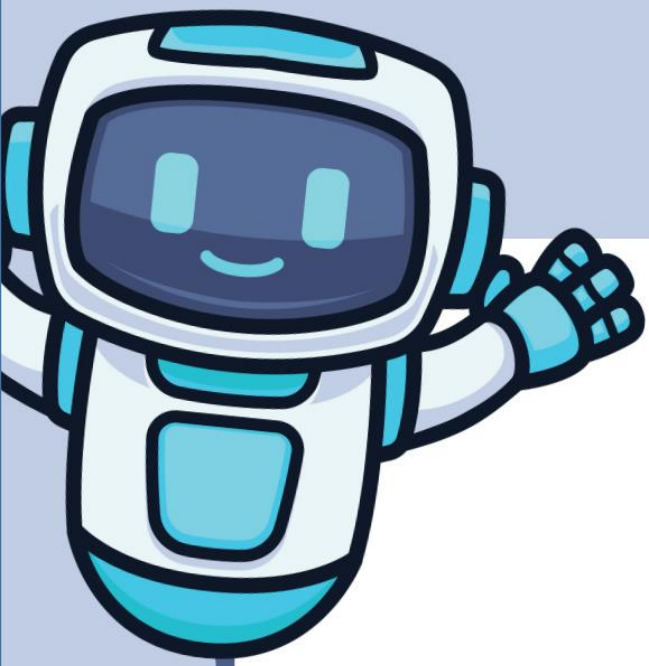
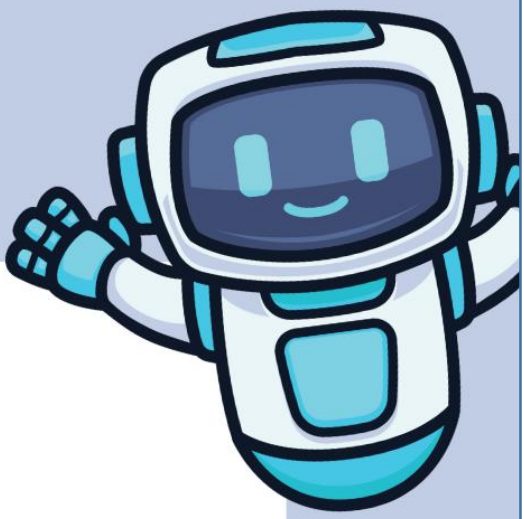




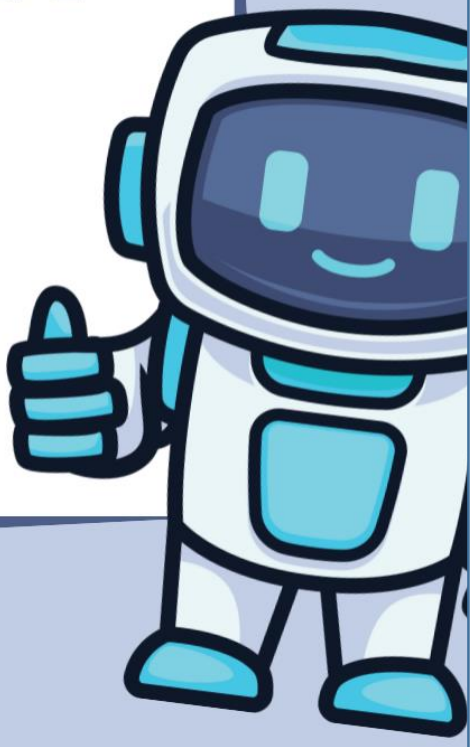
PROYECTO

# PROGRAMACION MULTIMEDIA


## POR: ANGEL GALLGO FELIPE

Campus FP Getafe
1º Desarrollo aplicaciones multiplataforma

29/11/2024



# Contenido

# ESPECIFICACIONES

## Propósito de la práctica:

El objetivo de esta práctica es desarrollar un tap game ejecutable en dispositivos JAVA/Android que permita a un jugador añadir su nombre de usuario, consultar sus puntuaciones, contactar con la empresa desarrolladora y jugar capturando los taps del usuario para superar pantallas y mejorar a nuestro "personaje". Libre temática.

Entre algunos de los juegos que actualmente siguen esta mecánica podríamos destacar

TapTitans (http://www.gamehive.com/games/tap-titans) y CookieClicker (http://orteil.dashnet.org/cookieclicker/).

## Escenario de la práctica:

Nuestra empresa PrayTheSun quiere desarrollar un tap game para atraer a cientos de jugadores. "TapSouls" sera un juego donde los usuarios podrán mejorar a su personaje luchando en diferentes niveles contra diferentes bosses. Debes desarrollar un MVP (mínimo producto viable) para que el equipo de QA pueda testear y la empresa publicar la app antes del lanzamiento en verano del juego.

Requisitos generales de la práctica:

El CTO junto al jefe de producto de nuestra empresa ha conseguido crear un gran equipo técnico donde nuestro Lead nos exige dar lo mejor de nosotros mismos.

Entre las exigencias que nos marcamos en nuestro equipo en cada sprint siempre destacamos la calidad de nuestro código. Es por eso que debes de entregar todo el código fuente del proyecto limpio, siguiendo criterios de responsabilidad única/modulado, entregas sin bugs, sin leaks de memoria, respetando estructuras de desarrollo Android nativas, test automatizados y con el código JAVA comentado en inglés.

La aplicación debe desarrollarse para que cualquier otro compañero pueda seguir aumentando sus funcionalidades en el futuro y nosotros podamos irnos de vacaciones sin recibir emails.

La práctica deberá ser entregada progresivamente, respetando ramas/features con commits/push de poco código y cada push comentado en un repositorio oficial de la Escuela (o equivalente). La práctica contendrá el código fuente del programa junto a la memoria y toda la información que se considere necesaria. NO SE DEBE entregar un fichero o librería ya compilados.

La práctica será realizada de forma individual. El código debe incluir en las cabeceras de cada clase su documentación pertinente, respetando la estructura de documentación de clases de JAVADoc.

## Tareas de la práctica:

Tarea 1

Tarea 1 .a Creación App y Navegación

Se creará una proyecto estructurado con la navegación básica entre las actividades principales.

• La aplicación debe ser desarrollada haciendo uso de la IDE de Android Studio y en código

JAVA/Kotlin.

• La navegación y las vistas deberá desarrollarse mediante Layouts XML y código

JAVA/Kotlin.

• Las actividades principales deberán incluir únicamente código relacionado con la vista.

Tarea 1 .b Población de Vistas y contenidos

• Se creará una clase independiente cuyo único objetivo será el de conectar con la API y devolver la información de la misma de forma asíncrona, inicialmente con datos locales de un fichero JSON u otro formato de texto.

• Se deberán de validar todos los campos en los que interactúe el usuario.

• Se deberá de facilitar feedback al realizar acciones incorrectas como la de añadir nombre de usuario.

• Se deberán parsear los datos del fichero JSON/otros desde una clase independiente y devolviendo un objeto con el contenido del mismo.

Tarea 2

• Se conectará una base de datos local o remota para recuperar los datos de “ranking” de usuarios.

• Se trabajará en varios hilos para evitar el bloqueo del hilo principal en la obtención de datos.

• Se verificarán y evitarán potenciales leaks de memoria.

Tarea 3

• Se persistirán los datos del usuario en el dispositivo y su evolución de personaje.

Tarea 4

Acompañando al código, entregará una memoria monográfica del proyecto con las siguientes secciones:

• App Project:

o Introducción de la app y demo técnica. Explicación de qué características se han desarrollado, patrones de desarrollo, estructura, tests. Un sencillo y orientativo diagrama de clases.

• UI/UX Flow Chart:

o Flujo de vistas y comportamiento aplicación-usuario.

• Short User Manual and Demo Example:

o Breve manual de compilación y uso de la demo técnica.

# Introducción

El proyecto **TapSouls** es un juego móvil desarrollado como práctica académica, cuyo propósito principal es implementar las mecánicas de un *tap game* clásico, donde los jugadores pueden mejorar a su personaje enfrentando diferentes niveles y derrotando jefes mediante interacciones rápidas en la pantalla. Este proyecto no solo busca ofrecer una experiencia divertida al usuario, sino también demostrar la aplicación de buenas prácticas de desarrollo, diseño modular y uso eficiente de tecnologías móviles.

## *Tecnologías utilizadas*

- **Android Studio**: Plataforma principal de desarrollo, que permite crear aplicaciones Android nativas de forma eficiente.
- **Java**: Lenguaje principal utilizado para la lógica del juego, procesamiento de datos y diseño de actividades.
- **SQLite**: Base de datos ligera y embebida que gestiona la persistencia de datos localmente. Se utiliza para almacenar información clave, como nombres de usuario, puntuaciones y progresos del juego.
- **JSON**: Formato para datos iniciales locales, utilizado durante la fase de prototipado para poblar vistas y simular una API.
- **XML**: Lenguaje utilizado para definir los layouts de las interfaces gráficas, asegurando una estructura visual clara y adaptada a dispositivos móviles.

## *Estructura del código*

El código del proyecto está diseñado para ser modular y fácil de mantener, siguiendo los principios de responsabilidad única y separación de capas. Se organiza de la siguiente manera:

1. **Capa de Presentación (UI)**:
   - Las actividades principales (`MainActivity`, `GameActivity`, `ScoreActivity`) contienen exclusivamente lógica relacionada con la interacción del usuario y navegación entre pantallas.
   - Las vistas están definidas en archivos XML, respetando la arquitectura nativa de Android.
2. **Capa de Lógica y Servicios**:
   - La clase `ApiConnector` gestiona la obtención y el procesamiento de datos locales en formato JSON.
   - Se validan todas las entradas del usuario mediante mensajes de error claros y retroalimentación visual.
3. **Capa de Datos**:
   - La clase `DatabaseHelper` actúa como puente entre la aplicación y la base de datos SQLite. Permite realizar operaciones CRUD (crear, leer, actualizar, eliminar) para almacenar y recuperar datos relacionados con usuarios y rankings.
   - Los datos persistentes aseguran que el progreso del jugador, como puntuaciones y mejoras, se mantenga entre sesiones de juego.

4. **Hilos y Eficiencia**:
    - El uso de hilos en operaciones intensivas, como la carga de datos o el acceso a la base de datos, garantiza que la interfaz de usuario no se bloquee, ofreciendo una experiencia fluida.

### *Objetivo del diseño*

El proyecto se desarrolla siguiendo buenas prácticas de documentación y pruebas automatizadas, permitiendo que otros desarrolladores puedan ampliar sus funcionalidades fácilmente en el futuro. Cada clase está comentada con Javadoc, y el control de versiones mediante *commits* progresivos asegura trazabilidad y claridad en el desarrollo.

## Código más relevante

### DatabaseHelper

DatabaseHelper es una clase que gestiona la interacción con la base de datos SQLite. Define las operaciones CRUD (Create, Read, Update, Delete) necesarias para almacenar y recuperar datos esenciales de la aplicación, como las puntuaciones de los usuarios.onCreate: Define la estructura de la tabla scores. A continuación el código mas relevante.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_TABLE = "CREATE TABLE " + TABLE_NAME + " (" +
            COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_USERNAME + " TEXT, " +
            COLUMN_SCORE + " INTEGER)";
    db.execSQL(CREATE_TABLE);
}
```

Método onCreate(SQLiteDatabase db)

Qué hace: Este método se invoca automáticamente la primera vez que se crea la base de datos. Su propósito es definir la estructura inicial de la base de datos, en este caso, la tabla scores.

Cómo lo hace: Crea una tabla llamada scores

Define tres columnas:

id: Identificador único (clave primaria), que se autoincrementa con cada nuevo registro.

username: Nombre del usuario, almacenado como texto.

score: Puntuación del usuario, almacenada como un número entero.

Ejecución de la instrucción SQL: Usa db.execSQL(CREATE_TABLE) para ejecutar el comando SQL que crea la tabla.

```
1 usage
public void addScore(String username, int score) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_USERNAME, username);
    values.put(COLUMN_SCORE, score);
    db.insert(TABLE_NAME,  nullColumnHack: null, values);
    db.close();
}
```

Qué hace: Inserta un nuevo registro en la tabla scores. Este registro incluye un nombre de usuario y una puntuación.

Cómo lo hace: Abre la base de datos en modo escritura

Usa this.getWritableDatabase() para obtener acceso a la base de datos y permitir la escritura.

Prepara los valores a insertar: Usa un objeto ContentValues para almacenar pares clave-valor, donde:

La clave (username, score) corresponde al nombre de las columnas en la tabla.

El valor corresponde a los datos que se quieren insertar.

Inserta los datos: Usa db.insert("scores", null, values) para insertar los valores en la tabla scores.

Devuelve el resultado: Retorna el ID del registro recién insertado si la operación fue exitosa, o -1 si falló.

```
usage
public ArrayList<String> getAllScores() {
    ArrayList<String> scores = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "SELECT * FROM " + TABLE_NAME + " ORDER BY " + COLUMN_SCORE + " DESC",  selectionArgs: null);
    if (cursor.moveToFirst()) {
        do {
            String score = cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_USERNAME)) + ": " +
                    cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_SCORE));
            scores.add(score);
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return scores;
```

Qué hace: Recupera las 10 puntuaciones más altas almacenadas en la tabla scores.

Cómo lo hace: Abre la base de datos en modo lectura

Usa this.getReadableDatabase() para obtener acceso de solo lectura.

Ejecuta una consulta SQL: Usa db.rawQuery(...) para ejecutar una instrucción SQL que:

Selecciona las columnas username y score de la tabla scores.

Ordena los resultados por la columna score en orden descendente (ORDER BY score DESC).

Limita el número de resultados a los 10 primeros (LIMIT 10).

Devuelve un Cursor: El objeto Cursor permite recorrer los resultados de la consulta.

## GameActivity

Gestiona la lógica principal del juego. Es responsable de registrar las interacciones del usuario (taps), actualizar la puntuación en tiempo real y almacenar los datos en la base de datos cuando se destruye la actividad. A continuación la parte más relevante del código.

```
tapButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        tapCount++;
        tapCounterTextView.setText(String.valueOf(tapCount));
        checkLevel();
    }
});
```

Qué hace: Incremento de tapCount: Cada vez que el usuario hace clic en el botón tapButton, el contador de taps (tapCount) se incrementa.

Actualización visual: El TextView (tapCounterTextView) se actualiza en tiempo real con el valor actual de tapCount.

Verificación del nivel: Llama al método checkLevel() para determinar si el usuario ha alcanzado un nuevo nivel basado en el número de taps.

Por qué es importante: Representa la interacción principal del usuario con el juego.

Combina lógica (incremento y verificación) con actualizaciones de la interfaz, proporcionando retroalimentación inmediata al usuario.

```
1 usage
private void checkLevel() {
    if (currentLevel < levelThresholds.length && tapCount >= levelThresholds[currentLevel]) {
        currentLevel++;
        updateButtonColor();
    }
}
```

Qué hace: Condición para cambiar de nivel

Comprueba si: currentLevel es menor que la longitud del array levelThresholds (aún hay niveles por alcanzar).tapCount supera o iguala el umbral para el siguiente nivel (levelThresholds[currentLevel]).

Actualización del nivel: Incrementa currentLevel en 1 si se cumple la condición.

Llama a updateButtonColor() para cambiar la apariencia del botón y notificar al usuario.

Por qué es importante: Implementa la lógica de progresión del juego, donde los niveles aumentan a medida que el usuario alcanza ciertos hitos (número de taps). Es una pieza clave

para mantener al usuario motivado mediante una mecánica de recompensas visuales (cambio de color y mensajes).

```
1 usage
private void updateButtonColor() {
    String levelMessage = "Level " + currentLevel;
    switch (currentLevel) {
        case 1:
            tapButton.setBackgroundColor(Color.YELLOW);
            Toast.makeText( context: this, levelMessage, Toast.LENGTH_SHORT).show();
            break;
        case 2:
            tapButton.setBackgroundColor(Color.GREEN);
            Toast.makeText( context: this, levelMessage, Toast.LENGTH_SHORT).show();
            break;
        case 3:
            tapButton.setBackgroundColor(Color.BLUE);
            Toast.makeText( context: this, levelMessage, Toast.LENGTH_SHORT).show();
            break;
        case 4:
            tapButton.setBackgroundColor(Color.MAGENTA);
            Toast.makeText( context: this, levelMessage, Toast.LENGTH_SHORT).show();
            break;
        case 5:
            tapButton.setBackgroundColor(Color.RED);
            Toast.makeText( context: this, levelMessage, Toast.LENGTH_SHORT).show();
            break;
        default:
            tapButton.setBackgroundColor(Color.LTGRAY);
            break;
    }
```

Qué hace: Mensaje del nivel: Define un mensaje para notificar al usuario del nuevo nivel alcanzado.

Cambio visual del botón: Usa un switch para asignar un color específico al botón tapButton basado en el nivel actual.

Notificación visual (Toast): Muestra un mensaje emergente indicando el nivel alcanzado.

Por qué es importante: Refuerza la mecánica de recompensa del juego:

Cambiar el color del botón proporciona un estímulo visual.

Mostrar un Toast notifica explícitamente al usuario que ha progresado al siguiente nivel.

Hace que el progreso en el juego sea tangible y emocionante para el usuario.

Resumen de la funcionalidad clave

Incremento y conteo de taps:

Cada tap se registra y se refleja en tiempo real en la interfaz.

Verificación de niveles:

El juego compara el número de taps con umbrales predefinidos para determinar si el usuario debe subir de nivel.

Actualización visual y notificaciones:

Al cambiar de nivel, se actualizan los colores del botón y se muestra un mensaje de progreso, mejorando la experiencia del usuario.

Estas tres partes juntas forman el núcleo interactivo del juego, asegurando que los usuarios sientan un progreso constante y tengan una experiencia inmersiva.

## MainActivity

El código de MainActivity es crucial porque establece la pantalla inicial de la aplicación y define la interacción principal del usuario con el sistema antes de acceder a otras funcionalidades, como jugar o consultar las puntuaciones. La clase ofrece una interfaz sencilla pero funcional que permite:

- Ingresar el nombre del usuario.
- Iniciar el juego con el nombre proporcionado.
- Navegar a la sección de puntuaciones para ver el ranking.

```
viewScoresButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: MainActivity.this, ScoreActivity.class);
        startActivity(intent);
    }
});
```

Qué hace: Ajusta dinámicamente el diseño de la pantalla principal para adaptarse a los contornos y barras del sistema (como barras de estado o navegación) en dispositivos modernos.

Cómo lo hace: ViewCompat.setOnApplyWindowInsetsListener:

Escucha los cambios en los márgenes y relleno de la ventana de la actividad.

Asegura que los elementos de la interfaz se ajusten correctamente a las barras del sistema.

Cálculo de los márgenes: Usa insets.getInsets(WindowInsetsCompat.Type.systemBars()) para determinar los márgenes necesarios.

Aplicación del relleno: Usa v.setPadding(...) para aplicar los márgenes calculados al View.

Por qué es importante: Garantiza una interfaz visual consistente y adaptada a dispositivos con distintas configuraciones de pantalla, como móviles con notch o pantallas de borde curvo.

```
startGameButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String username = usernameEditText.getText().toString();
        if (username.isEmpty()) {
            usernameEditText.setError("Username cannot be empty");
        } else {
            Intent intent = new Intent( packageContext: MainActivity.this, GameActivity.class);
            intent.putExtra( name: "username", username); // Pass the username
            startActivity(intent);
        }
```

Qué hace: Valida el nombre de usuario: Si el campo de texto está vacío, muestra un error en el EditText indicando que el nombre de usuario no puede estar vacío.

Navegación a GameActivity: Si el nombre de usuario es válido, crea un Intent para iniciar la actividad del juego (GameActivity).

Pasa el nombre de usuario a través del método intent.putExtra(...), asegurando que GameActivity pueda acceder a este dato.

Por qué es importante: Validación del usuario: Previene errores o incoherencias al garantizar que el usuario ingrese un nombre antes de continuar.

Interactividad: Proporciona una experiencia de usuario fluida al navegar al juego con un simple clic.

## ScoreActivity

La clase ScoreActivity está diseñada para gestionar la pantalla de puntuaciones. Su función principal es recuperar y mostrar las puntuaciones almacenadas en la base de datos, además de permitir al usuario restablecerlas. Este componente desempeña un papel clave al proporcionar a los usuarios acceso al historial de resultados del juego. A continuación, las partes más relevantes del código.

```
ListView scoreListView = findViewById(R.id.score_list);
ArrayList<String> scores = databaseHelper.getAllScores();

ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, scores);
scoreListView.setAdapter(adapter);
```

Qué hace esta sección:

1. Inicialización del ListView: Localiza el componente ListView (scoreListView) definido en el archivo de diseño activity_score.xml.

2. Recuperación de datos: Usa databaseHelper.getAllScores() para obtener un ArrayList<String> con todas las puntuaciones almacenadas en la base de datos SQLite.

3. Adaptador de datos: Crea un ArrayAdapter para convertir el ArrayList de puntuaciones en un formato que el ListView pueda mostrar.

Asocia este adaptador al ListView mediante setAdapter().

Por qué es importante:

Interfaz dinámica: Permite mostrar una lista de puntuaciones que se actualiza automáticamente al modificar la base de datos.

Integración con la base de datos: Recupera datos reales almacenados en SQLite, haciendo que la funcionalidad de la aplicación sea persistente y significativa.

```
Button resetScoreButton = findViewById(R.id.reset_score_button);
resetScoreButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        databaseHelper.deleteAllScores();
        scores.clear();
        adapter.notifyDataSetChanged();
    }
});
```

Qué hace esta sección:

1. Localización del botón: Encuentra el botón resetScoreButton en el archivo de diseño asociado.

2. Configuración del OnClickListener: Define una acción a realizar cuando el usuario haga clic en el botón.

3. Restablecimiento de datos: Llama al método deleteAllScores() de DatabaseHelper, que elimina todas las puntuaciones almacenadas en la base de datos.

Limpia el ArrayList<String> que contiene las puntuaciones (scores.clear()).

Notifica al adaptador que los datos han cambiado (adapter.notifyDataSetChanged()), lo que actualiza dinámicamente el ListView para reflejar el cambio.

Por qué es importante: Control de datos por el usuario: Permite al usuario restablecer las puntuaciones según lo desee, manteniendo el control sobre la información almacenada.

Actualización en tiempo real: Garantiza que la interfaz refleje inmediatamente los cambios realizados en los datos.

```
databaseHelper = new DatabaseHelper( context: this);

ListView scoreListView = findViewById(R.id.score_list);
ArrayList<String> scores = databaseHelper.getAllScores();
```

Qué hace esta sección:

1. Inicialización de DatabaseHelper: Crea una instancia de DatabaseHelper, que actúa como puente entre la aplicación y la base de datos SQLite.

2. Recuperación de puntuaciones: Usa el método getAllScores() de DatabaseHelper para obtener una lista de puntuaciones desde la base de datos.

Por qué es importante: Acceso centralizado a los datos:

DatabaseHelper encapsula toda la lógica de interacción con la base de datos, simplificando la gestión de datos en ScoreActivity.

Persistencia de datos: Garantiza que las puntuaciones no se pierdan al cerrar la aplicación, ya que están almacenadas en una base de datos.

## activity_game.xml

Este archivo XML define la interfaz gráfica de la actividad principal del juego (GameActivity). Utiliza un ConstraintLayout como contenedor principal, organizando los elementos de manera flexible y adaptable a diferentes tamaños de pantalla. Los elementos clave en esta interfaz incluyen un contador de taps y tres botones con diferentes funcionalidades (clicar, terminar partida y reiniciar partida).

```
<TextView
    android:id="@+id/tap_counter"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:textSize="48sp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>
```

Qué hace esta sección:

Propósito: Muestra la cantidad actual de taps realizados por el usuario durante la partida.

Su valor se actualiza dinámicamente desde GameActivity cada vez que el usuario hace clic en el botón "Clica".

Configuración visual:

android:text="0": Establece el valor inicial del contador en "0".

android:textSize="48sp": Aplica un tamaño grande al texto para garantizar visibilidad.

Posicionamiento: Centrado en la pantalla mediante restricciones (app:layout_constraintTop_toTopOf="parent", etc.).

Por qué es importante:

Este es el elemento visual principal que permite al usuario observar su progreso en tiempo real.

La configuración central y el tamaño grande mejoran la claridad y la accesibilidad, asegurando que sea fácil de leer incluso en dispositivos pequeños.

```
<Button
    android:id="@+id/tap_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Clica"
    app:layout_constraintTop_toBottomOf="@id/tap_counter"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>
```

Qué hace esta sección:

Propósito: Este botón captura la interacción principal del usuario. Cada clic en este botón incrementa el valor del contador de taps.

Configuración visual: android:text="Clica": Define el texto visible en el botón.

Posicionamiento: Colocado directamente debajo del contador de taps mediante la restricción app:layout_constraintTop_toBottomOf="@id/tap_counter".

Centrado horizontalmente en la pantalla.

Por qué es importante: Es el punto focal de la interacción del usuario con el juego.

Su posición estratégica debajo del contador asegura una experiencia intuitiva, guiando al usuario a ver el contador mientras hace clic.

## activity_main.xml

Este archivo XML define la interfaz de usuario para la actividad inicial de la aplicación, MainActivity, donde el usuario puede ingresar su nombre, iniciar una partida o consultar el ranking de puntuaciones. Utiliza un ConstraintLayout como contenedor principal para proporcionar un diseño flexible y adaptable.

```
<EditText
    android:id="@+id/edit_text_username"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="Introduce tu nombre"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@id/btn_start_game"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintVertical_chainStyle="packed"
    android:layout_margin="16dp"/>
```

Qué hace esta sección:

Propósito: Proporciona un campo de entrada para que el usuario introduzca su nombre antes de iniciar el juego.

Configuración visual: android:hint="Introduce tu nombre":

Muestra un texto guía dentro del campo de texto para indicar su propósito.
android:layout_width="0dp":

Expande el campo de texto para ocupar todo el ancho disponible entre los márgenes.

Restricciones: Está centrado verticalmente entre la parte superior del contenedor y el botón "Iniciar partida" mediante app:layout_constraintVertical_chainStyle="packed".

Por qué es importante:

Es el primer punto de interacción del usuario con la aplicación.

La validación del contenido de este campo es crucial para garantizar que el usuario proporcione un nombre antes de continuar.

```
<Button
    android:id="@+id/btn_view_scores"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ver puntuaciones"
    app:layout_constraintTop_toBottomOf="@id/btn_start_game"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="16dp"/>
```

Qué hace esta sección:

Propósito: Permite al usuario navegar a la pantalla de ranking de puntuaciones.

Configuración visual: android:text="Ver puntuaciones":

Define el texto visible en el botón.

Restricciones: Posicionado debajo del botón "Iniciar partida" (app:layout_constraintTop_toBottomOf="@id/btn_start_game").

Centrando horizontalmente el botón dentro del contenedor.

Por qué es importante:

Este botón ofrece acceso rápido al ranking, permitiendo al usuario consultar las puntuaciones almacenadas.

Refuerza la funcionalidad de la aplicación al ofrecer opciones claras desde la pantalla principal.

El archivo activity_main.xml define una interfaz clara y funcional que facilita la interacción inicial del usuario con la aplicación. Su diseño está centrado en la experiencia del usuario,

ofreciendo opciones bien definidas y una navegación intuitiva. La combinación de restricciones precisas y un diseño minimalista asegura una experiencia accesible y visualmente agradable en cualquier dispositivo.

## activity_score.xml

El archivo activity_score.xml define la interfaz de usuario para la actividad ScoreActivity, que permite a los usuarios ver el ranking de puntuaciones y restablecer los datos almacenados. Utiliza un ConstraintLayout como contenedor principal, que organiza los elementos de manera adaptable y moderna.

```
<ListView
    android:id="@+id/score_list"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@id/reset_score_button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>
```

Qué hace esta sección:

Propósito: Muestra las puntuaciones almacenadas en la base de datos.

Permite al usuario ver el ranking en una lista dinámica.

Configuración visual: android:layout_width="0dp" y android:layout_height="0dp":

Define un tamaño dinámico ajustado a las restricciones de diseño (match constraints).

Restricciones: La lista ocupa todo el espacio disponible entre la parte superior del contenedor (parent) y el botón "Reiniciar" (reset_score_button).

Por qué es importante: Es el elemento central de esta pantalla, proporcionando una visión clara y ordenada de las puntuaciones.

La configuración dinámica garantiza que la lista se adapte a diferentes tamaños y orientaciones de pantalla.

```
<Button
    android:id="@+id/reset_score_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Reiniciar "
    app:layout_constraintTop_toBottomOf="@id/score_list"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>
```

Qué hace esta sección:

Propósito: Permite al usuario eliminar todas las puntuaciones almacenadas en la base de datos.

Configuración visual: android:text="Reiniciar ":

Define el texto visible en el botón.

Restricciones: Posicionado justo debajo de la lista de puntuaciones (score_list).

Centrado horizontalmente y alineado con el borde inferior del contenedor (parent).

Por qué es importante:Brinda al usuario control sobre los datos, permitiéndole limpiar el ranking si lo desea.

Su posición debajo de la lista y centrada lo hace fácilmente accesible sin interferir con la visualización del ranking.

## AndroidManifiest.xml

El archivo AndroidManifest.xml es un componente esencial en cualquier proyecto Android. Actúa como un "mapa" que define la estructura y configuración de la aplicación, declarando actividades, permisos, temas y configuraciones globales. A continuación las partes más relevantes del código.

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="PM_HFinal_1T_AngelGallegoFelipe"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.PM_HFinal_1T_AngelGallegoFelipe"
    tools:targetApi="31">
```

Qué hace esta sección:

1. Propiedades principales:

android:allowBackup="true": Permite a los usuarios hacer copias de seguridad y restaurar datos de la aplicación en dispositivos compatibles.

android:dataExtractionRules y android:fullBackupContent: Especifican reglas para la copia de seguridad de datos, incluyendo qué información debe respaldarse.

android:icon y android:roundIcon: Definen los íconos de la aplicación para las interfaces de usuario del sistema.

android:label: Establece el nombre de la aplicación que aparece en la pantalla principal del dispositivo.

android:supportsRtl="true": Habilita el soporte para idiomas que se escriben de derecha a izquierda (RTL), como árabe o hebreo.

android:theme: Asigna el tema visual predeterminado de la aplicación.

Compatibilidad API: tools:targetApi="31":

Indica que la aplicación apunta a la API 31 (Android 12) para aprovechar características modernas.

Por qué es importante: Proporciona configuraciones globales para toda la aplicación.

Mejora la experiencia del usuario al permitir copias de seguridad y adaptaciones visuales.

```
<activity
    android:name=".ScoreActivity"
    android:exported="false" />
<activity
    android:name=".GameActivity"
    android:exported="false" />
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Qué hace esta sección:

1. Declaración de actividades:

Cada actividad de la aplicación debe registrarse en el manifiesto para que el sistema la reconozca.

android:name: Define el nombre de la actividad. Por ejemplo:

- .ScoreActivity: Muestra el ranking de puntuaciones.
- .GameActivity: Contiene la lógica principal del juego.
- .MainActivity: Es la pantalla inicial de la aplicación.

android:exported:Indica si la actividad está disponible para otras aplicaciones.

- false: La actividad solo puede ser llamada internamente.
- true: Permite que otras aplicaciones accedan a esta actividad (solo MainActivity en este caso).

<intent-filter> en MainActivity:

android.intent.action.MAIN:

Declara esta actividad como el punto de entrada principal de la aplicación.

android.intent.category.LAUNCHER:

Hace que esta actividad sea visible en la pantalla de inicio del dispositivo como un icono de aplicación.

Por qué es importante:

- Define cómo el sistema operativo interactúa con la aplicación.
- Garantiza que el usuario pueda iniciar la aplicación desde la pantalla principal.
- Controla el acceso a las actividades, mejorando la seguridad y modularidad.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
```

Qué hace esta sección:

Espacio de nombres: Declara el espacio de nombres de Android ( xmlns:android), necesario para interpretar correctamente los atributos.

Compatibilidad con herramientas: xmlns:tools:

Introduzca atributos utilizados por herramientas de desarrollo, como tools:targetApi.

Por qué es importante: Defina la estructura básica del archivo y habilite el uso de herramientas modernas durante el desarrollo.