SPRINT 3 - MEMORIA TÉCNICA FINAL

CROSSOVER BATTLE

2026

Realizado por :

Diego Barrero Mejía
Jorge Gómez Navarro
Mirtha Gómez Guadalupe

Asignatura:

Programación Multimedia
y Dispositivos Móviles

Profesor:

Beatriz Ferro

DESCRIPCIÓN DEL JUEGO

Nombre del juego	***Crossover Battle***
Motor de desarrollo	Godot Engine 4.5 (GDScript)
Tipo de juego	Plataformas / Brawler 2D multijugador local
Público objetivo	Jugadores casuales, 2 jugadores en el mismo teclado
Nombre del grupo	Equipo 6 – Jorge Gómez, Diego Barrero, Mirtha Gómez
Curso	Programación Multimedia y Dispositivos Móviles
Fecha	24 de febrero 2026

Videojuego de acción y plataformas en 2D centrado en el combate para dos jugadores en el mismo ordenador. Nuestro equipo se ha inspirado en clásicos del género brawler como ***Duck Game***, siendo el objetivo principal la supervivencia y la eliminación del oponente en diferentes escenarios.

- **Concepto central:** El juego enfrenta a dos jugadores en una batalla donde la rapidez de reflejos y la adaptabilidad son clave. La experiencia central gira en torno al combate frenético entre dos jugadores: la victoria no depende de habilidades fijas del personaje, sino de la capacidad de adaptación al arma aleatoria que se encuentre en el mapa en cada momento.
- **Identidad visual:** Se ha optado por un estilo **Píxel Art** que no solo facilita el rendimiento técnico, sino que aporta una estética retro coherente. La temática "Crossover" permite la convivencia de personajes de distintos universos (como Mulan, Steve de Minecraft o los carismáticos patos), dotando al juego de una identidad visual variada y divertida.
- **Flujo de juego:** Las partidas se dividen en rondas rápidas, cortas e intensas. Los jugadores aparecen en puntos opuestos del mapa, recogen armas del suelo generadas automáticamente por un sistema de spawner, y compiten ronda a ronda hasta que uno alcanza la puntuación de victoria configurada antes de empezar.

MOTOR ELEGIDO: GODOT ENGINE 4.X

Para el desarrollo de este proyecto, se seleccionó ***Godot Engine*** tras un análisis comparativo con otras herramientas como ***Unity***. Los motivos técnicos que justifican esta decisión son:

- **Arquitectura basada en Nodos y Escenas:** Godot utiliza un sistema jerárquico donde "todo es un nodo". Esta estructura ha permitido modularizar el juego de forma eficiente, de esta manera, por ejemplo, los personajes, las armas y los elementos de la interfaz (UI) son escenas independientes que se instancian y reutilizan, facilitando enormemente el mantenimiento del código.
- **Lenguaje GDScript:** El uso de ***GDScript*** (lenguaje nativo de Godot con sintaxis similar a Python) ha sido fundamental para el éxito y desarrollo de este juego. Al ser un lenguaje ligero y de alto nivel, nos permitió programar mecánicas complejas como la Cámara Inteligente y la lógica de los proyectiles de forma rápida y con menos errores de memoria que lenguajes como C#.

- **Especialización en 2D:** A diferencia de otros motores donde el 2D es un "3D simplificado", Godot posee un motor de renderizado 2D dedicado. Esto nos ha permitido trabajar con coordenadas de píxeles reales, simplificando la gestión de colisiones, el diseño de tilemaps para los niveles y el sistema de partículas para los disparos.
- **Ligereza y Código Abierto:** Al ser un software de código abierto y extremadamente ligero, el flujo de trabajo entre los miembros del equipo fue muy ágil, permitiendo ejecuciones y pruebas instantáneas sin largos tiempos de carga, algo vital en un desarrollo de corta duración como este.

DISEÑO VISUAL Y NIVELES

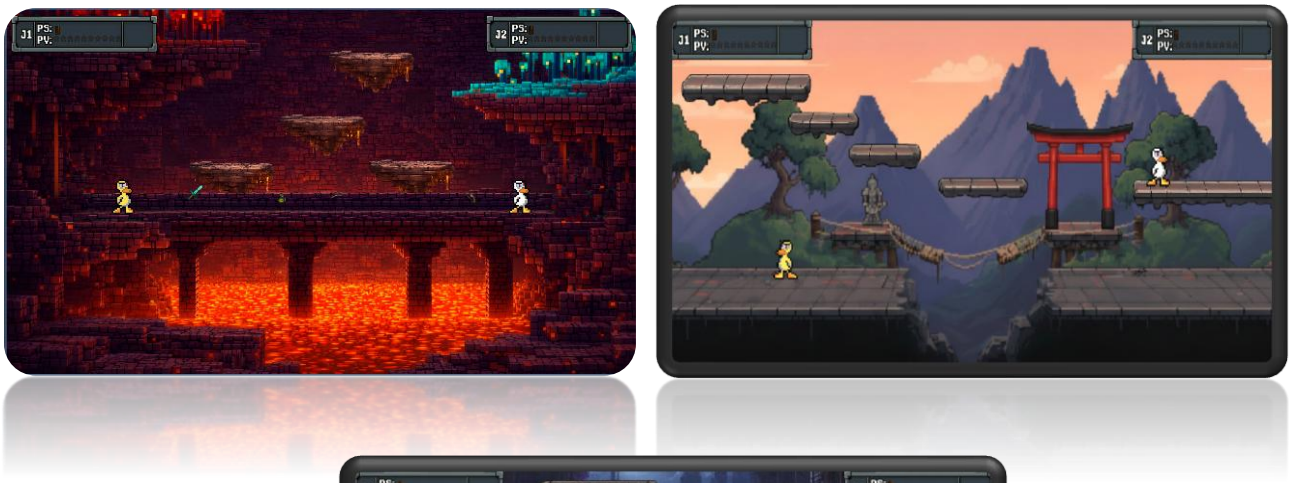
ESCENAS PRINCIPALES DEL JUEGO

Escena	Archivo.tscn	Descripción
Menú Principal	*scenes/ui/MenuInicio.tscn*	Pantalla de inicio con opciones ***Jugar***, ***Ajustes*** y ***Salir***.
Selección de personaje	*scenes/ui/CharacterSelect.tscn*	Pantalla donde cada jugador elige su personaje y se configuran las reglas de la partida (vidas y puntos para ganar).
Mapas de juego	*Scenes/mapas/Mapa#.tscn*	Mapas de combate con diseños y mecánicas distintas.
HUD	*scenes/ui/HUD.tscn*	Interfaz que aparece durante la partida: barras de vida, estrellas de puntuación y munición restante.
Cuenta atrás	*scenes/ui/Countdown.tscn*	Cuenta atrás de 3 segundos que aparece al inicio de cada ronda ante de que empiece el combate
Menú de pausa	*scenes/ui/MenuPausa.tscn*	Pausa el juego y da acceso a ajustes o vuelta al menú.
Pantalla de victoria	*scenes/ui/Victoria.tscn*	Aparece cuando alguien gana, mostrando el ganador, el marcador final y botones de Revancha o Volver al menú.

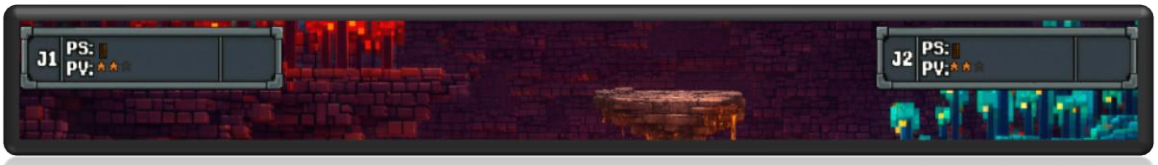
Ajustes	*scenes/ui/Ajustes.tscn*	Configuración de resolución, pantalla completa, volumen de musica y SFX, y reasignación de teclas.

Pantalla inicial – Menú Principal

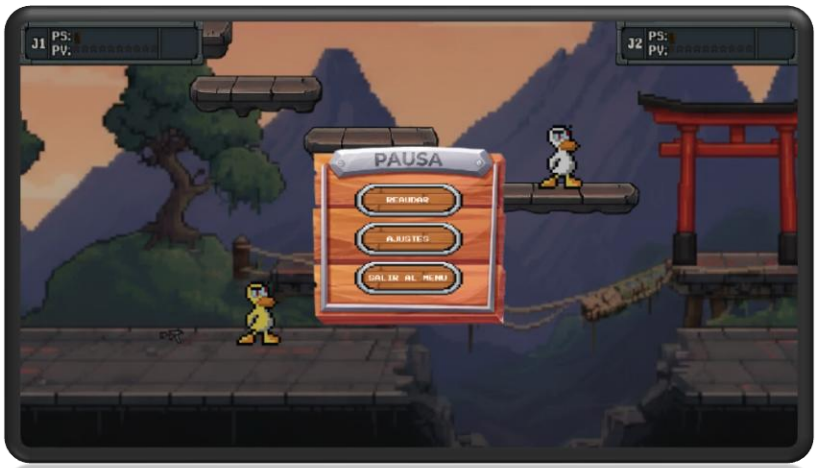
Pantalla Selección de Personajes



Mapas Aleatorios de Juego



HUD – Barras de vida, puntuación y munición



Menú de Pausa

Pantalla de Victoria

Pantalla de Ajustes

ESTILO VISUAL: COLORES, FONDOS, TEXTURAS Y LUCES

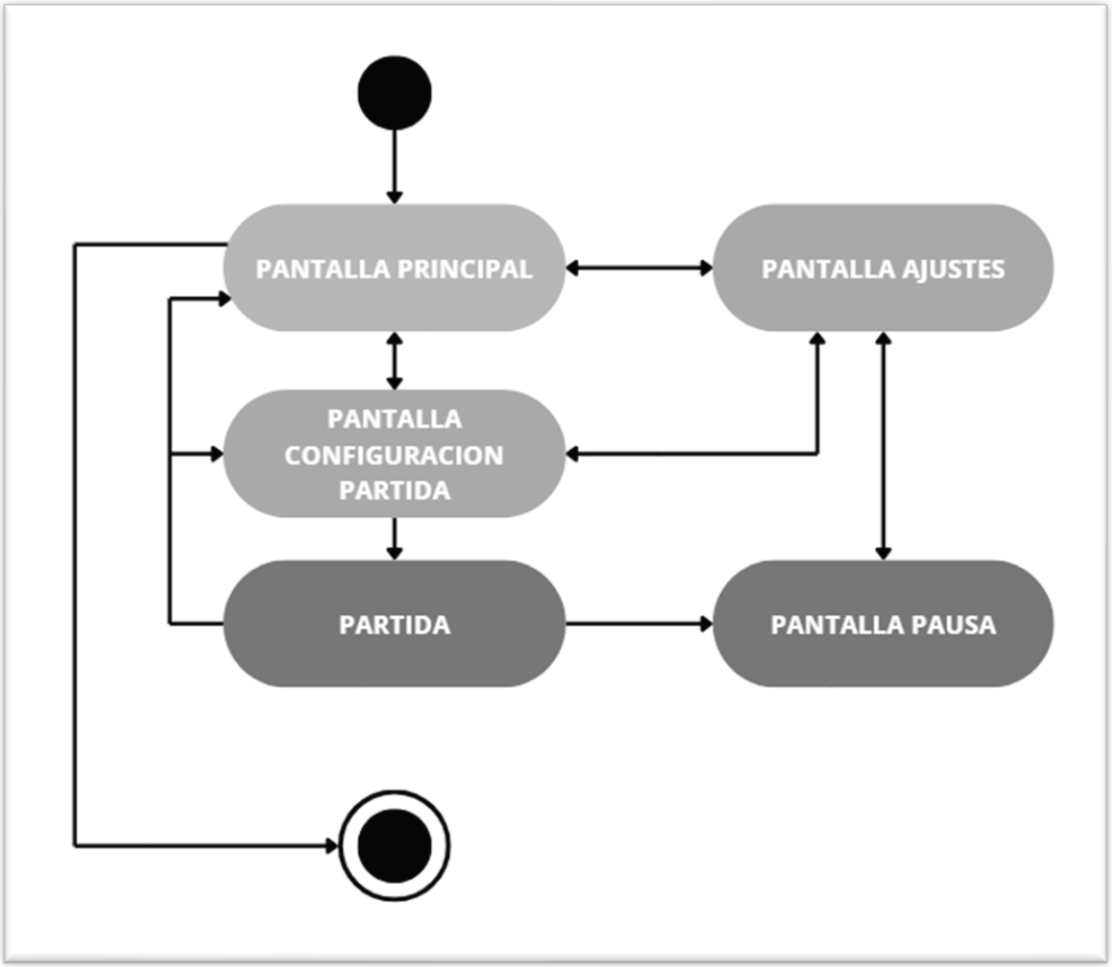
Desde el principio tuvimos claro que íbamos a usar ***Píxel Art*** como estilo visual. Esta decisión se tomó por dos razones: porque encaja perfectamente con el género brawler retro que nos inspiraba, y porque era más asequible para nosotros que intentar hacer sprites realistas. Además, el Píxel Art tiene ese punto de personalidad que hace que el juego tenga identidad propia.

La temática "Crossover" nos dio mucha libertad: al mezclar personajes de universos distintos (Mulan, Steve de Minecraft...) cada uno pudo diseñar sus personajes con su propio estilo sin que desentonara, porque la mezcla en si ya es parte del concepto del juego.

- ***Personajes***: sprites animados en píxel art de 32x32 a 64x64 px aproximadamente. Cada personaje tiene animaciones de reposo, caminar, saltar y atacar.
- ***Armas***: sprites independientes que se equipan al personaje y rotan según la dirección a la que se apunta. Se cuenta con mucha variedad de armas, algunas de ellas también inspiradas en el mundo de los personajes aportados por cada miembro.
- ***Mapas***: fondos y plataformas. Cada mapa ha sido creado también inspirado en el mundo de los personajes (como por ejemplo Minecraft inspirado en Steve o el Mapa Asiático inspirado en Mulán) incluyendo música y sonidos de daño específicos.
- ***HUD***: iconos sencillos de barras de vida y estrellas de puntuación para que sea fácil de leer de un vistazo.
- ***Resolución base***: 1280x720 px, con escalado automático para que se vea bien en otras resoluciones.

NAVEGACIÓN ENTRE PANTALLAS

El flujo de navegación del juego sigue siempre el mismo camino:



Desde cualquier punto del juego se puede acceder a Ajustes y volver exactamente al punto donde se estaba. Todo esto se ha gestionado con la función ***change_scene_to_file()*** de Godot desde los scripts de cada pantalla.

OBJETOS Y MECÁNICAS

OBJETOS PRINCIPALES

PERSONAJES JUGABLES

El juego cuenta con 14 personajes jugables. Están definidos en el script ***Global.gd*** como una lista con el nombre, la imagen de selección y la ruta de la escena.

Anadir un personaje nuevo es tan fácil como agregar una entrada a esa lista y crear la escena correspondiente:

Personaje	Archivo.tscn	Tipo
Alex (Minecraft)	*Alex.tscn*	Crossover
Steve (Minecraft)	*steve.tscn*	Crossover
Mulan	*mulan.tscn*	Crossover
Magic	*magic.tscn*	Crossover
Diego	*Diego.tscn*	Crossover
Eve	*Eve.tscn*	Crossover
John	*John.tscn*	Crossover
Jorge	*Jorge.tscn*	Crossover
Kratos	*Kratos.tscn*	Crossover
Sanyu	*Sanyu.tscn*	Crossover
Shan	*Shan.tscn*	Crossover

ARSENAL DE ARMAS

El juego tiene 12 armas diferentes divididas en dos categorias: armas de fuego y armas cuerpo a cuerpo. Todas heredan de una clase base llamada ***ArmaBase.gd***, lo que nos permitió reutilizar mucho código. Incluimos aquí también las balas de las armas de fuego y la bola de fuego.

Categoría	Armas	Script / Escena	Datos clave
Pistolas	Luger, M92, Revolver	*scenes/armas/pistolas/*	Daño medio, buena cadencia de disparo
Rifles	AK47, M15	*scenes/armas/rifles/*	Alta cadencia, daño medio
Escopetas	M24 (SawedOff)	*scenes/armas/escopetas/*	Mucho daño, pero dispara lento
Snipers	M24 Sniper	*scenes/armas/snipers/*	Daño máximo, ideal para larga distancia
Subfusiles	MP5	*scenes/armas/subfusiles/*	Cadencia muy alta, poco daño por bala
Arco	Arco + Flecha (Steve)	*scenes/armas/arco_steve/*	Munición infinita, daño 2, buen ritmo de disparo
Granada	Granada	*scenes/armas/granada/*	Daño de área, genera explosión con partículas
Espadas (Melee)	Espada Steve, Espada Mulan, Daga Sanyu	*scenes/armas/espada_steve, espada_mulan, daga_sanyu*	Cuerpo animado con AnimationPlayer a cuerpo, golpe
Hacha Kratos	Hacha	*scenes/armas/hacha_kratos/*	Cuerpo animado con AnimationPlayer a cuerpo, golpe
Escudo Kratos	Escudo Kratos	*scenes/armas/escudo-kratos /*	Cuerpo animado con AnimationPlayer a cuerpo, golpe
Bola Fuego	Bola de fuego	*scenes/armas/BolaDeFuego, BolaDeFuegoProyectil /*	Daño máximo, bola proyectada con animación

Balas	Balas armas fuego	*scenes/Bala, Bala_escopeta, Bala_fusil /*	Daño máximo, balas con animación para cada arma de fuego

MAPAS

El juego cuenta con 7 mapas específicos que heredan de la clase ***MapaController.gd***, lo que nos permitió meter mapas con distintas temáticas, reutilizando código y así poder dar versatilidad y personalidad al juego para garantizar la máxima experiencia al jugador.

Mapa	Archivo.tscn	Tipo
Mapa3	*Mapa3.tscn*	Inspirado en temática Minecraft
Mapa4	*Mapa4.tscn*	Inspirado en temática asiática (Mulan)
Mapa5	*Mapa5.tscn*	Inspirado en temática urbana
Mapa6	*Mapa6.tscn*	Inspirado en temática God of War (Kratos)
Mapa7	*Mapa7.tscn*	Inspirado en temática God of War (Kratos)
Mapa8	*Mapa8.tscn*	Temática neutral
Mapa9	*Mapa9.tscn*	Temática neutral

MECANICAS PRINCIPALES

SISTEMA DE VIDA Y DAÑO

Cada personaje tiene una variable de vida que se descuenta cuando recibe daño. Cuando llega a cero, el personaje muere y el rival se lleva un punto. La vida maxima se puede configurar entre 1 y 10 antes de empezar la partida desde la pantalla de selección de personajes.

Cuando un personaje muere, el script ***PersonajeBase.gd*** envía una señal al ***MapaController.gd***, que es el encargado de sumar el punto, comprobar si alguien ha ganado ya la partida y decidir si hay que iniciar una nueva ronda o mostrar la pantalla de victoria

SISTEMA DE ARMAS Y RECOGIDA

Las armas aparecen en el mapa de forma automática cada cierto tiempo gracias al ***SpawnerArmas.gd***. Cuando un personaje se acerca a un arma, la recoge automáticamente y se la equipa. Si ya llevaba otra arma, la suelta en el suelo para que el rival pueda cogerla. Esto genera situaciones tácticas interesantes.

Las armas melee (espadas y daga) funcionan de forma diferente a las de fuego: en lugar de disparar proyectiles, usan el ***AnimationPlayer*** para activar y desactivar la hitbox del arma en el momento exacto del golpe, lo que hace que el timing sea importante.

MUERTE Y RESPAWN (NUEVA RONDA)

Cuando termina una ronda (alguien muere), aparece una cuenta atrás de 3 segundos durante la cual los personajes no se pueden mover. Pasada la cuenta, los dos jugadores reaparecen en sus puntos de spawn y comienza la siguiente ronda. Si alguien alcanza la puntuación configurada, en lugar de nueva ronda aparece la pantalla de victoria.

CLONACION (SELECCIÓN DE PERSONAJE)

El sistema de selección instancia el personaje elegido en tiempo de ejecución usando load ***(ruta_escena). instantiate ()***. El mismo personaje puede ser elegido por ambos jugadores de manera simultánea, lo que técnicamente es una ***'clonación'***: es decir son dos instancias del mismo ***.tscn*** con ***player_id*** diferente para dar al juego total versatilidad y originalidad.

MOVIMIENTO Y FÍSICAS

Toda la lógica de movimiento de los personajes esta centralizada en ***PersonajeBase.gd***, que extiende de un ***CharacterBody2D***. De este modo, todos los personajes heredan de esta clase:

Mecánica	Implementación técnica
Correr	El personaje se mueve a 300 pixeles por segundo en horizontal. El sprite gira automáticamente según la dirección.
Saltar	Solo se puede saltar si el personaje está en el suelo (***is_on_floor()***). La velocidad de salto es -500 px/s
Caer / gravedad	Se aplica cada frame físico sumando a la velocidad vertical, igual que funciona en la realidad.

Camara inteligente	***CamaraInteligente.gd*** calcula el punto medio entre los dos jugadores y ajusta el zoom automáticamente para que siempre se vean los dos en pantalla.
Zona de muerte	Si un personaje sale de los límites del mapa, se aplica daño letal directo.

CONTROLES POR TECLADO

Acción	Jugador 1	Jugador 2
Mover izquierda	*A*	*Flecha izquierda*
Mover derecha	*D*	*Flecha derecha*
Mirar abajo	*S*	*Flecha Abajo*
Saltar	*Shift*	*K*
Disparar / Atacar	*V*	*Ñ*
Acción	*C*	*L*
Pausa	*ESC*	*ESC*

MULTIMEDIA Y MENÚS

MÚSICA Y EFECTOS DE SONIDO

El audio del juego lo gestionan dos scripts que están siempre ***Autoloads***: ***MusicManager.gd*** para musica de fondo y ***SFXManager.gd*** para los efectos de sonido Separar las dos cosas nos permitió controlar el volumen de cada uno por separado desde el menú de ajustes.

MÚSICA DE FONDO – MUSICMANAGER

Hay 6 soundtracks diferentes que suenan en los menús. El jugador puede cambiar de pista desde ***Ajustes*** según su gusto. Durante el combate, la musica cambia dependiendo del mapa. Cuando el juego está en pausa, la musica del mapa se pausa también, pero la musica del menú sigue sonando.

EFECTOS DE SONIDO (SFXMANAGER)

- ***Botones UI:*** sonido al pasar el ratón por encima y al hacer click en cualquier botón. Se conectan solos a todos los botones de la escena activa.
- ***Disparos:*** cada arma tiene su propio sonido de disparo.
- ***Pasos:*** el personaje alterna entre 4 sonidos de pasos distintos mientras camina, con un intervalo de 0.28 segundos entre cada uno para que suene natural.
- ***Ganar punto:*** suena un efecto cuando alguien se lleva una ronda.

MENÚS: INICIO, FIN Y PAUSA

- ***Menú de inicio:*** lo primero que ves al abrir el juego. Tiene los botones de Jugar, Ajustes y Salir. La musica empieza aquí.
- ***Selección de personajes:*** cada jugador puede navegar por el roster de personajes con sus botones y ver una vista previa. También se configuran aquí las vidas y los puntos necesarios para ganar.
- ***Menú de pausa:*** se abre con ESC durante la partida. Permite reanudar, ir a ajustes o volver al menú principal.
- ***Pantalla de victoria:*** aparece como una capa encima del mapa para que se vea el estado final de la partida detrás. Muestra quien ha ganado y el marcador, con opciones de revancha o volver al menú.
- ***Ajustes:*** se puede cambiar la resolución (1280x720, 1600x900, 1920x1080), activar o desactivar pantalla completa, ajustar el volumen de la musica y los efectos por separado, y reasignar cualquier tecla. Todo se guarda en un archivo de configuración para que no haya que volver a configurarlo cada vez.

CONTRIBUCIÓN A LA EXPERIENCIA DEL JUGADOR

La combinación de musica adaptativa por cada menú, los efectos de sonido procedentes de pasos, disparos y explosiones, sumado a la música que continua en pausa crea al jugador una experiencia auditiva que se ajusta a la temática escogida. El sistema de volumen en ***Ajustes*** permite que cada jugador personalice la experiencia sin perder los ajustes entre sesiones. De esta manera el método de juego puede adaptarse a cualquier tipo de jugador y de momento de juego.

EXPLICACIÓN TÉCNICA Y DOCUMENTACIÓN

COMO FUNCIONA EL JUEGO POR DENTRO

Por dentro el juego tiene tres capas bien diferenciadas que se comunican entre si:

- ***Los Autoloads (datos globales):*** son scripts que siempre están activos independientemente de la pantalla en la que esté el jugador. ***Global.gd*** guarda los personajes elegidos, las puntuaciones y la configuración de la partida. ***ConfigManager*** guarda los ajustes en disco.
- ***MapaController (la partida):*** es el director de cada partida. Se encarga de hacer aparecer a los personajes, escuchar cuando alguien muere, sumar puntos y decidir si hay nueva ronda o fin de partida.

- ***PersonajeBase (cada jugador):*** cada personaje gestiona el suyo propio (movimiento, animación, vida y arma equipada). Cuando pasa algo importante (como morir), no llama directamente a ***MapaController,*** sino que emite una señal, lo que mantiene el código más limpio y desacoplado.

SCRIPT CLAVE: PERSONAJEBASE.GD

Este es el script más importante del proyecto y del que más orgullosos estamos. Antes de tenerlo, cada personaje tenía su propio código de movimiento y había muchas cosas duplicadas. Cuando refactorizamos todo en una clase base, añadir personajes nuevos se volvió un proceso mucho más rápido.

Lo que hace ***PersonajeBase.gd*** es gestionar todo lo que tienen en común todos los personajes:

- ***Movimiento y salto:*** lee las teclas configuradas para cada jugador y mueve el personaje en consecuencia.
- ***Gravedad:*** se aplica cada frame para que el personaje caiga de forma natural.
- ***Vida:*** cuando cambia, avisa al ***HUD*** para que lo actualice. Cuando llega a cero, lanza la animación de muerte y avisa al ***MapaController***.
- ***Animaciones:*** elige automáticamente la animación correcta según lo que esté haciendo el personaje (quieto, caminando, saltando).
- ***Sonidos de pasos:*** alterna entre 4 audios distintos con un temporizador para que no suene repetitivo.
- ***Arma equipada:*** gestiona coger y soltar armas, y las coloca en la posición correcta en la mano del personaje**.**

HERRAMIENTAS Y RECURSOS USADOS

- ***Godot Engine 4.5 (GL Compatibility):*** motor principal, sin dependencias externas.
- ***GDScript (nativo de Godot):*** lenguaje de programación nativo de Godot.
- ***Assets gráficos:*** sprites en Píxel Art, parte de creación propia y parte de recursos libres adaptados.
- ***Assets de audio:*** efectos OGG libres de derechos y soundtracks originales del proyecto.
- ***ConfigFile (nativo Godot):*** para guardar los ajustes del jugador en disco entre sesiones.

TRABAJO EN EQUIPO Y METODOLOGÍA

ROLES Y ORGANIZACIÓN DEL GRUPO

Al ser tres personas, desde el principio intentamos repartir el trabajo según las habilidades y gustos de cada uno para que cada uno pudiera centrarse en lo que mejor se le daba:

Miembro	Rol principal	Tareas destacadas
Jorge	*Lead Artist & Base del Juego*	Desarrollo de la base jugable del proyecto (movimiento, física, estructura de escenas). Diseño de más personajes en Píxel Art, programación especial del personaje Alex (sistema de capas de sprite complejo) y estética general de la interfaz de usuario (UI). Desarrollo y temática Kratos y mapas neutrales (musica y SFX específicos).
Diego	*Programmer & Level Designer*	Implementación y refinamiento del código de mecánicas. Diseño de más personajes en Píxel Art y programación especial del personaje Steve (sistema de capas de sprite complejo) . Desarrollo y temática Minecraft (música y SFX específicos).
Mirtha	*Character Designer & Documentación*	Diseño y programación de los personajes Mulan, Shan y Sanyu. Diseño de las armas cuerpo a cuerpo Espada Mulan y Daga Sanyu (animaciones de ataque con AnimationPlayer). Diseño y construcción de los mapas Mapa4/Mulán y Mapa5/Urbano. Documentación.

METODOLOGÍA: SCRUM ADAPTADO

El proyecto se ha desarrollado siguiendo una metodología Scrum simplificada adaptada al contexto escolar, con sprints semanales:

Sprint	Objetivos	Resultado
Sprint 1	Prototipo jugable básico	Un personaje que se movía y saltaba, un mapa simple y un arma que disparaba. Suficiente para ver que el concepto funcionaba.
Sprint 2	Sistema de armas, personajes y mapas	Arsenal completo, todos los personajes del roster, tres mapas jugables y la pantalla de selección de personajes.
Sprint 3 (Final)	Audio, ajustes, pulido y documentación	Sistema de audio completo, ajustes persistentes, pantalla de victoria, cámara inteligente y todos los bugs importantes corregidos.

LO QUE NOS HA FUNCIONADO Y LO QUE CAMBIARIAMOS

LO QUE HA FUNCIONADO BIEN

- ***Trabajar con ramas de Git:*** cada uno trabajaba en su rama y luego hacíamos merge. Al principio nos costaba resolver los conflictos, pero al final del proyecto ya lo hacíamos de forma natural.
- ***La clase base PersonajeBase:*** fue una decisión que tomamos en el Sprint 2 después de ver que el código de cada personaje era casi idéntico. Refactorizar todo en una clase base nos ahorró muchísimo tiempo en los personajes siguientes.
- ***Los Autoloads para el estado global:*** tener ***Global.gd*** siempre disponible sin tener que pasarlo como parámetro de escena en escena, simplificó mucho las tareas.
- ***Las señales de Godot:*** usarlas para comunicar el personaje con el ***MapaController*** hizo que el código fuera mucho más fácil de depurar, porque cada parte del juego hace solo lo suyo.

LO QUE CAMBIARÍAMOS

- ***Empezar con Git desde el primer día:*** al principio nos pasábamos el proyecto por Teams o en clase, por lo que empezamos a usar Git correctamente, para evitar la sobreescritura.
- ***Planificar mejor las collision layers desde el principio:*** a mitad del Sprint 2 tuvimos que reorganizar todas las capas de colisión porque las balas atravesaban a los personajes o las armas no se podían recoger. Si lo hubiéramos pensado antes nos habría ahorrado varios días.

EXPOSICIÓN FINAL Y CIERRE

RESUMEN DEL RESULTADO FINAL

El resultado final es un juego completamente jugable, con todas las mecánicas principales funcionando y pulido suficiente para ser presentado. En resumen, ***Crossover Battle*** incluye:

- 11 personajes jugables con sprites y animaciones en Píxel Art.
- 12 armas distintas (pistolas, rifles, escopetas, snipers, subfusil, arco, granada y 3 armas melee, bola de fuego, escudo de Kratos, hacha de Kratos).
- 7 mapas de combate con diseños variados.
- Sistema de vida y puntuación completamente configurable antes de la partida (1-10 vidas, 1-20 puntos para ganar).
- Sistema de audio completo con 6 soundtracks, musica adaptativa por mapa y efectos de sonido para todas las acciones.
- Sistema de ajustes persistente: resolución, pantalla completa, volumen y reasignación de teclas.
- Flujo de juego completo: Menú -> Selección -> Partida -> Victoria -> Revancha.
- Camara inteligente dinámica que mantiene a ambos jugadores en pantalla en todo momento.

REFLEXIÓN FINAL: QUE APRENDIMOS Y QUE MEJORARÍAMOS

APRENDIZAJES TÉCNICOS

Este proyecto ha sido el más complejo que hemos hecho hasta ahora y hemos aprendido mucho, no solo de programación sino de cómo se organiza un proyecto real:

- ***Godot y GDScript:*** empezamos sin saber casi nada del motor y al final del proyecto nos manejábamos con soltura. El sistema de nodos, las señales y los Autoloads son conceptos que ahora entendemos de verdad porque los hemos usado en un proyecto real.
- ***La importancia de la arquitectura:*** al principio íbamos un poco a lo loco escribiendo código y luego teníamos que reescribirlo todo. Aprendimos que pensar un poco la estructura antes de ponerse a picar ahorra muchísimo tiempo.
- ***Control de versiones:*** Git paso de ser algo que usábamos por recomendación y sin entender mucho, a algo que agradecemos tener. Hubo momentos críticos donde poder volver a una versión anterior nos salvó de perder horas de trabajo.
- ***Depuración:*** tuvimos problemas con las colisiones, con la cámara, con las animaciones y con merges de Git. Cada problema que resolvimos nos enseñó algo nuevo.

PROBLEMAS REALES QUE TUVIMOS Y COMO LOS RESOLVIMOS

- ***Conflictos de Git:*** especialmente al principio, cuando hacíamos merge de ramas que tocaban los mismos archivos. Lo solucionamos estableciendo zonas de trabajo claras: cada uno tocaba sus escenas y sus scripts, y avisábamos antes de modificar algo compartido como ***Global.gd***.
- ***Bugs con las animaciones:*** los personajes a veces se quedaban congelados en la animación de salto o no volvían al estado de reposo correctamente. El problema estaba en que los estados del ***AnimationPlayer*** no se reseteaban bien. Lo arreglamos revisando las condiciones de transición entre animaciones en ***PersonajeBase***.
- ***Colisiones de armas:*** durante el Sprint 2 las balas no detectaban a los personajes o las armas del suelo no se podían recoger. El problema era que teníamos las Collision Layers y Masks mal configuradas. Tuvimos que reorganizarlo todo: Layer 1 para personajes, Layer 2 para armas, Layer 3 para el escenario, y configurar cada escena correctamente.
- ***La cámara se salía de pantalla:*** al principio usábamos una cámara fija y cuando los jugadores se alejaban uno de otro se salía del encuadre. La solución fue crear ***CamaraInteligente.gd***, que calcula el punto medio entre los dos jugadores y ajusta el zoom con ***lerp()*** para que siempre quepan los dos.

QUE AÑADIRÍAMOS CON MÁS TIEMPO

- Un modo para jugar solo contra una IA, aunque sea básica.
- Un selector de mapa en la pantalla de selección de personajes.
- Mas efectos visuales en los impactos y en la muerte de los personajes.
- Modo online para poder jugar sin estar en el mismo ordenador.