



BUSCAMINAS

Alfonso Rincón Cuerva
2º DAM

Programación Multimedia y de Dispositivos Móviles

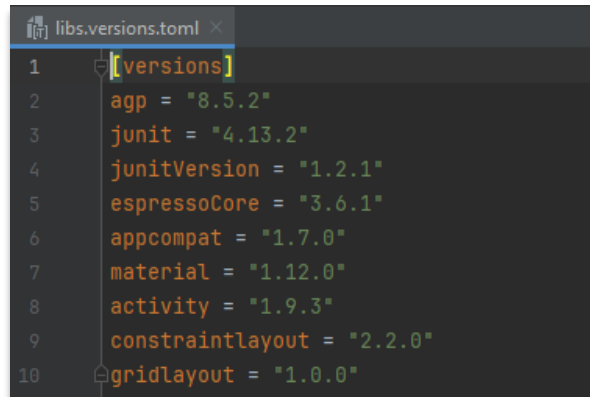
ÍNDICE

HERRAMIENTAS Y TECNOLOGÍAS	1
PROBLEMA PLANTEADO	2
SOLUCIÓN AL PROBLEMA	3
FUNCIONES UTILIZADAS	3
PRUEBAS REALIZADAS	10
ESTRUCTURA DE LA APLICACIÓN.....	10
EJECUCIÓN DE LA APLICACIÓN	11
COSAS QUE HE APRENDIDO	12
BIBLIOGRAFÍA.....	13

HERRAMIENTAS Y TECNOLOGÍAS VERSIONES

En el archivo **libs.versions.toml** encontramos versiones de diferentes bibliotecas y plugins del proyecto.

- **agp**: 8.5.2.
- **gridlayout**: 1.0.0.
- **activity**: 1.9.3.



```
1  [versions]
2      agp = "8.5.2"
3      junit = "4.13.2"
4      junitVersion = "1.2.1"
5      espressoCore = "3.6.1"
6      appcompat = "1.7.0"
7      material = "1.12.0"
8      activity = "1.9.3"
9      constraintlayout = "2.2.0"
10     gridlayout = "1.0.0"
```

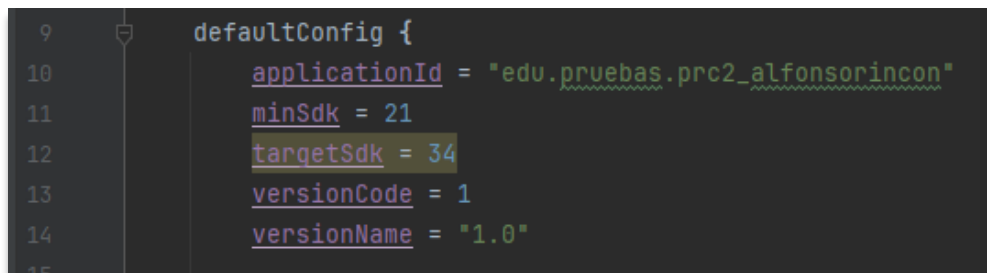
La versión de Gradle es de 8.7.



```
1  #Sun Nov 10 11:33:01 CET 2024
2  distributionBase=GRADLE_USER_HOME
3  distributionPath=wrapper/dists
4  distributionUrl=https\://services.gradle.org/distributions/gradle-8.7-bin.zip
5  zipStoreBase=GRADLE_USER_HOME
6  zipStorePath=wrapper/dists
```

En el archivo **build.gradle.kts**, podremos ver lo siguiente:

- **minSdk**: en esta aplicación es de 21. Esto significa que la versión de API mínima que soporta la app es la 21.
- **targetSdk**: es la versión de SDK para la cual está optimizada la app. En esta app, es de 34.



```
9  defaultConfig {
10      applicationId = "edu.pruebas.prc2_alfonsorincon"
11      minSdk = 21
12      targetSdk = 34
13      versionCode = 1
14      versionName = "1.0"
15  }
```

EMULADORES

- **Pixel API 30**: 5,0" 1080x1920 420dpi. **System image**: R (API 30)
- **Pixel 5 API 30**: 6,0" 1080x2340 440dpi. **System image**: R (API 30)

PLATAFORMA

El programa fue creado en la plataforma **Android Studio**, que es un entorno de desarrollo integrado oficial para crear aplicaciones Android. Incluye características como

editor de código, diseñador de interfaces, emulador de Android y sistema de compilación Gradle.

LENGUAJES

Los lenguajes utilizados para la creación de esta aplicación son los siguientes:

- **Java:** para la lógica de la aplicación e interactuar con los usuarios.
- **XML:** para agregar características y estilos a la aplicación.
- **Gradle:** es el sistema de compilación que utiliza Android.

PROBLEMA PLANTEADO

Se nos pide crear el clásico juego del Buscaminas. En este juego, tenemos un tablero con numerosas casillas, las cuales tienen la posibilidad de contener una mina. Las minas adyacentes a las casillas que contienen mina, contendrán un número, el cual nos indicará el número de minas con las cuales hacen frontera. La función del jugador será destapar aquellas minas que no contienen minas y señalar aquellas que sí contienen, con el fin de localizarlas todas.

A lo largo del juego, el jugador pondrá encontrarse frente a 4 casos diferentes:

CLICK: en el momento en el que el jugador haga Click en una casilla, podrán darse 2 casos:

- **CASILLA SIN MINA:** si se pulsa en una casilla sin mina, se destapará.
- **CASILLA CON MINA:** si la casilla tiene mina, se destapará todo el tablero, y el juego se habrá terminado.

LONG CLICK: cuando el jugador mantiene el Click sobre una casilla, se lanza este evento. Sirve para marcar donde hay una mina. Tendrá 2 posibles casos:

- **CASILLA SIN MINA:** como el jugador ha marcado una casilla que no tiene mina, perderá inmediatamente.
- **CASILLA CON MINA:** al haber marcado una casilla con mina, se pondrá un banderín, y el jugador podrá seguir jugando.

Además de la función del juego, la aplicación también cuenta con una ToolBar en la cual tendremos la posibilidad de poder configurar el juego a medida, ver una explicación de cómo se juega, seleccionar el personaje con el que se sienta más identificado, y reiniciar.

SOLUCIÓN AL PROBLEMA

Lo primero que he hecho ha sido crear el ToolBar con sus diferentes opciones, para el cual tuve que crear un XML. Una vez terminado, procedo a crear la funcionalidad del juego con diferentes funciones:



- Poner minas en el Array.
- Poner números de cercanía a las minas.
- Crear tablero.
- Destapar ceros cuando se pulse en uno de ellos.
- Derrota.
- Victoria.

Además, se agregan numerosos estilos, para dar una visión más llamativa a la aplicación. Los diálogos contienen sus XML propios, para así poder darles un aspecto más personalizado.

FUNCIONES UTILIZADAS

ICONO: en el archivo AndroidManifest.xml, asignaré un nuevo icono para la aplicación. Este consiste en un diseño de una mina marina, simbolizando el juego del buscaminas.

```

9   android:icon="@mipmap/icono_round"
10 android:label="PRC2_AlfonsoRincon"
11  android:roundIcon="@mipmap/icono_round"

```

VARIABLES PARA COLORES: en la carpeta **values** encontramos el archivo **colors.xml**. En este creo variables para todos los colores que usaré para cambiar el color de fondo y de botón, en cada uno de los estados.

```

colors.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <color name="black">#FF000000</color>
4      <color name="white">#FFFFFFF</color>
5      <color name="primero">#2168db</color>
6      <color name="dos">#10ad15</color>
7      <color name="tres">#ad2010</color>
8      <color name="cuatro">#122578</color>
9      <color name="cinco">#8410b5</color>
10     <color name="fondo">#DDDDDD</color>
11     <color name="transparente">#00000000</color>
12 </resources>

```

TOOLBAR: se crea dentro de la carpeta **menú**. Es un XML que contiene todos los ítems que contendrá este Toolbar. El ítem **“itemPersonaje”** tendrá la propiedad **kshowAsAction**, gracias a la cual se podrá poner como una imagen al lado de los 3 puntos.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3        xmlns:app="http://schemas.android.com/apk/res-auto">
4
5      <item android:id="@+id/itemPersonaje"
6            android:title="Personajes"
7            android:icon="@drawable/submarina"
8            app:showAsAction="ifRoom" ></item>
9
10     <item android:id="@+id/itemInstrucciones"
11           android:title="Instrucciones" ></item>
12
13     <item android:id="@+id/itemReiniciar"
14           android:title="Reiniciar juego" ></item>
15
16     <item android:id="@+id/itemConfiguracion"
17           android:title="Configurar el juego" ></item>
18 </menu>
```

Para poder añadirlo a la Activity se le añade un Toolbar al XML de dicha Activity, y se insertan las siguientes líneas de código en el onCreate:

```
// Crear la Toolbar
Toolbar toolbar=findViewById(R.id.toolbar);
supportActionBar(toolbar);
```

ACTIVIDAD PRINCIPAL: esta actividad contiene el logo del juego, y un botón para poder entrar a la actividad que contiene el juego en sí. Además, contiene un método **reproducirAudio()**, el cual se utiliza para reproducir la música de inicio y el sonido de la pulsación del botón.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);

    btnPlay=findViewById(R.id.btnPlay);
    reproducirAudio(R.raw.cancion_menu);

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}

@Override
protected void onResume() {
    super.onResume();

    // Iniciar el juego al pulsar en el botón
    btnPlay.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent i=new Intent(getApplicationContext(), JuegoActivity.class);
            // Para la música de fondo, y reproduzco el sonido de play
            mp.stop();
            reproducirAudio(R.raw.play);
            startActivity(i);
        }
    });
}

public void reproducirAudio(int ruta) {
    mp =MediaPlayer.create(context, ruta);
    mp.setVolume(1.0f, 1.0f);
    mp.start();
}
```

SPINNER: para poder crearlo, se crean 2 clases: la clase **Items** y la clase **Adapter**. La clase **Items** es la que se utiliza para cada ítem que contiene el Spinner, y el **Adapter** será la que contendrá todos estos **Items**.

Adapter contiene numerosos métodos, los cuales nos permitirán poder crear dicho Spinner, como son su **Constructor**, **getView**, **getDropDownView** y **vistaCreada**.

Finalmente, se llama a este Spinner desde el código de la actividad **JuegoActivity**. Se hará desde la opción de personaje designada en el **ToolBar**. Esta, mediante un **Inflater**, llamará al layout del diálogo para desplegar el Spinner, agregará todos los ítems al un **ArrayList** de **Items**, y se lo insertará al **Adapter**.

```
// Crear el Spinner, en caso de que se haya seleccionado la opción de personajes en el ToolBar
if(id == R.id.itemPersonaje) {
    dialogInflater = inflater.inflate(R.layout.personaje, root: null);

    // Spinner de diálogo
    dialogSpinner = dialogInflater.findViewById(R.id.spinner);

    // Agregamos todos los elementos al Spinner
    ArrayList<Items> elementosSpinner = new ArrayList<>();
    elementosSpinner.add(new Items(texto: "Bomba clásica", R.drawable.clasica));
    elementosSpinner.add(new Items(texto: "TNT", R.drawable.tnt));
    elementosSpinner.add(new Items(texto: "Granada", R.drawable.granada));
    elementosSpinner.add(new Items(texto: "Mina marina", R.drawable.submarina));
    elementosSpinner.add(new Items(texto: "Bomba Mario Bros", R.drawable.mariobros));

    // Creación del adaptador y asignación al Spinner del diálogo
    Adapter adapter = new Adapter(context: this, elementosSpinner);
    dialogSpinner.setAdapter(adapter);
    dialogSpinner.setOnItemSelectedListener(this);
}
```

Mediante el método **onItemSelected**, se cambiará el personaje principal, según la elección del usuario. Se utiliza el método **invalidateOptionsMenu()** para cambiar dicho personaje en el **ToolBar**.

CAMBIAR DIFICULTAD: es otra de las opciones del **ToolBar**. Al pulsar en la opción del **itemPersonaje**, se despliega un diálogo que contiene 3 **RadioButtons**, con los cuales

```
// Condicionales para ver que dificultad se ha elegido, y así poder crear una nueva partida
if(rdPrincipiante != null && rdPrincipiante.isChecked()) {
    partida.comenzar(x: 8, y: 8, minas: 10, juego: JuegoActivity.this);
    anchura=8;
    altura=8;
    minas=10;
    contMina=minas;
    tvContador.setText(String.valueOf(contMina));
} else if(rdAmateur != null && rdAmateur.isChecked()) {
    partida.comenzar(x: 12, y: 12, minas: 30, juego: JuegoActivity.this);
    anchura=12;
    altura=12;
    minas=30;
    contMina=minas;
    tvContador.setText(String.valueOf(contMina));
} else if(rdAvanzado != null && rdAvanzado.isChecked()) {
    partida.comenzar(x: 16, y: 16, minas: 60, juego: JuegoActivity.this);
    anchura=16;
    altura=16;
    minas=60;
    contMina=minas;
    tvContador.setText(String.valueOf(contMina));
}
```

podremos definir la dificultad en la cual queremos jugar. Según la dificultad elegida, se llamará al método **comenzar()**, el cual está en la clase **partida**, insertando unos valores u otros. También cambia el contenido del **TextView** que muestra cuantas minas hay, y el contenido de otras variables que se utilizan para guardar el número de minas y el tamaño de las casillas.

COMENZAR: este es el método con el que se podrá comenzar el juego, según la dificultad que queramos. Este llamará a los métodos que veremos a continuación.

```
// Método para definir la dificultad del juego. Por defecto, se inicia en Fácil
public void comenzar(int x, int y, int minas, JuegoActivity juego) { 7 usages  ▲ alfonsoaco *
    int[][] array=tablero.establecerMinas(x,y,minas);
    juego.crearTablero(array, x);
}
```

ESTABLECER MINAS: este es el método que se utiliza para crear el array de números gracias al cual se creará el tablero más tarde. Lo primero que hace este método, es en un array que tendrás las dimensiones que se le pase a través de los parámetros, insertar minas de forma aleatoria, con el valor de -1.

```
// Método para situar las minas de forma aleatoria a lo largo del Grid, además de poner los respectivos números en cada casilla
public int[][] establecerMinas(int anchura, int altura, int numMinas) { 1 usage  ▲ alfonsoaco
    int[][] tablero=new int[anchura][altura];

    // Método para insertar minas en el tablero de forma aleatoria
    int contMinas=0;
    int random;
    do {
        for (int i=0; i<tablero.length; i++) {
            for (int e=0; e<tablero[i].length; e++) {
                random=(int) (Math.random()*15);
                // Verificar que en la posición en cuestión, no haya una mina ya. Si el random es
                // 5, se colocará una mina.
                if(random==5 && tablero[i][e]!=-1 && contMinas<numMinas) {
                    contMinas++;
                    tablero[i][e]=-1;
                }
            }
        }
    } while(contMinas < numMinas);
}
```

Una vez insertadas todas las minas, tiene otro **for**, gracias al cual verificará cuantas minas rodean a cada posición del array. Por cada mina que se detecte en una posición que haga frontera con la posición actual en el que se encuentra el bucle, se sumará 1. Este método verifica mina en todas las posiciones: arriba, abajo, izquierda, derecha...

CREAR GRID: dentro de este método, encontraremos tanto la función para generar un tablero, como los eventos que se produzcan en este.

Empieza obteniendo el id del Grid, instaurando su números de columnas y filas utilizando los parámetros, y definiendo el tamaño de los botones. Estos se van a agregar a un Array de **View** que ya está definido previamente, el cual será un Array que contendrá todos estos botones. Se utiliza un View, ya que vamos a almacenar elementos distintos (**Buttons** e **ImageButtons**), y por tanto, no podríamos almacenarlos todos en un mismo Array de Buttons o ImageButtons.

Tras esto, tenemos un for, según el cual, mediante unos condicionales, identificaremos si es un Button o un ImageButton. Definiremos el tamaño y la altura para el botón, le pondremos un estilo, y lo añadiremos al Array.


```
// Método para crear el grid mediante el array de números
public void crearTablero(int[][] tablero, int tamañoGrid) { 1 usage Alfonso
    android.widget.GridLayout gridLayout=findViewById(R.id.gridLayout);

    // Línea de código, para evitar que cada vez que cambio de dificultad, se superpongan los botones, ya que se sobrecarga
    // la aplicación, y acaba petando. Con esto, se borran todos los botones, y se crea desde cero el nuevo grid.
    gridLayout.removeAllViews();

    gridLayout.setRowCount(tamañoGrid);
    gridLayout.setColumnCount(tamañoGrid);

    // Definir el tamaño del botón
    // getDisplayMetrics(): contiene información sobre la pantalla del dispositivo
    // widthPixels: devuelve el ancho en píxeles
    int dimensionBoton=getResources().getDisplayMetrics().widthPixels / tamañoGrid;
    botones=new View[tamañoGrid][tamañoGrid];

    for (int i=0; i<tablero.length; i++) {
        for (int e=0; e<tablero[i].length; e++) {
            // Caso mina
            if(tablero[i][e] == -1) {
                ImageButton boton=new ImageButton(context, this);

                // Tamaños del botón
                boton.setLayoutParams(new android.widget.GridLayout.LayoutParams());
                boton.getLayoutParams().width = dimensionBoton;
                boton.getLayoutParams().height = dimensionBoton;

                // Agregar los estilos creados en el XML a los botones
                boton.setBackgroundResource(R.drawable.estilos_boton);
                // Línea de código para poder hacer que la imagen se ajuste al tamaño del botón
                boton.setScaleType(ImageView.ScaleType.CENTER_INSIDE);

                // Guardar el botón en el array de View
                botones[i][e] = boton;
            }
        }
    }
}
```

Tras haber creado los botones, se les pondrán sus respectivos Listeners de Click y LongClick, que tendrán las condiciones vistas anteriormente.

DESTAPAR CEROS: este es un método recursivo que sirve para que, en el caso de haber pulsado en una casilla que contenga un 0, destapar todas las casillas adyacentes que no sean mina.

Este método lo que hace es verificar que esté dentro de los límites, y después, verificar que tipo de casilla es. En el caso de casillas distintas a 0 y que no sean ImageButton, simplemente las destapa. Si sí que son 0, no solo las destapa, sino que verifica las minas a su alrededor de forma recursiva.

```
public void destaparCeros(int fila, int columna, int[][] tablero) { 2 usages Alfonso
    // Comprobar que estamos dentro de los límites
    if ((fila >= 0 && fila < tablero.length) && (columna >= 0 && columna < tablero[0].length)) {
        View boton = botones[fila][columna];

        // Comprobamos que el botón no esté deshabilitado, ya que en este caso no se debe destapar
        if (boton.isEnabled()) {
            // Se verifica que no sea mayor de 0. Si lo es, se muestra la casilla y se termina la función
            if (tablero[fila][columna] != 0) {
                if (boton instanceof Button) {
                    mostrarNumero((Button) boton, fila, columna, tablero);
                }
                boton.setEnabled(false);
            }

            // Como es cero, se seguirá ejecutando el código de forma recursiva
            else {
                if (boton instanceof Button) {
                    mostrarNumero((Button) boton, fila, columna, tablero);
                }
                boton.setEnabled(false);
            }

            // For para llegar a las celdas adyacentes
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    // Con esto, evitamos que haga nada con la celda actual. El continue nos da la orden de seguir en el bucle con el siguiente valor
                    if (i == 0 && j == 0) continue;

                    // Llamada recursiva para las celdas adyacentes
                    destaparCeros(fila + i, columna + j, tablero);
                }
            }
        }
    }
}
```

MOSTRAR NÚMERO: es el método que se utiliza para, una vez se haya hecho Click en una casilla sin mina, mostrar el número que contiene. Para ello, compara con el Array que se crea con los métodos anteriores.

Según el número que obtenga, pondrá unos colores u otros, los cuales se obtienen del XML *colors.xml*. Por último, se deshabilita el botón, para no poder volver a pulsar en él.

```
// Método para mostrar el número con su color, al destapar un botón sin mina
public void mostrarNumero(Button boton, int i, int e, int[][] tablero) { 5 usages
    // Cambiar el color y fondo de los botones
    boton.setBackgroundColor(fondoBoton);

    if(tablero[i][e] == 1) {
        boton.setText("1");
        boton.setTextColor(unos);
    } else if(tablero[i][e] == 2) {
        boton.setText("2");
        boton.setTextColor(dos);
    } else if(tablero[i][e] == 3) {
        boton.setText("3");
        boton.setTextColor(tres);
    } else if(tablero[i][e] == 4) {
        boton.setText("4");
        boton.setTextColor(cuatro);
    } else if(tablero[i][e] == 5) {
        boton.setText("5");
        boton.setTextColor(cinco);
    }

    // Deshabilitar el botón
    boton.setEnabled(false);
}
```

MÉTODO DE DERROTA: este método se ejecuta una vez el jugador pierde la partida. Mediante un bucle, muestra el contenido de todas las casillas. En caso de que sea Button, muestra su número, y en caso de ImageButton, muestra la mina con el personaje seleccionado. También reproduce un sonido de derrota.

```
public void hasPerdido(int[][] tablero) { 2 usages  alfonso
    for (int i = 0; i < tablero.length; i++) {
        for (int e = 0; e < tablero[i].length; e++) {
            View view = botones[i][e];

            // Mostrar todos los botones que sean minas
            if (tablero[i][e] == -1 && view instanceof ImageButton) {
                ImageButton mina = (ImageButton) view;
                mina.setImageResource(personajeSeleccionado);
                mina.setBackgroundColor(fondoBoton);
                mina.setEnabled(false);

                // Mostrar todos los botones que no sean minas
            } else if (view instanceof Button) {
                Button boton = (Button) view;
                mostrarNumero(boton, i, e, tablero);
            }
        }
    }

    //Reproducimos el sonido de derrota
    reproducirAudio(R.raw.lose);
}
```

Tras ello, lanza un diálogo que utiliza una layout del XML **derrota.xml**.

MÉTODO VICTORIA: es similar al método de derrota, con la diferencia de que el sonido que muestra es el de victoria, y el diálogo es el del XML **victoria.xml**.

HANDLER: en esta aplicación, el Handler es utilizado para mostrar los mensajes de Victoria y Derrota con un retraso de 2 segundos. Dentro de esta función, está todo el código destinado a la creación del diálogo.

```
// Tras 2 segundos, se mostrará un diálogo que permitirá reiniciar el juego
new Handler().postDelayed(new Runnable() { ① alonsaco
    @Override ② alonsaco
    public void run() {
        // Inflator para poder coger el id del diálogo
        LayoutInflater inflater=getLayoutInflater();
        View dialogInflater=inflater.inflate(R.layout.derrota, null);

        // Builder de la alerta
        AlertDialog.Builder builder=new AlertDialog.Builder( context: JuegoActivity.this);
        builder.setView(dialogInflater);

        AlertDialog alerta=builder.create();
        // Quitar el color de fondo que tendrá la alerta, según el sistema
        alerta.getWindow().setBackgroundDrawableResource(R.color.transparente);

        Button btnReiniciar=dialogInflater.findViewById(R.id.btnDerrotaReiniciar);
        btnReiniciar.setOnClickListener(new View.OnClickListener() { ③ alonsaco
            @Override ④ alonsaco
            public void onClick(View view) {
                partida.comenzar(anchura, altura, minas, juego: JuegoActivity.this);
                contMina=minas;
                tvContador.setText(String.valueOf(contMina));

                mp.stop();
                alerta.dismiss();
            }
        });

        alerta.show();
    }
}, delayMillis: 2000);
```

REPRODUCIR AUDIO: método que se utilizará para reproducir audios. Se le pasa la ruta como parámetro, la cual será un id (int) de la carpeta **raw**.

```
//Función para reproducir los audios
public void reproducirAudio(int ruta) { ⑤ usages ⑥ alonsaco
    mp=MediaPlayer.create( context: this, ruta);
    mp.start();
}
```

PRUEBAS REALIZADAS

Se realizan diversas pruebas de caja negra para verificar la funcionalidad del programa. Tras ello, se programan sus respectivas soluciones:

1. EVITAR SATURACIÓN AL CAMBIAR DE DIFICULTAD

Para evitar que al cambiar de dificultad varias veces se sature el programa debido a que se han creado varios tableros y se han superpuesto unos por encima de otros, se utiliza la función `.removeAllViews()`.

2. DISTINTAS RESOLUCIONES:

Se ha probado la aplicación en pantallas con diferentes resoluciones, además de en dispositivos móviles reales. En todos funciona correctamente.

3. JUGAR EN DISTINTOS MODOS:

Se ha jugado en las diferentes dificultades, llegando a las condiciones de derrota y victoria, para poder comprobar su correcto funcionamiento. La ejecución se cumplió de manera exitosa.

ESTRUCTURA DE LA APLICACIÓN

drawable: aquí están todos los XML creados para los estilos de la alerta y botones, además de las imágenes de las bombas y las banderas.

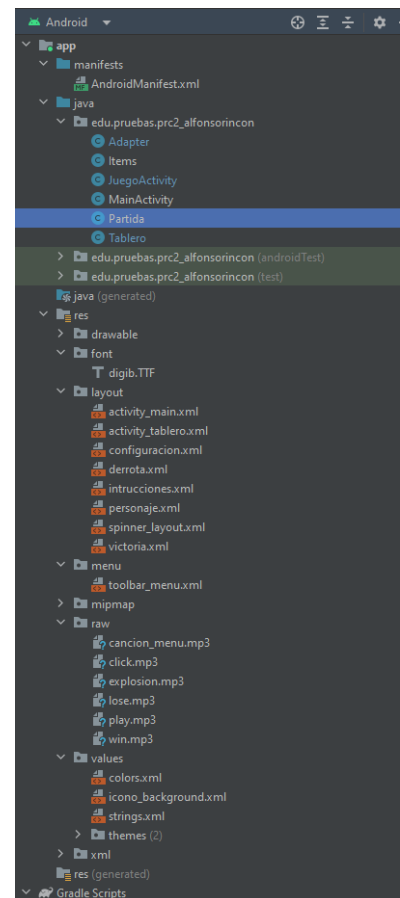
mipmap: dentro de esta carpeta están los archivos del icono de la aplicación.

raw: esta carpeta contiene los recursos de audio que utilizaremos en la aplicación. Cuando se ejecutan los métodos de reproducir audio, es a esta carpeta a la que accede para reproducirlos.

values: en sus archivos **colors.xml** y **string.xml** se encontrarán todas las variables de colores y strings que hemos creado.

menu: contiene el XML del ToolBar.

font: contiene la fuente que se utiliza en el TextView para mostrar el número de minas restantes.

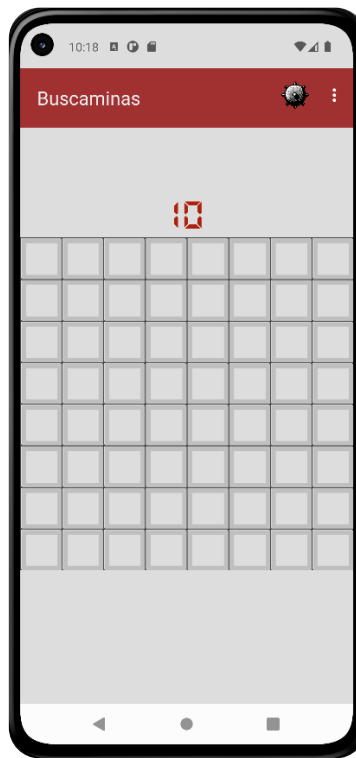


EJECUCIÓN DE LA APLICACIÓN

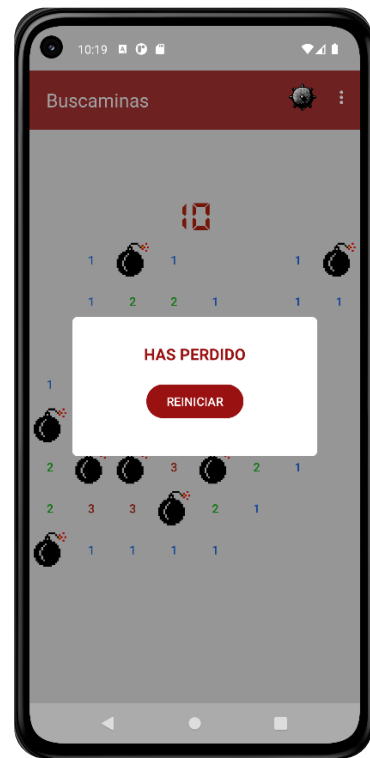
ESTADO INICIAL



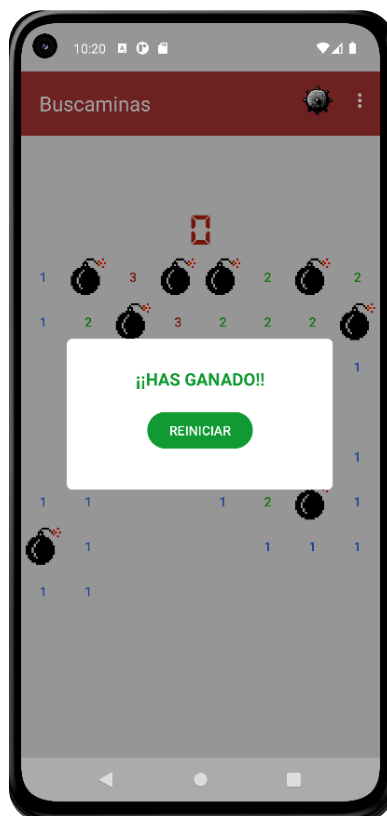
ENTRENAMIENTO



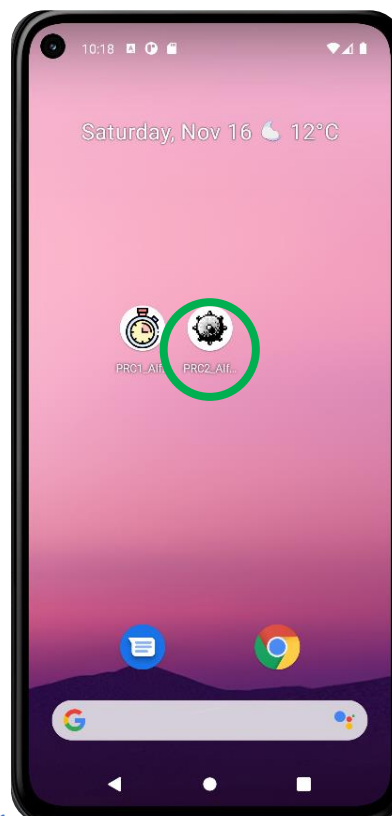
DESCANSO



ESTADO FINAL



ICONO



COSAS QUE HE APRENDIDO

1. Utilizar la función Handler.
2. Crear un ToolBar
3. Crear un Spinner con imágenes.
4. Agregar una fuente.
5. Crear diálogos personalizados.
6. El funcionamiento del juego Buscaminas.

BIBLIOGRAFÍA

AndroChunk. (2019, 16 febrero). *android custom spinner* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=U-jWiwie1o0>

Android Knowledge. (2023, 2 marzo). *Custom Toolbar or Action Bar in Android Studio using Java | Menu* [Vídeo]. YouTube.
<https://www.youtube.com/watch?v=zOsWCAsG2Xo>