**Minesweeper**

In this part of the Java 2D games tutorial, we will create a Minesweeper game clone.

**Minesweeper**

Minesweeper is a popular board game shipped with many operating systems by default. The goal of the game is to sweep all mines from a mine field. If the player clicks on the cell which contains a mine, the mine detonates and the game is over. Further a cell can contain a number or it can be blank. The number indicates how many mines are adjacent to this particular cell. We set a mark on a cell by right clicking on it. This way we indicate that we believe, there is a mine.

Board.java

package com.zetcode;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.Random;

import javax.swing.ImageIcon;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class Board extends JPanel {

private final int NUM\_IMAGES = 13;

private final int CELL\_SIZE = 15;

private final int COVER\_FOR\_CELL = 10;

private final int MARK\_FOR\_CELL = 10;

private final int EMPTY\_CELL = 0;

private final int MINE\_CELL = 9;

private final int COVERED\_MINE\_CELL = MINE\_CELL + COVER\_FOR\_CELL;

private final int MARKED\_MINE\_CELL = COVERED\_MINE\_CELL + MARK\_FOR\_CELL;

private final int DRAW\_MINE = 9;

private final int DRAW\_COVER = 10;

private final int DRAW\_MARK = 11;

private final int DRAW\_WRONG\_MARK = 12;

private final int N\_MINES = 40;

private final int N\_ROWS = 16;

private final int N\_COLS = 16;

private int[] field;

private boolean inGame;

private int mines\_left;

private Image[] img;

private int all\_cells;

private JLabel statusbar;

public Board(JLabel statusbar) {

this.statusbar = statusbar;

img = new Image[NUM\_IMAGES];

for (int i = 0; i < NUM\_IMAGES; i++) {

img[i] = (new ImageIcon(i + ".png")).getImage();

}

setDoubleBuffered(true);

addMouseListener(new MinesAdapter());

newGame();

}

private void newGame() {

Random random;

int current\_col;

int i = 0;

int position = 0;

int cell = 0;

random = new Random();

inGame = true;

mines\_left = N\_MINES;

all\_cells = N\_ROWS \* N\_COLS;

field = new int[all\_cells];

for (i = 0; i < all\_cells; i++)

field[i] = COVER\_FOR\_CELL;

statusbar.setText(Integer.toString(mines\_left));

i = 0;

while (i < N\_MINES) {

position = (int) (all\_cells \* random.nextDouble());

if ((position < all\_cells) &&

(field[position] != COVERED\_MINE\_CELL)) {

current\_col = position % N\_COLS;

field[position] = COVERED\_MINE\_CELL;

i++;

if (current\_col > 0) {

cell = position - 1 - N\_COLS;

if (cell >= 0)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

cell = position - 1;

if (cell >= 0)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

cell = position + N\_COLS - 1;

if (cell < all\_cells)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

}

cell = position - N\_COLS;

if (cell >= 0)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

cell = position + N\_COLS;

if (cell < all\_cells)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

if (current\_col < (N\_COLS - 1)) {

cell = position - N\_COLS + 1;

if (cell >= 0)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

cell = position + N\_COLS + 1;

if (cell < all\_cells)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

cell = position + 1;

if (cell < all\_cells)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

}

}

}

}

public void find\_empty\_cells(int j) {

int current\_col = j % N\_COLS;

int cell;

if (current\_col > 0) {

cell = j - N\_COLS - 1;

if (cell >= 0)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

cell = j - 1;

if (cell >= 0)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

cell = j + N\_COLS - 1;

if (cell < all\_cells)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

}

cell = j - N\_COLS;

if (cell >= 0)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

cell = j + N\_COLS;

if (cell < all\_cells)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

if (current\_col < (N\_COLS - 1)) {

cell = j - N\_COLS + 1;

if (cell >= 0)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

cell = j + N\_COLS + 1;

if (cell < all\_cells)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

cell = j + 1;

if (cell < all\_cells)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

}

}

@Override

public void paintComponent(Graphics g) {

int cell = 0;

int uncover = 0;

for (int i = 0; i < N\_ROWS; i++) {

for (int j = 0; j < N\_COLS; j++) {

cell = field[(i \* N\_COLS) + j];

if (inGame && cell == MINE\_CELL)

inGame = false;

if (!inGame) {

if (cell == COVERED\_MINE\_CELL) {

cell = DRAW\_MINE;

} else if (cell == MARKED\_MINE\_CELL) {

cell = DRAW\_MARK;

} else if (cell > COVERED\_MINE\_CELL) {

cell = DRAW\_WRONG\_MARK;

} else if (cell > MINE\_CELL) {

cell = DRAW\_COVER;

}

} else {

if (cell > COVERED\_MINE\_CELL)

cell = DRAW\_MARK;

else if (cell > MINE\_CELL) {

cell = DRAW\_COVER;

uncover++;

}

}

g.drawImage(img[cell], (j \* CELL\_SIZE),

(i \* CELL\_SIZE), this);

}

}

if (uncover == 0 && inGame) {

inGame = false;

statusbar.setText("Game won");

} else if (!inGame)

statusbar.setText("Game lost");

}

class MinesAdapter extends MouseAdapter {

@Override

public void mousePressed(MouseEvent e) {

int x = e.getX();

int y = e.getY();

int cCol = x / CELL\_SIZE;

int cRow = y / CELL\_SIZE;

boolean rep = false;

if (!inGame) {

newGame();

repaint();

}

if ((x < N\_COLS \* CELL\_SIZE) && (y < N\_ROWS \* CELL\_SIZE)) {

if (e.getButton() == MouseEvent.BUTTON3) {

if (field[(cRow \* N\_COLS) + cCol] > MINE\_CELL) {

rep = true;

if (field[(cRow \* N\_COLS) + cCol] <= COVERED\_MINE\_CELL) {

if (mines\_left > 0) {

field[(cRow \* N\_COLS) + cCol] += MARK\_FOR\_CELL;

mines\_left--;

statusbar.setText(Integer.toString(mines\_left));

} else

statusbar.setText("No marks left");

} else {

field[(cRow \* N\_COLS) + cCol] -= MARK\_FOR\_CELL;

mines\_left++;

statusbar.setText(Integer.toString(mines\_left));

}

}

} else {

if (field[(cRow \* N\_COLS) + cCol] > COVERED\_MINE\_CELL) {

return;

}

if ((field[(cRow \* N\_COLS) + cCol] > MINE\_CELL) &&

(field[(cRow \* N\_COLS) + cCol] < MARKED\_MINE\_CELL)) {

field[(cRow \* N\_COLS) + cCol] -= COVER\_FOR\_CELL;

rep = true;

if (field[(cRow \* N\_COLS) + cCol] == MINE\_CELL)

inGame = false;

if (field[(cRow \* N\_COLS) + cCol] == EMPTY\_CELL)

find\_empty\_cells((cRow \* N\_COLS) + cCol);

}

}

if (rep)

repaint();

}

}

}

}

First we will define the constants used in our game.

private final int NUM\_IMAGES = 13;

private final int CELL\_SIZE = 15;

There are 13 images used in this game. A cell can be surrounded by maximum of 8 mines, so we need numbers 1..8. We need images for an empty cell, a mine, a covered cell, a marked cell and finally for a wrongly marked cell. The size of each of the images is 15x15px.

private final int COVER\_FOR\_CELL = 10;

private final int MARK\_FOR\_CELL = 10;

private final int EMPTY\_CELL = 0;

...

A mine field is an array of numbers. For example 0 denotes an empty cell. Number 10 is used for a cell cover as well as for a mark. Using constants improves readability of the code.

private final int N\_MINES = 40;

private final int N\_ROWS = 16;

private final int N\_COLS = 16;

The minefield in our game has 40 hidden mines. There are 16 rows and 16 columns in this field. So there are 256 cells together in the minefield.

private int[] field;

The field is an array of numbers. Each cell in the field has a specific number. E.g. a mine cell has number 9. A cell with number 2, meaning it is adjacent to two mines, has number two. The numbers are added. For example, a covered mine has number 19, 9 for the mine and 10 for the cell cover etc.

for (int i = 0; i < NUM\_IMAGES; i++) {

img[i] = (new ImageIcon(i + ".png")).getImage();

}

Here we load our images into the image array. The images are named 0.png, 1.png ... 12.png.

The newGame() initiates the Minesweeper game.

all\_cells = N\_ROWS \* N\_COLS;

field = new int[all\_cells];

for (i = 0; i < all\_cells; i++)

field[i] = COVER\_FOR\_CELL;

These lines set up the mine field. Every cell is covered by default.

i = 0;

while (i < N\_MINES) {

position = (int) (all\_cells \* random.nextDouble());

if ((position < all\_cells) &&

(field[position] != COVERED\_MINE\_CELL)) {

current\_col = position % N\_COLS;

field[position] = COVERED\_MINE\_CELL;

i++;

...

In the while cycle we randomly position all mines in the field.

cell = position - cols;

if (cell >= 0)

if (field[cell] != COVERED\_MINE\_CELL)

field[cell] += 1;

Each of the cells can be surrounded up to 8 cells. (This does not apply to the border cells.) We raise the number for adjacent cells for each of the randomly placed mine. In our example, we add 1 to the top neighbor of the cell in question.

In the find\_empty\_cells() method, we find empty cells. If the player clicks on a mine cell, the game is over. If he clicks on a cell adjacent to a mine, he uncovers a number indicating how many mines the cell is adjacent to. Clicking on an empty cell leads to uncovering many other empty cells plus cells with a number that form a border around a space of empty borders. We use a recursive algorithm to find empty cells.

cell = j - 1;

if (cell >= 0)

if (field[cell] > MINE\_CELL) {

field[cell] -= COVER\_FOR\_CELL;

if (field[cell] == EMPTY\_CELL)

find\_empty\_cells(cell);

}

In this code, we check the cell that is left to an empty cell in question. If it is not empty, it is uncovered. If it is empty, we repeat the whole process by recursively calling the find\_empty\_cells()method.

The paintComponent() method turns numbers into images.

if (!inGame) {

if (cell == COVERED\_MINE\_CELL) {

cell = DRAW\_MINE;

} else if (cell == MARKED\_MINE\_CELL) {

cell = DRAW\_MARK;

} else if (cell > COVERED\_MINE\_CELL) {

cell = DRAW\_WRONG\_MARK;

} else if (cell > MINE\_CELL) {

cell = DRAW\_COVER;

}

}

If the game is over and we lost, we show all uncovered mines if any and show all wrongly marked cells if any.

g.drawImage(img[cell], (j \* CELL\_SIZE),

(i \* CELL\_SIZE), this);

This code line draws every cell on the window.

In the mousePressed() method we react to mouse clicks. The Minesweeper game is controlled solely by mouse. We react to left and right mouse clicks.

field[(cRow \* N\_COLS) + cCol] += MARK\_FOR\_CELL;

mines\_left--;

If we right click on an unmarked cell, we add MARK\_FOR\_CELL to the number representing the cell. This leads to drawing a covered cell with a mark in the paintComponent() method.

if (field[(cRow \* N\_COLS) + cCol] > COVERED\_MINE\_CELL) {

return;

}

Nothing happens if we click on the covered & marked cell. It must by first uncovered by another right click and only then it is possible to left click on it.

field[(cRow \* N\_COLS) + cCol] -= COVER\_FOR\_CELL;

A left click removes a cover from the cell.

if (field[(cRow \* N\_COLS) + cCol] == MINE\_CELL)

inGame = false;

if (field[(cRow \* N\_COLS) + cCol] == EMPTY\_CELL)

find\_empty\_cells((cRow \* N\_COLS) + cCol);

In case we left clicked on a mine, the game is over. If we left clicked on an empty cell, we call thefind\_empty\_cells() method which recursively finds all adjacent empty cells.

Mines.java

package com.zetcode;

import java.awt.BorderLayout;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.SwingUtilities;

public class Mines extends JFrame {

private final int FRAME\_WIDTH = 250;

private final int FRAME\_HEIGHT = 290;

private final JLabel statusbar;

public Mines() {

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(FRAME\_WIDTH, FRAME\_HEIGHT);

setLocationRelativeTo(null);

setTitle("Minesweeper");

statusbar = new JLabel("");

add(statusbar, BorderLayout.SOUTH);

add(new Board(statusbar));

setResizable(false);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

JFrame ex = new Mines();

ex.setVisible(true);

}

});

}

}

This is the main class.

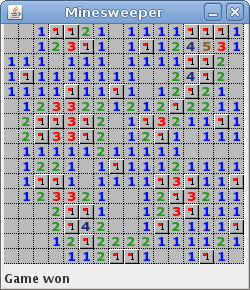


Figure: Minesweeper