Tetris in AspectJ

2003-01-20

By Gustav Evertsson, pt99gev@student.bth.se

1. Contents

2. Introduction

This paper is done as a part of the course Advanced Software Engineering (pad004) at Blekinge Institute of Technology.

As a compliment to the two previous reports [3, 4] that I have written about AspectJ have I done a small practical case study to see if the conclusions that I came up with are true. I have chosen to do this by programming a small game called Tetris. What I wanted to see is how AspectJ can be used to change an already complete program and some of the problems around this.

The code has been compiled and tested with the following versions:
AspectJ [1]: 1.1b2
Java [5]: 1.4.0.01

Problem description

I started by programming the Tetris game without any thoughts about how to implement the aspects later on. A description about this can be found in chapter 3. The focus of this report is on how the aspect that I developed works. This is done in chapter 4. I decided to make the following aspects. I wanted to test different parts of AspectJ so they are made to show different things that can be made with aspects:

- **Design checker** – I use it to check that the layer architecture that I use is correct. It will generate an error output during the compilation if it founds that something is wrong. (Chapter 4.1.1.)
- **Game Info** – This is a Panel for showing info about the game. Other aspects then add the info to this panel. I use this aspect as an example about how the key *dominates* can be used. (Chapter 4.2.1.)
- **Menu** – This aspect adds a menu to the gui and connects it to functionality that is already implemented. (Chapter 4.2.2.)
- **Counter** – This aspect counts deleted lines. It uses the Game Info aspect to add a label to the gui. (Chapter 4.3.1.)
- **Level** – This is the only aspect in this program that has a pointcut to another aspect. It uses the Counter to see how many lines that has been deleted and makes the game go faster when it reaches the next level. (Chapter 4.3.2.)
- **New Blocks** – This aspect adds two new types of blocks to the game. The thing that is special about this aspect is that it use the proceed() call in the advices. (Chapter 4.4.1.)
- **Next Block** – This is the only aspect that completely changes a method in the original program. It changes so the program shows what the next block is when the current block is dropped. (Chapter 4.4.1.)

What is Tetris?

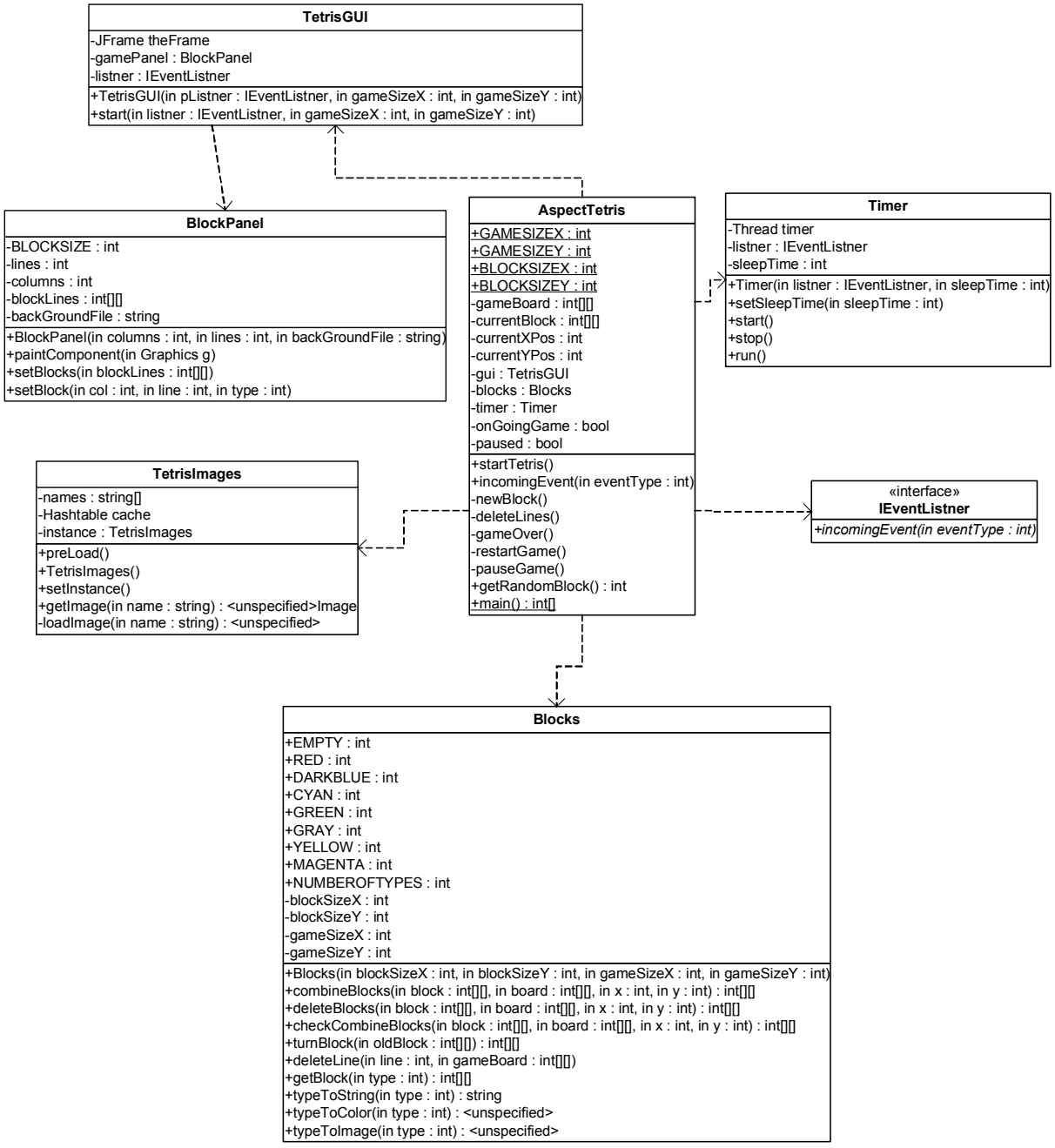
The rules in Tetris are easy to understand. The goal is to pack the blocks so they become lines. The lines are then deleted so the user can add more blocks. The blocks are randomly generated at the top of the game board and are slowly dropped down until they reach the bottom. The game ends if the blocks reach to the top of the game board.

3. Design

The design for the game was made to be as easy as possible and no extra features were included in the game from start. These where later added with aspects.

3.1. Overall architecture

The overall architecture is divided into three different parts. First the Main class that has all the game rules and handles the incoming events. The second is the Logic package that handles all the data manipulation. And the last is the gui package that draws everything on the screen. The main class creates the Logic and the gui classes and the gui classes then only calls the main class through the IEventLister interface. This is because I wanted to lower the coupling between the gui classes and the main class.



4. Different Aspects

These are the aspects that I have developed for this project.

4.1. Development aspects

4.1.1 Design Checker

This aspect was only used during the development phase. This aspect checks so the architecture is followed in the code. I use it to check that the layer architecture that I use is correct. It will generate an error output during the compilation if it founds that something is wrong.

```
package Aspects.Development;

public aspect DesignCheck {

	declare warning: call(Gui.BlockPanel.new(..)) && !within(Gui.*) &&
!within(Aspects..*): "Do not create BlockPanel outside the Gui package!";

	declare warning: call(* AspectTetris.*(..)) && !within(AspectTetris) &&
!within(Aspects..*): "Do not call AspectTetris outside the class, use the
IEventListner interface!";

}
```

It only requires two lines of code to do this. The first one checks so the BlockPanel class is not created outside the Gui package. The second checks so the AspectTetris class is not called directly (it must be done through the IEventListner interface).

4.2. Gui aspects

This is some aspects that change the graphical user interface in different ways.

4.2.1 Game Info

This is a Panel for showing info about the game. Other aspects then add the info to this panel. It is marked as dominating the rest of the aspects because the panel must be created first (It will generate a null pointer exception if another aspects tries to add anything to the panel if not the GameInfo aspects is executed first).

```
package Aspects.Gui;

import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import Gui.*;

public aspect GameInfo dominates Aspects.*.* {
	//initialization
	pointcut guiInit() : execution(Gui.TetrisGUI.new(..));

	public static JPanel infoPanel;

	before() : guiInit() {
		//System.out.println("Target: " + thisJoinPoint.getTarget());
		//System.out.println("This  : " + thisJoinPoint.getThis());
		TetrisGUI theGui = (TetrisGUI)thisJoinPoint.getThis();
		theGui.setLayout(new BorderLayout());
```

```
                theGui.setBorder(new EtchedBorder());

                infoPanel = new JPanel();
                infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y_AXIS));
                theGui.add(infoPanel, BorderLayout.WEST);
        }
}
```

In the declaration is the keyword *dominates* used to control in what order the aspect is executed. It means in this case that it is always executed before all other aspects in the Aspects package. The guiInit() pointcut is true when the TetrisGui class is initialized. The before() advice is then executed and the panel is added to the gui.

4.2.2. Menu

I added some functionality in form of "New Game" and "Pause" from start. This can be controlled with the keys F2 and F3. I wanted with this aspect show how a menu can be added and done to call the same functionality.

```
package Aspects.Gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import Gui.*;
import EventInterface.*;

public aspect Menu  implements ActionListener {
        protected JMenuItem newGameMI;
        protected JMenuItem pauseMI;
        protected JMenuItem exitMI;
        protected IEventListner tetris;

        //initialization
        pointcut guiInit() : execution(Gui.TetrisGUI.new(..));

        pointcut tetrisInit(IEventListner tetris) : execution(AspectTetris.new(..))
&& target(tetris);

        before() : guiInit() {
                TetrisGUI theGui = (TetrisGUI)thisJoinPoint.getThis();

                JMenuBar menuBar = new JMenuBar();

                JMenu fileMenu = (JMenu) menuBar.add(new JMenu("File"));
                newGameMI = (JMenuItem) fileMenu.add(new JMenuItem("New Game"));
                pauseMI = (JMenuItem) fileMenu.add(new JMenuItem("Pause"));
                fileMenu.add(new JSeparator());
                exitMI = (JMenuItem) fileMenu.add(new JMenuItem("Exit"));

                newGameMI.addActionListener(this);
                pauseMI.addActionListener(this);
                exitMI.addActionListener(this);

                theGui.add(menuBar, BorderLayout.NORTH);

        }

        before(IEventListner tetris) : tetrisInit(tetris) {
                this.tetris = tetris;
   }

   public void actionPerformed(ActionEvent e) {
                if (e.getSource().equals(newGameMI)) {
```

```
                        tetris.incomingEvent(IEventListner.NEWGAME);
        }
        else if (e.getSource().equals(pauseMI)) {
                        tetris.incomingEvent(IEventListner.PAUSE);
        }
        else if (e.getSource().equals(exitMI)) {
                        System.exit(0);
        }
   }

}
```

The first two pointcuts is used to get the instances of the AspectTetris and TetrisGui classes that are later used. The before guiInit() advice also adds the menu in the gui. The aspect implements the ActionListener interface in the same way as with a normal class.

4.3. Highscore aspects

These aspects are made to make the game funnier by adding a counter and levels to it.

4.3.1. Counter

The first thing to get the game a little bit funnier is to add a counter that counts the number of lines that is deleted. This aspect adds a label to the GameInfo aspect.

```
package Aspects.Highscore;

import java.awt.*;
import javax.swing.*;
import Aspects.Gui.*;

public aspect Counter {

        pointcut guiInit() : execution(Gui.TetrisGUI.new(..));

        pointcut deleteLines() : call(* AspectTetris.deleteLines());

        pointcut deleteLine() : call(* Logic.Blocks.deleteLine(..));

        pointcut newgame() : call(* AspectTetris.restartGame());

        pointcut gameOver() : call(* AspectTetris.gameOver());

        protected int currentLines = 0;
        protected int totalLines = 0;

        protected JLabel lineLabel;

        before() : guiInit() {
                lineLabel = new JLabel("Lines: 0");
                GameInfo.infoPanel.add(lineLabel);
        }

        before() : deleteLines() {
                currentLines = 0;
        }

        after() : deleteLines() {
                totalLines += currentLines;
                if(currentLines != 0)
                        //System.out.println("Deleted " + currentLines + " lines
(Total: " + totalLines + ").");
                        lineLabel.setText("Lines: " + totalLines);
        }

        before() : deleteLine() {
```

```
                currentLines++;
        }

        before() : newgame() {
                totalLines = 0;
                currentLines = 0;
                lineLabel = new JLabel("Lines: 0");
        }

        after() : gameOver() {
                System.out.println("Deleted totally " + totalLines + " lines.");

        }
}
```

It is made so it can count how many lines that is deleted in the same time. This can be used in the future if I decide to count score as well.

4.3.2. Level

The level aspect is made so it count the lines and when it reach a certain number of lines will it change the sleeping time for the timer so the game will go faster and faster. This aspect is a little bit special in the way that it has a pointcut that listen to another aspect.

```
package Aspects.Highscore;

import java.awt.*;
import javax.swing.JLabel;
import Aspects.Gui.*;
import EventInterface.*;
import Logic.*;

public aspect Levels {

        pointcut guiInit() : execution(Gui.TetrisGUI.new(..));

        pointcut timerInit(Timer timer, IEventListner listner, int sleepTime) :
execution(Logic.Timer.new(IEventListner, int)) && target(timer) && args(listner,
sleepTime);

        pointcut deleteLines(int totalLines) : set(int
Aspects.Highscore.Counter.totalLines) && args(totalLines);

        pointcut newgame() : call(* AspectTetris.restartGame());

        protected JLabel levelLabel;
        protected int nextLevel = 5;
        protected int currentLevel = 1;
        protected int sleepTime;
        protected Timer timer;


        before() : guiInit() {
                levelLabel = new JLabel("Level: " + currentLevel);
                GameInfo.infoPanel.add(levelLabel);
        }

        after(int totalLines) : deleteLines(totalLines) {
                if(totalLines > nextLevel) {
                        currentLevel++;
                        levelLabel.setText("Level: " + currentLevel);
                        nextLevel = nextLevel * 2;
                        sleepTime = (int)((double)sleepTime * 0.8);
                        timer.setSleepTime(sleepTime);
                        //System.out.println("New sleepTime: " + sleepTime);
```

```
                }
        }

        before(Timer timer, IEventListner listner, int sleepTime) : timerInit(timer,
listner, sleepTime) {
                this.timer = timer;
                this.sleepTime = sleepTime;
        }

        before() : newgame() {
                nextLevel = 5;
                currentLevel = 1;
                levelLabel.setText("Level: " + currentLevel);
        }
}
```

The two *Init pointcuts is used to get the instances to the classes in the same way as with the aspects above. The difference is in the pointcut deleteLines() that listen to changes with the totalLines variable in the Score aspect.

4.4. Logic aspects

4.4.1. New blocks

This aspect adds two new types of blocks. It adds this functionality to the Blocks class. It use the around advice to do this. It first checks if the type if of the old types and if it is so is the normal method called. This makes it possible to modify a method but it is not needed to rewrite all the old code.

```
package Aspects.Logic;

import java.awt.*;
import Logic.*;

public aspect NewBlocks {

        public static final int WHITE   = 7;
        public static final int BLUE    = 8;

        pointcut getBlock(int type) : call(int[][] Logic.Blocks.getBlock(int)) &&
args(type);

        pointcut typeToString(int type) : call(String
Logic.Blocks.typeToString(int)) && args(type);

        pointcut typeToColor(int type) : call(Color Logic.Blocks.typeToColor(int))
&& args(type);

        pointcut typeToImage(int type) : call(Image Logic.Blocks.typeToImage(int))
&& args(type);

        pointcut numberOfTypes() : get(static int Blocks.NUMBEROFTYPES);

        int around() : numberOfTypes() {
                return 9;
        }

        int[][] around(int type) : getBlock(type) {
                if(type < 7)
                        return proceed(type);

                int[][] newBlock = new int[4][4];
                for(int x = 0; x < 4; x++) {
                        for(int y = 0; y < 4; y++) {
                                newBlock[x][y] = Blocks.EMPTY;
```

```
                        }
                }

                if(type == WHITE ) {
                        newBlock[0][1] = WHITE;
                        newBlock[0][2] = WHITE;
                        newBlock[1][1] = WHITE;
                        newBlock[2][1] = WHITE;
                        newBlock[2][2] = WHITE;
                }
                else if(type == BLUE ) {
                        newBlock[1][0] = BLUE;
                        newBlock[1][1] = BLUE;
                        newBlock[1][2] = BLUE;
                        newBlock[2][1] = BLUE;
                        newBlock[2][2] = BLUE;
                }

                return newBlock;
        }

        String around(int type) : typeToString(type) {
                if(type < 7)
                        return proceed(type);
                else if(type == WHITE)
                        return "white";
                else if(type == BLUE)
                        return "blue";
                else
                        return "";

        }

        Image around(int type) : typeToImage(type) {
                if(type < 7)
                        return proceed(type);

                if(type == WHITE)
                        return TetrisImages.getImage("gray.gif");
                else if(type == BLUE)
                        return TetrisImages.getImage("magenta.gif");
                else
                        return null;

        }

        Color around(int type) : typeToColor(type) {
                if(type < 7)
                        return proceed(type);
                else if(type == WHITE)
                        return Color.WHITE;
                else if(type == BLUE)
                        return Color.BLUE;
                else
                        return Color.BLACK;

        }
}
```

The numberOfTypes() pointcut change the Blocks class so all classes start to count with the two new blocks. Then are the getBlock() and the different type converting method overridden to include the two new blocks.

4.4.2. Next block

This is the only aspect that completely changes a method. In this case change the getRandomBlock() method so it instead have a “next block” that is returned. It adds a BlockPanel the GameInfo aspect so the user can see what the next block is.

```
package Aspects.Logic;

import java.util.Random;
import java.awt.*;
import javax.swing.*;
import Aspects.Gui.*;
import Gui.*;
import Logic.*;

public aspect NextBlock {

	pointcut guiInit() : execution(Gui.TetrisGUI.new(..));

	pointcut getNextBlock() : call(* AspectTetris.getRandomBlock());

	protected BlockPanel nextBlockPanel;
	protected int nextBlock;

	before() : guiInit() {
		Random rn = new Random();
		nextBlock = rn.nextInt(Blocks.NUMBEROFTYPES);


		nextBlockPanel = new BlockPanel(4, 4, "");
		GameInfo.infoPanel.add(nextBlockPanel);
		nextBlockPanel.setBlocks(Blocks.getBlock(nextBlock));
	}

	int[][] around() : getNextBlock() {
		int currentBlock = nextBlock;

		Random rn = new Random();
		nextBlock = rn.nextInt(Blocks.NUMBEROFTYPES);
		nextBlockPanel.setBlocks(Blocks.getBlock(nextBlock));


		return Blocks.getBlock(currentBlock);
	}
}
```

The getRandomBlock() method in the AspectTetris class is modified so it holds one block and in this way can output what the next block is. The guiInit() pointput adds the nextBlockPanel to the gui.

4. Conclusion

First the positive sides that I found. I found it to be easier than I first had expected to add and/or modify the functionality with aspects. I found the code to be “cleaner” because the classes only needed to have the game logic. All extra features that can make the code more complex and harder to understand were placed in the aspects. The powerful pointcuts and advices made that the aspects didn’t have to include much code but could be made small and straightforward. The feature to check the design with aspects under the development may not have been that useful here because one person can have control over all the code but I think this can be very useful in bigger projects there no one have full control over everything.

One negative side of aspects that I found is that the developer must have full control over the code to know how the aspects affect the program. Problems may also occur if the original program is changed because it can be hard to know how the aspects react to the changes. It is just this uncertainness that I see as the big problem with aspects. The coupling between the classes and the different aspects are often low for different reasons but this makes it hard to know how they affect each other. Even the compiler itself changes so just because the developer feels finished with some part of the code it is possible that he must go back and modify it.

So my core conclusion I found is that you must know how the program works to be able to make good aspects. It is not possible to make aspects without system design and the source code. However if the developer have this knowledge about the program can the maintenance be made easier with aspects. There are things that you can’t make without aspects, checking the design for example.

5. References

1. AspectJ, http://www.eclipse.org/aspectj/

2. Ivan Kiselev, Aspect-Oriented Programming with AspectJ, 2001, Sams, ISBN: 0-672-32410-5

3. Gustav Evertsson, Aspect Oriented Programming, 2002

4. Gustav Evertsson, Strategies in Aspect Oriented Programming with AspectJ, 2002

5. Java, http://www.java.sun.com

6. Appendix

This is the source code from the project. The folder structure shall be the same as the package structure.

6.1 AspectTetris.java

```
import java.util.Random;
import Gui.*;
import EventInterface.*;
import Logic.*;

public class AspectTetris implements IEventListner {

	public static final int GAMESIZEX = 10;
	public static final int GAMESIZEY = 20;

	public static final int BLOCKSIZEX = 4;
	public static final int BLOCKSIZEY = 4;

	protected int[][] gameBoard;
	protected int[][] currentBlock;
	protected int currentXPos = (GAMESIZEX / 2) - (BLOCKSIZEX / 2);
	protected int currentYPos = 0;

	protected TetrisGUI gui;
	protected Blocks blocks;
	protected Timer timer;
	protected boolean onGoingGame = false;
	protected boolean paused = false;

	public void startTetris() {
		System.out.println("Starting AspectTetris...");

		TetrisImages.preLoad();

		blocks = new Blocks(BLOCKSIZEX, BLOCKSIZEY, GAMESIZEX, GAMESIZEY);

		// Creates an empty game board.
		gameBoard = new int[GAMESIZEX][GAMESIZEY];
		for(int x = 0; x < GAMESIZEX; x++) {
			for(int y = 0; y < GAMESIZEY; y++) {
				gameBoard[x][y] = Blocks.EMPTY;
			}
		}

		// Creates an empty block.
		currentBlock = new int[BLOCKSIZEX][BLOCKSIZEY];
		for(int x = 0; x < BLOCKSIZEX; x++) {
			for(int y = 0; y < BLOCKSIZEY; y++) {
				currentBlock[x][y] = Blocks.EMPTY;
			}
		}

		gui = TetrisGUI.start(this, GAMESIZEX, GAMESIZEY);
		gui.gamePanel.setBlocks(gameBoard);

		currentBlock = getRandomBlock();

		timer = new Timer(this, 700);
		timer.start();
		onGoingGame = true;
	}

	public void incomingEvent(int eventType) {
		if(!onGoingGame && eventType != IEventListner.NEWGAME)
```

```
                        return;
                if(paused && eventType != IEventListner.PAUSE && eventType !=
IEventListner.NEWGAME)
                        return;
                //System.out.println("New event of type " + eventType);
                if(eventType == IEventListner.UP) {    // Turn currentBlock...
                        if(blocks.checkCombineBlocks(blocks.turnBlock(currentBlock),
blocks.deleteBlocks(currentBlock, gameBoard, currentXPos, currentYPos),
currentXPos, currentYPos)){
                                currentBlock = blocks.turnBlock(currentBlock);
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                                gui.gamePanel.setBlocks(gameBoard);
                        }
                }
                else if(eventType == IEventListner.DOWN) {     // Fast down...
                        while(blocks.checkCombineBlocks(currentBlock,
blocks.deleteBlocks(currentBlock, gameBoard, currentXPos, currentYPos),
currentXPos, currentYPos + 1)){
                                currentYPos++;
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                        }

                        //System.out.println("The block has reached the bottom!");
                        blocks.combineBlocks(currentBlock, gameBoard, currentXPos,
currentYPos);
                        deleteLines();
                        newBlock();

                }
                else if(eventType == IEventListner.LEFT){
                        if(blocks.checkCombineBlocks(currentBlock,
blocks.deleteBlocks(currentBlock, gameBoard, currentXPos, currentYPos), currentXPos
- 1, currentYPos)){
                                currentXPos--;
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                                gui.gamePanel.setBlocks(gameBoard);
                        }
                }
                else if(eventType == IEventListner.RIGHT) {
                        if(blocks.checkCombineBlocks(currentBlock,
blocks.deleteBlocks(currentBlock, gameBoard, currentXPos, currentYPos), currentXPos
+ 1, currentYPos)){
                                currentXPos++;
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                                gui.gamePanel.setBlocks(gameBoard);
                        }
                }
                else if(eventType == IEventListner.TIMER) {
                        if(blocks.checkCombineBlocks(currentBlock,
blocks.deleteBlocks(currentBlock, gameBoard, currentXPos, currentYPos),
currentXPos, currentYPos + 1)){
                                currentYPos++;
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                                gui.gamePanel.setBlocks(gameBoard);
                        }
                        else {
                                //System.out.println("The block has reached the
bottom!");
                                blocks.combineBlocks(currentBlock, gameBoard,
currentXPos, currentYPos);
                                deleteLines();
                                newBlock();
                        }
```

```
                }
                else if(eventType == IEventListner.NEWGAME) {
                        restartGame();
                }
                else if(eventType == IEventListner.PAUSE) {
                        pauseGame();
                }


        }

        protected void newBlock() {

                currentXPos = (GAMESIZEX / 2) - (BLOCKSIZEX / 2);
                currentYPos = 0;



                currentBlock = getRandomBlock();

                if(!blocks.checkCombineBlocks(currentBlock, gameBoard, currentXPos,
currentYPos)){
                        gameOver();
                }
                else {
                        blocks.combineBlocks(currentBlock, gameBoard, currentXPos,
currentYPos);
                        gui.gamePanel.setBlocks(gameBoard);
                }
        }

        protected void deleteLines() {
                for(int i = 0; i < GAMESIZEY; i++) {
                        boolean complete = true;
                        for(int j = 0; j < GAMESIZEX; j++) {
                                if(gameBoard[j][i] == Blocks.EMPTY)
                                        complete = false;
                        }
                        if(complete) {
                                blocks.deleteLine(i, gameBoard);
                        }
                }
        }

        protected void gameOver() {
                System.out.println("Game Over!");
                onGoingGame = false;
                timer.stop();
        }

        protected void restartGame() {
                System.out.println("New Game...");

                // Creates an empty game board.
                gameBoard = new int[GAMESIZEX][GAMESIZEY];
                for(int x = 0; x < GAMESIZEX; x++) {
                        for(int y = 0; y < GAMESIZEY; y++) {
                                gameBoard[x][y] = Blocks.EMPTY;
                        }
                }

                newBlock();

                timer.start();
                paused = false;
                onGoingGame = true;
        }
```

```
	protected void pauseGame() {
		if(paused) {
			paused = false;
			timer.start();
		}
		else {
			paused = true;
			timer.stop();
			System.out.println("Pause...");
		}
	}

	public int[][] getRandomBlock() {
		Random rn = new Random();

		int randomValue = rn.nextInt(Blocks.NUMBEROFTYPES);
		System.out.println("New " + Blocks.typeToString(randomValue) + "
block.");
		return blocks.getBlock(randomValue);
	}



	public static void main(String[] args) {
		AspectTetris tetris = new AspectTetris();
		tetris.startTetris();
	}
}
```

6.2 Blocks.java

```
package Logic;

import java.awt.*;

public class Blocks {
	public static final int EMPTY          = -1;
	public static final int RED            = 0;
	public static final int DARKBLUE       = 1;
	public static final int CYAN           = 2;
	public static final int GREEN          = 3;
	public static final int GRAY           = 4;
	public static final int YELLOW  = 5;
	public static final int MAGENTA = 6;


	public static int NUMBEROFTYPES = 7;

	protected static int blockSizeX;
	protected static int blockSizeY;
	protected static int gameSizeX;
	protected static int gameSizeY;

	public Blocks(int blockSizeX, int blockSizeY, int gameSizeX, int gameSizeY)
{
		this.blockSizeX = blockSizeX;
		this.blockSizeY = blockSizeY;
		this.gameSizeX = gameSizeX;
		this.gameSizeY = gameSizeY;
	}

	public int[][] combineBlocks(int[][] block, int[][] board, int x, int y) {
		for(int i = 0; i < blockSizeX; i++) {
			for(int j = 0; j < blockSizeY; j++) {
				try {
					if(block[i][j] != Blocks.EMPTY) {
						board[i + x][j + y] = block[i][j];
					}
```

```
                                }
                                catch(Exception e) {
                                        System.out.println("Com Error " + e);
                                }

                        }
                }
                return board;
        }

        public int[][] deleteBlocks(int[][] block, int[][] board, int x, int y) {
                for(int i = 0; i < blockSizeX; i++) {
                        for(int j = 0; j < blockSizeY; j++) {
                                try {
                                        if(block[i][j] != Blocks.EMPTY) {
                                                board[i + x][j + y] = Blocks.EMPTY;
                                        }
                                }
                                catch(Exception e) {
                                        System.out.println("Del Error: " + e);
                                }

                        }
                }
                return board;
        }

        public boolean checkCombineBlocks(int[][] block, int[][] board, int x, int
y) {
                for(int i = 0; i < blockSizeX; i++) {
                        for(int j = 0; j < blockSizeY; j++) {
                                try {
                                        if(block[i][j] != Blocks.EMPTY) {
                                                if(j + y >= gameSizeY){
                                                        //System.out.println("Y SIZE
FEL, y:" + y + ", j:" + j);
                                                        return false;
                                                }
                                                else if(i + x < 0 || i + x >=
gameSizeX){
                                                        //System.out.println("X SIZE
FEL, x:" + x + ", i:" + i);
                                                        return false;
                                                }
                                                else if(board[i + x][j + y] !=
Blocks.EMPTY) {
                                                        //System.out.println("BLOCK
FEL");
                                                        return false;
                                                }
                                        }
                                }
                                catch(Exception e) {
                                        System.out.println("Check Error: " + e);
                                        return false;
                                }

                        }
                }
                return true;
        }

        public int[][] turnBlock(int[][] oldBlock) {
                int[][] newBlock = new int[blockSizeX][blockSizeY];

                for(int i = 0; i < blockSizeX; i++) {
                        for(int j = 0; j < blockSizeY; j++){
                                newBlock[j][i] = oldBlock[3-i][j];
```

```
                }
        }
        return newBlock;
}

public void deleteLine(int line, int[][] gameBoard) {
        for(int j = line; j >= 0; j--){
                for(int i = 0; i < gameSizeX; i++) {
                        if(j == 0)
                                gameBoard[i][j] = EMPTY;
                        else
                                gameBoard[i][j] = gameBoard[i][j - 1];
                }
        }
}


public static int[][] getBlock(int type) {
        // Reset current block first;
        int[][] newBlock = new int[blockSizeX][blockSizeY];
        for(int x = 0; x < blockSizeX; x++) {
                for(int y = 0; y < blockSizeY; y++) {
                        newBlock[x][y] = Blocks.EMPTY;
                }
        }

        if(type == RED ) {
                newBlock[1][0] = RED;
                newBlock[1][1] = RED;
                newBlock[1][2] = RED;
                newBlock[1][3] = RED;
        }
        else if(type == DARKBLUE ) {
                newBlock[1][2] = DARKBLUE;
                newBlock[2][1] = DARKBLUE;
                newBlock[2][2] = DARKBLUE;
                newBlock[3][1] = DARKBLUE;
        }
        else if(type == CYAN ) {
                newBlock[1][1] = CYAN;
                newBlock[2][1] = CYAN;
                newBlock[1][2] = CYAN;
                newBlock[2][2] = CYAN;
        }
        else if(type == GREEN ) {
                newBlock[1][1] = GREEN;
                newBlock[2][1] = GREEN;
                newBlock[2][2] = GREEN;
                newBlock[3][2] = GREEN;
        }
        else if(type == GRAY ) {
                newBlock[0][1] = GRAY;
                newBlock[1][1] = GRAY;
                newBlock[1][2] = GRAY;
                newBlock[2][1] = GRAY;
        }
        else if(type == YELLOW ) {
                newBlock[0][1] = YELLOW;
                newBlock[0][2] = YELLOW;
                newBlock[1][1] = YELLOW;
                newBlock[2][1] = YELLOW;
        }
        else if(type == MAGENTA ) {
                newBlock[0][1] = MAGENTA;
                newBlock[1][1] = MAGENTA;
                newBlock[2][1] = MAGENTA;
                newBlock[2][2] = MAGENTA;
        }
```

```
                return newBlock;
        }

        public static String typeToString(int type) {
                if(type == Blocks.RED)
                        return "red";
                else if(type == Blocks.DARKBLUE)
                        return "blue";
                else if(type == Blocks.CYAN)
                        return "cyan";
                else if(type == Blocks.GREEN)
                        return "green";
                else if(type == Blocks.GRAY)
                        return "gray";
                else if(type == Blocks.YELLOW)
                        return "yellow";
                else if(type == Blocks.MAGENTA)
                        return "magenta";
                else
                        return "";
        }

        public static Color typeToColor(int type) {
                if(type == Blocks.RED)
                        return Color.red;
                else if(type == Blocks.DARKBLUE)
                        return Color.blue;
                else if(type == Blocks.CYAN)
                        return Color.cyan;
                else if(type == Blocks.GREEN)
                        return Color.green;
                else if(type == Blocks.GRAY)
                        return Color.gray;
                else if(type == Blocks.YELLOW)
                        return Color.yellow;
                else if(type == Blocks.MAGENTA)
                        return Color.magenta;
                else
                        return Color.black;
        }

        public static Image typeToImage(int type) {
                if(type == Blocks.RED)
                        return TetrisImages.getImage("red.gif");
                else if(type == Blocks.DARKBLUE)
                        return TetrisImages.getImage("darkblue.gif");
                else if(type == Blocks.CYAN)
                        return TetrisImages.getImage("cyan.gif");
                else if(type == Blocks.GREEN)
                        return TetrisImages.getImage("green.gif");
                else if(type == Blocks.GRAY)
                        return TetrisImages.getImage("gray.gif");
                else if(type == Blocks.YELLOW)
                        return TetrisImages.getImage("yellow.gif");
                else if(type == Blocks.MAGENTA)
                        return TetrisImages.getImage("magenta.gif");
                else
                        return null;
        }
}
```

6.3 TetrisImages.java

```
package Logic;

import java.awt.*;
import java.awt.image.*;
import java.util.Hashtable;
import java.net.URL;
import java.net.URLClassLoader;


public class TetrisImages extends Component {

        private String[] names = {     "background.gif", "red.gif", "darkblue.gif",
"cyan.gif",

        "green.gif", "gray.gif", "yellow.gif", "magenta.gif"};
        private Hashtable cache;

        private static TetrisImages instance;

        public static void preLoad() {
                setInstance();
        }

        private TetrisImages() {
                cache = new Hashtable(names.length);
                for (int i = 0; i < names.length; i++) {
                        cache.put(names[i], loadImage(names[i]));
                }
        }

        private static void setInstance() {
                if(instance == null)
                        instance = new TetrisImages();
        }

        public static Image getImage(String name) {
        setInstance();
        if (instance.cache != null) {
                                if(instance.cache.containsKey(name))
                return (Image)instance.cache.get(name);
            else {
                Image img = instance.loadImage(name);
                instance.cache.put(name, img);
                return img;
            }
        }
        return null;
        }

        protected Image loadImage(String name) {
                URLClassLoader urlLoader =
(URLClassLoader)this.getClass().getClassLoader();
                URL fileLoc = urlLoader.findResource("images/" + name);
                Image img = this.getToolkit().createImage(fileLoc);

                MediaTracker tracker = new MediaTracker(this);
                tracker.addImage(img, 0);
                try {
                        tracker.waitForID(0);
                        if (tracker.isErrorAny()) {
                        System.out.println("Error loading image " + name);
                        }
                } catch (Exception ex) { ex.printStackTrace(); }
                return img;
        }
}
```

6.4 Timer.java

```
package Logic;

import EventInterface.*;

public class Timer implements Runnable {
        protected volatile Thread timer;
        protected IEventListner listner;
        protected int sleepTime;

        public Timer(IEventListner listner, int sleepTime) {
                this.listner = listner;
                this.sleepTime = sleepTime;
        }

        public void setSleepTime(int sleepTime) {
                this.sleepTime = sleepTime;
        }

        public void start() {
                timer = new Thread(this);
                timer.start();
        }

        public void stop() {
                timer = null;
        }

        public void run() {
                Thread me = Thread.currentThread();
                while (timer == me) {
                        try {
                                Thread.currentThread().sleep(sleepTime);
                        }
                        catch (InterruptedException e) { }
                        listner.incomingEvent(IEventListner.TIMER);
                }
        }
}
```

6.5 BlockPanel.java

```
package Gui;

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

public class BlockPanel extends JComponent {

        protected final int BLOCKSIZE = 15;
        protected int lines;
        protected int columns;
        protected int[][] blockLines;
        protected String backGroundFile;

        public BlockPanel(int columns, int lines, String backGroundFile) {
                this.lines = lines;
                this.columns = columns;
                this.backGroundFile = backGroundFile;

                blockLines = new int[columns][lines];
                for(int x = 0; x < columns; x++) {
                        for(int y = 0; y < lines; y++) {
                                blockLines[x][y] = Blocks.EMPTY;
```

```
                        }
                }
                //setBackground(Color.gray);
                //setSize( 100, 100);


        }

        public void paintComponent(Graphics g) {
                super.paintComponent(g);

                if(blockLines == null)
                        return;

                if(!backGroundFile.equals("")){
                        Image img = TetrisImages.getImage(backGroundFile);
                        g.drawImage(img, 0, 0, (ImageObserver)this);
                }
                else {
                        //setBackground(Color.BLACK);
                        g.setColor(Color.BLACK);
                        g.fillRect(0, 0, columns * BLOCKSIZE + 10, lines * BLOCKSIZE
+ 10);
                }
                for(int x = 0; x < columns; x++) {
                        for(int y = 0; y < lines; y++) {
                                if(blockLines[x][y] != Blocks.EMPTY){
                                        // Draw colored blocks

        //g.setColor(Blocks.typeToColor(blockLines[x][y]));
                                        //g.fill3DRect( x * BLOCKSIZE, y *
BLOCKSIZE, BLOCKSIZE, BLOCKSIZE, true);
                                        // Draw images
                                        Image img =
Blocks.typeToImage(blockLines[x][y]);
                                        g.drawImage(img, x * BLOCKSIZE + 5, y *
BLOCKSIZE + 5, (ImageObserver)this);
                                }
                        }
                }

        }
        /**
        * Takes an array and update the panel from it.
        */
        public void setBlocks(int[][] blockLines) {
                this.blockLines = blockLines;
                repaint(new Rectangle(getPreferredSize()));
        }

        public void setBlock(int col, int line, int type) {
                blockLines[col][line] = type;
                repaint(new Rectangle(getPreferredSize()));
        }

        public Dimension getMinimumSize() {
                return getPreferredSize();
        }

        public Dimension getMaximumSize() {
                return getPreferredSize();
        }

        public Dimension getPreferredSize() {
                return new Dimension(columns * BLOCKSIZE + 10, lines * BLOCKSIZE +
10);
        }
}
```

6.6 TetrisGUI.java

```
package Gui;

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

import java.beans.*;

import Logic.*;
import EventInterface.*;

public class TetrisGUI extends JPanel {
	static JFrame theFrame;

	public BlockPanel gamePanel;
	protected IEventListner listner;

	public TetrisGUI(IEventListner pListner, int gameSizeX, int gameSizeY) {
		this.listner = pListner;
		gamePanel = new BlockPanel(gameSizeX, gameSizeY, "background.gif");
		add(gamePanel);
		//gamePanel.setLocation( 75, 75);

		AbstractAction actionUp = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.UP);
				//System.out.println("Key: Up");
			}
		};

		AbstractAction actionDown = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.DOWN);
				//System.out.println("Key: Down");
			}
		};

		AbstractAction actionLeft = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.LEFT);
				//System.out.println("Key: Left");
			}
		};

		AbstractAction actionRight = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.RIGHT);
				//System.out.println("Key: Right");
			}
		};

		AbstractAction actionNewGame = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.NEWGAME);
				//System.out.println("Key: Right");
			}
		};

		AbstractAction actionPause = new AbstractAction() {
			public void actionPerformed(ActionEvent e) {
				listner.incomingEvent(IEventListner.PAUSE);
				//System.out.println("Key: Right");
			}
		};
		try{
```

```
                        KeyStroke strokeUp = KeyStroke.getKeyStroke(KeyEvent.VK_UP,
0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokeUp, "UP");
                        gamePanel.getActionMap().put("UP", actionUp);

                        KeyStroke strokeDown =
KeyStroke.getKeyStroke(KeyEvent.VK_DOWN, 0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokeDown, "DOWN");
                        gamePanel.getActionMap().put("DOWN", actionDown);

                        KeyStroke strokeLeft =
KeyStroke.getKeyStroke(KeyEvent.VK_LEFT, 0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokeLeft, "LEFT");
                        gamePanel.getActionMap().put("LEFT", actionLeft);

                        KeyStroke strokeRight =
KeyStroke.getKeyStroke(KeyEvent.VK_RIGHT, 0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokeRight, "RIGHT");
                        gamePanel.getActionMap().put("RIGHT", actionRight);

                        KeyStroke strokeNewGame =
KeyStroke.getKeyStroke(KeyEvent.VK_F2, 0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokeNewGame, "NEWGAME");
                        gamePanel.getActionMap().put("NEWGAME", actionNewGame);

                        KeyStroke strokePause =
KeyStroke.getKeyStroke(KeyEvent.VK_F3, 0);
                        gamePanel.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
).put(strokePause, "PAUSE");
                        gamePanel.getActionMap().put("PAUSE", actionPause);
                }
                catch(Exception e) {
                        System.out.println("Problem setting up key events: " + e);
                }
        }

        public static TetrisGUI start(IEventListner listner, int gameSizeX, int
gameSizeY) {
                TetrisGUI gui = new TetrisGUI(listner, gameSizeX, gameSizeY);
                theFrame = new JFrame("AspectTetris 1.0");
                theFrame.addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {System.exit(0);}
                });
                //theFrame.getContentPane().add("Center", gui);
                theFrame.getContentPane().add(gui);
                theFrame.pack();
                theFrame.setSize(300, 400);
                theFrame.setVisible(true);


                return gui;
        }
}
```

6.7 IEventListner.java

```
package EventInterface;

public interface IEventListner {

        public static final int UP              = 1;
        public static final int DOWN            = 2;
        public static final int LEFT            = 3;
        public static final int RIGHT           = 4;

        public static final int TIMER           = 5;

        public static final int NEWGAME = 6;
        public static final int PAUSE           = 7;

        // This method is called when a new events is fired.
        public void incomingEvent(int eventType);

}
```