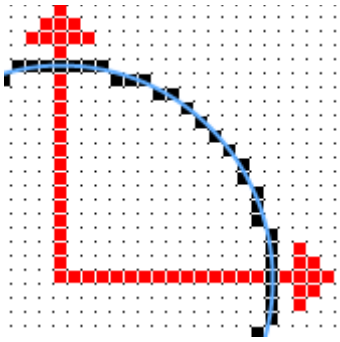
Plotten von Kreisen (nach Jack Bresenham) (C) Ac 3-1990 bis 2-2017

Wie zeichnet der Computer Kreise ? Natürlich mithilfe von Pixels, „picture elements“, also kleinen Bildpunkten. Die beste Approximation des Kreises ist durch diejenigen Pixels gegeben, die von der wahren Kreislinie den geringsten Abstand haben.



Betrachtet man die Pixels im Bereich zwischen 45° und 90° ,
so erkennt man, dass sie sich durch einen einfachen Algorithmus finden lassen:

```
Setze den ersten Punkt bei  x = 0  und  y = r ;   P(0/r)
WIEDERHOLE
   Erhöhe x um 1
   WENN P(x/y-1) näher an der Kreislinie liegt als P(x/y)
        DANN vermindere y um 1      (*)
   Setze den Punkt  P(x/y)
BIS x = y
```

Die Verfeinerung zu (*) ist nicht ganz einfach. Sie wird später betrachtet !

Mit jedem Punkt aus dem Bereich [45°;90°] kann man wegen der hochgradigen Symmetrie des Kreises 7 weitere Punkte plotten . Somit reicht der obige Algorithmus aus, um alle Kreispunkte zu plotten.

Kreisalgorithmus (Erläuterung weiter unten):

```
x = 0
y = r
delta = 3 - 2*r
zeichneAchtPunkte(x,y)    (**)
solange x < y wiederhole
   falls delta >= 0
      dann delta = delta + 4(x-y) + 10
           y = y - 1
        sonst delta = delta + 4x + 6
   x = x + 1
   zeichneAchtPunkte(x,y)
ende solange
```

(**) Es reicht hier aus, nur die 4 Punkte
(xm, ym+r) (xm, ym-r) (xm-r, ym) (xm+r, ym) zu zeichnen.

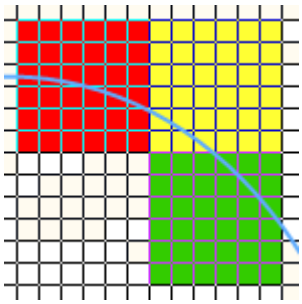
Algorithmus zeichneAchtPunkte(x,y): M = (xm, ym) = Kreismittelpunkt

```
Plot (xm + x, ym + y)
Plot (xm + x, ym - y)
Plot (xm - x, ym + y)
Plot (xm - x, ym - y)
Plot (xm + y, ym + x)
Plot (xm + y, ym - x)
Plot (xm - y, ym + x)
Plot (xm - y, ym - x)
```

Theorie: Erläuterung des Kreisalgorithmus

Zu lösen ist das Problem, wie man das jeweils nächste Pixel findet !

Pi(xi/yi) Si(xi+1/yi) = "äußerer Punkt" (gelb)



Ti(xi+1/yi-1) = "innerer Punkt" (grün)

Wir betrachten die "quadrierten Abstände" der oben angegebenen Punkte vom Ursprung O(0,0) und setzen:

$e = |OS_i|^2 - r^2 \geq 0$ für alle i

$f = |OT_i|^2 - r^2 < 0$ für alle i

$Delta = e + f$

Es gilt nun folgendes Kriterium:

Wenn T_i näher am Kreis liegt, dann ist Delta ≥ 0
Wenn S_i näher am Kreis liegt, dann ist Delta < 0

Im ersten Fall muss y um 1 vermindert werden, während y im zweiten Fall unverändert bleibt. Dies erklärt sich aus der Tatsache, dass der Betrag der Steigung des Kreises im betrachteten Winkelbereich immer zwischen 0 und 1 liegt. x wird natürlich bei jedem Schritt um 1 erhöht .

Für die quadrierten Abstände berechnen wir:

$|OS_i|^2 = (x_i+1)^2 + y_i^2$ und $|OT_i|^2 = (x_i+1)^2 + (y_i-1)^2$

Folglich ist $Delta_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$

Für i = 0 gilt: $x_o = 0$ und $y_o = r$

Es folgt daraus: $Delta_o = 3 - 2r$ Dies ist der Startwert für Delta; er ist immer negativ !!

Wie berechnen wir nun das nächste Delta ??

Falls $Delta_i < 0$, dann liegt S_i (der "äußere Punkt") näher am Kreis .
Dies bedeutet, dass lediglich x um 1 erhöht wird, während y unverändert bleibt .

Es gilt also:

$$Delta_{i+1} = 2(x_{i+1}+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$$

$$= 2[(x_{i+1})^2 + 2(x_{i+1}) + 1\ ] + y_i^2 + (y_i-1)^2 - 2r^2$$

$$= 2(x_{i+1})^2 + 4x_i + 4 + 2 + y_i + (y_{i-1})^2 - 2r^2$$

Es folgt:

$$\mathbf{Delta_{i+1} = Delta_i + 4x_i + 6}$$

Falls $Delta_i = 0$, dann liegt T_i (der "innere Punkt") näher am Kreis.
Daher wird x um 1 erhöht und y um 1 vermindert .

Dann gilt:

$$Delta_{i+1} = 2(x_{i+1}+1)^2 + (y_{i-1})^2 + (y_{i-1}-1)^2 - 2r^2$$

$$= \ldots\ldots\ldots$$

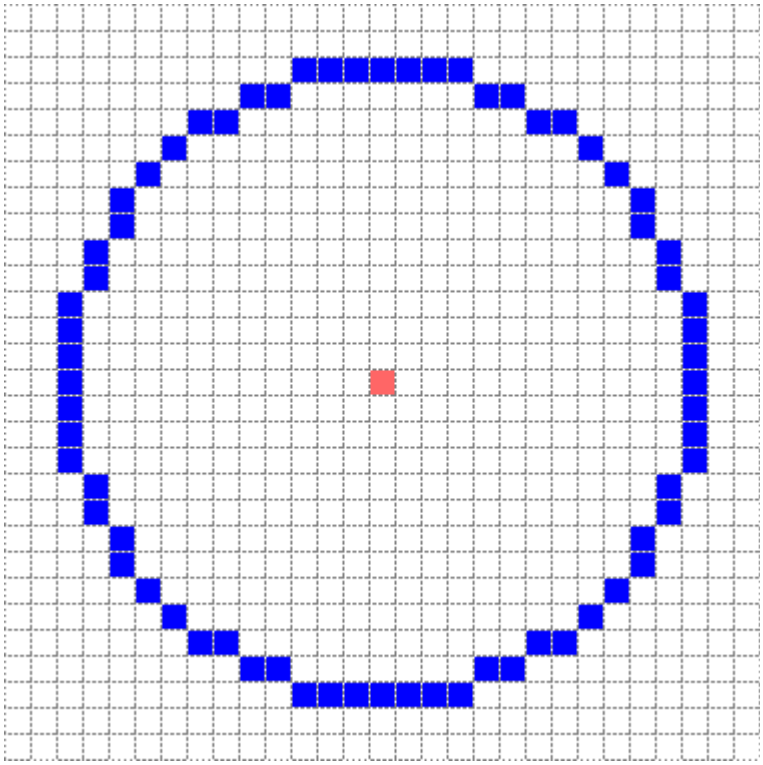
$$= \ldots\ldots\ldots$$

Schließlich:

$$\mathbf{Delta_{i+1} = Delta_i + 4(x_i-y_i) + 10}$$

Beispiel: xm = 15 ym = 19 r = 15

Der Kreismittelpunkt ist hier rot eingezeichnet !



JAVA-Programm:

```
import java.awt.*;
import java.awt.image.BufferedImage;

import javax.swing.*;

// Ac 24.02.2017

@SuppressWarnings("serial")
public class Bresenham extends JPanel {

   public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
       @Override
       public void run() {
             JFrame f = new JFrame();
             f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
             f.setTitle("Bresenham-Algorithmus für Kreise");
             f.setResizable(false);
             f.add(new Bresenham(), BorderLayout.CENTER);
             f.pack();
             f.setLocationRelativeTo(null);
             f.setVisible(true);
         }
    });
  }

   public static BufferedImage bufKos;
   public static Graphics2D grBufKos;


   // Konstruktor
   public Bresenham() {
      Dimension dim = new Dimension(640, 640);
      setPreferredSize(dim);
      setBackground(Color.white);
      bufKos = new BufferedImage(dim.width, dim.height, BufferedImage.TYPE_INT_RGB);
      grBufKos = bufKos.createGraphics(); // "Grafik-Kontext" des Puffers (für Graphics2D)
      grBufKos.setColor(Color.WHITE);
      grBufKos.fillRect(0, 0, dim.width, dim.height); // weißer Hintergrund für Puffer

      zeichneBresiKreis(300, 300, 120, Color.ORANGE);
      zeichneBresiKreis(300, 300, 12, Color.BLUE);

  }     // Ende Konstruktor


   @Override
   public void paintComponent(Graphics grPnl) { // zeichnet auf ein Panel
      Graphics2D g2d = (Graphics2D) grPnl;
      super.paintComponent(g2d);
      g2d.drawImage(bufKos, 0, 0, this); // zeichnet bufKos(Puffer) auf "this"=JPanel
      repaint();
   }


   public void plotPixel(int x, int y, Color col) {
      bufKos.setRGB(x, y, col.getRGB());
   }
```

```
  void zeichneBresiKreis(int xM, int yM, int r, Color farbe) {
     // Bresenham-Algorithmus für Kreise von Ac
     int x = 0;
     int y = r;
     int delta = 3 - 2*r;

     plotPixel (xM, yM + r, farbe);
     plotPixel (xM, yM - r, farbe);
     plotPixel (xM + r, yM, farbe);
     plotPixel (xM - r, yM, farbe);

     while (x < y) {
        if (delta >= 0) {
           delta = delta + 4*(x-y) + 10;
           y = y - 1;
        } else
           delta = delta + 4*x + 6;
        x = x + 1;
        plotPixel (xM + x, yM + y, farbe);
        plotPixel (xM + x, yM - y, farbe);
        plotPixel (xM - x, yM + y, farbe);
        plotPixel (xM - x, yM - y, farbe);

        plotPixel (xM + y, yM + x, farbe);
        plotPixel (xM + y, yM - x, farbe);
        plotPixel (xM - y, yM + x, farbe);
        plotPixel (xM - y, yM - x, farbe);

     } // while
  }


}
```