Instituto Puig Castellar

Ciclo formativo: Desarrollo de Aplicaciones Multiplataforma · Grado: CFGS

Proyecto intermodular · Curso 2025-2026

HUNTER SURVIVORS

Videojuego 2D tipo survivor con backend online, tabla de clasificación global y página web de presentación

// GODOT 4.6 · GDSCRIPT · FIREBASE

Autor: Ignacy Faldmo

Tutor: Luis Elía Talón

Fecha de entrega: 17/05/2026

Resumen del proyecto

Hunter Survivors es un videojuego 2D de acción tipo survivor desarrollado con Godot 4.6 y GDScript como proyecto final de ciclo. El jugador controla un piloto que debe sobrevivir a oleadas de enemigos mientras sus armas disparan automáticamente al objetivo más cercano. La puntuación equivale a las monedas acumuladas: cuanto más tiempo sobrevives, mayor es la puntuación.

El proyecto va mucho más allá de un juego local: integra autenticación real por email mediante Firebase Auth, guardado de progreso en la nube con Firestore, una tabla de clasificación global en tiempo real, estadísticas agregadas de todos los jugadores, y una página web de presentación pública con datos en vivo. El juego está disponible para descarga en Windows y Linux como GitHub Release, y la web está desplegada en GitHub Pages.

Se implementaron seis tipos de enemigos con comportamientos diferenciados, siete armas activas y dos pasivas con stacking simultáneo de hasta tres armas, un sistema de dificultad progresiva, setenta obstáculos físicos generados proceduralmente en el mapa, un HUD completamente dibujado con código personalizado, y una arquitectura de sistemas desacoplados que evita el código monolítico.

Palabras clave

Godot 4 · GDScript · Firebase · Firestore · Videojuego survivor · Leaderboard · GitHub Pages · Backend online · Sistemas desacoplados · Despliegue continuo

Abstract

Hunter Survivors is a 2D action survivor game developed with Godot 4.6 and GDScript as a final cycle project. The player controls a pilot who must survive escalating enemy waves while weapons fire automatically at the nearest target. Score equals accumulated coins — the longer you survive, the higher the score.

The project goes well beyond a local game: it integrates real email authentication via Firebase Auth, cloud progress saving with Firestore, a live global leaderboard, aggregated player statistics, and a public presentation website with real-time data. The game is available for download on Windows and Linux as a GitHub Release, and the website is deployed on GitHub Pages.

Keywords

Godot 4 · GDScript · Firebase · Firestore · Survivor game · Leaderboard · GitHub Pages · Online backend · Decoupled systems · Continuous deployment

Índice

1. Presentación del proyecto

1.1 Introducción

Hunter Survivors es un videojuego 2D de acción tipo survivor desarrollado íntegramente en Godot 4.6 con GDScript como proyecto final del ciclo formativo de Desarrollo de Aplicaciones Multiplataforma. La mecánica central está inspirada en Vampire Survivors (Poncle, 2022): el jugador se mueve por un mapa mientras sus armas disparan automáticamente al enemigo más cercano. El objetivo es sobrevivir el máximo tiempo posible mientras las oleadas de enemigos se hacen progresivamente más difíciles.

Lo que distingue este proyecto de un juego local convencional es la capa de infraestructura online construida alrededor del juego. El progreso del jugador persiste en la nube tras cada partida, existe una tabla de clasificación global en tiempo real visible tanto en el juego como en una página web pública, y los jugadores pueden registrarse y acceder a sus estadísticas desde cualquier dispositivo. El código fuente está publicado en GitHub con versiones descargables para Windows y Linux.

El proyecto fue desarrollado en solitario durante el segundo trimestre del curso 2025-2026, con un total estimado de más de 200 horas de trabajo efectivo.

A lo largo de esta memoria se describe el proceso completo de desarrollo del proyecto. En primer lugar se presenta el contexto y la justificación de la idea. A continuación se detalla la estrategia, la metodología y la planificación seguidas. Los apartados de análisis y diseño recogen los requisitos del sistema, los casos de uso y las decisiones técnicas tomadas. El apartado de desarrollo documenta la implementación de las funcionalidades principales y las dificultades encontradas junto con sus soluciones. Finalmente se presentan las conclusiones, los recursos utilizados y los anexos con información complementaria.

1.2 Contexto

Los juegos tipo survivor han experimentado un crecimiento muy rápido en el mercado independiente. **Vampire Survivors** fue el primer título que demostró el enorme interés del público por el género, con más de 3 millones de copias vendidas en sus primeros meses. A raíz de su éxito llegó una oleada de derivados: **20 Minutes Till Dawn**, **Brotato**, **Halls of Torment**, **Soul Knight Prequel**, entre otros muchos.

Al investigar el mercado para este proyecto, se observó que muy raramente estos juegos tienen mecánicas online donde el jugador pueda ver su posición en un ranking global, comparar estadísticas con otros jugadores o acceder a su progreso desde diferentes dispositivos. La ausencia de esta capa social representa una oportunidad: añadir persistencia online y una clasificación global puede aumentar significativamente la retención y la sensación de comunidad sin complicar las mecánicas base del género.

1.3 Justificación

La elección de un videojuego como proyecto final no fue casual. Los juegos tipo survivor han estado creciendo muy rápido en el mercado; el primer juego que demostró un gran interés por el género. Primero fue **Vampire Survivors**, seguido de esto vino la oleada de clones, reskins y diferentes visiones de cómo puede funcionar el género.

Al adentrarse en la investigación de este proyecto, se ha visto que muy raras veces dichos juegos tienen mecànicas online, donde el jugador puede comparar estadísticas, craftear builds y competir con otros jugadores en un ranking.
El género survivor también es técnicamente complicado: gestionar diversas entidades simultáneas, un sistema de armas concurrentes, escalado de dificultad en tiempo real y una interfaz reactiva. Todo en el game loop sin bloquearlo, obliga a aprender sobre cómo gestionar diferentes mecánicas sobre un juego que normalmente en el curso no se ha tocado de manera amplia

1.4 Objetivos

Objetivo general: Desarrollar un videojuego 2D tipo survivor con backend en la nube, tabla de clasificación global y página web de presentación pública.

Objetivos específicos:

1. Implementar el loop de juego completo: movimiento del jugador, seis tipos de enemigos, auto-disparo, spawning escalado y dificultad progresiva.
2. Desarrollar un sistema de armas con stacking de hasta 3 armas activas simultáneas, cada una con su propio cooldown, más 2 armas pasivas (aura y órbita).
3. Integrar Firebase Auth para autenticación real por email y contraseña, tanto en el juego como en la web.
4. Persistir el progreso del jugador en Firestore y sincronizarlo entre sesiones de juego.
5. Implementar una tabla de clasificación global con actualización automática al superar el récord personal.
6. Publicar una página web de presentación con datos en tiempo real desde Firebase.
7. Exportar el juego para Windows y Linux y publicarlo como GitHub Release descargable.
8. Aplicar una arquitectura de sistemas desacoplados que permita mantener y ampliar el código sin romper el juego.

Todos estos objetivos responden a una misma idea central: demostrar que un proyecto individual de fin de ciclo puede ir más allá de un prototipo local e integrar servicios reales en producción. El resultado no es solo un juego funcional, sino un sistema completo con usuarios reales, datos reales en la nube y una web pública operativa.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

El proyecto se planteó como un **MVP iterativo por fases**: primero el loop de juego básico funcional, después la progresión, luego el sistema de armas avanzado, la integración con Firebase, la UI y el polish, y finalmente la web y el despliegue. Esta estrategia garantiza tener siempre una versión jugable aunque incompleta, lo que facilitó la detección temprana de problemas de rendimiento y de diseño.

La viabilidad técnica fue evaluada antes de empezar. Los criterios clave fueron: que existiera un addon de Firebase para Godot 4 maduro (existía: godot-firebase v2.6), que el plan gratuito de Firebase fuera suficiente para los volúmenes esperados (sí, el plan Spark incluye 50.000 lecturas y 20.000 escrituras diarias), y que GitHub Pages pudiera servir el frontend sin coste adicional.

2.2 Metodología de trabajo

Se utilizó una metodología **iterativa incremental** sin sprints formales ni tablero Kanban. El trabajo se organizó mentalmente en fases funcionales: cada fase tenía un entregable jugable o demostrable antes de pasar a la siguiente. Las decisiones de diseño se tomaban sobre código que ya funcionaba, no sobre especificaciones abstractas escritas previamente.

La búsqueda de información fue constante a lo largo del desarrollo y se describe en detalle en la sección 5 (Desarrollo), dentro de cada problema específico donde fue necesaria.

Para la gestión del código se utilizó **Git con GitHub**. El flujo de trabajo inicial fue de branches de feature (una por sistema o funcionalidad), pero esto generó conflictos de merge graves que se documentan en la sección 5.3. A partir del incidente, el desarrollo se realizó principalmente en la rama main (la principal) con commits frecuentes.

2.3 Planificación por fases

El proyecto se organiza en seis fases que combinan el desarrollo del videojuego con la integración del backend en la nube y su visualización web.

Fase 1 · Configuración inicial y estructura del proyecto Se inicializa el repositorio Git, se configura la estructura de carpetas del proyecto en Godot y se integra el addon godot-firebase. Se crea la escena del jugador con movimiento básico y se establece la conexión inicial con Firebase. El resultado de esta fase es un entorno de trabajo operativo con el jugador moviéndose en pantalla y la base de Firebase configurada.

Fase 2 · Core gameplay: combate y enemigos Se implementa el sistema de disparo automático, los primeros tipos de enemigos con persecución básica, las colisiones y el spawning periódico. Se integran los controles WASD y se establecen las mecánicas de base del género survivor. El resultado es una versión mínima jugable.

Fase 3 · Sistema de armas, progresión y escalado Se desarrolla el sistema de armas con stacking de hasta tres simultáneas, cada una con cooldown independiente. Se añaden siete armas activas y dos pasivas. Se implementa el sistema de XP, niveles y dificultad progresiva con escalado de velocidad, salud y frecuencia de spawn.

Fase 4 · Firebase y backend Se integra Firebase Auth para autenticación real por email, Firestore para persistencia del progreso entre sesiones y la tabla de clasificación global en tiempo real. Se diagnostican y resuelven los principales bugs de Firebase.

Fase 5 · UI, polish y obstáculos Se desarrolla el HUD personalizado los menús de inicio, pausa y game over, y el sistema de setenta obstáculos físicos generados proceduralmente. Se implementan los efectos visuales post-proceso (CRT/bloom) mediante shader.

Fase 6 · Web, despliegue y entrega Se desarrolla la página web de presentación con estética terminal/classified, registro y login integrados, leaderboard en tiempo real y estadísticas globales. Se exporta el juego para Windows y Linux como GitHub Release y se despliega la web en GitHub Pages.

Para la planificación visual del proyecto se utilizó **Kaneo** como herramienta de diagrama de Gantt. Se eligió por ser open-source y no requerir registro ni instalación de software adicional — una herramienta enfocada exclusivamente en la visualización temporal del proyecto, sin funcionalidades innecesarias. Esto permitió tener una referencia visual clara de las fases y su distribución en el tiempo sin la complejidad de herramientas de gestión de proyectos más completas como Jira o Azure DevOps, que habrían sido excesivas para un proyecto individual.

Distribución temporal

Fase 1 · Configuración inicial y estructura del proyecto Duración: 1 semana

Fase 2 · Core gameplay: combate y enemigos Duración: 2 semanas

Fase 3 · Sistema de armas, progresión y escalado Duración: 3 semanas

Fase 4 · Firebase y backend Duración: 3 semanas

Fase 5 · UI, polish y obstáculos Duración: 2 semanas

Fase 6 · Web, despliegue y entrega Duración: 1 semana

Duración total: 12 semanas · Más de 200 horas de trabajo efectivo. Esta planificación refleja la distribución estimada del trabajo. El repositorio fue reiniciado el 22 de abril de 2026 tras problemas de gestión de ramas; los 34 commits previos quedan documentados en el README del proyecto.

3. Análisis

CU-01 · Consultar la web sin autenticación Actor: Visitante no registrado **Descripción:** El usuario accede a la página web de presentación sin necesidad de cuenta. **Flujo principal:**

1. El usuario accede a la web
2. Visualiza el leaderboard público con el top 10
3. Consulta las estadísticas globales (partidas jugadas, muertes, XP colectiva)

4. Descarga el ejecutable para Windows o Linux

Flujo alternativo: El usuario decide registrarse → redirige a CU-02 **Resultado:** El usuario obtiene información del juego y puede descargarlo sin necesidad de cuenta.

CU-02 · Registrarse e iniciar sesión Actor: Usuario no autenticado (web o juego) **Descripción:** El usuario crea una cuenta con email, contraseña y nombre de piloto, o inicia sesión con una cuenta existente. **Flujo principal:**

1. El usuario introduce email, contraseña y username
2. Firebase Auth crea la cuenta
3. Se crea el documento del usuario en Firestore
4. El usuario accede al juego o a su perfil en la web

Flujo alternativo: El email ya está registrado → se muestra error y se ofrece iniciar sesión · Contraseña incorrecta → se muestra error **Resultado:** El usuario queda autenticado y su progreso se sincroniza entre el juego y la web.

CU-03 · Jugar una partida Actor: Jugador autenticado **Descripción:** El jugador inicia y completa una partida completa. **Flujo principal:**

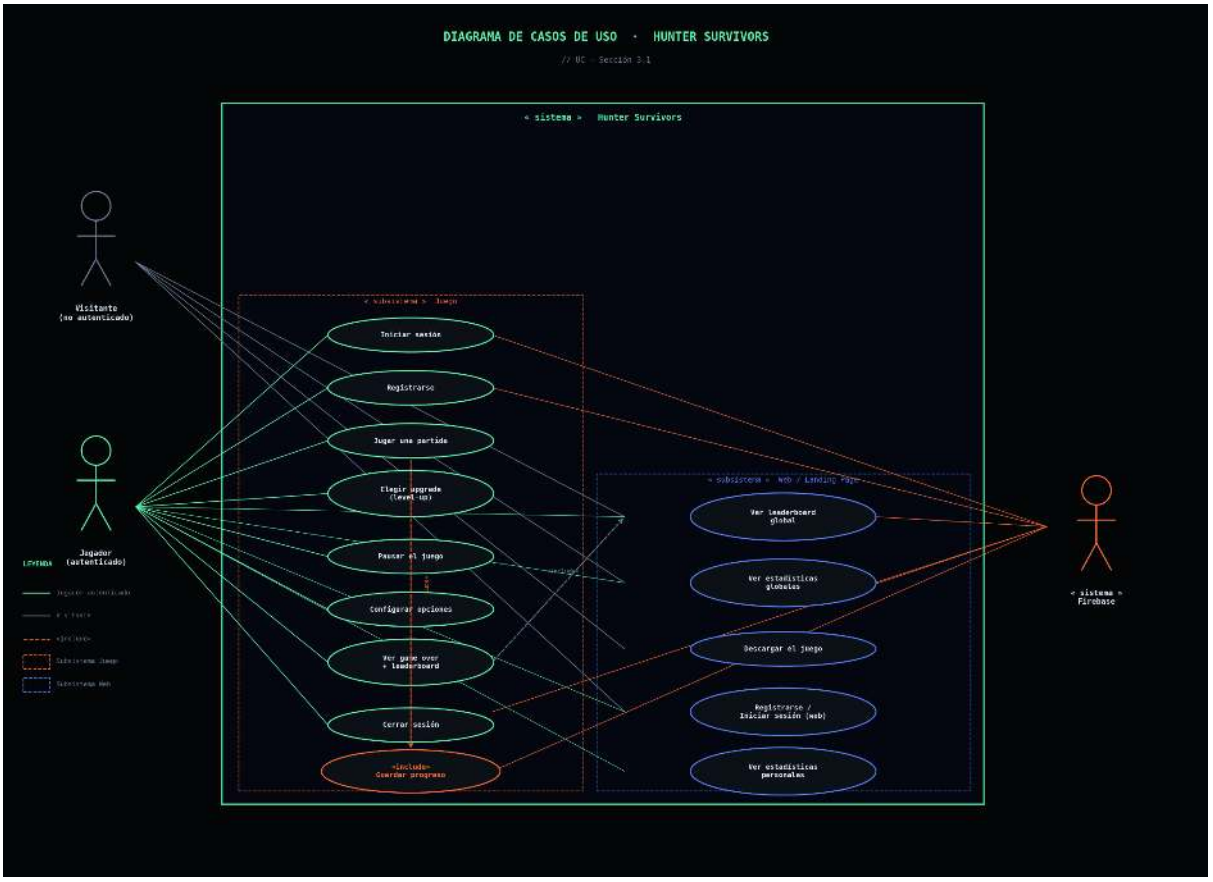
1. El jugador inicia sesión y pulsa Play
2. Se mueve con WASD/flechas
3. Las armas disparan automáticamente al enemigo más cercano
4. Al matar enemigos recoge orbes de XP
5. Al subir de nivel elige entre 3 opciones de mejora
6. La dificultad escala progresivamente hasta que el jugador muere

Flujo alternativo: El jugador pausa (ESC/P) → puede reanudar, reiniciar, ir al menú o cerrar sesión **Resultado:** Se registra el resultado de la partida y se guarda el progreso en Firestore.

CU-04 · Finalizar una partida y guardar progreso Actor: Jugador autenticado **Descripción:** Al morir, el sistema guarda automáticamente los resultados y actualiza el leaderboard si procede. **Flujo principal:**

1. El jugador pierde toda la vida
2. Se muestra la pantalla de game over con estadísticas del run (score, nivel, kills, XP, tiempo)
3. El progreso se persiste en Firestore
4. Si el score supera el récord anterior, el leaderboard global se actualiza
5. Se muestra la tabla de clasificación actualizada

Flujo alternativo: Error de conexión con Firebase → el progreso no se guarda pero el juego no se bloquea **Resultado:** El progreso queda persistido en la nube y el leaderboard refleja la nueva puntuación si procede.



3.2 Requisitos funcionales

- **RF-01 · Registro e inicio de sesión** El usuario puede registrarse e iniciar sesión con email y contraseña tanto en el juego como en la web. Las cuentas son compartidas entre ambas plataformas. **Prioridad:** Alta

- **RF-02 · Disparo automático** Las armas disparan automáticamente al enemigo más cercano sin ningún input del jugador. **Prioridad:** Alta

- **RF-03 · Stacking de armas** Se pueden tener hasta 3 armas activas simultáneamente, cada una con su propio cooldown independiente. **Prioridad:** Alta

- **RF-04 · Sistema de nivel y mejoras** Al subir de nivel, el jugador elige entre 3 opciones aleatorias: nuevas armas, pasivas o mejoras de estadísticas. **Prioridad:** Alta

- **RF-05 · Persistencia del progreso** El progreso del jugador (puntuación, nivel máximo, créditos, XP, muertes) se persiste en Firestore tras cada partida. **Prioridad:** Alta

- **RF-06 · Tabla de clasificación global** El leaderboard muestra el top 10 de jugadores ordenado por puntuación máxima, visible tanto en el juego como en la web. **Prioridad:** Alta

- **RF-07 · Estadísticas globales** Las estadísticas globales (partidas jugadas, muertes, XP, créditos) se actualizan de forma atómica tras cada partida y son visibles en la web sin autenticación. **Prioridad:** Media

- **RF-08 · Pausa con opciones** El juego se puede pausar con un overlay que ofrece reanudar, reiniciar, ir al menú principal o cerrar sesión. **Prioridad:** Media

- **RF-09 · Configuración de usuario** El menú de pausa y el menú principal incluyen opciones de configuración: screen shake, efectos post-proceso CRT/bloom y pantalla completa. **Prioridad:** Media

- **RF-10 · Web pública** La web muestra estadísticas globales y el leaderboard sin necesidad de autenticación. Los usuarios registrados pueden ver sus estadísticas personales. **Prioridad:** Media

- **RF-11 · Exportación multiplataforma** El juego exporta correctamente para Windows (.exe) y Linux (.x86_64) y está disponible como descarga pública en GitHub Releases. **Prioridad:** Alta

3.3 Requisitos no funcionales

- **RNF-01 · Rendimiento** El juego mantiene buen framerate hasta con una cantidad alta de enemigos renderizados. **Prioridad:** Alta

- **RNF-02 · Seguridad** Las reglas de Firestore garantizan que cada usuario solo puede leer y escribir sus propios datos. La colección leaderboard es de lectura pública para usuarios autenticados y la colección global_stats es de lectura pública sin autenticación, para la web. **Prioridad:** Alta

- **RNF-03 · Usabilidad** Los menús y el HUD son legibles a primera vista. Las mecánicas no requieren tutorial porque el disparo automático y el movimiento son suficientemente intuitivos para cualquier jugador. **Prioridad:** Media

- **RNF-04 · Portabilidad** El juego exporta correctamente para Windows 10+ y Linux (x86_64). Ambas versiones están disponibles como descarga pública en GitHub Releases. **Prioridad:** Alta

- **RNF-05 · Resiliencia** Los errores de Firebase fallan silenciosamente sin bloquear el gameplay ni la pantalla de game over. El juego es funcional sin conexión a internet, aunque no guarda el progreso. **Prioridad:** Alta

- **RNF-06 · Tamaño del ejecutable** El ejecutable pesa aproximadamente 100 MB debido al runtime de Godot 4 con Vulkan incluido. Este tamaño está documentado en el README del repositorio para que el usuario sepa qué esperar antes de descargar. **Prioridad:** Baja

3.4 Análisis de alternativas tecnológicas

Motor de juego: Godot 4 vs Unity

Unity fue descartado por varias razones. En septiembre de 2023, Unity anunció cambios en su modelo de licencia (Runtime Fee) que generaron una gran controversia y demostraron que el modelo de negocio podía cambiar unilateralmente sin previo aviso. Godot 4 es completamente open-source (licencia MIT), con un editor de tan solo ~100 MB frente a varios GB de Unity.

Otro factor determinante fue la curva de aprendizaje. Unity requiere C# como lenguaje principal, lo que implica familiarizarse con un lenguaje compilado, tipado estático y un ecosistema de herramientas externas. Godot ofrece GDScript, un lenguaje propio con sintaxis muy similar a Python:

interpretado, con tipado opcional y diseñado específicamente para el desarrollo de juegos. Esto permite centrarse en la lógica del juego desde el primer día, sin necesidad de configurar compiladores ni gestionar dependencias externas. Para un proyecto de ciclo formativo donde el tiempo es limitado, esta diferencia es significativa.

La comunidad de Godot 4 creció enormemente tras la controversia de Unity, lo que se tradujo en más recursos, tutoriales y respuestas disponibles en foros y comunidades online. Los primeros resultados de búsqueda para cualquier problema específico de Godot suelen llevar a soluciones actualizadas y relevantes.

Backend: Firebase vs Supabase vs backend propio

Supabase fue considerado como alternativa por tener una API más estándar (PostgreSQL relacional) y ser también open-source. Se descartó porque no existía un addon para Godot 4 tan maduro com godot-firebase. Un backend propio (Node.js + Express + PostgreSQL) fue descartado por el coste operacional (servidor) y la complejidad de mantenimiento para un proyecto individual.

Addon Firebase: godot-firebase vs llamadas HTTP directas

El addon godot-firebase v2.6 simplifica la mayoría de operaciones de Firestore (get_doc, set_doc, list) abstrayendo las llamadas HTTP a la API REST de Firebase en funciones de GDScript directamente utilizables. Esto reduce significativamente el código necesario para operaciones comunes como leer un documento, escribir datos o hacer queries sobre una colección.

Sin embargo, el addon presenta limitaciones en casos de uso más específicos. Durante el desarrollo se detectó un bug crítico en la implementación de los field transforms (operaciones atómicas como incrementar contadores), donde el parámetro currentDocument generaba una precondición incorrecta que impedía actualizar documentos ya existentes. Este tipo de problema es difícil de diagnosticar desde fuera porque el addon actúa como caja negra: el error que devuelve Firebase (FAILED_PRECONDITION) no indica directamente qué campo de la petición es el causante.

Se diagnosticó con dificultad, haciendo necesario el uso de otras herramientas. Esto demostró que ambos enfoques son completamente compatibles dentro del mismo proyecto: el addon para las operaciones estándar donde funciona correctamente, y llamadas HTTP directas para los casos donde se necesita control total sobre la petición.

4. Diseño

4.1 Arquitectura del sistema

El juego sigue una arquitectura de **sistemas desacoplados** donde `game.gd` actúa como orquestador ligero (~450 líneas) que conecta sistemas, gestiona señales y coordina transiciones de estado. La arquitectura surgió de una refactorización: en una versión intermedia del desarrollo, `game.gd` tenía más de 600 líneas mezclando nueve responsabilidades distintas. Añadir cualquier feature o corregir un bug requería leer y entender el fichero completo.

Cada responsabilidad se extrajo en su propia clase `RefCounted` (sin nodo, sin overhead de árbol de escena):

- `RunStats` — estadísticas de partida y lifetime. Métodos: `reset()`, `apply_loaded_profile()`, `on_pickup_collected()` → Array[int] de nuevos niveles, `finalize_run()` → dict de stats.
- `DifficultySystem` — tres escaladores (spawn interval, speed scale, health scale) y método `tick()`
- `SpawnSystem` — spawning ponderado por tipo y nivel, cálculo dinámico del intervalo.
- `LevelUpSystem` — cola de niveles pendientes y construcción de pools de ofertas aleatorias.
- `WeaponSystem` — array de hasta 3 WeaponItem activos con cooldowns independientes, armas pasivas.
- `SpatialHash` — particionamiento espacial para búsquedas de proximidad eficientes.
- `ObstacleSystem` — generación procedural de 70 obstáculos al inicio de cada partida.

4.2 Modelo de datos en Firestore

```
/users/{userId}  username: String  total_coins: int  // créditos acumulados total  highest_level: int  // mayor nivel alcanzado  best_score: int  // mejor puntuación  total_xp_collected: int  // XP acumulada lifetime  lifetime_deaths: int  // muertes totales  current_level, current_xp, current_health, move_speed...

/leaderboard/{userId}  username: String  best_score: int  last_updated: int (unix timestamp)

/global_stats/summary  total_games_played: int  total_deaths: int  total_xp_collected: int  total_coins_earned: int
```

Las **reglas de seguridad de Firestore** garantizan aislamiento: `/users` solo es accesible por el propio usuario autenticado; `/leaderboard` es de lectura para cualquier usuario autenticado y escritura solo del propio uid; `/global_stats` es de lectura pública (para la web) y escritura para cualquier usuario autenticado.

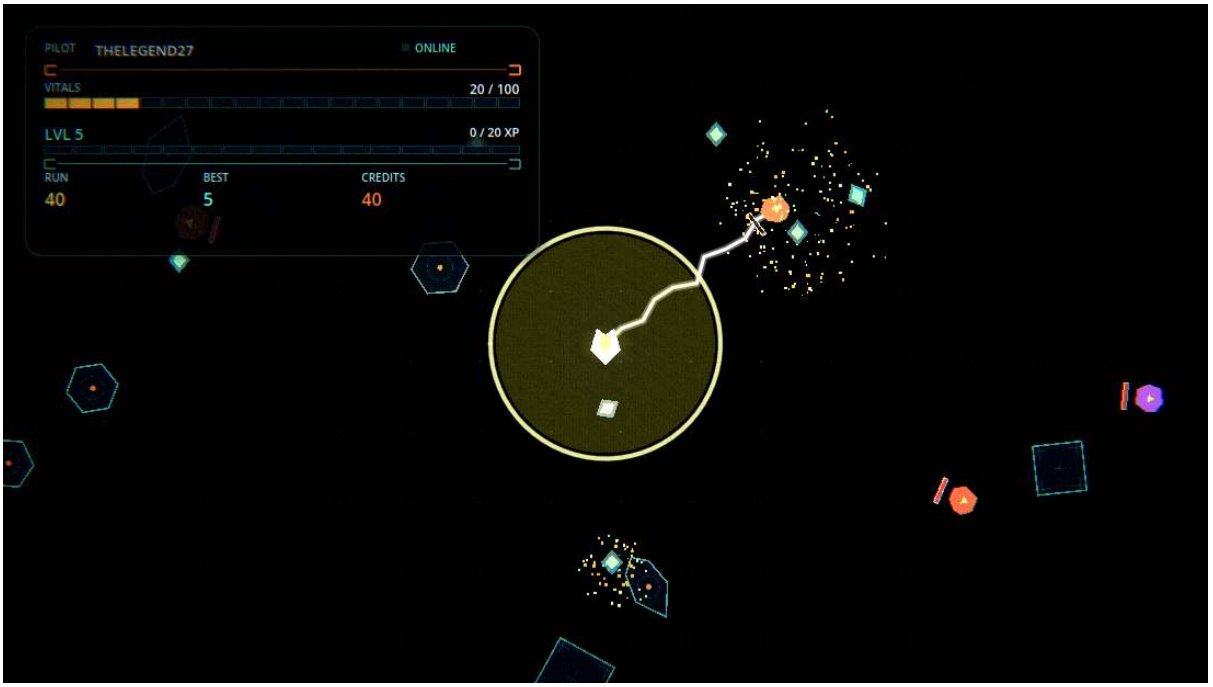
4.3 Diseño de interfaz

Estética visual — terminal/classified

La interfaz del juego y la web comparten un lenguaje visual coherente inspirado en interfaces tácticas de ciencia ficción. Los referentes estudiados fueron directamente inspirados por el videojuego de extracción **Marathon**, teniendo como referentes las paginas **marathonthegame.com** (Bungie, 2025) para el tono general sci-fi, **tauceti.gg** para la interfaz tipo terminal con cuadrículas y terminología táctica, y **cryoarchive.systems** para la estética de 'documento clasificado' con fuentes monoespaciadas y efectos CRT. La paleta de colores: fondo negro (#030507), acento teal (#5bf5af), acento naranja (#ff6b35).

HUD personalizado

El HUD está implementado íntegramente con `_draw()` en el hud, sin usar ningún nodo de UI de Godot. Esto da control total sobre el aspecto visual y garantiza que se vea exactamente como se diseñó. Incluye barra de salud segmentada (20 segmentos con código de color según HP), barra de XP (16 segmentos), nombre del piloto con indicador parpadeante y fila de estadísticas.



Página web

HTML/CSS/JS puro con Firebase JS SDK, sin frameworks. Tipografía monoespaciada IBM Plex Mono y Space Mono, sin bordes redondeados, efectos de scanlines CRT mediante CSS. Incluye registro y login directo, estadísticas personales tras autenticarse, leaderboard en tiempo real y estadísticas globales de todos los jugadores.

5. Desarrollo

5.1 Estructura del proyecto

```
hunter-survivors/├── scripts/core/           → game.gd + sistemas
desacoplados│   └── systems/          → SpawnSystem, LevelUpSystem,
SpatialHash, etc.├── scripts/entities/       → player, enemy, projectile,
obstacle, pickup├── scripts/ui/             → HUD, menús, login, game over,
pausa├── scripts/resources/      → WeaponItem (recurso personalizado de
Godot)├── scenes/                 → archivos .tscn├── addons/godot-firebase/
→ addon REST v2.6├── shaders/               → post_process.gdshader
(CRT/bloom)├── docs/                   → web (GitHub Pages)└── database.gd
→ Autoload Firebase Auth + Firestore
```

5.2 Implementación de funcionalidades principales

El jugador y la decisión de eliminar el disparo del player

Inicialmente, `player.gd` contenía la lógica de disparo: un `fire_interval` propio, búsqueda del enemigo más cercano y emisión de la señal `shoot_requested(origin, direction)`. Este diseño funcionó para el MVP pero imposibilitaba tener múltiples armas con cadencias independientes: todas disparaban al mismo ritmo que marcaba el jugador.

La solución fue eliminar completamente el disparo del jugador. `player.gd` quedó reducido a 12 líneas de movimiento puro. El `WeaponSystem` asumió la responsabilidad de buscar el objetivo más cercano y disparar cada arma según su propio cooldown. La búsqueda de objetivo usa el `SpatialHash` para evitar iterar todos los enemigos cada frame.

Sistema de armas — tres iteraciones

Iteración 1 — weapon_mode como entero: un campo `weapon_mode: int` determinaba el tipo de proyectil. Las armas no podían apilarse.

Iteración 2 — clases Weapon con tick(): cada arma era una clase propia con `tick(delta, runtime)`. Extensible pero generó conflictos de merge graves con el código paralelo.

Iteración 3 (final) — WeaponItem + WeaponData: el `WeaponSystem` mantiene `equipped_weapons: Array[WeaponItem]` con hasta 3 elementos y un array paralelo `_weapon_cooldowns`. Los datos de balance están en `weapon_data.gd` como diccionarios por modo. El método `tick_active_weapons()` decrementa cada cooldown y dispara cuando llega a cero.

SpatialHash — de búsquedas O(n) a O(1) por consulta

Con 70 enemigos activos y múltiples búsquedas por frame (aura, chain lightning, separación entre enemigos, daño de contacto), el rendimiento era un problema. La solución fue implementar un SpatialHash: una cuadrícula donde cada celda contiene referencias a los nodos que se encuentran en esa zona. Para una búsqueda de radio R en una cuadrícula de celda C, en lugar de iterar todos los n nodos se inspeccionan solo las celdas dentro del radio, reduciendo dramáticamente el número de comparaciones. El hash se reconstruye una vez por frame.

Obstáculos procedurales

Al inicio de cada partida, el `ObstacleSystem` genera 70 obstáculos `StaticBody2D` distribuidos aleatoriamente en un radio de ±1400 unidades del mapa, con una zona libre de 220 unidades

alrededor del spawn del jugador. Con un 35% de probabilidad, cada obstáculo genera un segundo obstáculo cercano formando clusters naturales de 2-3 elementos.

Los enemigos esquivan los obstáculos automáticamente gracias a `move_and_slide()`, sin necesidad de cambios en su IA. Los proyectiles los atraviesan porque su `_on_body_entered` solo actúa sobre nodos con `take_damage()`, que los obstáculos no tienen.

5.3 Dificultades encontradas — diagnóstico y solución

Dificultad 1 — Errors de inferencia de tipos en Godot 4.6

Cómo se detectó:

Al actualizar a Godot 4.6, el editor mostraba múltiples errores de parseo al abrir el proyecto:

```
Parser Error: Cannot infer the type of "collection" variable because the value doesn't have a set type.Parser Error: Cannot infer the type of "target_id" variable because the value doesn't have a set type.Trying to return value of type "Nil" from a function whose return type is "Array".
```

Investigación:

Se buscó en **Google** la combinación de los mensajes de error con 'Godot 4.6'. Los primeros resultados llevaron a un hilo en **r/godot** titulado 'GDScript 4.6 breaking changes — inference is stricter now' con más de 200 comentarios donde varios desarrolladores reportaban el mismo problema. Un miembro del equipo de Godot explicó que la versión 4.6 introdujo validación de tipos más estricta en tiempo de parseo.

Para el error de `Nil from Array`, se buscó en el **Discord oficial de Godot** (canal `#gdscript-help`) y se encontró que el addon godot-firebase tenía funciones marcadas como `-> Array` que en ciertas rutas retornaban el resultado de `await _handle_task_finished()`, un helper con tipo de retorno `Variant`. Un usuario explicó que esto era una incompatibilidad con los checks de Godot 4.6.

Solución:

- Para `"Cannot infer the type"`: anotaciones de tipo explícitas — `var collection: FirestoreCollection = Firebase.Firestore.collection("leaderboard")`
- Para `"target_id"`: `var target_id: int = enemy.get_instance_id()`
- Para el addon: se parchearon las funciones `query()`, `list()` y `_list()` capturando el resultado en `Variant` y verificando `if result is Array` antes de devolver.

> **i** *Lección: actualizar a una versión nueva de Godot a mitad del desarrollo es arriesgado. Mejor fijar la versión al inicio y actualizar solo entre fases.*

Dificultad 2 — Conflictos de merge: archivos con marcadores HEAD

Cómo se detectó:

Tras hacer `git merge` entre las ramas `feature/weapons` y `feature/firebase`, Godot mostraba un error de parseo al intentar cargar la escena:

```
GDScript::reload: Parse error at line 5: Expected end of statement.
```

Abrir el archivo revelaba los marcadores de merge de git que GDScript intentaba parsear como código:

```
<<<<<<< HEAD  signal aura_changed(radius: float, active: bool)=======  const WeaponDataScript = preload(...)>>>>>>> dcdc05faf4
```

Investigación:

Se buscó en **Stack Overflow** 'godot gdscript merge conflict markers parse error'. El problema era conocido: Godot no tiene ningún mecanismo especial para merge conflicts y los marcadores `<<<<<<<`, `=======` y `>>>>>>>` son caracteres inválidos en GDScript. En un hilo de **GitHub**

Discussions de Godot se discutía la posibilidad de integrar soporte de VCS en el editor, pero la solución era simplemente resolver los conflictos manualmente.

Gravedad:

Los archivos afectados fueron `weapon_system.gd`, `game.gd`, `level_up_system.gd` y `projectile.gd`. Cada uno tenía entre 2 y 4 bloques de conflicto de más de 100 líneas. Resolverlos tomó varias horas. El repositorio completo tuvo que ser reiniciado el 22 de abril de 2026 tras un estado imposible de resolver.

Solución:

Se resolvieron los conflictos archivo por archivo, eligiendo la versión de la rama que el desarrollador había trabajado más recientemente e integrando manualmente los cambios necesarios de la otra rama. Para futuros proyectos, la estrategia correcta es hacer merges frecuentes y pequeños, o trabajar en una sola rama con commits atómicos.

> ⚠ *El historial de 34 commits previo al reset se documenta en el README del proyecto para transparencia. La decisión de reiniciar fue pragmática: mejor repositorio limpio que estado de merge corrupto.*

Dificultad 3 — Leaderboard: query con orderBy requiere índice

Cómo se detectó:

El leaderboard de la pantalla de game over no se cargaba. En el output del editor de Godot apareció:

```
Firebase Error >> Action in error was: 4 (QUERY COLLECTION)
```

El código de acción 4 corresponde a `TASK_QUERY`, es decir, la llamada a `Firebase.Firestore.query()` con `order_by('best_score', DESCENDING)`.

Investigación:

Se buscó en **Google** 'Firestore query orderBy error 400 FAILED_PRECONDITION'. La **documentación oficial de Firebase** explicaba que Firestore requiere un índice compuesto para cualquier consulta con `orderBy` sobre un campo diferente al `__name__` del documento. El error 400 con status FAILED_PRECONDITION es exactamente esto. En **Stack Overflow** se encontraron varias preguntas con el mismo error, confirmando que la solución era o bien crear el índice manualmente en la Firebase Console, o bien evitar el orderBy.

Solución:

En lugar de crear el índice (que requeriría configuración externa fuera del código), se cambió la estrategia: `Firebase.Firestore.list('leaderboard', 200)` lista todos los documentos sin ordenar (GET simple, sin índice necesario), y la ordenación se hace client-side con `entries.sort_custom()` en GDScript. Para los volúmenes esperados (cientos de jugadores) esto es viable.

Dificultad 4 — El bug del IncrementTransform: el más difícil de diagnosticar

Cómo se detectó:

Las estadísticas globales de la web (muertes totales, XP, créditos, partidas) siempre mostraban '—'. Se verificaron las reglas de Firestore (correctas), se comprobó la autenticación (correcta). Se añadió un `push_warning()` en el callback del commit y apareció:

```
[GlobalStats] push failed: {"code": 400, "status": "FAILED_PRECONDITION",
"message": "Document already exists."}
```

El error era confuso: el documento **debería** existir y debería actualizarse. La primera partida funcionaba (creaba el documento), las siguientes fallaban.

Investigación:

Se buscó en Google 'Firestore IncrementTransform precondition document already exists' sin resultados directos para GDScript. Se buscó en **Stack Overflow** 'Firestore field transform currentDocument exists precondition'. Un artículo en inglés sobre la API REST de Firestore explicaba que el campo `currentDocument` en los field transforms actúa como precondición: `{"exists": false}` significa 'solo aplica si el documento NO existe'. Esto encajaba exactamente con el comportamiento observado.

Se abrió directamente el código fuente del addon: `addons/godot-firebase/firestore/field_transform_array.gd`. La función `serialize()` siempre incluía `"currentDocument": { "exists" : transform.document_exists }`. Y `IncrementTransform._init()` asignaba el parámetro `doc_must_exist: false` a `document_exists`. Se había llamado con `false` asumiendo que significaba 'créalo si no existe', pero en realidad `{"exists": false}` es una precondición que dice '**SOLO aplica si el documento NO existe**'.

Solución:

Se descartó el addon para esta operación y se implementó una llamada HTTP directa al endpoint `:commit` de Firestore sin el campo `currentDocument`. Sin precondición, Firestore crea el documento en la primera escritura y lo actualiza en todas las siguientes. Los contadores funcionaron correctamente desde entonces.

Dificultad 5 — HUD cortado en el borde inferior

Cómo se detectó:

Visualmente, los valores numéricos de la fila de estadísticas (RUN, BEST, CREDITS) no se veían en el HUD. La barra de salud y la de XP eran correctas pero la parte inferior del panel quedaba cortada.

Investigación:

Se trazó el layout manualmente calculando las posiciones Y de cada elemento. La función `draw_string()` de Godot dibuja texto con el **baseline en la coordenada Y indicada**. El contenido más bajo era el texto de estadísticas con fuente de 16px a y=150 (coordenadas locales del control). Con la altura mínima configurada en 157px, el texto de 16px llegaba a y≈172, fuera del área visible.

Se consultó la documentación de Godot sobre `draw_string()` para confirmar el comportamiento del baseline, y se verificó que los PanelContainer añaden márgenes internos que reducen el área de contenido.

Solución:

Se aumentaron las dimensiones en cascada: `TacticalHUD.custom_minimum_size` de 414×157 a 434×182, `HUDPanel.custom_minimum_size` de 450×185 a 470×210, y `MarginContainer.offset_bottom` de 205 a 230.

Dificultad 6 — Obstáculos que no aparecían

Cómo se detectó:

Después de implementar el sistema de obstáculos, no aparecía ninguno en el mapa. No había mensajes de error en el editor.

Investigación:

Se añadió un `print("spawn called")` al inicio de `ObstacleSystem.spawn()` y no apareció en la consola, lo que confirmaba que el método nunca se llamaba. Se verificó que `ObstacleSystemScript.new()` se ejecutaba en `game.gd`, pero la variable `ObstacleSystemScript` era un GDScript con error de parseo silencioso que lo dejaba inválido.

El error estaba en una línea que usaba `\` para continuar la línea (hábito de Python que no funciona igual en todos los contextos de GDScript):

```
var buddy := candidate + Vector2.RIGHT.rotated(randf() * TAU) \    *
randf_range(80.0, CLUSTER_RADIUS)
```

Se buscó en **la documentación de GDScript** si la barra invertida era un operador de continuación de línea válido. En GDScript 2 (Godot 4) sí está soportado, pero puede fallar en ciertos contextos de declaración de variables. La solución fue usar una variable intermedia, eliminando la ambigüedad.

Solución:

```
var dir := Vector2.RIGHT.rotated(randf() * TAU)var buddy := candidate + dir
* randf_range(80.0, CLUSTER_RADIUS)
```

⚠ *Este bug es especialmente traicionero: Godot no lanza un error explícito de 'obstacle_system.gd failed to load', simplemente el script queda inválido y la variable preloaded referencia un objeto null. El síntoma (método nunca llamado) requiere inferir que el problema está en la carga del script, no en su lógica.*

Dificultad 7 — Obstáculos concentrados cerca del origen del mundo

Cómo se detectó:

Con los obstáculos ya spawnando, se observó que todos aparecían en un radio de ~580 unidades desde el origen del mundo (0,0), no desde el jugador. El jugador spawnea en (576, 324), así que la distribución de obstáculos quedaba descentrada y asimétrica respecto al área de juego.

Solución:

La función de generación de posiciones usaba (0,0) como origen. Se corrigió pasando `player_world_pos` y usándolo como centro del scatter: `return player_pos + Vector2.RIGHT.rotated(angle) * dist`.

Dificultad 8 — Reglas de Firestore mal anidadas

Cómo se detectó:

Las estadísticas globales de la web mostraban '—' para todos los campos excepto 'Registered Pilots' que mostraba el número correcto. Los errores de consola del navegador mostraban código 403 (Forbidden) en las peticiones a `/global_stats/summary`.

Investigación:

Se revisó el panel de Firestore Rules en Firebase Console. Las reglas para `/global_stats` estaban definidas correctamente, pero estaban **anidadas dentro del bloque de** `/listings` por un error de llaves. Al estar mal anidadas, el path efectivo era `/listings/{listingId}/global_stats/{doc}` en lugar de `/global_stats/{doc}`. El path incorrecto nunca coincidía con las peticiones reales.

Solución:

Recolocar el bloque `match /global_stats/{doc}` al mismo nivel que `/users` y `/leaderboard`, fuera del bloque de `/listings`.

Dificultad 9 — Leaderboard no mostraba pilotos registrados

Cómo se detectó:

Después de corregir las reglas y que las estadísticas globales empezaran a aparecer, el contador de 'Registered Pilots' volvió a mostrar '—'. Antes mostraba el número correcto.

Diagnóstico:

El código JS de la web leía `d.total_players` del documento `global_stats/summary`. Pero la función `push_global_stats()` en el juego nunca escribía ese campo. El campo `total_players` simplemente no existía. Antes funcionaba porque cuando el documento no existía, la web caía en un fallback que contaba documentos de la colección `/leaderboard`. Con el documento existiendo (aunque sin el campo), ese fallback ya no se ejecutaba.

Solución:

En lugar de escribir el campo (lo que requeriría lógica para no duplicar por el mismo usuario), la web siempre obtiene el conteo de pilotos contando documentos en `/leaderboard`, independientemente de si el documento de global_stats existe o no. Este conteo es más preciso (un documento = un usuario real que ha jugado y puntuado) y no requiere mantenimiento.

6. Recursos y búsqueda de información

6.1 Documentación oficial

La **documentación oficial de Godot** (docs.godotengine.org) fue la referencia constante durante todo el desarrollo. Las páginas más consultadas fueron: `CharacterBody2D` (move_and_slide, colisiones), `Area2D` (body_entered, monitoring), `CanvasItem._draw()` (draw_rect, draw_circle, draw_string), el sistema de señales, `RefCounted` para sistemas sin nodo, y GDScript — tipos y type hints.

La **documentación de Firebase** (firebase.google.com/docs) fue necesaria para entender la API REST de Firestore, las field transforms, el formato de las reglas de seguridad y la autenticación con Identity Toolkit.

6.2 Reddit — r/godot

El subreddit **r/godot** fue especialmente útil para preguntas específicas de GDScript 2 que no estaban bien documentadas. La búsqueda habitual era `site:reddit.com/r/godot "[término del error]"` en Google, lo que generalmente encontraba hilos con la misma pregunta.

Los hilos más útiles fueron: uno sobre los cambios de inferencia de tipos en Godot 4.6 que generó más de 200 comentarios con soluciones, otro sobre cómo usar `RefCounted` para sistemas sin nodo de forma correcta, y varios sobre la integración con Firebase en Godot 4.

6.3 Discord oficial de Godot

El **Discord oficial de Godot** (discord.gg/godotengine) tiene canales especializados. El canal `#gdscript-help` fue donde se encontró la respuesta al problema de inferencia de tipos con autoloads. Un miembro explicó que `Firebase.Firestore` es de tipo `Variant` desde la perspectiva del parser porque es un autoload cargado como .tscn, y que la solución es siempre anotar el tipo explícitamente.

El canal `#showcase` fue útil para ver cómo otros desarrolladores habían estructurado proyectos similares con Firebase en Godot 4.

6.4 Stack Overflow

Para problemas de Firebase REST API (agnósticos de Godot), **Stack Overflow** con la etiqueta `google-cloud-firestore` fue muy útil. El diagnóstico del error FAILED_PRECONDITION del IncrementTransform se resolvió gracias a una respuesta que explicaba el campo `currentDocument` en los field transforms de la API REST de Firestore. Sin esa referencia, el debugging habría llevado mucho más tiempo.

También se consultó Stack Overflow para preguntas sobre GDScript (etiqueta `gdscript`), especialmente sobre el comportamiento de `await` con señales sin parámetros y sobre cómo funciona el parsing de `\` como carácter de continuación de línea.

6.5 GitHub — código de referencia

Se consultaron varios repositorios open-source para entender patrones de implementación:

- **GodotNuts/GodotFirebase** — el addon en sí. El código fuente fue necesario leerlo directamente para diagnosticar el bug del IncrementTransform. Sin leer `field_transform_array.gd`, el bug habría sido imposible de entender desde fuera.
- **Entradas de Godot Wild Jam** — varias entradas open-source de game jams con código de spawn systems y enemy AI en Godot 4 fueron consultadas para entender patrones de implementación.

6.6 YouTube y recursos en vídeo

- **GDQuest** — canal de tutoriales de Godot 4. Especialmente útil para entender el sistema de señales y la estructura de proyectos.
- **Game Developer Conference talks** — talks sobre diseño de juegos survivor en GDC Vault, especialmente el análisis de la arquitectura de sistemas de Vampire Survivors.
- **Fireship** — vídeos cortos sobre Firebase que ayudaron a entender conceptos de Firestore antes de buscar documentación específica para Godot.

6.7 Foros adicionales y otras fuentes

- **Godot Q&A** (godotengine.org/qa) — preguntas y respuestas sobre comportamientos específicos del motor.
- **GitHub Issues de godot-firebase** — se revisaron los issues abiertos del addon para ver si alguno documentaba los problemas encontrados.
- **Hilo de email del Tutor del proyecto**— feedback de un evaluador sobre la legibilidad del texto en la web llevó a ajustar los colores del CSS para mejorar el contraste.

7. Conclusiones

7.1 Conclusiones generales

Hunter Survivors es un proyecto que ha cubierto un espectro técnico amplio para un proyecto individual: motor de juego, scripting con GDScript, física 2D y colisiones, sistemas de entidades concurrentes, arquitectura de código desacoplada, integración con APIs de terceros, bases de datos NoSQL, reglas de seguridad en la nube, diseño web y despliegue continuo.

El resultado es un juego jugable y completo con usuarios reales, datos reales en Firebase y una web pública funcional. No es un prototipo: tiene autenticación, persistencia, clasificación global y descarga pública.

7.2 Consecución de objetivos

- ✔ Loop de juego completo con 6 tipos de enemigos, 7 armas activas y 2 pasivas.
- ✔ Sistema de stacking de hasta 3 armas simultáneas con cooldowns independientes.
- ✔ Autenticación real por email en juego y web, con cuentas compartidas.
- ✔ Progreso persistido en Firestore, sincronizado entre sesiones.
- ✔ Tabla de clasificación global en tiempo real.
- ✔ Web de presentación con datos en vivo y registro/login.
- ✔ Exportación funcional para Windows y Linux en GitHub Releases.
- ✔ Arquitectura en sistemas desacoplados con game.gd como orquestador.

7.3 Valoración de la metodología y planificación

La metodología iterativa funcionó bien para tener siempre una versión jugable. El principal error fue la gestión de branches en Git: trabajar en ramas paralelas sobre los mismos archivos core sin integración continua generó los conflictos de merge que obligaron a reiniciar el repositorio. En proyectos futuros, merges diarios o feature flags en una sola rama serían preferibles.

La búsqueda de información fue eficiente cuando se combinaron varios canales: Google para encontrar el tipo de error, Reddit/Discord para la explicación comunitaria, y la documentación oficial o el código fuente del addon para la solución definitiva. Las búsquedas en un solo canal raramente eran suficientes.

7.4 Visión de futuro

- Sistema de sonido — el juego es actualmente silencioso. Godot tiene un sistema de audio integrado con buses y efectos.
- Evoluciones de armas — combinar dos armas para crear una variante mejorada (Shotgun + Aura → Plasma Cannon).
- Refresh automático del id_token — actualmente caduca a 1 hora, lo que puede romper sesiones largas.
- Object pooling para proyectiles — especialmente relevante para la escopeta (5 proyectiles por disparo).
- Tests unitarios con GUT (Godot Unit Testing) para RunStats, DifficultySystem y LevelUpSystem.
- Soporte móvil con controles táctiles — Godot 4 exporta a Android/iOS con configuración adicional.

8. Glosario

Autoload: script singleton accesible globalmente en Godot, equivalente a un servicio. Se registra en la configuración del proyecto y está disponible desde cualquier escena sin necesidad de importarlo.

Bearer token: token de autenticación enviado en el header Authorization de las peticiones HTTP. Firebase lo utiliza para verificar la identidad del usuario en cada llamada a la API REST.

CharacterBody2D: nodo de Godot para personajes con física de movimiento y colisión. Proporciona el método move_and_slide() que gestiona automáticamente las colisiones con el entorno.

FAILED_PRECONDITION: código de error de Firestore que indica que una precondición en la petición no se cumple. En este proyecto apareció al intentar actualizar documentos ya existentes con una precondición incorrecta en los field transforms.

Field Transform: operación atómica de Firestore para modificar campos (como incrementar un contador) sin necesidad de leer el documento primero. Garantiza que la operación es segura en entornos con escrituras concurrentes.

Firestore: base de datos NoSQL orientada a documentos de Google Firebase. Los datos se organizan en colecciones y documentos, con soporte para lectura en tiempo real y reglas de seguridad declarativas.

GDScript: lenguaje de programación propio de Godot, con sintaxis similar a Python. No requiere compilador externo ni configuración adicional, lo que reduce la curva de aprendizaje respecto a otros motores que utilizan C#.

GitHub Pages: servicio de hosting gratuito que sirve sitios estáticos directamente desde un repositorio de GitHub. En este proyecto se utiliza para desplegar la web de presentación pública.

GitHub Releases: funcionalidad de GitHub para publicar versiones etiquetadas de un proyecto con archivos adjuntos. En este proyecto se utiliza para distribuir los ejecutables para Windows y Linux.

HUD (Heads-Up Display): interfaz superpuesta sobre el juego que muestra información relevante al jugador en tiempo real, como la salud, la XP o el score.

MIT (licencia): licencia de software libre que permite usar, modificar y distribuir el código sin restricciones, incluso en proyectos comerciales. Godot 4 está publicado bajo esta licencia.

MVP (Minimum Viable Product): versión mínima del producto que tiene las funcionalidades básicas para ser jugable o demostrable. En este proyecto se utilizó como punto de partida de cada fase de desarrollo.

RefCounted: clase base de Godot para objetos sin nodo que se liberan automáticamente cuando no hay referencias activas. Se utiliza en este proyecto para implementar los sistemas desacoplados sin overhead del árbol de escenas.

Spatial Hash: estructura de datos que divide el espacio en celdas para búsquedas de proximidad eficientes. Permite reducir las búsquedas de O(n) a O(1) por consulta, evitando iterar todos los enemigos cada frame.

Stacking (armas): posibilidad de tener múltiples armas activas simultáneamente, cada una con su propio cooldown independiente. En este proyecto se permite un máximo de tres armas activas a la vez.

StaticBody2D: nodo de Godot para objetos físicos inamovibles como obstáculos o paredes. Los enemigos los esquivan automáticamente gracias a move_and_slide(), sin necesidad de lógica adicional en su IA.

8. Bibliografía

- **Bibliografía**

- Branno. (2024). *Make a Vampire Survivors clone in Godot 4* [Lista de reproducción de YouTube]. https://www.youtube.com/playlist?list=PLtosjGHWDab682nfZ1f6JSQ1cjap7Ieeb
- GDQuest. (2024). *Your first 2D game from zero in Godot 4: Vampire Survivor style* [Vídeo de YouTube]. https://www.youtube.com/watch?v=GwCiGixlqiU
- Godot Engine. (2024). *GDScript reference*. https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/
- Godot Engine. (2024). *CharacterBody2D class reference*. https://docs.godotengine.org/en/stable/classes/class_characterbody2d.html
- Godot Engine. (2024). *CanvasItem._draw() reference*. https://docs.godotengine.org/en/stable/classes/class_canvasitem.html
- Godot Engine. (2024). *RefCounted class reference*. https://docs.godotengine.org/en/stable/classes/class_refcounted.html
- Google Firebase. (2024). *Firebase Authentication documentation*. https://firebase.google.com/docs/auth
- Google Firebase. (2024). *Firestore security rules*. https://firebase.google.com/docs/firestore/security/get-started
- GodotNuts. (2024). *GodotFirebase v2.6* [Software]. GitHub. https://github.com/GodotNuts/GodotFirebase
- GDQuest. (2024). *Godot 4 tutorials* [Canal de YouTube]. https://www.gdquest.com
- Fireship. (2024). *Firebase concepts* [Canal de YouTube]. https://www.youtube.com/c/Fireship
- Reddit. (2024). *r/godot community*. https://www.reddit.com/r/godot
- Godot Engine. (2024). *Godot Q&A*. https://godotengine.org/qa
- Bungie. (2025). *Marathon* [Referente de diseño visual]. https://www.marathonthegame.com

9. Anexos

Anexo A — Repositorio y descargas

Repositorio GitHub: https://github.com/emotionalshawty/hunter-survivors

Releases Windows/Linux: https://github.com/emotionalshawty/hunter-survivors/releases

Web de presentación: https://emotionalshawty.github.io/hunter-survivors

Anexo B — Reglas de seguridad Firestore finales

```
rules_version = '2';service cloud.firestore {  match
/databases/{database}/documents {    match /users/{userId} {      allow
read, write: if request.auth != null                          &&
request.auth.uid == userId;    }    match /leaderboard/{uid} {      allow
read:  if request.auth != null;      allow write: if request.auth != null
&& request.auth.uid == uid;    }    match /global_stats/{doc} {      allow
read:  if true;      allow write: if request.auth != null;    }  }}
```

Anexo C — Resumen de dificultades

9 dificultades técnicas documentadas:

1. Errores de inferencia de tipos en Godot 4.6 — parcheado el addon y añadidas anotaciones explícitas.
2. Conflictos de merge en Git — resolución manual archivo por archivo; reset del repositorio.
3. Leaderboard query con orderBy requiere índice — cambiado a list() + sort client-side.
4. Bug IncrementTransform (precondición incorrecta) — llamada HTTP directa al endpoint :commit.
5. HUD cortado en borde inferior — recalculadas dimensiones en cascada.
6. Obstáculos no spawnaban (barra invertida en GDScript) — variable intermedia.
7. Obstáculos concentrados en el origen del mundo — cálculo de posición relativo al jugador.
8. Reglas de Firestore mal anidadas — corrección de estructura de llaves.
9. Contador de pilotos desaparecía al existir el documento — conteo directo desde /leaderboard.

Nota sobre el uso de inteligencia artificial

Durante el desarrollo de este proyecto se ha utilizado **Claude (Anthropic)** como asistente de programación y redacción. El uso ha sido de carácter técnico y de soporte: depuración de errores específicos, refactorización de código existente, diagnóstico de bugs de Firebase, implementación de sistemas concretos y generación de la estructura inicial de esta memoria. Seguido esto, el texto se ha editado cuando se ha visto necesario

Todo el código generado con asistencia de IA ha sido revisado, comprendido y adaptado al proyecto. Las decisiones de diseño (arquitectura de sistemas, stack tecnológico, elecciones de Firebase vs alternativas) son propias. El autor comprende y puede explicar cada componente del código presente en el repositorio.

La redacción de contexto, justificación y reflexiones personales ha sido propia, con revisión de la IA para corrección lingüística y de formato. Los apartados de dificultades encontradas reflejan experiencias reales del desarrollo, no generadas por la IA.