Institut Puig Castellar
Ciclo Formativo: Desarrollo de Aplicaciones Multiplataforma
Grado Superior (CFGS)

Fish Collector: Idle

Videojuego incremental con servidor Firestore

Autor	Joel Lozano Barbancho
Tutor	Luis Elía
Curso	2025-2026
Fecha de entrega	17/05/2026

Proyecto intermodular - DAM2A

Resumen del proyecto

Fish Collector: Idle es un videojuego 2D de género *idle*, también conocido como incremental, desarrollado con Godot 4.6 y GDScript, que combina mecánicas de pesca casuales con funcionalidades multijugador en tiempo real. Los jugadores capturan peces mediante un sistema de clics, los coleccionan en un inventario, los venden para obtener monedas o los intercambian con otros usuarios a través de un mercado integrado, todo ello complementado por un chat global y un sistema de misiones y logros.

El backend del juego se sustenta sobre Firebase: Firebase Auth gestiona el registro e inicio de sesión, Firestore almacena el progreso de cada jugador, los mensajes del chat, la presencia en línea y las ofertas del mercado de intercambio. La integración con Godot se realiza mediante el plugin godot-firebase, gestionado a través de varios Autoloads especializados.

El juego está diseñado para ser ejecutado en diferentes plataformas, con exportación nativa tanto para Windows (x86_64) como para Android (arm64-v8a).

Palabras clave: *videojuego idle, pesca, Godot, Firebase, Firestore, multijugador, GDScript, Android, mercado de intercambio, chat en tiempo real*

Abstract

Fish Collector: Idle is a 2D idle video game developed with Godot 4.6 and GDScript, combining casual fishing mechanics with real-time multiplayer features. Players catch fish through a click-based system, manage an inventory, sell fish for coins, or trade them with other users through an integrated marketplace, all supported by a global chat and a missions and achievements system.

The game backend is built on Firebase: Firebase Auth handles user registration and login, while Firestore stores each player's progress, chat messages, online presence, and marketplace offers. Godot integrates with Firebase through the godot-firebase plugin, managed via multiple specialized Autoloads.

The game is designed to be played on different platforms, with native exports for both Windows (x86_64) and Android (arm64-v8a).

Keywords: *idle game, fishing, Godot, Firebase, Firestore, multiplayer, GDScript, Android, trade marketplace, real-time chat*

Índice

1. Presentación del proyecto

1.1 Introducción

Fish Collector: Idle es un videojuego 2D que combina la esencia del género *idle* con mecánicas de pesca y funcionalidades sociales en tiempo real. Los jugadores pueden capturar peces mediante un sistema de clics, coleccionarlos, venderlos para obtener monedas o intercambiarlos con otros usuarios, todo ello complementado con un chat global y un mercado común que fomentan la interacción comunitaria.

El desarrollo se ha llevado a cabo utilizando el motor Godot 4.6 y el lenguaje GDScript, herramientas open-source especialmente adecuadas para proyectos 2D orientados a dispositivos móviles. Para la gestión de datos, autenticación y comunicación en tiempo real se ha empleado Firebase, lo que permite ofrecer guardado en la nube, sincronización entre dispositivos y un sistema de chat con baja latencia.

1.2 Contexto

Los videojuegos se han consolidado como una de las formas de entretenimiento digital más extendidas en la actualidad. El género *idle*, también conocido como incremental, ha ido adquiriendo notable popularidad gracias a su diseño sencillo y su capacidad para ofrecer avances constantes incluso con una participación mínima del jugador.

Éste género se caracteriza por mecánicas simples que permiten progresar mediante acciones repetitivas o automatizadas, como la obtención incremental de recursos o la mejora gradual de habilidades. Su atractivo radica en la posibilidad de disfrutar de una experiencia relajada, accesible y compatible con sesiones de juego breves.

1.3 Justificación

Muchos juegos del género se centran exclusivamente en la obtención automática de recursos y la mejora continua, pero carecen de elementos que fomenten la interacción entre jugadores o la creación de una comunidad activa. Éste proyecto pretende cubrir ese vacío mediante la incorporación de un chat global y un mercado de intercambio entre usuarios.

Además, el proyecto tiene un valor formativo significativo: permite explorar tecnologías ampliamente utilizadas en el desarrollo de aplicaciones modernas, como Godot 4.6 y Firebase, integrando aspectos como el guardado en la nube, la autenticación de usuarios y la comunicación en tiempo real. También sirve como ejercicio de aprendizaje en arquitectura de software, diseño de interfaces y programación orientada a videojuegos.

1.4 Objetivos

Objetivos Generales

Se definen tres objetivos principales:

- Desarrollo del videojuego 2D, centrado en la pesca como mecánica principal, con exportación para Windows y Android, interfaz simple pero clara y diseño orientado a resolución movil (360x640).
- Implementación de un servidor conectado mediante Firebase, encargado de la autenticación de usuarios y el guardado en la nube.
- Integración de funcionalidades sociales dentro del juego: chat global en tiempo real, sistema de presencia online y mercado de intercambio de peces entre jugadores.

Objetivos Específicos

- Creación de un sistema de login y registro mediante Firebase Auth que permita sincronizar el progreso desde la nube.
- Implementación de un sistema de pesca basado en clics/taps, con peces clasificados por rareza y un minijuego de pesca.
- Implementación de un inventario donde los jugadores puedan almacenar peces capturados para coleccionismo, intercambios o venta.
- Implementación de un sistema de venta de peces que permita obtener monedas, y consumo de peces para recuperar stamina.
- Implementación de una tienda de objetos con mejoras disponibles que aceleran el ciclo de juego.
- Implementación de un sistema de misiones que incentive la progresión con objetivos y recompensas.
- Integración de un chat global en tiempo real sobre Firestore con reglas de seguridad que validan autenticación, longitud de mensaje y nombre de usuario.
- Implementación de un mercado de intercambio con validación de transacciones y actualización de estado de las ofertas.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

Fish Collector: Idle se plantea siguiendo una estrategia progresiva y modular que permite construir el videojuego desde sus elementos esenciales hasta las funcionalidades más avanzadas. La viabilidad técnica está respaldada por la elección de herramientas: Godot 4.6 facilita la creación de juegos 2D mediante GD Script, un lenguaje accesible de sintaxis similar a Python, y Firebase proporciona servicios de autenticación, base de datos en la nube y comunicación en tiempo real sin necesidad de gestionar infraestructura propia.

En términos de viabilidad temporal, el desarrollo se ha organizado en fases claras que permiten avanzar de forma ordenada. Finalmente, el proyecto es viable en términos de recursos ya que todas las herramientas utilizadas son gratuitas o de código abierto (Godot, Firebase Spark plan, plugin godot-firebase) y no requieren hardware especializado.

2.2 Metodología de trabajo

Para el desarrollo se ha adoptado una metodología ágil basada en un flujo Kanban, gestionado mediante la plataforma Taiga. Este enfoque permite organizar el trabajo de forma visual, flexible y adaptable, especialmente útil en proyectos individuales donde las prioridades pueden cambiar según el avance técnico.

https://tree.taiga.io/project/joellozanobarbancho-fishcollector-idle/

La metodología se basa en tres principios fundamentales:

- Visualización del trabajo: todas las tareas se representan mediante tarjetas en un tablero digital organizado en columnas (New, Ready, In Progress, Ready for Test, Done).
- Límite de trabajo en curso (WIP): solo se trabajan unas pocas tareas simultáneamente para evitar la dispersión y garantizar que cada funcionalidad se complete antes de iniciar la siguiente.
- Flujo continuo e incremental: cada tarea avanza por el tablero hasta completarse, lo que permite integrar, probar y mejorar el proyecto de forma continua.

Además del tablero Kanban, se han utilizado otras herramientas complementarias como Github, para el control de versiones y seguimiento del código, y Figma, para el diseño de la interfaz gráfica:

Este conjunto de herramientas, junto con la metodología ágil, permite un desarrollo ordenado, adaptable y orientado a resultados, asegurando que cada fase del proyecto avance de forma coherente y controlada.

2.3 Planificación

La planificación se estructura en fases de trabajo que se desarrollan de manera continua y adaptable:

Fase 1: Investigación y definición del proyecto

- Análisis de proyectos anteriores y referencias de juegos idle y de pesca.
- Identificación de mecánicas clave y del alcance del proyecto.
- Redacción de la propuesta inicial y recopilación de requisitos.

Fase 2: Diseño conceptual y técnico

- Diseño de la interfaz gráfica en Figma y definición del flujo de pantallas.
- Creación de diagramas de casos de uso, modelo de datos y arquitectura del sistema.
- Estructuración del modelo de datos para Firestore y definición de los Autoloads del proyecto.

Fase 3: Prototipado tecnico

- Configuración del proyecto en Godot 4.6 con resolución 360x640 y orientación vertical.
- Integración del plugin godot-firebase y pruebas iniciales de autenticación y base de datos.
- Implementación de un prototipo básico de pesca y UI.

Fase 4: Desarrollo del núcleo jugable

- Implementación completa de la mecánica de pesca y del minijuego asociado.
- Sistema de inventario y almacenamiento de peces.
- Sistema de venta de peces y economía básica.

Fase 5: Integración del servidor y funcionalidades sociales

- Sistema de login/registro con Firebase Auth y sincronización en Firestore.
- Chat global en tiempo real y presencia online.
- Mercado de intercambio y reglas de seguridad Firestore.

Fase 6: Misiones, logros y contenido adicional

- Implementación de un sistema de misiones y recompensas.
- Implementación de un sistema niveles mediantes experiencia obtenida al pescar para la progresión del jugador.
- Ajustes de balance y ampliación del contenido.

Fase 7: Pruebas y validación

- Pruebas funcionales de cada caso de uso en Windows y Android.
- Validación de las reglas de seguridad de Firestore.
- Corrección de errores y optimización final.

Fase 8: Documentación y entrega

- Redacción de la memoria del proyecto.
- Publicación de la release v1.0 en GitHub con ejecutables para Windows y Android.

3. Análisis

3.1 Casos de uso

Los casos de uso describen situaciones concretas que se pueden dar dentro del videojuego cuando un jugador interactúa con el sistema.

Caso de uso	Actor	Descripcion	Resultado esperado
Registrar usuario	Visitante	El visitante introduce sus datos para crear una cuenta nueva mediante Firebase Auth.	Usuario registrado correctamente
Iniciar sesión	Jugador	El jugador introduce sus credenciales, Firebase Auth verifica la identidad y se carga el progreso desde Firestore.	Acceso concedido y progreso cargado
Recuperación de contraseña	Jugador	El jugador solicita el restablecimiento de contraseña mediante correo electrónico a través de Firebase Auth.	Correo de recuperación enviado
Pescar pez	Jugador	El jugador interactúa con la escena principal mediante clics/taps, iniciando así el minijuego de pesca.	Captura de pez exitosa o fallida, dependiendo del resultado del minijuego de pesca
Consultar inventario	Jugador	El jugador accede al menú de inventario para ver y gestionar sus peces.	Inventario abierto y peces mostrados
Vender pez	Jugador	Desde el inventario el jugador vende peces para obtener monedas.	Monedas añadidas al balance del jugador
Comer pez	Jugador	Desde el inventario el jugador consume peces para recuperar stamina y poder seguir pescando.	Stamina recuperada
Comprar mejora	Jugador	El jugador gasta monedas en la tienda para adquirir mejoras.	Mejora aplicada al jugador
Cambiar de escena	Jugador	El jugador se desplaza a otras zonas de pesca a través del botón de cambiar de hábitat.	Nueva zona cargada con peces distintos
Publicar oferta	Jugador	El jugador publica una oferta de intercambio indicando los peces que ofrece y los que solicita a cambio.	Oferta visible en el mercado
Aceptar intercambio	Jugador	El jugador acepta una oferta activa y se actualizan los inventarios de ambos jugadores.	Transaccion completada
Hablar en el chat	Jugador	El jugador envía un mensaje por el chat general.	Mensaje visible para todos los jugadores conectados
Progreso de misiones	Jugador	Se actualiza el progreso de las misiones tras cada acción realizada por el jugador.	Misión completada y recompensa otorgada
Borrar datos de guardado	Jugador	El jugador clica en el botón de borrar datos de guardado	Archivo de guardado y cuenta eliminados de de Firestore y Firebase Auth
Cerrar juego	Jugador	El jugador clica en el botón de salir del juego	El juego se cierra y el save se sube a Firestore con éxito

3.2 Requisitos funcionales

1. El sistema permitirá registrar nuevos usuarios mediante correo electrónico y contraseña a través de Firebase Auth.
2. Los usuarios podrán iniciar sesión y cargar el progreso de su partida desde diferentes dispositivos a través de Firestore.
3. Los usuarios podrán solicitar un cambio de contraseña por correo electrónico desde el menú de inicio.
4. El jugador podrá pescar peces mediante clics/taps en la pantalla.
5. El sistema contará con un sistema de inventario que gestionará el almacenamiento y las operaciones sobre los peces del inventario.
6. El jugador podrá vender peces para obtener monedas o consumirlos para recuperar stamina.
7. El sistema incluirá un chat global en tiempo real y gestionará la presencia online de los jugadores.
8. El sistema gestionará misiones con objetivos y recompensas para guiar la progresión del jugador.
9. Los jugadores podrán publicar y aceptar ofertas de intercambio de peces a través de Firestore.
10. El sistema gestionará las mejoras adquiridas por el jugador y su efecto en las mecánicas de juego.
11. El sistema gestionará el nivel del jugador y los desbloqueos asociados a cada nivel.

3.3 Requisitos no funcionales

12. El juego deberá funcionar correctamente en Windows (x86_64) y Android (arm64-v8a) con una resolución base de 360x640 en orientación vertical.
13. El juego debe funcionar de forma estable con tiempos de respuesta adecuados tanto en la parte local como en las operaciones sobre Firestore.
14. La autenticación y la base de datos deben proteger la información de los usuarios mediante reglas de seguridad de Firestore que validan la identidad del usuario autenticado antes de cualquier operación de escritura.
15. El chat y el mercado deben funcionar con baja latencia, aprovechando los listeners en tiempo real de Firestore.
16. La interfaz debe ser intuitiva y clara, especialmente para jugadores casuales, con un tema visual coherente.
17. El sistema debe poder soportar un número razonable de usuarios concurrentes sin degradar el rendimiento, dentro de los límites del plan gratuito de Firebase.

3.4 Análisis de alternativas tecnológicas

Motor de juego

Se analizaron tres alternativas principales:

- **Unity (C#)**: Gran potencia y amplia comunidad, pero mayor curva de aprendizaje y complejidad superior para un proyecto 2D acotado.
- **Unreal Engine**: Excesivo para un juego 2D idle; orientado a proyectos 3D de gran escala.
- **Godot 4.6 (GDScript)**: Motor open-source, ligero, orientado al 2D, con un lenguaje de scripting accesible y compilación para Android y Windows sin licencias adicionales.

Se ha elegido Godot 4.6 por su adecuación al tipo de juego, su carácter open-source y su menor sobrecarga para un proyecto de estas dimensiones.

Servidor y base de datos

- **Base de datos relacional (MySQL/PostgreSQL) + servidor propio:** Gran control pero requiere configurar y mantener servidores, diseñar una API propia y gestionar la infraestructura.
- **Servicios backend tradicionales (Node.js, Spring Boot):** Permiten un backend a medida pero implican desarrollar desde cero la autenticación y la comunicación en tiempo real.
- **Firebase (Auth + Firestore):** Proporciona autenticación, base de datos NoSQL en la nube y sincronización en tiempo real sin necesidad de administrar servidores, con integración directa desde Godot mediante el plugin godot-firebase.

Se ha elegido Firebase porque simplifica enormemente la gestión del backend, con un plan gratuito (Spark) suficiente para el alcance del proyecto. La combinación Godot 4.6 + Firebase ofrece un equilibrio adecuado entre potencia, simplicidad y viabilidad.

4. Diseño

4.1 Arquitectura del sistema

La arquitectura del sistema sigue un modelo cliente-servidor donde el cliente es el videojuego desarrollado en Godot 4.6 y el servidor es la plataforma Firebase. La lógica del juego reside en el cliente, mientras que los datos persistentes y la comunicación entre usuarios se gestionan en la nube.

La capa de Godot se organiza en Autoloads (singletons globales) que encapsulan cada responsabilidad del sistema. Todos los Autoloads están declarados en el project.godot y son accesibles desde cualquier escena:

Autoload	Descripción y responsabilidad
Firebase.gd	Plugin godot-firebase. Punto de entrada principal para todas las llamadas a Firebase Auth y Firestore desde GDScript.
FirebaseManager.gd	Gestiona toda la comunicación con Firebase via HTTP REST. Cubre: autenticación, subida/descarga de partida a Firestore, chat global, presencia online y mercado de intercambio.
DataManager.gd	Carga y expone las bases de datos estáticas del juego. También contiene toda la lógica de consulta: selección aleatoria de peces por zona con pesos de rareza, multiplicadores por localización y acceso por ID a peces, niveles y localizaciones.
InventoryManager.gd	Gestiona el inventario de peces: adición, eliminación, venta y consulta de los peces del jugador, con sincronización a Firestore.
UpgradeManager.gd	Gestiona las mejoras adquiridas por el jugador y calcula sus efectos sobre las mecánicas (probabilidad de rareza, velocidad de recuperación, etc.).
QuestManager.gd	Controla el estado de las misiones activas, verifica el progreso tras cada acción relevante y otorga recompensas al completarlas.
LevelManager.gd	Gestiona la progresión de nivel del jugador y los desbloqueos asociados.
WindowManager.gd	Fuerza la ventana a modo WINDOWED sin redimensionar, bloquea el cambio a pantalla completa y fija la orientación de pantalla en portait.
Data.gd	Almacena temporalmente los datos de partida cargados desde Firestore y sirve como punto de acceso compartido entre los demás Autoloads durante la sesión.
File.gd	Gestiona la persistencia local de la partida. Crea los archivos de guardado con los valores por defecto del jugador, guarda y carga las partidas.

Scripts adicionales

Además de los Autoloads, el proyecto cuenta con otros scripts que implementan la lógica de escenas concretas y componentes visuales reutilizables:

Script	Descripción y responsabilidad
MainMenu.gd	Menú principal del juego. Contiene tres botones: navegar a la pantalla de login, navegar a la pantalla de registro y salir del juego.
RegisterScreen.gd	Pantalla de registro de nueva cuenta. Valida que todos los campos estén llenos, que el email tenga formato correcto y que las contraseñas coinciden. Guarda el nombre de usuario en Data.save_data antes de llamar a FirebaseManager.register().
LoginScreen.gd	Pantalla de inicio de sesión. Valida el formato del email, llama a FirebaseManager.login() y redirige a la pantalla principal del juego si el login es correcto. Gestiona errores de autenticación y el flujo de recuperación de contraseña via email.
FishingScreen.gd	Escena principal del juego. Orquesta la mecánica de pesca (cooldown, coste de estamina, cálculo de rareza), el inventario, la tienda, el chat, el mercado, las misiones, la fishpedia y el cambio de zona. Gestiona el minijuego SkillCheck y los popups de resultado.
SkillCheck.gd	Minijuego de pesca. Controla un temporizador de 5 segundos, el ángulo rotatorio del indicador y una zona de éxito aleatoria. Al hacer clic compara el ángulo actual con la zona verde: si coincide el intento es exitoso.
SkillCheckCanvas.gd	Componente visual del minijuego. Dibuja el circulo de fondo, el arco verde de la zona de exito y el indicador amarillo rotatorio.
FishSwimmer.gd	Componente decorativo de los fondos de pesca. Mueve peces y cangrejos horizontalmente por la pantalla, rebotando en los bordes. Los peces aplican un movimiento sinusoidal vertical; los cangrejos se desplazan solo en horizontal a nivel del suelo.
CooldownOrb.gd	Widget visual de recarga del tiempo de pesca. Dibuja un círculo con un sector relleno que representa el progreso del cooldown entre intentos de pesca.

El flujo general de datos es el siguiente: el jugador inicia sesión a través de FirebaseManager, que autentica al usuario con Firebase Auth y carga su perfil desde la coleccion *user_saves* de Firestore en DataManager. Durante la partida, los cambios de estado se sincronizan con Firestore a través de los Autoloads correspondientes. El chat y el mercado operan con listeners de Firestore que notifican los cambios en tiempo real a todos los clientes conectados.

4.2 Modelo de datos

El modelo de datos está estructurado en múltiples colecciones de Firestore. Las reglas de seguridad controlan el acceso a cada colección validando la autenticación del usuario y la integridad de los datos.

Colección: *user_saves*

Cada documento representa el estado guardado de un jugador. El ID del documento es el UID de Firebase Auth del usuario, garantizando que solo el propio usuario puede leer y modificar su partida.

Campo	Tipo y descripción
uid	**string** - Identificador único del usuario (Firebase Auth UID).
player_name	**string** - Nombre de usuario visible en el juego y en el chat (max. 32 caracteres).
coins	**int** - Monedas actuales del jugador.
stamina	**int** - Estamina disponible para pescar.
level	**int** - Nivel actual del jugador, gestionado por LevelManager.
xp	**int** - Experiencia acumulada hacia el siguiente nivel.
upgrades	**map** - Mejoras adquiridas y su nivel, gestionado por UpgradeManager.
inventory	**array/map** - Peces almacenados, gestionado por InventoryManager.
quests	**map** - Estado de las misiones activas y completadas, gestionado por QuestManager.
last_saved	**timestamp** - Fecha y hora del último guardado en la nube.

Colección: *global_chat_messages*

Almacena los mensajes del chat global. Las reglas de Firestore validan que el UID del mensaje coincida con el usuario autenticado, que el mensaje tenga entre 1 y 180 caracteres, y que el *player_name* tenga entre 1 y 32 caracteres. Los clientes no pueden eliminar mensajes.

Campo	Tipo y descripción
uid	**string** - UID del jugador remitente (debe coincidir con request.auth.uid).
player_name	**string** - Nombre visible del remitente (1-32 caracteres).
message	**string** - Contenido del mensaje (1-180 caracteres).
created_at_unix	**int** - Timestamp Unix de envío del mensaje.

Colección: *online_presence*

Registra la presencia en línea de cada jugador mediante un mecanismo de heartbeat. El ID del documento es el UID del usuario. Las reglas validan que solo el propio usuario puede actualizar su presencia.

Campo	Tipo y descripción
uid	**string** - UID del jugador (debe coincidir con request.auth.uid).
player_name	**string** - Nombre visible del jugador (1-32 caracteres).
last_seen_unix	**int** - Timestamp Unix de la última actualización del heartbeat.

Colección: *trade_offers*

Almacena las ofertas del mercado de intercambio. Las reglas de Firestore validan que la oferta sea creada por el usuario autenticado, que los campos *offering_fish* y *wanted_fish* sean listas, y que el estado inicial sea 'active'. Las actualizaciones permiten que el propio vendedor modifique su oferta, o que otro usuario autenticado cambie el estado a 'completed' para aceptar el intercambio.

Campo	Tipo y descripción
player_uid	**string** - UID del jugador que publica la oferta (debe coincidir con request.auth.uid en la creación).
player_name	**string** - Nombre del vendedor.
offering_fish	**list** - Lista de peces ofrecidos en el intercambio.
wanted_fish	**list** - Lista de peces solicitados a cambio.
status	**string** - Estado de la oferta: 'active' o 'completed'.
created_at_unix	**int** - Timestamp Unix de publicacion de la oferta.
updated_at_unix	**int** - Timestamp Unix de la última actualización (requerido en updates).

4.3 Diseño de interfaz

El diseño de la interfaz se ha realizado en Figma, siguiendo las restricciones de resolución 360x640 en orientación vertical. El esquema de color principal utiliza tonos azules y oscuros con bordes dorados/blancos, adecuados para una estética de videojuego de pesca.

La navegación entre pantallas se gestiona mediante cambios directos de escena con get_tree().change_scene_to_file(). La pantalla principal (FishingScreen) no cambia de escena durante el juego: todos los paneles funcionales (tienda, inventario, chat, misiones, mercado, opciones, fishpedia) son *dropdowns* o desplegables que se muestran y ocultan sobre la escena de pesca. La barra de navegación inferior contiene cinco botones: Store, Inventory, Quests, Social y Settings.

A continuación se describen en detalle las pantallas y paneles principales del juego:

Menú principal (MainMenu.tscn)

Pantalla de entrada al juego. Contiene un fondo decorativo y tres botones centrados verticalmente: Login, Register y Exit Game. No tiene lógica de juego: cada botón navega a la escena correspondiente o sale de la aplicación.

Pantalla de login (Login.tscn)

Formulario de inicio de sesión con dos campos de texto (Email y Password, este último en modo secreto), botón de Login, botón de vuelta al menú principal y botón “Forgot your password?” que se encarga de enviar un correo de recuperación de contraseña a la dirección especificada. También Incluye un popup de error sobre un CanvasLayer que muestra mensajes temporales o esperando clic del usuario. La lógica valida el formato del email antes de llamar a FirebaseManager.login() y gestiona los errores de autenticación (cuenta inexistente, credenciales inválidas, etc).

Pantalla de registro (Register.tscn)

Formulario de creación de cuenta con cuatro campos: Email, Username, Password y Confirm Password. El botón principal es “Create Account”. Al igual que el login, incluye el mismo sistema de popup de error. La lógica valida que todos los campos estén rellenos, que el email sea válido y que las contraseñas coincidan antes de llamar a FirebaseManager.register().

Pantalla principal de pesca (FishingScreen.tscn)

Ésta es la pantalla más importante. Se trata de la escena principal, que contiene todos los elementos de UI y de juego. Su estructura en la escena es la siguiente:

- **BackgroundLayer (CanvasLayer):** Contiene los sprites de fondo (uno para el hábitat de río y otro para el de mar) y las capas de peces decorativos. Solo uno de los fondos es visible en cada momento, dependiendo del hábitat seleccionado.
- **TopHud (VBoxContainer):** Barra superior con el contador de monedas y la barra de stamina, con color dinámico según el nivel. También muestra el nivel del jugador.
- **CatchArea (Control):** Zona invisible sobre el fondo de agua donde el jugador hace clic o toca para iniciar la pesca. Activada sólo cuando no hay ningún dropdown abierto.
- **CooldownOrb (Control):** Widget circular personalizado que se dibuja sobre el CatchArea y muestra visualmente el tiempo de recarga entre intentos de pesca.
- **ChangeSpotButton (Button):** Botón flotante para cambiar de hábitat. Se oculta automáticamente cuando cualquier menú desplegable está abierto.
- **HabitatPopupLayer (CanvasLayer):** Popup modal para seleccionar la zona de pesca activa (River o Sea).
- **BottomNavBar (Panel):** Barra de navegación inferior con cinco botones cada uno asociado a los diferentes menús desplegables.
- **StoreDropdown, InventoryDropdown, SocialDropdown, TradeDropdown, QuestDropdown, SettingsDropdown, FishpediaDropdown:** Paneles desplegables que se muestran sobre la escena al pulsar el botón correspondiente del BottomNavBar. Solo uno puede estar visible a la vez.
- **MessagePopupLayer (CanvasLayer):** Popup temporal centrado que muestra el resultado de cada intento de pesca, con el nombre y sprite del pez capturado.

- **RewardPopupLayer (CanvasLayer):** Popup de recompensa que aparece al completar una misión, mostrando el nombre y la recompensa obtenida durante unos segundos.

Minijuego de pesca (SkillCheck.tscn)

Escena modal instanciada en tiempo de ejecución cuando el intento de pesca tiene éxito en la tirada de probabilidad. El jugador tiene 5 segundos para hacer clic cuando el indicador amarillo se encuentre dentro del arco verde. El resultado se comunica a FishingScreen mediante la señal skill_check_completed.

Tienda de mejoras (StoreDropdown)

Panel desplegable que contiene un ScrollContainer con la lista de mejoras disponibles. Cada ítem muestra el nombre, nivel actual/máximo, descripción y coste del siguiente nivel. El botón Buy deduce el coste y aplica la mejora vía UpgradeManager; si el ítem está al máximo nivel muestra MAX deshabilitado.

Inventario (InventoryDropdown)

Panel desplegable con un GridContainer de tarjetas de pez. Cada tarjeta muestra el sprite del pez, el nombre con la cantidad acumulada, y cuatro botones: SELL (vender uno), SELL ALL (vender todos), EAT (consumir uno para recuperar stamina) y EAT ALL (consumir todos).

Social y Chat global (SocialDropdown)

Panel desplegable que contiene el chat general, con scroll automático, y un contador de jugadores online. Los mensajes se renderizan con timestamp [HH:MM] y nombre de usuario coloreado de forma única por hash. El panel hace polling cada 2 segundos a Firestore para actualizar mensajes y presencia. También contiene dos botones de acción que sirven para navegar hacia el mercado de intercambio y para publicar ofertas de intercambio.

Mercado de intercambio (TradeDropdown)

Panel desplegable con una lista de ofertas activas. Cada oferta muestra el nombre de la persona que la crea, los peces ofrecidos y solicitados con sus sprites y cantidades, y un botón Accept. Si la oferta es del jugador actual, el botón muestra “Your offer” y está deshabilitado. Al abrir el panel se comprueba automáticamente si hay recompensas pendientes de reclamar de intercambios completados.

Publicar oferta (PostOfferPopupLayer)

Popup modal generado en tiempo de ejecución por código (no es una escena independiente). Muestra dos columnas de selección: pez a ofrecer (limitado al inventario actual) y pez a solicitar (cualquier pez descubierto). Cada columna tiene botones de navegación (< y >) y de cantidad (+ y -). El botón Publish envía la oferta a Firestore y retira los peces ofrecidos del inventario. Cuando un jugador acepta una oferta de intercambio se lleba a cabo la transacción y ambos jugadores obtienen los peces que les corresponden.

Misiones (QuestDropdown)

Panel desplegable con una lista de misiones de dos quest packs disponibles. Cada entrada muestra el nombre, descripción, barra de progreso y recompensa. Las misiones del pack 2 aparecen bloqueadas (con icono de candado y atenuadas en gris) hasta completar la misión desbloqueadora. Las misiones completadas muestran la barra en verde y la etiqueta DONE.

Ajustes (SettingsDropdown)

Panel desplegable con tres opciones: Fishpedia (abre el FishpediaDropdown), Exit game (confirma y sale del juego) y Reset Data (confirma y borra todos los datos del usuario en Firestore y la cuenta de Firebase Auth). Las acciones de salida y borrado requieren confirmación mediante un diálogo modal generado por código.

Fishpedia (FishpediaDropdown)

Panel desplegable con una lista de todos los peces de la base de datos. Cada entrada muestra el sprite del pez (en versión ennegrecida si aún no se ha descubierto), el nombre, su hábitat, la descripción y el total de capturas.

5. Desarrollo

5.1 Estructura del proyecto

El repositorio del proyecto está organizado según la estructura estándar de un proyecto Godot 4.6. Los elementos principales son:

Directorio / Archivo	Contenido
Scripts/	Scripts GDScript (.gd) de todos los Autoloads y lógica de escenas.
Scenes/	Escenas Godot (.tscn) de cada pantalla del juego.
Assets/	Recursos gráficos: sprites, fondos y otros elementos.
Database/	Archivos de datos estáticos del juego en formato JSON: peces, niveles, mejoras, misiones y hábitats
Themes/	Tema visual del juego.
project.godot	Configuración del proyecto.
addons/godot-firebase/	Plugin godot-firebase para la integración con Firebase Auth y Firestore desde GDScript.
export_presets.cfg	Configuración de exportación para Windows Desktop (x86_64) y Android (arm64-v8a).
FIRESTORE_RULES_CHAT.txt	Reglas de seguridad de Firestore que protegen las colecciones del juego.
images/	Imágenes utilizadas para la página web / landing.
index.html	Código fuente de la página web / landing

5.2 Implementación de funcionalidades

Sistema de autenticación (FirebaseManager)

La autenticación se gestiona mediante el Autoload FirebaseManager, que encapsula las llamadas a Firebase Auth a través del plugin godot-firebase. Al iniciar sesión, el manager obtiene el token de autenticación del usuario y lanza la carga del perfil desde Firestore. Al registrar un nuevo usuario se crea el documento inicial en la colección *user_saves* con los valores por defecto.

Mecanica de pesca y minijuego

El sistema de pesca se activa cuando el jugador toca o hace clic en la zona de agua de la escena principal. La probabilidad de rareza del pez se calcula en función de tablas de datos estáticas almacenadas en la carpeta Database/ y de los modificadores de las mejoras activas gestionadas por UpgradeManager. A continuación se lanza el minijuego: el jugador deberá acertar en el momento justo en la zona verde del minijuego. Si el jugador tiene éxito el pez será añadido al inventario, de lo contrario la pesca habrá sido un fracaso.

Inventario y economía (InventoryManager / UpgradeManager)

El InventoryManager gestiona en memoria el inventario del jugador y lo sincroniza con la colección *user_saves* de Firestore. Desde el inventario el jugador puede vender peces (obtener monedas según su rareza) o consumirlos (recuperar stamina). El UpgradeManager gestiona las mejoras adquiridas y aplica sus efectos sobre las mecánicas del juego, como el aumento de probabilidad de peces raros o el multiplicador de monedas.

Progresion del jugador (LevelManager / QuestManager)

El LevelManager gestiona el nivel del jugador y los desbloqueos asociados a cada nivel. El QuestManager controla el estado de las misiones activas, verifica el progreso tras cada acción relevante (pesca, venta, intercambio) y otorga recompensas al completar objetivos.

Chat global en tiempo real

El chat global utiliza un listener de Firestore sobre la colección *global_chat_messages* para recibir mensajes en tiempo real. Las reglas de seguridad de Firestore validan que el uid del mensaje coincida con el usuario autenticado, que el mensaje tenga entre 1 y 180 caracteres, y que el *player_name* tenga entre 1 y 32 caracteres. Los clientes no pueden eliminar mensajes. El sistema también mantiene la colección *online_presence* con un mecanismo de heartbeat para mostrar quienes están conectados.

Mercado de intercambio

El mercado permite publicar ofertas con peces del inventario a través de la colección *trade_offers* de Firestore. Las reglas de seguridad validan que la oferta sea creada por el usuario autenticado, con los campos *offering_fish* y *wanted_fish* como listas y estado inicial 'active'. Al aceptar una oferta, el estado se actualiza a 'completed' y se actualizan los inventarios de ambos jugadores. El acceso al mercado está protegido: solo el vendedor puede modificar su oferta, y cualquier otro usuario autenticado puede cambiar el estado a 'completed' para aceptar el intercambio.

Exportación multiplataforma

El proyecto está configurado para exportar a dos plataformas mediante el export_presets.cfg:

- Windows Desktop (x86_64): genera el ejecutable FishCollectorIdle.exe. Incluye todos los recursos, archivos .env, .png y .json.
- Android (arm64-v8a): genera el APK FishCollectorIdle.apk. Requiere los permisos ACCESS_NETWORK_STATE e INTERNET para la comunicación con Firebase. La app soporta orientación portrait con modo inmersivo activado.

5.3 Pruebas

Pruebas funcionales

Se ha verificado cada caso de uso definido en el análisis, comprobando el correcto funcionamiento del sistema ante entradas válidas e inválidas. Los aspectos probados incluyen: el flujo completo de registro e inicio de sesión con Firebase Auth, la captura y almacenamiento de peces con InventoryManager, las transacciones de venta y compra, el envío y recepción de mensajes de chat respetando los límites de las reglas de Firestore, y la publicación y aceptación de ofertas en el mercado.

Pruebas de reglas de seguridad de Firestore

Se han validado las reglas de seguridad de Firestore, verificando que: un usuario no autenticado no puede leer ni escribir en ninguna colección, un usuario autenticado solo puede modificar su propia entrada en *user_saves* y *online_presence*, los mensajes de chat son rechazados si superan los 180 caracteres o si el uid no coincide con el usuario autenticado, y las ofertas de trade solo pueden ser creadas con estado 'active' y aceptadas por usuarios distintos al vendedor.

Pruebas en dispositivos

El juego ha sido probado en las dos plataformas objetivo: en Windows mediante el ejecutable generado con el preset Windows Desktop, y en Android mediante el APK instalado en dispositivo real, verificando la correcta adaptación de la interfaz a la resolución del dispositivo y el funcionamiento de las comunicaciones con Firebase.

Pruebas de usabilidad

Se han realizado sesiones de prueba con usuarios no familiarizados con el juego para evaluar la curva de aprendizaje y la claridad de la interfaz. Los principales hallazgos han derivado en ajustes como el tamaño de los elementos táctiles para dispositivos móviles y la selección de colores más llamativos.

6. Conclusiones

6.1 Conclusiones generales

El desarrollo de Fish Collector: Idle ha resultado en un videojuego 2D funcional que cumple con la propuesta inicial de combinar mecánicas *idle* de pesca con funcionalidades sociales en tiempo real. El resultado es una aplicación jugable y estable con un ciclo de juego completo, integración de Firebase y funcionalidades sociales operativas, publicada en su versión v1.0 para Windows y Android.

La integración de Godot 4.6 con Firebase mediante el plugin godot-firebase ha demostrado ser una combinación técnica viable para proyectos de esta escala, permitiendo implementar características avanzadas como la sincronización en la nube, el chat en tiempo real y el mercado de intercambio sin necesidad de gestionar infraestructura de servidor propia.

6.2 Consecución de objetivos

Los tres objetivos generales han sido alcanzados:

- El videojuego 2D ha sido desarrollado con éxito, con mecánicas de pesca funcionales, interfaz clara para resolución movil 360x640 y exportación para Windows y Android.
- El servidor Firebase ha sido integrado correctamente, con autenticación, guardado en la nube y comunicación en tiempo real.
- Las funcionalidades sociales han sido implementadas: chat global, presencia online y mercado de intercambio, con reglas de seguridad de Firestore que protegen la integridad de los datos.

Respecto a los objetivos específicos, todos han sido implementados. El QuestManager y el LevelManager cubren el sistema de misiones y progresión. Las reglas de seguridad de Firestore garantizan la integridad de los datos tanto en el chat como en el mercado.

6.3 Valoración de la metodologia y planificacion

La metodología Kanban ha sido especialmente adecuada para este proyecto individual, permitiendo una gestión flexible de las tareas y una adaptación eficaz a los imprevistos técnicos.

La planificación por fases ha facilitado un desarrollo ordenado y progresivo. La fase de integración del servidor y de implementación de las reglas de seguridad de Firestore ha requerido más tiempo del inicialmente estimado, especialmente el diseño de las reglas para *trade_offers* que permiten que un usuario distinto al vendedor pueda aceptar la oferta de forma segura.

6.4 Vision de futuro

Como posibles mejoras y ampliaciones del proyecto se identifican las siguientes líneas de trabajo:

- Publicación de la app en plataformas gratuitas como itch.io
- Ampliación del catálogo de peces, zonas de pesca y tipos de mejoras para ofrecer mayor variedad.
- Implementación de un sistema de amigos y rankings globales entre jugadores.
- Optimización del modo offline para que el jugador pueda seguir progresando sin conexión, sincronizando los cambios al reconectarse.
- Incorporación de notificaciones push mediante Firebase Cloud Messaging (FCM) para alertar sobre nuevas ofertas en el mercado.
- Migración al plan Blaze de Firebase si el crecimiento de usuarios supera los límites del plan Spark (gratuito).

7. Glosario

Términos	Definición
Idle game	Género de videojuego incremental con mecánicas de progresión automática o semiautomática con mínima intervención del jugador.
GDScript	Lenguaje de programación propio del motor Godot, con sintaxis similar a Python, orientado al desarrollo de videojuegos.
Godot 4.6	Motor de videojuegos open-source utilizado en este proyecto, con soporte para GL Compatibility renderer y exportación a Android y Windows.
GL Compatibility	Modo de renderizado de Godot basado en OpenGL ES 3.0, compatible con la mayoría de dispositivos Android.
godot-firebase	Plugin open-source que integra Firebase SDK en proyectos Godot, permitiendo llamadas a Auth y Firestore desde GDScript.
Firebase	Plataforma de desarrollo de aplicaciones de Google con servicios de autenticación, base de datos en la nube y mensajería en tiempo real.
Firestore	Base de datos NoSQL en la nube de Firebase que almacena datos en documentos organizados en colecciones, con soporte para listeners en tiempo real.
Firebase Auth	Servicio de Firebase para la autenticación de usuarios mediante correo/contraseña u otros proveedores de identidad.
Kanban	Metodología de gestión visual del trabajo que organiza las tareas en columnas según su estado.
Listener	Mecanismo de escucha de eventos en tiempo real de Firestore que notifica al cliente cuando se producen cambios en la base de datos.
WIP	Work In Progress. Límite máximo de tareas trabajadas simultáneamente en metodologías ágiles Kanban.
APK	Android Package. Formato de distribución de aplicaciones Android. La release v1.0 incluye FishCollectorIdle.apk para arm64-v8a.

8. Bibliografia

Documentacion oficial

- Godot Engine Documentation. (2024). Godot 4 Documentation. Recuperado de https://docs.godotengine.org
- Google Firebase. (2024). Firebase Documentation: Firestore, Authentication. https://firebase.google.com/docs
- Godot Firebase Plugin. (2024). godot-firebase GitHub repository. Recuperado de https://github.com/GodotNuts/GodotFirebase

Herramientas y metodología

- Atlassian. (2024). Kanban: A brief introduction. https://www.atlassian.com/agile/kanban
- Taiga. (2024). Taiga Project Management. https://taiga.io
- Figma. (2024). Figma Design Documentation. https://help.figma.com

Vídeos útiles de referencia

- How to make a Video Game - Godot Beginner Tutorial https://www.youtube.com/watch?v=LOhfqjmasi0&t=34s
- Basic Idle Game ¦ Godot [Tutorial] https://www.youtube.com/watch?v=VxQT5FNIIfs

Referencias de videojuegos idle

- Cookie Clicker (2013). Orteil. https://orteil.dashnet.org/cookieclicker/
- Idle Slayer (2020). Pablo Leban. https://idleslayer.com/
- Fish to Dish: Idle Sushi (2025). Kygua Tech https://store.steampowered.com/app/3399960/Fish_to_Dish_Idle_Sushi/

Repositorio del proyecto

https://github.com/joellozanobarbancho/FishCollectorIdle

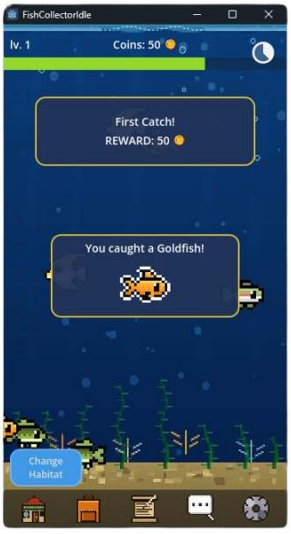
9. Anexos

Anexo A: Capturas de pantalla del juego

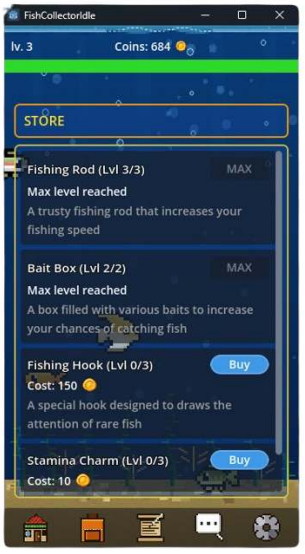
En este anexo se incluyen capturas de pantalla de las principales pantallas del juego: pantalla de login, escena de pesca, minijuego, inventario, tienda de mejoras, mercado de intercambio y chat global.

Pantalla principal de pesca

Minijuego de pesca (Skill Check)



Captura exitosa



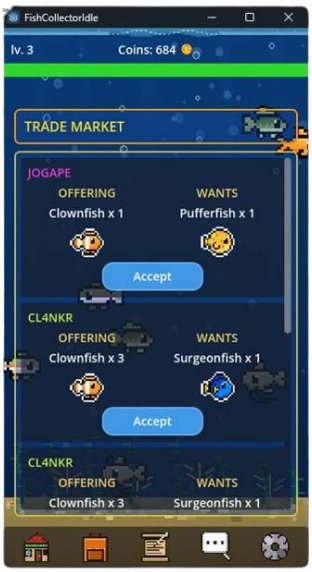
Menú de mejoras

Inventario

Misiones

Chat global

Publicar oferta de intercambio



Mercado de intercambio

Fishpedia

Anexo B: Reglas de seguridad de Firestore

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

      // Existing save document: only owner can read/write
      match /user_saves/{userId} {
        allow read, write: if request.auth != null && request.auth.uid ==
userId;
      }

      // Global chat messages for authenticated players
      match /global_chat_messages/{messageId} {
        allow read: if request.auth != null;

        allow create, update: if request.auth != null
             && request.resource.data.uid == request.auth.uid
             && request.resource.data.message is string
             && request.resource.data.message.size() > 0
             && request.resource.data.message.size() <= 180
             && request.resource.data.player_name is string
             && request.resource.data.player_name.size() > 0
             && request.resource.data.player_name.size() <= 32
             && request.resource.data.created_at_unix is int;

        // Optional: block delete from client
        allow delete: if false;
      }

      // Presence heartbeat per player (doc id should be uid)
      match /online_presence/{uid} {
        allow read: if request.auth != null;
        allow create, update: if request.auth != null
             && uid == request.auth.uid
             && request.resource.data.uid == request.auth.uid
```

```
            && request.resource.data.player_name is string
            && request.resource.data.player_name.size() > 0
            && request.resource.data.player_name.size() <= 32
            && request.resource.data.last_seen_unix is int;
        allow delete: if false;
      }

      // Trade offers marketplace
      match /trade_offers/{offerId} {
        allow read: if request.auth != null;

        allow create: if request.auth != null
            && request.resource.data.player_uid == request.auth.uid
            && request.resource.data.offering_fish is list
            && request.resource.data.wanted_fish is list
            && request.resource.data.player_name is string
            && request.resource.data.player_name.size() > 0
            && request.resource.data.status == "active"
            && request.resource.data.created_at_unix is int;

        allow update: if request.auth != null
            && request.resource.data.updated_at_unix is int
            && (
                  (resource.data.player_uid == request.auth.uid)
                  || (
                        resource.data.player_uid != request.auth.uid
                        && resource.data.status == "active"
                        && request.resource.data.status == "completed"
                  )
            );

        allow delete: if false;
      }
    }
}
```

Nota sobre el uso de inteligencia artificial

Durante el desarrollo de este proyecto se han utilizado herramientas de inteligencia artificial como apoyo en determinadas tareas del proceso, entre ellas: la consulta de dudas técnicas sobre la API de Firebase y GDScript, la revisión ortográfica y de estilo de la documentación, y la generación de sugerencias de estructura para el modelo de datos y las reglas de seguridad de Firestore.

En ningún caso el uso de estas herramientas ha sustituido el proceso de comprensión, análisis y toma de decisiones del autor. Todo el contenido ha sido revisado, verificado y asumido como propio, garantizando que refleja fielmente el trabajo realizado y los conocimientos adquiridos a lo largo del proyecto.

Licencia