LUDIA

# Proyecto App Android ATELEM

29/06/2025

Samuel Díaz Nogales

Alexandre Ibáñez Sobrevela

Gema Fernández Valero

Alumnos de 3º grupo Telemática.

# Índice:

# 1. Introducción.

El presente proyecto consiste en el desarrollo de una aplicación móvil para dispositivos Android, implementada en Java utilizando Android Studio como entorno de desarrollo. La aplicación ofrece una pequeña colección de minijuegos clásicos con el objetivo de entretener al usuario y demostrar conocimientos en programación móvil. Los juegos incluidos son: **Tres en Raya**, **Ahorcado** y **Tirar un Dado de seis caras**. Cada uno de ellos presenta una interfaz simple e intuitiva, diseñada para facilitar la experiencia del usuario, al mismo tiempo que se aplican principios fundamentales de lógica de programación, manejo de interfaces gráficas, control de eventos y estructura modular en Java.

## 1.1. Aspectos a tener en cuenta.

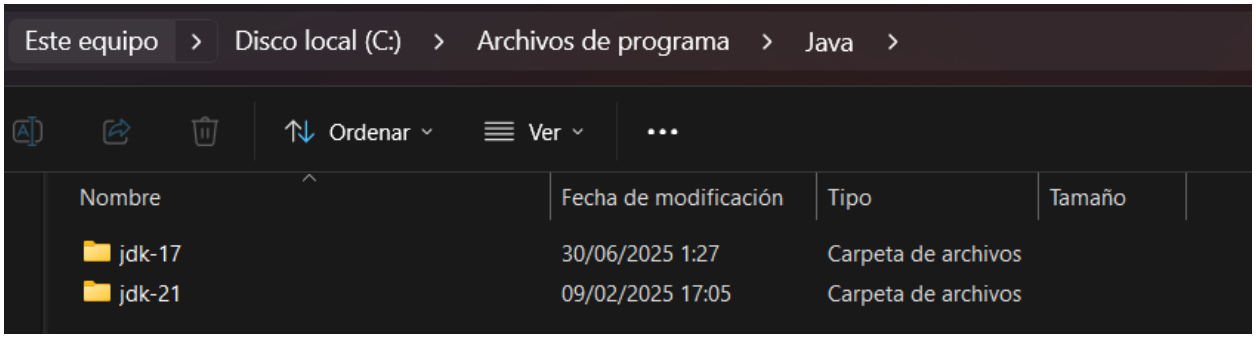
El proyecto se encuentra alojado en un repositorio público de GitHub, al cual se puede acceder desde el siguiente enlace:
https://github.com/gemafv/Trabajo-ATELEM-APP-Ludia.git

Seguidamente, si se quiere utilizar el repositorio se deberá utilizar el comando en línea de comandos: > git clone https://github.com/gemafv/Trabajo-ATELEM-APP-Ludia.git

Es fundamental para la correcta evaluación del proyecto que los profesores tengan instalada la versión **JDK 17**, ya que es la versión con la que fue desarrollado. La instalación puede realizarse desde el siguiente enlace oficial:

Descargar JDK 17.0.11 para Windows (x64)

Una vez instalada, la estructura de carpetas en el sistema debería quedar similar a la mostrada en la siguiente imagen.

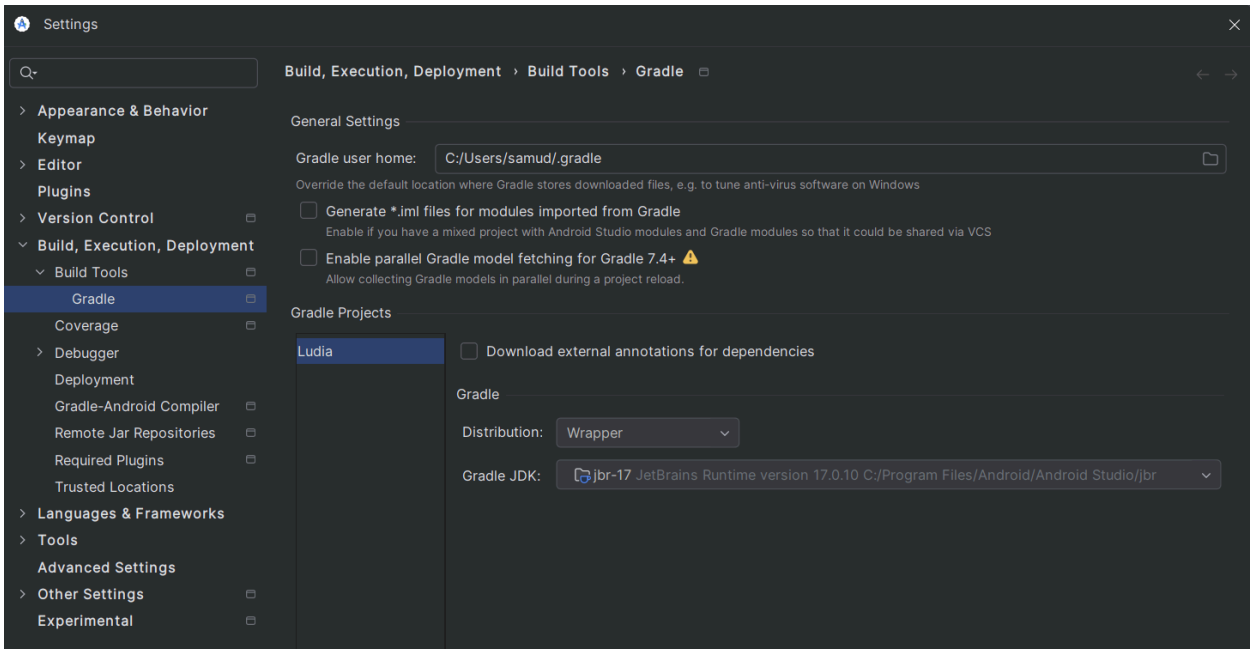


Para la correcta ejecución del proyecto, se ha configurado **Gradle** utilizando las siguientes opciones dentro de Android Studio:

- **Distribución**: Wrapper (se utilizará la versión indicada en el propio proyecto).
- **Gradle JDK**: Se ha configurado con la JDK instalada en la siguiente ruta:

```
C:\Program Files\Android\Android Studio\jbr
```

Esto se puede observar en la imagen adjunta, donde se muestra que Android Studio está utilizando **JetBrains Runtime (JDK 17.0.10)**, compatible con el entorno de desarrollo del proyecto.




## 1.2. Objetivos del proyecto.

El principal objetivo de este proyecto es diseñar y desarrollar una aplicación móvil funcional en el entorno Android, que sirva como una práctica integradora de los conocimientos adquiridos en programación con Java. A través de la creación de una aplicación compuesta por tres minijuegos, se busca:

- Aplicar conceptos de programación orientada a objetos en un contexto práctico.
- Familiarizarse con el entorno de desarrollo Android Studio y sus herramientas.

- Implementar interfaces de usuario intuitivas utilizando componentes nativos de Android.

- Desarrollar la lógica de funcionamiento de juegos clásicos, favoreciendo el razonamiento lógico y el diseño estructurado.

- Fomentar la reutilización de código y la organización modular del proyecto.

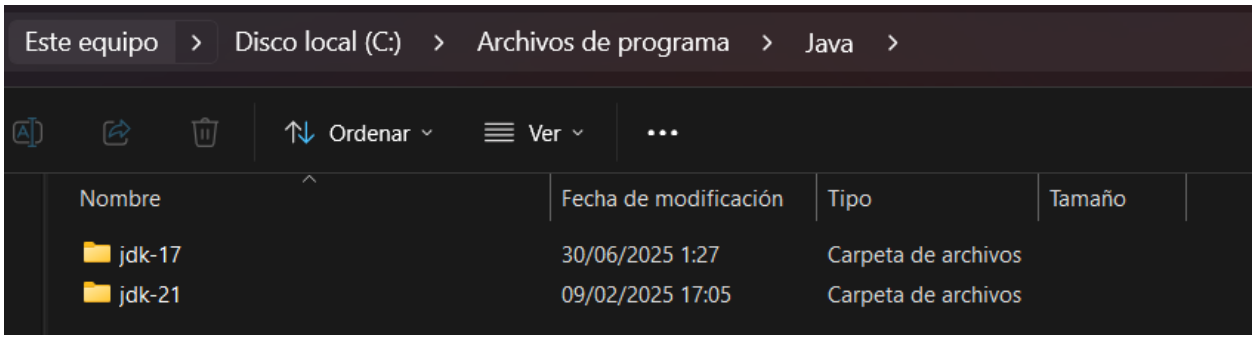
- Probar y depurar aplicaciones Android, haciendo uso de emuladores y dispositivos reales.

## 1.3. Herramientas y tecnologías utilizadas.

Para llevar a cabo el desarrollo de esta aplicación, se han utilizado las siguientes herramientas y tecnologías:

1. **Android Studio**: Entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Permite la gestión del diseño, el código y la compilación del proyecto.
2. **Java**: Lenguaje de programación principal utilizado para implementar la lógica del juego y manejar los eventos de la interfaz de usuario.
3. **SDK de Android**: Conjunto de herramientas que permite compilar, depurar y ejecutar aplicaciones para dispositivos Android.
4. **Emulador de Android**: Herramienta integrada en Android Studio que permite probar la aplicación en diferentes versiones de Android sin necesidad de un dispositivo físico.
5. **XML**: Utilizado para definir la estructura y diseño visual de las interfaces de usuario en cada actividad.
6. **Git**: Sistema de control de versiones para gestionar el desarrollo del proyecto y mantener un historial de cambios.

# 2. Análisis del proyecto.




## 2.1. Descripción general de la aplicación.

Es una aplicación pensada para ofrecer entretenimiento rápido y accesible a los usuarios a través de una variedad de minijuegos. La idea es que puedan divertirse y hacer que esos ratos libres o de espera se pasen volando. Además, la app busca ser visualmente atractiva, intuitiva y fácil de usar, para que tanto nuevos como antiguos usuarios disfruten la experiencia desde el primer momento.

## 2.2. Público al que va dirigido.

La aplicación está pensada principalmente para un público joven, entre los 12 y 35 años, ya que suelen tener ratos de ocio cortos y buscan formas rápidas de entretenerse. Aun así, esto no excluye a personas de mayor edad, ya que los minijuegos que incluye, como son el Tres en Raya, el Ahorcado o Tirar los Dados, son clásicos que casi todo el mundo conoce y puede disfrutar.

# 3. Diseño de la aplicación.

## 3.1. Arquitectura general del sistema.

La aplicación está desarrollada con una estructura modular basada en actividades (**Activities**), donde cada parte del proyecto está separada según su función. Se sigue una versión simple del patrón **Modelo–Vista–Controlador (MVC)**, donde las interfaces en XML se conectan con la lógica programada en Java. Cada juego tiene su propia actividad, y desde el menú principal se puede acceder a todos ellos. El proyecto está organizado siguiendo las convenciones de Android Studio, lo que facilita su comprensión y mantenimiento:

Dentro del directorio principal `app/src/main/` se encuentran las carpetas más importantes: `java/`, que contiene el código en Java organizado por actividades; `res/`, que incluye los recursos visuales como layouts, imágenes y estilos; y el archivo `AndroidManifest.xml`, que define las actividades y permisos. Esta organización ayuda a mantener separado lo visual, lo lógico y los datos del proyecto.

## 3.2. Pantalla de carga y logotipo.

La aplicación comienza con una pantalla de carga (Splash Screen) que muestra el logotipo de la app centrado sobre un fondo suave. El logotipo fue generado con inteligencia artificial, siguiendo un estilo simple, colorido y relacionado con el mundo de los videojuegos, buscando transmitir una sensación alegre y entretenida desde el primer momento.

El nombre "*Ludia*" hace referencia a la raíz latina **"lud-"**, que significa **jugar**, reforzando así la idea principal de la aplicación como plataforma simple y divertida. La pantalla de carga permanece unos segundos antes de acceder al menú principal, ofreciendo una transición fluida y profesional.

## 3.3. Menú principal.

El menú principal es el punto central de navegación de la aplicación. Se ha implementado mediante un diseño XML utilizando `RelativeLayout` como contenedor principal y `LinearLayout` para organizar los elementos de forma vertical y centrada.

Desde esta pantalla, el usuario puede acceder a los tres minijuegos disponibles: **Tres en Raya**, **Ahorcado** y **Tirar un Dado**, así como consultar sus estadísticas o cerrar sesión.

El **diseño XML del menú principal** presenta dos botones en la parte superior:

- Un `ImageButton` para acceder a las estadísticas, alineado en la esquina superior izquierda (`android:layout_alignParentStart="true"` y `android:layout_alignParentTop="true"`).

- Un `Button` para cerrar sesión, ubicado en la esquina superior derecha (`android:layout_alignParentEnd="true"` y `android:layout_alignParentTop="true"`).

Debajo de estos, **un `LinearLayout` vertical agrupa el logotipo, un título, los tres botones de juegos, y un texto de copyright**.

Este diseño modular permite que la parte superior (barra de acciones) y el contenido principal estén claramente diferenciados. Los botones de juego (`Button`) ocupan todo el

ancho disponible (`match_parent`) y comparten una estética coherente gracias al uso de `android:backgroundTint="#000000"`.

El fondo de pantalla se ha definido con `android:background="@drawable/fondomain"`, que corresponde con el fondo de pantalla usado para el menú principal y el menú de estadísticas.

El archivo `MainActivity.java` es el punto central de control de la navegación dentro de la aplicación. Aquí se gestiona la interacción con todos los botones que aparecen en el menú principal.

Cada botón está vinculado a su componente visual del XML mediante `findViewById`, lo que permite referenciarlo y asignarle una acción específica. A través del método `setOnClickListener`, se define el comportamiento de cada botón al ser pulsado. En este caso, se utilizan **Intents** para lanzar nuevas actividades correspondientes a cada uno de los juegos o funciones de la aplicación.

Por ejemplo:

- Al pulsar el botón **"Tres en Raya"**, se lanza `TresEnRayaActivity`.

Además, se ha añadido un botón de **Cerrar Sesión**, que aprovecha Firebase Authentication para cerrar la sesión actual del usuario. Una vez ejecutado `signOut()`, se redirige automáticamente al usuario a la pantalla de autenticación (`SignUpActivity`) y **se eliminan las actividades anteriores del historial para evitar volver atrás con el botón físico del dispositivo.** Esto se logra con el uso de las flags:

```
Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK
```

El bloque de código correspondiente es:

```
btnCerrarSesion.setOnClickListener(v -> {
    FirebaseAuth.getInstance().signOut(); // Cierra sesión en Firebase
    Intent intent = new Intent(MainActivity.this, SignUpActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK); // Limpia el backstack
    startActivity(intent);
    finish();
});
```

## 3.4. Juego 1. Tres en raya.

El juego de Tres en Raya está implementado en la clase `TresEnRayaActivity.java`, y su interfaz está definida en el archivo `activity_tres_en_raya.xml`. Este juego permite a dos jugadores alternar turnos en un tablero de 3x3, utilizando las letras X y O, con el objetivo de alinear tres símbolos iguales de forma horizontal, vertical o diagonal.

El archivo `TresEnRayaActivity.java` contiene la lógica principal del minijuego Tres en Raya, permitiendo la interacción entre dos jugadores, la detección de victorias o empates y el registro automático de resultados en Firebase. A continuación, se detalla cómo está estructurado y cómo funciona internamente el código.

**Inicialización y configuración:**

En el método `onCreate()`, se inicializan los botones de la interfaz mediante `findViewById()` y se enlazan con el XML usando `getIdentifier()` para construir dinámicamente los IDs de cada botón (`btn00`, `btn01`, ..., `btn22`). A cada botón se le asigna un `setOnClickListener()` que controla lo que sucede cuando un jugador pulsa sobre una casilla.

Además, se inicializan las variables necesarias para controlar el flujo del juego:

- `turnoJugador1`, que indica si es el turno del jugador X (true) o del jugador O (false).
- `ronda`, que lleva la cuenta del número de jugadas realizadas.
- `textTurno`, que muestra en pantalla de quién es el turno actual.

Se definen también dos botones adicionales:

- `btnReiniciar`, para reiniciar el tablero manualmente.
- `btnVolverMenu`, que utiliza un `Intent` para volver a la pantalla principal (`MainActivity`).

**Lógica del turno y detección de eventos:**

Cuando un jugador pulsa un botón, se comprueba si la casilla está vacía. Si lo está, se inserta una "X" o una "O" dependiendo del turno. A continuación, se incrementa la variable `ronda` y se verifica si hay un ganador mediante el método `comprobarGanador()`. Si alguien ha ganado (jugador X o Jugador O), se muestra un mensaje por `Toast`, se actualiza la base de datos en Firebase con `guardarResultadoEnFirestore()` (explicado más tarde) y se reinicia el tablero.

Si no hay ganador y las 9 casillas ya han sido ocupadas (`ronda == 9`), se considera empate y se actúa de forma similar. Si la partida sigue en curso, se cambia el turno y se actualiza el texto con `actualizarTextoTurno()`.

**Método `comprobarGanador()`:**

Este método verifica si hay una línea ganadora en el tablero. Primero copia los valores de los botones a una matriz `tablero`, y luego comprueba si alguna fila, columna o diagonal contiene tres símbolos iguales (X u O) y no vacíos. Si encuentra una coincidencia, devuelve `true` indicando que hay un ganador; si no, devuelve `false` y la partida continúa.

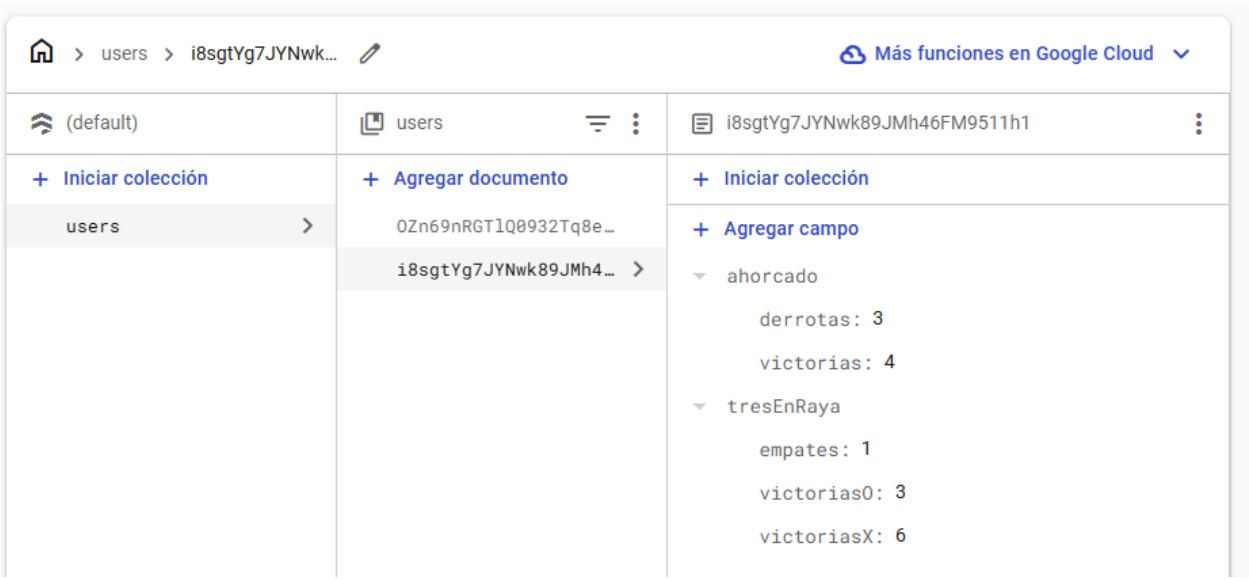
**Reinicio y salida:**

- `resetearTablero()` limpia el contenido de todos los botones, reinicia el número de rondas y restablece el turno al jugador 1.

- `volverAlMenu()` genera un `Intent` que lanza `MainActivity`, y con `finish()` se finaliza la actividad actual para que no quede en segundo plano.

**Firebase: Guardado de estadísticas**

El método `guardarResultadoEnFirestore(String resultado)` actualiza en Firebase las estadísticas del usuario autenticado.
Dependiendo del resultado (`victoriaX`, `victoriaO` o `empate`), se determina qué campo actualizar (`victoriasX`, `victoriasO`, `empates`). Se utiliza `FieldValue.increment(1)` para sumar al contador correspondiente dentro del documento del usuario.




En caso de que el documento no exista, se crea uno nuevo con los valores iniciales correctos mediante un `Map<String, Object>`.

**Interfaz gráfica:**

La interfaz utiliza un `RelativeLayout` como contenedor principal, lo que permite posicionar elementos de forma relativa entre sí. De este modo, el botón de **"Volver"** se mantiene anclado en la esquina superior izquierda de la pantalla de forma fija.

Bajo este botón, se ubica un `LinearLayout` vertical centrado en pantalla, que agrupa el contenido principal del juego: el texto que indica el turno actual, el tablero 3x3 representado mediante un `GridLayout`, y el botón de **"Reiniciar partida"**. Esta estructura jerárquica mejora la comprensión visual y facilita la interacción por parte del usuario.

El `GridLayout` cuenta con nueve botones personalizados (`@style/TicTacToeButton`), que funcionan como casillas del tablero. Este enfoque modular permite mantener el código limpio y reutilizable. Cada botón representa una posición única del tablero (identificados como `btnXY`) y está conectado al código Java mediante `findViewById` y `setOnClickListener`, asegurando una respuesta inmediata al pulsarlo.

A nivel estético, todos los elementos están dispuestos sobre un fondo gráfico (`@drawable/fondotresenraya`) utilizado solo en esta actividad. Los textos y botones utilizan colores oscuros para garantizar un buen contraste y legibilidad.

## 3.5. Juego 2. Ahorcado.

El segundo minijuego implementado en la aplicación es el clásico juego del Ahorcado. Su funcionamiento se estructura en varios bloques clave que permiten gestionar el progreso del jugador, comprobar letras, mostrar la interfaz gráfica y guardar estadísticas.

**Declaración de variables principales:**

Para comenzar, se define un array de palabras llamado `bancoPalabras` con más de 100 palabras diferentes. A partir de ese listado, se seleccionan aleatoriamente 5 palabras que se utilizarán como retos para cada nivel. Estas cinco palabras se guardan en el array `palabras`, que es el que se usará realmente en el juego.

El juego lleva el control del nivel actual con la variable `nivel`, así como de los fallos cometidos mediante la variable `errores`. También se utilizan las variables `palabraActual`, que guarda la palabra a adivinar en ese momento, y `palabraOculta`, que representa esa misma palabra pero con guiones bajos en lugar de letras aún no descubiertas.

**Inicialización del juego (onCreate):**

Al iniciar la actividad, se conectan todos los elementos de la interfaz (TextViews, EditText, ImageView, botones, etc.) con sus respectivas variables en Java. Además, se realiza la conexión con Firebase para poder guardar estadísticas más adelante. Después de inicializar todo, se seleccionan 5 palabras aleatorias a partir del array principal bancoPalabras, que contiene más de 100 palabras diferentes.

Para garantizar que las palabras cambian en cada partida, se utiliza el método `Collections.shuffle`, que mezcla aleatoriamente el orden de la lista antes de seleccionar las 5 definitivas. A continuación, se arranca el primer nivel del juego.

**Inicio de cada nivel:**

Cuando el jugador supera una palabra o comienza una nueva partida, el juego inicia un nuevo nivel. En este punto, se accede al array `palabras` y se selecciona la palabra correspondiente al índice actual almacenado en la variable `nivel`. Esa palabra se convierte automáticamente en una versión oculta, construyendo una cadena del mismo tamaño pero formada por guiones bajos (`_`), uno por cada letra. Esta transformación se realiza

mediante un `StringBuilder`, que facilita la actualización de letras descubiertas durante el juego.

Antes de empezar el reto, también se reinician ciertas variables clave: el contador de errores (`errores`) vuelve a cero y se limpia el conjunto que guarda las letras incorrectas introducidas en ese nivel (`letrasUsadasIncorrectas`).

Por último, se llama al método `actualizarUI()`, encargado de mostrar en la pantalla el estado inicial del nivel: la palabra con sus guiones bajos, el nivel actual, la imagen del muñeco del ahorcado correspondiente al número de errores (que inicialmente es cero) y un texto vacío para las letras incorrectas, que se irá rellenando conforme el jugador falle. Esto asegura que cada nivel comience limpio y de forma coherente.

**Comprobación de la letra introducida:**

Cada vez que el jugador introduce una letra en el campo de texto y pulsa el botón de "Probar", el juego recoge esa letra y la convierte a mayúsculas para compararla con la palabra actual. **Lo primero que hace el programa es asegurarse de que solo se ha introducido una única letra**. Si el jugador escribe más de una, se muestra un mensaje de advertencia y no se continúa con la comprobación.

Una vez validada la entrada, el juego recorre la palabra actual carácter por carácter para ver si la letra introducida está presente.

- ➔ **Si se encuentra la letra**, esta será colocada en su(s) posición(es) correspondiente(s) dentro de la `palabraOculta`, sustituyendo los guiones bajos. Esto permite que el jugador vea su progreso en la pantalla al descubrir nuevas letras.
- ➔ **Si la letra no se encuentra** en la palabra, se considera un fallo. En ese caso, se añade la letra al conjunto de letras incorrectas (para que no se repitan) y se incrementa el contador de errores. Además, la imagen del muñeco del ahorcado se actualiza para reflejar ese fallo, haciendo el juego visualmente más claro.

Cuando el número de errores alcanza el límite (6 intentos fallidos), el jugador pierde automáticamente. En ese momento se muestra un mensaje con la palabra correcta, se cambia la imagen del ahorcado al estado final y se pasa automáticamente al siguiente nivel.

Por otro lado, si el jugador consigue adivinar todas las letras antes de alcanzar el límite de errores, el juego detecta que la palabra ha sido completada correctamente. En ese caso se muestra un mensaje de felicitación y también se pasa al siguiente nivel.

En ambos resultados —ya sea victoria o derrota— se actualizan las estadísticas del jugador en Firebase y se reinician las variables internas para preparar el próximo reto. Además, se

vuelve a llamar al método de actualización de interfaz para reflejar todos los cambios en pantalla.

**Actualización de la interfaz gráfica:**

La interfaz gráfica se actualiza constantemente mediante un método específico. Este método muestra la palabra oculta con sus letras adivinadas y guiones bajos, indica el nivel actual, actualiza la imagen del muñeco del ahorcado según el número de errores y muestra la lista de letras incorrectas utilizadas.

**Registro de estadísticas en Firebase:**

Cada vez que el jugador completa un nivel (ya sea ganando o perdiendo), se actualizan sus estadísticas en Firebase. Dependiendo del resultado, se incrementa el contador de victorias, derrotas o “victorias magistrales” (cuando se superan todos los niveles sin fallos). Si es la primera vez que el jugador juega, se crea automáticamente su documento en la base de datos con las estadísticas iniciales.

**Interfaz gráfica:**

La interfaz gráfica del juego del Ahorcado está definida mediante un archivo XML que estructura y da estilo a todos los elementos visuales de la actividad. El diseño se basa en un `RelativeLayout` como contenedor principal, lo que permite posicionar los elementos de forma relativa entre ellos y con respecto al contenedor.

En la esquina superior izquierda se encuentra un botón con el texto "Volver", que permite regresar al menú principal. Este botón está anclado a la parte superior e izquierda del contenedor, asegurando su posición fija.

El contenido principal del juego está agrupado dentro de un `LinearLayout` vertical, centrado tanto horizontal como verticalmente. Dentro de este bloque se encuentran todos los elementos que forman parte de la experiencia de juego:

- Un `TextView` muestra el número de nivel actual en el que se encuentra el jugador.
- Una `ImageView` contiene la imagen del muñeco del ahorcado, que va cambiando según los errores cometidos.

- Otro `TextView` representa la palabra oculta, donde las letras adivinadas se revelan y las no descubiertas se muestran como guiones bajos.
- Un `EditText` permite al jugador escribir una letra para intentar adivinar la palabra.
- Un botón llamado "Probar" que el jugador pulsa para confirmar su intento.
- Finalmente, un `TextView` muestra las letras incorrectas ya utilizadas, con un estilo en cursiva y color rojo para mayor claridad.

## 3.6. Juego 3. Tirar dados.

El tercer minijuego incluido en la aplicación simula la tirada de un dado clásico de seis caras. Su funcionamiento es simple y directo, pensado para ofrecer una experiencia breve y útil.

**Inicialización y configuración:**

La actividad encargada del juego de Tirar Dados está definida en la clase `TirarDadoActivity.java`, con su interfaz gráfica en el archivo `activity_tirar_dado.xml`. Durante la inicialización (en el método `onCreate()`), se vinculan los elementos visuales del layout con sus referencias en Java mediante `findViewById()`. Estos elementos son:

- `imageDado`: `ImageView` que muestra la cara actual del dado.
- `btnTirarDado`: botón que ejecuta la animación y el resultado de la tirada.
- `btnVolverMenu`: botón para regresar al menú principal.

Además, se instancia un objeto `Random`, que se utilizará para generar los resultados aleatorios de cada tirada.

**Lógica y `método tirarDadoConAnimacion()`:**

Cuando el jugador pulsa el botón **"Tirar dado"**, se ejecuta el método `tirarDadoConAnimacion()`. Este método crea una animación de rotación (`RotateAnimation`) que gira el dado dos vueltas completas sobre su centro (720°) en 500

milisegundos. Durante la animación, el botón queda desactivado mediante `setEnabled(false)` para evitar múltiples clics simultáneos.

La animación utiliza un `AnimationListener`, que detecta cuándo ha terminado la rotación. En ese momento, se genera un número aleatorio entre 1 y 6 con `random.nextInt(6) + 1`. Este número representa la cara del dado obtenida, y se utiliza como argumento para llamar al método `mostrarImagenDado()`.

**Método `mostrarImagenDado()`:**

Este método recibe un número entero del 1 al 6 y, mediante una estructura `switch`, selecciona y muestra la imagen correspondiente usando `imageDado.setImageResource(...)`.

Las imágenes (`dado_1`, `dado_2`, ..., `dado_6`) están almacenadas en la carpeta `res/drawable/` y simulan las diferentes caras del dado. El cambio visual se realiza con `imageDado.setImageResource(...)`.

**Botón "Volver":**

La actividad incluye un botón adicional llamado **"Volver"**, situado en la esquina superior izquierda, que permite regresar al menú principal (`MainActivity`). Este botón lanza un `Intent` y termina la actividad actual con `finish()`, asegurando que no permanezca en la pila de navegación.

**Interfaz gráfica:**

El archivo XML `activity_tirar_dado.xml` define la interfaz de usuario de la actividad "Tirar Dado". Utiliza un `RelativeLayout` como contenedor principal, con un fondo personalizado (`@drawable/fondodado`) y márgenes interiores. En la esquina superior izquierda se coloca un botón "**Volver**" que permite regresar al menú principal.

En el centro de la pantalla se muestra una imagen (`ImageView`) que representa el dado, inicialmente con la cara 1 visible. Justo debajo se encuentra el botón "Tirar dado", centrado horizontalmente, que al pulsarse ejecuta una animación y actualiza la imagen del dado según el resultado.

La distribución es sencilla y clara, no muy sobrecargada pues es la actividad más sencilla de Ludia y su objetivo es una experiencia rápida y simple.

## 3.7. Inicio de sesión y base de datos.

Hemos implementado un sistema de inicio de sesión para permitir que **cada usuario tenga una cuenta individual dentro de la aplicación**. Esto nos permite almacenar y gestionar de forma personalizada los datos de cada jugador, como sus estadísticas y progreso, además de mantener un control claro sobre quién utiliza la app en cada momento.

Antes de implementar el sistema de autenticación y base de datos con Firebase en Android, es necesario realizar una serie de configuraciones previas en el proyecto:

1. **Integración de Firebase en el proyecto Android**:

   Primero se debe registrar la aplicación en Firebase Console e incluir el archivo `google-services.json` descargado en la carpeta `app/` del proyecto.

2. **Configuración del archivo `build.gradle`**:

   En el `build.gradle` (nivel proyecto) se debe incluir la clase de dependencia de Firebase.

```
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.google.gms.google.services) apply false
}
```

   En el `build.gradle` del módulo `app`, también se añaden las dependencias necesarias para Firebase Authentication y Firestore:

```
// Importa la plataforma Firebase BOM
implementation(platform("com.google.firebase:firebase-bom:32.2.2"))

// Declara las dependencias sin versiones específicas
implementation("com.google.firebase:firebase-auth-ktx")
implementation("com.google.firebase:firebase-firestore-ktx")
```

3. **Modificación del `AndroidManifest.xml`**:

   Declaramos la actividad `SignUpActivity` como principal al iniciar la aplicación:

```
<activity
    android:name=".SignUpActivity"
    android:exported="true">
```

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Una vez finalizada la configuración, se implementa el sistema de registro e inicio de sesión con autenticación por correo electrónico y contraseña, y el almacenamiento de estadísticas en la base de datos `Firestore`.

**Registro de usuario: SignUpActivity**

La clase `SignUpActivity` gestiona el proceso de creación de una cuenta nueva mediante correo electrónico y contraseña utilizando Firebase Authentication. Al iniciarse la actividad, se inicializan las referencias a los elementos de la interfaz, como los campos `EditText` para el email y la contraseña, el botón de registro (`signupButton`) y el enlace textual para cambiar a la pantalla de inicio de sesión (`loginRedirigirText`).

Cuando el usuario pulsa el botón "Registrarse", se validan los datos introducidos: se comprueba que el campo del email no esté vacío y, aunque no se verifica explícitamente en el código, también se espera que la contraseña tenga una longitud mínima aceptable según las políticas de Firebase (por defecto, 6 caracteres). Si el correo es válido, se ejecuta el método `createUserWithEmailAndPassword()` de `FirebaseAuth`, que lanza una tarea asíncrona.

Si la tarea se completa con éxito, se muestra un mensaje `Toast` de confirmación y se redirige automáticamente al usuario a `LoginActivity`, la pantalla de inicio de sesión. Si el registro falla (por ejemplo, porque el correo ya está en uso o la contraseña es débil), se muestra un mensaje de error con el motivo exacto, obtenido mediante `task.getException().getMessage()`.

Desde el punto de vista visual, el diseño se define en `activity_sign_up.xml`. Esta interfaz incluye una `CardView` centrada verticalmente sobre un fondo decorativo (`fondologin`), que contiene un formulario estilizado con campos de entrada personalizados (usando `custom_edittext`), iconos decorativos y botones con esquinas redondeadas (`cornerRadius`). Esto proporciona una apariencia moderna y coherente con el diseño general de la app.

Además, se proporciona una navegación clara: si el usuario ya tiene cuenta, puede pulsar el enlace “¿Ya tienes una cuenta? Inicia sesión”, que lo redirige directamente a `LoginActivity` sin necesidad de repetir pasos.

**Inicio de sesión: LoginActivity**

La clase `LoginActivity` es la encargada de gestionar el acceso de usuarios ya registrados a la aplicación. Su función principal es verificar las credenciales introducidas por el usuario (correo y contraseña) y permitir el acceso al contenido de la app si estos datos son válidos.

Al iniciar la actividad, se inicializan los elementos de la interfaz gráfica, como los `EditText` para introducir el correo (`loginEmail`) y la contraseña (`loginPassword`), el botón de inicio de sesión (`loginButton`) y el `TextView` que redirige al formulario de registro si el usuario aún no tiene cuenta (`signupRedirigirText`).

Cuando el usuario pulsa el botón "Acceder", se realiza una validación previa del campo de email utilizando `Patterns.EMAIL_ADDRESS.matcher(email).matches()` para comprobar que el formato del correo electrónico es válido. Si el email no está bien formado o si alguno de los campos está vacío, se notificará al usuario con errores en pantalla.

Si los campos son correctos, se ejecuta el método `signInWithEmailAndPassword()` proporcionado por FirebaseAuth. Esta operación se realiza de manera asíncrona, y si la autenticación es exitosa, se muestra un mensaje de éxito mediante `Toast` y se redirige automáticamente al usuario a `MainActivity`, donde puede comenzar a usar la aplicación. Si la autenticación falla (por ejemplo, si la contraseña es incorrecta o el usuario no existe), se muestra un mensaje de error genérico.

La interfaz gráfica se encuentra definida en el archivo `activity_login.xml`, y sigue el mismo patrón estético que la pantalla de registro (`SignUpActivity`). Toda la vista está organizada dentro de una `CardView` con esquinas redondeadas y sombra (`cardElevation`), centrada verticalmente sobre un fondo decorativo pastel. Los campos de entrada cuentan con iconos personalizados y estilos coherentes (`custom_edittext`), y los botones mantienen una paleta de colores oscuros sobre el fondo claro.

**Gestión de usuarios y estadísticas:**

Una vez que un usuario inicia sesión, su UID queda disponible para ser utilizado como clave en la base de datos `Firestore`.
Esto permite guardar estadísticas individuales de cada jugador (por ejemplo, victorias, empates, partidas jugadas, etc.) desde cualquier parte de la app. Este enfoque

estructurado permite registrar y consultar información de forma eficiente y segura, ya que solo los usuarios autenticados pueden acceder a sus datos.

Este módulo de autenticación y conexión con base de datos representa la base para personalizar la experiencia del usuario en la app y guardar el progreso, además de sentar las bases para futuras funcionalidades como rankings globales, historial de partidas, o perfiles.

## 3.8. Actividad Estadísticas.

La **actividad `EstadisticasActivity`** permite al usuario visualizar su progreso individual en los dos juegos principales de la aplicación: **Tres en Raya** y **Ahorcado**. Se trata de una pantalla informativa que obtiene los datos del usuario autenticado desde la base de datos de Firebase Firestore y los muestra en una interfaz organizada y accesible.

**Funcionamiento general:**

Cuando se inicia la actividad, se establece una conexión con Firebase (`FirebaseFirestore`) y se obtiene el usuario autenticado a través de `FirebaseAuth`. A continuación, se ejecutan dos métodos principales:

- `cargarEstadisticasTresEnRaya()`
- `cargarEstadisticasAhorcado()`

Cada uno accede a una sección distinta del documento del usuario en Firestore y actualiza los campos de texto (`TextView`) en pantalla con los datos recuperados.

**Estructura de los datos en Firebase:**

Cada usuario tiene un documento único dentro de la colección `"users"`, identificado por su **UID.** Dentro de ese documento, las estadísticas se organizan como objetos anidados bajo claves específicas para cada juego:

**Tres en Raya (`tresEnRaya`)**:

```
"tresEnRaya": {
    "victoriasX": 3,
```

```
    "victoriasO": 2,
    "empates": 5
}
```

Estos valores se actualizan automáticamente al finalizar cada partida en `TresEnRayaActivity` mediante el método `guardarResultadoEnFirestore()`. Este incrementa el valor correspondiente (`victoriasX`, `victoriasO` o `empates`) usando `FieldValue.increment(1)`. Si el documento del usuario aún no existe, se crea con valores iniciales.

**Ahorcado (`ahorcado`)**:

```
"ahorcado": {
    "victorias": 4,
    "derrotas": 1,
    "victoriasMagistrales": 1
}
```

En `AhorcadoActivity`, el resultado de cada nivel se guarda con el método `guardarResultadoEnFirestore()`, que también usa `FieldValue.increment(1)` para actualizar el campo correspondiente (`victorias`, `derrotas` o `victoriasMagistrales`). Si es la primera vez que se juega, se inicializa el objeto `ahorcado` en el documento del usuario.

**Tratamiento de datos en la app:**

Al recuperar los datos en `EstadisticasActivity`, se utilizan los métodos `cargarEstadisticasTresEnRaya()` y `cargarEstadisticasAhorcado()` para obtener el contenido de los objetos anidados desde Firebase. Para prevenir errores por datos ausentes o malformados, se hace una conversión segura con `getLongSafe()`, que comprueba el tipo del dato y devuelve un valor válido.

*En caso de que los datos no existan (por ejemplo, si el usuario aún no ha jugado), se muestran ceros por defecto en todos los campos mediante los métodos `setTresEnRayaStatsToZero()` y `setAhorcadoStatsToZero()`.*

## 4. Conclusiones.

El desarrollo de esta aplicación nos ha permitido experimentar de forma práctica el proceso completo de creación de una app móvil, comprendiendo tanto los desafíos técnicos como organizativos que conlleva. Ha sido una excelente oportunidad para mejorar el trabajo en equipo, repartiendo tareas de forma eficiente y colaborando activamente en las distintas fases del proyecto.

Además, hemos consolidado conocimientos teóricos adquiridos en clase, aplicándolos en un entorno real y funcional, lo que nos ha ayudado a comprender su utilidad práctica. Un aspecto especialmente enriquecedor ha sido el trabajo con XML para el diseño de la interfaz gráfica, área que no habíamos tratado en profundidad en la teoría. Esto nos ha llevado a investigar de forma autónoma, aprendiendo a buscar y contrastar información en documentación oficial, tutoriales y foros especializados.

En definitiva, este tipo de proyectos resulta muy valioso para nuestra formación, ya que nos aproxima a dinámicas reales del entorno profesional, permitiéndonos adquirir competencias técnicas, organizativas y de autoaprendizaje que serán clave en nuestro futuro laboral.

## 5. Bibliografía y recursos.

Android Developers. (s. f.). *Develop with Android Studio*, de https://developer.android.com/studio

Wikipedia. (2025). *Android Studio*, de https://es.wikipedia.org/wiki/Android_Studio

GeeksforGeeks. (2024). *Android Studio Tutorial*, de https://www.geeksforgeeks.org/android-studio-tutorial/

StackOverflow. (2015). *How to Debug in Android Studio?*, de https://stackoverflow.com/questions/22946386/how-to-debug-in-android-studio

Android Developers. (s. f.). *Layouts*, de https://developer.android.com/develop/ui/views/layout/declaring-layout

GeeksforGeeks. (2022). *A Complete Guide to Learn XML For Android App Development*, de https://www.geeksforgeeks.org/a-complete-guide-to-learn-xml-for-android-app-development/

Kella, S. (2023). *How Chat GPT is useful for android developers*. Medium, de https://medium.com/@sujithkella/how-chat-gpt-is-useful-for-android-developers-1a72695e2edb

StackOverflow. (2013). *Android Developer Documentation: download?*, de https://stackoverflow.com/questions/14558593/android-developer-documentation-download

OpenAI. (2025). *Asistencia de ChatGPT para resolución de errores y desarrollo en Android Studio*, de https://chat.openai.com/