2002-12-11

AMRÖJ

AMMUNITIONSRÖJNING

Ett litet spel av Peter Asplund och Anders Fjeldstad

Innehållsförteckning

Anders Fjeldstad
Peter Asplund
MT1A

Anders Fjeldstad
Peter Asplund
MT1A

Projektbeskrivning

Projektet gick ut på att utveckla ett litet spel av typ Minesweeper (Minröj). Spelet går ut på att röja ett ”minfält” bestående av ett antal rutor. Man spelar spelet genom att klicka på rutorna, som då ”grävs upp”. Under varje ruta kan man antingen stöta på en mina, eller en siffra som talar om hur många minor det totalt finns på de angränsande rutorna. Naturligtvis kan en ruta även vara helt tom. När man listat ut var det finns en mina högerklickar man på den rutan för att markera den med en liten flagga. Det blir då omöjligt att klicka på just den rutan. Om man sedan ångrar sig går det bra att avmarkera rutan genom att högerklicka på den igen.

Spelet är över när man har grävt upp samtliga rutor förutom de som innehåller minor. Att markera minorna är valfritt, men kan vara ett praktiskt verktyg på stora minfält. Om man klickar på en ruta som innehåller en mina har man förlorat och spelet är över.

När man klarat av en spelomgång kommer en poäng att räknas ut, baserad på antalet minor och rutor samt den tid som det tog att rensa minfältet. Om poängen är tillräckligt bra ombeds spelaren att skriva in sitt namn, som då kommer att lagras i en lista över de tio bästa poängen.

Vi valde att även implementera en funktion som gör att spelaren kan välja mellan tre olika svårighetsgrader.

Arbetsgång

Det är alltid viktigt att börja med att planera själva koden ordentligt, annars blir det lätt spaghetti av alltihop. Vi satte oss ned och diskuterade olika lösningar på de delproblem uppgiften innebar. Det tog kanske ett par timmar innan vi hade grundstrukturen klar för oss och vi kunde börja med själva programmerandet.

Spelet skulle i och för sig vara grafiskt, men vi valde att koncentrera oss enbart på själva spelmotorn i början. Vi delade upp arbetet så jämnt vi kunde mellan oss och skapade en version som var spelbar i textläge. Detta var möjligt eftersom spelet är detsamma under ytan, det grafiska gränssnittet är bara ett skal som gör det enklare och roligare för användaren att interagera med programmet.

I det här läget fungerade de mest grundläggande spelfunktionerna. Man gav programmet koordinater via tangentbordet och det utförde de funktioner som behövs under en spelomgång. Ännu fanns ingen tidtagning, highscorelista eller några olika svårighetsgrader. Man kunde inte heller starta om spelet, utan var då tvungen att starta om hela programmet. Detta gjorde dock inget, då det textbaserade gränssnittet bara var temporärt och existerade bara för att vi skulle kunna testa kärnan i spelmotorn.

Nästa steg var att utveckla och finslipa de sista funktionerna samt det grafiska gränssnittet. Då ingen av oss tidigare använt AWT eller Swing innebar det att mycket tid lades på att inhämta kunskaper från den bok vi använder (Java 2 Bible av Walsh, Couch och Steinberg) samt API:n på http://www.java.sun.com.

När spelet var spelbart i grafiskt läge lät vi några utomstående personer testa det och komma med synpunkter, vilket bidrog till ännu mer finslipning och några nya, mindre funktioner.

Programmets struktur

För att beskriva hur vi löste uppgiften vill vi här försöka förklara den grundläggande strukturen i korthet. Detaljer kan utläsas ur Appendix A, där vi bifogat den fullständiga källkoden.

För att representera minfältet internt har vi valt att använda en tvådimensionell array av typen Zone. Precis som man kan misstänka innehåller varje position i arrayen en instans av klassen Zone, som i sin tur är mallen för de egenskaper varje ruta på spelplanen måste ha. På detta sätt vet varje ruta själv om den är minerad, hur många minor det finns runtomkring den, om den är markerad, vilken bild som skall representera den grafiskt och så vidare.

Klassen Board är en mall för hur man hanterar den interna representationen av spelplanen, och genom att skapa en instans av denna klass har man automatiskt tillgång till en färdig spelplan med de vitala metoderna för att manipulera den. Board använder sig i sin tur av några statiska metoder man finner i klasserna Minelayer och Misc.

Highscorelistan lagras i en vanlig textfil i samma katalog som resten av spelet ligger. Spelet kan läsa in listan via metoder i klassen Highscore, som i sin tur använder instanser av Highscore_Input för att göra hanteringen av data smidigare. När man skapar en instans av Highscore kan man på ett enkelt sätt läsa highscorelistan och dessutom få en färdig JDialog returnerad via en metod i Highscore. Detta gör hanteringen av highscorelistan bekväm och enkel.

Genom att instansiera Game_Timer har man en färdig klocka man kan använda för att ha koll på tiden under varje spelomgång. Naturligtvis innehåller Game_Timer metoder som gör det enkelt att starta, stoppa och återställa tidtagningen.

Anders Fjeldstad
Peter Asplund
MT1A

Huvudklassen Amroj innehåller main()-metoden och styr programflödet. Klassen använder sig direkt eller indirekt av samtliga andra klasser. Genom att skapa en instans av Amroj startas spelet. I denna klass ligger också hela det grafiska gränssnittet. Detta kan tyckas rörigt, och det är det också. Mer om detta finns att läsa bland gruppmedlemmarnas egna kommentarer.

Gruppmedlemmarnas kommentarer

Denna del av rapporten är till för att gruppens medlemmar individuellt ska kunna berätta om den del av projektet de arbetat med. Vi gjorde naturligtvis vårt bästa för att dela upp arbetsuppgifterna så jämnt som möjligt, även om det inte varit helt lätt.

Anders Fjeldstad
Ca. 70 arbetstimmar (har ej räknat)
Zone.java Board.java Misc.java Amroj.java
Jag har programmerat litet grand tidigare, och även om jag inte arbetat med Java hade jag stor fördel av detta under projektets förlopp. När vi utvecklade den textbaserade testversionen av spelet stötte jag inte på några större svårigheter. Det mest avancerade jag var ansvarig för är själva sweepmetoden (clear() i Board.java). Jag fick dock en idé ganska omgående om hur jag skulle lösa uppgiften, och det visade sig fungera bra. Jag valde att använda rekursion för denna metod, då det är smidigt och inte kan skapa några obehagliga bieffekter i den här miljön. Den stora utmaningen för mig var att utveckla det grafiska gränssnittet. Då jag inte arbetat med något liknande tidigare var allt nytt för mig, och jag läste ganska mycket innan jag började med programmeringen. Trots detta blev det mycket fipplande fram och tillbaka innan allt fungerade som det skulle, vilket hade en negativ effekt. Jag hade nämligen skrivit det mesta av det grafiska i klassen Amroj:s konstruktor, och när jag var klar var det för mycket kod för att jag skulle orka dela upp den i olika klasser/metoder. Observera dock att programmet fungerar precis som det ska som det är nu, det är bara det att jag är missnöjd med utseendet i klassen Amroj. Jag skulle göra det annorlunda om jag deltog i ett till, liknande projekt, nu när jag har mer kunskap. Missförstå mig inte dock, jag är mycket nöjd med min arbetsinsats. Jag har åstadkommit flera moment jag inte hade någon aning om hur jag skulle klara från början, och jag har lärt mig många nyttiga saker om Java och objektorienterad programmering i allmänhet.

Peter Asplund
Ca. 70 arbetstimmar (har ej räknat)
Minelayer.java Game_Timer.java Highscore.java Highscore_Input.java
Min programmeringsbakgrund är ganska tunn, men lite erfarenhet har jag från gymnasiet. Där gjorde jag ett sorts "Maskenspel" i C++. Jag hade dock aldrig programmerat Java tidigare så det krävdes en hel del för att ro projektet i hamn. Det krångligaste (som vi trodde skulle vara enklast) var highscorelistan, det var väldigt mycket mer att tänka på än jag trodde från början. Det löste sig dock efter lite arbete och svett, på ett sätt som jag dessutom tycker är överskådligt och snyggt kodat. Vi använde oss mycket av objekt inuti arrayer, vilket jag upptäckte har en otrolig potential, utan denna möjlighet vet jag knappt hur vi hade lyckats knyta ihop säcken. Swing var inget jag visste något om, så jag försökte först lösa klockfunktionen med en egenkodad Thread, detta var dock väldigt krångligt. Det var då vi upptäckte att det redan fanns en timerfunktion i Swingbiblioteket som automatiskt kördes i en egen thread. Jag är definitivt nöjd med vad vi skapat på bara några dagars effektivt arbete. När man väl satte sig ned och knackade ned något man pratat om och lagt upp i förväg gick det snabbt. Man märkte verkligen att svårigheten är att lära sig tänka i programmeringsbanor, inte själva kodandet. Det får man upp ångan ganska bra på efter ett tag. Ofta så satte vi oss långt ifrån datorerna och skrev ned klasser och funktioner i pseudokod. Vi skrev sedan denna kod på papper, för att sedan när man kom hem, skriva in den i programmet. Då var allt redan klart utom själva kompileringen och testandet.

Appendix A : Källkoden

Klasserna står I bokstavsordning enligt nedan, klasshuvuden i fetstil.

Amroj
Board
Game_Timer
Highscore
Highscore_Input
Minelayer
Misc
Zone

```
/*
* Classname:     Amroj (The main class)
* Properties:    Contains the GUI and handles user interactive events
* Superclass:    javax.swing.JFrame
* Author:        Anders Fjeldstad
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Amroj extends JFrame
{
    private JMenuBar menu;
    private JMenu file, difficulty, help;
    private JMenuItem file_new_game, file_highscores, file_exit, diff_pussy, diff_private, diff_daredevil, help_about;
    private JPanel panel;               //This panel will contain the actual minefield
    private Container contentPane;      //The content pane, where all graphic components will be placed
    private JButton reset;              //A reset button, restarts the game
    private JPanel top;                 //The panel containing the score-, mines-, and time-panels
    private JPanel score_display;       //The panel containing the score-label
    private JPanel mines_display;       //The panel containing the number-of-mines-label
    private JLabel final_score;         //Graphical representation of the final score
    private JLabel num_mines;           //Graphical representation of the number of mines
    private Highscore highscores;       //The highscore class
    private Game_Timer game_time;       //The timer class
    private Menu_Handler menu_actions;  //The action-handler responsible for dealing with menu actions
    private Board field;                //This is the internal representation of the minefield
    private int x_size = 15;            //The x-size of the minefield (Default setting: 15)
    private int y_size = 15;            //The y-size of the minefield (Default setting: 15)
    private int mines = 45;             //The number of mines (Default setting: 45)
    private boolean game_running = false; //False when the game has not started, true when the user has clicked the first time

    /*
    * Methodname:    Amroj() (Constructor)
    * Purpose:       The default constructor of the main class, this is where the flow of the program is controlled
    *                Ok, I know this constructor is ugly as hell, but there is actually a reason for this.
    *                When I started developing the GUI, I had never worked with Swing or AWT before, which made the progress
    *                slow with lots of "try-this-try-that":s. I ended up with this heavy (but working) constructor.
    *                My apologies, I will make it look better the next time =)
    * Parameters:    None
    * Returns:       An object of type Amroj
    */
    public Amroj()
    {
        //First-time initiations
        contentPane = getContentPane();
        menu_actions = new Menu_Handler();
        panel = new JPanel();
        reset = new JButton("Reset (F2)");

        field = new Board(x_size, y_size, mines);          //Instanciate the minefield (internal representation)
        field.init_board();                                //Initiate the minefield
        game_time = new Game_Timer();
```

```
highscores = new Highscore(this);

panel.addMouseListener(new Mouse_Handler());       //This MouseAdapter handles mouse events on the minefield
reset.addActionListener(new Button_Handler());     //This ActionListener handles events generated by the reset-button

//Some settings for the main frame
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        //The application exits when the user closes the window
setLocation(200,200);                                  //The default location of the window
setResizable(false);                                   //Window not resizable
setTitle("AMRÖJ : Ammunitionsröjning");                //The only two swedish words in the whole program =)
setIconImage(new ImageIcon("sys_icon.gif").getImage());    //Set the app-icon to our beautiful 32x32 pixel mine

//Instanciate the menu
menu = new JMenuBar();
file = new JMenu("File");
difficulty = new JMenu("Difficulty");
help = new JMenu("Help");
file_new_game = new JMenuItem("New game");
file_highscores = new JMenuItem("Highscores");
file_exit = new JMenuItem("Exit");
diff_pussy = new JMenuItem("Pussy");
diff_private = new JMenuItem("Private");
diff_daredevil = new JMenuItem("Daredevil");
help_about = new JMenuItem("About");

//Set mnemonics and accelerators for the menu
file.setMnemonic('F');
difficulty.setMnemonic('D');
help.setMnemonic('H');
file_new_game.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F2, 0));
file_exit.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_MASK));
diff_pussy.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, InputEvent.CTRL_MASK));
diff_private.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, InputEvent.CTRL_MASK));
diff_daredevil.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_3, InputEvent.CTRL_MASK));

//Build the menubar
file.add(file_new_game);
file.add(file_highscores);
file.addSeparator();
file.add(file_exit);
difficulty.add(diff_pussy);
difficulty.add(diff_private);
difficulty.add(diff_daredevil);
help.add(help_about);
menu.add(file);
menu.add(difficulty);
menu.add(help);

//Add actionlisteners for the menu buttons
file_new_game.addActionListener(menu_actions);
file_highscores.addActionListener(menu_actions);
file_exit.addActionListener(menu_actions);
diff_pussy.addActionListener(menu_actions);
diff_private.addActionListener(menu_actions);
diff_daredevil.addActionListener(menu_actions);
help_about.addActionListener(menu_actions);

setJMenuBar(menu);

//Construct the top section
top = new JPanel();
score_display = new JPanel();
mines_display = new JPanel();
final_score = new JLabel();
num_mines = new JLabel();
score_display.setBackground(Color.white);
mines_display.setBackground(Color.white);
score_display.setBorder(BorderFactory.createTitledBorder("Score"));
```

```
        mines_display.setBorder(BorderFactory.createTitledBorder("Mines"));
        top.setLayout(new GridLayout(1, 3));
        score_display.setLayout(new BoxLayout(score_display, BoxLayout.X_AXIS));
        mines_display.setLayout(new BoxLayout(mines_display, BoxLayout.X_AXIS));
        final_score.setText("");    //The score label should not display the score until the game is completed
        final_score.setAlignmentX(Container.RIGHT_ALIGNMENT);
        num_mines.setText(String.valueOf(field.get_mines()));
        score_display.add(final_score);
        mines_display.add(num_mines);
        top.add(game_time.get_clock());     //Get the clock panel via the Game_Timer.get_clock()-method
        top.add(mines_display);
        top.add(score_display);

        //Construct the main panel : the minefield
        contentPane.setLayout(new BorderLayout());
        panel.setLayout(new GridLayout(field.get_y_size(), field.get_x_size()));
        init_graphic_minefield();                       //Each call to this method redraws the minefield

        contentPane.add(top, BorderLayout.NORTH);
        contentPane.add(panel, BorderLayout.CENTER);
        contentPane.add(reset, BorderLayout.SOUTH);

        pack();                 //Sets the window size to dimensions that are sums of all the components preferred sizes
        setVisible(true);
    }

    /*
    * Methodname:    main()
    * Purpose:       Starts the game by creating an instance of the Amroj class
    * Parameters: args (An array of command-line arguments
    * Returns:       None
    */
    public static void main(String[] args)
    {
        new Amroj();
    }

    /*
    * Methodname:    init_graphic_minefield()
    * Purpose:       Adds the labels of the minefield squares (Board.minefield) to the graphic representation
    * Parameters: None
    * Returns:       None
    */
    private void init_graphic_minefield()
    {
        for (int n = 0; n < field.get_y_size(); n++)
        {
            for (int m = 0; m < field.get_x_size(); m++)
            {
                panel.add(field.get_minefield()[m][n].get_label());
            }
        }
    }

    /*
    * Methodname:    reset_game()
    * Purpose:       Contains actions to be performed when the user restarts the game
    * Parameters: None
    * Returns:       None
    */
    private void reset_game(int new_x_size, int new_y_size, int new_num_mines)
    {
        game_running = false;

        if (panel.getMouseListeners().length == 0)
            panel.addMouseListener(new Mouse_Handler());

        contentPane.remove(panel);      //Remove the main panel from the frame
```

```
        panel.removeAll();              //Remove all contents (squares) from the main panel

        field = new Board(new_x_size, new_y_size, new_num_mines);  //Create a new minefield
        field.init_board();

        panel.setLayout(new GridLayout(field.get_y_size(), field.get_x_size()));
        num_mines.setText(String.valueOf(field.get_mines()));

        init_graphic_minefield();
        contentPane.add(panel, BorderLayout.CENTER);
        pack();

        game_time.reset();              //Reset the time
        final_score.setText("");    //Clear the score-label
    }

    /*
    * Methodname:    game_completed()
    * Purpose:      Contains actions to be performed when the game is completed successfully
    * Parameters: None
    * Returns:      None
    */
    private void game_completed()
    {
        game_time.stop();     //Stop the time
        panel.removeMouseListener(panel.getMouseListeners()[0]); //Prevent the minefield from registering further mouseclicks
        int score = Misc.calculate_points(field.get_mines(), (field.get_x_size() + 1) * (field.get_y_size() + 1),
        game_time.get_time());
        final_score.setText(String.valueOf(score)); //Display the final score

        if (highscores.good_enough(score))      //Check whether the score is good enough for the highscore list
        {
            //Get input, update highscores
            String name = JOptionPane.showInputDialog("Congratulations!\n" +
            "Your score is one of the top ten best!\n\nPlease input your name.");
            if (name == null)
                name = "Unknown";

            highscores.new_highscore(new Highscore_Input(name, score));
        }
        display_highscores();//Show the highscore list
    }

    /*
    * Methodname:    game_over()
    * Purpose:      Contains actions to be performed when the user has stepped on a mine
    * Parameters: None
    * Returns:      None
    */
    private void game_over()
    {
        game_time.stop();     //Stop the time
        panel.removeMouseListener(panel.getMouseListeners()[0]); //Prevent the minefield from registering further mouseclicks
        field.clear_all();    //Clear the whole minefield (show where the mines were)
    }

    /*
    * Methodname:    display_highscores()
    * Purpose:      Displays the highscore list as a centered dialog above the main window
    * Parameters: None
    * Returns:      None
    */
    private void display_highscores()
    {
        highscores.get_highscores(this.getX(), this.getY(), this.getWidth(), this.getHeight()).setVisible(true);
    }

    /*********************************************************************************************
```

```
    *
    * Classname: Mouse_Handler (Internal class to Amroj)
    * Properties:  Handles mouse clicks on the main minefield-panel
    * Superclass: java.awt.event.MouseAdapter
    * Author:       Anders Fjeldstad
    *
    ************************************************************************************************/
    private class Mouse_Handler extends MouseAdapter
    {
        public void mouseReleased(MouseEvent e)
        {
            if (e.getButton() == e.BUTTON1)         //Left mouse button was clicked
            {
                if (game_running == false)        //Start the game if it is not running already
                {
                    game_running = true;
                    game_time.start();
                }
                switch(field.clear_zone(Misc.calc_x_zone(e.getX()), Misc.calc_y_zone(e.getY()))) //Clear the clicked zone
                {
                    case 0:
                        //Continue playing
                        break;
                    case -1:
                        //GAME OVER
                        game_over();
                        break;
                    case 1:
                        //GAME COMPLETED
                        game_completed();
                        break;
                }
            }
            if (e.getButton() == e.BUTTON2)         //Middle mouse button was clicked
                ;
            if (e.getButton() == e.BUTTON3)         //Right mouse button was clicked
            {
                if (game_running == false)        //Start the game if it is not running already
                {
                    game_running = true;
                    game_time.start();
                }
                field.mark_zone(Misc.calc_x_zone(e.getX()), Misc.calc_y_zone(e.getY())); //Mark/unmark the clicked zone
            }
        }
    }

    /************************************************************************************************
    *
    * Classname: Button_Handler (Internal class to Amroj)
    * Properties:  Handles buttonclicks (on the button Reset only)
    * Superclass: - (Implements interface ActionListener)
    * Author:       Anders Fjeldstad
    *
    ************************************************************************************************/
    private class Button_Handler implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if (e.getSource() == reset)
            {
                //Reset game
                reset_game(field.get_x_size(), field.get_y_size(), field.get_mines());
            }
        }
    }

    /************************************************************************************************
```

```
    *
    * Classname: Menu_Handler (Internal class to Amroj)
    * Properties:  Handles events generated by the menu of the frame
    * Superclass: - (Implements interface ActionListener)
    * Author:       Anders Fjeldstad
    *
    ********************************************************************************************/
    private class Menu_Handler implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if (e.getSource() == file_new_game)
                reset_game(field.get_x_size(), field.get_y_size(), field.get_mines());//Reset the game
            if (e.getSource() == file_highscores)
                display_highscores();//Show highscores
            if (e.getSource() == file_exit)
                System.exit(0);             //Quit the application
            if (e.getSource() == diff_pussy)
                reset_game(10, 10, 20); //Change the level of difficulty, start new game
            if (e.getSource() == diff_private)
                reset_game(15, 15, 45); //Change the level of difficulty, start new game
            if (e.getSource() == diff_daredevil)
                reset_game(20, 20, 80); //Change the level of difficulty, start new game
            if (e.getSource() == help_about)
                //Show about-popup
                JOptionPane.showMessageDialog(panel, "AMRÖJ version 1.1\nPeter Asplund\nAnders Fjeldstad",
                "About AMRÖJ", JOptionPane.PLAIN_MESSAGE, new ImageIcon("sys_icon.gif"));
        }
    }
}


/*
* Classname:    Board
* Properties:   Contains the actual battlefield plus methods to manipulate it during gameplay
* Superclass:   Object
* Author:       Anders Fjeldstad
*/

import javax.swing.*;

public class Board
{
    private Zone[][] minefield;         //A matrix of Zone:s representing the minefield
    private ImageIcon[] pics;     //This is an array containing the images for the grapical representation of the minefield
    private int mines = 0;          //The number of mines
    private int cleared_zones = 0;    //The number of cleared squares, used to determine whether the game is completed or not

    public Board(int x_size, int y_size, int number_of_mines)
    {
        minefield = new Zone[x_size][y_size];
        mines = number_of_mines;
        pics = new ImageIcon[12];
    }

    public void init_board()
    {
        Misc.load_pics(pics);                       //Load the images into the 'pics'-array
        Minelayer.init_array(minefield, pics);          //Initiate the minefield array
        Minelayer.mineplacer(mines, minefield);         //Randomly place the mines on the minefield
        Minelayer.check_neighbours(minefield);          //Tell each square what number to show when cleared
        cleared_zones = 0;
    }

    /*
    * Methodname:    mark_zone()
    * Purpose:      Marks the given square, or unmarks if the square is marked already
    * Parameters: x_pos, y_pos (coordinates of the zone to be cleared)
```

```
    * Returns:      -
    */
    public void mark_zone(int x_pos, int y_pos)
    {
        if (!minefield[x_pos][y_pos].is_cleared())
        {
            if (minefield[x_pos][y_pos].is_marked())
                minefield[x_pos][y_pos].unmark();
            else
                minefield[x_pos][y_pos].mark();
        }
    }

    /*
    * Methodname:    clear_zone()
    * Purpose:      Clears a zone with given coordinates, and in turn clears ajacent zones if the first square has no nearby mines
    * Parameters: x_pos, y_pos (coordinates of the zone to be cleared)
    * Returns:      0 when game is to continue, 1 when the game is successfully completed and -1 when the game is over
    */
    public int clear_zone(int x_pos, int y_pos)
    {
        if (!minefield[x_pos][y_pos].is_marked() && minefield[x_pos][y_pos].is_mined())
        {
            return -1;  //The player stepped on a mine: Game over
        }
        if (!minefield[x_pos][y_pos].is_cleared() && !minefield[x_pos][y_pos].is_marked())
        {
            minefield[x_pos][y_pos].clear();                    //Clear the specified zone
            cleared_zones++;                                        //Increase the cleared_zones counter

            if (minefield[x_pos][y_pos].nearby_mines() == 0)
            {
                //Clear nearby zones if the zone has no nearby mines
                for (int i = -1; i <= 1; i++)
                {
                    for (int j = -1; j <= 1; j++)
                    {
                        if ((x_pos + j >= 0) && (x_pos + j < minefield.length) && (y_pos + i >= 0) && (y_pos + i <
                        minefield[0].length))
                        {
                            if (!minefield[x_pos + j][y_pos + i].is_cleared())
                                clear_zone(x_pos + j, y_pos + i);
                        }
                    }
                }
            }
        }
        if (minefield.length * minefield[0].length - cleared_zones - mines == 0)
        {
            return 1;   //Remaining uncleared zones equals the number of mines: the player has completed the game
        }
        else
            return 0;   //Continue the game
    }

    /*
    * Methodname:    clear_all()
    * Purpose:      Clears the whole minefield, used when the user has lost the game to show where the mines were
    * Parameters: none
    * Returns:      none
    */

    public void clear_all()
    {
        for (int i = 0; i < minefield[0].length; i++)
        {
            for (int j = 0; j < minefield.length; j++)
            {
```

```
                minefield[j][i].clear();
            }
        }
    }

    public Zone[][] get_minefield()
    {
        return minefield;
    }

    public int get_mines()
    {
        return mines;
    }

    public int get_x_size()
    {
        return minefield.length;
    }

    public int get_y_size()
    {
        return minefield[0].length;
    }
}

*
* Name:        Game_Timer
* Properties: The timer in the game, counts from 0 onward until stopped by game. Also contains graphic information
*              about the clock.
* Superclass: Object
* Author:      Peter Asplund

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Game_Timer
{
    Timer timer;                    //Intiate the Timer
    JPanel clock;
    JLabel counter;                 //The visual clock
    int current_time = 0;       //Starts at zero

    /*
    * Methodname:   Game_Timer() (Constructor)
    * Purpose:          The constructor of the Game_Timer class. It creates the clock
    * Parameters: none
    * Returns:          An object of type Game_Time
    */

    public Game_Timer()
    {
        timer = new Timer(1000, new count()); //1000 milliseconds between every call to count()
        clock = new JPanel();
        counter = new JLabel();
        clock.setBorder(BorderFactory.createTitledBorder("Time"));
        clock.setLayout(new GridLayout(1, 1));
        clock.setBackground(Color.white);
        counter.setText("0");
        counter.setBackground(Color.white);
        clock.add(counter);
    }

    /*****************************************************************************************
    *
    * Classname:      count()
    * Purpose:          Recieves an Action from the timer and adds 1 to the current_time variable
```

```
    * Parameters: s
    * Returns:        none
    *
    ********************************************************************************************/

    private class count implements ActionListener
    {
        public void actionPerformed(ActionEvent s)
        {
            counter.setText(String.valueOf(current_time)); //Sets the text of the clock
            current_time++;
        }
    }

    /*
    * Methodname:   start()
    * Purpose:      The method to start the timer
    * Parameters: none
    * Returns:      none
    */

    public void start()
    {
        counter.setText(String.valueOf(current_time));
        current_time++;
        timer.start();
    }

    /*
    * Methodname:   stop()
    * Purpose:      The method to stop the timer
    * Parameters: none
    * Returns:      none
    */

    public void stop()
    {
        timer.stop();
    }

    /*
    * Methodname:   reset()
    * Purpose:      Resets the timer to zero and sets the clocktext to 0
    * Parameters: none
    * Returns:      none
    */

    public void reset()
    {
        if (timer.isRunning())
            timer.stop();

        current_time = 0;
        counter.setText(String.valueOf(current_time));
    }

    /*
    * Methodname:   get_time()
    * Purpose:      The public method to return the current_time to other classes
    * Parameters: none
    * Returns:      current_time int variable
    */

    public int get_time()
    {
        return current_time;
    }
```

```
    /*
    * Methodname:   get_clock()
    * Purpose:      The public method to return the finished JPanel clock with graphic and information
    * Parameters: none
    * Returns:      none
    */

    public JPanel get_clock()
    {
        return clock;
    }
}

*
*Classname: Highscore
*Properties:  Handles the highscore. Takes input, creates an array containing objects of Highscore_Input type, reads
*             from file, arranges highscores in order, prints to file.
*Superclass: Object
*Author:      Peter Asplund
*

import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Highscore
{
    Highscore_Input[] highscores = new Highscore_Input[10]; //A matrix of Highscore_Input:s containing the Highscore
    information
    private String dirname = ".";                                   //The directory of the highscore file
    private String filename = "highscore.txt";                      //The name of the highscore file
    private File highscore_file = new File(dirname, filename); //The object highscore_file
    private PrintWriter highscore_stream;                           //The highscore_stream buffer, used to print all of the
    information at once
    private BufferedReader old_highscore_stream;                    //The old_highscore_stream buffer, used to read all the
    highscore information at once

    //Initiating the Swing components
    private JPanel names;
    private JPanel scores;
    private JDialog highscore_dialog;
    private JLabel[] name_labels;
    private JLabel[] score_labels;

    /*
    * Methodname:   Highscore (Constructor)
    * Purpose:      The constructor of the Highscore class.
    * Parameters: none
    * Returns:      An object of type Highscore
    */

    public Highscore(JFrame my_frame)
    {
        names = new JPanel();
        scores = new JPanel();
        highscore_dialog = new JDialog(my_frame, "Highscores", true);          //Creates a new visible frame
        name_labels = new JLabel[10];
        score_labels = new JLabel[10];
        highscore_dialog.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);    //Tells the frame what to do when being
        closed
        highscore_dialog.setResizable(false);                                       //Not resizable
        highscore_dialog.getContentPane().setLayout(new GridLayout(1, 2));      //Creates a grid to place JLabel in
        highscore_dialog.getContentPane().setBackground(Color.magenta);
        names.setLayout(new BoxLayout(names, BoxLayout.Y_AXIS));                //Creates a boxlayout inside the grid
        scores.setLayout(new BoxLayout(scores, BoxLayout.Y_AXIS));
        for (int i = 0; i < 10; i++)
        {
```

```
            name_labels[i] = new JLabel();
            score_labels[i] = new JLabel();
        }
    }

    /*
    * Methodname:   init_label_arrays()
    * Purpose:      Initiates the Labelarrays name_labels and score_labels
    * Parameters: none
    * Returns:      none
    */

    private void init_label_arrays()
    {
        for (int i = 0; i < 10; i++)
        {
            name_labels[i].setText(highscores[i].get_name());      //Sets the name to the one in the highscores[i] object.
            name_labels[i].setAlignmentX(Component.LEFT_ALIGNMENT);
            score_labels[i].setText(String.valueOf(highscores[i].get_score()));    //Sets the score to the one in the highscores[i]
                                                                                   //object
            score_labels[i].setAlignmentX(Component.RIGHT_ALIGNMENT);
        }
    }

    /*
    * Methodname:   init_filewrite()
    * Purpose:      Makes it possible to write to a file
    * Parameters: none
    * Returns:      none
    */

    private void init_filewrite()
    {
        try
        {
            if (!highscore_file.canWrite())                 //Performs action if file is not writeable
                highscore_file.createNewFile();       //Creates new file

            highscore_stream = new PrintWriter(new BufferedWriter(new FileWriter(highscore_file)));//Creates highscore_stream
                                                                                                   //containing the information
                                                                                                   //where the highscore file is
                                                                                                   //and how to write to it
            for(int i = 0; i < 10; i++)         //Prints the scores and names inside the highscores array
            {
                highscore_stream.println(highscores[i].get_name());
                highscore_stream.println(highscores[i].get_score());
            }
            highscore_stream.close(); //Closes and saves the file
        }
        catch (IOException s)               //If the file can't be created or written to
        {
            System.out.println("FileCreationError.sys.admin.delete-hdd.w00t.b00b");
        }
    }

    /*
    * Methodname:   init_fileread()
    * Purpose:      Makes it possible to read a file
    * Parameters: none
    * Returns:      none
    */

    private void init_fileread()
    {
        try
        {
            if (!highscore_file.canWrite())                 //Performs the action of the file does not exist or can't be written to
                highscore_file.createNewFile();       //Creates new file
```

```
            old_highscore_stream = new BufferedReader(new FileReader(highscore_file)); //Creates the old_highscore_stream
buffer to hold the old information in the highscore file
        }
        catch (IOException s)
        {
            System.exit(1);             //Exit the program if file can't be created or written to
        }
    }

    /*
    * Methodname:   read_highscore()
    * Purpose:      Reads the highscore file and inserts the information in the highscore_file array
    * Parameters: none
    * Returns:      none
    */

    private void read_highscore()
    {
        try
        {
            for (int i = 0; i < 10; i++)
            {
                String temp_name = old_highscore_stream.readLine();   //Reads the line and saves it in temp_name
                if (temp_name == null)                                //If the name does not exist, write Empty instead of null
                    temp_name = "Empty";
                String p00p = old_highscore_stream.readLine();  //Reads the score as a string and saves it in the p00p string
                int temp_score = 0;                             //If the score does not exist, save as zero
                if (p00p != null)                               //If the highscore exist, save as int inside temp_score variable
                    temp_score = Integer.parseInt(p00p);

                highscores[i] = new Highscore_Input(temp_name, temp_score);      //Initiate highscore array
            }
            old_highscore_stream.close();       //Close the file
        }
        catch(IOException s)
        {
            System.out.println("Funkar int");
            System.exit(1);                         //Exit the program if the file unreadable
        }
    }

    /*
    * Methodname:   organize_highscore()
    * Purpose:      Organizes the highscore inputs in order. Starts at position 0, then organize the highest value to position 0.
Then moves on and checks from position 1 and onward, and saves
                    the highest value of the remaining positions in position 1, and so on.
    * Parameters: none
    * Returns:      none
    */

    private void organize_highscore()
    {
        Highscore_Input temp;    //Temporary variable of type Highscore_Input when 'moving' values
        for (int i = 0; i < 10; i++)  //Scans through the array
        {
            int temp_position = i;      //Saves the position of the highest value
            for (int n = i; n < 10; n++)
            {
                if (highscores[temp_position].get_score() < highscores[n].get_score())       //Compares the highest value so far to
the value in position n
                    temp_position = n;                                                                  //If n is larger, save
the position in temp_position
            }
            temp = highscores[i];                                                                   //Temporary store the
information of highscores[i] inside temp object
            highscores[i] = highscores[temp_position];                                          //Moves the new higher value
to the position of the old highest value
```

```
            highscores[temp_position] = temp;                                   //Store the old value in the
old position of the new highest value
        }
    }

    /*
    * Methodname:   good_enough()
    * Purpose:      Checks wether the player has a good enough score to enter the highscore
    * Parameters: points
    * Returns:      a boolean (true = good enough, false = not good enough)
    */

    public boolean good_enough(int points)
    {
        init_fileread();                        //Reads the file
        read_highscore();                       //Reads the highscore array
        organize_highscore();                   //Organize the array (so nobody can mess around with it while running the
program)
        if (points > highscores[9].get_score())   //If the user is good enough, return true
            return true;
        else
            return false;
    }

    /*
    * Methodname:   new_highscore()
    * Purpose:      Inserts the new highscore in the array in position 9
    * Parameters: none
    * Returns:      none
    */

    public void new_highscore(Highscore_Input new_input)
    {
        init_fileread();
        organize_highscore();               //Organize highscore (So noboy has messed with it during gameplay)
        highscores[9] = new_input;          //Save his/hers score in position 9 (It is always the last one in the highscore that will
drop out
        organize_highscore();               //Organize highscore (So noboy has messed with it during gameplay)
        init_filewrite();                   //Write to the highscore file
    }

    /*
    * Methodname:   get_highscores()
    * Purpose:      Returns the JDialog to other classes needing the information
    * Parameters: none
    * Returns:      A get_highscores object of JDialog type
    */

    public JDialog get_highscores(int x, int y, int owner_width, int owner_height)
    {
        init_fileread();                    //Initiates the file
        read_highscore();                   //Reads the file
        organize_highscore();               //Organizes the highscore array
        init_label_arrays();                //Initiates the label arrays
        for (int i = 0; i < 10; i++)
        {
            names.add(name_labels[i]);
            scores.add(score_labels[i]);
        }
        highscore_dialog.getContentPane().add(names);
        highscore_dialog.getContentPane().add(scores);
        highscore_dialog.pack();            //Fixes the design of the JDialog
        highscore_dialog.setLocation(x + (owner_width / 2) - (highscore_dialog.getWidth() / 2), y + (owner_height / 2) -
(highscore_dialog.getHeight() / 2));
        return highscore_dialog;            //Returns the object
    }
}
```

```
* Name:        Highscore
* Properties: Object containing highscore input information like name and score
* Superclass:Object
* Author:      Peter Asplund
*
**********************************************************************************************/

public class Highscore_Input
{
    private int score;
    private String name;

    public Highscore_Input(String new_name, int new_score)
    {
        score = new_score;
        name = new_name;
    }
/*
* Name:             get_score()
* Properties:   Returns the score of the object as a int
* Parameters:none
* Returns:      Int score
*/

    public int get_score()
    {
        return score;
    }

/*
* Name:             Highscore()
* Properties:   Returns the name of the person as a string
* Parameters:none
* Returns:      String name
*/

    public String get_name()
    {
        return name;
    }
}

* Name:             Minelayer
* Properties:   Creates 2D Array of type Zone and randomly inserts mine information into the array objects, wether it is mined or
not and how many adjacent mines it has.
* Superclass:  Object
* Author:       Peter Asplund
*/

import javax.swing.*;

public class Minelayer
{
    /*
    * Methodname:   init_array()
    * Purpose:          Initiate the array x and creates an instant of the object Zone in every arrayposition
    * Parameters: images
    * Returns:          none
    */

    public static void init_array(Zone[][] x, ImageIcon[] images)
    {
        for (int y_pos = 0; y_pos < x[0].length; y_pos++)   //Enters y-positions in array
        {
            for (int x_pos = 0; x_pos < x.length; x_pos++)  //Enters x-positions in array
            {
                x[x_pos][y_pos] = new Zone(images); //Makes an Instant of Objecttype Zone in array position (x , y)
            }
```

```
        }
    }

    /*
    * Methodname:  mineplacer()
    * Purpose:     Randomly places mines inside the minearray
    * Parameters:  images
    * Returns:     none
    */

    public static void mineplacer(int number_mines, Zone[][] x)
    {
        int x_position;
        int y_position;
        boolean work_done = false;  //Tells wether the mineplacing was succesful

        for (int i = 0; i < number_mines; i++)
        {
            work_done = false;          //No mines placed
            while (!work_done)          //Place mine as long as no mined was placed this loop
            {
                x_position = (int) (x.length * Math.random());
                y_position = (int) (x[0].length * Math.random());
                if (!x[x_position][y_position].is_mined())
                {
                    x[x_position][y_position].mine();
                    work_done = true;   //Returns that the mineplacening was succesful
                }
            }
        }
    }

    /*
    * Methodname:   check_neighbours()
    * Purpose:      Checks wether there are any adjacent mines and adds one to the nearby_mines variable inside the
zoneobject
    * Parameters: none
    * Returns:      none
    */

    public static void check_neighbours(Zone[][] x)
    {
        for (int y_pos = 0; y_pos < x[0].length; y_pos++)       // Rows
        {
            for (int x_pos = 0; x_pos < x.length; x_pos++)      //Columns
            {
                if (x[x_pos][y_pos].is_mined())
                {
                    for (int i = -1; i <= 1; i++)
                    {
                        for (int j = -1; j <= 1; j++)
                        {
                            if ((x_pos + j >= 0) && (x_pos + j < x.length) && (y_pos + i >= 0) && (y_pos + i < x[0].length))
                            {
                                x[x_pos + j][y_pos + i].add_nearby_mine(); //adds 1 to the nearby_mines
                            }
                        }
                    }
                }
            }
        }
    }
}

/*
* Classname:    Misc
* Properties:   Contains various static methods
* Superclass:   Object
```

```
* Author:           Anders Fjeldstad
*/

import javax.swing.*;

public class Misc
{
    /*
    * Methodname:    load_pics()
    * Purpose:       Loads the pictures used by the game into a given array
    * Parameters: pics (An array of type ImageIcon)
    * Returns:       -
    */
    public static void load_pics(ImageIcon[] pics)
    {
        pics[0] = new ImageIcon("cleared.gif");
        pics[1] = new ImageIcon("1.gif");
        pics[2] = new ImageIcon("2.gif");
        pics[3] = new ImageIcon("3.gif");
        pics[4] = new ImageIcon("4.gif");
        pics[5] = new ImageIcon("5.gif");
        pics[6] = new ImageIcon("6.gif");
        pics[7] = new ImageIcon("7.gif");
        pics[8] = new ImageIcon("8.gif");
        pics[9] = new ImageIcon("not_cleared.gif");
        pics[10] = new ImageIcon("marked.gif");
        pics[11] = new ImageIcon("mine.gif");
    }

    /*
    * Methodname:    calc_x_zone()
    * Purpose:       Takes the x-position of the mouse and calculates the respective square of the minefield
    * Parameters: mouse_x_pos
    * Returns:       An integer representing the x-position on the minefield
    */
    public static int calc_x_zone(int mouse_x_pos)
    {
        final int ZONE_SIDE_LENGTH = 20;

        return(mouse_x_pos / ZONE_SIDE_LENGTH);
    }

    /*
    * Methodname:    calc_y_zone()
    * Purpose:       Takes the y-position of the mouse and calculates the respective square of the minefield
    * Parameters: mouse_y_pos
    * Returns:       An integer representing the y-position on the minefield
    */
    public static int calc_y_zone(int mouse_y_pos)
    {
        final int ZONE_SIDE_LENGTH = 20;

        return(mouse_y_pos / ZONE_SIDE_LENGTH);
    }

    /*
    * Methodname:    calculate_points()
    * Purpose:       Calculates the score based on the number of mines and squares and the time passed
    * Parameters: number_mines, number_fields, time_passed
    * Returns:       An integer that represents the score
    */
    public static int calculate_points(int number_mines, int number_fields, int time_passed)
    {
        final int CONSTANT  = 500;
        int points = (CONSTANT * number_mines * number_fields)/ time_passed;
        return points;
    }
```

```
}


/*
* Classname:    Zone
* Properties:   Contains the basic information of one square of the battlefield
* Superclass:   Object
* Author:       Anders Fjeldstad
*/

import javax.swing.*;

public class Zone
{
    private boolean mined = false;        //False if the square is not mined, true if it is
    private boolean cleared = false; //False if the square is not cleared, true if it is
    private boolean marked = false;       //Flase if the square is not marked, true if it is
    private int nearby_mines = 0;         //The total number of mines on the ajacent squares
    private JLabel label;                 //The graphical representation of the square
    private ImageIcon[] pics;             //A variable containing the address to the "global" pics array

    /*
    * Methodname:   Zone() (Constructor)
    * Purpose:      Initiates and returns an object of type Zone
    * Parameters: p (An array of type ImageIcon)
    * Returns:      An object of type Zone
    */
    public Zone(ImageIcon[] p)
    {
        label = new JLabel();
        label.setHorizontalAlignment(JLabel.LEFT);
        label.setVerticalAlignment(JLabel.TOP);
        pics = p;
        label.setIcon(pics[9]);//Set the current icon to the empty, uncleared zone
    }

    /*
    * Methodname:   clear()
    * Purpose:      Clears this square, setting its label to display the correct picture
    * Parameters: -
    * Returns:      -
    */
    public void clear()
    {
        cleared = true;

        if (is_mined())
            label.setIcon(pics[11]);            //Set the icon to the mine image
        else
        {
            if (nearby_mines == 0)
                label.setIcon(pics[0]);         //Set the icon to the empty, cleared zone
            else
                label.setIcon(pics[nearby_mines]);  //Set the icon to reflect the number of nearby mines
        }
    }

    /*
    * Methodname:   add_nearby_mine()
    * Purpose:      Add 1 to the number of total mines on the ajacent squares
    * Parameters: -
    * Returns:      -
    */
    public void add_nearby_mine()
    {
        nearby_mines++;
    }
```

```
    /*
    * Methodname:    mark()
    * Purpose:      Marks this square, setting its label to display the correct picture
    * Parameters: -
    * Returns:      -
    */
    public void mark()
    {
        marked = true;
        label.setIcon(pics[10]);    //Set the icon to the marked, uncleared zone
    }

    /*
    * Methodname:    unmark()
    * Purpose:      Unmarks this square, setting its label to display the correct picture
    * Parameters: -
    * Returns:      -
    */
    public void unmark()
    {
        marked = false;
        label.setIcon(pics[9]);     //Set the icon to the empty, uncleared zone
    }

    public boolean is_mined()
    {
        return mined;
    }

    public boolean is_cleared()
    {
        return cleared;
    }

    public boolean is_marked()
    {
        return marked;
    }

    public int nearby_mines()
    {
        return nearby_mines;
    }

    public void mine()
    {
        mined = true;
    }

    public JLabel get_label()
    {
        return label;
    }
}
```