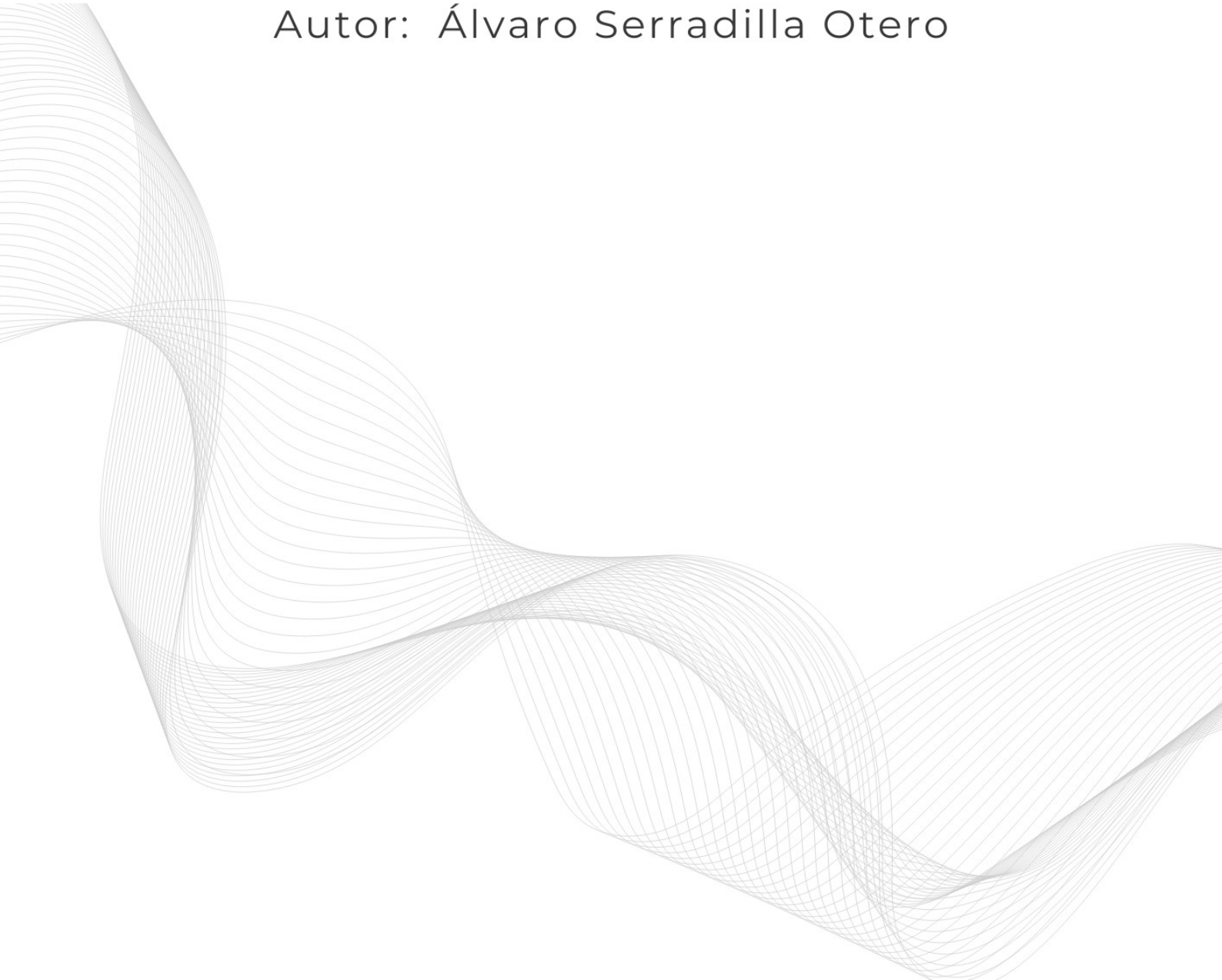
PROGRAMACIÓN MULTIMEDIA Y DE DISPOSITIVOS MÓVILE

DOCUMENTACIÓN BUSCAMINAS

Autor: Álvaro Serradilla Otero



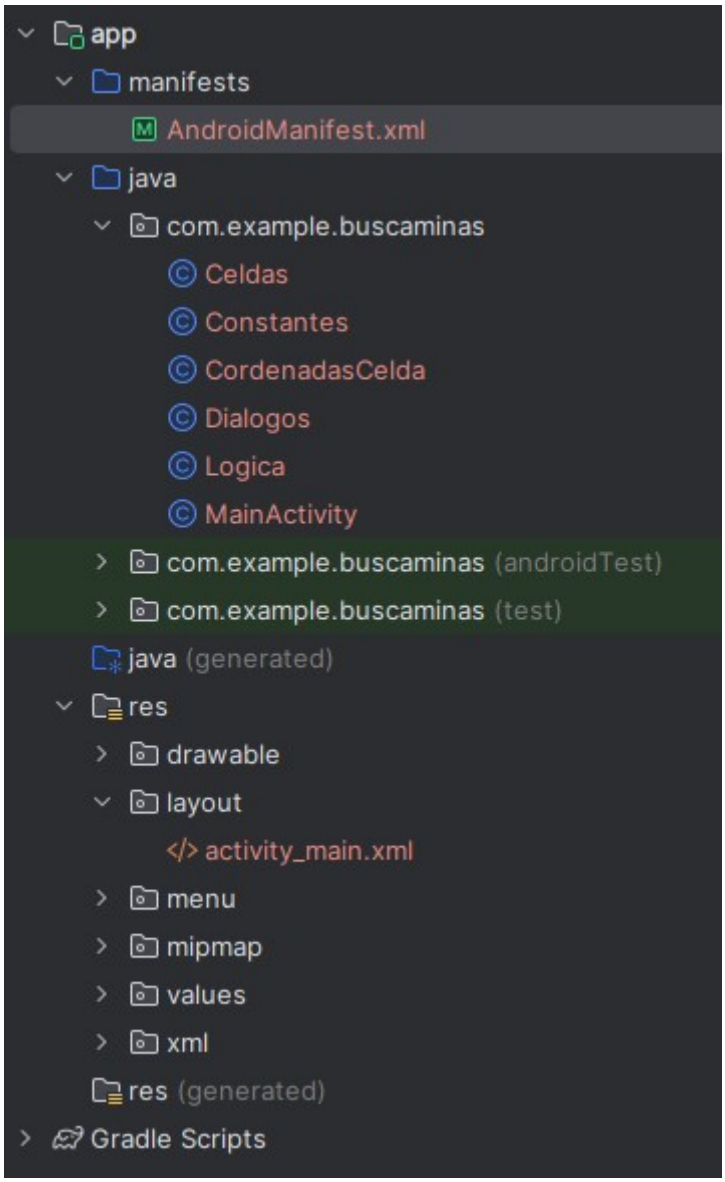
Índice

1. Recursos

Para realizar este programa hemos seleccionado el entorno de desarrollo de Android Studio en su versión llamada Meerkat, la versión de API seleccionada a sido lollipop ya que nos permitirá comercializar nuestra app para un publico mayor.

Los lenguajes de programación escogidos fueron java y xml, el lenguaje de java se utilizara para las clases de la aplicación que nos servirá para programar las funciones del programa y el lenguaje xml lo utilizaremos para establecer todo el apartado visual.

El árbol de carpetas de nuestra aplicación quedara tal que así:



2. Descarga

Para descargar el programa hemos utilizado la plataforma de GitHub donde se puede descargar la carpeta del programa y probarlo, el enlace para descargarlo es el siguiente:

https://github.com/AlvaroSerra24/DAM2.git

Aquí podrás encontrar esta practica junto con otras realizadas por mi.

3. Clases

En el caso de este ejercicio contamos con varias clases para organizar mejor nuestro programa, en dichas clases contamos con varios métodos para poder realizar que el programa cumpla con las funciones del juego buscaminas.

Para empezar hablare de las clases las cual nos sirven para almacenar datos como lo son la clase constantes que nos sirve para almacenar el tamaño según la dificultad del tablero y su numero de bombas.

```java
package com.example.buscaminas;

//Clase que contiene el tamaño de los tableros y el numero de minas que tienen.
public class Constantes {  12 usages
    //Constantes para el tamaño del tablero.
    public static final int FACIL = 8;  4 usages
    public static final int MEDIO = 12;  2 usages
    public static final int DIFICIL = 16;  2 usages
    //Constantes de el número de minas de cada tablero
    public static final int MINAS_FACIL = 10;  2 usages
    public static final int MINAS_MEDIO = 30;  1 usage
    public static final int MINAS_DIFICIL = 60;  1 usage
}
```

También contamos con una clase que nos sirve para almacenar las coordenadas de las celdas.

```java
package com.example.buscaminas;

//Clase que almacena las coordenadas de una celda en el tablero.
public class CordenadasCelda {  5 usages
    public int fila; //La fila de la celda.  2 usages
    public int columna; //La columna de la celda.  2 usages

    //Constructor para inicializar las coordenadas de la celda.
    public CordenadasCelda(int fila, int columna) {  1 usage
        this.fila = fila;
        this.columna = columna;
    }
}
```

A parte contamos con una clase la cual contiene todos los diálogos que se irán mostrando por el programa.

```java
//Muestra un diálogo con las instrucciones del juego.

public static void mostrarDialogoInstrucciones(Context context) {  1 usage
    new AlertDialog.Builder(context)
            .setTitle("Instrucciones")
            .setMessage("Cuando pulsas en una casilla, sale un número que identifica cuántas minas hay alrededor. "
                    + "Ten cuidado porque si pulsas en una casilla que tenga una mina escondida, perderás. "
                    + "Si crees o tienes la certeza de que hay una mina, haz un click largo sobre la casilla para señalarla. "
                    + "No hagas un click largo en una casilla donde no hay una mina porque perderás. "
                    + "Ganas una vez hayas encontrado todas las minas.")
            .setPositiveButton( text: "OK", ( DialogInterface dialog,  int which) -> dialog.dismiss())
            .setCancelable(true)
            .show();
}

//Muestra un diálogo de derrota cuando el jugador pulsa una mina.

public static void mostrarDialogoDerrota(Context context, Runnable reiniciarJuego) {  2 usages
    new AlertDialog.Builder(context)
            .setTitle("Has perdido")
            .setMessage("Has cometido un error. ¿Quieres volver a jugar?")
            .setPositiveButton( text: "Reiniciar", ( DialogInterface dialog,  int which) -> reiniciarJuego.run())
            .setCancelable(false)
            .show();
}

//Muestra un diálogo de victoria cuando el jugador ha revelado todas las celdas sin minas.

public static void mostrarDialogoVictoria(Context context, Runnable reiniciarJuego) {  1 usage
    new AlertDialog.Builder(context)
            .setTitle("¡Enhorabuena!")
            .setMessage("¡Has ganado! ¿Quieres reiniciar el juego?")
            .setPositiveButton( text: "Reiniciar", ( DialogInterface dialog,  int which) -> reiniciarJuego.run())
            .setCancelable(false)
            .show();
}
```

```java
//Interfaz para notificar cuando se ha seleccionado una dificultad.

public interface DificultadSeleccionadaListener {  1 usage
    void onDificultadSeleccionada(int dificultad);  1 usage
}
//Muestra un diálogo para que el usuario seleccione la dificultad del juego.
public static void mostrarDialogoCambioDificultad(Context context, DificultadSeleccionadaListener listener) {  1 usage
    RadioGroup opcionesDificultad = new RadioGroup(context);
    opcionesDificultad.setOrientation(RadioGroup.VERTICAL);

    RadioButton opcionFacil = new RadioButton(context);
    opcionFacil.setText("Fácil");
    opcionFacil.setTag(Constantes.FACIL);

    RadioButton opcionMedio = new RadioButton(context);
    opcionMedio.setText("Medio");
    opcionMedio.setTag(Constantes.MEDIO);

    RadioButton opcionDificil = new RadioButton(context);
    opcionDificil.setText("Dificil");
    opcionDificil.setTag(Constantes.DIFICIL);

    opcionesDificultad.addView(opcionFacil);
    opcionesDificultad.addView(opcionMedio);
    opcionesDificultad.addView(opcionDificil);

    new AlertDialog.Builder(context)
            .setTitle("Selecciona el tamaño del tablero")
            .setView(opcionesDificultad)
            .setPositiveButton( text: "Aceptar", ( DialogInterface dialog,  int which) -> {
                int opcionElegidaId = opcionesDificultad.getCheckedRadioButtonId();
                if (opcionElegidaId != -1) {
                    RadioButton seleccionarOpcion = opcionesDificultad.findViewById(opcionElegidaId);
                    int dificultad = (int) seleccionarOpcion.getTag();
                    listener.onDificultadSeleccionada(dificultad);
                }
            })
            .setNegativeButton( text: "Cancelar",  listener: null)
            .show();
}
```

Luego contamos con una clase de celdas para representar cada una de las celdas del programa, donde podemos manejar los dos tipos de click que se le pueden dar a una celda.

```java
// Clase que representa una celda individual en el tablero de Buscaminas.

public class Celdas {  2 usages
    private static final String TAG = "Cell";  2 usages

    private TextView textView;  15 usages
    private int fila;  10 usages
    private int columna;  10 usages
    private OnCellRevealedListener listener;  5 usages
    private boolean tieneBandera = false; // Indica si la celda tiene una bandera  4 usages

    // Interfaz para notificar cuando una celda ha sido revelada.

    public interface OnCellRevealedListener {  2 usages
        void onCellRevealed();  2 usages
    }

    // Constructor que inicializa la celda con sus coordenadas y tamaño.

    public Celdas(Context context, int fila, int columna, int ancho, int alto, int margin) {  1 usage
        this.fila = fila;
        this.columna = columna;
        this.textView = new TextView(context);
        configurarTextView(ancho, alto, margin);
        configurarEventos(context);
    }

    // Configura las propiedades visuales del TextView que representa la celda.

    private void configurarTextView(int ancho, int alto, int margin) {  1 usage
        GridLayout.LayoutParams params = new GridLayout.LayoutParams();
        params.setMargins(margin, margin, margin, margin);
        params.width = ancho;
        params.height = alto;
        params.rowSpec = GridLayout.spec(fila);
        params.columnSpec = GridLayout.spec(columna);
        textView.setLayoutParams(params);
        textView.setBackgroundColor(Color.LTGRAY);
        textView.setGravity(Gravity.CENTER);
        textView.setTextColor(Color.BLACK);
    }
```

En el caso de un simple click se manejara de la siguiente forma.

```java
// Maneja el evento de clic normal en la celda.

private void manejarClick(Context context) {  1 usage
    Log.d(TAG,  msg: "Click en celda: (" + fila + ", " + columna + ")");
    MainActivity activity = (MainActivity) context;
    Logica logica = activity.getMinefield();

    if (tieneBandera) {
        Toast.makeText(context,  text: "Esta celda tiene una bandera.", Toast.LENGTH_SHORT).show();
        return; // No permitir revelar una celda con bandera
    }

    if (logica.hayMina(fila, columna)) {
        textView.setText("💣");
        textView.setBackgroundColor(Color.RED);
        Dialogos.mostrarDialogoDerrota(activity, activity::iniciarJuego);
    } else {
        List<CordenadasCelda> celdasReveladas = new ArrayList<>();
        logica.revelarCelda(fila, columna, celdasReveladas);
        activity.actualizarCeldas(celdasReveladas);
        if (listener != null) {
            listener.onCellRevealed();
        }
    }
}
```

Y en caso de un click largo de la siguiente forma.

```
// Maneja el evento de clic largo en la celda para marcar o desmarcar una mina.

private void manejarClickLargo(Context context) {  1 usage
    Log.d(TAG,  msg: "Click largo en celda: (" + fila + ", " + columna + ")");
    MainActivity activity = (MainActivity) context;
    Logica logica = activity.getMinefield();

    if (logica.esRevelado(fila, columna)) {
        Toast.makeText(context,  text: "No se puede marcar una celda revelada.", Toast.LENGTH_SHORT).show();
        return; // No permitir colocar una bandera en una celda revelada
    }

    if (tieneBandera) {
        // Quitar la bandera
        textView.setText("");
        textView.setBackgroundColor(Color.LTGRAY);
        tieneBandera = false;
        logica.colocarBandera(fila, columna); // Actualizar la lógica para quitar la bandera
    } else {
        // Colocar la bandera
        textView.setText("🚩");
        textView.setBackgroundColor(Color.WHITE);
        tieneBandera = true;
        logica.colocarBandera(fila, columna); // Actualizar la lógica para colocar la bandera

        if (!logica.hayMina(fila, columna)) {
            // Si la bandera está incorrectamente colocada, mostrar derrota
            textView.setBackgroundColor(Color.RED);
            Dialogos.mostrarDialogoDerrota(activity, activity::iniciarJuego);
        } else {
            Toast.makeText(context,  text: "Bandera correctamente colocada.", Toast.LENGTH_SHORT).show();
        }
    }

    if (listener != null) {
        listener.onCellRevealed();
    }
}
```

Para continuar hablaremos de la lógica del programa, la cual esta representada en la clase lógica donde vemos métodos para calcular el numero de minas dependiendo de la dificultad seleccionada o un método para generar las bombas dentro del tablero entre otros métodos.

```java
// Calcula el número total de minas basándose en el tamaño del tablero.

private int calcularTotalMinas(int filas) {  1 usage
    switch (filas) {
        case Constantes.FACIL:
            return Constantes.MINAS_FACIL;
        case Constantes.MEDIO:
            return Constantes.MINAS_MEDIO;
        case Constantes.DIFICIL:
            return Constantes.MINAS_DIFICIL;
        default:
            return Constantes.MINAS_FACIL;
    }
}

// Genera minas aleatoriamente en el tablero sin superponerlas.

private void generarMinas() {  1 usage
    minas = new boolean[filas][columnas];
    Random random = new Random();
    int colocarMinas = 0;

    while (colocarMinas < totalMinas) {
        int randomRow = random.nextInt(filas);
        int randomCol = random.nextInt(columnas);
        if (!minas[randomRow][randomCol]) {
            minas[randomRow][randomCol] = true;
            colocarMinas++;
        }
    }
}
```

Ademas de esto también contamos con métodos que sirven para la recursividad a la hora de revelar las celdas como son los métodos para revelar las celdas o contar las celdas adyacentes a la celda revelada y métodos para colocar las banderas en las casillas.

```java
// Comprueba si una celda específica ha sido revelada.

public boolean esRevelado(int fila, int columna) { return revelar[fila][columna]; }

// Coloca o quita una bandera en una celda específica.

public boolean colocarBandera(int fila, int columna) {  2 usages
    if (revelar[fila][columna]) {
        return false; // No se puede colocar una bandera en una celda ya revelada
    }
    return banderas[fila][columna];
}

// Revela una celda específica y, si no hay minas cercanas, revela de forma recursiva las celdas adyacentes.

public void revelarCelda(int fila, int columna, List<CordenadasCelda> celdasReveladas) {  2 usages
    if (fila < 0 || fila >= filas || columna < 0 || columna >= columnas || revelar[fila][columna] || banderas[fila][columna]) {
        return;
    }

    revelar[fila][columna] = true;
    celdasReveladas.add(new CordenadasCelda(fila, columna));
    if (!minas[fila][columna]) {
        if (contarBombasCerca(fila, columna) == 0) {
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    revelarCelda( fila: fila + i,  columna: columna + j, celdasReveladas);
                }
            }
        }
    }
}
```

Y por ultimo en esta clase también contamos contamos con un método que nos sirve para poner las condiciones de victoria en el juego las cuales son que todas las casillas con bombas deben tener la bandera puesta.

```java
// Comprueba si el jugador ha ganado

public boolean verificarVictoria() {  1 usage
    for (int fila = 0; fila < filas; fila++) {
        for (int columna = 0; columna < columnas; columna++) {
            if (minas[fila][columna] && !banderas[fila][columna]) {
                return false;
            }
            if (banderas[fila][columna] && !minas[fila][columna]) {
                return false;
            }
        }
    }
    return true;
}
```

Por ultimo contamos con las clase MainActivity la cual nos sirve para generar el layout del programa y colocarlo en el grid layout, ademas de iniciar el juego y ir actualizándolo para que nuestras acciones se vean reflejadas en la ventana.

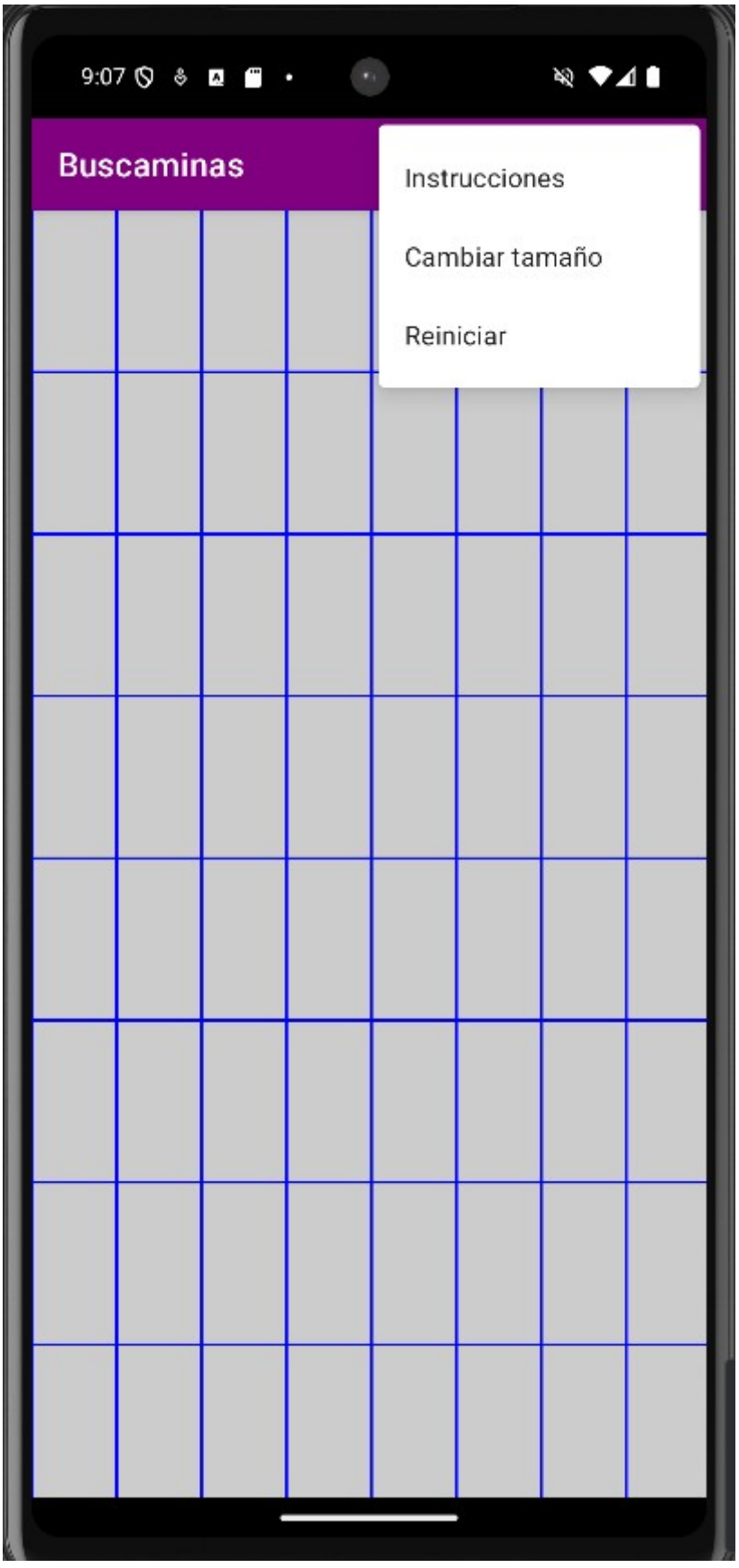
```java
// Infla el menú de opciones en la barra de acciones.

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}

// Maneja las selecciones de ítems en el menú de opciones.

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int opcion = item.getItemId();
    if (opcion == R.id.action_change_size) {
        Dialogos.mostrarDialogoCambioDificultad( context: this,  int dificultad -> {
            filas = dificultad;
            columnas = dificultad;
            iniciarJuego();
        });
        return true;
    } else if (opcion == R.id.action_restart) {
        iniciarJuego();
        return true;
    } else if (opcion == R.id.action_instructions) {
        Dialogos.mostrarDialogoInstrucciones( context: this);
        return true;
    } else {
        return super.onOptionsItemSelected(item);
    }
}

// Inicia o reinicia el juego configurando el campo de minas y el tablero en la UI.

public void iniciarJuego() {  6 usages
    logica = new Logica(filas, columnas);
    configurarGridLayout();
}
```

```java
// Obtiene la instancia actual de Minefield.

public Logica getMinefield() { return logica; }

// Configura el GridLayout creando y añadiendo las celdas al tablero.

public void configurarGridLayout() {  1 usage
    gridLayout.removeAllViews();
    gridLayout.setRowCount(filas);
    gridLayout.setColumnCount(columnas);
    gridLayout.setBackgroundColor(Color.BLUE);

    Log.d( tag: "MainActivity",  msg: "Configurando GridLayout con filas: " + filas + ", columnas: " + columnas);

    gridLayout.post(() -> {
        int anchoGrid = gridLayout.getWidth();
        int altoGrid = gridLayout.getHeight();
        int margin = 2;
        int anchoMargenTotal = margin * (columnas + 1);
        int altoMargenTotal = margin * (filas + 1);
        int anchoCelda = (anchoGrid - anchoMargenTotal) / columnas;
        int altoCelda = (altoGrid - altoMargenTotal) / filas;

        Log.d( tag: "MainActivity",  msg: "Ancho de celda: " + anchoCelda + ", Alto de celda: " + altoCelda);

        for (int fila = 0; fila < filas; fila++) {
            for (int columna = 0; columna < columnas; columna++) {...}
        }
    });
}

// Actualiza la apariencia de una lista de celdas reveladas en la UI.

public void actualizarCeldas(List<CordenadasCelda> celdasReveladas) {  1 usage
    for (CordenadasCelda coord : celdasReveladas) {
        actualizarCelda(coord.fila, coord.columna);
    }
}
```

```java
// Actualiza la apariencia visual de una celda especifica en la UI basada en su estado en Minefield.

public void actualizarCelda(int fila, int columna) {  1 usage
    int index = fila * columnas + columna;
    if (index >= 0 && index < gridLayout.getChildCount()) {
        View view = gridLayout.getChildAt(index);
        if (view instanceof TextView) {
            TextView textView = (TextView) view;
            boolean esRevelado = logica.esRevelado(fila, columna);
            boolean hayMina = logica.hayMina(fila, columna);
            boolean hayBandera = logica.hayBandera(fila, columna);

            if (esRevelado) {
                textView.setBackgroundColor(Color.WHITE);
                if (hayMina) {
                    textView.setText("💣");
                } else {
                    int bombasCerca = logica.contarBombasCerca(fila, columna);
                    textView.setText(bombasCerca > 0 ? String.valueOf(bombasCerca) : "");
                }
                textView.setEnabled(false);
            } else if (hayBandera) {
                textView.setText("🚩");
                textView.setBackgroundColor(Color.WHITE);
            } else {
                textView.setText("");
                textView.setBackgroundColor(Color.LTGRAY);
            }
        }
    }
}

// Verifica si el jugador ha ganado y si es así se muestra el mensaje de victoria.

private void verificarVictoria() {  1 usage
    if (logica.verificarVictoria()) {
        Dialogos.mostrarDialogoVictoria( context: this, this::iniciarJuego);
    }
}
```

4. Interfaz

Este programa cuenta con un layout en la parte superior en el cual pone el nombre del juego en este caso Buscaminas ademas cuenta con un menú de opciones desarrollado para poder cambiar la dificultad del juego, reiniciar la partida y ver las reglas del juego, ademas de esto cuenta con el tablero de juego en el resto de la pantalla.



5. Funcionalidad

El programa realiza las funciones clásicas del buscaminas en la cual tenemos que ir revelando las celdas sin hacer click en una bomba ademas podemos ir poniendo banderas en las bombas para marcar la casilla, las condiciones de juegos son las de marcar todas las casilla con bomba con una bandera una vez hecho eso el juego nos mostrara un mensaje de victoria, en el caso de hagamos click en una bomba o marquemos una casilla vaciá con una bandera nos mostrara un mensaje de derrota.