Introducción

Buscaminas es un juego individual. El objetivo del juego es limpiar un campo de minas sin chocar con una mina. Buscaminas no sólo está disponible para Windows, sino también para otras plataformas (incluyendo la mayoría de las variantes de Linux). Buscaminas es muy popular en el mundo de Windows, se ha incluido en Windows desde Windows 3.1.

En este artículo, vamos a crear un clon dragaminas para Android. Vamos a tratar de poner en práctica la mayor parte de las características que están disponibles en el Buscaminas de Windows. Este artículo está dirigido a los desarrolladores de nivel intermedio-avanzado y espera que la familiaridad con Java y el desarrollo de Android.

Sobre el juego

En Buscaminas, se nos presenta con una rejilla de bloques y algunos de ellos contienen al azar minas. En nuestra aplicación, nos limitaremos a la aplicación típica de Nivel Inicial. El número de filas y columnas de nuestra aplicación será 9 y el número total de las minas será de 10. Ampliar el juego para el Intermedio, Avanzado, es fácil, sin embargo (al requerir un cambio en el valor de las 3 variables en el código).

Bueno, en este artículo, vamos a no hablar mucho acerca de cómo jugar el juego, pero vamos a hablar de algunas de las características que hay que pensar antes de su aplicación (versión Windows):

1. Izquierda haga clic en un bloque, se abre el bloque.

2. Haga clic derecho en un bloque permite marcar un bloque marcado como (el mío confirmada por debajo); bloques marcados se pueden marcar con signos de interrogación (dudas sobre la presencia de minas), y pregunta bloques marcados pueden ser no-marcados también.

3. Primer bloque nunca contiene una mina debajo, lo que reduce el dolor de adivinar incluso el primer bloque.

4. Si un bloque descubierto está en blanco, los bloques cercanos se abren de forma recursiva hasta que se abra un bloque de números, simulando un efecto dominó.

5. Al hacer clic en el botón left-right/middle en un bloque, donde todas las minas en los bloques cercanos ya están marcados, descubre todos los bloques cubiertos cercanas.

6. Timer inicia al hacer clic en el primer bloque y no después de la selección de un nuevo juego.

Basta de hablar ahora, vamos a empezar a trabajar en él.

Un paso a la vez

La mejor manera de explicar o aplicar cualquier sistema complejo es tomar un paso a la vez. Seguiremos este enfoque aquí. En primer lugar, vamos a hablar de la interfaz gráfica de usuario, el diseño y look-and-feel de la aplicación. También vamos a hablar de algunas de las técnicas utilizadas al crear el diseño. Después de eso vamos a hablar acerca de cómo mantener la noción del tiempo en una aplicación Android. A continuación vamos a discutir las diferencias entre el ratón, hacer clic, y eventos de toque y cómo vamos a responder a las acciones del usuario. El último paso será acerca de cómo implementar el juego completo y la mayoría de las características descritas en la sección Sobre el juego.

La vista y la sensación

Vamos a hablar de algunos de los aspectos de diseño de una interfaz gráfica de usuario para un juego Buscaminas. Vamos a hablar sobre el diseño general de la aplicación y también algunas técnicas utilizadas en la creación del juego.

diseño de aplicaciones

Para Buscaminas, vamos a utilizar una TableLayout. Añadimos tres filas a la TableLayout:

1. La primera fila contiene tres columnas de temporizador, el nuevo botón de juego, y el contador de la mía. Por temporizador y el contador de la mina, se ha utilizado una TextView. Para el nuevo botón de juego, hemos utilizado un ImageButton.

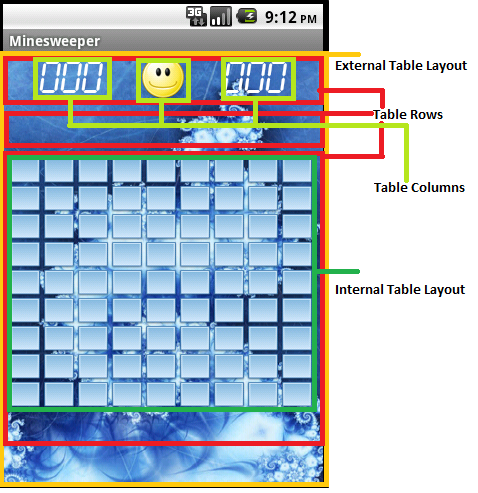
2. La segunda fila contiene un TextView vacío con una altura de 50 píxeles. Simplemente ayuda como separador entre la fila superior y el campo de minas.

3. Esta fila contiene otra TableLayout, que se utiliza para mostrar el campo de minas. Añadimos filas de botones a esta TableLayout dinámicamente.

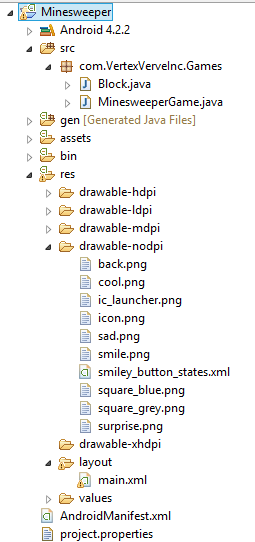
Capturas\_

|  |  |
| --- | --- |
|  |  |
| ***Toma- Inicio*** | ***Toma – Nuevo Juego*** |
|  |  |
| ***Captura- Ganaste!*** | ***Captura- Perdiste*** |

LAYOUTS USADOS:



Estructura del Proyecto:



|  |
| --- |
| Block.java |
| package com.VertexVerveInc.Games;  import android.content.Context;  import android.graphics.Color;  import android.graphics.Typeface;  import android.util.AttributeSet;  import android.widget.Button;  public class Block extends Button  {  private boolean isCovered; // todavía está cubierto de bloques  private boolean isMined; // qué el módulo tiene una mina debajo  private boolean isFlagged; // es un bloque marcado como una mina potencial  private boolean isQuestionMarked; // está marcado ? bloque  private boolean isClickable; // bloque puede aceptar eventos de clic  private int numberOfMinesInSurrounding; // número de minas en los bloques cercanos  public Block(Context context)  {  super(context);  }  public Block(Context context, AttributeSet attrs)  {  super(context, attrs);  }  public Block(Context context, AttributeSet attrs, int defStyle)  {  super(context, attrs, defStyle);  }  // establecer las propiedades predeterminadas para el bloque  public void setDefaults()  {  isCovered = true;  isMined = false;  isFlagged = false;  isQuestionMarked = false;  isClickable = true;  numberOfMinesInSurrounding = 0;  this.setBackgroundResource(R.drawable.square\_blue);  setBoldFont();  }  // Marcar el bloque como desabilidato/abierto  // Actualizar el número de minas cercanas  public void setNumberOfSurroundingMines(int number)  {  this.setBackgroundResource(R.drawable.square\_grey);    updateNumber(number);  }  // Configurar icono mina para el bloque  // Set de bloques como deshabilitar / abrir si se pasa false  public void setMineIcon(boolean enabled)  {  this.setText("M");  if (!enabled)  {  this.setBackgroundResource(R.drawable.square\_grey);  this.setTextColor(Color.RED);  }  else  {  this.setTextColor(Color.BLACK);  }  }  // Establecer la mía marcado  // establecer bloques como deshabilidatos/abrir si se pasa false  public void setFlagIcon(boolean enabled)  {  this.setText("F");  if (!enabled)  {  this.setBackgroundResource(R.drawable.square\_grey);  this.setTextColor(Color.RED);  }  else  {  this.setTextColor(Color.BLACK);  }  }  // Establecer mina como signo de interrogación  // Establecer de bloques como deshabilitar / abrir si se pasa false  public void setQuestionMarkIcon(boolean enabled)  {  this.setText("?");    if (!enabled)  {  this.setBackgroundResource(R.drawable.square\_grey);  this.setTextColor(Color.RED);  }  else  {  this.setTextColor(Color.BLACK);  }  }  // Establecer bloques como discapacitados / abrir si se pasa false  // Else activar / cerrarlo  public void setBlockAsDisabled(boolean enabled)  {  if (!enabled)  {  this.setBackgroundResource(R.drawable.square\_grey);  }  else  {  this.setBackgroundResource(R.drawable.square\_blue);  }  }  // limpiar todos los icons/text  public void clearAllIcons()  {  this.setText("");  }  // establecer la fuente en negrita  private void setBoldFont()  {  this.setTypeface(null, Typeface.BOLD);  }  // descubrir este bloque  public void OpenBlock()  {  // no puede descubrir una mina que no está cubierta  if (!isCovered)  return;  setBlockAsDisabled(false);  isCovered = false;  // comprobar si tiene la mina  if (hasMine())  {  setMineIcon(false);  }  // actualizar el recuento mina cercana  else  {  setNumberOfSurroundingMines(numberOfMinesInSurrounding);  }  }  // configurar el texto como en las inmediaciones recuento mina  public void updateNumber(int text)  {  if (text != 0)  {  this.setText(Integer.toString(text));  // Seleccionar un color diferente para cada número  // Ya hemos saltado 0 conteo mina  switch (text)  {  case 1:  this.setTextColor(Color.BLUE);  break;  case 2:  this.setTextColor(Color.rgb(0, 100, 0));  break;  case 3:  this.setTextColor(Color.RED);  break;  case 4:  this.setTextColor(Color.rgb(85, 26, 139));  break;  case 5:  this.setTextColor(Color.rgb(139, 28, 98));  break;  case 6:  this.setTextColor(Color.rgb(238, 173, 14));  break;  case 7:  this.setTextColor(Color.rgb(47, 79, 79));  break;  case 8:  this.setTextColor(Color.rgb(71, 71, 71));  break;  case 9:  this.setTextColor(Color.rgb(205, 205, 0));  break;  }  }  }  // sistema del bloque como una mina debajo  public void plantMine()  {  isMined = true;  }  // Mina fue abierta  // Cambiar el icono de bloqueo y color  public void triggerMine()  {  setMineIcon(true);  this.setTextColor(Color.RED);  }  // se bloque todavía cubierta  public boolean isCovered()  {  return isCovered;  }  // no el bloque tiene cualquier mina debajo  public boolean hasMine()  {  return isMined;  }  // establecer el número de minas cercanas  public void setNumberOfMinesInSurrounding(int number)  {  numberOfMinesInSurrounding = number;  }  // get number of nearby mines  public int getNumberOfMinesInSorrounding()  {  return numberOfMinesInSurrounding;  }  // se bloque marcado como marcado  public boolean isFlagged()  {  return isFlagged;  }  // marcar como bloque marcado  public void setFlagged(boolean flagged)  {  isFlagged = flagged;  }  // se bloque marcado como un signo de interrogación  public boolean isQuestionMarked()  {  return isQuestionMarked;  }  // establecer signo de interrogación para el bloque  public void setQuestionMarked(boolean questionMarked)  {  isQuestionMarked = questionMarked;  }  // bloque puede recibir evento de clic  public boolean isClickable()  {  return isClickable;  }  // desactivar el bloqueo para recibir eventos de clic  public void setClickable(boolean clickable)  {  isClickable = clickable;  }  } |

|  |
| --- |
| MinesweeperGame.java |
| package com.VertexVerveInc.Games;  import java.util.Random;  import android.app.Activity;  import android.graphics.Typeface;  import android.os.Bundle;  import android.os.Handler;  import android.view.Gravity;  import android.view.View;  import android.view.View.OnClickListener;  import android.view.View.OnLongClickListener;  import android.widget.ImageButton;  import android.widget.ImageView;  import android.widget.LinearLayout;  import android.widget.TableRow.LayoutParams;  import android.widget.TableLayout;  import android.widget.TableRow;  import android.widget.TextView;  import android.widget.Toast;  public class MinesweeperGame extends Activity  {  private TextView txtMineCount;  private TextView txtTimer;  private ImageButton btnSmile;  private TableLayout mineField; // diseño de la mesa para añadir minas  private Block blocks[][]; //bloques de campo de minas  private int blockDimension = 24; //anchura de cada bloque  private int blockPadding = 2; // relleno entre bloques  private int numberOfRowsInMineField = 9;  private int numberOfColumnsInMineField = 9;  private int totalNumberOfMines = 10;  // cronómetro para controlar el tiempo transcurrido  private Handler timer = new Handler();  private int secondsPassed = 0;  private boolean isTimerStarted; // comprobar si el temporizador ya iniciada o no  private boolean areMinesSet; //comprobar si las minas se siembran en bloques  private boolean isGameOver;  private int minesToFind; // número de minas aún por descubrir  @Override  public void onCreate(Bundle savedInstanceState)  {  super.onCreate(savedInstanceState);  setContentView(R.layout.main);    txtMineCount = (TextView) findViewById(R.id.MineCount);  txtTimer = (TextView) findViewById(R.id.Timer);    // programar el estilo de fuente para el recuento del temporizador y el mío al estilo de LCD  Typeface lcdFont = Typeface.createFromAsset(getAssets(),  "fonts/lcd2mono.ttf");  txtMineCount.setTypeface(lcdFont);  txtTimer.setTypeface(lcdFont);    btnSmile = (ImageButton) findViewById(R.id.Smiley);  btnSmile.setOnClickListener(new OnClickListener()  {  @Override  public void onClick(View view)  {  endExistingGame();  startNewGame();  }  });    mineField = (TableLayout)findViewById(R.id.MineField);    showDialog("Click en emoticon para jugar", 2000, true, false);  }  private void startNewGame()  {  // minas y restos vegetales do de los cálculos  createMineField();  // mostrar todos los bloques de la interfaz de usuario  showMineField();    minesToFind = totalNumberOfMines;  isGameOver = false;  secondsPassed = 0;  }  private void showMineField()  {  // Recordemos que no mostraremos 0 ª y última filas y columnas  // que se utilizan para fines de cálculo sólo  for (int row = 1; row < numberOfRowsInMineField + 1; row++)  {  TableRow tableRow = new TableRow(this);  tableRow.setLayoutParams(new LayoutParams((blockDimension + 2 \* blockPadding) \* numberOfColumnsInMineField, blockDimension + 2 \* blockPadding));  for (int column = 1; column < numberOfColumnsInMineField + 1; column++)  {  blocks[row][column].setLayoutParams(new LayoutParams(  blockDimension + 2 \* blockPadding,  blockDimension + 2 \* blockPadding));  blocks[row][column].setPadding(blockPadding, blockPadding, blockPadding, blockPadding);  tableRow.addView(blocks[row][column]);  }  mineField.addView(tableRow,new TableLayout.LayoutParams(  (blockDimension + 2 \* blockPadding) \* numberOfColumnsInMineField, blockDimension + 2 \* blockPadding));  }  }  private void endExistingGame()  {  stopTimer(); // detenerse si el temporizador está en marcha  txtTimer.setText("000"); // Reiniciar todos los textos  txtMineCount.setText("000"); // reiniciar cuenta minas  btnSmile.setBackgroundResource(R.drawable.smile);    // eliminar todas las filas del campo minado TableLayout  mineField.removeAllViews();    // configurar todas las variables para apoyar el final del partido  isTimerStarted = false;  areMinesSet = false;  isGameOver = false;  minesToFind = 0;  }  private void createMineField()  {  // Tomamos una fila fila adicional para cada lado  // En general dos filas adicionales y dos columnas adicionales  // Primera y la fila / columna última se utilizan sólo con fines cálculos  // La fila y columnas marcadas como x sólo son utilizadas para mantener los recuentos de cerca por las minas  blocks = new Block[numberOfRowsInMineField + 2][numberOfColumnsInMineField + 2];  for (int row = 0; row < numberOfRowsInMineField + 2; row++)  {  for (int column = 0; column < numberOfColumnsInMineField + 2; column++)  {  blocks[row][column] = new Block(this);  blocks[row][column].setDefaults();  // Pasar fila actual y el número de columna como int definitiva es que los detectores de eventos  // De esta manera podemos asegurar que cada detector de eventos se asocia a  // Particular, instancia de bloque sólo  final int currentRow = row;  final int currentColumn = column;  // Añadir Click Listener  // Esto se trata como si hiciera click izquierdo del ratón  blocks[row][column].setOnClickListener(new OnClickListener()  {  @Override  public void onClick(View view)  {  // comenzar con temporizador en la primera posición  if (!isTimerStarted)  {  startTimer();  isTimerStarted = true;  }  // establecer minas en primer clic  if (!areMinesSet)  {  areMinesSet = true;  setMines(currentRow, currentColumn);  }  // Esto no es el primer clic  // Comprobar si el bloque actual está marcado  // Si la bandera no hacen nada  // Lo que la operación es manejada por LongClick  // Si el bloque no se encuentra en posición y luego descubrir los bloques cercanos  // Hasta que lleguemos minas numeradas  if (!blocks[currentRow][currentColumn].isFlagged())  {  // abrir los bloques cercanos hasta que lleguemos bloques numerados  rippleUncover(currentRow, currentColumn);    // Qué hemos hecho clic una mina  if (blocks[currentRow][currentColumn].hasMine())  {  // game over  finishGame(currentRow,currentColumn);  }  // check si ganamos el juego  if (checkGameWin())  {  // marcar el juego como ganado!  winGame();  }  }  }  });  // Agregamos Long Click listener  // Como si hicieramos click derecho con el mouse  blocks[row][column].setOnLongClickListener(new OnLongClickListener()  {  public boolean onLongClick(View view)  {  // simula click medio  // Si se trata de una pulsación larga en una mina abierta luego  // Abrir todos los bloques circundantes  if (!blocks[currentRow][currentColumn].isCovered() && (blocks[currentRow][currentColumn].getNumberOfMinesInSorrounding() > 0) && !isGameOver)  {  int nearbyFlaggedBlocks = 0;  for (int previousRow = -1; previousRow < 2; previousRow++)  {  for (int previousColumn = -1; previousColumn < 2; previousColumn++)  {  if (blocks[currentRow + previousRow][currentColumn + previousColumn].isFlagged())  {  nearbyFlaggedBlocks++;  }  }  }  // Si el número de bloque marcado es igual a cerca del número de minas  // A continuación, abra los bloques cercanos  if (nearbyFlaggedBlocks == blocks[currentRow][currentColumn].getNumberOfMinesInSorrounding())  {  for (int previousRow = -1; previousRow < 2; previousRow++)  {  for (int previousColumn = -1; previousColumn < 2; previousColumn++)  {  // no abra bloques marcados  if (!blocks[currentRow + previousRow][currentColumn + previousColumn].isFlagged())  {  // bloques abiertos hasta que lleguemos bloque numerada  rippleUncover(currentRow + previousRow, currentColumn + previousColumn);  // hemos hecho clic una mina  if (blocks[currentRow + previousRow][currentColumn + previousColumn].hasMine())  {  // game over  finishGame(currentRow + previousRow, currentColumn + previousColumn);  }  //ganamos el juego  if (checkGameWin())  {  // marcamos como ganado  winGame();  }  }  }  }  }  // Que ya no queremos juzgar este gesto tan regreso  // No regresar de aquí en realidad desencadenar otra acción  // Que puede ser marcado como marca bandera o pregunta en blanco  return true;  }  // si el bloque clic está activada, puede hacer clic o marcado  if (blocks[currentRow][currentColumn].isClickable() &&  (blocks[currentRow][currentColumn].isEnabled() || blocks[currentRow][currentColumn].isFlagged()))  {  // por los clics largos establecen:  // 1. bloques vacíos al marcado  // 2. pabellón de signo de interrogación  // 3. signo de interrogación para dejar en blanco  // caso 1. establecer bloques en blanco para marcado  if (!blocks[currentRow][currentColumn].isFlagged() && !blocks[currentRow][currentColumn].isQuestionMarked())  {  blocks[currentRow][currentColumn].setBlockAsDisabled(false);  blocks[currentRow][currentColumn].setFlagIcon(true);  blocks[currentRow][currentColumn].setFlagged(true);  minesToFind--; //reduce mine count  updateMineCountDisplay();  }  // el caso 2. establecer con bandera de signo de interrogación  else if (!blocks[currentRow][currentColumn].isQuestionMarked())  {  blocks[currentRow][currentColumn].setBlockAsDisabled(true);  blocks[currentRow][currentColumn].setQuestionMarkIcon(true);  blocks[currentRow][currentColumn].setFlagged(false);  blocks[currentRow][currentColumn].setQuestionMarked(true);  minesToFind++; // aumentar el número de minas  updateMineCountDisplay();  }  // caso 3. cambiar de cuadro en blanco  else  {  blocks[currentRow][currentColumn].setBlockAsDisabled(true);  blocks[currentRow][currentColumn].clearAllIcons();  blocks[currentRow][currentColumn].setQuestionMarked(false);  // si se marca a continuación, incrementar la cuenta mía  if (blocks[currentRow][currentColumn].isFlagged())  {  minesToFind++; // aumentar el número de minas  updateMineCountDisplay();  }  // eliminar el estado marcado  blocks[currentRow][currentColumn].setFlagged(false);  }    updateMineCountDisplay(); // actualizar la mía visualizar  }  return true;  }  });  }  }  }  private boolean checkGameWin()  {  for (int row = 1; row < numberOfRowsInMineField + 1; row++)  {  for (int column = 1; column < numberOfColumnsInMineField + 1; column++)  {  if (!blocks[row][column].hasMine() && blocks[row][column].isCovered())  {  return false;  }  }  }  return true;  }  private void updateMineCountDisplay()  {  if (minesToFind < 0)  {  txtMineCount.setText(Integer.toString(minesToFind));  }  else if (minesToFind < 10)  {  txtMineCount.setText("00" + Integer.toString(minesToFind));  }  else if (minesToFind < 100)  {  txtMineCount.setText("0" + Integer.toString(minesToFind));  }  else  {  txtMineCount.setText(Integer.toString(minesToFind));  }  }  private void winGame()  {  stopTimer();  isTimerStarted = false;  isGameOver = true;  minesToFind = 0; //colocar el contador de minas a 0  //colocar icono de duda  btnSmile.setBackgroundResource(R.drawable.cool);  updateMineCountDisplay(); // update mine count  // Desabilitar todos los botones  // configurar marcados todos los bloques sin bandera  for (int row = 1; row < numberOfRowsInMineField + 1; row++)  {  for (int column = 1; column < numberOfColumnsInMineField + 1; column++)  {  blocks[row][column].setClickable(false);  if (blocks[row][column].hasMine())  {  blocks[row][column].setBlockAsDisabled(false);  blocks[row][column].setFlagIcon(true);  }  }  }  // Mostrar Mensaje  showDialog("Ganaste en " + Integer.toString(secondsPassed) + " segundos!", 1000, false, true);  }  private void finishGame(int currentRow, int currentColumn)  {  isGameOver = true; // Marcar juego como terminado  stopTimer(); // Parar Cronometro  isTimerStarted = false;  btnSmile.setBackgroundResource(R.drawable.sad);  // Mostrar todas las minas  // Desabilitar los bloques  for (int row = 1; row < numberOfRowsInMineField + 1; row++)  {  for (int column = 1; column < numberOfColumnsInMineField + 1; column++)  {  // Desabilitar los bloques  blocks[row][column].setBlockAsDisabled(false);    // block tiene una mina y no esta con bandera  if (blocks[row][column].hasMine() && !blocks[row][column].isFlagged())  {  // colocar icono de mina  blocks[row][column].setMineIcon(false);  }  // Block esta con bandera pero no tiene mina  if (!blocks[row][column].hasMine() && blocks[row][column].isFlagged())  {  // colocar icono de bandera  blocks[row][column].setFlagIcon(false);  }  // block esta con bandera  if (blocks[row][column].isFlagged())  {  // desabilitar boton  blocks[row][column].setClickable(false);  }  }  }  // desencadenar mina  blocks[currentRow][currentColumn].triggerMine();  // mostrar mensaje  showDialog("Trataste en " + Integer.toString(secondsPassed) + " segundos!", 1000, false, false);  }  private void setMines(int currentRow, int currentColumn)  {  // establecer minas excluyendo la ubicación donde el usuario hace clic  Random rand = new Random();  int mineRow, mineColumn;  for (int row = 0; row < totalNumberOfMines; row++)  {  mineRow = rand.nextInt(numberOfColumnsInMineField);  mineColumn = rand.nextInt(numberOfRowsInMineField);  if ((mineRow + 1 != currentColumn) || (mineColumn + 1 != currentRow))  {  if (blocks[mineColumn + 1][mineRow + 1].hasMine())  {  row--; // mina ya está allí, no se repiten para un mismo bloque  }  // colocar mina en esta hubicacion  blocks[mineColumn + 1][mineRow + 1].plantMine();  }  // excluye el usuario hace clic en ubicación  else  {  row--;  }  }  int nearByMineCount;  // contar el número de minas en los alrededores de los bloques  for (int row = 0; row < numberOfRowsInMineField + 2; row++)  {  for (int column = 0; column < numberOfColumnsInMineField + 2; column++)  {  //para cada bloque encuentra cerca recuento mina  nearByMineCount = 0;  if ((row != 0) && (row != (numberOfRowsInMineField + 1)) && (column != 0) && (column != (numberOfColumnsInMineField + 1)))  {  // check in all nearby blocks  for (int previousRow = -1; previousRow < 2; previousRow++)  {  for (int previousColumn = -1; previousColumn < 2; previousColumn++)  {  if (blocks[row + previousRow][column + previousColumn].hasMine())  {  // comprobar en todos los bloques cercanos  nearByMineCount++;  }  }  }  blocks[row][column].setNumberOfMinesInSurrounding(nearByMineCount);  }  // para las filas laterales (0 ª y última fila / columna)  // ajusta cuenta como 9 y marcarlo como leído  else  {  blocks[row][column].setNumberOfMinesInSurrounding(9);  blocks[row][column].OpenBlock();  }  }  }  }  private void rippleUncover(int rowClicked, int columnClicked)  {  // no abra filas marcadas o extraído  if (blocks[rowClicked][columnClicked].hasMine() || blocks[rowClicked][columnClicked].isFlagged())  {  return;  }  // abrir bloque clickeado  blocks[rowClicked][columnClicked].OpenBlock();  // si hace clic bloque tiene minas cercanas y luego no abrir más  if (blocks[rowClicked][columnClicked].getNumberOfMinesInSorrounding() != 0 )  {  return;  }  // abrirá el próximo 3 filas y 3 columnas de forma recursiva  for (int row = 0; row < 3; row++)  {  for (int column = 0; column < 3; column++)  {  // Comprobar todas las condiciones que haya marcado  // Si cumplen los bloques posteriores se abren  if (blocks[rowClicked + row - 1][columnClicked + column - 1].isCovered()  && (rowClicked + row - 1 > 0) && (columnClicked + column - 1 > 0)  && (rowClicked + row - 1 < numberOfRowsInMineField + 1) && (columnClicked + column - 1 < numberOfColumnsInMineField + 1))  {  rippleUncover(rowClicked + row - 1, columnClicked + column - 1 );  }  }  }  return;  }  public void startTimer()  {  if (secondsPassed == 0)  {  timer.removeCallbacks(updateTimeElasped);  // decirle temporizador para ejecutar llamar después de 1 segundo  timer.postDelayed(updateTimeElasped, 1000);  }  }  public void stopTimer()  {  // desactivar rellamadas  timer.removeCallbacks(updateTimeElasped);  }  // temporizador devolver la llamada cuando está marcada temporizador  private Runnable updateTimeElasped = new Runnable()  {  public void run()  {  long currentMilliseconds = System.currentTimeMillis();  ++secondsPassed;  if (secondsPassed < 10)  {  txtTimer.setText("00" + Integer.toString(secondsPassed));  }  else if (secondsPassed < 100)  {  txtTimer.setText("0" + Integer.toString(secondsPassed));  }  else  {  txtTimer.setText(Integer.toString(secondsPassed));  }    // agregar notificacion  timer.postAtTime(this, currentMilliseconds);  // Notificar a llamar después de 1 segundo  // Básicamente a permanecer en el bucle temporizador  timer.postDelayed(updateTimeElasped, 1000);  }  };    private void showDialog(String message, int milliseconds, boolean useSmileImage, boolean useCoolImage)  {  // Mostrar mensaje  Toast dialog = Toast.makeText(  getApplicationContext(),  message,  Toast.LENGTH\_LONG);  dialog.setGravity(Gravity.CENTER, 0, 0);  LinearLayout dialogView = (LinearLayout) dialog.getView();  ImageView coolImage = new ImageView(getApplicationContext());  if (useSmileImage)  {  coolImage.setImageResource(R.drawable.smile);  }  else if (useCoolImage)  {  coolImage.setImageResource(R.drawable.cool);  }  else  {  coolImage.setImageResource(R.drawable.sad);  }  dialogView.addView(coolImage, 0);  dialog.setDuration(milliseconds);  dialog.show();  }  } |

|  |
| --- |
| main.xml |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <TableLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  android:stretchColumns=*"0,2"*  android:background=*"@drawable/back"*>  <TableRow>  <TextView  android:id=*"@+id/Timer"*  android:layout\_column=*"0"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"48px"*  android:gravity=*"center\_horizontal"*  android:padding=*"5dip"*  android:textColor=*"#FFFFFF"*  android:textSize=*"40sp"*  android:text=*"000"* />    <ImageButton android:id=*"@+id/Smiley"*  android:layout\_column=*"1"*  android:background=*"@drawable/smiley\_button\_states"*  android:scaleType=*"center"*  android:padding=*"5dip"*  android:layout\_width=*"48px"*  android:layout\_height=*"48px"*/>    <TextView  android:id=*"@+id/MineCount"*  android:layout\_column=*"2"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"48px"*  android:gravity=*"center\_horizontal"*  android:padding=*"5dip"*  android:textColor=*"#FFFFFF"*  android:textSize=*"40sp"*  android:text=*"000"* />  </TableRow>    <TableRow>  <TextView  android:layout\_column=*"0"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"50px"*  android:layout\_span=*"3"*  android:padding=*"10dip"*/>  </TableRow>    <TableRow>  <TableLayout  android:id=*"@+id/MineField"*  android:layout\_width=*"260px"*  android:layout\_height=*"260px"*  android:gravity=*"bottom"*  android:stretchColumns=*"\*"*  android:layout\_span=*"3"*  android:padding=*"5dip"* >  </TableLayout>  </TableRow>  </TableLayout> |