



TURUN YLIOPISTO
Tietojenkäsittelytieteet

KURSSeihin liittyvien harjoitustöiden ja laboratoriotöiden kansilehti

Opintojakso (kurssi)	Harjoitustyöprojekti II (TKO_5102)
Työn tunnus	Harjoitustyöprojekti II
Työn aihe	Game of life
Ohjaaja	Mika Murtojärvi

Tekijä(t)

Sukunimi	Etunimi	Koul.ohj.	Matr.nro	Sähköpostiosoite
Nieminen	Vesa	TKT	59533	vaniem@utu.fi

Henkilökunta täyttää:

Toimenpide	Päivämäärä	Nimikirjoitus
Työ annettu		
Suunnitelma hyväksytty		
Työ vastaanotettu		
Työ hyväksytty		

Sisällysluettelo

1.	Tehtävän kuvaus ja analysointi	3
2.	Ratkaisuperiaate	4
3.	Ohjelman ja sen osien kuvaaminen	5
3.1.	Class GameOfLife.....	5
3.2.	Class Display.....	7
3.3.	Class LifeView.....	9
3.4.	Class LifeModel.....	11
3.5.	Class LifeController.....	13
4.	Testausjärjestely.....	14
Liite A.	Tehtäväpaperi	16
Liite B.	Kommentoitu ohjelmalistaus	17
1.	GameOfLife.java.....	17
2.	Display.java.....	19
3.	LifeView.java.....	20
4.	LifeModel.java.....	23
5.	LifeController.java.....	28
Liite C.	Käyttöohje	31

1. Tehtävän kuvaus ja analysointi

Tehtävänä oli toteuttaa Game of Life Java applettina. Tehtävänannossa lukee, että maailman tulisi olla äärettömän kokoinen, mutta en keksinyt miten tuollaisen oikein ohjelmoisi joten pitäydyin tavanomaisemmassa tavassa, eli tein maailmasta yhtenäisen. Tarkoittoaen, että maailman toisesta laidasta pääsee toiselle, eli vasemman reunan ylämentäessä tullaan oikeaan reunaan jne. Alunperin päätin tehdä ohjelmasta vahvasti olio-ohjelmoinnin pääperiaatteiden mukaisen käyttämällä MVC (model-view-controller) lähestymistapaa, jotta jälkepäin olisi helppoa kääntää ohjelma eri kielille/alustoille.

Syöttötietoina ohjelmassa käytetään hiiren vasenta ja oikeaa nappia. Vasemmalla napilla piirrellään uusia soluja ruudulle ja oikealla pysäytetään peli. Pitäydyin näinkin minimalistisessa ratkaisussa, koska se tuntui mielestäni varsin pätevältä. Jälkepäin on kuitenkin todella helppoa lisätä haluttuja ominaisuuksia, kuten vaikkapa pelinopeuden säätöä tai erilaisten статистиikkojen näyttöä.

Ohjelma vaatii toimiakseen Java SDK 5.0:n. Voi olla, että toimii vanhemmillakin versioilla, mutten erikseen ole testannut niitä.

2. Ratkaisuperiaate

Gameoflife.jpg tiedostossa on Javan työkaluilla generoitu luokkakaavio ohjelmastani.

Tein ohjelman luokkakohtaisen suunnittelun useita kuukausia sitten, mutta tenttien sun muiden harjoitustöiden ja kiireiden takia en ole kerennyt toteuttamaan tätä työtä kuin nyt vasta aivan viime hetkellä. Suunnittelun kuitenkin ollessa valmiina oli työhön verrattain helppo ryhtyä, eikä itse ongelmakaan mikään hirveän monimutkainen ollut.

Ratkaisussani on viisi luokkaa: GameOfLife, Display, LifeView, LifeModel ja LifeController. Heti alun alkaen päätin, että teen ohjelman hyvin modulaarisesti, jotta sitä on helppoa siirrellä paikasta toiseen ja muokata jälkeenpäin. Loppujen lopuksi olen varsin tyytyväinen tulokseen.

GameOfLife toimii ohjelman pääluokkana. Sen voi käynnistää joko applettina tai sovelluksena (kts. käyttöohje). Luokka sisältää yksinkertaisen pelisilmukan, jota ajetaan omassa säikeessään. Pelisilmukassa päivitetään maailman tilanne ensin mallitasolla ja sen jälkeen tulostetaan muuttunut tilanne ruudulle noin joka sekunnin kymmenys.

Display luokka tarjoaa yleiskäyttöisiä metodeja graafisuuden esittämiseen mm. tavan ajaa applettia sovelluksena käärimällä se JFrame-tyyppiseen ikkunaan.

LifeView on canvas-tyyppinen luokka, jossa toteutetaan tuplapuskuroitu (doublebuffered) mallimaailman kuvan piirto. Se toimii MVC mallin mukaisesti luonnollisesti view, eli näkymä tyyppinä.

LifeModel on itse Game of Life pelin malli. Se sisältää metodit yhden tai useamman elämänsyklin ajolle, tietyn solun naapureiden lukumäärän laskun ja tarvittavat muutosmetodit joilla voidaan mallin tilaa muuttaa.

LifeController on yksinkertainen tapahtumankäsittelijä, joka toimii ohjelman kontrollina. 3 erilaista tapahtumaa käsitellään: vasemman hiirennapin painallus, vasemman hiirennapin alhaallapitäminen ja oikean hiirennapin painallus.

3. Ohjelman ja sen osien kuvaaminen

3.1. Class GameOfLife

```
public class GameOfLife
extends javax.swing.JApplet
implements java.lang.Runnable
```

Harjoitustyöprojekti 2 course exercise. The GameOfLife is the root class and starts the game.

Constructor Detail

GameOfLife

```
public GameOfLife()
    Setup for the game.
```

Method Detail

start

```
public void start()
    Initiates the game.
Overrides:
    start in class java.applet.Applet
```

run

```
public void run()
Specified by:
    run in interface java.lang.Runnable
```

destroy

```
public void destroy()
    Shuts the game down before closing the applet.
Overrides:
    destroy in class java.applet.Applet
```

gameLoop

```
public void gameLoop()
    The gameloop. Draws the screen and update the model.
```

main

```
public static void main(java.lang.String[] args)
```

GameOfLife starts here.

Parameters:

args - Not used.

3.2. *Class Display*

```
public class Display
extends java.lang.Object
```

The Display class is a framework for displaying both Swing applications and applets easily. Based on Bruce Eckel's Console class.

Constructor Detail

Display

```
public Display()
```

Method Detail

title

```
public static java.lang.String title(java.lang.Object o)
```

Create a title string from the class name

Parameters:

o - The object whose name is going to be used.

Returns:

The retrieved String.

run

```
public static void run(javax.swing.JFrame frame,
                        int width,
                        int height)
```

Framework method for JFrame type displaying.

Parameters:

frame - The JFrame to use.

width - The desired width.

height - The desired height.

run

```
public static void run(javax.swing.JApplet applet,
                        int width,
                        int height)
```

Framework method for JApplet type displaying.

Parameters:

applet - The JApplet to use.

width - The desired width.

height - The desired height.

run

```
public static void run(javax.swing.JPanel panel,  
                       int width,  
                       int height)
```

Framework method for JPanel type displaying.

Parameters:

panel - The JPanel to use.

width - The desired width.

height - The desired height.

3.3. Class *LifeView*

```
public class LifeView
extends java.awt.Canvas
```

LifeView is used to display the Game of life.

Constructor Detail

LifeView

```
public LifeView(int width,
                 int height,
                 LifeModel model)
```

The constructor. Precondition: width / model.getWidth() should yield an even number. height / model.getHeight() should yield an even number.

Parameters:

width - The width of the canvas
height - The height of the canvas
model - The LifeModel to draw

Method Detail

draw

```
public void draw()
    Draws the state of the world on the canvas.
```

init

```
public void init()
    The needed initializations for this object. One should not use it before this method is called and this method should not be called before there exists a JFrame, JPanel or a JApplet which can be used as a peer.
```

getModel

```
public LifeModel getModel()
    Returns:
    the world model. Do not modify!
```

getCellWidth

```
public int getCellWidth()
    Returns:
    the cell width on screen
```

getCellHeight

`public int getCellHeight()`

Returns:

the cell height on screen

3.4. Class LifeModel

```
public class LifeModel
extends java.lang.Object
```

LifeModel contains the Game of life world.

Constructor Detail

LifeModel

```
public LifeModel(int width,
                 int height)
```

Creates a new Game of life type world. Precondition: width \geq 3 & height \geq 3.

Parameters:

width - The width of the world to be created.

height - The height of the world to be created.

Method Detail

changeCell

```
public void changeCell(int x,
                      int y)
    throws java.lang.ArrayIndexOutOfBoundsException
```

Changes the state of the Cell at coordinate (x,y). Alive -> dead or dead -> alive.

Parameters:

x - The x-coordinate.

y - The y-coordinate.

Throws:

java.lang.ArrayIndexOutOfBoundsException

live

```
public void live()
    Makes the world live one cycle.
```

live

```
public void live(int x)
    Makes the world live x cycles. Precondition: x > 0.
Parameters:
x - The amount of life cycles to live.
```

getWorld

```
public java.lang.Boolean[][] getWorld()
    Used to get the state of the world. One should not modify the Boolean table that is returned!
```

Returns:
the state of the Game of life world.

getWidth

public int **getWidth**()

Returns:
world width

getHeight

public int **getHeight**()

Returns:
world height

getPause

public boolean **getPause**()

Returns:
is the game paused?

switchPause

public void **switchPause**()

Switches the state of the game. Unpaused -> paused, paused -> unpaused.

3.5. Class LifeController

```
public class LifeController
extends javax.swing.event.MouseInputAdapter
```

Controlling side of the game.

Constructor Detail

LifeController

```
public LifeController(LifeView view)
```

Parameters:

view - the view to use with this controller

Method Detail

takeAction

```
public void takeAction(java.awt.event.MouseEvent e)
```

A multipurpose eventhandler. Manages 3 different mouse button presses. 1) Button1 pressed == change cell state 2) Anything but Button3 being dragged == change the states of multiple cells 3) Button3 pressed == pause the game

mousePressed

```
public void mousePressed(java.awt.event.MouseEvent e)
```

Mousepressed eventhandling

Specified by:

mousePressed in interface java.awt.event.MouseListener

Overrides:

mousePressed in class javax.swing.event.MouseInputAdapter

mouseDragged

```
public void mouseDragged(java.awt.event.MouseEvent e)
```

Mousedragged eventhandling

Specified by:

mouseDragged in interface java.awt.event.MouseMotionListener

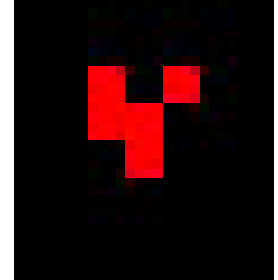
Overrides:

mouseDragged in class javax.swing.event.MouseInputAdapter

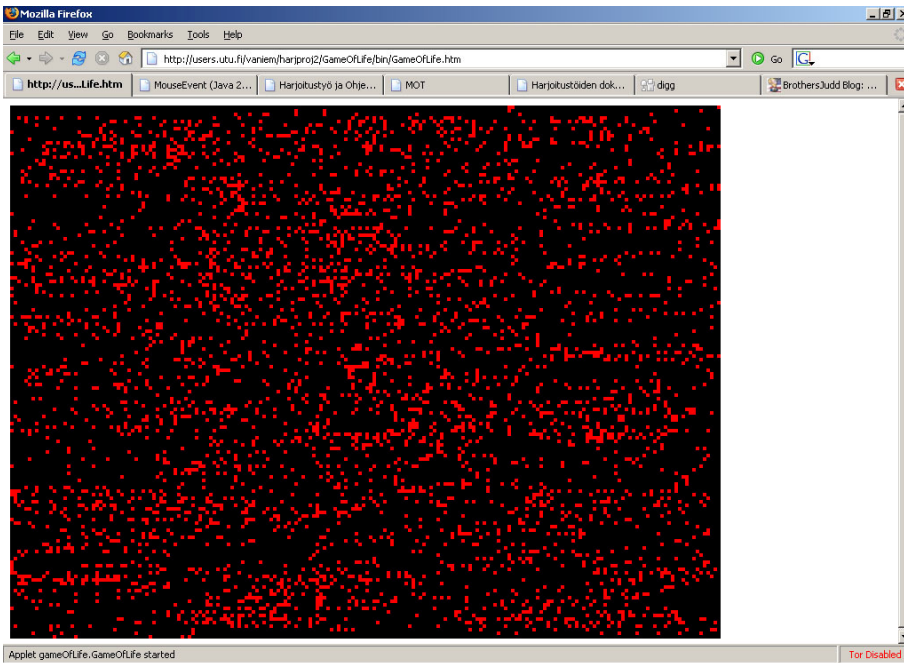
4. Testausjärjestely

Testausjärjestely toteutettiin mm. ajamalla peliä puolen tunnin ajan applettina ja sovelluksena. Erilaisia ”elämänmuotoja” testattiin Internetistä löytyvien esimerkkien opastuksella. Esimerkiksi Glider-niminen ”elämänmuoto” (kuva 1.) on viiden solun muodostama viistosti liikkuva kapistus. Kaikki kokeillut ”elämänmuodot” toimivat niin kuin piti.

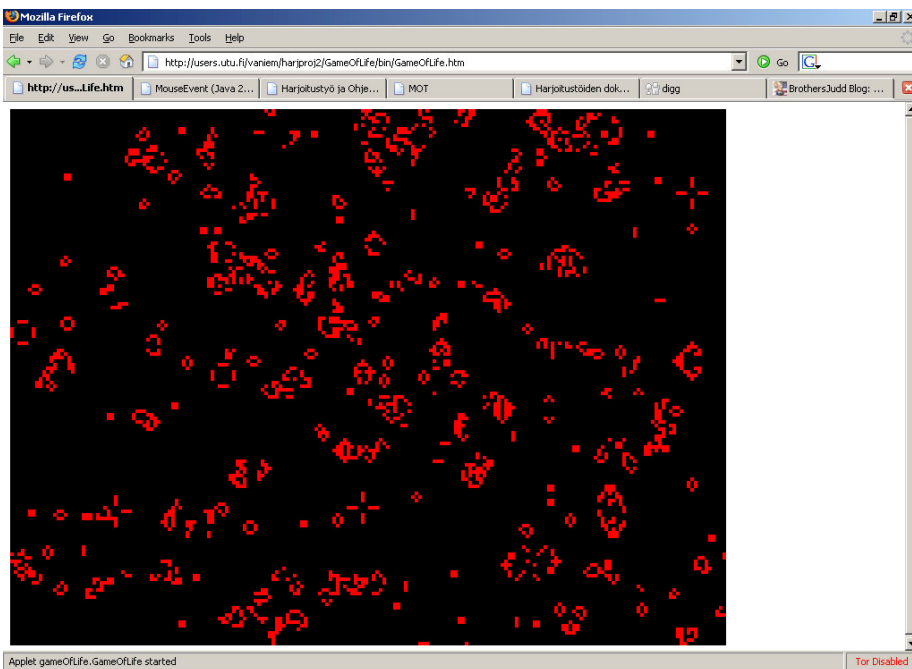
Kuva 2. näyttää kaaottisen alkutilan joka jo muutaman sekunnin jälkeen muotoutuu tavanomaisemman Game of Lifen näköiseksi (Kuva 3.). Lopulta, noin kymmenen minuutin kuluttua tilanne stabiloituu kuvan 4. mukaiseksi.



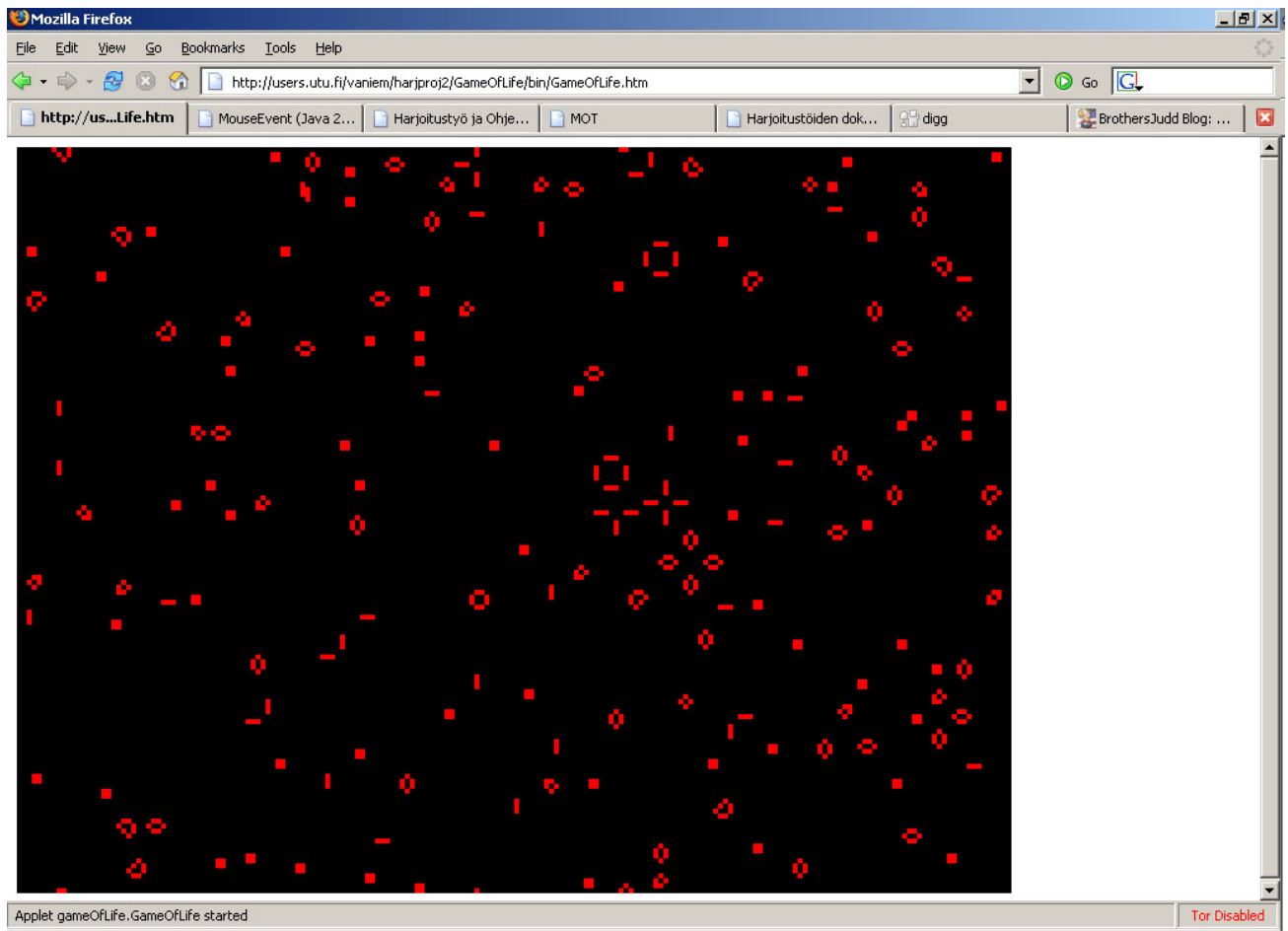
Kuva 1.



Kuva 2.



Kuva 3.



Kuva 4.

Liite A. Tehtäväpaperi

The Game Of Life on peli, joka on kiehtonut matemaatikkoja jo kolmen vuosikymmenen ajan. Life on J. Conwayn n. 1970 suunnittelema peli, jossa pelaaja antaa alkutilanteen ja peli suoriutuu sitten itse lopusta. The Game Of Life toimii ruututasolla (kuvaruudulla), joka on periaatteessa ääretön kaikkiin suuntiin (kuvaruudulla näkyy vain osa tasosta). Ruutu eli solu on kullakin hetkellä joko elossa tai kuollut (ruudun muodostamat pikselit päällä / pois päältä), ja sen olomuoto seuraavalla hetkellä määräytyy täysin kahdeksan lähinaapurinsa olomuodosta. Jos naapureista on tarkalleen:

0-1 elossa, niin solu kuolee ikävään.

2 elossa, niin solun olotila ei muutu.

3 elossa, niin solu elää (eli se joko jatkaa eloaan tai syntyy).

4-8 elossa, niin solu rutistuu kuoliaaksi.

Liite B. Kommentoitu ohjelmalistaus

1. GameOfLife.java

```
package gameOfLife;

//=====
//--GameOfLife.java-----
//=====

import java.awt.Dimension;
import java.util.Random;
import javax.swing.JPanel;
import javax.swing.JApplet;
import gameOfLife.model.*;
import gameOfLife.view.*;
import gameOfLife.controller.*;

/*<applet
code = gameOfLife.GameOfLife
width = 800
height = 600>
</applet>*/

/**
 * Harjoitustyöprojekti 2 course exercise.
 * The GameOfLife is the root class and starts the game.
 * @author Vesa Nieminen
 * @version 2006-05-07
 * @since 2006-05-06
 */
public class GameOfLife extends JApplet implements Runnable {

//=====
//--Variables-----
//=====

    /** The width of the screen */
    private int width;
    /** The height of the screen */
    private int height;
    /** The view of the game */
    private LifeView view;
    /** The model of the game ie. the game world itself */
    private LifeModel model;
    /** True if the game is currently being "run". */
    private boolean gameRunning = true;
    /** The random number generator. Not used yet */
    private Random random = new Random(System.currentTimeMillis());
    /** The thread the game runs on */
    private Thread gameThread;

//=====
//--Constructor-----
//=====

    /**
     * Setup for the game.
     */
}
```

```

    */
    public GameOfLife() {
        width = 800;
        height = 600;
        // Create a new LifeModel with a world size of 200x150
        model = new LifeModel(200, 150);
        // Create a new LifeView with a screen size of 800x600
        view = new LifeView(width, height, model);
        // Add the control side of the game
        view.addMouseListener(new LifeController(view));
        view.addMouseMotionListener(new LifeController(view));
        // Get hold of the content of the frame and set the resolution
        JPanel panel = (JPanel) getContentPane();
        panel.setPreferredSize(new Dimension(width, height));
        panel.setLayout(null);
        panel.add(view);
    } // GameOfLife()

//=====
//--Methods-----
//=====

/**
 * Initiates the game.
 */
public void start () {
    gameThread = new Thread (this);
    gameThread.start ();
} // start()

/**
 * @inheritDoc
 * Initializes the view and starts the gameloop.
 */
public void run () {
    view.init();
    gameLoop();
} // run()

/**
 * Shuts the game down before closing the applet.
 */
public void destroy() {
    gameRunning = false;
} // destroy()

/**
 * The gameloop. Draws the screen and update the model.
 *
 */
public void gameLoop() {
    // Keep looping until the game ends
    while(gameRunning) {
        // draw the screen
        view.draw();
        // live one life cycle
        model.live();
        // Finally pause for a bit. Note: this should run us at about
        // 100 fps but in windows this might vary each loop due to a
        // bad implementation of timer.
        try { Thread.sleep(100); } catch (Exception e) {
            System.out.println("Exception: " +e);
        } //catch
    } // while
} // gameLoop()

```

```
//=====
//--Program entry point-----
//=====

/**
 * GameOfLife starts here.
 * @param args Not used.
 */
public static void main(String[] args) {
    Display.run(new GameOfLife(), 800, 600);
} // main(String[])

} // class GameOfLife
```

2. Display.java

```
package gameOfLife.view;

//=====
//--Display.java-----
//=====

import javax.swing.*;

/**
 * The Display class is a framework for displaying both Swing applications
 * and applets easily. Based on Bruce Eckel's Console class.
 * @author Vesa Nieminen
 * @version 2006-05-06
 * @since 2006-05-06
 */
public class Display {

//=====
//--Static methods-----
//=====

/**
 * Create a title string from the class name
 * @param o The object whose name is going to be used.
 * @return The retrieved String.
 */
public static String title(Object o) {
    String t = o.getClass().toString();
    // Remove the word "class":
    if(t.indexOf("class") != -1)
        t = t.substring(6);
    return t;
} // title(Object)

/**
 * Framework method for JFrame type displaying.
 * @param frame The JFrame to use.
 * @param width The desired width.
 * @param height The desired height.
 */
public static void
run(JFrame frame, int width, int height) {
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width, height);
    frame.setVisible(true);
}
```

```

} // run(JFrame, int, int)

/**
 * Framework method for JApplet type displaying.
 * @param applet The JApplet to use.
 * @param width The desired width.
 * @param height The desired height.
 */
public static void
run(JApplet applet, int width, int height) {
    JFrame frame = new JFrame(title(applet));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(applet);
    frame.setSize(width, height);
    frame.pack();
    applet.init();
    applet.start();
    frame.setVisible(true);
} // run(JApplet, int, int)

/**
 * Framework method for JPanel type displaying.
 * @param panel The JPanel to use.
 * @param width The desired width.
 * @param height The desired height.
 */
public static void
run(JPanel panel, int width, int height) {
    JFrame frame = new JFrame(title(panel));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(panel);
    frame.setSize(width, height);
    frame.setVisible(true);
} // run(JPanel, int, int)

} // class Display

```

3. LifeView.java

```

package gameOfLife.view;

//=====
//--LifeView.java-----
//=====

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferStrategy;
import gameOfLife.model.LifeModel;

/**
 * LifeView is used to display the Game of life.
 * @author Vesa Nieminen
 * @version 2006-05-07
 * @since 2006-05-06
 */
public class LifeView extends Canvas {

//=====
//--Variables-----
//=====

```

```

// width of the screen
private int width;
// height of the screen
private int height;
// the world model used
private LifeModel model;
// the width of a cell on screen
private int cellWidth;
// the height of a cell on screen
private int cellHeight;

/**
 * The strategy that allows the usage accelerated page flipping,
 * so that the graphical content of the game won't flicker annoyingly
 * and isn't blightingly slow.
 */
private BufferStrategy strategy;

//=====
//--Constructor-----
//=====

/**
 * The constructor.
 * Precondition: width / model.getWidth() should yield an even number.
 * height / model.getHeight() should yield an even number.
 * @param width The width of the canvas
 * @param height The height of the canvas
 * @param model The LifeModel to draw
 */
public LifeView(int width, int height, LifeModel model) {
    super();
    this.width = width;
    this.height = height;
    this.model = model;
    cellWidth = width / model.getWidth();
    cellHeight = height / model.getHeight();

    // Set up this canvas size
    setBounds(0,0,width,height);

    // Tell AWT not to bother repainting our canvas since we're
    // going to do that ourself in accelerated mode
    setIgnoreRepaint(true);
} // LifeView()

//=====
//--Methods-----
//=====

/**
 * Draws the state of the world on the canvas.
 */
public void draw() {
    // Get hold of a graphics context for the accelerated surface
    // and blank it out
    Graphics2D g = (Graphics2D) strategy.getDrawGraphics();
    g.setColor(Color.black);
    g.fillRect(0,0,width,height);

    // Set the color for the cells and the background
    g.setColor(Color.RED);
    g.setBackground(Color.BLACK);
}

```

```

    // Get the world to b
    Boolean[][] b = model.getWorld().clone();
    for (int i = 0; i < b.length; ++i) {
        for (int j = 0; j < b[0].length; ++j) {
            if (b[i][j] != null)
                // Draw the cells on the canvas one by one
                g.fillRect(i*cellWidth, j*cellHeight, cellWidth, cellHeight);
        } // for
    } // for
    // Completed drawing so clear up the graphics
    // and flip the buffer over
    g.dispose();
    strategy.show();
} // draw()

/**
 * The needed initializations for this object. One should not use it
 * before this method is called and this method should not be called
 * before there exists a JFrame, JPanel or a JApplet which can be used
 * as a peer.
 */
public void init() {
    // Request that this Component (canvas) gets the input focus,
    // and that this Component's top-level ancestor become
    // the focused Window (comment taken from the Java API).
    requestFocus();

    // Create the buffering strategy which will allow AWT
    // to manage the accelerated graphics
    addNotify();
    createBufferStrategy(2);
    strategy = getBufferStrategy();
} // init()

/**
 *
 * @return the world model. Do not modify!
 */
public LifeModel getModel() {
    return model;
} // getModel()

/**
 *
 * @return the cell width on screen
 */
public int getCellWidth() {
    return cellWidth;
} // getCellWidth()

/**
 *
 * @return the cell height on screen
 */
public int getCellHeight() {
    return cellHeight;
} // getCellHeight()
} // class Canvas

```

4. LifeModel.java

```
package gameOfLife.model;

//=====
//--LifeModel.java-----
//=====

/**
 * LifeModel contains the Game of life world.
 * @author Vesa Nieminen
 * @version 2006-05-06
 * @since 2006-05-06
 *
 */
public class LifeModel {

//=====
//--Variables-----
//=====

    // The current state of the world
    private Boolean[][] world;
    // A temporary state of the world
    private Boolean[][] buffer;
    // Is the game paused?
    private boolean pause;
    // The maximum x-coordinate in the world
    private int xMax;
    // The maximum y-coordinate in the world
    private int yMax;
    // A temporary variable defined as a class variable.
    // Holds the count of neighbouring cells.
    private int count;

//=====
//--Constructor-----
//=====

    /**
     * Creates a new Game of life type world.
     * Precondition: width >= 3 & height >= 3.
     * @param width The width of the world to be created.
     * @param height The height of the world to be created.
     */
    public LifeModel(int width, int height) {
        world = new Boolean[width][height];
        buffer = new Boolean[width][height];
        pause = false;
        xMax = world.length - 1;
        yMax = world[0].length - 1;
    } // LifeModel(int, int)

//=====
//--Methods-----
//=====

    /**
     * Changes the state of the Cell at coordinate (x,y). Alive -> dead or
     * dead -> alive.
     * @param x The x-coordinate.
     * @param y The y-coordinate.
     */
    public void changeCell(int x, int y)
```

```

throws ArrayIndexOutOfBoundsException{
    if (x < 0 | world.length <= x)
        throw new ArrayIndexOutOfBoundsException();
    if (y < 0 | world[0].length <= y)
        throw new ArrayIndexOutOfBoundsException();
    if (world[x][y] == null) {
        createCell(x, y);
    }// if
    else {
        deleteCell(x, y);
    }// else
}// changeCell(int, int)

/**
 * Creates a new Cell in coordinate (x,y).
 * Precondition: a Cell doesn't exist in the world at (x,y) and the
 * coordinates are valid (inside the world).
 * @param x X-coordinate.
 * @param y Y-coordinate.
 */
private void createCell(int x, int y) {
    Boolean b = new Boolean(true);
    world[x][y] = b;
}// createCell(int, int)

/**
 * Deletes a Cell from the world.
 * Precondition: a Cell exists in the world at (x,y) and the coordinates are
 * valid (inside the world).
 * @param x X-coordinate.
 * @param y Y-coordinate.
 */
private void deleteCell(int x, int y) {
    world[x][y] = null;
}// deleteCell(int, int)

/**
 * Gets the number of neighbouring Cells of a Cell at a given coordinate.
 * @param x X-coordinate.
 * @param y Y-coordinate.
 * @return number of neighbouring Cells of a Cell at (x,y).
 */
private int getNeighbourCount(int x, int y) {
    count = 0;
    // check if x or y is at the edge of the world. Makes the count
    // calculation more complex if it is, so this if statement provides
    // a simple optimization jump. Effectively reduces the coefficient
    // in the cost function.
    if (x == 0 || x == xMax || y == 0 || y == yMax ) {
        // left edge
        if (x == 0) {
            // anywhere but top & bottom on the left edge of the world
            if (y != 0 && y != yMax) {
                if (world[x][y-1] != null) ++count;
                if (world[x][y+1] != null) ++count;
                if (world[x+1][y-1] != null) ++count;
                if (world[x+1][y] != null) ++count;
                if (world[x+1][y+1] != null) ++count;
                if (world[xMax][y-1] != null) ++count;
                if (world[xMax][y] != null) ++count;
                if (world[xMax][y+1] != null) ++count;
            }
            return count;
        }// if
        // top left corner of the world
        if (y == 0) {

```

```

        if (world[0][1] != null) ++count;
        if (world[1][0] != null) ++count;
        if (world[1][1] != null) ++count;
        if (world[xMax][0] != null) ++count;
        if (world[xMax][1] != null) ++count;
        if (world[0][yMax] != null) ++count;
        if (world[1][yMax] != null) ++count;
        if (world[xMax][yMax] != null) ++count;
        return count;
    }// if
    // lower left corner of the world
    if (y == yMax) {
        if (world[0][0] != null) ++count;
        if (world[1][0] != null) ++count;
        if (world[xMax][0] != null) ++count;
        if (world[0][yMax-1] != null) ++count;
        if (world[1][yMax-1] != null) ++count;
        if (world[1][yMax] != null) ++count;
        if (world[xMax][yMax-1] != null) ++count;
        if (world[xMax][yMax] != null) ++count;
        return count;
    }// else if

}// if
// right edge
if (x == xMax) {
    // anywhere but top & bottom on the right edge of the world
    if (y != 0 && y != yMax) {
        if (world[x][y-1] != null) ++count;
        if (world[x][y+1] != null) ++count;
        if (world[xMax-1][y-1] != null) ++count;
        if (world[xMax-1][y] != null) ++count;
        if (world[xMax-1][y+1] != null) ++count;
        if (world[0][y-1] != null) ++count;
        if (world[0][y] != null) ++count;
        if (world[0][y+1] != null) ++count;
        return count;
    }// if
    // top right corner of the world
    if (y == 0) {
        if (world[0][0] != null) ++count;
        if (world[0][1] != null) ++count;
        if (world[xMax-1][0] != null) ++count;
        if (world[xMax-1][1] != null) ++count;
        if (world[xMax][1] != null) ++count;
        if (world[xMax-1][yMax] != null) ++count;
        if (world[xMax][yMax] != null) ++count;
        if (world[0][yMax] != null) ++count;
        return count;
    }// else if
    // lower right corner of the world
    if (y == yMax) {
        if (world[0][0] != null) ++count;
        if (world[xMax-1][0] != null) ++count;
        if (world[xMax][0] != null) ++count;
        if (world[0][yMax-1] != null) ++count;
        if (world[0][yMax-1] != null) ++count;
        if (world[xMax-1][yMax-1] != null) ++count;
        if (world[xMax][yMax-1] != null) ++count;
        if (world[xMax-1][yMax] != null) ++count;
        return count;
    }// else if
}// if
// top edge
if (y == 0) {

```

```

// anywhere but at the left & right side on the top edge of
// the world
if (x != 0 && x != xMax) {
    if (world[x-1][y] != null) ++count;
    if (world[x-1][y+1] != null) ++count;
    if (world[x][y+1] != null) ++count;
    if (world[x+1][y] != null) ++count;
    if (world[x+1][y+1] != null) ++count;
    if (world[x-1][yMax] != null) ++count;
    if (world[x][yMax] != null) ++count;
    if (world[x+1][yMax] != null) ++count;
    return count;
} // if
} // if
// bottom edge
if (y == yMax) {
    // anywhere but at the left & right side on the bottom edge of
    // the world
    if (x != 0 && x != xMax) {
        if (world[x-1][0] != null) ++count;
        if (world[x][0] != null) ++count;
        if (world[x+1][0] != null) ++count;
        if (world[x-1][yMax-1] != null) ++count;
        if (world[x][yMax-1] != null) ++count;
        if (world[x+1][yMax-1] != null) ++count;
        if (world[x-1][yMax] != null) ++count;
        if (world[x+1][yMax] != null) ++count;
        return count;
    } // if
} // if
} // if
// x and y are not at the edge of the world, ie. they are inside.
else {
    if (world[x-1][y-1] != null) ++count;
    if (world[x][y-1] != null) ++count;
    if (world[x+1][y-1] != null) ++count;
    if (world[x-1][y] != null) ++count;
    if (world[x+1][y] != null) ++count;
    if (world[x-1][y+1] != null) ++count;
    if (world[x][y+1] != null) ++count;
    if (world[x+1][y+1] != null) ++count;
    return count;
} // else
return count;
} // getNeighbourCount(int, int)

/**
 * Makes the world live one cycle.
 */
public void live() {
    // return if pause is in effect
    if (pause) return;
    buffer = new Boolean[world.length][world[0].length];
    for (int i = 0; i < world.length; ++i) {
        for (int j = 0; j < world[0].length; ++j) {
            // 0-1 neighbours => cell dies of loneliness
            if (getNeighbourCount(i,j) <= 1 && world[i][j] != null) {
                buffer[i][j] = null;
                continue;
            } // if
            // 2 neighbours => the cell continues to live
            if (getNeighbourCount(i,j) == 2 && world[i][j] != null) {
                buffer[i][j] = world[i][j];
                continue;
            }
        }
    }
}

```

```

        }// if
        // 3 neighbours => a new cell is born if there isn't one already
        if (getNeighbourCount(i,j) == 3 && world[i][j] == null) {
            buffer[i][j] = new Boolean(true);
            continue;
        }// if
        // 3 neighbours and there already exists a cell
        if (getNeighbourCount(i,j) == 3 && world[i][j] != null) {
            buffer[i][j] = world[i][j];
            continue;
        }// if
        // 4-8 neighbours => the cell crumples up (dies)
        if (getNeighbourCount(i,j) > 3) {
            buffer[i][j] = null;
            continue;
        }// if
    }// for
} // set the world variable to point at the new state of the world
world = buffer;
} // live()

/**
 * Makes the world live x cycles.
 * Precondition: x > 0.
 * @param x The amount of life cycles to live.
 */
public void live(int x) {
    for (int i = 0; i < x; ++i) {
        live();
    } // for
} // live(int)

/**
 * Used to get the state of the world. One should not modify the Boolean
 * table that is returned!
 * @return the state of the Game of life world.
 */
public Boolean[][] getWorld() {
    return world;
} // getWorld()

/**
 *
 * @return world width
 */
public int getWidth() {
    return world.length;
} // getWidth()

/**
 *
 * @return world height
 */
public int getHeight() {
    return world[0].length;
} // getHeight()

/**
 *
 * @return is the game paused?
 */
public boolean getPause() {
    return pause;
} // getPause()

```

```

/**
 * Switches the state of the game. Unpaused -> paused, paused -> unpaused.
 *
 */
public void switchPause() {
    if (pause) pause = false;
    else pause = true;
} // switchPause()

} // class LifeModel

```

5. LifeController.java

```

package gameOfLife.controller;

//=====
//--LifeController.java-----
//=====

import javax.swing.event.MouseInputAdapter;
import java.awt.event.MouseEvent;
import gameOfLife.view.LifeView;
import gameOfLife.model.LifeModel;

/**
 * Controlling side of the game.
 * @author Vesa Nieminen
 * @version 2006-05-07
 * @since 2006-05-07
 */
public class LifeController extends MouseInputAdapter {

//=====
//--Variables-----
//=====

    // x-coordinate on the screen
    private int xCoord;
    // y-coordinate on the screen
    private int yCoord;
    // x-coordinate in the world
    private int xPos;
    // y-coordinate in the world
    private int yPos;
    // a temporary x-coordinate in the world
    private int newXPos;
    // a temporary y-coordinate in the world
    private int newYPos;
    // a temporary variable that holds the id of a mousebutton that was pressed
    private int button;
    // a temporary variable that is used to exclude mouse button3 & dragging
    // from happening
    private static boolean noDragging;
    // the view of the world
    private LifeView view;
    // the world itself
    private LifeModel model;

//=====
//--Constructor-----
//=====

```

```

/**
 *
 * @param view the view to use with this controller
 */
public LifeController(LifeView view) {
    this.view = view;
    model = view.getModel();
    noDragging = false;
} // LifeController (LifeModel)

//=====
//--Methods-----
//=====

/**
 * A multipurpose eventhandler.
 * Manages 3 different mouse button presses.
 * 1) Button1 pressed == change cell state
 * 2) Anything but Button3 being dragged == change the states of multiple
 * cells
 * 3) Button3 pressed == pause the game
 */
public void takeAction(MouseEvent e) {
    button = e.getButton();
    switch (button) {
        // Button1 pressed
        case MouseEvent.BUTTON1:
            noDragging = false;
            xCoord = e.getX();
            yCoord = e.getY();
            xPos = xCoord / view.getCellWidth();
            yPos = yCoord / view.getCellHeight();
            model.changeCell(xPos, yPos);
            break;
        // mouse dragging
        case 0:
            if (noDragging) break;
            xCoord = e.getX();
            yCoord = e.getY();
            newXPos = xCoord / view.getCellWidth();
            newYPos = yCoord / view.getCellHeight();
            if (xPos != newXPos || yPos != newYPos) {
                xPos = newXPos;
                yPos = newYPos;
                model.changeCell(xPos, yPos);
            } // if
            break;
        // Button3 pressed
        case MouseEvent.BUTTON3:
            noDragging = true;
            model.switchPause();
            break;
    } // switch (button)
} // takeAction()

/**
 * Mousepressed eventhandling
 */
public void mousePressed(MouseEvent e) {
    takeAction(e);
} // mousePressed (MouseEvent)

/**
 * Mousedragged eventhandling

```

```
    */  
    public void mouseDragged(MouseEvent e) {  
        takeAction(e);  
    } // mousePressed (MouseEvent)  
} // class LifeController
```

Liite C. Käyttöohje

Kaikissa esimerkeissä käyttöjärjestelmänä on Windows XP ja ohjelma on c:\gameOfLife\ hakemistossa.

Kääntäminen:

```
javac -cp "c:\gameOfLife;" gameOfLife/GameOfLife.java
```

Kääntämisen yhteydessä on huolehdittava, että classpath on määritetty tarkalleen oikein. Esimerkiksi puolipisteen poistaminen hakemistopolun lopusta tuottaa vain help tekstin tulostuksen konsoliin.

Appletviewerillä käynnistys:

```
appletviewer GameOfLife.htm
```

GameOfLife.htm oli C: aseman juuressa. Mikäli se oli samassa hakemistossa kuin ohjelman pääluokka, niin ei appletviewer löytänyt luokkia.

Selaimella käynnistys:

Avaa GameOfLife.htm selaimella ja huolehdi, että kaikki tiedostot ovat oikeissa hakemistoissa.

Java ohjelmana käynnistys:

```
java -classpath "c:\gameOfLife\;" gameOfLife.GameOfLife
```

Mielenkiintoisena nuanssina luokan nimen pitää sisältää luokkarakenteen mukainen polku pisteen kera, muuten ei ajo onnistu.