ARENA FÉNIX · GUÍA PRÁCTICA

UNITY + FIREBASE = VIDEOJUEGOS DINÁMICOS



DAVID CANTÓN NADALES

MICROSOFT MVP
DESDE 2024

17TH UNITY AWARDS
FINALISTA
BEST DEVLOG SERIES

LIBRO DE PROYECTO / ARENA FÉNIX

Unity + Firebase

= videojuegos dinámicos

Crea una arena que cambia después de publicar

David Cantón Nadales

Primera edición · 2026

Dedicatoria

A mis padres, **José y Reyes**.

Gracias por creer siempre en mí. Por hacerlo cuando las cosas salían bien y, sobre todo, cuando todavía estaba buscando la manera de que salieran.

De vosotros aprendí lo que vale tener a alguien al lado que te diga «prueba otra vez». Este libro está lleno de esas segundas oportunidades.

Con todo mi cariño,

David

Prólogo

La primera vez que un personaje se mueve porque has pulsado una tecla, cuesta resistirse a pulsarla otra vez. Ya sabes que se va a mover. Da igual. Vuelves a hacerlo para comprobar que eso que hace diez minutos era una idea ahora está ahí, obedeciendo una regla que has escrito tú.

Hay otro momento que me gusta igual. Tu juego ya está instalado en otro dispositivo. Abres un panel, publicas un reto y, al entrar en el juego, aparece una misión nueva. El personaje sigue siendo el mismo. Tú tampoco has enviado otra versión. Lo que ha cambiado es el mundo que ese personaje encuentra al cruzar la puerta.

Este libro va de aprender a hacer eso. Vamos a construir una arena de supervivencia con un guardián, unas ascuas bastante pesadas y un fénix que ha confundido la expresión «aparecer al final» con «hacer una entrada dramática». El juego empieza siendo pequeño a propósito. Así podremos dedicar el esfuerzo a entender qué ocurre cuando una parte de él vive en un dispositivo y otra en un servidor.

Lo haremos paso a paso. Primero jugarás. Luego cambiarás una regla y comprobarás su efecto. Más adelante tendrás que resolver preguntas menos vistosas y muy importantes. ¿Qué sucede si el teléfono pierde cobertura al reclamar un premio? ¿Qué pasa si el jugador pulsa el botón dos veces? ¿Quién puede publicar una misión? ¿Y cómo damos marcha atrás si hemos mandado nueve enemigos a perseguir a alguien que acaba de aprender dónde está el botón de ataque?

Si estás aprendiendo, te invito a tocar los ejemplos. Cambia una cifra, quita la red, prueba un valor absurdo y fíjate en el resultado. Antes de pulsar **Play**, intenta adivinar qué va a ocurrir. Si aciertas, tendrás una explicación que poner a prueba. Si te equivocas, acabas de encontrar algo que merece la pena entender.

Cuando llegues al evento final, lo interesante será que sepas cómo repetirlo con tu propio juego. El fénix puede quedarse aquí. Bastante calor da ya.

David Cantón

Antes de entrar en la arena

Vas a necesitar Unity, nociones básicas de C# y ganas de probar cosas. Si sabes abrir una escena y entiendes un `if`, puedes empezar. El proyecto incluye la arena y sus mecánicas. Tendrás ocasión de modificarlas sin tener que construir un juego entero antes de llegar a Firebase.

La lectura tiene tres recorridos. **Local** usa datos incluidos en el juego o el laboratorio HTTP. **Emuladores Firebase** usa servicios de prueba que corren en tu equipo. **Nube** usa un proyecto Firebase propio. Un emulador Android hace de teléfono y puede hablar con la nube. Fíjate en los dos extremos de cada prueba, el lugar donde corre Unity y el lugar donde viven los datos.

Primero completa los capítulos 1 a 12 con Arena Fénix. Los capítulos 13 a 19 son prácticas guiadas que empiezan en una copia del juego. Cada una requiere su propio SDK o servicio. El capítulo 20 reúne el catálogo, las reglas, el escudo y el premio en un ensayo local.

Cómo orientarte en los ejemplos

- `ArenaFenix/` es el proyecto de Unity. Abre la carpeta que contiene `Assets`, `Packages` y `ProjectSettings`.
- `firebase-demo/` contiene el laboratorio, reglas, funciones y el panel local. Los ejercicios usan datos ficticios.
- `Assets/Arena/` contiene el código del juego. `ArenaDirector` coordina una partida, `ArenaStage` dibuja la arena y `ArenaData` define las reglas y datos que viajan entre capas.
- La tabla de comprobaciones de cada práctica indica dónde mirar y qué resultado esperar. Conserva siempre una copia del ejemplo antes de cambiarlo.

La ruta práctica empieza en Unity 6 en macOS y continúa hacia Android. En el apéndice encontrarás las versiones comprobadas. Consulta la matriz de Firebase para Unity para iOS u otras plataformas.

Puedes hacer buena parte del libro con servicios locales. Una cuenta de Google y un proyecto remoto entran en juego cuando llegues a los ejercicios marcados para consola. Si eres menor y necesitas crear cuentas, asociar una tienda o activar facturación, haz esa parte con una persona adulta. En las prácticas con coste potencial indicamos el requisito y la ruta de laboratorio.

Créditos y edición

Unity + Firebase = videojuegos dinámicos
David Cantón Nadales
Primera edición en español. Revisión editorial 1.1, noviembre de 2026.

Arena Fénix es un ejemplo didáctico. Las capturas se identifican por su entorno y las ilustraciones narrativas por su pie. Consulta las licencias de los materiales de terceros antes de redistribuirlos y usa tu propio proyecto Firebase para las prácticas.

Unity es una marca de Unity Technologies o de sus afiliadas en Estados Unidos y otros países. Firebase y Microsoft MVP pertenecen a sus respectivos titulares. David Cantón Nadales es Microsoft MVP desde 2024 y fue finalista de Best Devlog Series en los 17th Unity Awards. Esta es una publicación independiente del autor.

La ilustración de portada y las ilustraciones narrativas se generaron con ayuda de IA a partir del concepto y el avatar del autor. El diseño de los diagramas y las capturas técnicas tienen finalidades distintas. Encontrarás la procedencia indicada junto a las figuras.

Índice

Parte I · Enciende la arena 9

01 · Una arena que ya se puede jugar 10

02 · Conecta Firebase sin acoplarlo al juego 14

03 · Authentication y la primera partida sin formulario 17

04 · Cloud Firestore y el progreso que te espera al volver 20

Parte II · Cambia la arena con el juego instalado 23

05 · Remote Config y las reglas de la próxima partida 24

06 · Firestore y un catálogo de misiones 29

07 · Storage y Addressables para estrenar una apariencia 32

08 · Hosting y un panel para publicar sin abrir Unity 35

Parte III · Quién puede hacer qué 38

09 · Cloud Functions y un premio que se entrega una vez 39

10 · Una clasificación que dice de dónde salen sus puntos 42

11 · Realtime Database y una sala de espera 44

12 · Reglas de seguridad, App Check y pruebas de acceso 46

Parte IV · Aprende de las partidas 49

13 · Analytics y la primera oleada que nadie acaba 50

14 · A/B Testing y dos maneras de empezar 52

15 · Cloud Messaging y una invitación que llega a tiempo 55

16 · Crashlytics y una pista en medio del humo 57

17 · App Distribution y Test Lab para salir del propio ordenador . . . 59

Parte V · Abre las puertas del evento 61

18 · Firebase AI Logic y el fénix comentarista 62
19 · AdMob y una recompensa opcional 64
20 · El evento de fin de semana 66
21 · Mantén la arena cuando crece 69

Apéndices **71**

A · Qué servicio elegir 72
B · Glosario de la arena 75
C · Cuaderno de comprobaciones 77
D · Recursos para seguir 79
E · Qué se comprobó para esta edición 80

PARTE I

Enciende la arena



Figura 01. Ilustración. El guardián prepara una maqueta de la arena antes de probar el primer enemigo.

En esta parte abrirás la arena, separarás el juego de sus fuentes de datos y aprenderás a reconocer al jugador que guarda su progreso.

CAPÍTULO 01

Una arena que ya se puede jugar

La primera tarea del libro es perder una partida. En serio. Si te quedas quieto y dejas que se acerquen las ascuas, verás que baja la vida y que el juego vuelve al menú. Si nada de eso sucede, tenemos un problema estupendo para investigar. Si sucede, ya tienes un punto de partida que funciona.

Abre la escena y encuentra el pulso

1. Desde Unity Hub, añade la carpeta `ArenaFenix` como proyecto y ábrela con el Editor indicado en `ProjectSettings/ProjectVersion.txt`. Espera a que termine la importación.
2. En **Project**, abre `Assets/Scenes/Arena.unity`. Si vas a regenerar la escena, usa **Arena Fénix ☐ Preparar escena del libro** en una copia de trabajo.
3. Pulsa **Play** y luego **Entrar en la arena**. Muévete con WASD o las flechas. Con **Espacio** lanzas un pulso circular. En la interfaz también puedes mantener pulsado sobre el suelo para moverte y usar el botón de pulso.

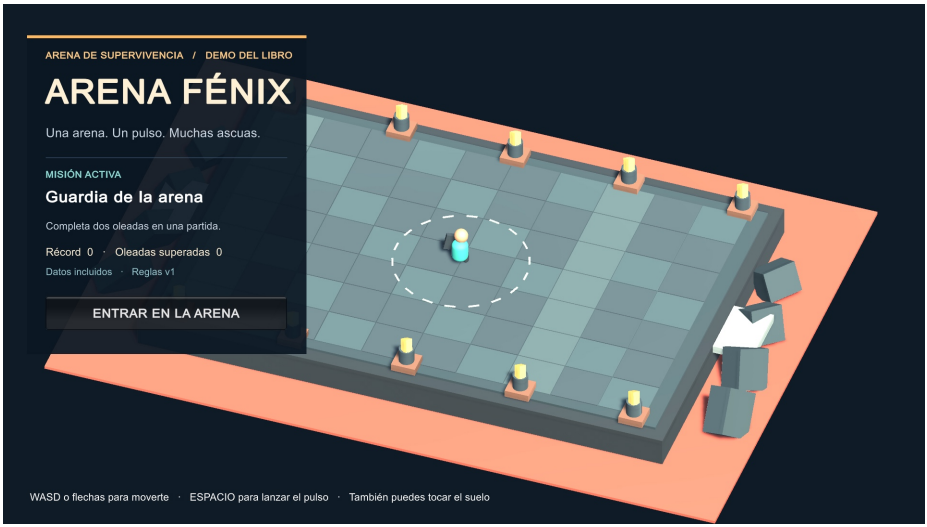


Figura 02. Captura de Unity. El vestíbulo muestra la misión de guardia, el récord local y la versión de las reglas incluidas.

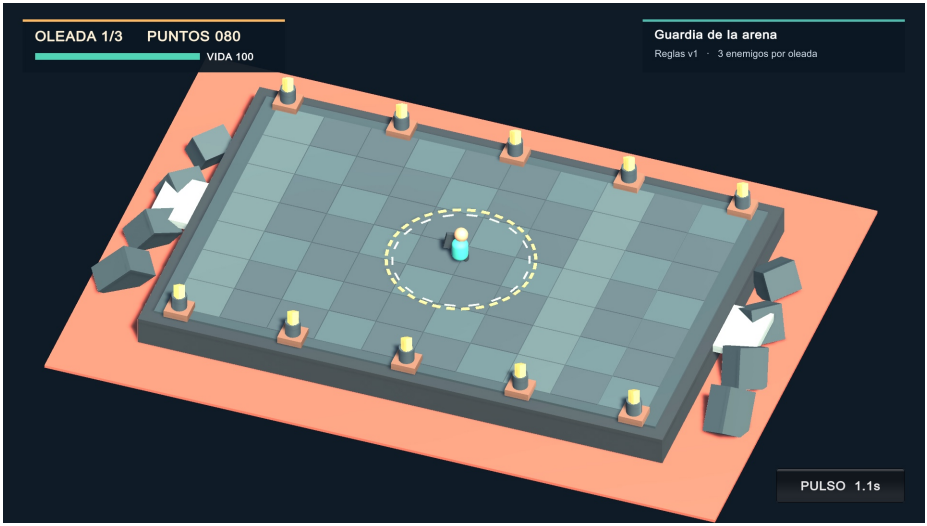


Figura 03. Captura de Unity. El pulso dibuja un círculo en el suelo y el marcador confirma que terminó la primera oleada.

El pulso tiene un radio y un tiempo de espera. Acércate a una ascua, espera a que entre en el círculo y ataca. Si pulsas demasiado pronto, le habrás ofrecido una demostración de fuegos artificiales gratis. Derrotar a un enemigo suma diez puntos y completar una oleada añade cincuenta. En la última oleada, el fénix ocupa el lugar de una

ascua. Aguanta tres impactos, retrocede en los dos primeros y suma puntos cuando cae.

Del componente al juego

En la jerarquía hay un objeto **Arena Fénix** con el componente `ArenaDirector`. Al empezar, crea el escenario y el guardián. En cada fotograma lee la entrada, mueve al personaje y llama a `Tick`, el paso que acerca enemigos y actualiza los temporizadores. `SpawnWave` coloca a las ascuas en torno a la arena. El pulso recorre los enemigos de atrás hacia delante porque algunos salen de la lista al ser derrotados.

Esa separación te permite localizar un fallo. Si la vida baja aunque no toques el teclado, estás mirando la simulación. Si el personaje no sigue el clic, revisa cómo se traduce la posición del cursor al suelo. Si el menú no aparece, comprueba la consola antes de cambiar la velocidad de las ascuas.

Prueba	Qué deberías ver
Entra y quédate quieto	Las ascuas se acercan y la vida cae por tramos.
Lanza el pulso a distancia	Aparece el círculo y comienza la recarga.
Lanza el pulso con una ascua cerca	Desaparece la ascua y sube la puntuación.
Termina y vuelve a entrar	Empieza una partida nueva y se conserva el mejor resultado local.

Tu primer cambio

Sal de Play y abre `Assets/Arena/ArenaData.cs`. En `ArenaRules`, cambia el valor inicial de `enemiesPerWave` de tres a cuatro. Deja que Unity compile, inicia otra partida y mira la esquina superior derecha. Ya tienes un cambio de dificultad comprobable.

Vuelve a tres antes de continuar. En el capítulo 5 haremos el mismo experimento con una compilación ya hecha. Tener una referencia nos ahorra el clásico «me parece que ahora hay más bichos».

Una prueba que vale más que diez «a mí me funciona»

Juega dos rondas de la misma forma. En la primera, lanza el pulso lejos de las ascuas. En la segunda, espera a que crucen el círculo de ataque. Anota la vida que te queda después del primer pulso, cuántos enemigos siguen en pantalla y cuánto tarda en recuperarse el botón. Podrás volver a ese registro cuando cambies la dificultad.

Si quieres comprobar la última oleada sin pasar por todas, pon `totalWaves = 1` en una copia. La primera oleada será también la última y aparecerá el fénix. Recupera el valor antes de hacer las pruebas de configuración remota.

CAPÍTULO 02

Conecta Firebase sin acoplarlo al juego

Imagina que se corta la red justo cuando el jugador pulsa **Entrar**. A él le da igual dónde guardas la configuración. Quiere jugar. Nuestro primer trabajo al conectar Firebase es decidir qué hace Unity mientras los servicios responden, y qué hace cuando tardan demasiado o fallan.

Una puerta para cada fuente de datos

El director conoce la interfaz `IArenaSource`. El laboratorio HTTP y el adaptador de Firebase responden a las mismas peticiones. El laboratorio cambia reglas y misiones de ejemplo. Devuelve progreso vacío y deja el guardado en el dispositivo. El adaptador de Firebase puede consultar el perfil y enviar la partida a una función. La fuente se elige al iniciar la escena. El movimiento y el combate pueden empezar con las reglas incluidas.

```
public interface IArenaSource
{
    string Label { get; }
    Task<ArenaSnapshot> ReadAsync();
    Task<ArenaProgress> ReadProgressAsync();
    Task SaveAsync(ArenaRecord record);
}
```

Un `Task` representa una operación que puede terminar más tarde. Al usar `await`, el hilo de Unity puede seguir dibujando mientras esperamos. Lee los datos fuera del bucle de fotogramas y actualiza las pantallas cuando llegue el resultado. Si un evento del SDK se ejecuta en otro hilo, vuelve al hilo principal antes de tocar un objeto de Unity.



Figura 04. Diagrama. ArenaDirector conserva el juego local y consulta a Firebase o al laboratorio mediante la misma interfaz de datos.

Prepara la conexión real

1. Crea un **proyecto de pruebas** en Firebase. Registra una aplicación Android con el identificador que figura en **Project Settings ☐ Player ☐ Android**. En la demo usamos `net.davidcanton.arenafenix`. Si cambias el identificador en Unity, registra ese mismo valor en Firebase.
2. Obtén la configuración Android de tu aplicación con la CLI o con el flujo oficial y coloca `google-services.json` en `Assets`. En iOS usa el identificador y el archivo de configuración de iOS. El SDK genera su configuración de Editor a partir de la configuración móvil.
3. Importa las bibliotecas indicadas en `README.md` para la práctica. Usa una sola versión del SDK. Añade el símbolo `ARENA_FIREBASE` cuando estén importadas y activa **Connect Firebase** en el componente de la escena. Revisa la guía oficial de integración para los requisitos de tu plataforma.
4. En la ruta de nube, `FirebaseArenaSource` espera a `CheckAndFixDependenciesAsync` y comprueba el resultado. La ruta de emuladores en macOS crea una instancia con las opciones

ficticias de `demo-arena-fenix`. Si la conexión falla, la arena puede empezar con sus reglas incluidas.

El archivo móvil identifica un proyecto Firebase. Las credenciales de administración son otra cosa y se quedan en el servidor. Nunca importes en Unity un archivo de cuenta de servicio para resolver un error de permisos.

Aprende a leer el estado

En una prueba conectada, el vestíbulo indica qué adaptador está seleccionado y muestra la revisión activa. Para confirmar que llegó un cambio de la nube, compara esa revisión con la que publicaste. Si la lectura completa falla, el director puede conservar las últimas reglas válidas de la sesión. Si el catálogo no responde o la misión ha vencido, puede ofrecer el reto incluido. Tras reiniciar, el progreso local sigue en `PlayerPrefs`. Cada SDK decide qué datos persistentes puede recuperar. Comprueba las dos situaciones por separado.

Reto. Anota qué parte de tu próximo juego debe funcionar siempre, aunque no haya señal. Separa entrar a jugar, guardar el resultado local y competir en una clasificación. Son tres promesas diferentes.

CAPÍTULO 03

Authentication y la primera partida sin formulario

A los diez segundos de abrir un juego, el jugador quiere saber si es divertido. Pedirle que invente una contraseña con un jeroglífico y tres símbolos misteriosos puede esperar. Con Authentication podemos empezar con una cuenta anónima y ofrecer una forma de conservarla cuando tenga sentido.

El invitado ya tiene identidad

En el proyecto Firebase de pruebas, abre **Authentication** ☐ **Sign-in method** y habilita **Anonymous** para la ruta de consola. En el laboratorio, arranca el emulador de Auth con el proyecto de demostración del README. `FirebaseArenaSource` configura el emulador antes de usar Auth cuando activas esa ruta.

En Unity se comprueba `CurrentUser` y se llama a `SignInAnonymouslyAsync` cuando aún no hay una sesión. El resultado aporta un UID, un identificador que usaremos en rutas como `players/{uid}`. No lo confundas con un apodo ni lo muestres en una clasificación. Si alguien escribe «SuperFénix» como apodo, sigue siendo el mismo UID.

```
var user = auth.CurrentUser;
if (user == null)
    user = (await auth.SignInAnonymouslyAsync()).User;
```

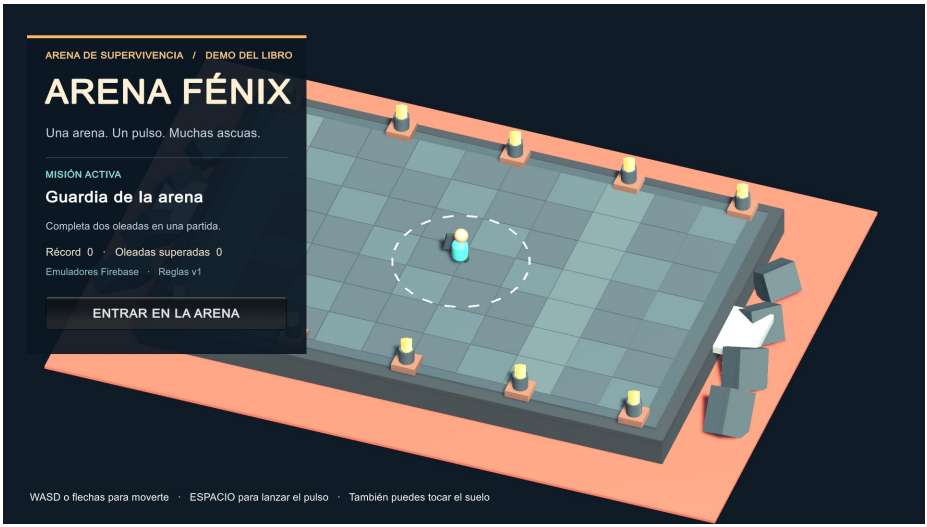


Figura 05. Captura de Unity. La compilación con Firebase muestra los datos recibidos de los emuladores.

En el emulador, mira la pestaña **Authentication** y comprueba que aparece una cuenta anónima. En un dispositivo con el proyecto de pruebas, repite la apertura de la app para comprobar que recupera la sesión en condiciones normales. Una cuenta invitada sin vincular se puede perder al borrar datos o cambiar de dispositivo.

Vincula la cuenta en el momento adecuado

«Conservar mi progreso» es un botón para añadir como práctica. En una copia de Unity y un proyecto de pruebas, habilita el proveedor de correo y contraseña y usa un correo ficticio que aún no pertenezca a una cuenta. Guarda el UID anónimo, construye la credencial con `EmailAuthProvider.GetCredential` y llama a `LinkWithCredentialAsync` sobre ese mismo `FirebaseUser`. Comprueba que el UID devuelto coincide. En la segunda instalación, inicia sesión con el proveedor vinculado para recuperar el mismo perfil.

Haz una prueba negativa con una credencial que ya pertenezca a otra cuenta. Si Firebase la rechaza, conserva la sesión que estaba abierta y ofrece una elección clara. La arena fusiona los máximos de un mismo

perfil. Una fusión entre dos UID distintos necesitaría su propio procedimiento y una política sobre compras y premios consumidos.

Comprueba las dos identidades

1. Apunta si el usuario anónimo aparece en el emulador. Usa únicamente perfiles ficticios.
2. Crea dos sesiones de prueba. Comprueba que sus UID son distintos.
3. Intenta leer el perfil de una desde la otra. El capítulo 12 automatiza este caso y muestra por qué autenticarse no basta para ver datos ajenos.

Puedes seguir la implementación en la guía de cuentas anónimas para Unity y la guía para vincular proveedores. El panel editorial del libro usa una cuenta de pruebas con un permiso diferente. Haber sobrevivido tres oleadas no te asciende a administrador.

CAPÍTULO 04

Cloud Firestore y el progreso que te espera al volver

Has superado la arena en el autobús. En casa abres el juego en otro teléfono. ¿Qué debe viajar? Para nuestra primera versión bastan el mejor resultado y el número máximo de oleadas superadas. Elegir campos que se puedan fusionar con una regla sencilla nos permite practicar antes de introducir inventarios y monedas.

Diseña el documento antes de escribirlo

```
players/{uid}
  bestScore       entero de 0 a 700
  bestWaves       entero de 0 a 5
  updatedAt       hora del servidor
```

Un documento es una unidad de datos. La colección agrupa documentos con un propósito común. El UID forma parte de la ruta para que las reglas puedan comparar la identidad de la sesión con la ruta solicitada. El récord de práctica llega a 700 como máximo con cinco oleadas, nueve enemigos por oleada y cincuenta puntos extra por oleada. La clasificación y los premios tienen rutas separadas y escrituras de servidor.

El guardado local del ejemplo conserva máximos. Un valor remoto más bajo no borra automáticamente el progreso que el dispositivo ya conoce. Para escribir los máximos remotos, el ejemplo de servidor utiliza una transacción. La transacción lee el valor existente y calcula el mayor. Si otra operación cambia el documento mientras tanto, Firestore puede volver a intentar la operación.



Figura 06. Diagrama. Dos dispositivos guardan resultados y el servidor combina los máximos del perfil.

Práctica con dos dispositivos de prueba

1. Arranca Auth, Firestore y Functions con el comando completo del README y ejecuta el sembrado. En Unity, la ruta Firebase lee `players/{uid}` de la sesión autenticada. La escritura pasa por `recordRun` en Functions.
2. Ejecuta `node functions/test-local.mjs` desde `firebase-demo`. El verificador envía 170 y luego 80 con la misma identidad. El perfil conserva 170 y una segunda identidad no puede leerlo.
3. Para la práctica opcional en dos dispositivos, usa la cuenta vinculada del capítulo 3 en ambos. Cierra y vuelve a abrir para distinguir lo que recuperó Firestore de lo que apareció de inmediato desde `PlayerPrefs`, que guarda un récord de la instalación. Una sesión invitada nueva tiene otro UID y otro perfil.
4. Desconecta la red en una partida local. Comprueba que el resultado de la sesión se conserva en el equipo. El ejemplo local no promete una cola duradera de reenvío: para añadirla, guarda un identificador por partida y reintenta el mismo envío cuando vuelva la conexión.

Una cola fiable conserva estado entre reinicios, limita reintentos y tiene forma de mostrar al jugador qué falta por sincronizar. Repite cada envío con el mismo identificador. `PlayerPrefs` sirve aquí para un récord de práctica que el jugador puede modificar en su equipo. Los saldos y premios con valor dependen de decisiones del servidor.

El siguiente salto

Una moneda gastada no se puede combinar tomando el máximo de dos saldos. Dos dispositivos podrían gastar los mismos créditos. Para ese caso, el servidor autoriza cada operación y la registra. El diseño de los datos empieza por la pregunta que quieres poder responder cuando algo va mal.

Primeros pasos con Firestore · Transacciones y operaciones por lotes

PARTE II

Cambia la arena con el juego instalado



Figura 07. Ilustración. Dos palancas cambian el siguiente desafío mientras el guardián observa las ascuas.

Publicarás parámetros, misiones y archivos. Cada uno viaja por un camino distinto y cada cambio se prueba con una compilación que el jugador ya tiene.

CAPÍTULO 05

Remote Config y las reglas de la próxima partida

Vamos a enviar dos ascuas más a la arena. El jugador conserva el mismo ejecutable. La persona que opera el juego publica unos valores distintos y el juego los aplica cuando empieza la siguiente partida. Hemos separado la **regla** de la **forma en que se ejecuta**.

Primero ensaya sin cuentas

Arranca `node lab.mjs` desde la carpeta `firebase-demo`. Abre `http://127.0.0.1:5177`. El panel sirve la configuración de ejemplo en `/current.json`. En macOS, puedes pasarle a la compilación de prueba el argumento que se indica en el README:

```
-arena-lab http://127.0.0.1:5177/current.json
```

1. Deja el escenario base. Entra en la arena y cuenta tres enemigos. Mira también el número de revisión en el HUD.
2. Con la partida abierta, pulsa **Activar fin de semana** en el panel. La partida conserva su copia de las reglas. Termina la ronda y vuelve al menú.
3. Entra de nuevo. La revisión pasa a 2, la primera oleada trae cinco enemigos y el pulso se recupera antes.
4. Pulsa **Recuperar reglas base** y repite. Tres enemigos vuelven a ser suficientes para una tarde.



Figura 08. Captura del navegador. El laboratorio local sirve la revisión 2 y cinco enemigos por oleada a Unity.

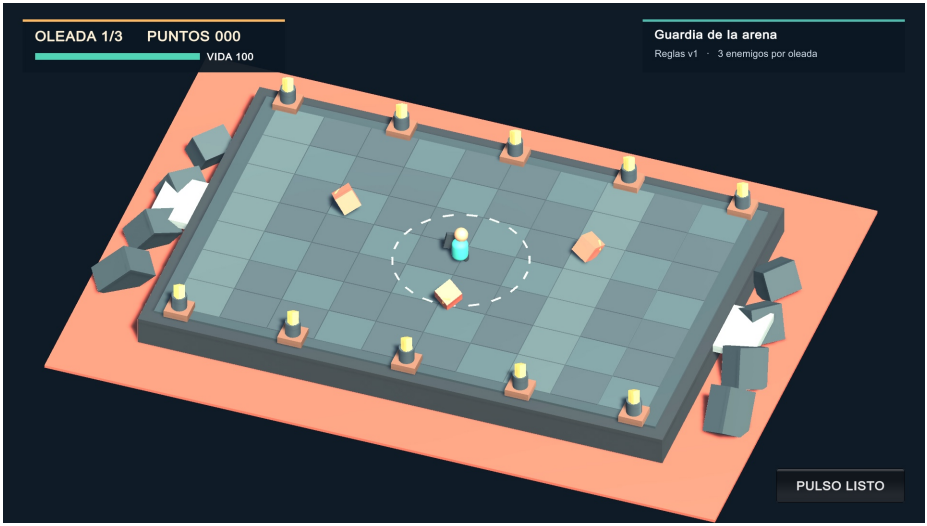


Figura 09. Captura de Unity. La compilación con las reglas incluidas empieza con tres enemigos.

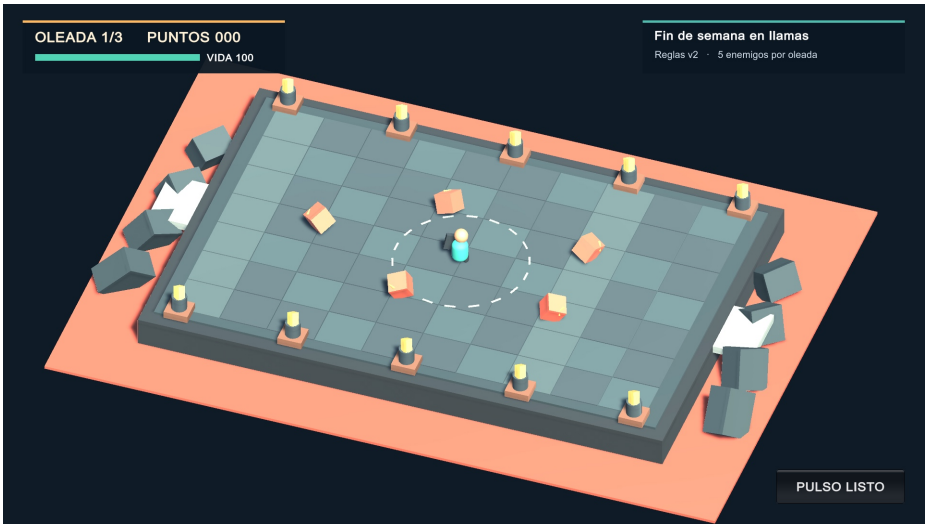


Figura 10. Captura de Unity. Una nueva partida en la misma compilación muestra la revisión 2 y cinco enemigos por oleada.

Este primer ensayo usa el adaptador HTTP del libro. Firebase Emulator Suite cubre el catálogo, la identidad, las funciones y los archivos. Para probar **Remote Config real**, usa el SDK y el proyecto en la nube. El capítulo 20 combina el HTTP de reglas con los emuladores para ensayar todo en local. Servicios de Emulator Suite

Publica los parámetros en Remote Config

En tu proyecto de pruebas, abre **Remote Config**, crea parámetros numéricos y guarda la plantilla inicial. En el repositorio, `remote_config.json` contiene la revisión 2 para comparar. Configura primero la revisión 1 para que haya un valor anterior al que volver.

Parámetro	Base	Fin de semana
`arena_rules_revision`	1	2
`arena_enemies_per_wave`	3	5
`arena_pulse_cooldown`	1,10	0,85
`arena_total_waves`	3	3

Los números decimales de la plantilla JSON se escriben con punto. `SetDefaultsAsync` instala los valores de respaldo. `FetchAndActivateAsync` recupera y activa cambios cuando corresponde. El adaptador sustituye los enteros de fuera de rango por sus valores base. `ArenaRules.Copy` acota el tiempo de pulso entre 0,35 y 3 segundos. Publica 99 enemigos en un proyecto de pruebas y comprueba que la siguiente partida válida usa 3.

La revisión es un parámetro que elegimos nosotros para diagnosticar una partida. El número de versión que muestra el historial de Firebase pertenece a la plantilla entera. Anota ambos si necesitas explicar qué pasó. Para volver atrás, selecciona la versión anterior en el historial de Remote Config y verifica el resultado en **una partida nueva**.

Los valores que recibe el cliente pueden inspeccionarse. Guarda las claves privadas y las decisiones de autorización en el servidor. Parámetros de Remote Config para clientes

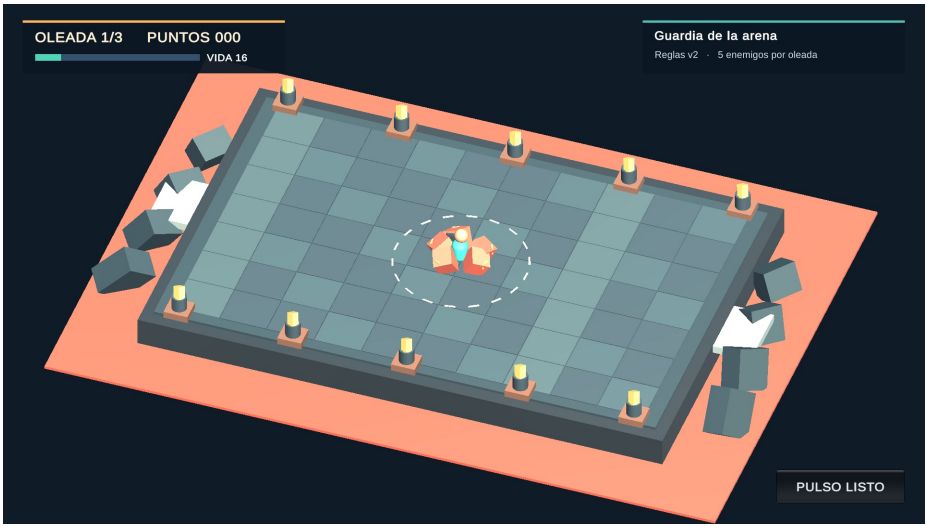


Figura 11. Captura del emulador Android. La compilación conectada a Firebase en la nube inicia la primera oleada con la revisión 2 y cinco enemigos.

Cuándo se aplica un cambio

El SDK puede usar los valores activados en una lectura anterior. El adaptador de la demo configura diez minutos como intervalo mínimo de recuperación remota. Al abrir el vestíbulo y al entrar en una partida, Unity consulta al adaptador. Si sigues dentro del intervalo, puedes recibir los valores que ya tenía. Usa el intervalo reducido solo en una compilación de pruebas. Mira la revisión del HUD antes de dar por hecho que el cambio llegó.

Si necesitas reaccionar a una publicación mientras el juego está abierto, consulta la opción de actualización en tiempo real y guarda los valores como candidatos. En la arena esperamos al siguiente inicio de partida para que el combate sea predecible.

Prueba de salida. Mantén una partida viva durante la publicación. Comprueba que sigue con su revisión, arranca otra y anota qué cambió. Así distingues un dato que llegó de una regla que de verdad está usando el juego.

Remote Config para Unity · Estrategias de carga

CAPÍTULO 06

Firestore y un catálogo de misiones

Para anunciar que «hay una misión de fin de semana», cuatro números se quedan cortos. Necesitas un título, un objetivo, una ventana de tiempo y una manera de saber si el cliente entiende lo que recibe. Es el momento de dar a las misiones su propio catálogo.

Diseña la ficha y el puntero

`missions/{id}` guarda los datos de una misión. `live/active` apunta a la que queremos mostrar ahora. Un puntero es un documento pequeño que lleva el ID de otro. Cambiarlo permite recuperar una misión anterior mientras siga activa. En este ejemplo cada publicación tiene un ID propio, para que una recompensa de ayer siga refiriéndose al mismo objetivo.

```
{
  "title": "Lluvia de ascuas",
  "summary": "Completa dos oleadas en la arena.",
  "targetWaves": 2,
  "minimumBuild": 1,
  "startsAt": "2026-12-05T12:00:00Z",
  "endsAt": "2026-12-07T12:00:00Z",
  "status": "published"
}
```

Adapta las fechas de ejemplo a la hora del ensayo. Usamos una ventana UTC que incluye el inicio y excluye el final. Una misión que acaba a las 18:00 deja de estar activa a las 18:00 exactas. El servidor comprueba su hora al autorizar un premio. El cliente usa su reloj para orientar al jugador y puede equivocarse.

Abre el catálogo desde Unity

1. Autentica al invitado y lee `live/active`. Si no existe o el campo `missionId` está vacío, conserva una misión incluida.
2. Carga el documento de la misión. Comprueba que la versión instalada puede interpretarla, que está en su ventana y que pide como máximo las oleadas disponibles. `Build = 1` en `ArenaDirector` es la versión de contenido de este ejercicio.
3. Enseña el título y el objetivo antes de entrar. Copia la misión con la que empezó la partida. Si el equipo publica otra durante el combate, el jugador termina con el objetivo que vio al empezar.
4. Lee al abrir la escena y al entrar en una partida. Las reglas de Firestore permiten consultar fichas publicadas con una sesión válida. El estado `published` abre la lectura. Para guardar una sorpresa hasta su estreno, conserva la ficha como borrador o créala al publicar.



Figura 12. Diagrama. Remote Config aporta parámetros, el catálogo de Firestore apunta a la misión activa y Storage sirve el paquete visual.

Haz que falle de forma predecible

En Firestore Emulator UI, cambia temporalmente `minimumBuild` de la misión activa a 9. Abre otra partida. El cliente de práctica tiene versión 1 y usa la misión incluida. Recupera `minimumBuild: 1` y repite la prueba poniendo `endsAt` en una fecha pasada. Restaura ambos campos o ejecuta el sembrado al acabar.

Después simula un error de edición. Publica una misión que exige cinco oleadas cuando la arena está configurada para tres. El juego la sustituye por la misión incluida al iniciar una ronda. Si la regla solo deja una oleada, ofrece un entrenamiento de una oleada. Aunque los campos sean válidos por separado, el objetivo tiene que ser alcanzable con esas reglas.

Reto. Añade al panel una previsualización con las reglas que recibirá el grupo de prueba. Si no puedes garantizar que todos tendrán el mismo número de oleadas, comprueba que el objetivo es alcanzable con todas las variantes compatibles. Firebase entrega el catálogo y Remote Config por separado.

Una decisión sobre los eventos que terminan

En Arena Fénix puedes terminar la partida que empezaste con la misión en plazo. La función de reclamación, en cambio, exige que el evento siga activo al pedir el premio. Si tu juego promete un margen después del cierre, almacena una hora límite para reclamar y decide qué pruebas de la partida se aceptan en ese margen. Escribe las dos fechas en la ficha de operaciones y pruébalas por separado. Una hora mal explicada puede hacer que un reto perfectamente programado se sienta injusto.

CAPÍTULO 07

Storage y Addressables para estrenar una apariencia

En el catálogo cabe el nombre de una apariencia. La imagen, el audio y los modelos viven en archivos. Para ver bien la diferencia, vamos a seguir el camino de un escudo. Primero llega una imagen pequeña. Después veremos qué cambia al publicar contenido de Unity con Addressables.

Primer ejercicio con un archivo versionado

Usa el emulador de Storage y el archivo de prueba `seasons/fin-de-semana-v2/escudo.png`. Con la sesión de un jugador, pide el archivo por su ruta y pon un tamaño máximo. El ejemplo de lectura recibe los bytes mediante el SDK. Comprueba que la imagen se puede decodificar antes de ponerla en el juego. Mientras carga, muestra la apariencia incluida.

En el panel de operaciones del capítulo 8, la casilla **Mostrar el escudo de temporada** añade `skinVersion` a la misión. El sembrado ya colocó el archivo en Storage. En Unity, `FirebaseArenaSource.ReadShieldAsync` descarga hasta 128 KiB. `ArenaDirector` decodifica la imagen, comprueba sus dimensiones y la muestra en el vestíbulo. Al volver a una misión sin apariencia, se libera la textura anterior.

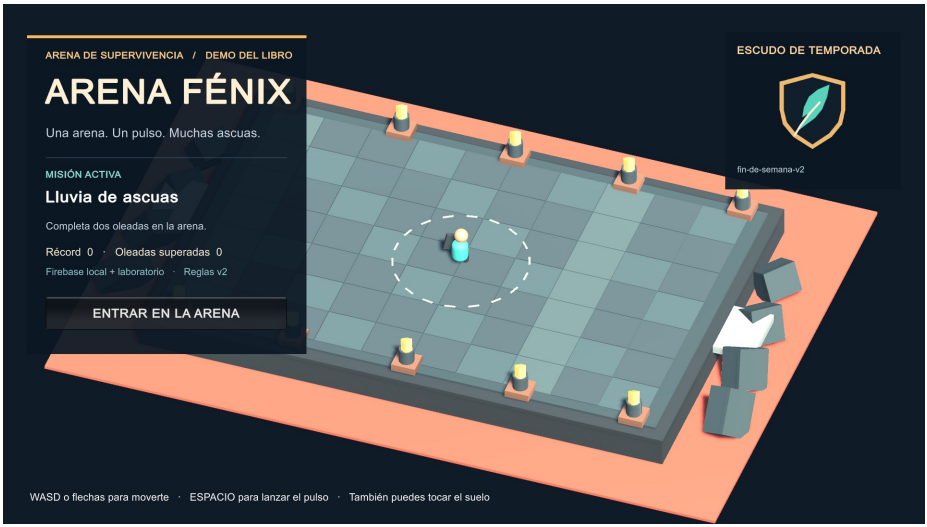


Figura 13. Captura de Unity. El vestíbulo muestra el escudo descargado del emulador de Storage junto a la misión publicada.

La versión forma parte de la ruta. Publica el nuevo archivo en una ruta nueva. El cliente que aún tiene la versión anterior puede seguir consultando el archivo que conoce. Si el archivo está incompleto, apunta de nuevo a la versión anterior. Haz la prueba eliminando **una copia de ensayo** del paquete y observa si la arena sigue siendo jugable.

Para un recurso de temporada pequeño puede bastar un PNG. Publicar una escena, un prefab o audio con dependencias te lleva al siguiente ejercicio. En la nube, Cloud Storage for Firebase requiere el plan Blaze. El laboratorio de este capítulo usa el emulador y permite hacer la prueba con datos locales. Requisitos de Cloud Storage

Segundo ejercicio con Addressables

1. Abre Package Manager en una copia e instala la versión de Addressables que Unity indique como compatible con tu Editor. Marca un prefab de escudo como Addressable, dale una dirección estable y sitúalo en un grupo de contenido remoto. Este segundo ejercicio empieza con tu propia configuración de Addressables.

2. Configura el perfil con una ruta de compilación y una URL de carga para **la plataforma que estás probando**. Activa el catálogo remoto. Construye el contenido y guarda el estado de esa compilación para las actualizaciones.
3. Sube el catálogo y los paquetes a un servidor que pueda responder las URL esperadas por Addressables. En una práctica usa una URL local. En un despliegue real decide quién puede leer esos archivos. Un enlace accesible a cualquiera funciona como un recurso público y no sirve para proteger una compra.
4. Carga el prefab por su dirección con `Addressables.InstantiateAsync`, conserva el identificador de esa operación y usa `Addressables.ReleaseInstance` al retirar la instancia. Muestra la apariencia de respaldo si falla. La versión instalada ya conoce cómo usar el prefab. Un cambio que necesita código nuevo requiere una compilación nueva.

Un catálogo remoto es un índice de dónde están los paquetes y qué contienen. Firebase puede indicar qué versión está activa. Addressables resuelve cómo cargar los recursos de Unity. Si mezclas el paquete de Android con el de macOS, el fénix seguirá siendo lo único que funciona bien en llamas.

Paquete Addressables para Unity 6 · Cloud Storage para Unity

CAPÍTULO 08

Hosting y un panel para publicar sin abrir Unity

Hasta ahora el equipo ha escrito valores a mano. En un proyecto que crece, la persona que programa el pulso y la persona que prepara los retos pueden ser dos personas distintas. Vamos a dar a esta última una pantalla que le enseñe lo que está a punto de publicar.

Sigue el recorrido de una publicación

Arranca Hosting, Auth, Firestore y Functions en el emulador y ejecuta el sembrado del README. Abre `http://127.0.0.1:5100/operaciones.html`. El panel permite leer la misión activa con una sesión invitada y publicar con una cuenta editorial ficticia.

1. Abre el panel como visitante. Lee la misión actual. El formulario de publicación está deshabilitado hasta que se obtiene una sesión con permiso.
2. Entra como **editora de prueba**. El sembrado local ha creado la cuenta y le ha asignado el permiso `arenaEditor`. El panel lee el permiso del token.
3. Rellena un ID nuevo y publica. `publishMission` vuelve a verificar el permiso, comprueba que la ventana ya está abierta y guarda el documento y el puntero en la misma operación. El panel local propone una ventana de un día. Prepara el recurso con antelación y pulsa publicar al comenzar el ensayo.
4. Desde otra ventana o desde el juego conectado al emulador, vuelve a leer la misión. Recupera después `guardian` desde el panel. Confirma el cambio en los dos clientes.



Figura 14. Captura de Hosting local. Una sesión editorial publica una misión mediante Cloud Functions y el panel muestra el nuevo puntero.

Los botones del navegador ayudan a guiar al usuario. La autorización la decide la función. Si alguien llama a la URL de la función sin el permiso, se le deniega la publicación aunque haya copiado toda la interfaz.

Del panel local al equipo real

En Firebase Hosting puedes preparar una vista previa de la web. El panel descargable se bloquea fuera de `http://127.0.0.1` y sus cuentas son ficticias. Para llevarlo a un equipo real hace falta adaptar la conexión, los roles y el inicio de sesión. Comprueba a qué proyecto apuntan los SDK antes de abrir una vista previa a otras personas, porque puede usar los datos del proyecto conectado.

Antes de dar permisos reales, separa quien redacta de quien aprueba, decide dónde queda el historial y define qué versión se puede recuperar. El ejemplo crea una misión por publicación y cambia un

puntero. Un juego en producción puede añadir un flujo de revisión, un registro con el responsable y una comprobación de que los recursos a los que apunta la misión existen.

Lo que debes conservar de cada publicación

Apunta el ID de misión, los valores de Remote Config y la ruta del recurso. Si alguien del equipo ve «Lluvia de ascuas» pero su compañero ve «Guardia de la arena», esos tres datos permiten empezar por el lugar correcto. Pídeles también que indiquen si entraron antes o después de la publicación y qué compilación tienen.

En una práctica con dos pestañas, intenta publicar el mismo ID dos veces. La segunda operación debe fallar y el puntero debe conservar el ID publicado. Después recupera la misión anterior con el botón de reversión.

Pruebas y vistas previas con Firebase Hosting

PARTE III

Quién puede hacer qué



Figura 15. Ilustración. El guardián comprueba una ficha mientras una ascua intenta entrar con un casco prestado.

Resolverás qué puede hacer cada jugador, cómo se guarda una partida y por qué un premio se entrega una sola vez aunque la petición se repita.

CAPÍTULO 09

Cloud Functions y un premio que se entrega una vez

Imagina que terminas una misión y pulsas **Reclamar**. El servidor concede la insignia, pero la conexión se corta antes de que llegue la respuesta. Pulsas otra vez. ¿Ha ocurrido dos veces, una vez o ninguna? Primero vas a probar esa situación desde el verificador. Después podrás añadir el botón a Unity.

Identifica la partida antes de enviarla

Al comenzar, Unity crea un identificador de partida de 32 caracteres. Lo conserva para todos los intentos de guardar ese resultado. La función `recordRun` obtiene el UID de la sesión y escribe en `players/{uid}/runs/{runId}`. Si el mismo resultado llega otra vez, devuelve `duplicate`. Si llega con la misma clave y una puntuación distinta, rechaza la operación.

```
Primer envío          partida R7 → recorded
Mismo envío otra vez  partida R7 → duplicate
R7 con otro resultado          → already-exists
```

Aquí hemos abreviado el identificador para leerlo mejor. En el código es un GUID. Este patrón se llama **idempotencia**. Repetir la misma operación deja el estado como si se hubiera hecho una sola vez.

Reclama la insignia de práctica

`claimMission` lee la partida guardada y la ficha de la misión. Comprueba que pertenecen al mismo jugador y objetivo, que el evento está en plazo y que el resultado supera las oleadas exigidas. Si todo cuadra, crea `players/{uid}/claims/{missionId}`. Esa ruta significa «una insignia de práctica por jugador y misión».

La lectura de la reclamación y su creación forman una transacción. Dos peticiones pueden llegar juntas. Si una crea la insignia mientras la otra está en marcha, la segunda vuelve a leer antes de escribir y devuelve la insignia ya concedida. La respuesta distingue `claimed` de `alreadyClaimed`, así que la interfaz puede decir «Ya la tienes» sin descontar ni sumar nada.

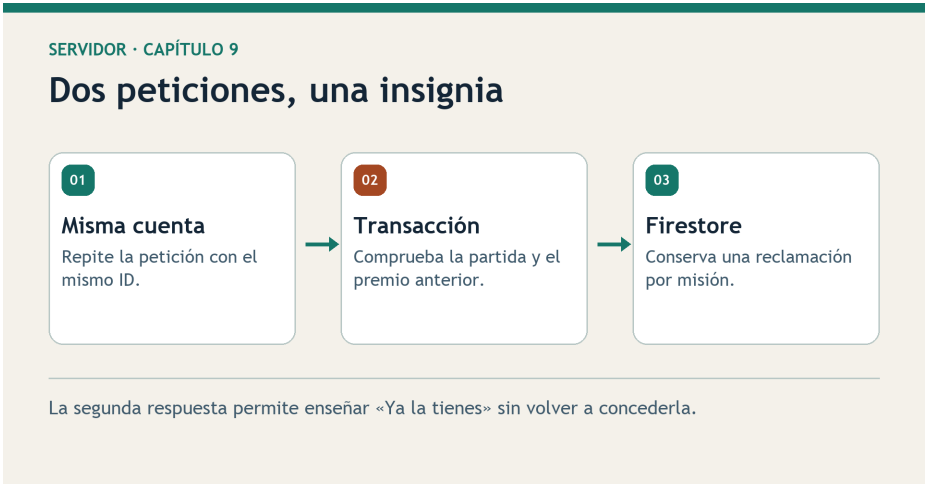


Figura 16. Diagrama. Dos peticiones llegan a la función y la transacción conserva una sola reclamación por jugador y misión.

Lanza la prueba de las dos peticiones

En el laboratorio, ejecuta `node functions/test-local.mjs` después del sembrado. El verificador envía un resultado ficticio de dos oleadas, llama a la función dos veces en paralelo y comprueba que las respuestas son una concesión y una reclamación ya existente. Otra sesión intenta reclamar usando ese ID y recibe un rechazo, porque la partida pertenece al primer jugador.

Para conectar el ejercicio a un botón de Unity, `FirebaseArenaSource` ofrece `ClaimAsync`. Llama primero a `SaveAsync` con el `ArenaRecord` terminado y espera la confirmación. Conserva ese mismo registro si hay que reintentar. Puedes bloquear el botón mientras esperas y mostrar «Pendiente» o «Confirmado». La transacción protege el dato incluso si dos peticiones llegan juntas.

Para desplegar Cloud Functions en la nube, Firebase exige el plan Blaze. La prueba de este capítulo se ejecuta en local. Usa las insignias para ensayar el flujo. Una recompensa con valor económico necesitaría evidencia de partida generada por un sistema en el que puedas confiar.

Funciones invocables desde una app · Desplegar y probar Cloud Functions

CAPÍTULO 10

Una clasificación que dice de dónde salen sus puntos

Una tabla con nombres y números parece muy oficial. Podríamos añadirle una corona y un borde dorado. Eso seguiría sin responder la pregunta importante: ¿quién decidió que esos puntos eran válidos?

Monta primero la clasificación de práctica

Usamos `leaderboards/{missionId}/scores/{uid}`. Cada jugador tiene una fila por misión. Cuando llega un resultado mejor y la misión está activa, `recordRun` actualiza la fila en la misma transacción que guarda el récord. Si el resultado es menor, conserva el anterior. Las filas del laboratorio llevan `source: "practice"`.

```
var board = db.Collection("leaderboards")
    .Document(missionId)
    .Collection("scores");

var firstPage = await board
    .OrderByDescending("score")
    .Limit(20)
    .GetSnapshotAsync();
```

El límite hace predecible el tamaño de la consulta y coincide con la regla que permite leer la clasificación. El cliente recibe los ID de los documentos, que aquí son UID de prueba. Si quieres ocultar esos ID en un juego público, publica una vista separada con alias. Define un criterio de desempate y qué ocurre si cambia la primera página entre dos consultas.

Elige dónde confiar

El móvil calcula el resultado de Arena Fénix. Una persona puede modificar ese cliente o enviar peticiones directamente. Que el servidor reciba un número dentro del rango esperado solo demuestra que ese número tiene forma válida. App Check ayuda a limitar ciertos accesos, y sigue haciendo falta verificar lo que ocurrió en la partida.

Para una competición real, decide antes de crear la tabla quién puede certificar un resultado. Por ejemplo, una simulación autoritativa puede calcular la puntuación en un servidor de juego. Para una prueba asincrónica, el servidor puede asignar un reto y verificar una reproducción si la mecánica se diseñó para ello. Si tu prototipo todavía no puede comprobarlo, llama a la tabla «clasificación de práctica». En el libro esa etiqueta es parte del diseño.

Practica dos consultas y un fraude evidente

1. Registra dos resultados de la misma sesión con IDs distintos. El inferior no desplaza al mejor.
2. Consulta con `Limit(20)`. Repite sin límite. El verificador comprueba que las reglas rechazan la segunda consulta.
3. Intenta escribir directamente el récord desde el cliente. Aunque la sesión sea tuya, la regla solo te permite leerlo.
4. Prueba con una puntuación absurda a través de la función. La validación rechaza el rango. Pregúntate qué clase de puntuación falsa sí podría pasar. Esa respuesta indica dónde necesitas otra capa.

CAPÍTULO 11

Realtime Database y una sala de espera

En la arena alguien pulsa **Listo**. En la pantalla de otra persona debería verse en seguida. También debería desaparecer la presencia si alguien cierra el juego sin despedirse. Para esta práctica usamos Realtime Database y reservamos Firestore para el catálogo y el progreso.

Separa la intención de la conexión

Dentro de `rooms/{room}/members/{uid}` guardamos `ready` y un mapa `connections`. El primer valor es lo que el jugador ha decidido. El mapa indica cuántas conexiones siguen presentes. Dos ventanas con la misma cuenta pueden compartir el estado de listo y mantener conexiones diferentes.

```
rooms/partida-1/members/{uid}
  ready: true
  connections/{connectionId}: true
```

En el ejemplo, `FirebaseArenaSource.CreateRoomAsync` devuelve una instancia de `ArenaRoom`. Llama a `Begin` al abrir el vestíbulo, a `SetReadyAsync` al cambiar el botón y a `LeaveAsync` al salir. Al conectarte, observa `/.info/connected`. Cuando sea verdadero, crea un hijo con ID nuevo y registra primero la retirada con `OnDisconnect`. Solo después escribe `true` en ese hijo. Si se corta la red antes de registrar la retirada, podrías dejar un fantasma en el vestíbulo. En este juego preferimos que los fantasmas sean una decisión artística.

Prueba con dos ventanas y una salida brusca

El verificador local cambia `ready` con una sesión propia y comprueba que otra no puede escribirlo. Para probar presencia, conecta `ArenaRoom` desde tu propia interfaz de vestíbulo. En dos sesiones con el mismo UID, abre la misma sala y mira `connections` en Emulator UI. Cierra una sesión. Cuando el emulador detecte la desconexión, debería retirar solo ese hijo. El botón **Listo** expresa una intención. Para mostrar «listo y conectado», comprueba también que quede alguna conexión.

El ejemplo de reglas da lectura de la sala a usuarios autenticados que conocen su ID. Es adecuado para una prueba local con datos ficticios. Para salas privadas, añade invitaciones o membresía verificadas por el servidor. Quien adivine el nombre de una sala de prueba no debería descubrir datos personales.

El capítulo enseña coordinación de vestíbulo. Una partida de acción añade latencia, predicción, corrección de movimiento y un sistema que decide qué ocurrió realmente. Usa un servicio de red de juego para esa parte si tu diseño lo requiere.

Realtime Database en Unity · Presencia y operaciones al desconectar

CAPÍTULO 12

Reglas de seguridad, App Check y pruebas de acceso

Ya sabemos abrir la puerta. Ahora toca intentar abrir todas las que deberían permanecer cerradas. El objetivo de este capítulo es que puedas demostrar quién lee qué, quién escribe y qué ocurre cuando la aplicación envía una petición inesperada.

Tres comprobaciones distintas

Pregunta	Herramienta en la práctica
¿Quién está haciendo la petición?	Authentication aporta el UID de la sesión.
¿Puede esa sesión acceder a ese dato?	Security Rules y las funciones deciden según la ruta, el propietario y el rol.
¿La petición viene de una app que supera la verificación configurada?	App Check aporta una señal de verificación del cliente.

Las reglas de Firestore del proyecto permiten leer un perfil a su propietario y reservan las escrituras sensibles para las funciones. Esas funciones usan Admin SDK. Las reglas de cliente no se aplican al Admin SDK, de modo que la función comprueba identidad, permiso, tipos y rangos antes de escribir.

Lee una regla con un ejemplo concreto

```
function owns(uid) {
  return request.auth != null && request.auth.uid == uid;
}

match /players/{uid} {
  allow get: if owns(uid);
}
```

La ruta solicita un perfil específico. Si la sesión de Lucía pide el perfil de Mario, el UID de la sesión y el de la ruta son distintos. El acceso falla. En esta regla no hay permiso de escritura. No añadas uno genérico para que pase una prueba que todavía no sabes explicar.

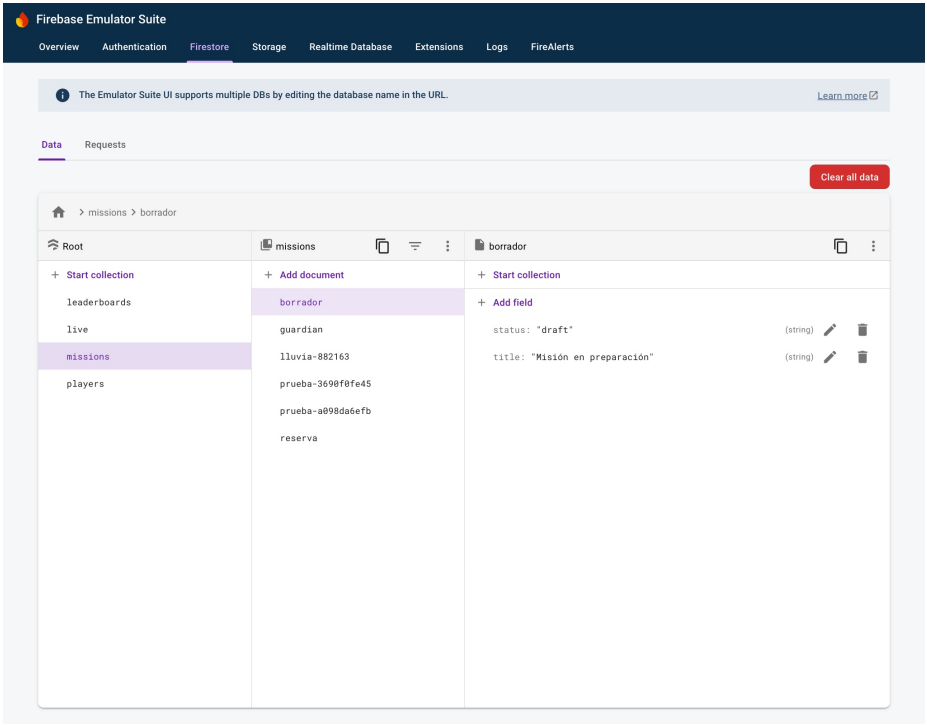


Figura 17. Captura del emulador. Firestore muestra el catálogo y las colecciones usadas por los casos de acceso de Arena Fénix.

Haz una ronda de intentos

El script del proyecto inicia sesiones ficticias y llama a los emuladores. Comprueba que un invitado lee el contenido publicado, que el borrador permanece privado, que un usuario no lee el perfil de otro y que una sesión normal no publica una misión. También verifica que la cuenta editorial sí puede hacerlo. Cada caso permitido tiene cerca un caso que debe fallar.

Para App Check en una compilación móvil, configura el proveedor que corresponde a la plataforma y al canal de distribución. Registra un proveedor de depuración únicamente para la práctica. Mira primero las métricas de solicitudes válidas y no verificadas. En las funciones invocables de segunda generación, el bloqueo se activa con `enforceAppCheck: true` en las opciones de `onCall`. Las funciones locales del libro prueban la autorización por UID y rol. Ensaya el bloqueo en tu proyecto de pruebas antes de aplicarlo en el juego público.

Reglas y autenticación · App Check para Unity · App Check en funciones invocables

PARTE IV

Aprende de las partidas



Figura 18. Ilustración. Desde la torre, el guardián observa lo que hacen los aventureros en la arena.

Convertirás las preguntas sobre el juego en eventos, experimentos y pruebas. También prepararás el recorrido de una versión desde la beta hasta el lanzamiento.

CAPÍTULO 13

Analytics y la primera oleada que nadie acaba

A tu mejor amigo le parece que la primera oleada es facilísima. Tu mejor amigo también te ha visto programarla y sabe exactamente dónde aparecerán los enemigos. ¿Cómo compruebas si una persona que acaba de instalar el juego entiende qué hacer?

Escribe la pregunta antes de elegir eventos

Nuestra pregunta es «¿qué proporción de las instalaciones que empiezan su primera partida y permiten medirla supera la primera oleada de esa misma partida?». Marca la primera partida al empezar y conserva esa condición mientras dura la ronda. Si alguien pierde y vence en la segunda, solo entra en el denominador de este embudo.

Evento propuesto	Momento de emisión	Datos útiles
`arena_first_run`	Al comenzar la primera partida de una instalación de prueba	Versión de reglas y versión del juego.
`arena_first_wave_clear`	Al completar la primera oleada de esa primera partida	Versión de reglas y número de enemigos.
`arena_run_end`	Cuando termina una partida	Oleadas completadas, duración agrupada y resultado.

Una instalación no equivale automáticamente a una persona. Define la unidad que puedes medir, y usa el mismo criterio en el numerador y el denominador. Para un embudo básico, compara a quienes iniciaron la práctica en la misma ventana. Evita usar un recuento

bruto de eventos si un jugador puede emitir el mismo evento varias veces.

Haz una sesión de prueba de principio a fin

1. Habilita Analytics en el proyecto de pruebas y sigue `RECETAS_MOVILES.md` para conectar los eventos a la arena. Configura el estado inicial de recopilación antes de distribuir la app. Usa eventos sin correos, nombres reales, mensajes de chat ni identificadores personales.
2. En una compilación de desarrollo Android, activa el modo de depuración de Analytics para **tu identificador de paquete**. Usa el comando del manual, con el paquete de Arena Fénix solo si no lo has cambiado.
3. Abre **DebugView**, inicia la arena, supera la primera oleada y termina la partida. Deberías ver los eventos ordenados, con valores compatibles con lo que aparece en el HUD.
4. En otra instalación de prueba, pierde la primera partida y completa la oleada en la segunda. El evento de éxito de la primera partida debe quedar ausente. Al acabar, desactiva el modo de depuración.

```
adb shell setprop debug.firebase.analytics.app \
  net.davidcanton.arenafenix
```

Al acabar, ejecuta `adb shell setprop debug.firebase.analytics.app .none.`.

DebugView comprueba que llegan los eventos. Para medir el juego necesitas participantes y una ventana de observación. Si falta el evento de inicio, comprueba la instrumentación antes de calcular el porcentaje de éxito.

Reto. Añade una señal de «primer pulso acertado» y explica qué pregunta sobre el juego te permite responder.

Eventos de Analytics en Unity · DebugView

CAPÍTULO 14

A/B Testing y dos maneras de empezar

La prueba local decía que cinco ascuas resultaban divertidas. ¿También para quien todavía está aprendiendo a moverse? Vamos a poner una idea a prueba, conservando una variante de referencia y una pregunta que se pueda responder.

Prepara la hipótesis

«Con tres enemigos en la primera oleada, más jugadores nuevos la completarán sin que aumente el abandono temprano». Define antes de lanzar el experimento:

- **Referencia y variante.** Cinco enemigos por oleada frente a tres. Mantenemos constantes el daño y el tiempo de recarga. El parámetro del ejemplo afecta a todas las oleadas. Podemos medir la primera por separado.
- **Métrica principal.** Proporción de nuevos jugadores de la prueba que supera la primera oleada.
- **Señal de vigilancia.** Primeras partidas con inicio y sin final registrado en un plazo fijo. Es una aproximación al abandono. Un cierre o la pérdida de la conexión también pueden dejar un evento sin registrar.
- **Ventana y público.** Elige una duración, las versiones compatibles y qué usuarios de prueba pueden participar. Consulta las estimaciones y límites de Firebase al configurarlo.

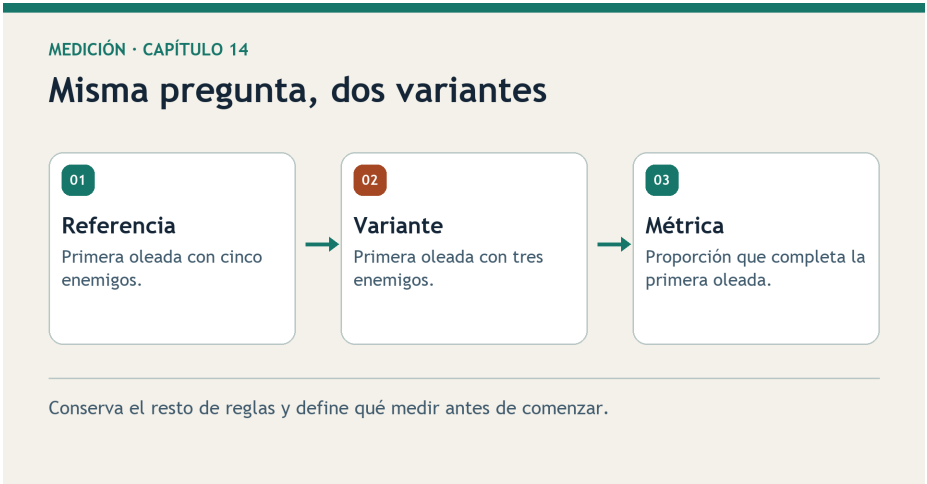


Figura 19. Diagrama. Dos grupos reciben una variante, completan la misma práctica y se comparan con la misma métrica.

Ensaya el flujo antes de interpretarlo

En Firebase, crea un experimento con Remote Config y Analytics. Selecciona `arena_enemies_per_wave` y las dos variantes. Usa la instalación de ensayo para verificar que el valor ya está activo al iniciar la primera partida medida. Si defines un evento de activación para el experimento, emítelo después de aplicar la variante. Comprueba los eventos y el número de enemigos en el HUD.

Con dos instalaciones de prueba puedes demostrar que ambas variantes funcionan. Para afirmar que una funciona mejor necesitas una muestra y un periodo adecuados. Evita cerrar la prueba en la primera tarde porque apareció un resultado que te gustó. Si la variante rompe el juego, el criterio es otro. Detén o revierte para proteger la experiencia y registra el motivo.

Experimento o despliegue gradual

Un **experimento** compara una hipótesis. Un **despliegue gradual** reduce a cuántas personas afecta un cambio mientras observas fallos y decides ampliar. En la operación final del libro usaremos la segunda idea para activar un evento, y A/B Testing para probar una pregunta de diseño. La diferencia es lo que decides a partir del resultado.

Aprende a explicar un resultado pequeño

Supón que el grupo de referencia tuvo 12 instalaciones de prueba y que 7 completaron la primera oleada. En la variante completaron 8 de 12. El resultado sirve para practicar el cálculo y ver si ambos grupos registran el evento. Una diferencia de una persona en un grupo tan pequeño deja muchas explicaciones abiertas. Documenta la prueba como comprobación del sistema y reserva la conclusión de diseño para una muestra adecuada.

Experimentos con Remote Config · Despliegues graduales de Remote Config

CAPÍTULO 15

Cloud Messaging y una invitación que llega a tiempo

Tu evento empieza hoy. Envías una notificación. Una persona la abre mañana, cuando el evento ya ha terminado. ¿Qué ve? Si la respuesta es una pantalla vacía, acabas de enviarle una invitación a admirar un fondo.

La notificación lleva una pista

Para un aviso de prueba, usa un título, un texto corto y un identificador de misión como dato. Unity recibe el mensaje y decide qué pantalla puede abrir. A continuación consulta el catálogo y comprueba si la misión sigue activa. Si el mensaje dice `fin-de-semana` y ese evento acabó, ofrece el reto disponible en ese momento.

```
{
  "type": "arena_event",
  "missionId": "fin-de-semana"
}
```

Esa carga indica el destino. No contiene la recompensa, las reglas del evento ni autorización. El catálogo y la función siguen teniendo la última palabra. Evita enviar datos privados en un canal de notificaciones.

Prueba los tres estados del juego

1. Integra Cloud Messaging en una aplicación de pruebas. Usa Android con servicios de Google compatibles o prepara APNs en iOS. En Unity, registra `TokenReceived` y `MessageReceived` y comprueba la actividad de Android que recibe los datos al tocar un aviso. Guía de Cloud Messaging para Unity

2. Pide permiso en el momento en que el jugador entiende para qué sirven los avisos. Comprueba qué ve si elige que no.
3. Envía desde la consola una notificación a la instalación de laboratorio e incluye `type` y `missionId` como datos. Comprueba primer plano, segundo plano y apertura al tocar el aviso. Para esta prueba cierra la app normalmente. Forzar su detención es otra prueba y puede cambiar la entrega.
4. Expira la misión y abre otra vez el aviso. Comprueba que ofrece una salida útil.

Guarda las credenciales para enviar mensajes en el servidor. Un tema de FCM ayuda a agrupar avisos, pero el acceso a una recompensa sigue dependiendo de la cuenta y del estado del servidor. Si el jugador retira el permiso o quiere menos avisos, el juego debe respetar su elección.

CAPÍTULO 16

Crashlytics y una pista en medio del humo

Alguien te dice «se me ha cerrado el juego». Tú respondes «¿cuándo?». «Cuando estaba jugando». Suena gracioso hasta que te toca reproducirlo. Vamos a preparar un informe que sí diga dónde empezar a mirar.

Registra lo que ayuda a reproducir

El informe necesita la versión de la aplicación y el contexto técnico. En la arena interesan la revisión de reglas, el ID de catálogo y si estábamos en el vestíbulo o en combate. Evita guardar el contenido de una cuenta, tokens o mensajes escritos por jugadores.

```
Crashlytics.SetCustomKey("arena_rules_revision", "2");
Crashlytics.SetCustomKey("arena_screen", "lobby");
Crashlytics.Log("Catálogo de temporada activado");
```

En el código real, pasa los valores activos en lugar de esas cadenas de muestra. La idea es que cuando aparezcan dos informes puedas distinguir una versión mala de un evento mal configurado.

Provoca y recupera un informe de prueba

1. Integra el paquete de Crashlytics en una compilación móvil de desarrollo y sigue la configuración de símbolos de tu plataforma. Los símbolos ayudan a que el informe señale funciones reconocibles.
2. Registra un error controlado en el entorno de ensayo. Si haces la prueba de cierre fatal propuesta por la documentación, sitúala detrás de una condición de desarrollo y ejecútala solo en el dispositivo de pruebas.

3. Abre de nuevo el juego para permitir el envío del informe. En la consola, busca el problema y comprueba la versión y las claves.
4. Quita el disparador de prueba antes de crear una compilación para otras personas.

Compara qué puedes investigar con una traza que solo dice «objeto nulo» y otra que añade pantalla, versión y catálogo. En una incidencia real, decide si desactivar el contenido ayuda mientras preparas una corrección de código.

Crashlytics para Unity · Forzar un cierre de prueba

CAPÍTULO 17

App Distribution y Test Lab para salir del propio ordenador

En tu Mac todo va bien. En el teléfono de alguien, el botón del pulso se queda justo debajo de un recorte de pantalla. Conviene encontrar esas diferencias antes de que el evento anuncie a todo el mundo que ya puede venir.

Prepara una compilación para una beta

En Unity selecciona el perfil de Android y genera un APK de pruebas con identificador y versión visibles. La ruta del libro usa un proyecto de desarrollo y datos ficticios. Antes de enviar un enlace, instala la compilación en un emulador o dispositivo que controles y comprueba que abre.

En App Distribution, sube la versión, escribe qué deben probar las personas invitadas y crea un grupo pequeño. Incluye instrucciones concretas. «Prueba el juego» provoca comentarios vagos. «Pierde la conexión al volver del combate y dime qué dice el menú» suele dar una pista que puedes repetir.

Pasa una prueba corta en más de un dispositivo

Usa Test Lab para comprobar la instalación, el arranque y la estabilidad en una matriz de dispositivos cuando tu proyecto lo permita. Una exploración automatizada puede encontrar un cierre. Para un juego, prepara también pasos de entrada y expectativas claros. Los controles dibujados por el motor pueden necesitar una prueba diseñada para la escena.

Caso	Resultado esperado
Inicio sin red	Aparece el menú y puede empezar una partida local.
Pantalla alargada	Los botones quedan dentro del área visible.
Vuelta desde segundo plano	El juego conserva o recupera un estado comprensible.
Reto vencido	El vestíbulo ofrece una misión válida.
Descarga incompleta	Sigue disponible la apariencia base.

Decide si la versión puede salir

Separa errores que impiden jugar de detalles que puedes arreglar más tarde. Para cada bloqueo registra dispositivo, versión, pasos y resultado. Para una corrección, repite el mismo caso y prueba que todavía funcionan el inicio y el final de partida.

Una prueba de laboratorio aporta cobertura. Una persona jugando aporta contexto. Usa ambos. Al cerrar la beta, conserva el informe y la compilación que pasó las pruebas. Si publicas cambios después, sabrás qué versión recuperar.

App Distribution en Android · Firebase Test Lab

PARTE V

Abre las puertas del evento



Figura 20. Ilustración. El guardián y los aprendices llegan al evento mientras el fénix sobrevuela la arena.

Añadirás un comentario del fénix, estudiarás una recompensa opcional y ensayarás una temporada completa, incluida la forma de volver atrás.

CAPÍTULO 18

Firebase AI Logic y el fénix comentarista

Al fénix le cuesta aceptar que un guardián de medio metro le gane con un pulso. Podemos darle una frase al final de la partida. Si perdió, quizá diga «tenías un plan. Yo tenía plumas». Si tarda la red, el jugador seguirá viendo su puntuación y leerá una frase local.

Dale un trabajo pequeño a la IA

Nuestro NPC comenta tres números del juego. No necesita acceder al correo, nombre, chat ni inventario del jugador. Tampoco decide si ha ganado ni concede objetos. Diseña la tarea para que, aunque la respuesta cambie, el juego sepa qué hacer con ella.

```
Eres el fénix de una arena de aventuras.
Comenta una partida en una frase amable y breve.
Oleadas: 2. Puntos: 170. Victoria: no.
```

En Firebase AI Logic elige un modelo disponible para la plataforma y el proyecto. El nombre del modelo puede cambiar. Usa el selector de modelos de Firebase cuando prepares la práctica. Si permites cambiar ese nombre con Remote Config, mantén una lista de modelos compatibles y un valor incluido.

Haz una llamada de prueba

1. En un proyecto de pruebas, activa Firebase AI Logic y Gemini Developer API, el proveedor que utiliza el código del capítulo. Comprueba disponibilidad, cuota y facturación. Instala `FirebaseAI` y configura App Check para la ruta móvil.
2. Crea una instancia de `FirebaseAI` y pide un modelo generativo. Genera el texto una vez por resultado, al abrir el resumen. Pasa solo los números que ya calculó la arena.

3. Limita la espera y la longitud del resultado. Si la respuesta está vacía, se bloquea o el servicio tarda más de lo que permite la pantalla, muestra el texto incluido. Una validación de longitud ayuda a la interfaz. La moderación y la elección de contenido adecuado para tu público son decisiones aparte.

```
var ai = FirebaseAI.GetInstance(
    FirebaseAI.Backend.GoogleAI());
var model = ai.GetGenerativeModel(
    modelName: modeloDePrueba);
var response = await model.GenerateContentAsync(
    prompt, cancellationToken);
var line = response.Text;
```

No pongas una clave privada del proveedor en el ejecutable. Haz también una prueba con el servicio apagado. El resultado principal, que son las oleadas y los puntos, debe seguir disponible. Un fénix que se queda sin comentario es gracioso. Una pantalla de resultados que se queda pensando durante un minuto lo es bastante menos.

Firebase AI Logic en Unity · Preparación para producción

CAPÍTULO 19

AdMob y una recompensa opcional

Imagina que el menú ofrece «Ver un anuncio para estrenar una pose del guardián». El botón solo aparece en la prueba cuando hay un anuncio disponible. El jugador puede continuar con su partida. Nos interesa practicar cómo cambia el estado desde que se carga el anuncio hasta que se confirma el premio.

Construye el flujo con anuncios de prueba

1. En una compilación separada, instala el plugin Google Mobile Ads para Unity. Añade el identificador de aplicación de prueba y usa la unidad de anuncios recompensados de prueba de la documentación.
2. Resuelve primero las opciones de privacidad y la elegibilidad de la app. Inicializa la biblioteca una vez cuando puedas pedir anuncios y usa `RewardedAd.Load` con la unidad de prueba. Si falla la carga, deja la partida disponible y limita los reintentos.
3. Cuando `CanShowAd()` sea verdadero, habilita el botón. En `Show`, usa el callback de recompensa como señal de que se ha completado el flujo. Cerrar el anuncio y recibir una recompensa son eventos diferentes.
4. Tras cerrar o descartar el anuncio, libera el objeto y prepara el siguiente cuando proceda. Un anuncio cargado se muestra una sola vez.

```
if (ad != null && ad.CanShowAd())
{
    ad.Show(reward => ConfirmarPremioDePrueba());
}
```

`ConfirmarPremioDePrueba` representa tu implementación. En el ejercicio desbloquea una pose local ficticia. Si el premio importa para el inventario, estudia la verificación del servidor de AdMob y procesa su identificador de transacción una sola vez. Es el mismo principio de la insignia del capítulo 9 aplicado a otra señal.

Decide dónde tiene sentido la opción

Ensaya cargar, cerrar sin completar, completar y quedarse sin red. Mide si el botón añade algo a la experiencia y si alguien puede entender la elección sin leer un manual. Para juegos dirigidos a menores, revisa requisitos de la tienda, configuración de anuncios y tratamiento de datos con quien vaya a publicar el juego. El ejemplo principal del libro se puede jugar sin publicidad.

Anuncios recompensados para Unity · Verificación del lado del servidor

CAPÍTULO 20

El evento de fin de semana

Ya puedes pasar del «tenemos un juego» al «este sábado tenemos algo que hacer dentro del juego». Vamos a montar una edición de **Fin de semana en llamas** y a ensayar también el cierre. El fénix está listo. El calendario debería estarlo también.

Prepara una ficha de publicación

Decisión	Valor del ensayo
Objetivo	Superar dos oleadas.
Reglas	Revisión 2. Cinco enemigos por oleada, pulso cada 0,85 segundos.
Misión	ID propio, una ventana UTC y cliente mínimo 1.
Apariencia	Escudo de temporada, ruta versionada y alternativa incluida.
Premio	Insignia de práctica concedida una vez por jugador y misión.
Reversión	Recuperar `guardian` y las reglas base.

Arranca las dos fuentes locales

Arranca y siembra los emuladores como en el README. En otro terminal, ejecuta `node lab.mjs`. Abre el laboratorio en el puerto 5177 y el panel de operaciones en el 5100. Usa una compilación macOS con el SDK instalado. Desde la carpeta de la app, abre Unity con ambos argumentos.

```
open -n -a ./ArenaFenix.app --args \
  -arena-firebase-test emulator \
  -arena-lab http://127.0.0.1:5177/current.json
```

El HTTP del laboratorio solo aporta las reglas en este recorrido. Authentication, Firestore, Functions y Storage vienen de Firebase Emulator Suite. Verás «Firebase local + laboratorio» en el vestíbulo. El escudo ya está cargado por el sembrado.

Haz un ensayo con un compañero o dos sesiones

1. En el laboratorio elige las reglas base y confirma `guardian` en el panel de operaciones. Abre una partida. El HUD indica revisión `1` y tres enemigos por oleada.
2. Publica **Lluvia de ascuas** con un ID nuevo y el escudo marcado. En el laboratorio pulsa **Activar fin de semana**. La partida abierta conserva su objetivo y reglas. La próxima partida debe mostrar la misión nueva, revisión `2` y cinco enemigos. Si quieres ver el escudo antes de entrar, vuelve a abrir la app con los mismos argumentos después de publicar.
3. Con la misión de dos oleadas activa, ejecuta `node functions/test-local.mjs --mision-activa` desde `firebase-demo`. El verificador registra un resultado ficticio y reclama la insignia dos veces. Espera `claimed` y `alreadyClaimed`. Si ya añadiste el botón de Unity, repite el ensayo con una partida que hayas jugado.
4. En el panel de operaciones recupera `guardian` y en el laboratorio activa **Recuperar reglas base**. Empieza otra partida y verifica que han vuelto las tres ascuas. Si cortas el laboratorio, la compilación mantiene las últimas reglas leídas durante esa ejecución.

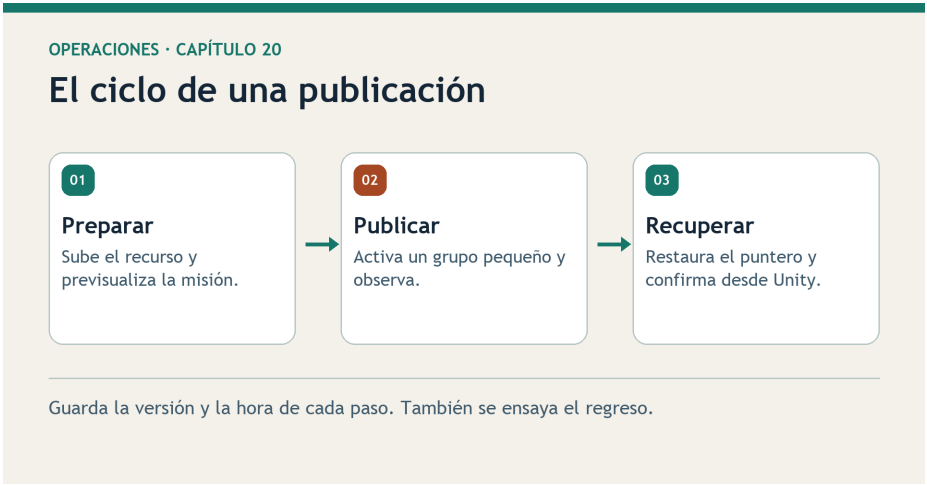


Figura 21. Diagrama. Publicar, comprobar con un grupo pequeño y recuperar la versión anterior forman el mismo recorrido de operaciones.

Reúne las pruebas de que funcionó

Anota la revisión del juego, el ID de misión, la versión de reglas y la hora de cada comprobación. Guarda una captura del vestíbulo y otra de la primera oleada. En producción, los dos cambios viajan por vías distintas. Antes de abrir el evento a más gente, comprueba desde un cliente de prueba que ha recibido una combinación válida.

Ensaya una retirada con un motivo concreto

Imagina que el catálogo está bien y que el escudo descarga, pero la primera oleada es demasiado difícil. Devuelve primero la dificultad a su valor anterior. Comprueba el efecto en una partida nueva y conserva la misión si su objetivo sigue siendo alcanzable. Ahora imagina que el enlace del paquete está roto. Recupera la misión anterior o publica una que no dependa del archivo mientras arreglas el paquete.

Al recuperar `guardian`, el evento anterior conserva su plazo de reclamación. Para suspender sus premios, cierra la ventana en el servidor y avisa a quienes ya estaban jugando.

CAPÍTULO 21

Mantén la arena cuando crece

El evento ha salido bien. Ahora llega el trabajo que permite repetirlo sin convertir todos los viernes en una emergencia. Pon nombre a los costes, mide el estado del juego y deja escritas las decisiones de publicación.

Calcula de qué depende el consumo

Para empezar, cuenta operaciones. Si mil jugadores abren el vestíbulo dos veces y cada apertura recupera dos documentos, el escenario supone unas cuatro mil lecturas. Es un cálculo de diseño. La facturación real depende del servicio, región, plan y comportamiento de consultas, caché y escuchas.

Un archivo de dos megabytes descargado por mil dispositivos supone alrededor de dos gigabytes transferidos en el escenario simple. Versionar el archivo y conservarlo en caché permite evitar descargas repetidas. Si el evento trae diez archivos, anota qué paquetes se descargan antes de jugar y cuáles pueden esperar.

Recurso	Qué observar	Decisión de diseño
Firestore	Lecturas y consultas por visita	Leer al entrar y limitar resultados.
Storage	Descargas y bytes transferidos	Versionar y almacenar recursos en caché.
Functions	Invocaciones, errores y duración	Validar temprano y reusar el ID al reintentar.
Remote Config	Recuperación y revisión activa	Mantener valores incluidos y cambios compatibles.
Analytics y Crashlytics	Señales útiles y datos sensibles	Instrumentar con un propósito y definir retención.

Consulta los precios y cuotas de cada servicio al pasar a producción. Las alertas avisan cuando el gasto cruza un umbral y los servicios pueden seguir activos. Para reducir el consumo, define tamaños máximos, frecuencia de peticiones y una forma de desactivar cada función costosa.

Conserva un procedimiento de operaciones

Registra el responsable, la fecha y la forma de revertir cada publicación. Si eliminas una cuenta, borra su perfil, `runs`, `claims`, sus filas de clasificación y las copias de otros servicios. Firestore conserva las subcolecciones al borrar el perfil. Borrar datos en Firestore

Tu último reto. Para tu próximo juego, escribe tres frases. «En marcha puedo cambiar…». «Sin red el jugador todavía puede…». «El servidor siempre comprueba…».

Precios de Firebase · Límites y cuotas de Firestore

Apéndices

Abre estas páginas cuando te atasques

Si estás eligiendo una base de datos, empieza por el mapa de servicios. Si una palabra del capítulo te ha hecho ir tres veces hacia atrás, búscala en el glosario. Si algo funciona solo en tu ordenador, abre el cuaderno de comprobaciones y anota el estado de partida. Y si estás preparando una copia limpia del ejemplo, consulta los recursos y versiones probadas.

Puedes volver a la arena en cualquier punto. Para comparar una práctica nueva con el comienzo, desactiva la conexión en el Inspector y vuelve a jugar con las tres ascuas del primer capítulo. Es una referencia sencilla, que siempre está en el proyecto.

APÉNDICE A

Qué servicio elegir

Empieza por el dato que necesitas mover. Si es un valor pequeño que decide cómo se ejecuta una mecánica, ve a Remote Config. Si son fichas con campos y relaciones entre documentos, Firestore. Si quieres mostrar conexiones de una sala, Realtime Database. Para archivos, Storage o un alojamiento compatible con la estrategia de descarga de Unity.

Servicio	Uso en videojuegos	Recorrido del libro
Authentication	Invitados, vinculación e identidad	SDK de Unity y emulador de Auth.
Cloud Firestore	Perfiles, catálogo y clasificación	SDK de Unity, reglas, consultas y transacciones.
Realtime Database	Presencia y salas	Modelo de conexiones y reglas de propietario.
Remote Config	Dificultad, reglas y activación gradual	Laboratorio y plantilla de consola.
Cloud Storage	Imágenes y paquetes	Archivo versionado en el emulador.
Cloud Functions	Premios, validación y publicación	Funciones invocables y transacciones locales.
Hosting	Panel web	Panel probado con Firebase Emulator Suite.
App Check	Verificación de cliente	Práctica de configuración y plan de cumplimiento.
Analytics	Embudos y eventos	Ejercicio de instrumentación móvil y DebugView.

Servicio	Uso en videojuegos	Recorrido del libro
A/B Testing	Comparar variantes	Diseño de una prueba de dificultad con Remote Config.
Cloud Messaging	Avisos de eventos	Enrutado por ID de misión y control de vigencia.
Crashlytics	Diagnóstico de fallos	Contexto técnico y ejercicio móvil de prueba.
App Distribution	Entregar una beta	Pasos para compilar, compartir y recoger casos.
Test Lab	Probar en dispositivos	Matriz de casos y estrategia de lanzamiento.
Firebase AI Logic	Comentario breve del NPC	Ejercicio opcional con respuesta local de respaldo.
AdMob	Anuncios recompensados	Plugin separado y anuncio de prueba.

Servicios que pueden encajar alrededor del juego

Firebase SQL Connect. Útil si tu panel requiere consultas relacionales, por ejemplo temporadas, recompensas y relaciones de catálogo en PostgreSQL. Para este libro queda del lado web o servidor. Comprueba los SDK que ofrece antes de diseñar una integración con Unity. SQL Connect

App Hosting. Opción para un panel web que necesite ejecución de servidor y una cadena de despliegue web. El panel sencillo del laboratorio sirve archivos estáticos desde Hosting y llama a Functions. App Hosting

Extensions. Pueden automatizar tareas concretas alrededor del backend. Revisa qué recursos crean, con qué permisos y en qué región. Para una tabla de clasificación, conserva un flujo de escritura que puedas explicar y auditar. Extensions

In-App Messaging y Performance Monitoring. Comprueba la matriz actual de SDK antes de comprometerte a integrarlos en Unity. Para mostrar avisos dentro de Arena Fénix, podemos leer una ficha desde Firestore y dibujarla en el menú. Para tiempos de carga, medimos desde que empieza la descarga hasta que Unity presenta el resultado. El catálogo Firebase incluye herramientas con soporte distinto según la plataforma. Productos de Firebase · Matriz de Unity

BigQuery. Permite explorar datos exportados con más detalle cuando el proyecto lo requiere. Una primera pregunta se puede responder con el embudo de Analytics. Antes de exportar, define la necesidad y la retención. Exportación a BigQuery

Dynamic Links. La documentación de Firebase fija el cierre del servicio el 25 de agosto de 2025. Si mantienes un juego antiguo, revisa su recorrido de invitaciones y migra enlaces según las necesidades de tu plataforma. Migración de Dynamic Links

APÉNDICE B

Glosario de la arena

Adaptador. Pieza que transforma la forma de hablar de un servicio en la interfaz que espera tu código.

Asset. Recurso que utiliza el juego. Puede ser una textura, un prefab, audio o una escena.

Catálogo remoto. Índice que relaciona identificadores de contenido con los recursos que puede descargar un cliente.

Claim. Atributo de identidad que la aplicación puede comprobar. También usamos «reclamación» para el documento que registra un premio concedido. Son conceptos distintos.

Compilación. Versión ejecutable de un juego generada a partir de su proyecto. También se usa la palabra inglesa *build*.

Emulador. Herramienta que reproduce un dispositivo o un servicio para hacer pruebas en un entorno controlado.

Estado autoritativo. Estado que determina el resultado aceptado por el sistema. En una competición se suele calcular fuera del cliente.

Idempotencia. Propiedad de una operación que permite repetirla sin acumular un segundo efecto.

Oleada. Grupo de enemigos que entra en la arena antes de que empiece el siguiente turno.

Parámetro remoto. Valor que la app consulta fuera del ejecutable y valida antes de usarlo.

Puntero. Dato pequeño que dice qué documento o versión usar.

Reversión. Volver a una configuración o catálogo anterior y verificar que el juego lo usa.

Sidecar. En este libro, manera de describir los servicios que acompañan al juego y se conectan mediante una interfaz. La simulación sigue viviendo en Unity.

UID. Identificador que Firebase Authentication asigna a la cuenta. Sirve para autorizar operaciones y relacionar datos con el jugador.

APÉNDICE C

Cuaderno de comprobaciones

Hoja de una práctica

Qué anotar	Ejemplo
Código y versión	Unity 6, parche exacto y commit o versión del ZIP.
Entorno	Local, emulador de servicio, emulador Android o consola remota.
Estado inicial	Reglas v1, misión Guardia de la arena.
Cambio realizado	Elegir el escenario de fin de semana.
Prueba	Empezar una partida y contar enemigos.
Resultado	Cinco enemigos, revisión 2, misma compilación.
Reversión	Restaurar el escenario base y volver a contar.

Preguntas para comprobar si lo has entendido

1. La misión cambia de nombre a mitad de partida. ¿Qué debería hacer el HUD actual? ¿Dónde has guardado la copia?
2. Llegan dos peticiones con el mismo ID y puntuaciones diferentes. ¿Cuál es el resultado que puedes aceptar y cómo lo identificas?
3. Tu equipo quiere subir una animación nueva. ¿Cuándo basta publicar contenido remoto y cuándo hace falta compilar?
4. Un usuario que ya inició sesión puede leer su perfil. ¿Qué prueba demuestra que no puede leer el de otra persona?

5. Si un despliegue gradual produce muchos fallos, ¿cómo distingues un problema de reglas, catálogo o ejecutable?

Pistas. Separa estado activo de estado candidato, usa una clave por operación, comprueba qué código conoce la compilación, prueba con dos identidades y conserva la revisión junto al informe del fallo.

APÉNDICE D

Recursos para seguir

Proyecto del libro. Descarga los ejemplos desde davidcanton.net/unity-firebase/. El descargable contiene `ArenaFenix`, `firebase-demo`, las instrucciones de reproducción y `RECETAS-MOVILES.md` con fragmentos para los capítulos avanzados. Las pruebas distinguen la arena local, el adaptador HTTP, el SDK y los emuladores. Usa la carpeta `evidencia` para consultar qué ejecución respaldó cada captura.

Documentación viva. Primeros pasos de Firebase en Unity · Firebase para videojuegos · Notas de versión del SDK de Unity · Manual de Unity

Continúa con tu propio juego. Si quieres practicar el editor, el movimiento y las escenas antes de ampliar el sidecar, en Curso práctico con Unity 3D tienes los ejemplos del libro anterior.

APÉNDICE E

Qué se comprobó para esta edición

Entorno	Evidencia
Unity 6000.6.3f1 en macOS	En una copia limpia del proyecto, la misma compilación pasó de 3 a 5 enemigos y volvió a 3. La misión que pedía 5 oleadas con solo 3 disponibles se sustituyó por la incluida.
Firebase Unity SDK 14.1.0	Authentication y Firestore cargaron la misión publicada en los emuladores. Storage entregó su escudo. El ensayo local utilizó opciones ficticias de Firebase y el laboratorio HTTP.
Firebase CLI 15.35.0, Node.js 22 y Emulator Suite	28 comprobaciones de lectura, permisos, publicación, concurrencia de premios, escritura en salas y archivos. El verificador del capítulo 20 repitió la reclamación con la misión del ensayo.
Firebase en la nube y emulador Android	Registro Android, Authentication anónimo, plantilla de Remote Config y reglas de Firestore. La compilación Android abrió una partida con revisión 2 y 5 enemigos al leer Remote Config en la nube.
Panel web	Publicación desde Hosting local con un rol de edición. Confirmación de la misión activa en el cliente.

Addressables, la interfaz de presencia, el botón de reclamación y los capítulos 13 a 19 son ejercicios de integración. Las capturas indican el entorno de cada prueba.