DOCUMENTO FUNCIONAL DEL PROYECTO

ALUMNO/GRUPO: Ignacy Faldmo

---

1. Introducción y contexto

El proyecto consiste en el desarrollo de un videojuego de acción 2D estilo Vampire Survivors, donde el jugador se desplaza por el mapa mientras el personaje ataca automáticamente a oleadas de enemigos con dificultad creciente. Sobre este bucle de juego se construye un sistema de progresión con subida de nivel, elección de mejoras y obtención de armas y objetos de diferentes rarezas, todo ello acompañado de un inventario persistente y estadísticas que se guardan en la nube para poder continuar partidas y mantener el progreso entre sesiones.

Como elemento diferenciador, el juego incorporará un sistema de comercio online seguro entre jugadores, que permitirá publicar y aceptar ofertas de intercambio de objetos mediante un backend basado en Firebase.

Para este proyecto se utilizarán Godot 4 y GDScript en el cliente, junto con Firebase Authentication, Cloud Firestore y Cloud Functions para la autenticación, almacenamiento de inventarios y procesado atómico de los intercambios.

El desarrollo se organizará en un plazo aproximado de 6–8 meses, con control de versiones en GitHub y una planificación técnica que incluye definición de requisitos, análisis de riesgos y criterios claros de validación del producto final.

2. Análisis de requisitos

Para desarrollar el proyecto de forma adecuada, hay que establecer un set de requisitos que se tendrá que alcanzar, estos se dividen en 2 grupos, los funcionales (su cumplimiento es clave en el funcionamiento del juego) y los no funcionales(los que no son críticos para el funcionamiento del mismo). Hecho esto, se puede analizar las limitaciones y dificultades que se pueden encontrar en el proceso.

2.1. Requisitos funcionales (RF)

Los requisitos funcionales forman el core del proyecto.

Código	Descripción del requisito funcional
RF1	El sistema permitirá registrar/iniciar sesión a usuarios (anónimo y/o email) mediante Firebase Auth.
RF2	El jugador podrá crear una partida nueva o continuar una existente (carga de progreso en la nube).
RF3	Controlar el personaje y desplazarse por el mapa; los ataques se ejecutan automáticamente.
RF4	Generación de oleadas de enemigos con dificultad progresiva.
RF5	Sistema de experiencia y subida de nivel con selección de mejoras (armas/habilidades).
RF6	Sistema de botín: los enemigos y jefes sueltan objetos/armas con rarezas.
RF7	Gestión de inventario: recoger, equipar, desequipar y consultar armas/objetos.
RF8	Guardado automático del progreso (estadísticas, inventario, mejoras) en la nube.
RF9	Comercio básico entre jugadores: listar ofertas, solicitar intercambio y confirmación segura.
RF10	Aparición de jefes en hitos temporales con recompensas especiales.
RF11	Pantallas de UI: menú principal, selección de personaje, inventario, mejoras y comercio.
RF12	Opciones del juego: audio, idioma, sensibilidad/controles y accesibilidad básica.

2.2. Requisitos no funcionales (RNF)

Los requisitos no funcionales son los objetivos que mejoran la experiencia pero no son vitales para el éxito del proyecto.

Código	Descripción del requisito no funcional
RNF1	La página web será accesible en móviles
RNF2	Soporta 100+ enemigos simultáneos en pantalla.
RNF3	Guardado automático cada 10 segundos.
RNF4	Hash bcrypt para credenciales de usuario.
RNF5	Interfaz intuitiva con máximo 3 clics para funciones principales.
RNF6	Tiempo de aprendizaje máximo 5 minutos para nuevos jugadores.
RNF7	Código documentado con comentarios en funciones críticas.
RNF8	Configuración en formatos JSON/YAML.
RNF9	Sincronización con servidor para progreso en la nube.

2.3. Restricciones

Como en todo desarrollo, el proyecto tiene ciertos límites técnicos, de tiempo y de recursos que van a marcar hasta dónde podemos llegar y qué decisiones de diseño hay que tomar. Estas son las principales restricciones a las que nos enfrentamos:

1. Restricciones Tecnológicas y de Entorno

- **Dependencia de Godot y GDScript:** Todo el proyecto gira en torno a Godot 4.x. Aunque el motor no requiere del uso directo de APIs como Vulkan o DirectX, si necesitamos algo muy específico que Godot no traiga de serie, dependeremos de plugins de la comunidad o soluciones externas. Además, al usar GDScript en lugar de C++, ganamos velocidad programando, pero hay que vigilar el rendimiento en procesos muy pesados para que el juego no tenga bajones de rendimiento.
- **Backend como punto crítico de fallo:** La arquitectura depende al 100% de la disponibilidad y estado de Firebase. Si los servicios de Google caen o el usuario pierde la conexión, funciones clave como loguearse, acceder al inventario o comerciar dejarán de funcionar.
- **Librerías de terceros:** Para conectar Godot con servicios externos dependemos de librerías Open Source (como los adaptadores de Firebase o GodotSteam). El riesgo aquí es que si el motor se actualiza y los creadores de estas librerías no sacan parches rápido, podríamos quedarnos bloqueados en una versión antigua de Godot o tener problemas de compatibilidad.

2. Restricciones de backend (Plan Gratuito)

Al usar el plan "Spark" (gratuito) de Firebase para el proyecto, tenemos unos límites claros que no podemos superar en esta fase:

- **Conexiones simultáneas:** La base de datos en tiempo real tiene un máximo de 100 usuarios a la vez. Para el proyecto final sobra, pero es un techo técnico importante.
- **Almacenamiento:** Solo disponemos de 1 GB en Cloud Firestore. Esto nos obliga a diseñar la base de datos de forma muy eficiente, guardando solo lo justo y necesario (nada de logs masivos o archivos pesados) para no saturar el espacio.

3. Restricciones de Despliegue y Hardware

Entorno de pruebas limitado: Las pruebas y el debugging se harán únicamente en los equipos disponibles: PC con Windows y Linux (Ubuntu).

Peso del ejecutable: Se ha puesto un límite de unos 200 MB para el juego base más los assets. Esto implica que habrá que utilizar assets con alta comprensión o con una fidelidad que permita el tamaño.

4. Restricciones Temporales y de Recursos

Cronograma Fijo: El ciclo de desarrollo está acotado estrictamente a 6-8 meses. Esta ventana temporal es inamovible y obliga a priorizar las mecánicas del MVP (Producto Mínimo Viable) sobre características secundarias. Cualquier retraso en la fase de núcleo jugable impactará directamente en la calidad o existencia del mismo.

3. Análisis de usuarios y roles

Para configurar los puntos de acceso y los permisos del programa frente a la base de datos, hay que analizar qué usuarios podrán interactuar con la app.

Rol	Descripción	Permisos principales
Administrador	Gestiona usuarios y recursos.	Alta, baja y modificación de datos.
Usuario/ Jugador	Interactúa con el juego de forma normal	Crear, modificar y consultar sus propios datos
Visitante	Consulta información del juego (página web)	Sin permisos

4. Casos de uso / Escenarios de uso

Para ver como se podrá interactuar con la base de datos de la aplicación, habrá que analizar de que maneras se interactúa la base de datos/aplicación.

Código	Nombre del caso de uso	Actor principal	Descripción	Resultado esperado
CU1	Registrar usuario	Visitante	El usuario introduce sus datos y se crea una cuenta.	Usuario registrado correctamente.
CU2	Iniciar sesión	Usuario	Introduce credenciales y accede al sistema.	Acceso concedido.
CU3	Creación de fichero de guardado	Usuario	Juega al juego, guarda su progreso	La partida se carga con el progreso, inventario y estadísticas del archivo.
CU4	Iniciar nueva partida	Jugador	Elige un personaje y un mapa de los disponibles para comenzar a jugar.	La partida se inicia con el personaje y mapa seleccionados.
CU5	Controlar personaje	Jugador	Mueve al personaje por el mapa usando el teclado o un mando.	El personaje se desplaza fluidamente por el escenario.

CU6	Combatir contra enemigos	Jugador/Sistema	El personaje ataca automáticamente a los enemigos cercanos que aparecen en oleadas	Los enemigos reciben daño y son eliminados al llegar a cero su vida.
CU7	Progresar de nivel	Jugador	Recoge gemas de experiencia de enemigos derrotados para subir de nivel	Al subir de nivel, aparece una pantalla para seleccionar una nueva arma o mejora.
CU8	Recolectar botín y objetos	Jugador	Camina sobre cofres, oro u otros objetos que sueltan los enemigos.	Los objetos se añaden al inventario y el oro se suma al total del jugador
CU9	Evolucionar armas	Sistema	Combina un arma de nivel máximo con su accesorio pasivo correspondiente.	El arma se mejora
CU10	Finalizar partida	Jugador/Sistema	La vida del personaje llega a cero, finalizando la partida	Aparece la pantalla de "Game Over" con las estadísticas de la sesión
CU11	Cargado del progreso	Jugador	Selecciona un archivo de guardado existente para continuar su partida	La partida se carga con el progreso, inventario y estadísticas del archivo.
CU12	Consultar desbloqueos	Usuario	Accede a los menús para ver los personajes, armas y logros desbloqueados	El sistema muestra correctamente todo el contenido desbloqueado por el usuario.

5. Modelo de datos o estructura de la información

El proyecto utilizará sistema de base de datos basado en json, ya que el juego tendrá su backend gestionado por Firebase y sus diferentes sistemas/servicios. Esto hará que diversas cosas se puedan automatizar

Diagrama de Estructura de Datos

Colección: `players`

- **Documento: `{player_uid}`** (ID único de autenticación del jugador)
    - **Campo:** `displayName` (string)
    - **Campo:** `createdAt` (timestamp)
    - **Campo:** `lastLogin` (timestamp)
    - **Sub-colección: `inventory`**
        - **Documento: `{item_instance_id}`** (ID único de la instancia del objeto)
            - **Campo:** `itemId` (string) - *Ej: "fire_sword", "ice_bow"*
            - **Campo:** `rarity` (string) - *Ej: "Común", "Raro", "Legendario"*
            - **Campo:** `power` (number)
            - **Campo:** `level` (number)
            - **Campo:** `acquiredAt` (timestamp)

Colección: `trade_listings`

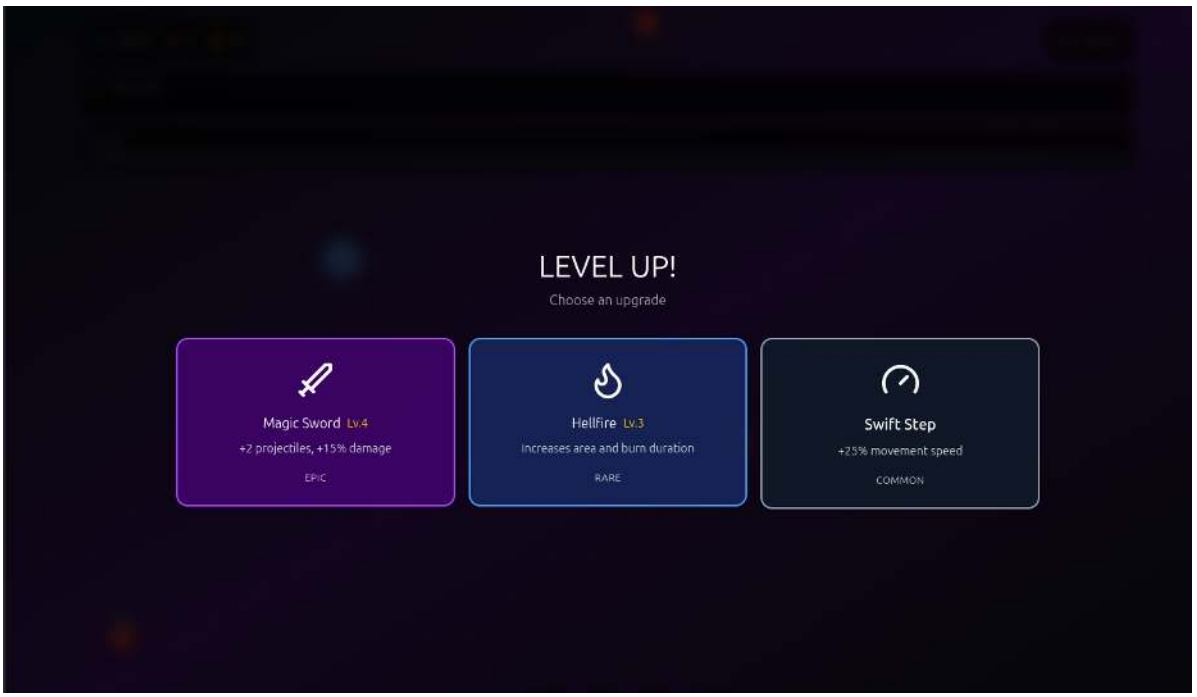
- ***Documento: `{trade_listing_id}`** (ID único de la oferta de intercambio)*
    - ***Campo:** `sellerUid` (string) - Referencia al ID del documento en la colección `players`.*
    - ***Campo:** `sellerName` (string)*
    - ***Campo:** `offeredItemInstanceId` (string) - Referencia al ID del documento en la sub-colección `inventory` del vendedor.*
    - ***Campo:** `offeredItemData` (map) - Copia de los datos del objeto para mostrar en la UI.*
        - ***Sub-campo:** `itemId` (string)*
        - ***Sub-campo:** `rarity` (string)*
    - ***Campo:** `desiredItemDescription` (string) - Ej: "Busco cualquier hacha rara."*
    - ***Campo:** `status` (string) - Ej: "active", "completed", "cancelled".*
    - ***Campo:** `createdAt` (timestamp)*

6. Diseño de la interfaz

El diseño del UI constituye un elemento fundamental para garantizar una experiencia de juego fluida y accesible. En este apartado se detalla la estructura, la estética y la funcionalidad de los diferentes menús y elementos visuales del sistema. El objetivo principal ha sido crear una navegación intuitiva que permita al jugador gestionar su inventario y realizar intercambios comerciales de manera eficiente, manteniendo siempre la coherencia visual con el estilo artístico del proyecto.

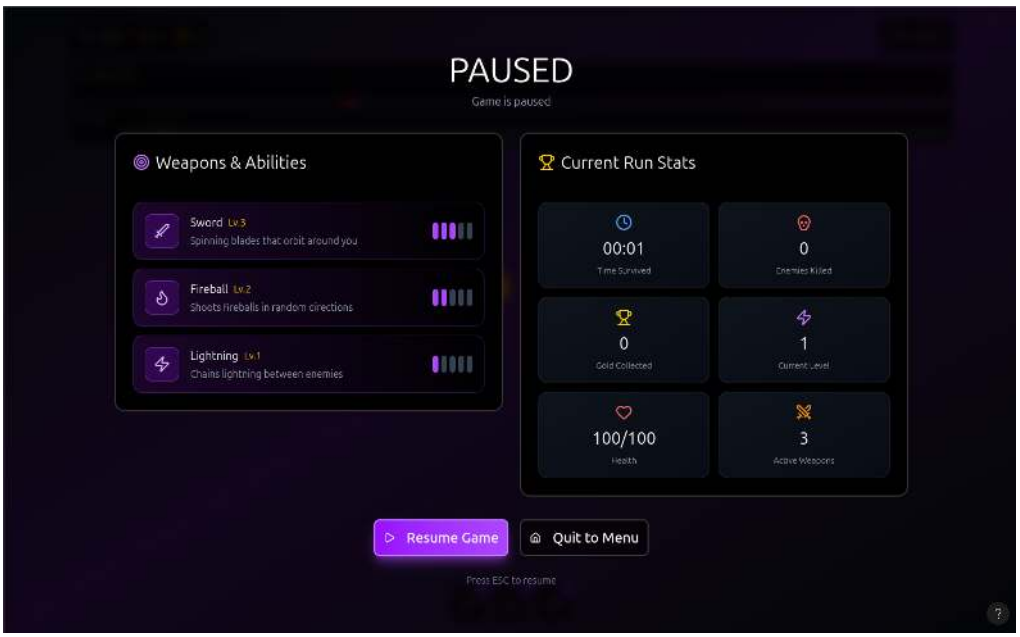
Ejemplo de pantalla de inicio (fuente: Halls of Torment, un juego creado el el mismo motor de juego, godot)

Otro ejemplo de pantalla de inicio diseñado en figma. Salen diferentes datos disponibles al jugador para seguir metricas que ha llegado



Pantalla de subida de nivel, se puede escoger entre diversas opciones que mejorarían el personaje en la misma partida.

Pantalla de inventario (en el caso del proyecto, solo habrán armas disponibles) del jugador, se pueden ver las diferentes estadísticas que tiene el jugador. (Halls of Torment)



Ejemplo de menú de pausa del jugador, se pueden ver las diferentes opciones y estadísticas que tiene el jugador.

Pantalla de fin de partida(fuente: Halls of Torment), se pueden ver las diferentes estadísticas que ha llegado el jugador. Esto se sincronizará en la base de datos.

7. Planificación técnica

Para llevar a cabo el desarrollo del proyecto de manera eficiente y organizada, se ha definido un ecosistema de herramientas y tecnologías que cubren desde la programación hasta la gestión del ciclo de vida del software.

Lenguajes y Frameworks: El desarrollo del núcleo del videojuego se realizará íntegramente en GDScript, que es el lenguaje de programación nativo al motor Godot, lo que facilita una sintaxis clara (similar a Python), un prototipado rápido de mecánicas y un rendimiento optimizado para la lógica de juegos 2D sin la sobrecarga de lenguajes más complejos.

Infraestructura de Datos (Backend): La persistencia de datos y los servicios en la nube se gestionarán a través de Firebase. Esta plataforma actuará como todo el backend, proporcionando la base de datos NoSQL (Firestore) necesaria para guardar inventarios y perfiles de usuario, así como los servicios de autenticación y funciones de servidor (Cloud Functions) para la lógica.

Herramientas de Diseño y Edición: La creación de assets multimedia se apoya en software estándar de la industria:

- **Gráficos:** Se utilizará la suite de Adobe (Photoshop/Illustrator) para el tratamiento y creación de imágenes, animaciones y sprites.
- **Prototipado UI/UX:** El diseño de interfaces, menús y la experiencia de usuario se maquetará previamente en Figma. Esto permite iterar sobre los mockups visuales antes de implementarlos en el motor.
- **Audio:** La edición de sonidos se realizará mediante aplicaciones de la suite de Adobe y Audacity.

Gestión y Organización: Para el seguimiento de las tareas y el control del progreso, se utilizará Trello. Esta herramienta permitirá aplicar una metodología ágil (tipo Kanban), organizando las funcionalidades en tarjetas y estados (To Do, Doing, Done) para visualizar el avance del desarrollo.

Cronograma: La planificación se ha plasmado en un diagrama de Gantt. Este mismo diagrama define las fases del proyecto, las dependencias entre tareas y los hitos de entrega clave, asegurando que el desarrollo se ajuste al plazo estimado de 6 a 8 meses.

8. Análisis de riesgos

8.1. Identificación de riesgos

- Complejidades en la integración con Firebase
- Proyecto ambicioso para el plazo
- Fallos de seguridad en el sistema de comercio
- Desbalance en el gameplay
- Pérdida de datos críticos

8.2. Valoración y respuesta

Una vez se ha encontrado un grupo de riesgos, hay que analizarlos más a fondo:

Riesgo	Probabilidad	Impacto	Plan de prevención o contingencia
Complejidad de la integración con Firebase	Media	Alta	Realizar una "prueba de concepto" en los primeros meses, creando un mini-proyecto para validar que el plugin de Godot-Firebase funciona correctamente con la autenticación y escritura en Firestore.
Alcance ambicioso	Alta	Alta	Seguir rigurosamente el cronograma de 7 meses con objetivos mensuales. Utilizar un tablero Kanban para visualizar y limitar el trabajo "en curso".
Fallos de seguridad en el comercio	Media	Alta	Toda la lógica de confirmación de intercambios se ejecutará exclusivamente en una Cloud Function.
Desbalance del gameplay	Alta	Media	Crear una hoja de cálculo para definir y ajustar las estadísticas de armas, enemigos y la curva de dificultad.
Pérdida de datos críticos	Baja	Alta	Utilizar Git y realizar push al repositorio de GitHub al final de cada sesión de trabajo. Activar las copias de seguridad automáticas de Cloud Firestore (Point-in-Time Recovery), aunque sea por un período corto.

9. Validación y criterios de éxito

Criterios de aceptación

Bucle de juego funcional

El juego debe ofrecer el bucle básico completo: el jugador se mueve, el personaje ataca automáticamente, aparecen oleadas de enemigos con dificultad creciente y el jugador recoge XP que le permite subir de nivel y elegir mejoras.

Preservación del progreso en la nube

Al terminar una partida, el inventario de objetos raros y los datos esenciales del jugador se deben guardar en Firebase (Authentication + Cloud Firestore) y cargarse correctamente en sesiones posteriores.

Comercio online seguro y funcional

Un jugador podrá publicar una oferta desde su inventario, otros jugadores podrán ver la oferta y aceptarla; la operación de intercambio se debe procesar de forma atómica en el servidor mediante una Cloud Function.

Interfaz y navegación completas

Todas las pantallas críticas (menú principal, HUD en partida, pausa, inventario, comercio y fin de partida) serán accesibles y operativas. Esto significa que los botones y flujos principales no deben dejar al usuario en estados inconsistentes y que las acciones básicas (iniciar partida, abrir inventario, publicar/aceptar oferta) se realizan sin errore

Pruebas previstas (funcionales, de usuario, de rendimiento).

- **Pruebas de la Cloud Function de comercio:** Se ejecutarán test específicos sobre la función que procesa intercambios para comprobar escenarios positivos (intercambio válido), negativos (oferta inválida, objeto ya vendido) y ataques comunes (intentos de duplicación).
- **Pruebas de aceptación de usuario:** En puntos clave se convocará a un pequeño grupo de testers externos para que realicen tareas concretas: sobrevivir X minutos, derrotar al jefe, recoger un objeto raro y ejecutar un intercambio. Se recopilará feedback sobre usabilidad, diversión y claridad, que servirá para ajustar balance y UI antes de cada entrega mayor.
- **Pruebas de rendimiento:** Se medirán FPS y uso de memoria en escenarios de carga elevada (muchos enemigos, proyectiles y efectos simultáneos).

Indicadores de calidad o resultados esperados.

- **Rendimiento (FPS objetivo):** El juego debe mantener unos 60fps en PC, esto se hará haciendo diferentes pruebas de FPS con diferentes tamaños de oleadas

- **Estabilidad (tasa de fallos aceptable):** La build final deberá de tener una tasa de bugs y errores mínimos, con una falta de fallos críticos que evitan el completado de una partida de unos 15 minutos.
- **Usabilidad (tasa de éxito de tareas básicas):** Un jugador nuevo debe poder completar tareas definidas (iniciar sesión, comenzar partida, realizar un intercambio) sin consultar documentación. Durante las pruebas de usuario al menos 2/3 de los testers deberían completar esas tareas sin ayuda significativa para que se considere exitoso.

10. Conclusión

A lo largo de este documento, se ha realizado un análisis exhaustivo que establece una base sólida para la construcción del videojuego. Las decisiones principales tomadas son las siguientes:

- **Funcionalidades del sistema:** Se ha definido que el proyecto consistirá en un videojuego de acción 2D estilo *Vampire Survivors*. El sistema integrará un bucle de combate y supervivencia contra oleadas de enemigos, un sistema de progresión basado en mejoras y la obtención de objetos raros, y un inventario persistente para el jugador. La característica central que diferencia el proyecto será un sistema de comercio online que permitirá a los jugadores intercambiar estos objetos, todo gestionado a través de un backend seguro.

- **Tecnologías a utilizar:** El desarrollo se llevará a cabo con el motor Godot 4 y el lenguaje GDScript. Para toda la infraestructura online, se utilizará la plataforma Firebase, utilizando Cloud Firestore para la base de datos de inventarios y ofertas, Firebase Authentication para la gestión de usuarios anónimos y Cloud Functions para garantizar transacciones seguras en el sistema de comercio. El control de versiones se gestionará con Git y GitHub.

- **Valor del proyecto:** El proyecto no solo ofrece una experiencia de juego atractiva y rejugable, sino que también introduce una innovación en el género al integrar una economía de jugador a través del comercio online. A nivel técnico, representa una oportunidad para aplicar y validar conocimientos en desarrollo de videojuegos, arquitectura de backend serverless y gestión de datos en tiempo real, culminando en un producto completo y funcional.

Los próximos pasos en la ejecución del proyecto darán inicio a la fase de implementación:

1. **Preparar el entorno de desarrollo:** Instalar las versiones requeridas de Godot Engine, las plantillas de exportación para PC y las herramientas online de comandos de Firebase.
2. **Crear la estructura inicial:** Configurar el repositorio en GitHub con las ramas principales (`main`, `develop`) y el archivo `.gitignore`. A la vez crear el proyecto en la consola de Firebase y definir el esquema inicial de las colecciones `players` y `trade_listings` en Cloud Firestore.
3. **Empezar la implementación:** Siguiendo el cronograma, comenzar el desarrollo del primer sprint, enfocado en el caso de uso prioritario: el bucle de juego principal (movimiento del jugador, combate básico y generación de enemigos).