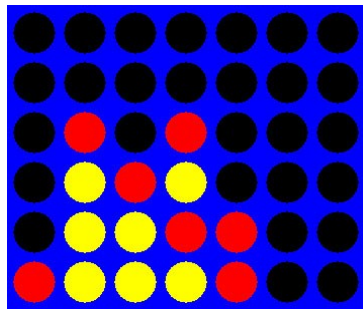


Dozent : Michael Reiser
Student : Markus Hediger

1. Einleitung
2. Handy-Programmierung
3. Algorithmus des Vier Gewinnt
4. Keine Fließkomma-Berechnungen möglich im CLDC 1.0
5. Umwandlung in ein Midlet
6. Schlusswort
7. Anhang

1. Einleitung

Nachdem ich die Seminarvorlesung „MIDlets“ bei Herrn Reiser besucht hatte, habe ich mich für eine Seminararbeit entschieden. Um MIDlets zu programmieren, die beispielsweise auf javafähigen Mobiltelefonen laufen, bedient man sich der Java Micro Edition (J2ME Micro Edition). Mein Ziel war es ein kleines Spiel zu implementieren, um mich mit der J2ME-Programmierung auseinandersetzen zu können. Es sollte das „Vier Gewinnt“-Spiel sein.



Beim „Vier Gewinnt“-Spiel gibt zwei Mitspieler (Rot und Geld), die einer nach dem anderen Steine setzen können. Es geht darum, auf einem Spielfeld von sieben mal sechs Feldern vier Steine als erster nebeneinander reihen zu können. Das Spielfeld steht aufrecht auf dem Tisch, so dass man bei den sieben vertikalstehenden „Slots“ einen Stein setzen kann, und der sich entweder aufs unterste Feld oder auf einen Stein, der bereits dort eingeworfen wurde, setzt. Die vier gewinnenden Steine müssen direkt nebeneinander zu liegen kommen, entweder vertikal, horizontal oder diagonal. Wie man beim Bild im Text sieht, hat Rot mit vier diagonal-stehenden Steinen gewonnen.

Wie bei einem Schachspiel, soll man auch hier gegen den Computer, sprich das Handy, spielen können. Da es bei diesem Seminar um MIDlets Programmierung handelt und nicht etwa um Implementierung von Spiel-Algorithmen sei hier angemerkt, dass ich den MiniMax-Algorithmus, mit dem der Rechner seine Züge berechnet, nicht selber implementiert habe. Dabei hat mir ein bestehendes Applet als Grundlage gedient. In diesem „Vier Gewinnt“-Applet war der Algorithmus bereits programmiert, so dass ich mich auf die J2ME-Implementierung konzentrieren konnte, was auch das Ziel in diesem Seminar war.

2. Handy-Programmierung

Ziel war also ein Programm fürs „Handy“. Damit dieses Programm auf möglichst vielen Mobiltelefonen – vor allem auch auf alten Handy - funktioniert, habe ich darauf geachtet, dass Standards wie CLDC 1.0 und MIDP 1.0 verwendet werden :

- Die CLDC-Konfiguration (Connected Limited Device Configuration) ist eine abgespeckte Java Virtual Machine (JVM). Dabei handelt es sich um eine Laufzeitumgebung, die auf dem Gerät installiert sein muss. Man spricht dann von javafähigen Mobile Geräten. Wenn man sich der CLDC 1.0 bedient, muss man jedoch gewisse Abstriche machen : Sie kennt zum Beispiel keine Fließkomma-Arithmetik. Mein Handy (Nokia 3410) z. B. unterstützt nur CLDC 1.0. Deshalb habe ich auch CLDC 1.0 gewählt.
- Beim Begriff MIDP handelt es sich um eine ergänzende Klassenbibliothek zur CLDC zur Erstellung von umfangreichen Anwendungen, wie beispielsweise Graphic User Interfaces (GUI's) und Games (2D), Medien, Kommunikation und Persistenz. Auch hier sollte die erste Version MIDP 1.0 zum Einsatz kommen, damit die Anzahl Mobiltelefone steigt, auf denen das „Vier Gewinn“ funktionieren sollte. Die MIDP 2.0 hat Vorteile wie Trusted MIDlets und die Push Registry, mit der man Midlets automatisch bei bestimmten Ereignissen starten kann. In Hinblick auf die Implementierung von einem „Vier Gewinn“-Spiel, ist der Unterschied zu MIDP 1.0 nicht weiter von Belang.

Zur Programmierung des Projekts kam das „SUNs J2ME Wireless Toolkit“ (J2ME) zum Einsatz. Sie lässt sich elegant aus der Programmierumgebung Eclipse bedienen.

3. Algorithmus des Vier Gewinn

Aufgrund der Tatsache, dass die CLDC 1.0 keine Fließkomma-Arithmetik unterstützt, muss gewährleistet sein, dass beim Berechnen des nächsten Zugs vom Computer keine Fließkomma-Berechnungen (Double, Float) gemacht werden. Um den Computer dazu zu bringen, dass er von sich selber aus einen Zug macht, gibt es Algorithmen, die der Computer durchspielen kann. Er macht verschiedene mögliche Züge und wählt dann den besten aus, so dass er zum einen den Gegner (Mensch) vor dem Gewinnen hindert und sich selbst in die Position des Gewinner setzt.

Wie schon in der Einleitung erwähnt, habe ich den Algorithmus für die Züge des Computers nicht selber implementiert. Als Grundlage habe ich mich des „Vier Gewinn“-Applets von *W. Westje & M. Dom* © 1999-2004 bedient. Dieses Applet und vor allem die Implementierung des Algorithmus wurde von Michael Dom (<http://www.minet.uni-jena.de/~dom/privat/index.html>) entwickelt. Der Entwickler stellt den Algorithmus aber nicht einfach zur Verfügung. Ich habe mich eines Java Decompilers bedient (<http://www.kpdus.com/jad.html>), der zum privaten Gebrauch legal ist. Mit diesem Tool konnte ich die CLASS-Files von Herrn Dom „decompilieren“ und hatte so den Quellcode vom „Vier Gewinn“-Applet. Es galt von nun an, dieses Applet in ein Midlet umzuschreiben. Es sei hier ausdrücklich angefügt, dass meine hier erstellte Midlet-Software nicht kommerziell verwendet werden kann, dies natürlich aus Urheberrechtsgründen. Nur im Rahmen dieses Seminars habe ich den Quell-Code von Herrn Dom im Sinne von Reverse Engineering benützt. Selbst ein Download dieses Midlets auf meiner Homepage anzubieten, wäre aus Urheberrechtsgründen widerrechtlich, denn bei der Implementierung von Herrn Dom wurde der AlphaBeta-Algorithmus optimiert und ist somit sein geistigen Eigentum. Ich möchte dem noch hinzufügen, dass der Algorithmus auch wirklich gut ist, im Gegensatz zu vielen anderen Algorithmen, die in gängigen „Vier Gewinn“-Spielen implementiert sind.

Das Programm ist in vier Klassen aufgeteilt (Quellcode im Anhang) :

- Meister.java (Implementierung des Algorithmus)
- Spielfeld.java (Definiert Spielfeld-Grösse und kontrolliert, ob gewonnen wurde)
- SpielfeldGrafik.java (Zeichnen des Spielfelds)
- Applet4GW.java (Eigentlicher Ablauf des Programms)

4. Keine Fließkommaberechnungen möglich im CLDC 1.0

Wie man in der Datei Meister.java sieht, hat es dort Berechnungen mit dem Datentyp Double, der eben ein Fließkommawert ist. Dieser Double-Wert wird im Algorithmus dazu verwendet, Fehlermöglichkeiten gegeneinander zu vergleichen. Damit nun das Programm mit CLDC 1.0 geschrieben werden konnte, die wie oben genannt keine Fließkommazahlen-Berechnungen unterstützt, mussten die Double-Werte durch Integer-Werte ersetzt werden (siehe Zeile 1409 bis 1449 in Meister.java, Anhang Kapitel 7). Mögliche Double-Werte in diesem Beispiel wären Werte von 0/7 bis 7/7. Ich habe nun diese Double-Werte durch Integer-Werte ersetzt und mit 100 multipliziert, beim Vergleich dann musste die Skala erweitert werden : nicht mehr 0 : 1, sondern 0 : 100. Schlussendlich vergleicht der Algorithmus nicht mehr 1/7 mit 2/7, sondern 100/7 (als Integer : 14) mit 200/7 (als Integer : 28). Mit diesem Trick konnte ich erreichen, dass der Algorithmus tatsächlich als CLDC 1.0-Anwendung implementiert werden konnte, weil keine weiteren Fließkommazahlen mehr im Code vorhanden waren (Siehe Anhang Kapitel 7.3). Der Rest der Anwendung galt es nun noch abzuändern, so dass sie schliesslich als Midlet im Wireless Toolkit und schliesslich auf dem Handy funktionieren würde.

5. Umwandlung in ein Midlet

Ähnlich wie bei einem Applet, muss die Grundform eines Midlets Methoden wie startApp(), pauseApp() und destroyApp() vorweisen, damit das Midlet zum Funktionieren kommt. Wie man im Anhang (Kapitel 7.1) sehen kann, wurde dies in der Datei 4GW.java getan.

Im Konstruktor VGW() wird das Display erzeugt, dann wird abgefragt, ob das Gerät Farben unterstützt oder nicht :

```
(Display) display = Display.getDisplay(this);  
(Boolean) colors = display.isColor();
```

Datei : VGW.java (Anhang Kapitel 7.1)

Es folgen nun zwei Abfragen, bevor das Spiel starten kann. Zum einen will man wissen, ob ein Spiel Mensch – Rechner oder Rechner – Mensch gespielt werden soll. Hat man entschieden, kann man noch verschiedene Schwierigkeitsstufen auswählen. Es gibt die Stufen Einfach, Mittel und Schwierig. Der Unterschied besteht in der Rechenzeit, die der Computer beim Ziehen zur Verfügung hat. Bei Stufe Einfach hat er 2.5 Sekunden Zeit, bei Stufe Schwierig hat er 7.5 Sekunden.

Die Abfragen werden durch Listen bewerkstelligt, die vom J2ME zur Verfügung gestellt werden :

```
list = new List("Spielarten:", List.IMPLICIT);  
display.setCurrent( list );
```

Datei : VGW.java (Anhang Kapitel 7.1)

Hat man diese Angaben erstmals, wird schliesslich ein Objekt aus SpielfeldGrafik-Klasse generiert und aufs Display gesetzt. Die Parameter 0 resp. Maximumtime sind die Abfragen, ob zuerst der Mensch beginnen soll, und was die maximale Rechenzeit sein soll :

```
spielflaeche = new SpielfeldGrafik(this, 0, maximumtime);
display.setCurrent( spielflaeche );
```

Datei : VGW.java (Anhang Kapitel 7.1)

In der Klasse SpielfeldGrafik resp. SpielfeldGrafikFarbig ist nun das eigentliche Midlet am Laufen.

```
class SpielfeldGrafikFarbig extends Canvas implements CommandListener, Runnable { .. }
```

In der Methode init() werden Objekte aus den Klassen Meister und Spielfeld angelegt, so dass das Spiel beginnen kann.

```
meister = new Meister();
partie = new Spielfeld();
```

Datei : SpielfeldGrafikFarbig.java (Anhang Kapitel 7.2)

In der Methode paint() wird das Spielfeld gezeichnet (Canvas). Im Konstruktor wird die Methode start() aufgerufen, die einen Thread erzeugt (Runnable). Dies bewirkt, dass der Rechner durch die Methode run() seine Berechnung für den nächsten Zug ausführt, sobald der Mensch wieder einen Stein gesetzt hat.

<pre>private synchronized void mannZug() { if(!darfZiehen) return; int i=0; if (schieber==(-3*schrittweite)) i=0; if (schieber==(-2*schrittweite)) i=1; if (schieber==(-schrittweite)) i=2; if (schieber==0) i=3; if (schieber==schrittweite) i=4; if (schieber==(2*schrittweite)) i=5; if (schieber==(3*schrittweite)) i=6; darfZiehen = false; status = partie.ziehen(amZug, i + 1); behandleStatus(); schieber=0; } private synchronized void rechnerZug() { if(rechnergiegner[amZug - 1] && status == 0) { meister.setzeSpielfeld(partie.getFeld()); meister.setzeSpielstaerke(maxZeit); meister.setzeGegnertyp(1); // oder 0 status = partie.ziehen(amZug, meister.besterZug(amZug)); behandleStatus(); schieber=0; } }</pre>	<pre>public void start() { if(rechnerAktivator == null) { rechnerAktivator = new Thread(this); rechnerAktivator.start(); } } public void run() { while(rechnerAktivator != null) { try { Thread.sleep(70L); } catch(InterruptedException _ex) { } if(rechnergiegner[amZug - 1] && status == 0) { removeCommand(Werfe); rechnerZug(); addCommand(Werfe); setCommandListener(this); } if(status == 1 status == 2 status == 3) { removeCommand(Werfe); } } }</pre>
--	--

Datei : SpielfeldGrafikFarbig.java (Anhang Kapitel 7.2)

In der Methode `mannZug()` wird zuerst rausgefiltert, wo sich der Schieber befindet und dann `'status = partie.ziehen(amZug, i + 1)'` übergeben. Das gleiche geschieht bei `rechnerZug()`, dort wird jedoch nicht `i+1` an `partie.ziehen(..)` übergeben, sondern `meister.besterZug(amZug)`. Mit `behandelteStatus()` wird bei jedem erfolgten Zug überprüft, ob das Spiel zu Ende ist (Siehe Datei `SpielfeldGrafik.java` im Anhang Kapitel 7.2).

Im Unterschied zum Applet, bei dem man per Mausklick einen Stein setzen kann, ist die Bedienung des Programms ausschliesslich durch Tastendruck auf dem Mobiltelefon möglich (`CommandListener`). Wenn nun also der Mensch am Zug ist, erscheint oberhalb des Spielfelds ein Dreieck, das als Schieber fungiert. Durch die linke und obere Pfeiltaste lässt sich der Schieber nach links bewegen, mit der rechten und unteren Pfeiltaste wird er nach rechts bewegt. Wenn der Mensch `SELECT` oder den Befehl „Werfe“ betätigt, wird der Stein geworfen.

```
(Command) Werfe = new Command("Werfe", Command.SCREEN, 1);
addCommand(Werfe);
setCommandListener(this);
```

bzw.

```
public void commandAction(Command c, Displayable d)
{
    if (c == cmExit) midlet.exitMIDlet();
    if (c == Werfe) {
        mannZug();
    }
}
```

```
protected void keyPressed(int keyCode) {
    switch (getGameAction(keyCode))
    {
        case FIRE: mannZug();
            break;
        case UP: schieber=schieber-schrittweite;
            break;
        case DOWN: schieber=schieber+schrittweite;
            break;
        case LEFT: schieber=schieber-schrittweite;
            break;
        case RIGHT: schieber=schieber+schrittweite;
            break;
    }
    repaint();
}
```

Datei : `SpielfeldGrafikFarbig.java` (Anhang Kapitel 7.2)

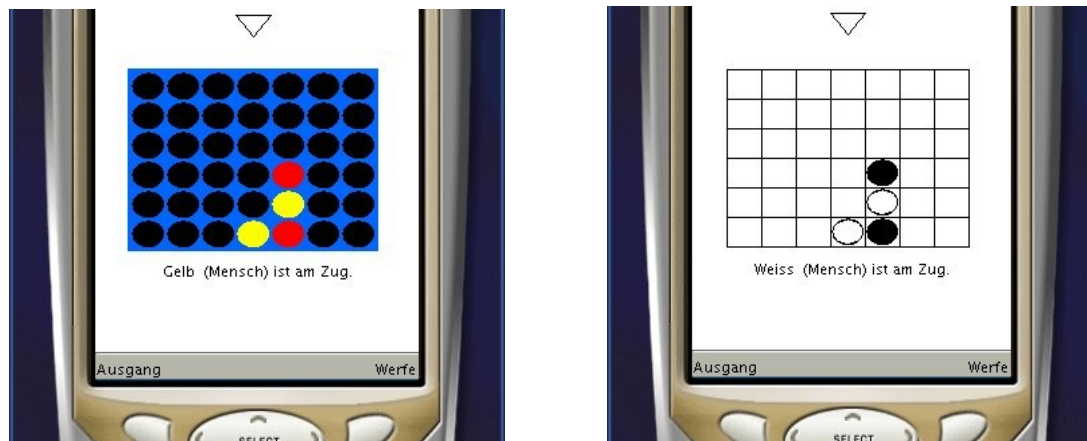
Eine noch zu bestehende Hürde, dieses Programm auf möglichst allen (javafähigen) Handys zum Laufen zu bringen, ist die dynamische Anpassung des Spielfelds an die Auflösung des jeweiligen Geräts. Es gibt da Auflösungen von 95x64 Pixels, aber auch von 352x288 Pixels. Um diese dynamische Anpassung vorzunehmen, liest das Midlet die Breite des Display per Befehl `getWidth()` aus, teilt diesen durch neun, da das Spielfeld 7 Spalten hat, damit haben wir auf der Seite jeweils eine „schrittweite“ zur Verfügung.

```
schrittweite=getWidth()/9;
schritthoehe=(schrittweite*7)/8;
```

Die „schritthoehe“ wird dann sieben Achtel der „schrittweite“ ausmachen. Wichtig ist hierbei, dass der Wert „schrittweite“ zuerst mit sieben multipliziert wird und erst dann mit acht dividiert, da es sich eben nicht um Fließkommazahlen handelt!

$30 \cdot 7 / 8$ ergibt 26.25, also 26 als Integerwert. Im Gegensatz dazu ergibt $(30/8) \cdot 7 = 21$, weil zuerst die Division $30/8 = 3.75$ den Integerwert 3 ausgibt, und dann mit 7 multipliziert wird. Wenn man nicht darauf achtet, kann es dazu kommen, dass die Spielfeld-Grafik auf gewissen Mobiltelefonen nicht in richtigen Dimensionen erscheint.

Nun haben wir ein funktionierendes Midlet, das im Wireless Tool Kit läuft. Wenn ein Handy keine Farben unterstützt, wird die Klasse SpielfeldGrafik ausgeführt, andernfalls wird SpielfeldGrafikFarbig ausgeführt.



Wie man sieht, heissen die Farben in farblosen Modus nicht Gelb und Rot, sondern Weiss und Schwarz.

6. Schlusswort

Es war eine interessante Erfahrung, sich mit dem J2ME auseinanderzusetzen, vor allem weil Mobile Devices wie Mobiltelefone von unserem heutigen Alltag nicht mehr wegzudenken sind. Ich habe jetzt ein Vier Gewinnt auf meinem Handy, das mich öfters schlägt, als mir lieb ist. Der implementierte Algorithmus vom Michael Dom ist wirklich gut. Weil er den Algorithmus optimiert hat, kann ich dieses Game nicht mal als Open Source auf meiner Homepage zur Verfügung stellen. Um dies zu tun, müsste ich wohl Herrn Dom höflich anfragen, ob er mir eine Art Lizenz erteilt. Der Code ist zwar nicht patentiert, aber urheberrechtlich geschützt. Da es im Internet genügend schon implementierte Algorithmen für derartige Spiele gibt, kann ich mir vorstellen, an einer Weiterentwicklung dieses Programms zu arbeiten. Dies ist aber nicht Grundlage dieses Seminars, in dem es über MIDlets-Programmierung geht.

Als erstes musste ich mich mit dem Ablauf des Programms als Applet einarbeiten, um dann schliesslich das ganze in ein J2ME-Hülle packen zu können. Auf gewissen Mobiltelefonen sieht man, dass die JVM (Java Virtual Machine) die Bilder nicht gerade schnell wiedergibt, so dass man die Grenzen von Java-Games aufgezeigt kriegt, selbst beim „Vier Gewinnt“. Da bei jedem Zug das Canvas neu gezeichnet wird, sieht man das bei gewissen Mobiltelefonen beim Aufbau des Bildes. Es fällt bei einem Spiel wie „Vier Gewinnt“ jedoch nicht so sehr ins Gewicht.

7. Anhang

7.1 VGW.java

Seite 8

7.2 SpielfeldGrafik.java

Seite 10

7.3 Meister.java

Seite 15

Da es den Rahmen dieses schriftlichen Dokuments sprengen würde, habe ich nicht den ganzen Quelltext aufs Papier gebracht. Der ganze Midlet, sowie auch den Applet-Code, steht in elektronischer Form beiliegend zur Verfügung. Es werden hier nur die wichtigsten Code-Schnitzel wie VGW.java, SpielfeldGrafikFarbig.java und ein Teil von Meister.java aufgeführt.

Das ganze Programm besteht aus folgenden Komponenten und ist mit der mitgelieferten CD elektronisch vorhanden:

Quellcode :

- Meister.java
- Spielfeld.java
- SpielfeldGrafik.java
- SpielfeldGrafikFarbig.java
- VGW.java

Binärcode und Descriptor :

- VierGewinnt.jar
- MANIFEST.MF
- VierGewinnt.jad

7.1 VGW.java

In der Datei VGW.java wird zuerst abgefragt, ob der Mensch oder der Rechner beginnen soll. In der zweiten Abfrage stellt man die Schwierigkeitsstufe :

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class VGW extends MIDlet implements CommandListener
{
    private Display display;
    private SpielfeldGrafik spielflaeche;
    private SpielfeldGrafikFarbig spielflaecheFarbig;

    private Command cmExit;
    private Command Starten;
    private Command Weiter;

    private List list;
    private String spielArt;
    private long maximumtime=5000L;
    private boolean colors;

    public VGW()
    {
        display = Display.getDisplay(this);
        colors = display.isColor();

        list = new List("Spielarten:", List.IMPLICIT);

        cmExit = new Command("Beenden", Command.EXIT, 1);
        list.addCommand(cmExit);
        list.setCommandListener(this);
        Weiter = new Command("Weiter", Command.SCREEN, 1);
        list.addCommand(Weiter);
        list.setCommandListener(this);

        list.append("Mensch - Rechner",null);
        list.append("Rechner - Mensch",null);
    }

    protected void startApp()
    {
        display.setCurrent( list );
    }

    protected void pauseApp()
    {}

    protected void destroyApp( boolean unconditional )
    {}

    public void exitMIDlet()
    {
        destroyApp(true);
        notifyDestroyed();
    }
}
```

```

public void commandAction(Command c, Displayable d)
{
    if (c == cmExit) exitMIDlet();
    if (c == Starten) {
        if (list.getString(list.getSelectedIndex()).equals("Einfach")) maximumtime=2500L;
        if (list.getString(list.getSelectedIndex()).equals("Mittel")) maximumtime=5000L;
        if (list.getString(list.getSelectedIndex()).equals("Schwer")) maximumtime=7500L;
        if ( spielArt.equals("Mensch - Rechner")) {
            if (colors) {
                spielflaecheFarbig = new SpielfeldGrafikFarbig(this, 0, maximumtime);
                display.setCurrent( spielflaecheFarbig );
            }
            else {
                spielflaeche = new SpielfeldGrafik(this, 0, maximumtime);
                display.setCurrent( spielflaeche );
            }
        }
        else {
            if (colors) {
                spielflaecheFarbig = new SpielfeldGrafikFarbig(this, 1, maximumtime);
                display.setCurrent( spielflaecheFarbig );
            }
            else {
                spielflaeche = new SpielfeldGrafik(this, 1, maximumtime);
                display.setCurrent( spielflaeche );
            }
        }
    }
    if (c == Weiter) {
        spielArt = list.getString(list.getSelectedIndex());
        list = new List("Stufen:", List.IMPLICIT);
        list.append("Einfach",null);
        list.append("Mittel",null);
        list.append("Schwer",null);
        list.removeCommand(Weiter);
        Starten = new Command("Starten", Command.SCREEN, 1);
        list.addCommand(Starten);
        list.setCommandListener(this);
        display.setCurrent( list );
    }
}
}
}

```

7.2 SpielfeldGrafikFarbig.java

Das eigentliche Spiel findet in der Datei SpielfeldGrafikFarbig.java statt :

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

class SpielfeldGrafikFarbig extends Canvas implements CommandListener, Runnable
{
    private Command cmExit;
    private Command Werfe;
    private Command NeuesSpiel;
    private String keyText = null;
    private VGW midlet;
    private int schieber=0;
    private int Breite, Hoehe;
    private int schrittweite, schritthoehe, abstandx, abstandy, xendpoint, yendpoint;
    private final int HOEHE = 6;
    private final int BREITE = 7;
    private int feld[][];

    private Meister meister;
    private Spielfeld partie;
    private int amZug;
    private int status;
    private String StatusText;
    private long maxZeit;
    private boolean darfZiehen;
    private boolean rechnergegner[];
    private Thread rechnerAktivator;

    public SpielfeldGrafikFarbig(VGW midlet, int welcheArt, long Zeit)
    {
        this.midlet = midlet;
        cmExit = new Command("Ausgang", Command.EXIT, 1);
        addCommand(cmExit);
        setCommandListener(this);
        Werfe = new Command("Werfe", Command.SCREEN, 1);
        addCommand(Werfe);
        setCommandListener(this);

        feld = new int[7][7];

        maxZeit = Zeit;
        rechnergegner = new boolean[2];

        init();
        start();

        if (welcheArt == 1) SpielartWechsel();
    }

    public void setFeld(int ai[][])
    {
        for(int i = 0; i < 7; i++)
        {
            for(int j = 0; j <= 6; j++)
                feld[i][j] = ai[i][j];
        }
    }
}
```

```

protected void paint(Graphics g)
{
    g.setColor(255, 255, 255);
    Breite=getWidth();
    Hoehe=getHeight();
    g.fillRect(0, 0, Breite, Hoehe);

    schrittweite=Breite/9;
    abstandx=schrittweite;
    xendpoint=abstandx+(7*schrittweite);

    schritthoehe=(schrittweite*7)/8;
    abstandy=((Hoehe-(6*schritthoehe))/2)+2;
    yendpoint=abstandy+(6*schritthoehe);

    g.setColor(0,100,255); // Blau
    // g.setFont(Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,Font.SIZE_MEDIUM));

    g.fillRect(abstandx-2, abstandy-2, (7*schrittweite)+4, (6*schritthoehe)+4);

    for (int i=0; i<7; i++) { // horizontal
        for (int j=1; j<=6; j++) { // vertikal
            switch(feld[i][j])
            {
                case 0: // '\0' Schwarz
                    g.setColor(0,0,0);
                    g.fillArc(abstandx+(i*schrittweite)+1,yendpoint-(j*schritthoehe)+1,schrittweite-2,schritthoehe-2,0,360);
                    break;

                case 1: // '\001' Gelb
                    g.setColor(255,255,0);
                    g.fillArc(abstandx+(i*schrittweite)+1,yendpoint-(j*schritthoehe)+1,schrittweite-2,schritthoehe-2,0,360);
                    break;

                case 2: // '\002' Rot
                    g.setColor(255,0,0);
                    g.fillArc(abstandx+(i*schrittweite)+1,yendpoint-(j*schritthoehe)+1,schrittweite-2,schritthoehe-2,0,360);
                    break;
            }
        }
    }

    if (schieber>(3*schrittweite)) schieber--(3*schrittweite);
    if (schieber<-(3*schrittweite)) schieber=3*schrittweite;

    // Bewegbares Dreieck
    g.setColor(0,0,0);
    if((rechnergegner[amZug - 1] && status == 0) || status==1 || status==2 || status==3) {}
    else {
        g.drawLine(abstandx+(3*schrittweite)+schieber,(abstandy/2)-2,abstandx+(3*schrittweite)+schrittweite+schieber,(abstandy/2)-2);
        g.drawLine(abstandx+(3*schrittweite)+schieber,(abstandy/2)-2,abstandx+(3*schrittweite)+(schrittweite/2)+schieber,(abstandy/2)+(2*schrittweite/3)-3);
        g.drawLine(abstandx+(3*schrittweite)+schrittweite+schieber,(abstandy/2)-2,abstandx+(3*schrittweite)+(schrittweite/2)+schieber,(abstandy/2)+(2*schrittweite/3)-3);
    }

    drawText(g);
}

private void drawText(Graphics g)
{
    g.drawString((StatusText),Breite/2,yendpoint+10,Graphics.HCENTER | Graphics.TOP);
}

```

```

public void commandAction(Command c, Displayable d)
{
    if (c == cmExit) midlet.exitMIDlet();
    if (c == Werfe) {
        mannZug();
    }
    if (c == NeuesSpiel) {
        spielBeginnen();
    }
}

protected void keyPressed(int keyCode)
{
    switch (getGameAction(keyCode))
    {
        case FIRE: mannZug();
            break;
        case UP: schieber=schieber-schrittweite;
            break;
        case DOWN: schieber=schieber+schrittweite;
            break;
        case LEFT: schieber=schieber-schrittweite;
            break;
        case RIGHT: schieber=schieber+schrittweite;
            break;
        default:
            keyText = getKeyName(keyCode);
    }
    repaint();
}

public void init()
{
    meister = new Meister();
    partie = new Spielfeld();
    rechnergegner[0] = false;
    rechnergegner[1] = true;
    new Long(0L);
    spielBeginnen();
}

public void start()
{
    if(rechnerAktivator == null)
    {
        rechnerAktivator = new Thread(this);
        rechnerAktivator.start();
    }
}

public void stop()
{
    rechnerAktivator = null;
}

```

```

public void run()
{
    while(rechnerAktivator != null)
    {
        try
        {
            Thread.sleep(70L);
        }
        catch(InterruptedException _ex) {}
        if(rechnergegner[amZug - 1] && status == 0) {
            removeCommand(Werfe);
            rechnerZug();
            addCommand(Werfe);
            setCommandListener(this);
        }
        if(status == 1 || status == 2 || status == 3) {
            removeCommand(Werfe);
        }
    }
}

private synchronized void spielBeginnen()
{
    darfZiehen = false;
    amZug = 1;
    status = 0;

    partie.init();
    setFeld(partie.getFeld());

    repaint();

    removeCommand(NeuesSpiel);
    addCommand(Werfe);
    setCommandListener(this);

    meister.resetHashtables();
    StatusText=(amZug != 1 ? "Rot (Rechner)" : "Gelb (Mensch)") + " ist am Zug.";
    if(!rechnergegner[amZug - 1])
        darfZiehen = true;
}

private synchronized void SpielartWechsel() {
    byte byte0;
    for(int ume=1; ume<=2; ume++) {
        if(ume==1)
            byte0 = 1;
        else
            byte0 = 2;
        if(rechnergegner[byte0 - 1])
        {
            rechnergegner[byte0 - 1] = false;
            if(amZug == byte0 && status == 0)
            {
                darfZiehen = true;
                return;
            }
        }
        else
        {
            rechnergegner[byte0 - 1] = true;
            if(amZug == byte0)
                darfZiehen = false;
        }
    }
    StatusText=(amZug != 1 ? "Rot" : "Gelb")+ " "+(darfZiehen==false && rechnergegner[amZug-1]==true ?
        "(Rechner) ueberlegt..." : "(Mensch) ist am Zug.");
    repaint();
}

```

```

private synchronized void mannZug() {
    if(!darfZiehen)
        return;
        int i=0;

        if (schieber==(-3*schrittweite)) i=0;
        if (schieber==(-2*schrittweite)) i=1;
        if (schieber==(-schrittweite)) i=2;
        if (schieber==0) i=3;
        if (schieber==schrittweite) i=4;
        if (schieber==(2*schrittweite)) i=5;
        if (schieber==(3*schrittweite)) i=6;

        darfZiehen = false;
        status = partie.ziehen(amZug, i + 1);
        behandleStatus();

        schieber=0;
}

private synchronized void rechnerZug()
{
    if(rechnergiegner[amZug - 1] && status == 0)
    {
        meister.setzeSpielfeld(partie.getFeld());
        meister.setzeSpielstaerke(maxZeit);
        meister.setzeGegnertyp(1); // oder 0
        status = partie.ziehen(amZug, meister.besterZug(amZug));
        behandleStatus();

        schieber=0;
    }
}

private synchronized void behandleStatus()
{
    switch(status)
    {
        case 0: // '\0'
            setFeld(partie.getFeld());
            repaint();
            amZug = (amZug & 1) + 1;
            StatusText=(amZug != 1 ? "Rot" : "Gelb")+ " "+(darfZiehen==false && rechnergiegner[amZug-1]==true ?
                "(Rechner) ueberlegt..." : "(Mensch) ist am Zug.");
            if(!rechnergiegner[amZug - 1])
                darfZiehen = true;
            return;

        case 1: // '\001'
            setFeld(partie.getFeld());
            repaint();
            StatusText="Gelb"+" ("+(darfZiehen==false && rechnergiegner[amZug-1]==true ? "Rechner" :
                "Mensch")+") hat gewonnen.";
            return;

        case 2: // '\002'
            setFeld(partie.getFeld());
            repaint();
            StatusText="Rot"+" ("+(darfZiehen==false && rechnergiegner[amZug-1]==true ? "Rechner" : "Mensch")+")
                hat gewonnen..";
            return;

        case 3: // '\003'
            setFeld(partie.getFeld());
            repaint();
            StatusText="Unentschieden.";
            return;
    }
}
}
}

```

7.3 Meister.java

Hier ein Teil der Datei Meister.java, wie sie im Midlet implementiert ist. Man achte auf die markierten Stellen, denn dort mussten die Abänderung fürs Midlet inbezug auf das Applet gemacht werden, da das Midlet bloss die MIDP 1.0 benützt. An den markierten Stellen war Double-Werte, die durch Integerwerte ersetzt wurden (siehe weiter oben im Kapitel 4) :

```
int d = 0;
if(flag)
{
    int k7 = (i6 & -2) + 2;
    if(i4 >= k7)
    {
        int ad[] = new int[7];
        int j8 = 0;
        int j9 = 0;
        int ad2[] = new int[7];
        berechnungsStop = false;
        timeoutZeit = l - 100L;
        int ai4[] = new int[7];
        //System.out.println("\nZugmoeglichkeiten des Gegners:");
        int j1 = 4;
        for(int i13 = 0; i13 < 7; i13++)
        {
            j1 += (i13 & 1) != 0 ? i13 : -i13;
            if(feld[j1 - 1][0] < 6 && !aflag[j1 - 1])
            {
                //System.out.print(" " + j1);
                ai4[j1 - 1] = gegnerMoeglichkeiten(j1, i, 3, k3);
                //System.out.print("(" + ai4[j1 - 1] + ")");
            }
        }

        if(!berechnungsStop)
        {
            int k13 = k7;
            do
            {
                //System.out.println("\nFehlermoeglichkeiten [Gegner gewinnt nicht] (" + k13 + " Halbzuege):");
                int k1 = 4;
                for(int j14 = 0; j14 < 7; j14++)
                {
                    k1 += (j14 & 1) != 0 ? j14 : -j14;
                    if(feld[k1 - 1][0] < 6 && !aflag[k1 - 1] && (!flag1 || ad[k1 - 1] != 0))
                    {
                        //System.out.print(" " + k1);
                        int k15 = ai4[k1 - 1] <= 0 ? 0 : gegnerFehlerMoeglichkeiten2(k1, i, k13, 3, k3);
                        int d4 = ai4[k1 - 1] <= 0 ? 100 : (k15*100) / ai4[k1 - 1];
                        ad2[k1 - 1] = d4;
                        //if(!berechnungsStop)
                        //System.out.print("(" + k15 + "/" + ai4[k1 - 1] + ")");
                    }
                }
            }
        }
    }
}
```

```

if(!berechnungsStop)
{
    d = 0;
    j9 = 0;
    int d3 = 0;
    for(int l1 = 1; l1 <= 7; l1++)
        if(feld[l1 - 1][0] < 6 && !aflag[l1 - 1] && (!flag1 || ad[l1 - 1] != 0))
            {
                ad[l1 - 1] = ad2[l1 - 1];
                if(ad[l1 - 1] > d)
                    {
                        d3 = d;
                        d = ad[l1 - 1];
                        j8 = l1;
                        j9 = 1;
                    }
                else
                    if(ad[l1 - 1] == d)
                        j9++;
                    else
                        if(ad[l1 - 1] > d3)
                            d3 = ad[l1 - 1];
            }

    if(k13 == k7)
        flag1 = true;
    if(j9 == 1 && d3 == 0)
        {
            //System.out.print("\nAbbruch: Eindeutiges Ergebnis gefunden.");
            //System.out.print("\nErgebnis: " + j8);
            return j8;
        }
}
if(berechnungsStop)
{
    //System.out.print("\nAbbruch: Timeout.");
    break;
}
if(d == 0)
{
    //System.out.print("\nAbbruch: Es existieren keine Fehlermöglichkeiten.");
    break;
}
if((k13 += 2) <= i4)
    continue;
//System.out.print("\nAbbruch: Spielfeld voll.");
break;
} while(true);
if(flag1 && j9 == 1)
{
    //System.out.print("\nErgebnis: " + j8);
    return j8;
}
if(flag1)
{
    for(int k14 = 0; k14 < 7; k14++)
        if(ad[k14] != d)
            aflag[k14] = true;
    }
}
}
}
}

```