

MusiKe

Mario López Sáez

6 de julio de 2025






Índice

1	Descripción del Proyecto	2
2	Estructura del Proyecto	2
3	Assets	2
3.1	Instrumentos 🎵	2
3.2	Iconos de Instrumentos	3
3.3	Audios de Instrumentos	3
3.4	Otros Recursos Gráficos 🎨	3
4	Activities	4
4.1	MainActivity 🏠	4
4.2	StudyActivity 📖	4
4.3	RoundSelectionActivity 🔄	4
4.4	GameActivity 🎮	4
4.5	FinishActivity 🏁	5
4.6	CircleProgressView 🕒	5
5	Aspectos Técnicos	5
5.1	Vinculación de Recursos y Datos	5
5.2	Evolución de la Lógica del Juego	5
5.3	Gestión de la Interfaz y el Estado	5
5.4	Gestión de la Orientación de Pantalla	6

1 Descripción del Proyecto

MusiKe es un juego educativo e interactivo diseñado para que usuarios de todas las edades descubran y aprendan los instrumentos de la orquesta a través de divertidos retos y minijuegos. La aplicación está desarrollada para Android usando Java y la librería AppCompat.

2 Estructura del Proyecto






-  **app/src/main/java/mario/khz/musike**: contiene las actividades y clases principales de la aplicación.
-  **app/src/main/res/drawable**: recursos gráficos vectoriales (iconos de instrumentos, fondos y botones).
-  **app/src/main/res/raw**: archivos de audio con sonidos de los instrumentos de la orquesta.
-  **README.md**: descripción general del proyecto y logo.
-  **tex/MusiKe.tex**: este documento de documentación.

3 Assets

3.1 Instrumentos

A continuación se listan los instrumentos incluidos en la aplicación, junto a sus recursos gráficos y de audio:

-  Violín: **res/drawable/violin.xml**, **res/raw/violin.mp3**
-  Viola: **res/drawable/viola.xml**, **res/raw/viola.mp3**
-  Violonchelo: **res/drawable/cello.xml**, **res/raw/cello.mp3**
-  Contrabajo: **res/drawable/contrabajo.xml**, **res/raw/contrabajo.mp3**
-  Piano: **res/drawable/piano.xml**, **res/raw/piano.mp3**
-  Flauta: **res/drawable/flauta.xml**, **res/raw/flauta.mp3**
-  Oboe: **res/drawable/oboe.xml**, **res/raw/oboe.mp3**
-  Clarinete: **res/drawable/clarinete.xml**, **res/raw/clarinete.mp3**
-  Fagot: **res/drawable/fagot.xml**, **res/raw/fagot.mp3**
-  Trompeta: **res/drawable/trompeta.xml**, **res/raw/trompeta.mp3**
-  Trombón: **res/drawable/trombon.xml**, **res/raw/trombon.mp3**
-  Trompa: **res/drawable/trompa.xml**, **res/raw/trompa.mp3**
-  Tuba: **res/drawable/tuba.xml**, **res/raw/tuba.mp3**
-  Arpa: **res/drawable/arpa.xml**, **res/raw/arpa.mp3**
-  Marimba: **res/drawable/marimba.xml**, **res/raw/marimba.mp3**

-  Vibráfono: res/drawable/vibrafono.xml, res/raw/vibrafono.mp3
-  Pequeña percusión: res/drawable/pequena_percusion.xml, res/raw/pequena_percusion.mp3
-  Platos de choque: res/drawable/platos_de_choque.xml, res/raw/platos_de_choque.mp3
-  Campanólogo: res/drawable/campanologo.xml, res/raw/campanologo.mp3
-  Caja: res/drawable/caja.xml, res/raw/caja.mp3

3.2 Iconos de Instrumentos

Los iconos de los instrumentos han sido obtenidos de <https://www.freepik.com/> y procesados posteriormente con el siguiente código:

```

1  COLOR=white
2  for f in *.png; do
3      name="{f%.png}"
4
5      # 1 PNG -> PNM compatible con Potrace
6      pngtopnm -mix "$f" > "$name.pnm"
7      # 2 Vectoriza con Potrace
8      potrace "$name.pnm" -s -o "$name.svg"
9      rm "$name.pnm"
10
11     # 3 Quita fills/strokes existentes y añade blanco
12     sed -i -E \
13         -e 's/fill="[^"]*"//g' \
14         -e 's/stroke="[^"]*"//g' \
15         -e "s/<(path|rect|circle|ellipse|polygon|polyline)/& fill=\"$COLOR\" stroke=\"$COLOR\"/gI" \
16         "$name.svg"
17 done

```

Figura 1: Código de procesamiento de SVG.

3.3 Audios de Instrumentos

Los audios han sido obtenidos de un CD que me ha prestado un profesor de Música para la realización de este proyecto.

3.4 Otros Recursos Gráficos 🎨

Además de los iconos de los instrumentos, la aplicación utiliza otros recursos gráficos para construir una interfaz de usuario coherente y atractiva:

- **Fondos con degradado:** Las pantallas principales de la aplicación utilizan fondos con un suave degradado diagonal. Estos se definen en res/drawable/ como ficheros XML de tipo <shape>, lo que permite una representación eficiente y escalable sin necesidad de imágenes de mapa de bits.
- **Botones redondeados:** Todos los botones interactivos presentan esquinas redondeadas para una estética moderna y amigable. Al igual que los fondos, se definen mediante XML de tipo <shape>, especificando el radio de las esquinas (<corners>) y el color de fondo.
- **Iconos de la interfaz:** Para mejorar la navegación y la comprensión visual, se utilizan iconos vectoriales en varios puntos de la aplicación (ej: en los botones del menú principal). Estos iconos provienen de la librería de código abierto *Tabler Icons* (también en este documento) y están integrados como VectorDrawable en formato XML, garantizando una visualización nítida en cualquier densidad de pantalla.

4 Activities

La arquitectura de la aplicación se articula en torno a un conjunto de Activities, cada una responsable de una pantalla y una funcionalidad concreta. A continuación, se detalla el propósito y funcionamiento de cada una de ellas.

4.1 MainActivity

Es la actividad principal y el punto de entrada de la aplicación. Su función es presentar al usuario la pantalla de bienvenida y actuar como un distribuidor central, ofreciendo las dos modalidades principales de la aplicación:

- **Modo Estudio:** Inicia StudyActivity para el aprendizaje libre.
- **Modo Juego:** Lanza RoundSelectionActivity para comenzar una partida.

4.2 StudyActivity

Esta actividad está diseñada para el aprendizaje y la consulta. Proporciona una interfaz donde el usuario puede explorar los instrumentos de la orquesta. Al seleccionar un instrumento, la actividad reproduce su sonido característico, permitiendo una familiarización auditiva y visual sin la presión de un entorno de juego.

Para la presentación visual de los instrumentos, esta actividad utiliza un componente auxiliar llamado StudyGridAdapter, que se encarga de:

- **Estructurar la cuadrícula:** Organiza los instrumentos en una rejilla de 4 columnas mediante un GridLayout.
- **Crear celdas interactivas:** Genera para cada instrumento un contenedor con un botón de imagen que muestra el icono vectorial del instrumento y una etiqueta con su nombre.
- **Gestionar la reproducción:** Implementa los listeners que reproducen el sonido correspondiente cuando el usuario pulsa sobre un instrumento.

4.3 RoundSelectionActivity

Actúa como una antesala al modo de juego. Su único propósito es permitir al usuario configurar la partida, seleccionando el número de rondas que desea jugar. Una vez que el usuario ha hecho su elección, esta actividad inicia la GameActivity, pasándole la configuración seleccionada como un *extra* en el Intent.

4.4 GameActivity

Es el núcleo interactivo de MusiKe. Esta actividad gestiona la lógica del juego de identificación de instrumentos, que se desarrolla de la siguiente manera:

- **Inicio y carga:** Al comenzar, lee el fichero `assets/instruments.json` para cargar la información de los instrumentos. A continuación, baraja la lista para crear un orden aleatorio de preguntas para la partida.
- **Desarrollo de la ronda:** En cada ronda, reproduce el sonido de un instrumento y presenta al usuario cuatro botones con nombres de instrumentos como posibles respuestas. Para aumentar el desafío, las opciones incorrectas se eligen preferentemente de la misma familia que la respuesta correcta.
- **Interacción y feedback:** El usuario selecciona una opción. La aplicación le proporciona feedback visual inmediato (verde para acierto, rojo para error). Si falla, se le indica cuál era la respuesta correcta. Tras una breve pausa, la actividad avanza automáticamente a la siguiente ronda.
- **Finalización:** Una vez completadas todas las rondas, GameActivity recopila las estadísticas de la partida (número de aciertos, respuestas del usuario y respuestas correctas) y lanza la FinishActivity para mostrar los resultados.

4.5 FinishActivity 🦉

Esta es la pantalla final del ciclo de juego. Su objetivo es mostrar al usuario un resumen de su rendimiento en la partida. Presenta la puntuación final (ej: "8 de 10 aciertos") y una tabla detallada que muestra las respuestas correctas para cada ronda, reforzando así el aprendizaje.

4.6 CircleProgressView 🕒

Aunque no es una Activity, es un componente de interfaz de usuario (View) personalizado de vital importancia en `GameActivity`. Su función es mostrar una barra de progreso circular que se actualiza en tiempo real, sincronizada con la reproducción del audio del instrumento. Esto proporciona al usuario una referencia visual clara de la duración del sonido y del tiempo restante para responder, mejorando significativamente la experiencia de juego.

5 Aspectos Técnicos

En esta sección se abordan decisiones de implementación y detalles técnicos clave que definen la robustez y escalabilidad de la aplicación.

5.1 Vinculación de Recursos y Datos

Se ha adoptado una estrategia de nomenclatura unificada para los recursos. Cada instrumento definido en `assets/instruments.json` tiene un campo `file` (ej: "violin") que actúa como identificador único. Este identificador se corresponde directamente con el nombre de los ficheros de recursos asociados:

- **Recurso gráfico:** `res/drawable/violin.xml`
- **Recurso de audio:** `res/raw/violin.mp3`

Esta convención permite cargar dinámicamente los recursos en el código Java mediante `getResources().getIdentifier()`, simplificando enormemente la gestión y la adición de nuevos instrumentos, ya que no es necesario modificar el código para vincular los nuevos assets.

5.2 Evolución de la Lógica del Juego

La mecánica de selección de preguntas en `GameActivity` ha pasado por una mejora significativa:

- **Implementación inicial:** En una primera fase, en cada ronda se seleccionaba un instrumento de forma completamente aleatoria de la lista total. Este enfoque, aunque sencillo, permitía que un mismo instrumento pudiera aparecer varias veces en una misma partida, resultando en una experiencia de juego repetitiva.
- **Implementación final:** Para solucionar esto, se implementó el método `generarOrdenRondas()`. Al inicio de cada partida, este método crea una lista con todos los instrumentos, la baraja de forma aleatoria y la almacena. Las rondas posteriores consumen instrumentos de esta lista pre-barajada, garantizando que cada instrumento aparezca como máximo una vez por partida, lo que resulta en una experiencia más variada.

5.3 Gestión de la Interfaz y el Estado

La aplicación gestiona el estado y la interfaz de usuario de manera eficiente:

- **Paso de datos:** La comunicación entre Activities se realiza a través de `Intent extras`, un mecanismo estándar en Android para pasar datos como el número de rondas o los resultados finales.
- **Actualización en tiempo real:** El progreso de la reproducción de audio se visualiza en el `CircleProgressView` gracias a un `Handler` que actualiza la interfaz en segundo plano a intervalos regulares, proporcionando una respuesta visual fluida y sin bloquear el hilo principal.
- **Liberación de recursos:** En el método `onDestroy()` de la actividad, se liberan explícitamente los recursos del `MediaPlayer` y se detienen las actualizaciones del `Handler`. Esta práctica es fundamental para evitar fugas de memoria (*memory leaks*) y asegurar el buen rendimiento de la aplicación.

5.4 Gestión de la Orientación de Pantalla

Se ha tomado una decisión estratégica respecto a la rotación de pantalla para optimizar la experiencia de usuario y la eficiencia del desarrollo:

- **Bloqueo general a modo retrato:** La mayoría de las actividades de la aplicación, incluyendo `GameActivity`, están configuradas para funcionar exclusivamente en orientación vertical (retrato).
- **Excepción de `StudyActivity`:** La única excepción es la `StudyActivity`, que permite la rotación. Su diseño, basado en un `ScrollView`, se adapta de forma natural a la orientación apaisada sin perjudicar la usabilidad.
- **Prevención de reinicio en `GameActivity`:** La razón principal para bloquear la rotación en la `GameActivity` es evitar el reinicio del ciclo de vida de la actividad que ocurre por defecto en Android al girar el dispositivo. Dicho reinicio provocaría la pérdida completa del estado de la partida (rondas, puntuación, etc.). Implementar la lógica para guardar y restaurar este estado (mediante `onSaveInstanceState`) habría añadido una complejidad considerable al desarrollo. Por tanto, forzar la orientación vertical se consideró una solución más pragmática y robusta para garantizar la continuidad del juego.