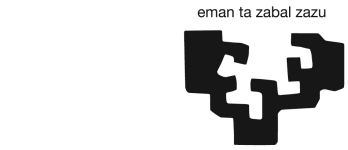

Ingeniería Informática de Gestión y Sistemas de Información

Desarrollo Avanzado de Software

RedJack

Autor:
Luken Bilbao Rojo

16 de marzo de 2025

Índice

1. Introducción

RedJack es una aplicación para dispositivos móviles desarrollada mediante el framework de desarrollo nativo Android Studio. El objetivo del programa desarrollado es ser entretenimiento para el usuario, el cual podrá jugar a una variante del clásico juego de cartas Blackjack.

Además de esto, este proyecto tiene como objetivo evaluar las capacidades de su autor para poner en práctica técnicas y métodos relativos a Android Studio vistos en clase, a fin de lograr una aplicación funcional, eficiente e intuitiva.

Cabe destacar que la aplicación está desarrollada en Java en su totalidad, utilizando Android Studio y siguiendo además las mejores prácticas de desarrollo para Android.

Nótese que este proyecto se ha llevado a cabo entorno a los criterios en el enunciado del documento 'Primera entrega individual'. Aclaro este punto porque **algunos de los criterios establecidos en la entregan difieren de ls presentes en dicho documento**.

2. Información general

En esta breve sección se expondrá información relevante en lo que a pruebas se refiere.

- **GitHub de la aplicación**: https://github.com/cursedx0/DAS-RedJack (primera entrega en branch 'master').
- **Usuario de pruebas**: user
- **Contraseña de pruebas**: pw

3. Objetivos alcanzables

En esta sección se resumirán los objetivos alcanzados y su papel dentro del contexto de la aplicación.

3.1. Objetivos obligatorios

	Uso de ListView+CardView personalizado o de RecyclerView+CardView para mostrar listados de elementos con diferentes características.	Realizado. Las cartas de la mano del jugador y del dealer se encuentran en una ListView, siendo cada carta un CardView.
	Usar una base de datos local, para listar, añadir y modificar elementos y características de cada elemento.	Realizado. La base de datos almacena los distintos usuario, así como su saldo de monedas correspondiente.
	Uso de diálogos.	Realizado. Al acabar la partida, el usuario recibe un diálogo dando a conocer el resultado y un resumen de sus ganancias o pérdidas, así como su saldo actual.
	Usar notificaciones locales.	Realizado. El usuario recibe una notificación al iniciar sesión en la aplicación. Esto puede desactivarse desde la sección de preferencias.
	Control de la pila de actividades.	Realizado. La creación de nuevas actividades y los cambios de pantalla no suponen una sobrecarga de la pila de actividades ni generan problemas internos en la aplicación.

Cuadro 1: Objetivos obligatorios del proyecto.

3.2. Objetivos adicionales

	Permitir que una misma funcionalidad se comporte de manera distinta dependiendo de la orientación (o del tamaño) del dispositivo mediante el uso de Fragments (1 punto).	No realizado.
	Hacer la aplicación multiidioma y añadir la opción de cambiar de idioma en la propia aplicación (1 punto).	Realizado. El idioma de la aplicación puede alternarse entre español e inglés desde la sección de preferencias.
	Uso de ficheros de texto (1 punto).	Realizado. Se usa un fichero history.txt para guardar registro de las últimas 20 manos jugadas. Puede consultarse en la sección Historial.
	Uso de Preferencias, para guardar las preferencias del usuario en cuanto a mostrar/esconder cierta información,elegir colores para la aplicación, o cualquier otra cosa relacionada con la visualización de la aplicación (0.5 punto).	Realizado. En la sección de preferencias pueden personalizarse opciones como el tema y el idioma, así como activar o desarctivar la notificación de inicio de sesión.
	Crear estilos y temas propios, para personalizar fondos, botones, etc. (0,5 puntos).	Realizado. En la sección de preferencias puede elegirse entre 4 temas disponibles.
	Usar intents implícitos para abrir otras aplicaciones, contactos, etc. (0,5 puntos).	Realizado. Al terminar una partida, se le da la posibilidad al usuario de compartir su resultado mediante el botón Compartir.
	Añadir una barra de herramientas (ToolBar) personalizada en la aplicación así como un panel de navegación (Navigation Drawer) (1 punto).	Parcialmente realizado. La aplicación dispone de una barra de herramientas desde la que se accede al menú oculto, pero carece de panel de navegación.

Cuadro 2: Objetivos adicionales del proyecto.

4. Diseño

En esta sección se describirán las clases de Java utilizadas más relevantes para el correcto funcionamiento de la aplicación y sus atributos y métodos más

importantes, así como un breve sección explicativa sobre la base de datos y sus usos.

4.1. Clases de Java

4.1.1. Baraja.java

La clase Baraja es la encargada de gestionar todas las cartas sobre la mesa y fuera de ella. Fisicamente representaría, valga la redundancia, una baraja francesa de 52 cartas (no hay comodines).

Se utiliza exclusivamente en la actividad PlayActivity, donde el jugador se enfrentará al dealer en una variante del Blackjack.

Estos son los atributos que contiene:

- **private List<String> mazo**: esta lista de strings representará la totalidad de la baraja inicialmente. Las cartas que contendrá (representada cada un con una string única) entrarán y saldrán de ella a lo largo de las partidas. Podría decirse que representa la lista de toda carta fuera de la mesa.
- **private List<String> manoDealer**: otra lista de cartas, pero esta vez serán las cartas en la mano del dealer.
- **private List<String> manoJugador**: otra lista de cartas, pero esta vez serán las cartas en la mano del jugador.
- **private static final String[] PALOS**: esta lista representa los palos de la baraja francesa (ç"para çlover", "d"para "diamond", "h"para "heartsƶ "s"para "spades"). Se utilizará para inicializar las cartas en el mazo.
- **private static final String[] VALORES**: esta lista representa los números de la baraja francesa, incluyendo la .ª"para el as y "J", "Qƶ "K"para sus cartas homónimas. Se utilizará para inicializar las cartas en el mazo.

Para trabajar con estos elementos, se hará uso de los siguiente métodos:

- **public Baraja()**: constructora de la clase. Simplemente inicializa los atributos a nuevas listas de strings y ejecuta el método inicializar baraja.
- **public void inicializarMazo()**: contruye el mazo usando un par de bucles for, recorriendo el atributo PALOS y VALORES, formando strings con el formato PALO+VALOR. Por ejemplo, el siete de tréboles sería 'c7'.
- **public void barajar()**: mediante Collections.shuffle(), desordena las cartas en el mazo.

- **public boolean repartirJugador()**: se retira una carta del mazo (la primera) y se introduce en la mano del jugador.
- **public boolean repartirDealer()**: mismo concepto que el anterior método, pero para la mano del dealer.
- **public int[] calcMano(List<String> mano)**: método utilizado para calcular el valor de una mano introducida como parámetro. A pesar de que a primera vista pueda parecer un método meramente computacional, debemos tener en cuenta que el valor del as varía según lo decida el dueño de la mano. Es por esto que se devuelve una lista de enteros y no un simple entero.
- **public List<String> getMazo()**: método getter para acceder al mazo.
- **public List<String> getManoJugador()**: método getter para acceder a la mano del jugador.
- **public List<String> getManoDealer()**: método getter para acceder a la mano del dealer.

4.1.2. BaseActivity.java

Esta actividad servirá de base para la creación del resto de actividades de la aplicación. Casi todas las actividades heredarán de esta. Este tipo de estructura resulta útil a la hora de aplicar cambios en todas las actividades, especialemente cambios provocados por ajustes em ñas preferencias de la aplicación.

Estos son los atributos que contendrá:

- **protected boolean NOTIS_LOGIN**: booleano que determina si las notificaciones al iniciar sesión están habilitadas. Su valor se comprueba antes de lanzarse la notificación en PlayActivity.
- **protected static final String FILE_NAME**: string que define el nombre que tomará el archivo de texto con el historial de partidas.

Estos serán los métodos que usará:

- **protected void onCreate(Bundle savedInstanceState)**: método onCreate. Obtiene preferencias como el tema o la preferencia de notificaciones antes de creat la actividad.
- **protected void attachBaseContext(Context newBase)**: obtiene el idioma de las preferencias y cambia el idioma de la aplicación haciendo uso del método setLocale explicado más adelante.
- **protected void setLocale(String languageCode)**: método complementario del anterior que además recrea la actividad para aplicar el cambio de idioma.

- **protected void setThemeMode(String theme, boolean reset)**: método encargado de establecer el tema. Será llamado desde las preferencias con el tema a establecer como parámetro.
- **public boolean onCreateOptionsMenu(Menu menu)**: método de inflado de toolbar sobreescrito.
- **public boolean onOptionsItemSelected(MenuItem item)**: otro método de toolbar sobreescrito.

4.1.3. HistoryActivity.java

Esta actividad será la encargada de mostrar el historial de partidas. Podría tener definico como atributo el textView en el que se imprime el historial, pero dado el número reducido de accesos a este no lo he considerado necesario.

Estos serán los métodos que implementará:

- **protected void onCreate(Bundle savedInstanceState)**: método onCreate. Accede al textView por id y establece el onClickListener del botón de retroceso.
- **private String leerHistorial()**: método encargado de obtener el historial y escribirlo en el textView. En caso de darse un error (por ejemplo, si no se ha jugado ninguna partida y no haya archivo de historial), se mostrará un mensaje diciendo que no hay historial disponble.

4.1.4. MainActivity.java

Actividad en la que se lanzará la aplicación. Esta mostrará una página de login, en la que el usuario deberá introducir su usuario y contraseña para acceder a su sesión.

Estos son los métodos que usará:

- **protected void onCreate(Bundle savedInstanceState)**: método onCreate. Encargado de toda la lógica de esta actividad, que será crear una instancia de la clase miBD y comprobar si el usuario existe al pulsarse el botón de login. En caso de no existir devolverá un mensaje de error. En caso de existir e introducirse la contraseña correcta, se recogerán de la base de datos la información de dicho jugador para pasársela a la actividad PlayActivity. Esta información (como el id del usuario o sus monedas) será importante para que cada usuario tenga sus propias pérdidas y ganancias.

4.1.5. PlayActivity.java

Actividad responsable de toda la lógica de juego. En ella residen botones, inputs, una toolbar y demás elementos visuales que no mencionaré entre sus atributos, así como pequeños métodos con funciones poco relevantes para este escrito.

Estos son algunos de sus atributos:

- **private boolean jugandoFlag**: usado para saber cuando el jugador está jugando. Se establece a 'false' cuando el jugador recibe el diálogo de victoria, derrota o empate. Usado para advertir al jugador antes de cerrar sesión durante una mano.
- **private boolean enablePrefs**: flag similar a la anterior, pero solo es 'true' cuando el jugador se encuentra en la pantalla de introducir su apuesta. Utilizada para deshabilitar las preferencias durante las partidas (buscando la complejidad técnica).
- **private int apuesta**: apuesta realizada en la mano actual.
- **private int saldo**: saldo actual del jugador.
- **private int id**: id del usuario en la base de datos.
- **private int[] posiblespuntos**: array de enteros que define el número menor y mayor de puntos que puede tener el jugador cuando tiene un as en su mano. Existe un equivalente para el dealer. Dado que el as puede valer 1 u 11 puntos, no tiene sentido darle el valor de 11 a 2 ases, porque sumarían 22 y el jugador perdería autommáticamente. Es por esto que solo hacen falta dos posibles valores. La tercera posición del array se utiliza para llevar la cuenta de cuántos ases hay en la mano.
- **private int puntos**: una vez el jugador elige una puntuación entre las dos posibles, esta se guarda aquí.
- **private int totalElegido**: puntuación elegida por el jugador. Puede valer 0, 1 o 2 si no se ha elegido aun.

Algunos de los métodos utilizados en esta clase son los siguientes:

- **protected void onCreate(Bundle savedInstanceState)**: método onCreate. Encargado de establecer los onClickListeners y mandar la notificación de inicio de sesión (si así lo dictan las preferencias).
- **private void empezarPartida()**: encargado de establecer la UI y repartir cartas al principio de la partida. También calcula el valor de la mano del jugador y retira de la base de datos tantas monedas como las que se apuestan. Esto asegura que el jugador no saldrá de la partida cuando se vea encerrado para evitar perder dinero.

- **private void comprobarAses()**: función encargada de comprobar la incorporación de nuevos ases a la mano y de su gestión si es necesaria. Por ejemplo, si aparece un as y elegimos el valor bajo (1), al salirnos un segundo as deberemos elegir de nuevo si queremos que valga 1 u 11. Si en este caso eligieramos 11, al salir un tercer as no se nos preguntaría por el valor a seleccionar.

- **private void compartirResultado(int resultado)**: método llamado al pulsar el botón de compartir, que llamará a un intent explícito para compartir nuestro resultado y lo bien que lo pasamos jugando a este juego.

- **public void guardarPartida(String resultadoPartida)**: método encargado de registrar el resultado de la partida en el archivo historial. Las nuevas partidas se escriben en la primera línea para mejor visualización en la HistoryActivity.

- **public boolean onOptionsItemSelected(MenuItem item)**: método con el objetivo de gestionar las pulsaciones de los MenuItems guardados en el menú oculto de la toolbar.

4.1.6. Preferencias.java

Clase que define el fragment de preferencias, desplegable desde el menú oculto de la toolbar de PlayActivity.

Estos pocos métodos bastarán para su correcto funcionamiento:

- **public void onCreatePreferences(@Nullable Bundle savedInstanceState, @Nullable String rootKey)**: método onCreate. Define la preferencia salirPref, que funciona como botón de retroceso eliminando el fragment.

- **public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String s)**: método que controla los cambios en las preferencias y actúa en consecuencia, cambiando el idioma, el tema, etcétera.

- **public void onResume()**: método que registra el listener.

- **public void onPause()**: método que quita del registro el listener.

4.2. Base de datos

La base de datos consta de una sola tabla que almacena cada usuario y su saldo de monedas. Dicha tabla tendrá las siguientes columnas:

- **'Id' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL**: id que identificará a cada usuario. Utilizado a la hora de hacer operaciones UDPATE con el saldo del jugador tras las partidas.

- **'Nombre' VARCHAR(255)**: nombre de usuario. Utilizado en el login y en la notificación personalizada.
- **'Pw' VARCHAR(255)**: contraseña del usuario. Utilizada en el inicio de sesión.
- **'Coins' INTEGER**: número de monedas del usuario. Utilizado para apostar en el juego.

5. Manual de uso

En esta sección se explicará el funcionamiento y flujo habitual de la aplicación con algunas capturas de pantalla de esta.

1. Al entrar en la aplicación, nos encontramos con una pantalla para iniciar sesión. Las credenciales de prueba se encuentran el la sección Ïnformación general".

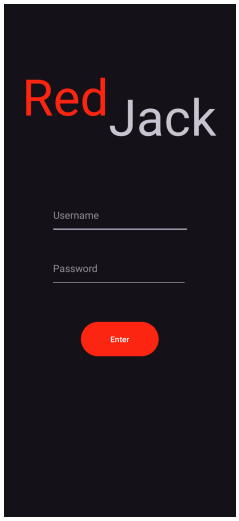

Figura 1: MainActivity.java. El tema e idioma por defecto pueden diferir de los mostrados en las imágenes.

2. Al introducir unas credenciales válidas, se nos muestra la siguiente pantalla. Se nos da la bienvenida con una notificación personalizada con nuestro nombre de usuario. Abajo, podemos ver un input para introducir un entero que representará nuestra apuesta y un botón para empezar el juego (siempre que nuestra apuesta sea suficiente sin sobrepasar nuestro saldo).

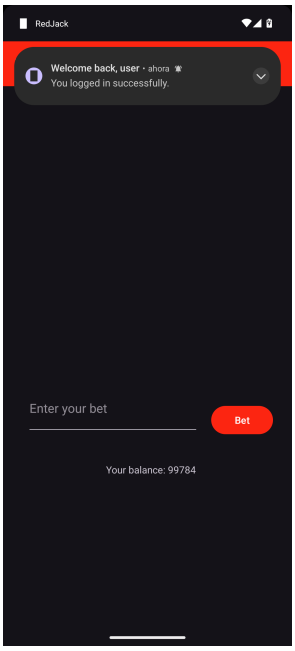

Figura 2: PlayActivity.java. La notificación puede desactivarse desde las preferencias.

3. Arriba a la derecha, en la toolbar, podemos ver un icono de 3 puntos. Esto es el menú oculto, que contiene 3 opciones.

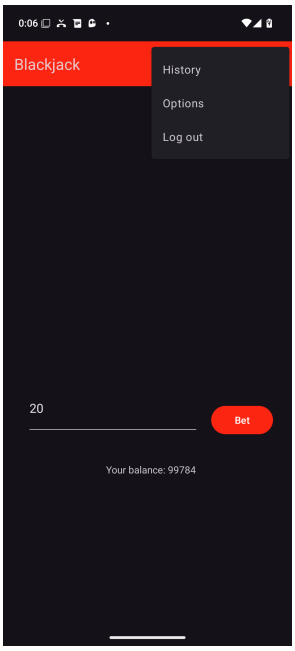

Figura 3: Menú oculto expandido.

a) El primer ítem del menú es el historial. Aquí podremos consultar los detalles de nuestras últimas 20 partidas. Puede deslizarse el dedo para ver registros más antiguos.

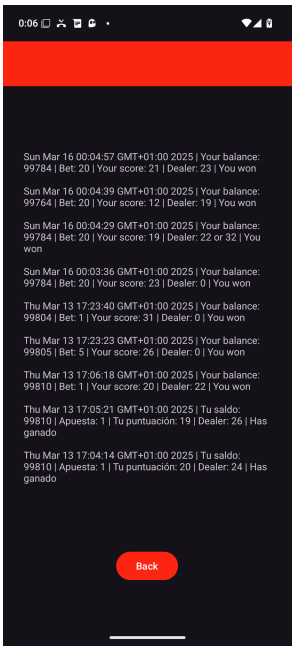

Figura 4: Historial con partidas. Las partidas se registran con el idioma en el que se jugaron. De no haberse jugado ninguna, se mostrará un mensaje indicándolo.

b) Como segundo ítem tenemos las preferencias. Al seleccionarlas, se desplegará sobre la pantalla un pequeño menú con opciones a configurar, como el tema o el idioma. Estos dos despliegan un diálogo de selección única y modifican la actividad actual y todas en adelante. La tercera preferencia desactiva las notificaciones de inicio de sesión. La última preferencia esconde el menú.

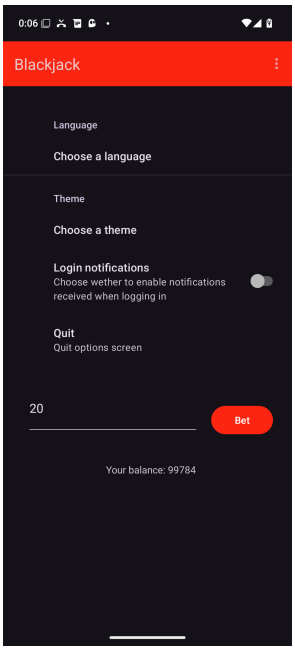

Figura 5: Preferencias expandidas.

- Cabe recalcar que las preferencias estarán deshabilitadas durante una mano. Se podrán acceder de nuevo una vez el usuario esté en la pantalla de apuesta.

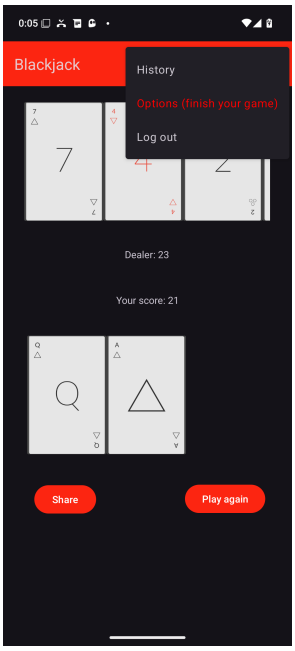

Figura 6: Preferencias deshabilitadas. Debemos acabar la partida del todo antes de poder acceder a ellas de nuevo. Esto solamente se hace por buscar complejidad técnica.

c) El último ítem es el de cerrar sesión, que borra la actividad actual y nos devuelve a la actividad de inicio de sesión (figura 1). En caso de que estemos en medio de una mano, se nos pedirá una confirmación con un dálogo, ya que cerrar sesión en medio de una mano supone perder el dinero apostado.

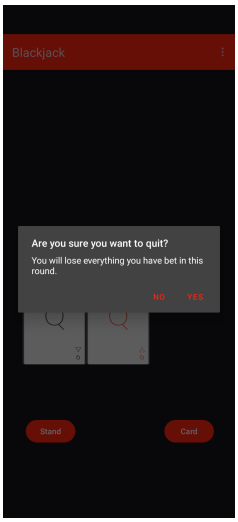

Figura 7: Diálogo de confirmación para salir del juego y perder el dinero apostado.

4. En caso de apostar, el juego dará comienzo. Se nos mostrarán dos nuevos botones, se nos repartirán dos cartas y se nos mostrará nuestra puntuación actual. Nuestro objetivo será quedarnos más cerca de 21 puntos que el dealer sin pasarnos. Para ello usaremos los botones de carta (que agregará una carta a nuestra mano) y plantarse (que cederá el turno al dealer). En caso de que nos aparezca un as en la mano, **deberemos elegir que puntuación queremos jugar antes de plantarnos**.

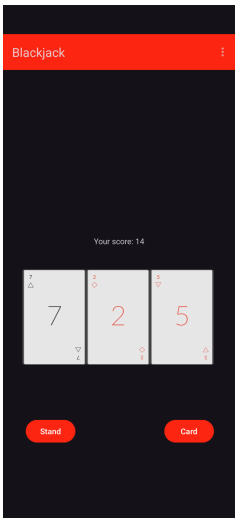

Figura 8: La puntuación y número de cartas aumentan a medida que vamos pidiendo.

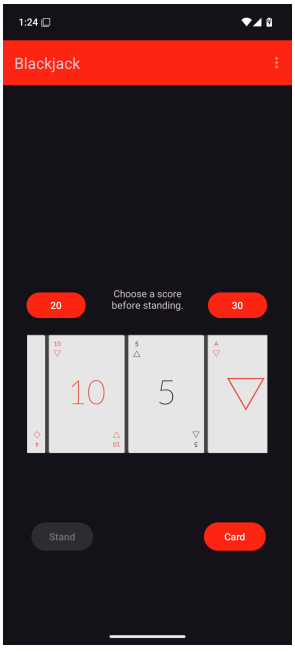

Figura 9: Se pide al jugador elegir una puntuación. Esto puede pasar varias veces en una sola mano. Los detalles técnicos están descritos en el método comprobarAses(), en la sección de diseño.

5. Al pulsar el botón de plantarnos, comenzará el turno del dealer. Este irá cogiendo cartas hasta superar la puntuación del jugador o perder. En caso de igualar la puntuación, cogerá otra carta a menos que la puntuación sea 21, caso que generaría un empate y reembolsaría la apuesta del jugador. Al terminar el juego, el resultado se mostrará en un diálogo.

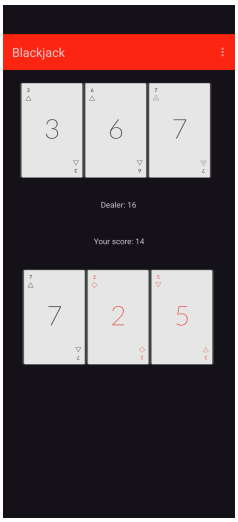

Figura 10: El dealer va sumando cartas hasta ganar al jugador o pasarse de 21.

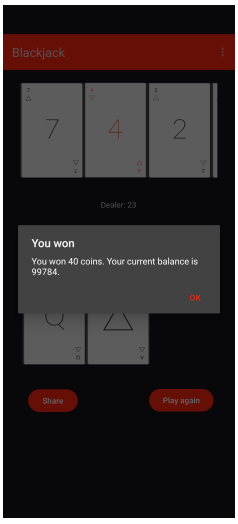

Figura 11: Diálogo de victoria.

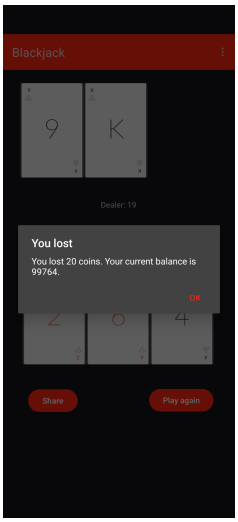

Figura 12: Diálogo de derrota.

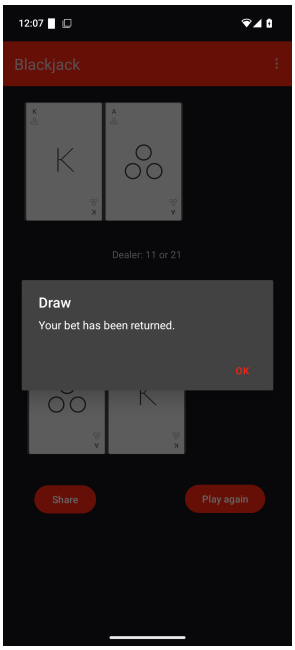

Figura 13: Diálogo de empate. Para provocarlo rápidamente, es recomendable descomentar las líneas de testing en PlayActivity.

6. Al retirar el diálogo de resultado, podemos ver dos botones, uno para com-

partir y otro para volver a jugar. El botón de volver a jugar nos devuelve a la pantalla para introducir una apuesta. El botón de compartir llama a un intent explícito para elegir con qué aplicación compartir nuestro resultado y nuestra buena experiencia con el juego.

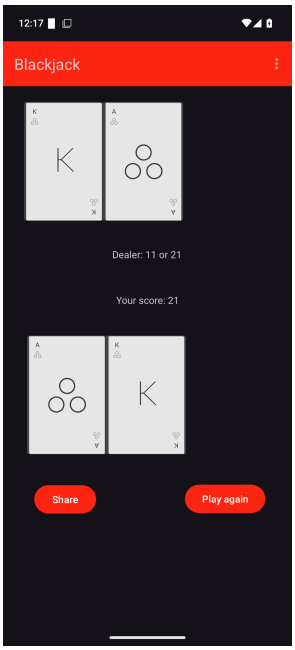

Figura 14: Pantalla final donde elegir si jugar de nuevo o compartir el resultado.

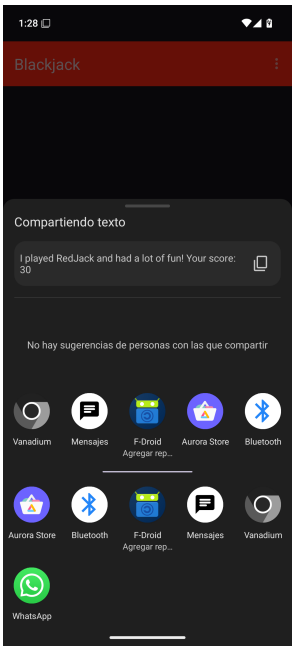

Figura 15: Desplegable que se ve para compartir.

6. Dificultades

A lo largo del proyecto, me he encontrado con algunas dificultades en lo que a desarrollo y diseño se refiere. Estas serán listadas en esta sección.

- La construcción de la base de datos. A pesar de ser una única tabla con escasas columnas, el hecho de ser persistente dificulta los cambios una vez ya se ha cargado. Soluciones como el borrado de caché fueron de gran ayuda en este punto.
- El fragment de preferencias. El hecho de querer desplegarlo sobre la actividad de juego (en vez de generar un nueva actividad como con el historial) dificultó su implementación, dado que el fragment interfería con la interacción con el input de apuesta y su respectivo botón. Con un poco de paciencia e integrando bien el fragmentContainer con la actividad, logré un equilibrio entre complejidad y accesibilidad.
- Lógica de juego. A pesar de no ser fiel a las reglas clásicas del Blackjack (ya que es una variante), la lógica detrás del valor de los ases requirió un tiempo de trabajo considerable. Con un buen plan de pruebas y recorriendo todas las casuísticas posibles, aseguré e hice funcionar todas las posibilidades en el mar de situaciones que cualquier juego de naipes tendría.

- Manejo de evento de retroceso. Se intentó capturar el evento de retroceso de Android para evitar cerrar sesión durante una mano, pero se desestimó por motivos de API.

7. Fuentes

- **Android Studio Documentation**: https://developer.android.com/develop
- **Theme Documentation Android Studio**: https://developer.android.com/develop/ui/views/theming/themes
- **Apuntes de eGela**: https://egela.ehu.eus
- **Overleaf Documentation**: https://www.overleaf.com/learn
- **ChatGPT**: chatgpt.com
- **Enable/disable menuItems**: https://stackoverflow.com/questions/5440601/android-how-to-enable-disable
- **How to add/change theme**: https://stackoverflow.com/questions/16579448/how-to-change-or-add-theme-to-android-studio
- **Where is gradle.properties**: https://stackoverflow.com/questions/37380417/android-studio-where-is-gradle-properties-file
- **Menu Documentation Android Studio**: https://developer.android.com/develop/ui/views/components/menus
- **How to sleep Android Studio**: https://stackoverflow.com/questions/1520887/how-to-pause-sleep-thread-or-process-in-android
- **How to string 'placeholders'**: https://stackoverflow.com/questions/3656371/is-it-possible-to-have-placeholders-in-strings-xml-for-runtime-values
- **Enable/disable buttons**: https://www.geeksforgeeks.org/how-to-enable-disable-button-in-android/
- **Access drawable resources**: https://stackoverflow.com/questions/16369814/how-to-access-the-drawable-resources-by-name-in-android
- **String parameters in runtime (placeholders)**: https://developer.android.com/guide/topics/resources/runtime-changes
- **Horizontal RecycledViews**: https://www.geeksforgeeks.org/android-horizontal-recyclerview-with-examples/