándose en las dimensiones de la caja que contiene a los jugadores y la distancia de la cámara.
- **Ajuste del FOV:** La función ***Mathf.Lerp*** se usa para suavizar la transición del FOV actual al nuevo tamaño deseado en función del tiempo (***Time.deltaTime***) y la velocidad de zoom (***zoomSpeed***). Además, el FOV final está limitado por el valor mínimo definido en ***minFOV*** y el máximo permitido según los límites del escenario.
- **Límites del FOV:** Para evitar que la cámara se aleje demasiado del escenario, el FOV está limitado por el tamaño máximo permitido del escenario, tanto en el eje X como en el Y. Esto asegura que la cámara no muestre áreas fuera del escenario y mantenga una vista clara de la acción dentro del área jugable.

```
float CalculateFOVSize(Rect box)
{
    // Dimensiones de la caja.
    float boxHeight = Mathf.Abs(box.height);
    float boxWidth = Mathf.Abs(box.width);

    // Calculo de FOV deseado.
    float distanceBox = Mathf.Abs(cam.transform.position.z);
    float fovVertical = 2 * Mathf.Atan(boxHeight / (2 * distanceBox)) * Mathf.Rad2Deg;
    float fovHorizontal = 2 * Mathf.Atan(boxWidth / (2 * (distanceBox / cam.aspect))) * Mathf.Rad2Deg;

    // Ajuste de la caja dentro de la cámara
    float desiredFOV = Mathf.Max(fovVertical, fovHorizontal);

    // Ajustar distancia dependiendo de las blastlines.
    float maxAllowedWidth = rightLimitX - leftLimitX - padding * 2;
    float maxAllowedHeight = topLimitY - bottomLimitY - padding * 2;

    float maxAllowedFOVHorizontal = 2 * Mathf.Atan(maxAllowedWidth / (2 * (distanceBox / cam.aspect))) * Mathf.Rad2Deg;
    float maxAllowedFOVVertical = 2 * Mathf.Atan(maxAllowedHeight / (2 * distanceBox)) * Mathf.Rad2Deg;

    return Mathf.Clamp(Mathf.Lerp(cam.fieldOfView, Mathf.Max(fovVertical, fovHorizontal), Time.deltaTime * zoomSpeed),
                      minFOV, Mathf.Min(maxAllowedFOVHorizontal, maxAllowedFOVVertical));
}
```

Proceso de creación - Cálculo general

- **Rama fighter:** Esta rama del proyecto se enfoca al sistema de pelea del juego, tanto en la creación de colisiones, así como en las interacciones y colisiones entre ellos. A continuación, nombrarán algunos de los procesos que se han realizado:

- **Creación de colliders:** cada collider está hecho a mano, todos siendo *GameObject* dentro del hueso correspondiente. Estos *GameObjects* acaban con el sufijo “_Box” el cual lo identifica como un *hitbox*. Además de esto, cada uno contiene el *tag* “**PlayerColliders**” y el *layer* “**HitBoxes**” el cual no es capaz de interactuar con ninguna colisión que no sea de la misma *layer*.

Estos colliders contienen dos componentes específicos, el primero siendo este un “**Box Collider**” el cual registra si han colisionado los personajes, y por ultima el script “**Hit Boxes State**” el cual es distribuido por otro script.

“**Hit Boxes State**” maneja el comportamiento de la **hitbox** en la cual se le ha asignado, determinando si están en estado de **vulnerabilidad** o **daño**, además de ajustar el color para del estado, y gestionando las colisiones con las hitboxes de otros personajes.

La función encargada de la gestión de colisiones es la llamada “**CheckCollision()**”, la cual, utiliza **Physics.OverlapBox** para identificar las hitboxes que se superponen en un área definida por el “**Box Collider**” del personaje.

Luego, recorre las hitboxes detectadas y verifica que no sea la propia hitbox y que pertenezca a otro personaje (comparando el objeto padre con la variable **father**). Si la hitbox del personaje está en estado de ataque (**indexState == 1**) y colisiona con una hitbox en estado vulnerable (**indexState == 0**), se registra la colisión con el enemigo, guardando su collider en la variable **enemyCollider**.

Finalmente, si no se detectan nuevas colisiones pero antes había una, se borra la referencia al enemigo para evitar errores de detección. Este sistema controla cuándo una hitbox golpea a otra, diferenciando entre personajes y estados de vulnerabilidad y ataque.

```
void CheckCollision()
{
    BoxCollider box = GetComponent<BoxCollider>();

    Collider[] hitColliders = Physics.OverlapBox(box.transform.TransformPoint(box.center),
        box.size / 2, box.transform.rotation, LayerMask.GetMask("HitBoxes"));

    bool noCollisions = true;

    foreach (BoxCollider hitCollider in hitColliders)
    {
        if (hitCollider != box && hitCollider.GetComponent<HitBoxesState>().father != father)
        {
            if (indexState == 1 && hitCollider.GetComponent<HitBoxesState>().indexState == 0)
            {
                noCollisions = false;
                enemyCollider = hitCollider;
            }
        }
    }

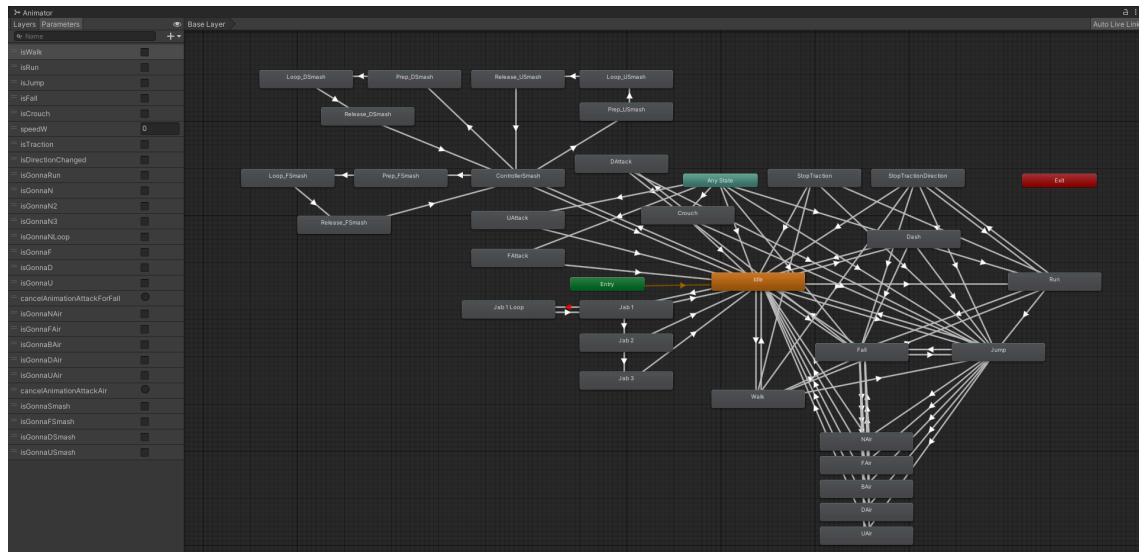
    if (noCollisions && enemyCollider != null)
    {
        enemyCollider = null;
    }
}
```

Proceso de creación - Creación de colliders

- **Asignación de animaciones:** para cada personaje, se ha asociado una serie de animaciones que corresponden a las acciones que puede realizar dentro del juego. Estas animaciones están organizadas y gestionadas a través del sistema Animator de Unity, donde se especifican los tiempos de ejecución para cada una.

En el diagrama final del Animator, se pueden observar las transiciones entre animaciones, que se activan según las condiciones definidas en el código. Los eventos de las animaciones llaman a las funciones correspondientes que ajustan las hitboxes, estados, y otros elementos del gameplay. Esto asegura que cada movimiento se ejecute con precisión según lo planeado.

Cabe destacar que algunas de estas animaciones están correlacionadas o tienen funciones especiales, como es el caso de los **ataques fuertes**, los cuales tienen una animación de preparación, una animación *loop* antes de soltar un ataque y luego el *lanzamiento* del ataque, pero estos pueden soltarse en cualquier momento que desee el jugador.



Proceso de creación - Asignación de animaciones

■ **Eventos:** Para gestionar las colisiones y los estados de animación de forma eficiente, se implementaron eventos específicos en cada animación. Estos eventos fueron configurados en el software en formato **.mb** o **.fbx**, accediendo a la sección de **Animation** y luego a **Events**. A continuación, se describen los tipos de eventos utilizados:

- **Evento de activación de colisión:** Este evento se usa para activar o desactivar las colisiones durante una animación. Se configura para llamar a una función específica en el script que maneja las colisiones, permitiendo que la colisión detecte y responda correctamente a los impactos según el estado actual del personaje. Este evento es crucial para sincronizar las interacciones físicas con la animación.
- **Evento de cancelación de animación:** Para evitar que las animaciones permanezcan activas después de finalizar, se añadió un evento al final de cada animación. Este evento desactiva los estados de animación y colisiones que ya no son necesarios, asegurando que la transición entre animaciones sea fluida y sin problemas.
- **Evento especial:** En el caso de movimientos específicos como los jabs, se implementaron eventos especiales. Estos eventos se activan a partir de un frame concreto en la animación, permitiendo que el personaje avance al siguiente jab o acción relacionada. Estos eventos son esenciales para la secuenciación precisa de movimientos complejos y combos.

- **Asignación de valores de cada movimiento:** cuando ya están los eventos hechos en cada animación, se procede hacer un script, asociado al *Animator* del personaje, que se encarga de gestionar las **hitboxes** durante diferentes fases de los ataques de un personaje en un juego.

El objetivo es ajustar los valores de estas **hitboxes** en tiempo real, aplicando cambios como el tamaño, el ángulo de lanzamiento, la velocidad y el daño que generan.

Estas actualizaciones ocurren a través de eventos de animación que activan funciones específicas como *Jab1()*, *Jab2()*, y *Jab3()*, cada una correspondiente a una animación del personaje.

Un detalle a tener en cuenta, pese a que todas las clases tienen una estructura muy similar, cada una llama a una pieza de las *hitboxes* del personaje diferentes.

```
void Jab1(int i)
{
    foreach (BoxCollider hit in state.hitBoxes)
    {
        if (hit.name == "H_R_Box")
        {
            if (i == 1)
            {
                hit.GetComponent<HitBoxesState>().DamageAction();
                hit.GetComponent<HitBoxesState>().scale = new Vector3(0.15f, 0.15f, 0.15f);
                hit.GetComponent<HitBoxesState>().launchAngle = 5;
                hit.GetComponent<HitBoxesState>().launchSpeed = 5;
                hit.GetComponent<HitBoxesState>().damage = 3;
            }
            else if (i == 0)
            {
                hit.GetComponent<HitBoxesState>().Vulnerable();
                hit.GetComponent<HitBoxesState>().scale = Vector3.zero;
                hit.GetComponent<HitBoxesState>().launchAngle = 0;
                hit.GetComponent<HitBoxesState>().launchSpeed = 0;
                hit.GetComponent<HitBoxesState>().damage = 0;
            }
        }

        if (hit.name == "A_R_2_Box")
        {
            if (i == 1)
            {
                hit.GetComponent<HitBoxesState>().DamageAction();
                hit.GetComponent<HitBoxesState>().scale = new Vector3(0.1f, 0.1f, 0.1f);
                hit.GetComponent<HitBoxesState>().launchAngle = 2;
                hit.GetComponent<HitBoxesState>().launchSpeed = 1;
                hit.GetComponent<HitBoxesState>().damage = 2;
            }
            else if (i == 0)
            {
                hit.GetComponent<HitBoxesState>().Vulnerable();
                hit.GetComponent<HitBoxesState>().scale = Vector3.zero;
                hit.GetComponent<HitBoxesState>().launchAngle = 0;
                hit.GetComponent<HitBoxesState>().launchSpeed = 0;
                hit.GetComponent<HitBoxesState>().damage = 0;
            }
        }
    }
}
```

Proceso de creación - Asignación de valores de cada movimiento

■ **Asignación de valores al enemigo:** a continuación se ha desarrollado un sistema para gestionar la interacción entre el personaje del jugador y los enemigos en el juego, específicamente en lo que respecta a la asignación de valores críticos como el **daño**, la **velocidad** de lanzamiento, el **ángulo** de lanzamiento, y la **dirección** del impacto cuando un enemigo es golpeado. Este proceso se lleva a cabo en el método “*SaveEnemyBox()*” dentro del script “*HitBoxesController*”:

- **Detección de colisión:** revisa todas las “hitboxes” del jugador para comprobar si alguna ha colisionado con un enemigo. Si detecta una colisión, se identifica al enemigo afectado.
- **Asignación de valores:** una vez identificado el enemigo, se le asignan varios valores importantes, como la velocidad y el ángulo de lanzamiento, el daño recibido, y la dirección en la que será lanzado.
- **Prevención de reasignación:** para evitar que los valores se reasignen repetidamente en un corto periodo de tiempo, el método marca al enemigo como “golpeado” (*isHitted*).

```
void SaveEnemyBox()
{
    foreach (BoxCollider state in hitBoxes)
    {
        if (state.GetComponent<HitBoxesState>().enemyCollider != null)
        {
            Transform enemyParent = state.GetComponent<HitBoxesState>().enemyCollider.transform;
            MovementBasis enemyState = null;

            while (enemyParent != null)
            {
                enemyState = enemyParent.GetComponent<MovementBasis>();

                if (enemyState != null)
                {
                    break;
                }

                enemyParent = enemyParent.parent;
            }

            if (state != null)
            {
                if (!enemyState.isHitted)
                {
                    enemyState.launchSpeed = state.GetComponent<HitBoxesState>().launchSpeed + ((framesMaxCharge / 10) * 2);
                    enemyState.launchAngle = state.GetComponent<HitBoxesState>().launchAngle;
                    enemyState.damage = state.GetComponent<HitBoxesState>().damage + (((int)framesMaxCharge / 10) * 2);
                    enemyState.isHitted = true;
                    enemyState.knockbackBool = true;
                    if (transform.eulerAngles.y < 180) enemyState.direction = 1;
                    else enemyState.direction = -1;
                }
            }
        }
    }
}
```

Proceso de creación - Asignación de valores al enemigo

- **Knockback:** es un elemento crucial en este género el *knockback*, ya que determina cómo un personaje es empujado o lanzado tras recibir un golpe. El sistema de knockback es responsable de calcular la fuerza y dirección del impacto, así como de aplicar estas fuerzas al personaje afectado, creando una respuesta física realista y coherente en el juego.

Aquí se ha implementado en el script de movimiento principal, “**MovementBasis**”, para que no se pisara con el resto de movimiento normal que tienen los personajes y actúe de manera extraña:

➤ **Cálculo del ángulo y la dirección del Knockback:**

El método comienza verificando si el personaje ha sido golpeado (*knockbackBool* es verdadero). Si es así, se calcula el ángulo de lanzamiento en función de la dirección del impacto. Si el impacto viene desde la izquierda, el ángulo se ajusta restando 180 grados.

Los ángulos en grados se convierten a radianes (*angleRadX* y *angleRadY*) para su uso en cálculos trigonométricos posteriores.

```
void Knockback()
{
    if (knockbackBool)
    {
        float launchAngleX = launchAngle;

        if (direction == -1) launchAngleX -= 180;

        angleRadY = launchAngle * Mathf.Deg2Rad;
        angleRadX = launchAngleX * Mathf.Deg2Rad;
```

Proceso de creación - Cálculo del ángulo y la dirección del Knockback

➤ **Aplicación del daño y ajuste del porcentaje:**

Si el personaje no ha sido dañado previamente en este impacto (*damagedOneTime* es falso), se incrementa el porcentaje de daño acumulado (*percentage*) en función del daño recibido. Este porcentaje se utiliza para escalar la fuerza del knockback, haciendo que los personajes con mayor daño acumulado sean lanzados más lejos.

El daño se aplica visualmente al personaje a través de un método externo (*IsDamaged*), y se marca el personaje como dañado para evitar que el daño se aplique múltiples veces en un solo impacto.

```
if (!damagedOneTime)
{
    percentage += damage;

    if (percentage >= 999) percentage = 999;

    GetComponent<DamagePlayer>().IsDamaged(percentage);

    damagedOneTime = true;
}
```

Proceso de creación - Aplicación del daño y ajuste del porcentaje

➤ **Cálculo de la velocidad de Knockback:**

La velocidad del *knockback* se calcula utilizando las funciones trigonométricas *Mathf.Cos* y *Mathf.Sin*, que determinan las componentes X e Y de la velocidad en función del ángulo de lanzamiento y la velocidad de lanzamiento (*launchSpeed*). Esta velocidad se escala por el porcentaje de daño acumulado, lo que aumenta la magnitud del *knockback* a medida que el personaje recibe más daño.

➤ **Aplicación de fuerzas y gestión del Knockback:**

Se aplican fuerzas adicionales como la resistencia al aire (*dragForce*) y la gravedad (*gravityKnockback*) para simular un movimiento más realista durante el knockback.

La fuerza resultante se aplica al *Rigidbody* del personaje utilizando *AddForce*, lo que impulsa al personaje en la dirección calculada.

```
knockbackBool = false;

knockbackSpeed = new Vector3(Mathf.Cos(angleRadX) * launchSpeed * percentage,
    Mathf.Sin(angleRadY) * launchSpeed * percentage, 0);
}

Vector3 dragForce = dragBase * knockbackSpeed;
Vector3 gravityKnockback = new Vector3(1, 1, 0) * 9.81f;

knockbackSpeed = knockbackSpeed - (dragForce + gravityKnockback);

GetComponent<Rigidbody>().AddForce(knockbackSpeed * 0.1f, ForceMode.Impulse);
```

Proceso de creación - Cálculo de la velocidad de Knockback/Aplicación de fuerzas y gestión del Knockback

- **KO:** el segundo elemento crucial de este género, la muerte o el *ko*, se ha realizado con un sistema de contacto en base de unas “*blastlines*” las cuales serían el límite de los escenarios, llegado a ese límite, el jugador se muere y vuelve a reaparecer con 0 de daño. Para hacer esto, en el script “*DamagePlayer*” que lo tiene adjuntado cada *player* dispone de un código específico en “*OnTriggerEnter()*”.

El jugador a la hora de tocar un “*GameObject*” con el *tag* de “*Blastline*” obtendrá los valores del porcentaje para ponerlo a 0 y la velocidad del *rigidbody* para frenarlo y seguidamente de poner al jugador en una nueva posición designada como la “reaparición”.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Blastline")
    {
        ko = true;

        GetComponent<MovementBasis>().percentage = 0;
        GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);

        player.position = new Vector3(0, 13, 0);
    }
}
```

Proceso de creación - KO

- **GitHub:** plataforma principal para subir el proyecto online y poder tenerlo a mano tanto para mi como para cualquier usuario que quiera consultararlo o probarlo. Además también es usado para colgar una versión estable del proyecto.

En el siguiente enlace se puede entrar en el Github para ver y descargar el proyecto o la *build*: [Proyecto de Github](#).

5.3. Gestión de comunidad y retroalimentación

5.3.1. Creación y mantenimiento de la comunidad

Dado que el proyecto está orientado tanto a un público casual como a jugadores más veteranos y hardcore, se propuso reunir un grupo diverso de testers que representarán diferentes perfiles. La idea era obtener retroalimentación de una variedad de jugadores: desde aquellos con experiencia en el género platform fighter hasta aquellos que nunca habían jugado este tipo de juegos antes.

Para ello, se seleccionó a un grupo de nueve personas, con edades comprendidas **entre los 10 y 30 años**, cada una con un perfil de jugador distinto. Las identidades de los participantes se mantendrán en el anonimato para preservar su privacidad. Durante el desarrollo del proyecto, este grupo probó todas las versiones desde el principio hasta el final, aportando ideas, sugerencias y comentarios valiosos que fueron tenidos en cuenta durante el proceso de mejora.

Con el fin de optimizar el flujo de trabajo y facilitar la comunicación entre los testers y el equipo de desarrollo, se creó un servidor privado en *Discord*, una plataforma popular para la gestión de comunidades. Este servidor permitió una interacción constante entre los playtesters y el desarrollador, y proporcionó un espacio para organizar las pruebas y gestionar la retroalimentación de manera eficiente.

El servidor de Discord se organizó en distintos canales, cada uno diseñado con un propósito específico, lo que ayudó a estructurar la interacción dentro de la comunidad. Los principales canales fueron los siguientes:

- **Updates:** Aquí se publicaban las nuevas versiones del juego junto con una breve descripción de las características implementadas en cada actualización. También se colgaban las encuestas que los testers debían completar tras probar cada versión.
- **Welcome playtesters:** Un canal de bienvenida con información general sobre el funcionamiento del servidor y la finalidad de los otros canales. Este canal servía como guía para los nuevos playtesters.
- **Playtesters:** Un canal de discusión general donde los testers podían compartir sus impresiones, clips de juego y debatir entre ellos sobre sus experiencias. Este canal también permitía la interacción directa con el desarrollador, lo que facilitaba un feedback sin intermediarios.
- **Bugs:** El canal dedicado a la identificación y reporte de errores. Para facilitar la revisión de los fallos, se solicitaba que los testers acompañaran sus reportes con videos o capturas de pantalla que evidenciaran los problemas detectados.
- **Ideas:** Este canal estaba destinado a la recopilación de sugerencias, opiniones y propuestas de mejora por parte de los testers. Era un espacio donde podían

expresar sus ideas sobre mecánicas, funcionalidades o cualquier otro aspecto del juego que consideran relevante.

5.3.2. Proceso de pruebas y retroalimentación

El ciclo de pruebas y retroalimentación del proyecto se gestionó de manera iterativa, siguiendo un flujo organizado para maximizar la calidad del feedback recibido. Cada fase de prueba comenzaba con el lanzamiento de una nueva versión significativa del juego. El proceso era el siguiente:

- **Lanzamiento de la versión:** Cada vez que se completaba una versión con cambios significativos, ésta se publicaba en el canal de "**Updates**" dentro del servidor de *Discord*, junto con una breve descripción de las novedades y mejoras incluidas.
- **Publicación de encuestas:** Junto con la nueva versión, se compartía una encuesta para que los playtesters pudieran dar su opinión de manera estructurada. La encuesta estaba diseñada para recoger feedback detallado sobre las nuevas mecánicas, cambios en el gameplay, y posibles errores o inconsistencias.
- **Recepción de feedback en canales:** Durante el período de prueba, los testers tenían libertad para compartir opiniones e impresiones en los canales "**playtesters**" y "**ideas**". También podían reportar errores y fallos en el canal "**bugs**", donde se solicitaba que añadieran clips de vídeo que ayudaran a reproducir el problema.
- **Recordatorio de encuestas:** Tras unos días de haber lanzado la nueva versión, se enviaban recordatorios a los testers que aún no habían completado la encuesta, para asegurar que todos proporcionan feedback.
- **Análisis de encuestas y toma de decisiones:** Una vez recopiladas todas las encuestas, se revisaba cuidadosamente la retroalimentación recibida. Las opiniones se anotaban para su posterior puesta en común, donde se discutían los cambios que debían implementarse en la siguiente versión. Las ideas nuevas o sugerencias específicas se consideraban dependiendo de la carga de trabajo y la prioridad del momento.

5.3.3. Diseño de encuestas

Con el fin de mejorar el desarrollo y pulir las mecánicas del juego, se llevaron a cabo varias encuestas dirigidas a los testers, quienes participaron activamente en la fase de pruebas. Estas encuestas se diseñaron para recopilar datos cuantitativos y cualitativos acerca de sus experiencias, preferencias y sugerencias. A través de sus respuestas, se obtuvieron *insights* valiosos que permitieron ajustar diferentes aspectos del juego, desde la jugabilidad hasta el equilibrio de los personajes.

Las encuestas se organizaban en varias secciones:

- **Datos personales:** Información básica del jugador, como edad o nivel de estudios, para comprender mejor su perfil.
- **Gustos personales de videojuegos:** Preguntas sobre los géneros de videojuegos que más juegan, para conocer sus preferencias y relacionarlas con sus impresiones del proyecto.
- **Uso de control:** Información sobre el dispositivo de entrada preferido, si usaban mando o teclado, y su experiencia con él.
- **Preguntas sobre nuevas mecánicas:** Opiniones específicas sobre mecánicas recientemente implementadas, su funcionamiento y disfrute.
- **Preguntas sobre mecánicas antiguas:** Revisión de mecánicas ya existentes en el juego, buscando detectar posibles mejoras o errores no reportados anteriormente.
- **Detección de errores en las mecánicas preguntadas:** Espacios para que los testers informen sobre cualquier error o comportamiento no esperado relacionado con las mecánicas discutidas.
- **Opiniones sobre mecánicas o cosas del proyecto para el futuro:** Recolección de sugerencias y feedback sobre cómo podrían evolucionar ciertos aspectos del juego.
- **Aportaciones:** Espacio libre para que los testers compartieran ideas o comentarios adicionales.

Las encuestas completas se encuentran disponibles en el [Anexo](#), donde podrás acceder a ellas a través de los enlaces correspondientes o visualizar capturas de pantalla de las mismas.

6. Validación del proyecto

A continuación, se presentan los resultados más relevantes extraídos de las encuestas desarrolladas en el proyecto, que han servido como guía para las iteraciones en el desarrollo:

Encuesta de datos de los playtesters

- El **90%** son de procedencia **española**, siendo este **10%** de procedencia **americana**. Esto influye en la cultura y gusto, ya que cada país tiene **sus propias culturas y gustos**, los cuales **no siempre coinciden** con el resto del mundo.
- El **20%** de los usuarios eran de **género femenino**, mientras que el **80%** de **género masculino**. Esto debería mostrar que a un género le gusta más este tipo de producto, pero al ser tan poca cantidad de gente no se puede demostrar nada en claro, aunque sea algo que se puede tener mínimamente en cuenta.
- Los estudiantes de **Bachillerato** o menor grado de estudios, suelen jugar unas **10 horas de media a la semana**, mientras que los estudiantes de **Universidad** suelen jugar menos de **5 horas de media a la semana**. Por contraparte los **graduados** suelen jugar más de **20 horas a la semana**.

Con esto, los grupos de estudios menores a **Bachillerato** y los **graduados** han sido los que más tiempo emplean, han sido el grupo que más fuerza han tenido a la hora de aportar más detalles en aspectos técnicos como las mecánicas de juego y los controles, mientras que los **universitarios** han aportado comentarios más generales sobre la experiencia y la accesibilidad del juego sin llegar a entrar mucho en detalle.

- El **100%** de los testers son jugadores casuales en el género **platformer**, mientras que solo un **10%** de los testers son jugadores casuales en el género **fighter**. Esta información proporciona información en los comentarios, ya que toda información que aporten de la parte *platformer* del proyecto será 100% válida, mientras que la parte *fighter* del proyecto se deberá aclarar algunas ideas y mecánicas que suelen estar presentes en este género de juegos que usuarios comunes del género entenderán más fácilmente que aquellos que no estén acostumbrados.

Encuesta de la versión 0.05.22022024

- Un **70%** de los testers **utilizaron mando**, lo que les permitió **experimentar más mecánicas** del juego y ofrecer una **opinión más precisa** sobre el resultado final esperado. Los testers que usaron **un mando de PlayStation** mencionaron que el **tapping se sentía rígido** en ocasiones, mientras que aquellos que usaron un mando de **Nintendo Switch o Xbox** lo encontraron **satisfactorio**.

- La **posibilidad de caminar fue bien recibida**, especialmente el hecho de poder acelerar la velocidad de desplazamiento con el tiempo. Sin embargo, el **60%** de los testers no estuvieron de acuerdo con la mecánica de "**caminar hacia una plataforma sin caerse**", lo que llevó a su **descarte**.
- El **60%** de los usuarios consideraron que el **movimiento lateral era lento**, sugiriendo que la **velocidad debería aumentar entre un 10% y un 20%**.
- La distribución del escenario con **tres plataformas**, fue **ampliamente aceptada**, con pequeñas recomendaciones como aumentar el tamaño del escenario o elevar algunas plataformas.
- El **100%** de los testers coincidió en que el juego debía correr a **60 fps** constantes, un dato crucial, ya que cualquier cambio en la tasa de fotogramas podría haber afectado significativamente las mecánicas relacionadas con el *platforming*.
- De manera general, los testers solicitaron la inclusión de una **visualización de hitboxes, un maniquí para pruebas, y una línea de trayectoria** para medir el salto del personaje o el impacto cuando es golpeado por un rival.

Encuesta de la versión 0.10.15972024

- El **66%** opina que la mecánica de **dash inicial** es **correcta**, aunque se comenta que **podría tener más potencia**.
- El **66%** opina que la mecánica de **frenada** es **correcta**.
- En esta nueva versión se cambió enteramente el salto, añadiendo el ya nombrado **short hop** y **full hop**. Estos fueron bien recibidos por el **100%** de los usuarios, aunque algunos mencionaron que sería útil ajustar los tiempos del salto para que no fueran tan precisos.
- El **movimiento aéreo** en esta versión **se diferenciaba** del movimiento en el suelo, y se observó que, al saltar después de moverse, el personaje ganaba más velocidad. El **33%** de los testers consideró que esta mecánica **estaba bien** como estaba, mientras que el **66%** restante opinó que la **velocidad aérea debería reducirse**, ya que era excesiva.
- Se introdujo la **caída rápida** en esta versión, la cual fue **aceptada** en general, aunque el **100%** de los usuarios coincidió en que la **velocidad de caída era demasiado alta** y debería ser **un poco más lenta**.
- Se añadió una **nueva cámara dinámica** que sigue al jugador. El **66%** la encontró **agradable**, aunque mencionaron que la cámara **se movía demasiado rápido** para seguir de manera efectiva la acción en pantalla.

- El modo "**Maestría de Personaje**" no recibió las opiniones esperadas; solo el **66%** de los testers lo encontró **algo interesante**, pero notaron que **la cámara y los movimientos** del personaje **no estaban bien adaptados** para el propósito del modo.
- Se implementaron nuevos **feedbacks** para el **doble salto y la caída rápida**, que obtuvieron una aceptación del **100%**, ya que fueron considerados **simples, rápidos y fácilmente comprensibles**.

Los resultados de las encuestas se encuentran disponibles en el [Anexo](#), donde podrás visualizar las gráficas de estas mismas.

7. Conclusiones

7.1. Evaluación del proyecto

7.1.1. Desarrollo técnico

A lo largo del desarrollo de mi proyecto, he aprendido varias lecciones clave que me ayudaron a superar los problemas técnicos que enfrenté y a refinar las mecánicas del juego. Aquí comparto algunas de las experiencias más importantes:

- **Colisiones con Rigidbody:** Inicialmente intenté mover al personaje mediante el *transform*, pero me di cuenta de que las colisiones no funcionaban correctamente con esta metodología. El movimiento no se actualizaba bien y resultaba en comportamientos inesperados. Al pasarme a un sistema basado en *Rigidbody*, logré que las colisiones funcionaran correctamente, ya que este método siempre detecta el movimiento cuando se usa junto con otro *GameObject* que tiene un *Collider*.
- **Tapping y velocidad de entrada:** Mi idea inicial para desarrollar el *tapping* era medir la **velocidad** de entrada desde la **zona muerta** hasta la **zona de activación** máxima del personaje. Sin embargo, este método resultó inconsistente. La solución fue contar los **frames** desde que se registra la entrada, y dependiendo del número de frames, ejecutar una acción diferente, lo cual me proporcionó un control más preciso.
- **Salto y gravedad personalizados:** El sistema de física de Unity no ofrecía el nivel de control que necesitaba para mi juego. Esto me llevó a desarrollar un salto y un sistema de gravedad propios, lo que me permitió ajustar de manera avanzada las físicas del personaje y obtener un resultado más controlado y acorde a la sensación que buscaba.
- **Problemas con el nuevo modelo de input manager:** Me topé con complicaciones al intentar implementar soporte para dos mandos utilizando el nuevo sistema de inputs de Unity. Mi solución fue asignar un valor numérico a cada mando y gestionar los inputs inicializando dos gamepads. Aunque descubrí que era posible configurarlo casi sin necesidad de script, opté por un enfoque manual debido a la complejidad de la automatización.
- **Input lag en diferentes mandos:** Al implementar el sistema de salto, me encontré con que cada mando tenía un *input lag* diferente. Esto complicó el desarrollo del **short hop** y **full hop**, ya que debía minimizar el retraso para que no fuera notable. Al investigar más, observé que en otros juegos del mismo género también existe un pequeño delay en el salto, lo que indica que usan una técnica similar a la que desarrollé, contando cuántos frames se presiona el botón de salto para decidir si se ejecuta un **short hop** o un **full hop**.

- **Ignorar colisiones con el suelo:** Un problema que surgió fue que el personaje no podía atravesar ciertas plataformas. Después de investigar, encontré la función **Physics.IgnoreCollision**, que permite que dos colliders se ignoren. Implementé un sistema en el que el personaje detecta cuándo activar o desactivar esta función usando un segundo collider dentro del personaje que actúa como trigger.
- **Colisiones en combates:** Para las colisiones de combate, inicialmente pensaba en usar colisiones normales, activándose y desactivándose según fuera necesario. Cada hueso del personaje tiene un hijo que contiene una colisión en modo "*recibir golpe*". Cuando un enemigo con una colisión de ataque toca esta colisión, se detecta una única vez, lo que resultó ser un enfoque efectivo.
- **Activación de colisiones y eventos:** No sabía cómo activar las colisiones durante los ataques ni cómo hacer que se leyieran los scripts correspondientes. Finalmente, descubrí los **eventos** en las animaciones de Unity, que permiten ejecutar una función en un frame específico. Esto me dio la precisión necesaria para activar las colisiones en el momento exacto del ataque.
- **Knockback:** Pensé que simplemente hacer que el enemigo saliera volando sería suficiente para implementar el **knockback**, pero resultó ser más complicado. Este sistema pasó por varias versiones hasta que encontré una fórmula que se ajustaba a las necesidades del juego y generaba una sensación de impacto y control más satisfactoria.

7.1.2. Iteración basada en feedback

A lo largo de las iteraciones y pruebas con los *playtesters*, fui aprendiendo lecciones clave que moldearon el desarrollo y la dirección de mi juego. El *feedback* continuo jugó un papel crucial en mejorar la experiencia del jugador y ajustar el diseño del juego de acuerdo a las expectativas y necesidades de los usuarios. Aquí algunos de los puntos más destacados:

- **Velocidad y equilibrio del juego:** Uno de los primeros problemas que surgió durante las iteraciones fue que el juego a veces se sentía lento. Al principio no lo consideré un asunto prioritario, pero con el tiempo, y tras recibir múltiples comentarios, me di cuenta de que los jugadores querían una experiencia más **equilibrada, rápida y controlable**. Controlar los *stats* de los personajes parecía sencillo al principio, pero resultó ser un desafío considerable, requiriendo muchos ajustes finos para mantener ese balance entre **fluidez** y **control**.
- **Complejidad de las mecánicas:** A medida que iba iterando, también observé que algunas de las mecánicas o ideas que tenía no resonaban bien con los jugadores. Aunque a mí me parecían interesantes, resultaron ser difíciles de entender o demasiado complicadas para implementar de manera efectiva. Esto

generaba frustración en los testers, lo cual me hizo replantearme el diseño de ciertas funciones para hacerlas más accesibles y disfrutables.

- **Necesidad de control por parte del jugador:** Otro aspecto que surgió del feedback fue el deseo de los jugadores de tener el mayor control posible sobre sus acciones en el juego. Descubrí que era fundamental ofrecerles la posibilidad de personalizar y ajustar ciertos aspectos de la jugabilidad, de modo que se sintieran más conectados con sus decisiones. El control total es una prioridad para los usuarios en este tipo de juegos, y tuve que encontrar maneras de darles esa opción.
- **Física y realismo en el juego:** A pesar de que el juego no busca ser completamente fiel a las leyes físicas del mundo real, los jugadores preferían que las físicas siguieran siendo comprensibles y tangibles dentro de un marco lógico. Me di cuenta de que aunque ciertos elementos podrían ser exagerados o irreales, era importante que estos siguieran teniendo una coherencia intuitiva para los jugadores, lo que evitaba que la experiencia se sintiera surrealista o desconectada de lo que ellos esperaban de un juego de este género.

7.1.3. Logros clave

- Al inicio del proyecto aborde varios objetivos que quería abordar, cada uno a su manera, algunos de estos fueron:
Desarrollo de un Game Design Document: uno de los objetivos principales de este proyecto era desarrollar un *GDD*, el cual pudiera abordar incongruencias comunes en las mecánicas y jugabilidad de este género, ofreciendo así una referencia útil para futuros desarrolladores.
- **Equilibrio entre jugadores:** a lo largo del desarrollo, he ajustado las mecánicas para asegurar que el juego sea atractivo tanto para jugadores casuales como hardcore. Pese que en algunos casos, no todas las mecánicas eran del agrado de los dos tipos de jugadores, este equilibrio ha sido mantenido para evitar que los jugadores se sientan aburridos o frustrados, y contribuya a crear una experiencia de juego más inclusiva y satisfactoria.
- **Comunicación con la comunidad:** he conseguido establecer un grupo de *playtesters* y hecho uso de encuestas para recoger sus opiniones, lo que ha permitido que la comunidad participe activamente en el proceso de desarrollo. Esta colaboración ha sido esencial para realizar ajustes y mejoras en el juego, asegurando que el producto final refleje las preferencias y necesidades de los jugadores.
- **Desarrollo de un proyecto propio:** he desarrollado un proyecto que ha sido reconocido por la comunidad y que sirve como un proyecto principal. Este hecho me ha permitido consolidar un producto que pueda seguir desarrollando y en un futuro lanzarlo al mercado.

7.1.4. Conclusiones generales

A lo largo del desarrollo de este proyecto, he recorrido un proceso complejo y enriquecedor que me ha permitido no solo crear un juego del género platform fighter, sino también aprender valiosas lecciones en torno al diseño de videojuegos, la retroalimentación de la comunidad y la iteración constante para mejorar la experiencia de juego. Este proyecto ha sido una búsqueda de equilibrio, innovación y una reflexión sobre cómo unir las expectativas de jugadores casuales y hardcore en un solo título.

En términos de desarrollo técnico, aprendí la importancia de crear sistemas personalizados, desde un sistema propio de salto y gravedad hasta un manejo más sofisticado de las colisiones mediante *rigidbody*. Este proceso no fue sencillo, pero me permitió desarrollar un control preciso sobre la jugabilidad, algo esencial en un juego donde la física y el movimiento tienen un papel tan destacado. También fue necesario lidiar con aspectos como el *input lag* de los mandos y ajustar la mecánica de *tapping* para lograr una transición suave entre caminar y correr, lo cual me dio una perspectiva más profunda sobre los desafíos técnicos que conlleva implementar sistemas responsivos.

Además, uno de los aspectos más reveladores ha sido la importancia del feedback. La interacción con la comunidad de *playtesters* a través de encuestas y pruebas me ayudó a identificar áreas de mejora y a ajustar el balance del juego. Al principio, no era evidente lo crucial que sería el equilibrio de velocidad y la implementación de nuevas mecánicas, pero las iteraciones me mostraron lo esencial que es escuchar a los jugadores y adaptar el diseño a sus expectativas. El juego ha evolucionado gracias a la interacción constante con los usuarios, quienes me ayudaron a identificar qué elementos no funcionaban como se esperaba y a mejorar la fluidez general del juego.

Finalmente, este proyecto no solo ha sido una meta personal de desarrollo técnico y diseño, sino también una plataforma de aprendizaje sobre cómo gestionar una comunidad y cómo iterar constantemente basándome en la retroalimentación de los jugadores. El camino no ha sido sencillo, pero me ha permitido desarrollar un producto que representa no solo mi visión, sino también la de la comunidad, y que tiene el potencial de seguir creciendo y mejorando en futuras iteraciones. Lo aprendido a nivel técnico y en cuanto a la interacción con la comunidad será fundamental en mi trayectoria como desarrollador de videojuegos.

7.2. Líneas de futuro

7.2.1. Mejoras potenciales

- Mejorar la movilidad en tierra, que a veces se siente algo rígida, especialmente al jugar con mando.
- Optimizar el sistema de colisiones con el suelo para lograr una detección más precisa, permitiendo que los personajes puedan moverse por rampas y escaleras sin ralentizarse o presentar problemas.
- Implementar un sistema de *hitboxes* más avanzado, separando las *hurtboxes* (cajas que reciben daño) de las *hitboxes* (cajas que infligen daño), lo que permitirá una mayor personalización y precisión en los combates.
- Refinar el sistema de *knockback*, permitiendo que los personajes reboten al ser golpeados contra paredes o techos y que se puedan encadenar varios golpes con distintos movimientos.
- Ajustar la duración de las animaciones, ya que muchas duran 24 frames. Aumentar la variedad de tiempos animará a una jugabilidad más dinámica.
- Revisar el diseño del concepto y los escenarios "**Maestría de personaje**", añadiendo un sistema de puntos y nuevas ideas que incentive un mayor desafío y entretenimiento.

7.2.2. Nuevas funcionalidades

- Incluir un ataque de carrera (*dash attack*) que se pueda ejecutar mientras el personaje está corriendo.
- Incorporar **ataques especiales** únicos para cada personaje, que añadan más profundidad estratégica.
- Añadir la mecánica de **escudo** y sus respectivas **esquivas laterales** para aumentar las opciones defensivas.
- Introducir la habilidad de **esquivar en el aire**, ampliando la movilidad y el dinamismo durante los combates.
- Implementar la posibilidad de **agarrar** a los oponentes, ofreciendo una nueva dimensión táctica.
- Desarrollar un "**ataque final**" que se cargue con el tiempo, añadiendo una capa extra de estrategia a las partidas.

7.2.3. Planes cara a futuro

En el futuro, tengo la intención de colaborar con una *concept artist* que además es guionista, para desarrollar una narrativa sólida para el juego. Esta colaboración permitirá crear un modo historia y una plantilla de personajes con trasfondos ricos. La idea es que todos los personajes pertenezcan a una escuela secundaria japonesa, típica de las historias de dibujos animados japoneses llamados *anime*, representando distintos grupos sociales o clubes, lo que influirá tanto en su personalidad como en su estilo de combate.

Aunque esta parte está aún en desarrollo, ya hemos comenzado a planificarlo con la visión de convertir el juego en un producto comercializable, que no solo sirva como una pieza clave en nuestros portfolios, sino que también ofrezca una experiencia atractiva y divertida para los jugadores.

8. Bibliografía

8.1. Artículos

ACERCA DEL JUEGO. (s. f.). MULTIVERSUS. <https://multiversus.com/es-es/game>

Brawlhalla - the free to play fighting game. (2024, 29 mayo). Brawlhalla.

<https://www.brawlhalla.com/>

colaboradores de Wikipedia. (2023, 29 diciembre). *Masahiro Sakurai*. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Masahiro_Sakurai

Como crear un documento de diseño de videojuegos (GDD). (s. f.).

<http://www.hagamosvideojuegos.com/2015/06/como-crear-un-documento-de-diseno-de.html>

Definir la audiencia objetivo de un juego indie – CEOindie.me. (2017, 12 septiembre).

<https://ceoindie.me/2017/09/12/definir-la-audiencia-objetivo-de-un-juego-indie/>

Entradas, V. M. (2016, 11 abril). *Documentación en Videojuegos: Documento de diseño (GDD)*. EL DOCUMENTALISTA AUDIOVISUAL.

<https://eldocumentalistaaudiovisual.com/2015/02/06/documentacion-en-videojuegos-documento-de-diseno-gdd/>

Fallout-Brotherhood-of-Steel-2-Design-Document.pdf. (s. f.). Google Docs.

<https://drive.google.com/file/d/1b6TVJHAjtsK12qmDn9M8CdoUHS1OHwY/view>

Game Marketing Genie. (2020, 27 marzo). *How to Create Player Buyer Personas For Video Game Marketing.*

<https://www.gamemarketinggenie.com/blog/create-player-buyer-personas>

Gerstmann, J. (1999, 18 febrero). Super Smash Bros. Review. *GameSpot*.

<https://www.gamespot.com/reviews/super-smash-bros-review/1900-2543713/>

Home. (s. f.). <https://rivalsofaether.com/#zetterburn>

Mizuumi. (2023, 16 octubre). *Rivals of Aether / System - Mizuumi, fighting games wiki*. Mizuumi.

https://wiki.gbl.gg/w/Rivals_of_Aether/System#:~:text=also%20inflict%20dama

ge.-, Knockback, Damage%20%%20KB_Scaling%20%%200.12%20*%20KB_Adjustment

Muriel Garreta Domingo, Enric Mor Pera. (2011). *Diseño centrado en el usuario - Universitat Oberta de Catalunya*

https://openaccess.uoc.edu/bitstream/10609/76105/6/Introducci%C3%B3n%20la%20interacci%C3%B3n%20persona%20ordenador_M%C3%B3dulo%203_Dise%C3%B1o%20centrado%20en%20el%20usuario.pdf

Nickelodeon All-Star Brawl 2. (2024, 4 marzo). All-Star Brawl.

<https://nickelodeonallstarbrawl.com/>

Normandy, R. O. (2012, 4 agosto). *Smash Jumping physics - How do they work? Smashboards.*

<https://smashboards.com/threads/smash-jumping-physics-how-do-they-work.325557/>

Popcron. (s. f.). *GitHub - popcron/extra-controllers: Adds support for GameCube controllers to the unity input system along with other controllers.* GitHub.
<https://github.com/popcron/extra-controllers>

Sickr. (2018, 19 julio). *Sakurai Shares His Thoughts On The Reception Of Super Smash Bros Ultimate - My Nintendo News.* My Nintendo News.
<https://mynintendonews.com/2018/07/19/sakurai-shares-his-thoughts-on-the-reception-of-super-smash-bros-ultimate/>

Smash Bros. DOJO!! (s. f.-a).

https://www.smashbros.com/wii/en_us/howto/basic/basic01.html

Smash Bros. DOJO!! (s. f.-b).

https://www.smashbros.com/wii/en_us/howto/basic/basic03.html

Smash Bros. DOJO!! (s. f.-c).

https://www.smashbros.com/wii/en_us/howto/basic/basic04.html

Smash Bros. DOJO!! (s. f.-d).

https://www.smashbros.com/wii/en_us/howto/basic/basic05.html

Smash Bros. DOJO!! (s. f.-e).

https://www.smashbros.com/wii/en_us/howto/basic/basic06.html

Smash Bros. DOJO!! (s. f.-f).

https://www.smashbros.com/wii/en_us/howto/basic/basic08.html

Smash Bros. DOJO!! (s. f.-g).

https://www.smashbros.com/wii/en_us/howto/basic/basic09.html

Smash Bros. DOJO!! (s. f.-h).

https://www.smashbros.com/wii/en_us/howto/technique/technique11.html

Smash Bros. DOJO!! (s. f.-i).

https://www.smashbros.com/wii/en_us/gamemode/various/various14.html

Smash Bros. DOJO!! (s. f.-j).

https://www.smashbros.com/wii/en_us/howto/basic/basic07.html

Smashpedia, C. T. (s. f.). *Game Mechanics*. Smashpedia.

https://supersmashbros.fandom.com/wiki/Category:Game_Mechanics

SmashWiki. (2013, 27 marzo). *Category:Hitboxes - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Category:Hitboxes>

SmashWiki. (2018, 3 marzo). *Category:Gameplay - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Category:Gameplay>

SmashWiki. (2020, 6 febrero). *Attack - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Attack>

SmashWiki. (2021a, septiembre 1). *Category:Game physics - SmashWiki, the Super Smash Bros. wiki*. SmashWiki.

https://www.ssbwiki.com/Category:Game_physics

SmashWiki. (2021b, septiembre 1). *Category:Techniques - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Category:Techniques>

SmashWiki. (2022a, octubre 23). *Walk - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Walk>

SmashWiki. (2022b, diciembre 2). *Up special move - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Up_special_move

SmashWiki. (2022c, diciembre 25). *Forward throw - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Forward_throw

SmashWiki. (2023a, enero 14). *Move (disambiguation) - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. [https://www.ssbwiki.com/Move_\(disambiguation\)](https://www.ssbwiki.com/Move_(disambiguation))

SmashWiki. (2023b, febrero 14). *Gravity - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Gravity>

SmashWiki. (2023c, abril 12). *Air dodge - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Air_dodge

SmashWiki. (2023d, abril 12). *Hitlag - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Hitlag>

SmashWiki. (2023e, junio 7). *Forward aerial - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Forward_aerial

SmashWiki. (2023f, junio 28). *Back aerial - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Back_aerial

SmashWiki. (2023g, agosto 16). *Down throw - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Down_throw

SmashWiki. (2023h, octubre 9). *Back throw - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Back_throw

SmashWiki. (2023i, octubre 21). *Traction - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Traction>

SmashWiki. (2023j, noviembre 7). *Down special move - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Down_special_move

SmashWiki. (2023k, noviembre 28). *Down tilt - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Down_tilt

SmashWiki. (2023l, noviembre 28). *Smash attack - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. https://www.ssbwiki.com/Smash_attack

SmashWiki. (2023m, diciembre 5). *Wavedash - SmashWiki, the Super Smash Bros. wiki*. SmashWiki. <https://www.ssbwiki.com/Wavedash>

SmashWiki. (2023n, diciembre 27). *Damage - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Damage>

SmashWiki. (2024a, enero 6). *Neutral special move - SmashWiki, the Super Smash Bros. wiki.* SmashWiki. https://www.ssbbwiki.com/Neutral_special_move

SmashWiki. (2024b, enero 14). *Side special move - SmashWiki, the Super Smash Bros. wiki.* SmashWiki. https://www.ssbbwiki.com/Side_special_move

SmashWiki. (2024c, enero 27). *Pummel - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Pummel>

SmashWiki. (2024d, enero 28). *Roll - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Roll>

SmashWiki. (2024e, febrero 4). *Shield - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Shield>

SmashWiki. (2024f, febrero 10). *Pause - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Pause>

SmashWiki. (2024g, febrero 12). *Pivoting - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Pivoting>

SmashWiki. (2024h, febrero 24). *Up tilt - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Up_tilt

SmashWiki. (2024i, marzo 3). *Hitstun - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Hitstun>

SmashWiki. (2024j, marzo 4). *Forward tilt - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Forward_tilt

SmashWiki. (2024k, marzo 4). *Up throw - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Up_throw

SmashWiki. (2024l, marzo 10). *Wall jump - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Wall_jump

SmashWiki. (2024m, marzo 14). *Knockback - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Knockback>

SmashWiki. (2024n, marzo 28). *Forward smash - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Forward_smash

SmashWiki. (2024o, abril 6). *Taunt - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Taunt>

SmashWiki. (2024p, abril 10). *Spot dodge - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Spot_dodge

SmashWiki. (2024q, abril 14). *Versus Mode - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Versus_Mode

SmashWiki. (2024r, abril 27). *Sweet spot (hitbox) - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. [https://www.ssbbwiki.com/Sweet_spot_\(hitbox\)](https://www.ssbbwiki.com/Sweet_spot_(hitbox))

SmashWiki. (2024s, mayo 1). *Grab - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Grab>

SmashWiki. (2024t, mayo 1). *Throw - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Throw>

SmashWiki. (2024u, mayo 1). *Time - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Time>

SmashWiki. (2024v, mayo 3). *KO - SmashWiki, the Super Smash Bros. wiki.* SmashWiki.
<https://www.ssbbwiki.com/KO>

SmashWiki. (2024w, mayo 5). *Jump - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Jump>

SmashWiki. (2024x, mayo 5). *Mine / Craft / Create Block - SmashWiki, the Super Smash Bros. wiki.* SmashWiki. https://www.ssbbwiki.com/Mine_-_Craft_-_Create_Block

SmashWiki. (2024y, mayo 5). *Weight - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Weight>

SmashWiki. (2024z, mayo 9). *Home-Run Contest - SmashWiki, the Super Smash Bros. wiki.* SmashWiki. https://www.ssbbwiki.com/Home-Run_Contest

SmashWiki. (2024aa, mayo 11). *Down aerial - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Down_aerial

SmashWiki. (2024ab, mayo 11). *Down smash - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Down_smash

SmashWiki. (2024ac, mayo 11). *Up aerial - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Up_aerial

SmashWiki. (2024ad, mayo 11). *Up smash - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Up_smash

SmashWiki. (2024ae, mayo 12). *Dash attack - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Dash_attack

SmashWiki. (2024af, mayo 13). *Edge - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Edge>

SmashWiki. (2024ag, mayo 13). *Target Smash! - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Target_Smash

SmashWiki. (2024ah, mayo 16). *Adventure Mode: The Subspace Emissary - SmashWiki, the Super Smash Bros. wiki.* SmashWiki.
https://www.ssbbwiki.com/Adventure_Mode:_The_Subspace_Emissary

SmashWiki. (2024ai, mayo 16). *Dash - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Dash>

SmashWiki. (2024aj, mayo 19). *Tech - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Tech>

SmashWiki. (2024ak, mayo 20). *Mode - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. https://www.ssbbwiki.com/Mode#In_Super_Smash_Bros._Ultimate

SmashWiki. (2024al, mayo 23). *Item - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Item>

SmashWiki. (2024am, mayo 26). *Controller - SmashWiki, the Super Smash Bros. wiki.*
SmashWiki. <https://www.ssbbwiki.com/Controller>

Stages. (s. f.). smashbrothers.at. <https://smashbrothers.at/en/rules/stages/>

Stages – Rivals of Aether. (s. f.). <https://rivalsofaether.com/stages/>

Stats - Ultimate Frame Data. (s. f.) <https://ultimateframedata.com/stats>

SuperComboWiki. (2023, diciembre 2). *Nickelodeon All-Star Brawl 2 / Modes - SuperCombo Wiki, a repository for all fighting game strategy knowledge.*

SuperComboWiki.

https://wiki.supercombo.gg/w/Nickelodeon_All-Star_Brawl_2/Modes

SuperComboWiki. (2023, diciembre 22). *Nickelodeon All-Star Brawl 2 / Stages - SuperCombo Wiki, a repository for all fighting game strategy knowledge.*

SuperComboWiki.

https://wiki.supercombo.gg/w/Nickelodeon_All-Star_Brawl_2/Stages

Super Smash Bros DOJO!! (2000, febrero 4). *¡Secreto! 77 técnicas de puño de Smash Bros. - Super Smash Bros DOJO!!*

https://www-nintendo-co-jp.translate.goog/n01/n64/software/nus_p_nalj/smash/LvFrameLayout.html?_x_tr_sl=ja&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=wapp

Super Smash Bros. DX Dojo!! (2002) *¡¡Puño de recuento de encuestas!! Questionnaire - Super Smash Bros. DX Dojo!!*

https://www-nintendo-co-jp.translate.goog/n01/n64/software/nus_p_nalj/smash/flash/syukeiken/return583.html?_x_tr_sl=ja&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=wapp

Super Smash Bros. for Nintendo 3DS / Wii U: Smash. (s. f.). www.smashbros.com.

<https://www.smashbros.com/wiiu-3ds/es/howto/entry9.html>

Super Smash Bros. N64 Review - IGN. (2018, 4 diciembre). IGN.

<https://www.ign.com/articles/1999/04/28/super-smash-bros>

Super Smash Bros. Ultimate. (s. f.). https://www.smashbros.com/es_ES/

Super Smash Bros. Ultimate Stage Comparison Tool. (s. f.). Stages.

<https://www.tournameta.com/stage-comparison/>

Super Smash Brothers - GameCritics.com. (s. f.).

<https://web.archive.org/web/20120225111748/http://www.gamecritics.com/review/smashbros/main.php>

The Fighting Game Glossary / Infil.net. (s. f.). <https://glossary.infil.net/>

Ubisoft - Brawlhalla. (s. f.). Ubisoft.com.

<https://www.ubisoft.com/es-es/game/brawlhalla/brawlhalla#1IX6UQ0eSv9zdCgmQwAPv7>

Video Game Insights - Games industry data and analysis. (s. f.-a). Video Game Insights.

<https://vginsights.com/game/383980>

Video Game Insights - Games industry data and analysis. (s. f.-b). Video Game Insights.

<https://vginsights.com/game/2017080>

Video Game Insights - Games industry data and analysis. (s. f.-c). Video Game Insights.

<https://vginsights.com/game/1818750>

Wiki, C. T. B. (s. f.). *Modes.* Brawlhalla Wiki.

<https://brawlhalla.fandom.com/wiki/Modes>

Wiki, C. T. M. (s. f.). *Game Modes.* MultiVersus Wiki.

https://multiversus.fandom.com/wiki/Game_Modes

Wiki, C. T. N. (s. f.). *Masahiro Sakurai.* Nintendo Wiki.

https://nintendo.fandom.com/es/wiki/Masahiro_Sakurai

Wiki, C. T. O. (s. f.). *Hero Mastery.* Overwatch Wiki.

https://overwatch.fandom.com/wiki/Hero_Mastery

Wiki, C. T. R. o. A. (s. f.-a). *Abyss endless.* Rivals Of Aether Wiki.

https://rivals-of-aether.fandom.com/wiki/Abyss_Endless

Wiki, C. T. R. o. A. (s. f.-b). *Rivals of Tether.* Rivals Of Aether Wiki.

https://rivals-of-aether.fandom.com/wiki/Rivals_of_Tether

Wiki, C. T. R. o. A. (s. f.-c). *Story mode.* Rivals Of Aether Wiki.

https://rivals-of-aether.fandom.com/wiki/Story_Mode

tu. (2019, septiembre 9). [CEDEC 2019] ¿Cuáles son algunas ideas para batallas justas entre personajes de varias IP? El equipo de desarrollo habla sobre la creación de imágenes de "Super Smash Bros. Special) - Gamer.net

<https://www.4gamer.net/games/412/G041234/20190906163/>

S, M. A. (2019, 13 abril). Los 12 principios de animación. *HyperBlog*.
<https://www.tesseractspace.com/blog/los-12-principios-de-animation/>

8.2. Videos

Masahiro Sakurai on Creating Games. (2022a, septiembre 7). *Draw the Light, Not the Asset [Graphics]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=FuAtKjEuck8>

Masahiro Sakurai on Creating Games. (2022b, septiembre 18). *Assigning animations [Animation]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=fV8xIP480qk>

Masahiro Sakurai on Creating Games. (2022c, septiembre 23). *Make it «Pop» [Effects]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=kcYDrtRvuKg>

Masahiro Sakurai on Creating Games. (2022d, septiembre 28). *Clarity vs. Style [UI]* [Vídeo]. YouTube. https://www.youtube.com/watch?v=UjW_TTNtXEM

Masahiro Sakurai on Creating Games. (2022e, octubre 20). *Super Smash Bros. [Game Concepts]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=i3IOWaVDbx0>

Masahiro Sakurai on Creating Games. (2022f, octubre 28). *Emphasize Objects with Collision [Graphics]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=FfPN4ZGgBpo>

Masahiro Sakurai on Creating Games. (2022g, noviembre 11). *Breaking Down Attack Animations [Animation]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=LewXWM7HDd8>

Masahiro Sakurai on Creating Games. (2022h, noviembre 30). *Distinguishing Between Major and Minor Elements [Graphics]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=NBXt-Uq7dHg>

Masahiro Sakurai on Creating Games. (2022i, diciembre 14). *Making Lead-ins Instant and Impactful [Animation]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=E8DKndKkHw8>

Masahiro Sakurai on Creating Games. (2023a, enero 16). *Attack poses [Animation]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=99gdMDF7V2E>

Masahiro Sakurai on Creating Games. (2023b, febrero 22). *Follow-Throughs Make the Impact [Animation]* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=cIB0BUe6lhk>

Masahiro Sakurai on Creating Games. (2023c, mayo 5). *Screen shake [Effects]* [Vídeo].
YouTube. <https://www.youtube.com/watch?v=2JXR7IASog>

Masahiro Sakurai on Creating Games. (2023d, mayo 13). *Posing suggestions [Animation]* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=LsSF_H-W-w8

Masahiro Sakurai on Creating Games. (2023e, julio 25). *Hit marks [Effects]* [Vídeo].
YouTube. <https://www.youtube.com/watch?v=B-P4ysHSjCg>

Masahiro Sakurai on Creating Games. (2023f, agosto 1). *Color-Coding your Game [UI]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=CWvLMK4OKko>

Masahiro Sakurai on Creating Games. (2023g, agosto 18). *A Fight Between Live Action and Animation [Graphics]* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=lub-5GqW55Q>

Masahiro Sakurai on Creating Games. (2023h, agosto 29). *Damage Animations [Animation]* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=0xHE3ypX96U>

Masahiro Sakurai on Creating Games. (2023i, septiembre 8). *Knockback in Super Smash Bros. [Programming & Tech]* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=HVktK4a9Yfo>

Masahiro Sakurai on Creating Games. (2023j, septiembre 15). *Make important elements bigger [UI]* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=va8wt2yGviY>

Masahiro Sakurai on Creating Games. (2023k, octubre 6). *Billboards [Effects]* [Vídeo].
YouTube. https://www.youtube.com/watch?v=_o29OOOarPY

Masahiro Sakurai on Creating Games. (2024a, enero 12). *Flipped Animation [Animation]* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=TJDMDxnWU5g>

Masahiro Sakurai on Creating Games. (2024b, marzo 8). *Button Settings [UI]* [Vídeo].
YouTube. <https://www.youtube.com/watch?v=8nfMt4oG6hg>

NickLeo. (2023, 10 noviembre). *BEGINNER GUIDE to Nickelodeon All-Star Brawl 2 - Mastery Series: Part 1* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=Q6X4xhVMakg>

RelaxAlax. (2024, 15 abril). *Super Smash Bros. UI* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=lxeSnOquNcU>

9. Anexos

Encuestas de playtesters

En el siguiente enlace se puede encontrar la encuesta “**Información personal playtesters**”: [Encuesta 1](#).

En el siguiente enlace se puede encontrar la encuesta “**Playtesters v0.05.22022023**”: [Encuesta 2](#).

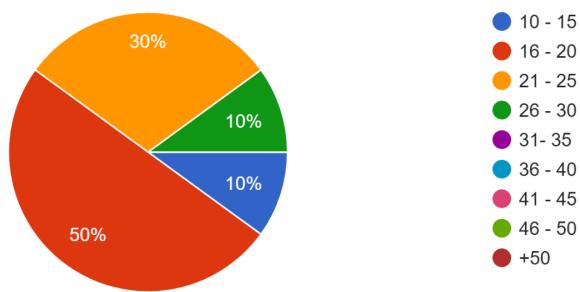
En el siguiente enlace se puede encontrar la encuesta “**Playtesters v0.10.15072024**”: [Encuesta 3](#).

Resultados encuestas de playtesters

Información personal playtesters

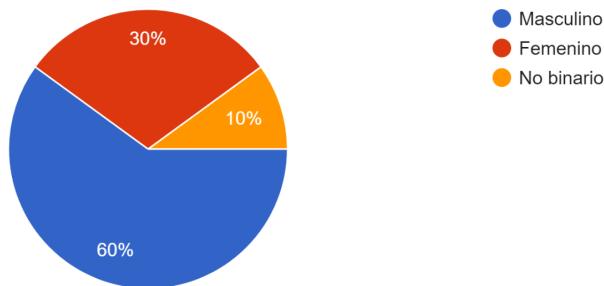
¿Quedad tienes?

10 respuestas



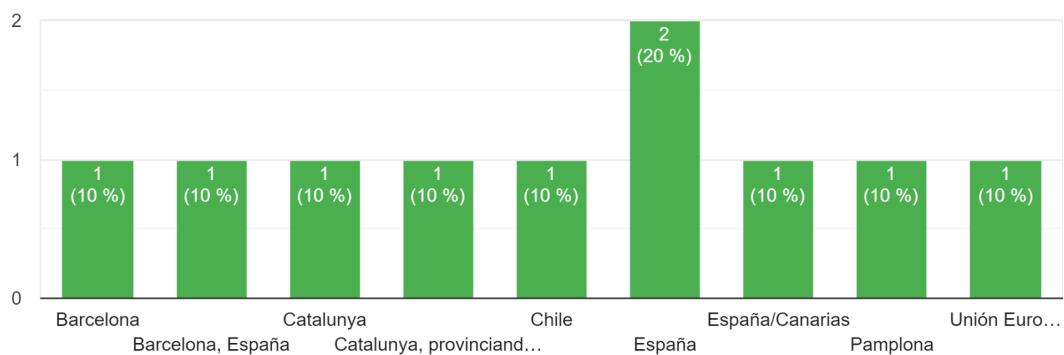
¿Con que género te identificas?

10 respuestas



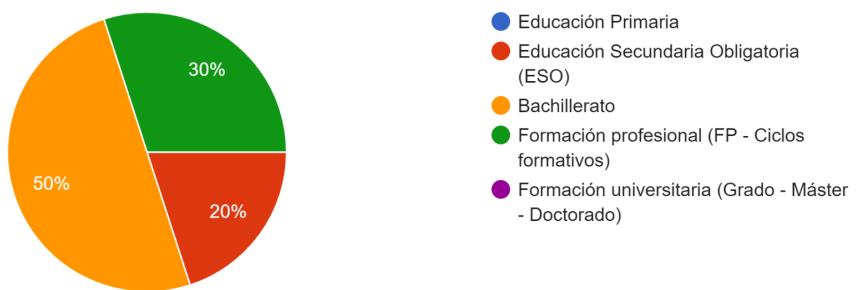
¿De dónde eres?

10 respuestas



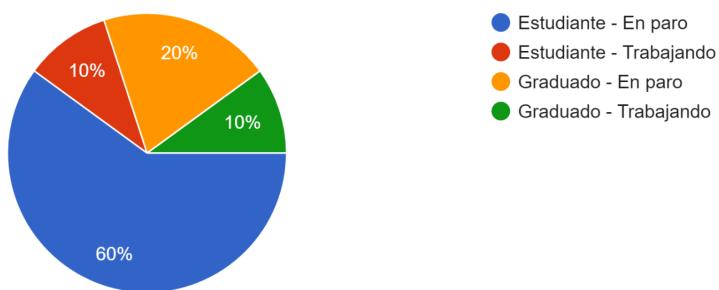
¿Cuál es tu nivel de estudios?

10 respuestas



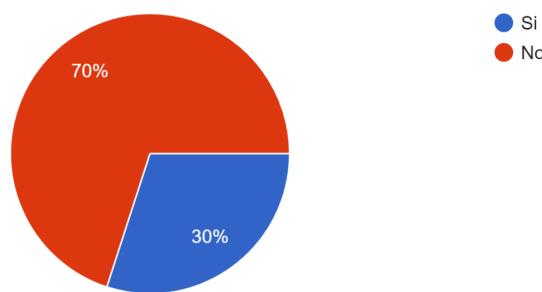
¿Cuál es tu situación laboral?

10 respuestas



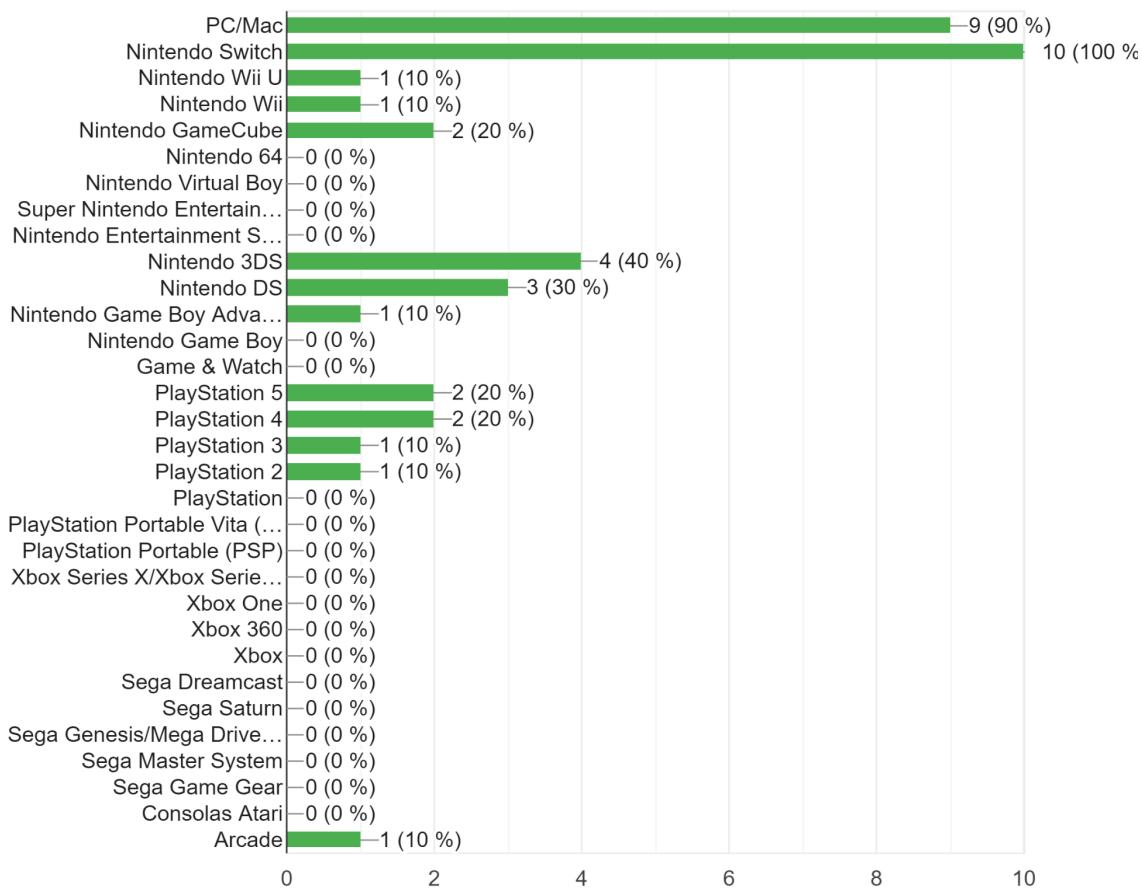
¿Tienes experiencia previa como playtester?

10 respuestas



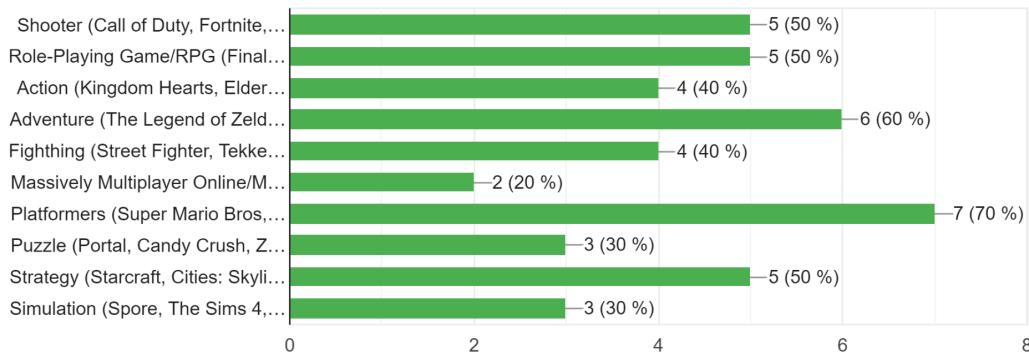
¿En que plataformas sueles jugar? Todo aquel juego que juguéis de dicha plataforma cuenta como que jugáis a esa plataforma también.

10 respuestas



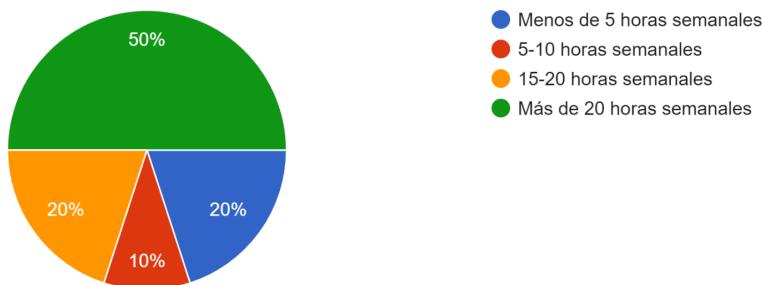
¿Qué género de juegos son tus preferidos?

10 respuestas



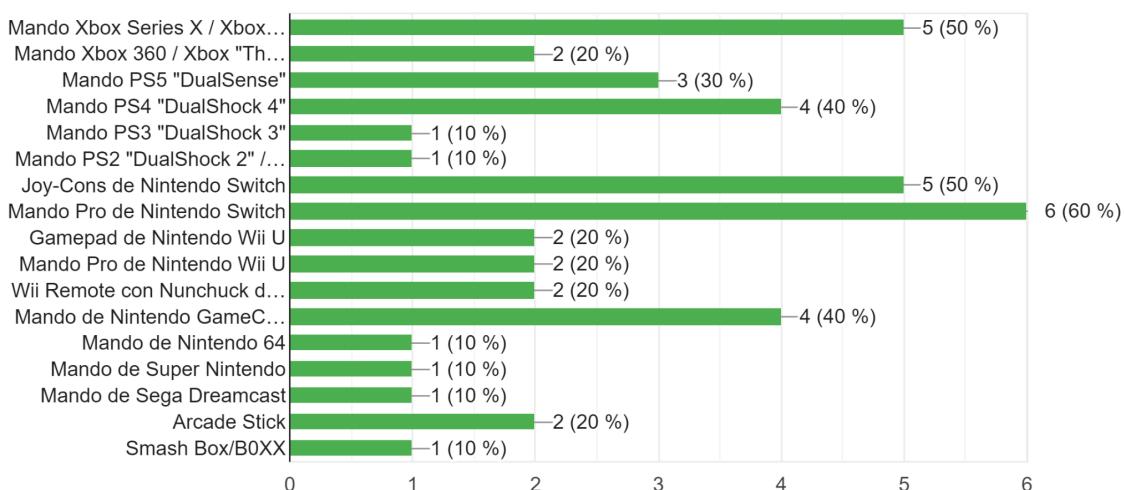
¿Cuánto tiempo sueles jugar semanalmente?

10 respuestas

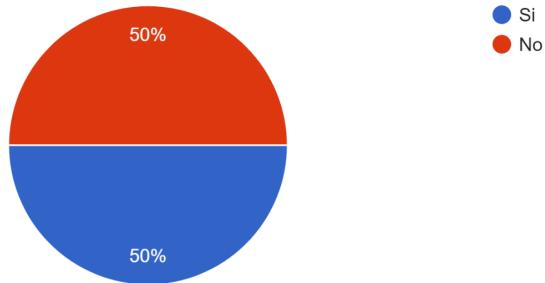


¿De qué mandos dispones que se puedan conectar al ordenador?

10 respuestas



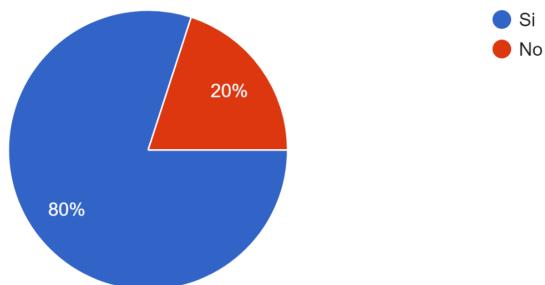
¿Quieres que se te reconozca con tu nick y se te mantenga en el anonimato? (Ten en cuenta que cualquier dato tuyo puede ser mostrado en el proyecto...rán todos los datos tuyos personales irrelevantes.
10 respuestas



Playtesters v0.05.22022023

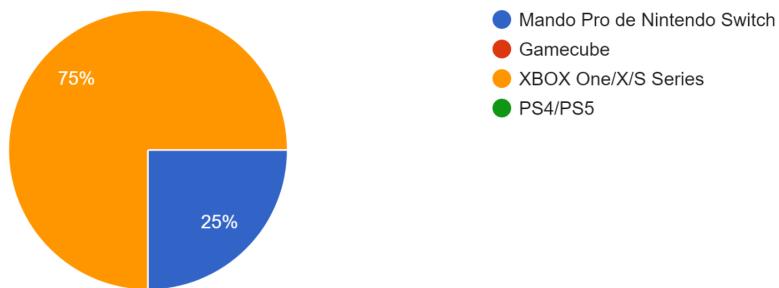
¿Has utilizado mando?

5 respuestas



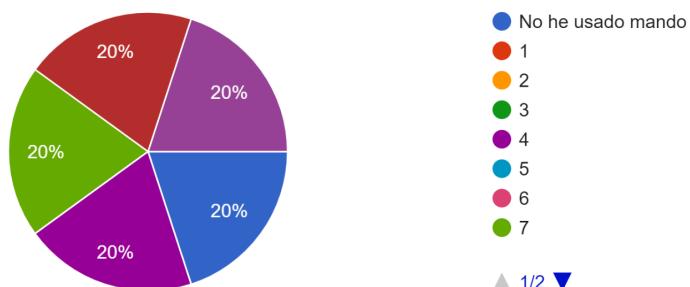
Si has contestado Si, ¿Qué mando has utilizado?

4 respuestas



En esta versión, se ha intentado recrear el tapping, para que los jugadores con mandos puedan realizar más acciones como andar, correr o bajar d...lataforma. Del 1 - 10, ¿Qué tan cómodo era usarlo?

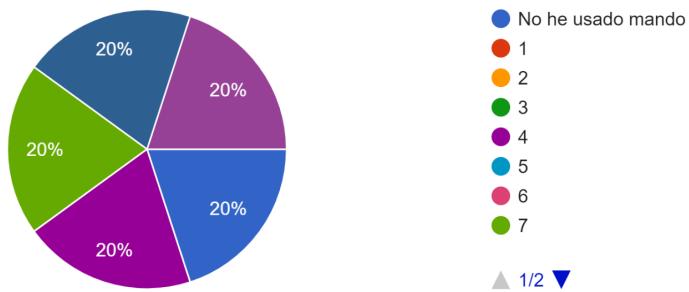
5 respuestas



▲ 1/2 ▼

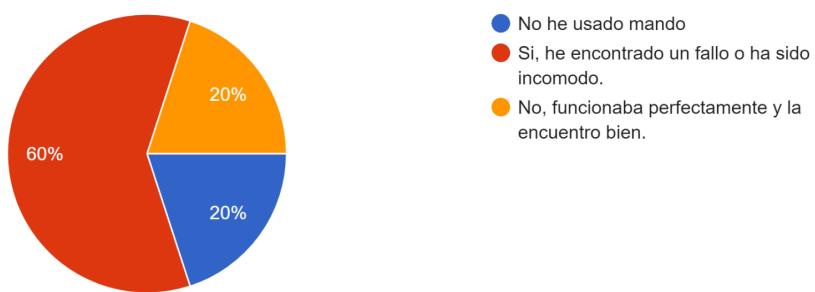
Para andar, se debe de mover el joystick izquierdo en horizontal lentamente. Del 1 - 10, ¿Qué tal se siente el poder acelerar el andar?

5 respuestas



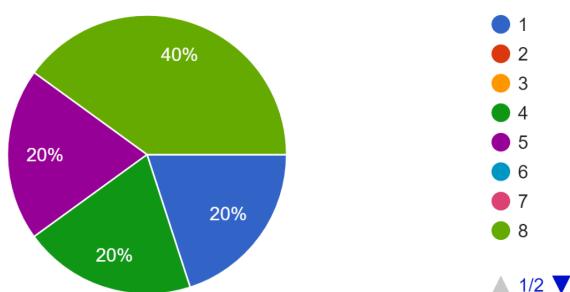
Si estás encima de una plataforma y vas andando hacia una de las puntas, el personaje no caerá y se quedará estático allí hasta que corras o salte...ta mecánica o ha sido incomoda en algún momento?

5 respuestas



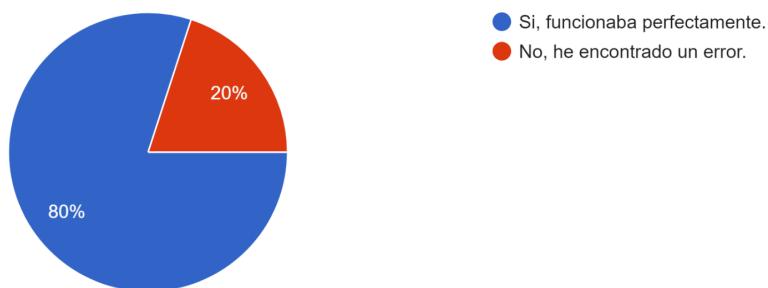
Para correr, se debe mover el joystick izquierdo en horizontal rápidamente hacia una de las puntas. Del 1 - 10, ¿Qué tal se siente el poder correr?

5 respuestas



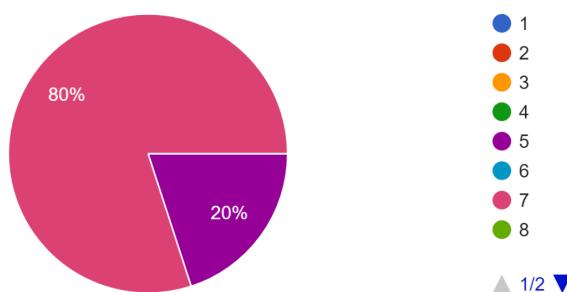
Para bajar de una plataforma, se debe de mover el joystick izquierdo rápidamente hacia la punta de abajo. ¿Lo has encontrado correcto su funcionamiento?

5 respuestas



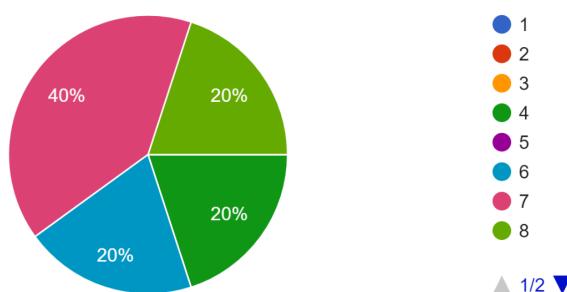
En esta versión, se podía saltar pulsando el botón Y/X/Triángulo/Cuadrado/I. Del 1 al 10, ¿Qué tan cómodo te ha parecido el salto y el doble salto?

5 respuestas



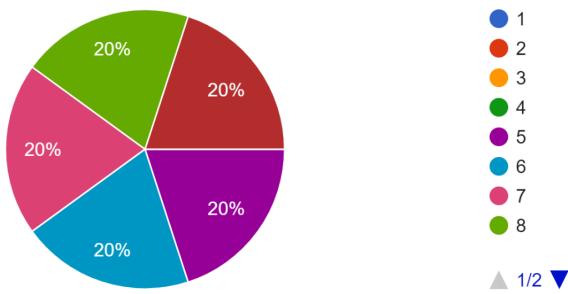
Después de un salto o al dejarte caer de una plataforma existe una gravedad. Del 1 al 10, ¿Qué tan suave o fluido te ha parecido este?

5 respuestas



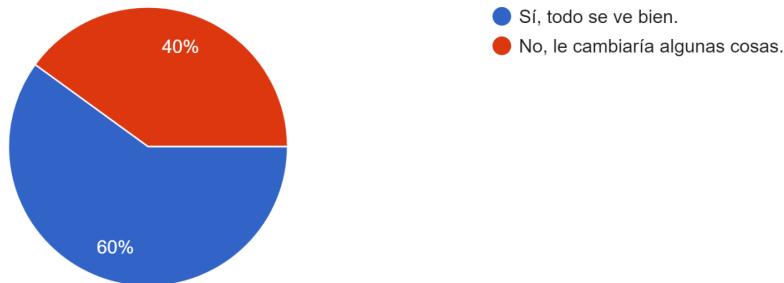
Si saltas debajo de una plataforma fina, esta la podrás traspasar. Del 1 al 10, ¿Qué te han bien ha funcionado?

5 respuestas



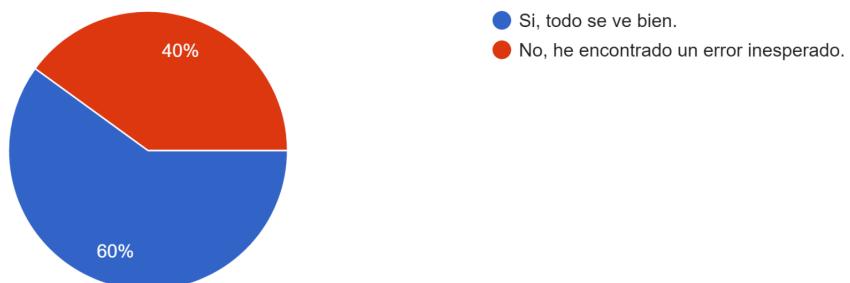
Como escenario base se ha escogido un el modelo Battlefield, común en este tipo de juegos. ¿Te parece que las alturas de las plataformas y el tamaño...ambiar su posición, las que hay son provisionales.

5 respuestas



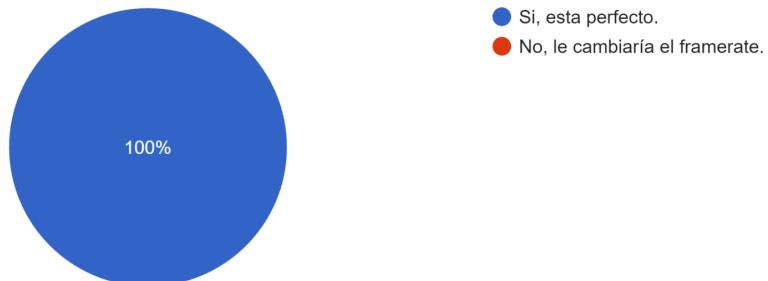
Las zonas de muerte que hay ahora mismo son provisionales, pero tienen la funcionalidad de que reaparezcas y añadirte daño al contador de abajo...as encontrado que funcionase todo correctamente?

5 respuestas



Este juego corre de normal a 60fps, ¿Opinas que esta velocidad para este tipo de juego es correcto o se la cambiarías?

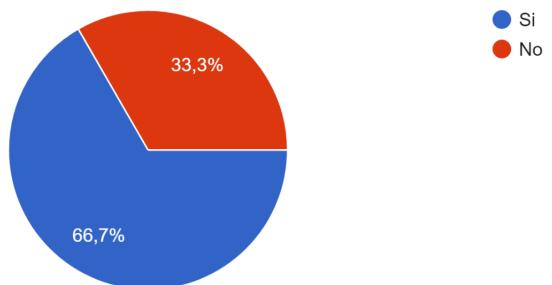
5 respuestas



Playtesters v0.10.15072024

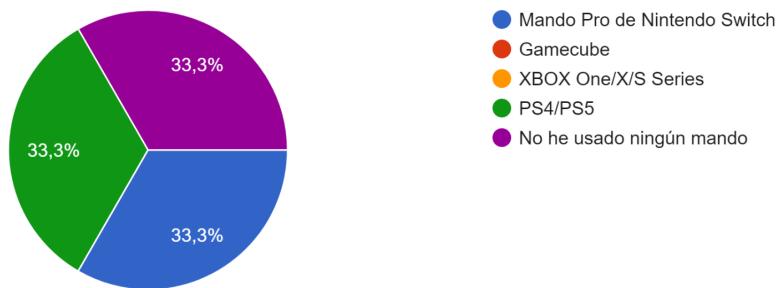
¿Has utilizado mando?

3 respuestas



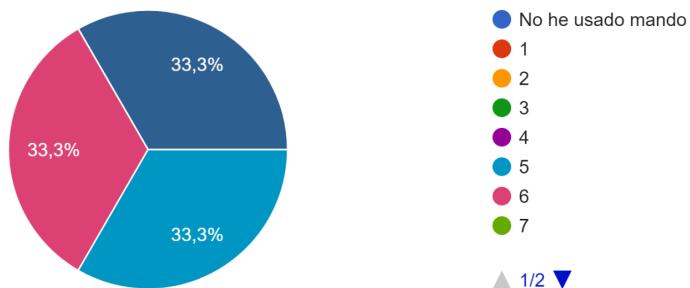
Si has contestado Si, ¿Qué mando has utilizado?

3 respuestas

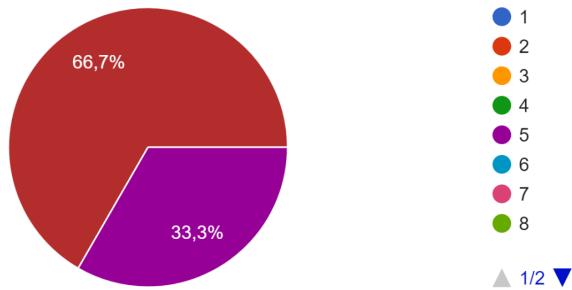


En esta versión, se ha retocado el tapping, ya nombrado anteriormente, para que sea más suave. Del 1 - 10, ¿Qué tan cómodo era usarlo?

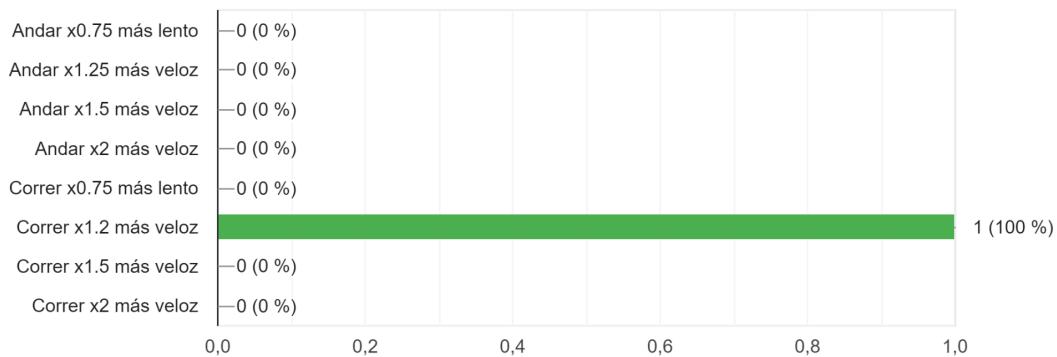
3 respuestas



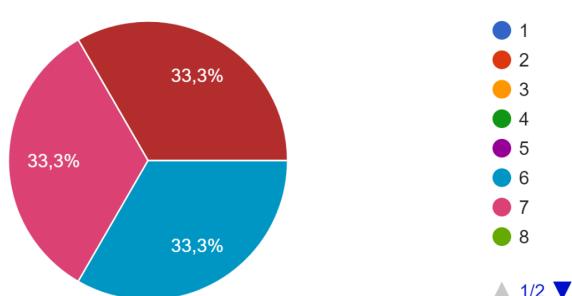
Se ha modificado suavemente la velocidad de andar y correr. Del 1 - 10, ¿Qué tan satisfecho estas con la velocidad? (Independientemente de si has usado teclado y no has podido probar el andar)
3 respuestas



Si la respuesta es inferior a 7, ¿Cómo de rápido te gustaría que fuera el personaje?
1 respuesta

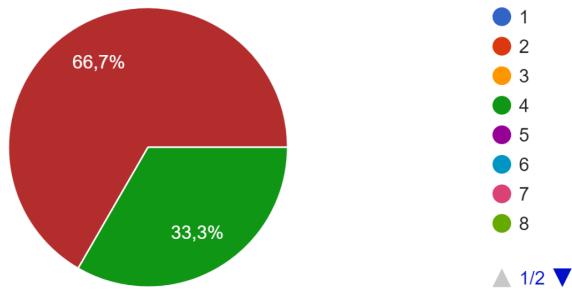


En esta versión, se ha añadido un dash inicial antes de empezar a correr. Del 1 - 10, ¿Qué tal se siente o/y funciona?
3 respuestas



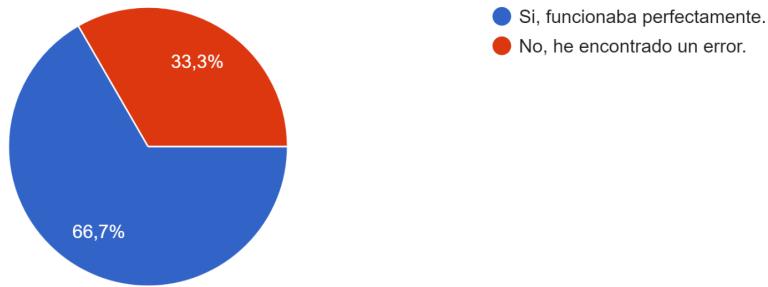
En esta versión, se ha añadido una frenada después de correr, la cual te permite pivotear y cambiar de dirección rápidamente. Del 1 - 10, ¿Qué tal se siente o/y funciona?

3 respuestas



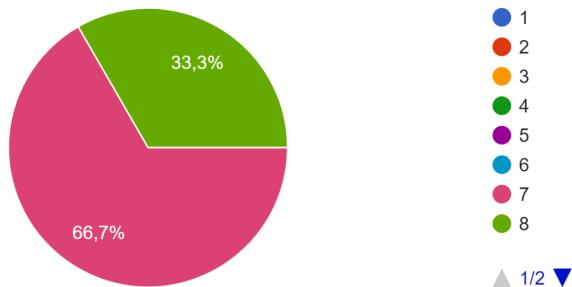
Anteriormente, no existía colisiones con la pared, pero en esta versión se ha implementado. ¿Lo has encontrado correcto su funcionamiento?

3 respuestas



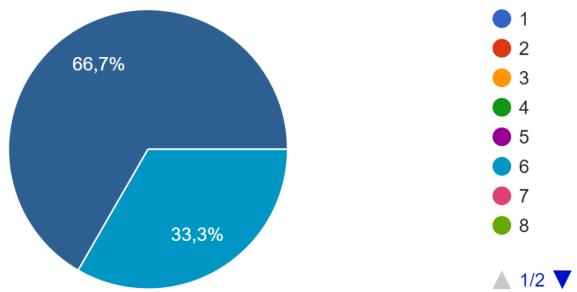
En esta versión, se ha añadido un short hop, referenciado a un salto bajo, y un full hop, referenciado a un salto alto o entero. Para cambiar entre estos... ha parecido este nuevo salto y su funcionamiento?

3 respuestas



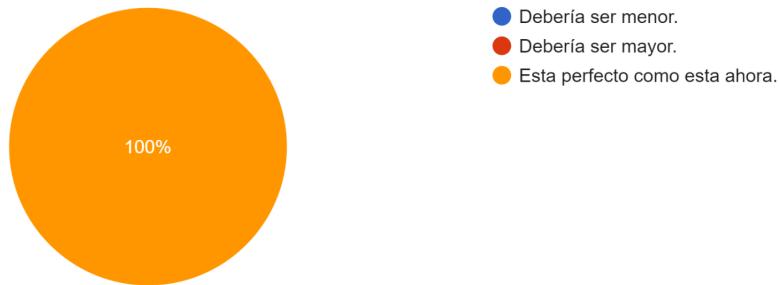
En esta versión, se ha añadido un feedback visual del doble salto. Del 1 al 10, ¿Qué tan entendible te ha parecido?

3 respuestas



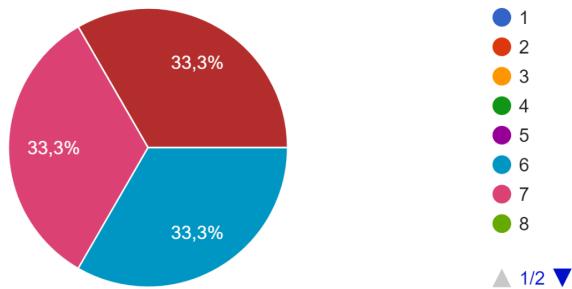
Actualmente el doble salto tiene la misma fuerza vertical que un full hop, ¿En tu opinión, crees que debería ser inferior la fuerza de salto del doble salt...onaje y por lo tanto el "modelo a seguir" del resto.*

3 respuestas



En esta versión se ha modificado gradualmente la gravedad del personaje para que alcance una velocidad máxima y no la supere. Del 1 al 10, ¿Qué tal te ha parecido la velocidad actual?

3 respuestas



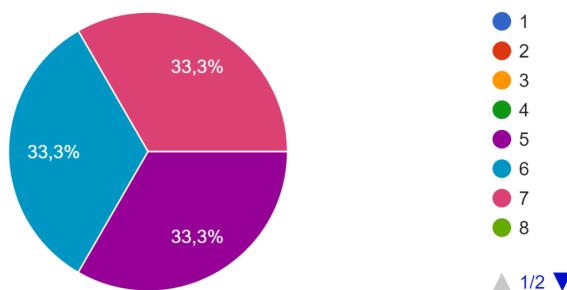
Si la respuesta es inferior a 7, ¿Qué tanto la disminuirías o aumentarías?

1 respuesta



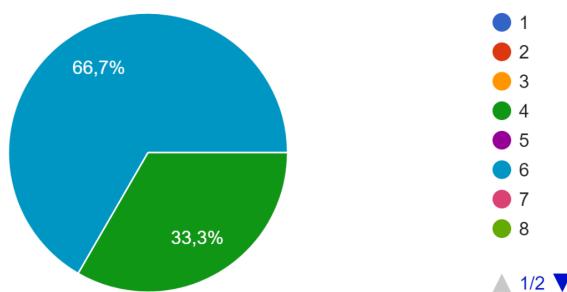
Anteriormente, el personaje se movía a la misma velocidad que cuando andaba mientras estuviera en el aire. En esta versión se ha ajustado para tener...salto. Del 1 al 10, ¿Qué tal se siente o/y funciona?

3 respuestas

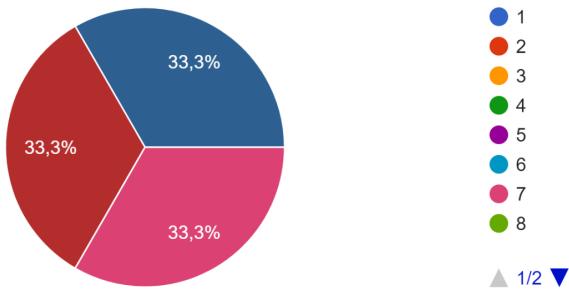


En esta versión, se ha añadido el fast fall o caída rápida, el cual se ejecuta con la palanca izquierda hacia abajo (con el teclado es la S) únicamente un...edad. Del 1 al 10, ¿Qué te han bien ha funcionado?

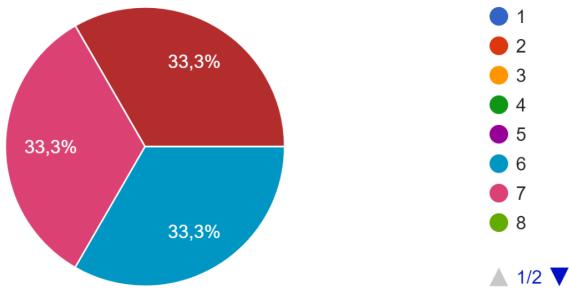
3 respuestas



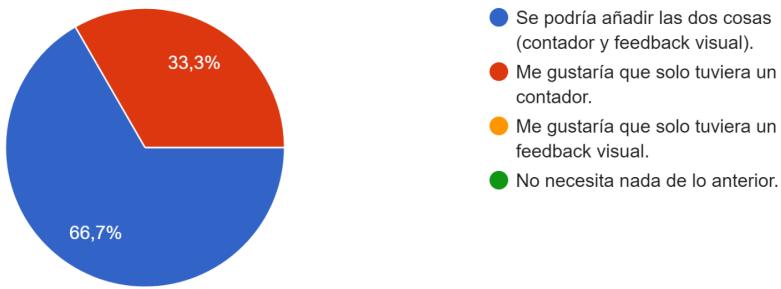
En la anterior versión, se añadió una caída de la plataforma, la cual en esta versión se ha mejorado considerablemente para que mientras estés en el aire...te. Del 1 al 10, ¿Qué tal se siente o/y funciona?
3 respuestas



Anteriormente la cámara era estática y no se movía del sitio, pero en esta versión se ha cambiado para que se mueva de manera dinámica entre las pos...a. Del 1 al 10, ¿Qué tal se siente o/y funciona?
3 respuestas



En futuro tambien se le añadirá que pueda golpear o incluso que pueda hacer algún golpe concreto (al estilo de una marioneta). ¿Debería de añadirle ...ejarlo que salga volando por cada golpe que reciba?
3 respuestas



En esta versión, se ha añadido un modo de un solo jugador llamado Break the targets (pendiente de nombre final), el cual el jugador debe de romper una...de golpear). Del 1 al 10, ¿Qué tal te ha parecido?
3 respuestas

