STEM FUSE
A Revolution in STEM Education

Mobile App:IT

MineSweeper Bug Fixing

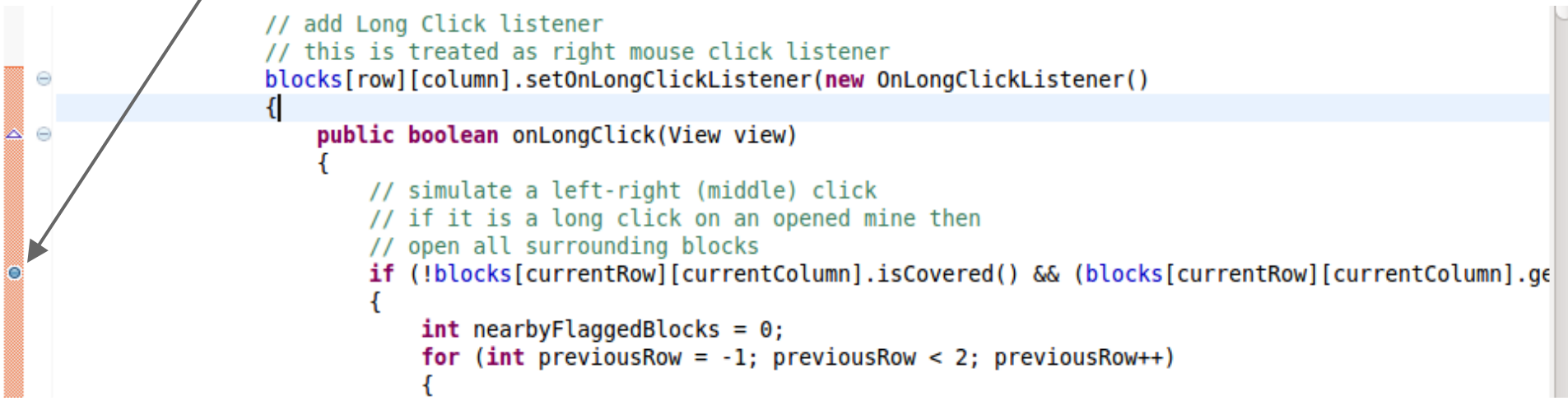
BUG LIST

Here is a list of all the issues that will need to be fixed in the Minesweeper application.

- Ability to mark tiles as a bomb (F) or as a question mark (?) is not working.
- Need to make sure all bomb counter numbers show up in a different color. Currently they all show up in black.
- Grid needs to be 9 tiles by 9 tiles. It is currently set to 8 tiles by 8 tiles.
- Number of mines needs be be decreased from 15 mines per game to 10 mines per game.
- Timer is not displaying the time when the game is ended.
- Initial Dialog box is not showing up when the app starts

MARKING BOMBS

- The first bug we will be attempting to fix is the marking of bombs and question marks. If you run the application and perform a long click, you will notice that nothing is happening.
- The **onLongClick** method in the **MinesweeperGame.java** file gets triggered when the long click event occurs. Lets set a breakpoint on the first line in that method.



```
// add Long Click listener
// this is treated as right mouse click listener
blocks[row][column].setOnLongClickListener(new OnLongClickListener()
{
    public boolean onLongClick(View view)
    {
        // simulate a left-right (middle) click
        // if it is a long click on an opened mine then
        // open all surrounding blocks
        if (!blocks[currentRow][currentColumn].isCovered() && (blocks[currentRow][currentColumn].ge
        {
            int nearbyFlaggedBlocks = 0;
            for (int previousRow = -1; previousRow < 2; previousRow++)
            {
```

MARKING BOMBS

- Now run the application in Debug mode by going to **Run > Debug**.
- In the application, perform a long click. This should trigger the breakpoint that you just set (If you get a Confirm Perspective Switch message, click Yes).
- Walk through the click handler all the way through using the **Step Over** button. 
- The below code is what controls the bomb flagging.

```
// case 1. set blank block to flagged
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();
}
```

MARKING BOMBS

- You will notice that if you step over that **if statement**, the code inside the **if statement** is not getting run.
- If we analyze what the **if statement** is saying, it goes something like "If the current row and column (the current tile) is flagged as a bomb and the current tile is marked with a question mark, then flag the tile as a bomb". Does this seem correct?
- The **if statement** should be flagging the tile as a bomb if it is not marked as a bomb yet and it is not marked as a question mark yet. What the **if statement** should read is "If the current tile is NOT flagged as a bomb and the current tile is NOT marked with a question mark, then flag the tile as a bomb.

MARKING BOMBS

- We will now fix the **if statement** by adding an exclamation point (!) to the beginning of each tile check. This adds the NOT that will be required in order to get the **if statement** working.

```
// case 1. set blank block to flagged
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();
}
```

- Now run the application and see if the long click works to flag as a bomb. Try to flag the tile as a question mark now, there still seems to be a problem.
- Debug the application again paying attention the next **if statement** in that section. This one has the same **logic error**.

MARKING BOMBS

- We will now fix the second **if statement** by adding an exclamation point (!) to the beginning of the tile check. This adds the NOT that will be required in order to get the **if statement** working.

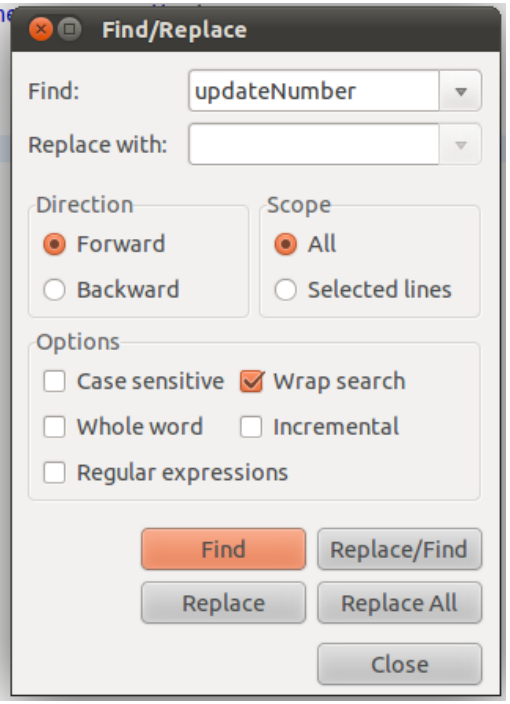
```
// case 2. set flagged to question mark
else if (!blocks[currentRow][currentColumn].isQuestionMarked())
{
    blocks[currentRow][currentColumn].setBlockAsDisabled(true);
    blocks[currentRow][currentColumn].setQuestionMarkIcon(true);
    blocks[currentRow][currentColumn].setFlagged(false);
    blocks[currentRow][currentColumn].setQuestionMarked(true);
    minesToFind++; // increase mine count
    updateMineCountDisplay();
}
```

- Now run the application and see that the long clicks are working correctly.

BOMB COUNTER TEXT COLOR

- The bomb counter text color gets set in the **Block.java** file in the **updateNumber** method. Open up the **Block.java** file and search for the **updateNumber** method. If you go to **Edit > Find/Replace** or use **Ctrl+F** you can easily search for the method name.

```
            setNumberOfSurroundingMines(numberOfMine
        }
    }

    // set text as nearby mine count
    public void updateNumber(int text)
    {
        if (text != 0)
        {
            this.setText(Integer.toString(text));
            this.setTextColor(Color.BLACK);
        }
    }

    // set block as a mine underneath
    public void plantMine()
    {
        isMined = true;
    }

    // mine was opened
    // change the block icon and color
    public void triggerMine()
    {
        setMineIcon(true);
        this.setTextColor(Color.RED);
```



BOMB COUNTER TEXT COLOR

- Notice the **this.setTextColor** line of code. This is setting the color of the bomb counter text (the 1, 2, 3, etc numbers that display how many bombs are nearby).

```
// set text as nearby mine count
public void updateNumber(int text)
{
    if (text != 0)
    {
        this.setText(Integer.toString(text));
        this.setTextColor(Color.BLACK);
    }
}
```

- We basically need to write some code that says, "If the number is 1, use this color", "If the number is 2, user this color", and so on up until 8 (the maximum number of bombs that can be around any single tile.

BOMB COUNTER TEXT COLOR

- Instead of writing 8 different **if statements** (which would work), we will instead use a more efficient **switch statement**. This will allow us to check the number value against a larger number of conditions much easier. Here is the new code for the **updateNumber** method. Replace your current **updateNumber** method with the code on the next slide.

```
// set text as nearby mine count
public void updateNumber(int text)
{
    if (text != 0)
    {
        this.setText(Integer.toString(text));

        // select different color for each number
        // we have already skipped 0 mine count
        switch (text)
        {
            case 1:
                this.setTextColor(Color.BLUE);
                break;
            case 2:
                this.setTextColor(Color.YELLOW);
                break;
            case 3:
                this.setTextColor(Color.RED);
                break;
            case 4:
                this.setTextColor(Color.CYAN);
                break;
            case 5:
                this.setTextColor(Color.MAGENTA);
                break;
            case 6:
                this.setTextColor(Color.GREEN);
                break;
            case 7:
                this.setTextColor(Color.BLACK);
                break;
            case 8:
                this.setTextColor(Color.GRAY);
                break;
        }
    }
}
```

We use a **switch statement** here to test the value of the **text** variable. Each **case** section tests if the **text** variable is equal to that value, if so, it will run that code. The **break statement** then tells the code to only run one section of the **switch statement.**

After adding this code, set a **breakpoint** on the **switch statement** line and debug the application to see how it works.

BOMB COUNTER TEXT COLOR

- Here is a screenshot after adding the various text colors.

DISPLAY 9X9 GRID

- The grid dimensions are controlled by two variables in the **MinesweeperGame.java** file. The variable names controlling the dimensions are **numberOfRowsInMineField** and **numberofColumnsInMineField**. Go ahead and change these values to 9 instead of 8.

- Run the application to verify the grid dimensions of the game have increased to 9x9.

SET MINE COUNT TO 10

- The mine count is controlled by a variable in the **MinesweeperGame.java** file. The variable name controlling the mine count is **totalNumberOfMines**. Change this value to 10.

- Run the application to ensure the number of mines has indeed decreased from 15 to 10.

FIX TIMER DISPLAY

- In the **MinesweeperGame.java** file, the **finishGame** method and the **winGame** method displays a dialog message with the time in seconds that the game was played. The problem is the timer is currently displaying 0.

This should say, "You tried for 3 seconds!"

FIX TIMER DISPLAY

- The line of code displaying the dialog is in the **finishGame** method is:

```
// show message
showDialog("You tried for " + Integer.toString(secondsPassed) + " seconds!", 1000, false, false);
```

- This means that the **Integer.toString(secondsPassed)** section is returning a 0. This means that something must be wrong with the **secondsPassed** variable.
- Lets set a breakpoint at the top of the **finishGame** function and walk through the code monitoring the **secondsPassed** variable.

Here we set a breakpoint to monitor the **secondsPassed** variable.

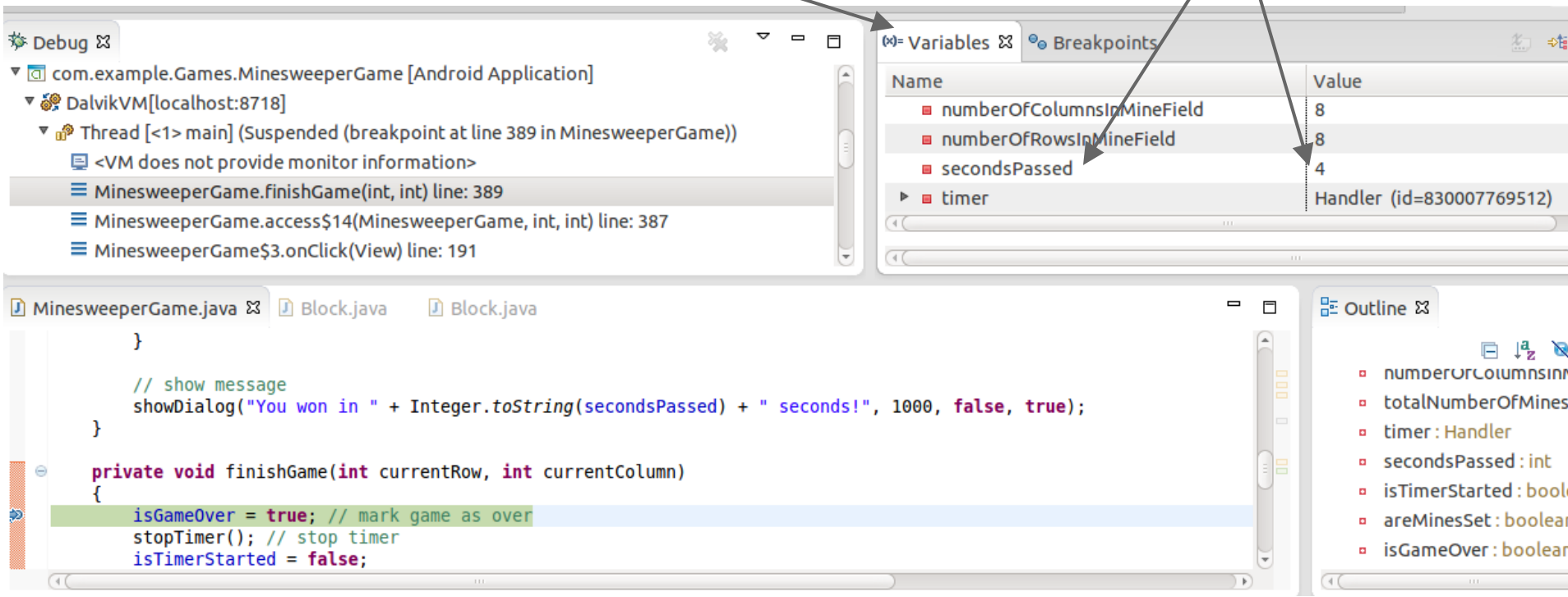
Now let's debug the application

```
private void finishGame(int currentRow, int currentColumn)
{
    isGameOver = true; // mark game as over
    stopTimer(); // stop timer
    isTimerStarted = false;
    btnSmile.setBackgroundResource(R.drawable.sad);
    secondsPassed = 0;

    // show all mines
    // disable all blocks
    for (int row = 1; row < numberOfRowsInMineField + 1; row++)
    {
        for (int column = 1; column < numberOfColumnsInMineField + 1; column++)
        {
            // disable block
            blocks[row][column].setBlockAsDisabled(false);

            // block has mine and is not flagged
            if (blocks[row][column].hasMine() && !blocks[row][column].isFlagged())
            {
                // set mine icon
                blocks[row][column].setMineIcon(false);
            }

            // block is flagged and doesn't not have mine
            if (!blocks[row][column].hasMine() && blocks[row][column].isFlagged())
            {
                // set flag icon
                blocks[row][column].setFlagIcon(false);
            }

            // block is flagged
            if (blocks[row][column].isFlagged())
            {
                // disable the block
                blocks[row][column].setClickable(false);
            }
        }
    }
```

DEBUGGING THE TIMER DISPLAY

In the **Variables** pane, expand the **this** variable and scroll down to find the **secondsPassed** variable.

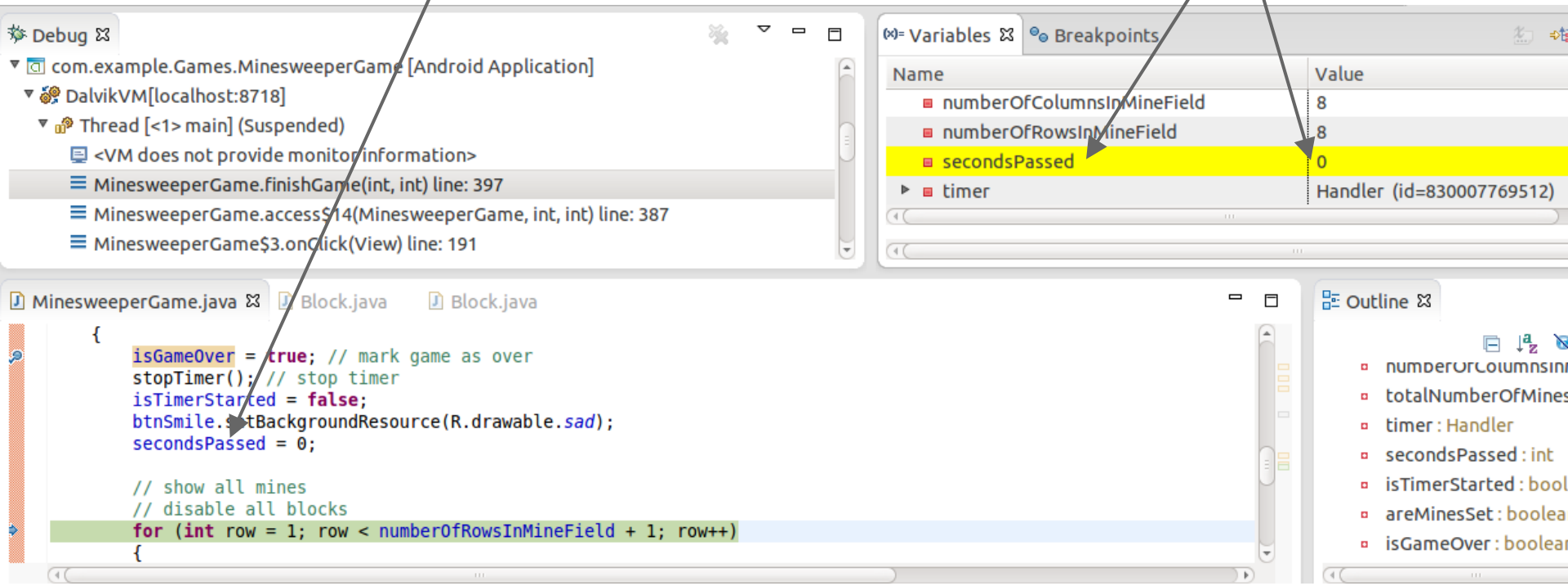
Here you can see the **secondsPassed** variable is currently set to 4.



DEBUGGING THE TIMER DISPLAY

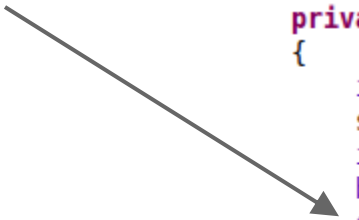
If you step through a few lines of code, you see that the **secondsPassed** variable gets reset.

Here you can see the **secondsPassed** variable was reset to 0 before the showDialog method was called.



FIX TIMER DISPLAY

- Fixing this error is as simple as removing the following line of code:



```
private void finishGame(int currentRow, int currentColumn)
{
    isGameOver = true; // mark game as over
    stopTimer(); // stop timer
    isTimerStarted = false;
    btnSmile.setBackgroundResource(R.drawable.sad);
    secondsPassed = 0;

    // show all mines
    // disable all blocks
    for (int row = 1; row < numberOfRowsInMineField + 1; row++)
```

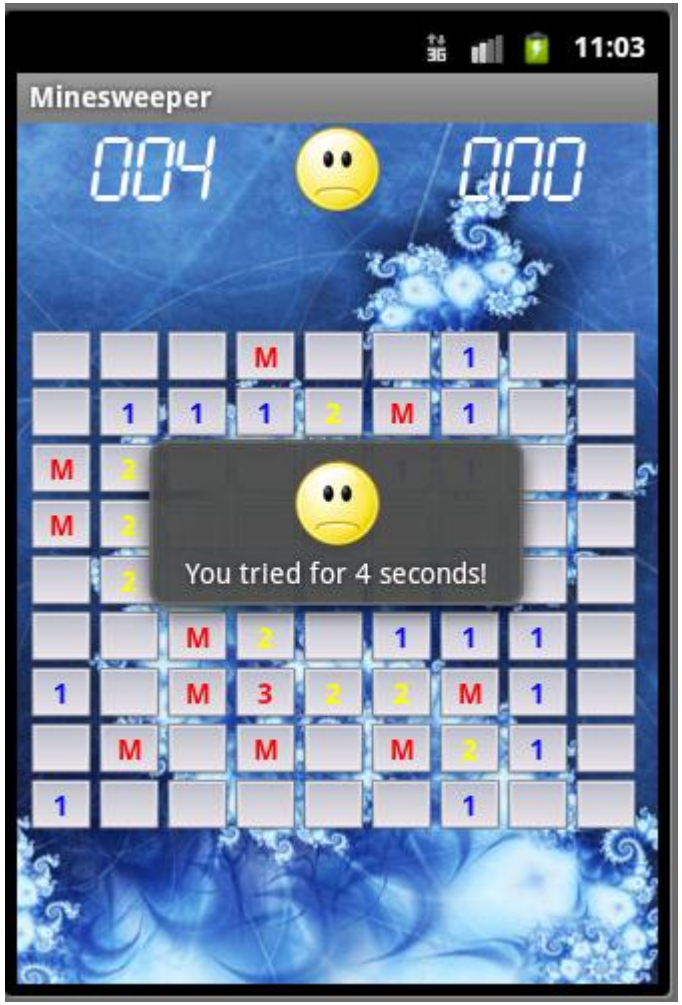
The end result for the **finishGame** method should look like:

```
private void finishGame(int currentRow, int currentColumn)
{
    isGameOver = true; // mark game as over
    stopTimer(); // stop timer
    isTimerStarted = false;
    btnSmile.setBackgroundResource(R.drawable.sad);

    // show all mines
    // disable all blocks
    for (int row = 1; row < numberOfRowsInMineField + 1; row++)
```

FIX TIMER DISPLAY

- The application now displays the time if you lose.

- On your own, debug the **winGame** method to fix the timer for that dialog message as well.



INITIAL DIALOG BOX DISPLAY

- The last bug we need to fix is to display the informational dialog window when the game first starts. The dialog should look like this:

- This is just a simple dialog message that needs to be added to the **onCreate** method in the **MinesweeperGame.java** file.

INITIAL DIALOG BOX DISPLAY

- If you look at how the other dialog messages were displayed, they are simply a call to the **showDialog** method. We will add a call to the method of our own, in the **onCreate** method of the **MinesweeperGame.java** file. The line of code we are adding is:

```
showDialog("Click smiley to start New Game", 2000, true, false);
```

The full **onCreate** method is on the next slide.

```java
@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);

    // set font style for timer and mine count to LCD style
    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 smiley to start New Game", 2000, true, false);
}
```

ON YOUR OWN

- Change the dimensions of the grid and the number of mines and play the game. Notice how easy it is to change the difficulty of the game by changing these simple variables. Also notice how making these values variables makes it easy to change the gameplay without having to dig through methods to find out how the grid and mines are created.