Notizen und Übersichten zu

C#.Graphics

Ulrich Müller

Inhaltsverzeichnis

Graphics **3**

Allgemeines 3
Das Graphics-Objekt 3
Das Koordinatensystem und Transformationen 3
Pen und Brush 4
Die Graphics-Draw Methoden 4
Drucken von Graphiken 5
Literatur 5

Das Projekt DivGraph.CSproj **6**

Allgemeines 6
Programmrahmen 6
    Aufruf von ShowGraf 7
Die Basisklassen 8
    UrGraf 8
    Turtle 9
Graphiken – Beispiel : FarbQuadrate 10
Turtle-Graphik – TurtleRose 11
    Das WinLogo Original 12
Sierpinski-Graphik : Turte – Draw 13
    Turtle-Lösung 13
    Draw-Lösung 14

Turtle-Graphik **15**

Allgemeines 15
Referenz 16

Graphics

Allgemeines

Das Dokument soll anhand eines Beispiel-Programmes Hinweise zum Einsatz der Eigenschaften und Methoden des Graphics Objekts und des DrumRum aus dem Namensraum System.Drawing geben.

Dazu hier eine Übersicht der eingesetzten Graphics-Methoden und in den weiteren Abschnitten eine Beschreibung von Programmrahmen und Graphik-Programmen (Klassen).

Ausführlichere wird auf die Klasse Turtle zur Programmierung im Logo Turtle Stil eingegangen.

Das Graphics-Objekt

Gezeichnet wird auf einem (meist temporären) Graphik-Objekt eines graphikfähigen Controls (Form, PictureBox, aber auch Button). Das Objekt kann durch

```
Graphics g = pictureBox.CreateGraphics();
```

angelegt werden. Bei Aufruf in Eventroutinen eines Paint-Ereignisses ist es bereits angelegt und auf die angegebene Hintergrundfarbe gelöscht. Das Graphik-Objekt wird dann über das PaintEventArgs e des Paint-Events mit e.Graphics angesprochen.

Will man eine Graphic auf dem Drucker ausgeben tut man das am besten über das PrintPageEventArgs e (wieder e.Graphics) in der entsprechenden Ereignisroutine (siehe Kapitel "Drucken von Graphiken").

Das Graphik-Objekt kann durch

```
g.Clear(Color.farbName);
```

gelöscht werden.

Das Koordinatensystem und Transformationen

Koordinatennullpunkt ist standardmäßig die linke, obere Ecke einer Zeichenfläche. Maßeinheit : Pixel.

PageUnit : Einstellung der Maßeinheit. g.PageUnit = GraphicsUnit.GName (Display1/100 Zoll, Document 1/300 Zoll, Inch 25.4 mm, Millimeter, Pixel, Point 1/72 Zoll. Bis auf Pixel für Druckausgabe gedacht)

PageScale : Vergrößern/Verkleinern der gesamten Graphikausgabe einschl. Strichstärke und Schrifthöhe.

ScaleTransform : Vergrößern/Verkleinern. Sonderbehandlung für Texte.

TranslateTransform : Verschieben des Koordinatennullpunktes in x- und y-Richtung.

ResetTransform

Pen und Brush

Unterschieden wird zwischen der Klasse **Pen** zur Bestimmung von Farbe und Stärke von Strichen :

```
pyth.Stift = new Pen(Color.Blue, 1);
```

Der Zeichenstift des Objektes pyth (vom Typ Pen) bekommt ein neues Pen-Objekt mit der Stiftfarbe Blue und der Strichstärke 1 zugewiesen.

und der Klasse **SolidBrush** (und weiteren) zur Bestimmung der Farbe von Flächen :

```
canvas.FillRectangle(new SolidBrush(Color.Blue), 0, 0, 50, 50);
```

Es wird ein blaues Quadrat von der Größe 50x50 in die linke obere Ecke gezeichnet.

Die Farben werden über die statische Klasse **Color** bereitgestellt :

```
canvas.Clear(Color.Blue);
canvas.Clear(Color.Argb(27,45,103);
```

Der Hintergrund der Zeichenfläche wird auf blau gelöscht, im zweiten Anlauf auf eine Farbe eigener Anmischung.

Die Graphics-Draw Methoden

Linien werden über eine Vielzahl von Draw-Methoden erstellt. Meist gibt es auch noch passende Fill-Methoden zur Erstellung von Flächen. Die Anzahl der Überladungen ist teilweise beträchtlich.

DrawLine / DrawLines

Zeichnen einer einzelnen Linie / mehrere Linien

```
stift = new Pen(Color.Black, 1);
canvas.DrawLine(stift, 0, 0, 100, 100);
```

Es wird eine schwarze Linie in Stärke 1 von der linken, oberen Ecke zu den Koordinaten 100, 100 gezeichnet. Siehe Sierpinski.

```
canvas.DrawLines(new Pen(Color.Blue, 1), new PointF(x1, y1),
                                         new PointF(x2, y2),
                                        new PointF(x4, y4),
                                        new PointF(x3, y3),
                                        new PointF(x1, y1)));
```

Es wird ein geschlossenes Viereck (vielleicht ein Quadrat) in beliebiger Winkellage in blau gezeichnet. Die Punktkoordinaten wird als PoinF-Objekte mit float Werten vorgegeben. Siehe Pythagoras

DrawRectangle / DrawRectangles / FillRectangle / FillRectangles

Zeichnen eines / mehrerer Rechtecke parallel zu X- und Y-Achse

```
Pen stift = new Pen(Color.Red, 1);
canvas.DrawRectangle(stift, 100, 100, 50, 50);
```

Es wird ein Quadrat mit Kantenlänge 50 an Position 100,100 (linke, obere Ecke) mit roten Linien gezeichnet. FillRectangle siehe Lissajous.

DrawEllipse / FillEllipse

Zeichnen von Kreisen / Ellipsen – mit Linien, gefüllt.

```
canvas.DrawEllipse(new Pen(Color.Red, 1) 50, 50, 30, 30);
```

Es wird mit roter Linie ein Kreis in ein Quadrat mit Kantenlänge 30 gezeichnet, dessen linke obere Ecke auf 50,50 liegt.

Drucken von Graphiken

Das Drucken von Graphiken geschieht in einer Eventroutine (prtAus_PrintPage) eines Objekts (prtAus) von PrintDocument dessen Druck über prtAus-Print(); angestoßen wurde, es wird pro Event jeweils eine Seite gedruckt (siehe ShowGraf in frmMain).

Per Default wird mit GraphicsUnit.Display (1/100 Zoll pro Einheit) gedruckt.

```
cnavas.PageUnit = GraphicsUnit.Display;
```

Hier wird beim Graphics-Objekt `canvas` die Eigenschaft PageUnit auf GraphicsUnit.Display eingestellt. Eignet sich für normale Bildschirmausdrucke recht gut.

Literatur

Kühnel : Visual C# - Das umfassende Handbuch - Spracheinführung, OOP und Windows-Programmierung. Sehr verständlich : Graphik (Kapitel 22) / Drucken (Kapitel 23).
Galileo Computing. ISBN 3-89842-9 2.Auflage.

Das Projekt DivGraph.CSproj

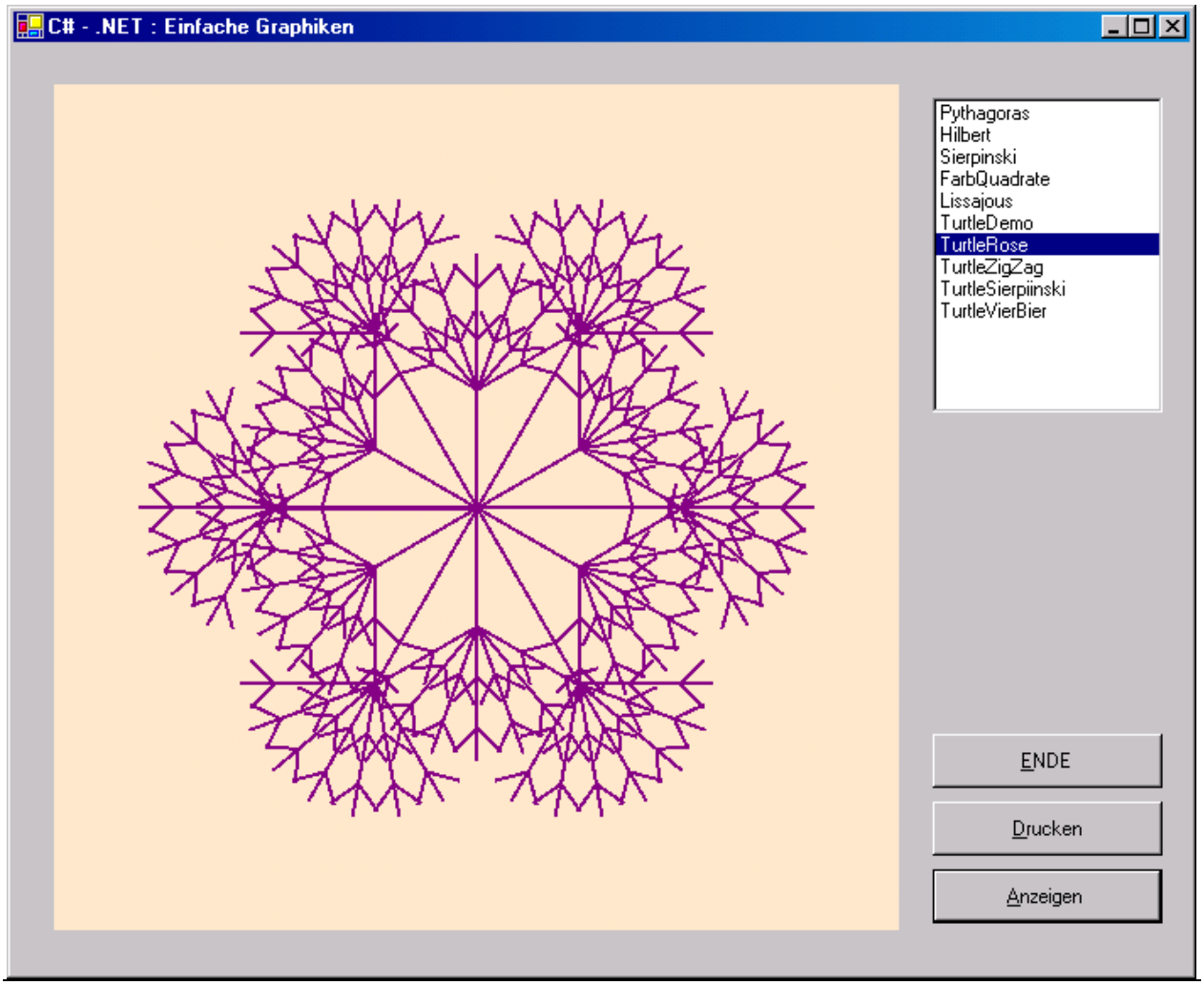
Allgemeines

DivGraph zeigt anhand einer Reihe – meist aus Lehrbüchern bekannter – Graphiken den Umgang mit Graphiken unter C# unter Nutzung des System.Drawing Namespaces.

Die einzelnen Graphiken sind jeweils in eigenen Klassen untergebracht, die ihrerseits wieder von Basisklassen abgeleitet werden. Interessant ist die Basisklasse Turtle, die die Graphik-Befehle (Turtle-Graphik) von Logo / WinLogo nahezu 1:1 auf eine C#-Klasse abbildet. Einige Klassen verwenden Zufallselemente um die Graphiken zu variiern.

Alle Graphiken können auch gedruckt werden.

Programmrahmen



Die Hauptform dient zum Anzeigen der in eigenen Klassen untergebrachten Graphiken in einer PictureBox (`picAus`).

Zentrales Element ist die Routine `ShowGraf(object sender, Graphics g)` zur Instanzierung der einzelnen Graphiken und dem Setzen spezieller Eigenschaften, außerdem wird zwischen der Anzeige in `picAus` und der Druckausgabe unterschieden :

```
private void ShowGraf(object sender, Graphics g) {
  switch (lstAuswahl.SelectedIndex) {
    case 0:
      Pythagoras pythagoras = new Pythagoras(picAus);
      pythagoras.Stift = new Pen(Color.Blue, 1);
      pythagoras.TanPi = 1.0F;
      if(sender.GetType().Name == "PrintDocument") {
        pythagoras.Background = Color.White;
        g.TranslateTransform(150, 150);}
      else {pythagoras.Background = Color.AntiqueWhite;}
        pythagoras.Action(g);
        break;
    case 1:
    .......
```

Die Auswahl der Graphik erfolgt über die ListBox `lstAuswahl`. Im zugehörenden `case` wird die entsprechende Graphik-Klasse zunächst instanziert, einige spezielle Eigenschaften gesetzt und dann anhand `sender` nach Drucken oder Ausgabe in `picAus` differenziert. Bei Druckausgabe wird der Koordinatennullpunkt der Graphik mit `TranslateTransform` noch etwas in Richtung Seitenmitte verschoben (Maßeinheit hier `Display` = 1/100 Zoll). Bei Ausgabe in `picAus` gilt hier die linke obere Ecke als Koordinatennullpunkt und als Maßeinheit die Einheit `Pixel` (jeweils Default-Werte). Der eigentliche Aufruf der Graphik erfolgt über die Methode `Action` mit dem Parameter `g` für ein bereits instanziertes Graphik-Objekt.

Aufruf von ShowGraf

Der Aufruf von ShowGraf erfolgt entweder über die Click-Routinen von `lstAuswahl` bzw. Anzeigen / Drucken (indirekt) oder das Paint-Event von `picAus` :

```
private void picAus_Paint(object sender,
                          System.Windows.Forms.PaintEventArgs e) {
    ShowGraf(sender, e.Graphics);}
```

Bei Aufruf über die Paint-Routine existiert bereits ein Graphik-Objekt : `e.Graphics`, das als Parameter übergegeben wird.

```
private void cmdAction_Click(object sender, System.EventArgs e) {
       ShowGraf(sender, picAus.CreateGraphics());}
```

Bei Aufruf über eine Click-Routine muß für das Control in das gezeichnet werden soll erst noch ein Graphik-Objekt erzeugt werden : `picAus.CreateGraphics();`

```
private void cmdDrucken_Click(object sender, System.EventArgs e) {
       prtAus.Print();}

private void prtAus_PrintPage(object sender,
       System.Drawing.Printing.PrintPageEventArgs e) {
       ShowGraf(sender, e.Graphics);}
```

Bei der Druckausgabe wird über die Click-Routine zunächst ein Druckauftrag ausgelöst, der dann seinerseits ein PrintPage-Event (für eine Seite) auslöst. Hier ist dann bereits ein Graphics-Objekt angelegt. `prtAus` ist ein PrintDocument-Objekt.

Die Basisklassen

UrGraf

Abstracte Basisklasse zur Erstellung einer Graphik. Die eigentliche Graphik muß von UrGraf abgeleitet werden. Die Klasse bietet ein allgemeines Control `ausgabe` zur Feststellung der Maße der Graphikfläche sowie Eigenschaften für `Background` und `Pen`. Die Graphik wird über die abstrakte Methode Action erstellt (wird der Parameter für die Zeichenfläche weggelassen, wird auf `ausgabe` gezeichnet :

```
public abstract class UrGraf    {
   protected Control  ausgabe;
   protected Graphics canvas;
   protected Pen      stift;
   protected Color    background;

   public UrGraf(Control Ausgabe)      {
      this.ausgabe    = Ausgabe;
      this.stift      = new Pen(Color.Black, 1);
      this.background = Color.White;
   }
   public Pen Stift {
      get {return stift;}
      set {stift = value;}
   }
   public Color Background {
      get {return background;}
      set {background = value;}
   }
   public virtual void Action() {
      Action(ausgabe.CreateGraphics());
   }
   public abstract void Action(Graphics Canvas);
}
```

In der abgeleiteten Klasse werden die Draw-/Fill-Methoden zum Erstellen der Graphik verwendet.

Turtle

Basisklasse zur Erstellung von Turtle-Graphiken. Die Graphiken können sowohl in einer abgeleiteten Klasse (wie hier geschehen) als auch unter Nutzung einer Instanz von Turtle erstellt werden. Die Klasse Turtle bietet den bei der Programmiersprache Logo zur Erstellung von Turtle-Graphiken üblichen Befehlssatz – in deutscher und englischer Version – die Schreibweise wurde den Gegebenheiten von C# angepaßt. Das Koordinatensystem liegt hier mit 0,0 in der Bildmitte (Version Abelson, WinLogo verwendet die linke untere Ecke), Es werden Winkelgrade verwendet :

```
public class Turtle {
  private const float WR = 180F / (float)Math.PI;

    protected Control  ausgabe;
    protected Graphics canvas;
    protected Pen      stift;
    protected Color    background;
    private   bool     justStarted;

    private bool  stiftState;
    private float currentX;
    private float currentY;
    private float currentDir;

    private Random rnd;

    public Turtle(Control Ausgabe) {
      this.ausgabe     = Ausgabe;
      this.stift       = new Pen(Color.Black, 1);
      this.background  = Color.White;
      this.justStarted = true;
    }

    public void Draw(Graphics Canvas) {Bild(Canvas);}
    public void Draw(Graphics Canvas, Color CName) {
     Bild(Canvas, CName);}
    public void Bild(Graphics Canvas) {Bild(Canvas,
                    this.background);}
    public void Bild(Graphics Canvas, Color CName) {
      this.canvas = Canvas;
      this.canvas.Clear(CName);
      if(justStarted) {
        this.canvas.TranslateTransform(this.ausgabe.Width/2,
                                       this.ausgabe.Height/2);
        justStarted = false;
      }
      this.stiftState = false;
      this.currentX = 0;
      this.currentY = 0;
      this.currentDir = 0;
    }
    public void ClearScreen() {LoescheBild();}
    public void LoescheBild() {
      this.canvas.Clear(this.background);
    }
..........
```

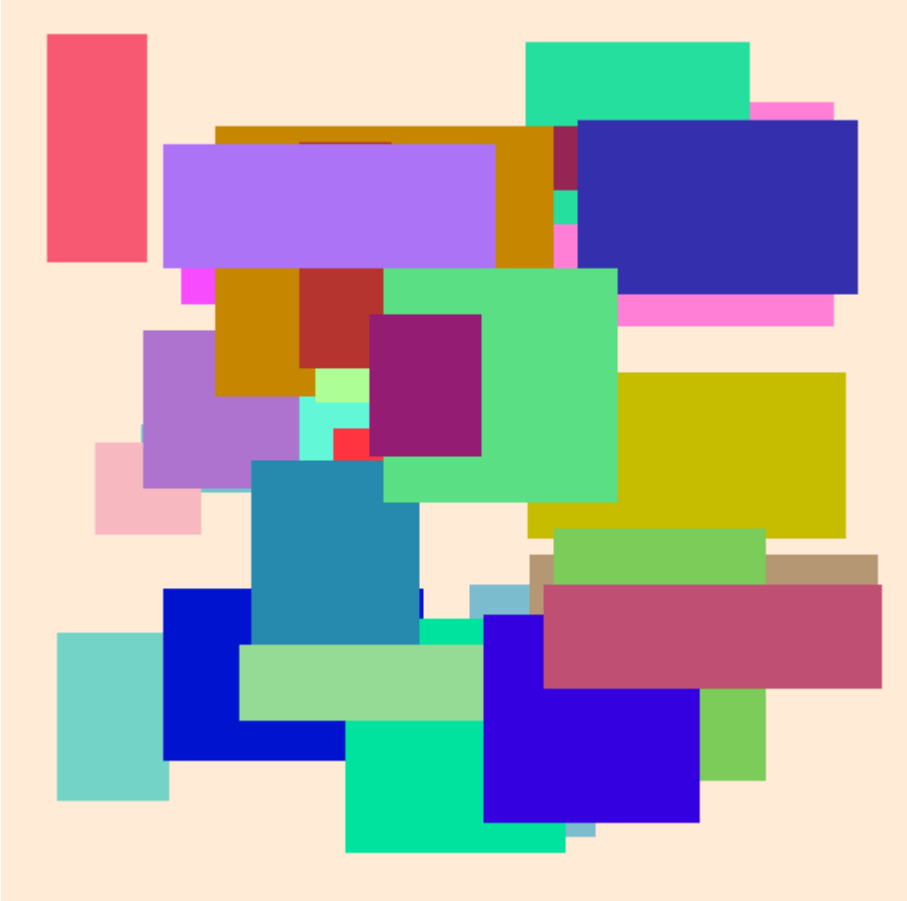
Turtle bildet die Befehle der Turtle-Graphik auf die Draw-Methoden von Graphics ab. Da die Draw-Methoden immer vollständige Positionsangaben (bei Line z.B. Von- und Zielkoordinaten. Turtle aber mit relativen Koordinaten arbeitet, werden eine Reihe von globalen Variablen zur Beschreibung des aktuellen Status erforderlich.

```
this.canvas.TranslateTransform(this.ausgabe.Width/2,
                               this.ausgabe.Height/);
```

verschiebt den Koordinatennullpunkt in die Bildmitte, da das dauerhaft geschieht, hier nur beim ersten Aufruf von Draw.

Eine Referenz aller Befehle der Turtle-Graphik findet sich ein einem besonderen Abschnitt.

Graphiken – Beispiel : FarbQuadrate



Ausgabe von farbigen Rechtecken in zufälliger Größe und Anordnung :

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace DivGraph {
  public class FarbQuadrate : UrGraf {

    const int maxBreite = 180;
    const int maxHoehe  = 140;

    SolidBrush Farbe;
    Random     rnd = new Random();
    int        X0, Y0, XB, YH;
    int        anzahl = 28;
```

```
    public FarbQuadrate(Control Ausgabe) : base(Ausgabe) {
    }
    public int Anzahl {
      get{return anzahl;}
      set{anzahl = value;}
    }
    public override void Action(Graphics Canvas) {
      base.canvas = Canvas;
      base.canvas.Clear(base.background);
      for(int i = 0; i < anzahl; i++) {
        Farbe = new SolidBrush(Color.FromArgb(rnd.Next(0,255),
             rnd.Next(0,255), rnd.Next(0,255)));
        X0 = rnd.Next(40, ausgabe.Width - maxBreite);
        Y0 = rnd.Next(30, ausgabe.Height - maxHoehe);
        XB = rnd.Next(40, maxBreite);
        YH = rnd.Next(32, maxHoehe);
        canvas.FillRectangle(Farbe, X0, Y0, XB, YH);
      }
    }
  }
}
```

Untergebracht in einem eigenen File, deswegen auch die using-Angaben. Abgeleitet von UrGraf. Zentrale Befehl FillRectangle dessen Parameter sämtlich über Zufallszahlen beeinflußt werden. Die Füllfarbe des jeweiligen Rechtecks wird über ein immer wieder neu erstelltes Objekt vom Typ SolidBrush bestimmt, dabei werden die einzelnen Farbwerte immer wieder neu "angerührt". Die Lage und die Größe werden in vorgegebenen Grenzen variiert.

Turtle-Graphik – TurtleRose

Ein Beispiel (Bild siehe oben), das immer wieder in Logo-Büchern zu finden ist :

```
public class TurtleRose : Turtle {
  public TurtleRose(Control Ausgabe) : base(Ausgabe) {}

  public void Action(Graphics Canvas) {
    Bild(Canvas);
    Canvas.ScaleTransform(2, 2);
    StiftAb(); PenColor(Color.Purple);
    Rose();
  }
  private void Vau() {
    Left(45); Forward(10);
    Back(10); Right(90);
    Forward(10); Back(10);
    Links(45);
  }
  private void Zweig() {
    Forward(15); Vau();
    Forward(15); Vau();
    Forward(10); Back(40);
  }
  private void Busch() {
    Left(60);
    for(int i = 0; i < 6; i++) {Zweig(); Right(20);}
    Zweig();
    Left(60);
  }
  private void Baum() {
```

```
      Forward(35); Busch(); Back(35);
   }
   private void Rose() {
      for(int i=0; i<6; i++) {Baum(); Right(60);}
      Rechts(30); Forward(60);
      for(int i=0; i<6; i++) {
                           Busch(); Back(60); Right(60); Forward(60);}
      Home(); AufKurs(0);
   }
}
```

Gestartet wird mit Action wo erstmal die Zeichenfläche gelöscht wird. In Ermangelung eines Logo-Befehls wird dann mit ScaleTransform die Größe der Zeichnung verdoppelt.

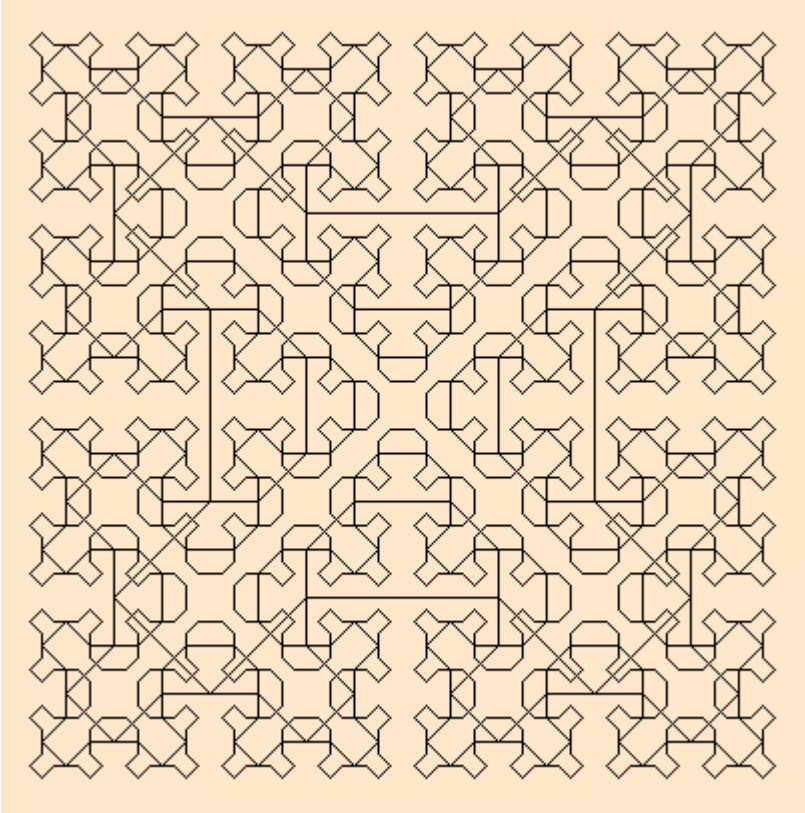
Dann kommen in schönster Logo-Manier Methoden mit den aufeinander aufbauenden Elementen der Graphik.

Das WinLogo Original

```
pr Baum
  vorwärts 35 Busch rückwärts 35
ende

pr Busch
  links 60
  wiederhole 6 [Zweig rechts 20]
  Zweig
  links 60
ende

pr Rose
  wiederhole 6 [Baum rechts 60]
  rechts 30 vorwärts 60
  wiederhole 6 [Busch rückwärts 60 rechts 60 vorwärts 60]
  mitte aufkurs 0
ende

pr Vau
  links 45 vorwärts 10 rückwärts 10
  rechts 90 vorwärts 10  rückwärts 10
  links 45
ende

pr Zweig
  vorwärts 15 Vau
  vorwärts 15 Vau
  vorwärts 10 rückwärts 40
ende
```

Damit es nicht zu einfach ist : mit deutschen Befehlen und in alphabetischer Anordnung der Routinen. Man sieht : Logo verzichtet auf Klammern und das Semikolon. Das wiederhole ist natürlich hier viel übersichtlicher als der etwas sperrige C# for Befehl.

Sierpinski-Graphik : Turte – Draw



Turtle-Lösung

```
public class TurtleSierpinski : Turtle {
  private int N;
  private float H0, H, X0, Y0;
  public TurtleSierpinski(Control Ausgabe) : base(Ausgabe) {}

  public void Action(Graphics Canvas) {
    N  = 4;
    H0 = 384;
    H  = H0 / 4;
    X0 = 0;
    Y0 = 100;
    Bild(Canvas); StiftHoch();
    SieRunde(1);
  }
  private void SieRunde(int i) {
    if(i > N) return;
    X0 -= H;
    H  /= 2;
    Y0 += H;
    PenUp(); AufXY(X0, Y0); StiftAb();
    SubA(i); AufXY(XKo()+H, YKo()-H);
    SubB(i); AufXY(XKo()-H, YKo()-H);
    SubC(i); AufXY(XKo()-H, YKo()+H);
    SubD(i); AufXY(XKo()+H, YKo()+H);
    SieRunde(i+1);
  }
  private void SubA(int i) {
```

```
    if(i < 1) return;
    SubA(i-1); AufXY(XKo()+H, YKo()-H); StiftAb();
    SubB(i-1); AufX(XKo()+2*H);
    SubD(i-1); AufXY(XKo()+H, YKo()+H);
    SubA(i-1);
  }
.......
```

Genutzt werden hier die absoluten Koordinatenangaben mit AufXY ... Hier taucht auch die bei Logo so beliebte Rekursion auf, Motto : Keine Schleifen, wenns auch rekursiv geht.

Draw-Lösung

```
public class Sierpinski : UrGraf {
  int     tiefe, breite;
  float   korr;
  float   H, X0, Y0, X1, Y1;

  public Sierpinski(Control picAus) : base(picAus) {
    tiefe = 5;
    breite = 384;
    korr =(base.ausgabe.Width - breite) / 2;
  }
  public override void Action(Graphics Canvas) {
    base.canvas = Canvas;
    base.canvas.Clear(base.background);
    H = breite / 4;
    X0 = 2 * H + korr;
    Y0 = 3 * H + korr;
    for(int i = 1; i <= tiefe; i++) {
      X0 -= H;
      H = H / 2;
      Y0 += H;
      X1 = X0;
      Y1 = Y0;
      subA(i); canvas.DrawLine(stift, X1, Y1, X1+H, Y1-H);
      X1 += H; Y1 -= H;
      subB(i); canvas.DrawLine(stift, X1, Y1, X1-H, Y1-H);
      X1 -= H; Y1 -= H;
      subC(i); canvas.DrawLine(stift, X1, Y1, X1-H, Y1+H);
      X1 -= H; Y1 += H;
      subD(i); canvas.DrawLine(stift, X1, Y1, X1+H, Y1+H);
    }
  }
  private void subA(int i) {
    if(i < 1) return;
    subA(i-1); canvas.DrawLine(stift, X1, Y1, X1+H, Y1-H);
    X1 += H;  Y1 -= H;
    subB(i-1); canvas.DrawLine(stift, X1, Y1, X1+2*H, Y1);
    X1 += 2*H;
    subD(i-1); canvas.DrawLine(stift, X1, Y1, X1+H, Y1+H);
    X1 += H; Y1 += H;
    subA(i-1);
  }
......
```

Recht ähnlich, es fallen die längeren Draw-Methoden mit nachfolgendem Speichern der aktuellen Position auf.

Turtle-Graphik

Allgemeines

Die Basisklasse Turtle bietet in Form von C# Methoden die Elemente der Turtle-Graphik von Logo an. Zugrunde liegt der Befehlsumfang nach : "Harald Abelson : Einführung in Logo", deutsch, iWT-Verlag 1983, ISBN 3-88322-023-X. Gedacht ist sie für Logo-Fans, die auchmal – ganz vorsichtig – eine aktuelle Programmiersprache ein wenig probieren wollen.

Das geht so : Der Programmrahmen einschließlich Variablen-Definitionen und Kontrollstrukturen ist reines C#. Eine Turtle-Graphik wird in Form einer Klasse geschrieben, die von der Basis-Klasse Turtle mit den Turtle-Graphik Befehlen von Logo abgeleitet sind. Dabei werden die Turtle-Graphik Befehle auf Methoden der Klasse Turtle abgebildet.

Für den Anfang reicht es, dem bestehenden Programmrahmen DivGraph ein neue Klassendatei hinzuzufügen, dazu kann ein bestehendes kopiert werden (z.B. TurtleDemo). Die Klasse bietet eine Zeichenflächen von 500 x 500 Einheiten (Pixeln) mit dem Koordinatenursprung 0,0 in der Mitte, also mit X- und Y-Werten von –250 bis +250 (entsprechend Abelson, WinLogo hat den Koodinatenursprung in der linken unteren Ecke, .NET in der linken oberen Ecke). Die Darstellung erfolgt einheitlich als 32bit-Gleitkommazahl (float), nur Anzahlen werden als 32bit-Integer (int) dargestellt. Gradangaben erfolgen in Grad (0 – 360°). Die Strichstärke der Turtle ist einheitlich 1. PenColor ist auf Color.Black und der Background auf Color.White voreingestellt.

Alternativ kann die Klasse Turtle natürlich auch als selbständiges Objekt eingesetzt werden.

Die Turtle-Graphik Befehle sind in deutsch und englisch verfügbar. Auf Abkürzungen der Befehle wurde aber verzichtet.

Referenz

AufKurs / SetHeading

Einstellung der Zeichenrichtung in Grad von 0 – 360. 0° : nach oben.

AufKurs(Winkel) / SetHeading(Angle)

Kein Rückgabewert

Beispiel :

```
SetHeading(90);
Forward(100);
```

Die Turtle wird um 100 Einheiten in der eingestellten Position (Auf/Ab) und Farbe waagerecht nach rechts bewegt.

AufX / SetX

Einstellung einer neuen X-Position, ggf. waagerecht zeichnend.

AufX(XKoor) / SetX(XCoor)

Kein Rückgabewert.

Beispiel :

```
PenColor(Color.Blue);
PenDown();
SetX(100);
```

Es wird ein waagerechter, blauer Strich von der augenblicklichen Position auf Position X = 100 gezeichnet.

AufY / SetY

Einstellung einer neuen Y-Position, ggf senkrecht zeichnend

AufY(YKoor) / SetY(YCoor)

Kein Rückgabewert

Beispiel :

```
PenUp();
SetY(-100);
```

Die Position Y = -100 wird angesteuert.

AufXY / SetXY

Einstellung einer neuen X/Y-Position, ggf. zeichnend.

AufXY(XKoor, YKoor) / SetXY(XCoor, YCoor)

Kein Rückgabewert.

Beispiel :

```
SetXY(-250, 250);
```

Die linke, obere Ecke wird angesteuert, bei PenDown wird ein Strich gezogen.

Bild / Draw

Herstellen der Ausgangsstellung (Löschen der Zeichenfläche, PenUp, Koordinaten 0,0, Winkel 0°). Muß zu Beginn einer Zeichnung aufgerufen werden.

Bild(Canvas [, CName]) / Draw(Canvas [, CName])

Canvas : Zeichenfläche, CName : Hintergrundfarbe
Kein Rückgabewert.

Beispiel :

```
private void picAus_Paint( .... PaintEventArgs e) {
     Draw(e.Graphics, Color.AntiqueWhite);
}
```

In der Routine für das Paint-Ereignis der Zeichenfläche picAus (eine PictureBox) wird die Zeichenfläche für die Turtle-Graphik auf Ausgangsstellung gebracht. Dazu werden das bereits instanzierte Objekt der Zeichenfläche (e.Graphics) und die Hintergrundfarbe AntiqueWhite als Parameter übergeben.

Farbe / PenColor

Festlegen der Farbe des Zeichenstifts für die nachfolgenden Aktionen.

Farbe(CName) / PenColor(CName)

Kein Rückgabewert.

Beispiel :

```
PenColor(Color.Blue);
```

Die nachfolgenden Striche werden in Blue gezeichnet.

Kurs / Heading

Feststellen der augenblicklichen Zeichenrichtung

Kurs() / Heading()

Rückgabe : augenblickliche Zeichenrichtung in Grad.

Beispiel :

```
SetHeading(Heading() - 90);
Forward(100);
```

Die aktuelle Zeichenrichtung wird um 90° gegen den Uhrzeigersinn geändert, der nachfolgende Strich in Länge 100 folgt dieser Richtung.

Links / Left

Ändern der augenblicklichen Zeichenrichtung entgegen dem Uhrzeigersinn

Links(Winkel) / Left(Angle)

Kein Rückgabewert.

Beispiel :

```
Left(90);
Forward(100);
```

Die aktuelle Zeicherichtung wird um 90° gegen den Uhrzeigersinn geändert, der nachfolgende Strich in Länge 100 folgt dieser Richtung (entspricht dem Heading Beispiel oben).

LoescheBild / ClearScreen

Löschen der Zeichenfläche in der eingestellten Hintergrundfarbe ohne Veränderung der sonstigen Einstellungen

LoeschBild() / ClearScreen()

Kein Rückgabewert.

Beispiel :

```
Background(Color.White);
ClearScreen();
```

Löschen der Zeichenfläche auf weißen Hintergrund.

Mitte / Home

Setzen der Koordinaten auf Bildmitte (0,0), kein Strich.

Mitte() / Home()

Kein Rückgabewert

Beispiel :

```
Home();
PenDown();
SetHeading(0);
Forward(100);
```

Es wird ein Strich von der Bildmitte, 100 Einheiten lang, in waagerechter Richtung nach rechts gezeichnet.

Rechts / Right

Ändern der augenblicklichen Zeichenrichtung im Uhrzeigersinn

Rechts(Winkel) / Right(Angle)

Kein Rückgabewert

Beispiel :

```
SetHeading(0);
Right(90);
PenDown();
Back(100);
```

Strich in Länge 100 waagerecht nach links. (Setzen der Zeichenrichtung nach oben, Ändern nach waagerecht rechts, Strich in Gegenrichtung.

Richtung / Towards

Festlegen der Zeichenrichtung von der augenblicklichen Position auf die angegebenen Koordinaten.

Richtung(XKoor, YKoor) / Towards(XCoor, YCoor)

Rückgabewert : Richtung in Grad.

Beispiel :

```
SetXY(0, 0);
SetHeading(Towards(250, 250));
PenDown();
Forward(100);
```

Strich in Länge 100 von Bildmitte in Richtung rechte, obere Ecke.

Rueckwaerts / Back

Bewegen der Turtle um die angegebenen Einheiten entgegen der vorgegebenen Richtung, ggf. zeichnend.

Rueckwaerts(Delta) / Back(Delta)

Kein Rückgabewert.

Beispiel :

```
SetXY(50, 50);
SetHeading(90);
PenDown();
Back(100);
Left(90);
Back(100);
Left(90);
Back(100);
Left(90);
Back(100);
```

Zeichnen eines Quadrates mit Kantenlänge 100 in Bildmitte im Rückwärtsgang.
Siehe auch Beispiel Vorwaerts.

StiftAb / PenDown

Zeichenstift senken

StiftAb() / PenDown()

Kein Rückgabewert.

Beispiel :

```
PenUp();
SetHeading(0);
Forward(100);
```

Strich in Länge 100 nach oben.

StiftHoch / PenUp

Zeichenstift heben.

StiftHoch() / PenUp()

Kein Rückgabewert.

Beispiel :

```
SetXY(-50, 50);
SetHeading(90);
PenDown();
Forward(100);
PenUp();
Right(90);
Forward(10);
Right(90);
PenDown();
Forward(100);
```

Zeichnen von zwei Parallelen im Abstand von 10 Einheiten von –50,50 nach 50,40

Vorwaerts / Forward

Bewegen der Turtle um die angegebenen Einheiten in der vorgegebenen Richtung, ggf. zeichnend.

Vorwaerts(Delta) / Forward(Delta)

Kein Rückgabewert.

Beispiel :

```
SetXY(50, 50);
SetHeading(180);
PenDown();
Forward(100);
Right(90);
Forward(100);
Right(90);
Forward(100);
Right(90);
Forward(100);
```

Zeichnen eines Quadrates mit Kantenlänge 100 in Bildmitte rechtsrum im Vorwärtsgang.

XKo / XCor

Feststellen des augenblicklichen Wertes der X-Koordinate.

XKo() / XCor()

Rückgabe : augenblicklicher Wert der X-Koordinate

Beispiel : siehe YKo

YKo / YCor

Feststellen des augenblicklichen Wertes der Y-Koordinate.

YKo() / YCor()

Rückgabe : augenblicklicher Wert der Y-Koordinate.

Beispiel :

```
SetXY(-50, 50);
SetHeading(90);
PenDown();
Forward(100);
PenUp();
SetY(YCor() - 10);
PenDown();
Back(100);
```

Zeichnen von zwei Parallelen im Abstand von 10 Einheiten von –50,50 nach 50,40. Siehe auch Beispiel PenUp.

StarteZufall / Randomize

Starte eine Folge von Zufallszahlen.

StarteZufall([Seed]) / Randomize([Seed])

Kein Rückgabewert. Wenn ein Ausgangswert (Seed) angegeben ist, wird die immer gleiche Serie von Zufallszahlen gestartet, sonst wird der Startwert von der Uhrzeit abgeleitet.

Beispie :

```
Randomize(107);
```

Start einer Zufallszahlenserie mit dem Startwert 107.

Zufallszahl / Random

Abruf der nächsten Zufallszahl.

Zufallszahl(n) / Random(n)

Rückgabewert : Zufallszahl (int) im Bereich 0 – n-1.

Beispiel :

```
SetX(Random(11));
```

Die Turtle bewegt sich waagerecht auf eine X-Koordinate zwischen 0 und 10.

Pause

Anhalten des Programmablauf um eine vorgegebenen Zeit.

Pause(ZSek) / Pause(TSec)

Kein Rückgabewert. Pausenzeit in 1/10 Sekunden.

Beispiel :

```
Pause(100);
```

Das Programm wird 10 Sekunden angehalten.