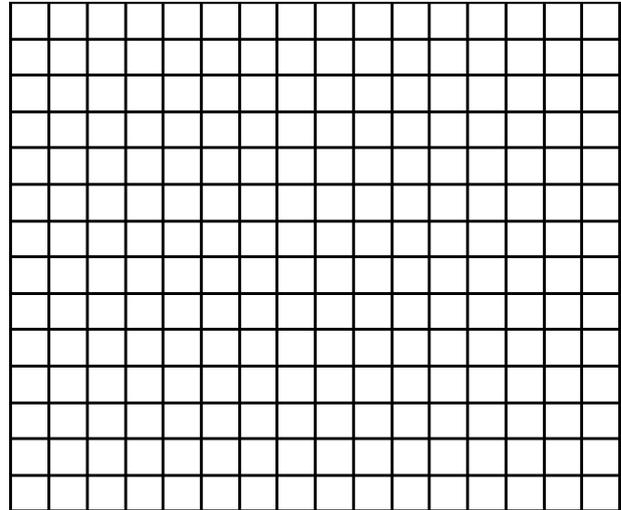
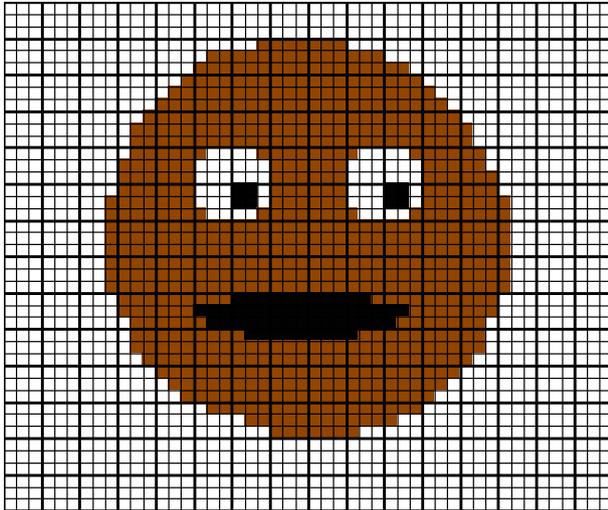




Bildgröße

Eine typische Anwendung eines Bildprogramms besteht in der Anpassung der Größe eines Bildes. Dabei nimmt die Anzahl der Pixel zu oder ab und es nicht automatisch klar, welche Farbe die neuen Pixel bekommen sollen.



Bildquelle: eigenes Werk

Aufgabe

1. Im Beispiel hat das neue Bild (rotes Raster) weniger Pixel als das Originalbild.
 Entscheide, welche Farbe die Pixel im neuen Bild bekommen sollen. Dabei muss immer das ganze Pixel in einer einzigen Farbe ausgemalt werden. Male das rechte Raster entsprechend an. Beschreibe, wie du dabei vorgegangen bist.

Bei allen Verfahren wird als Farbe des neuen Pixels die Farbe derjenigen Pixel im Originalbild herangezogen, die in der Nähe der Position des neuen Pixel liegen. Dazu muss für jedes Pixel des neuen Bildes berechnet werden, an welcher Stelle dieses Pixel im alten Bild gelegen hätte:

$$x_{alt} = \frac{x_{neu}}{breite_{neu} - 1} \cdot (breite_{alt} - 1)$$

$$y_{alt} = \frac{y_{neu}}{höhe_{neu} - 1} \cdot (höhe_{alt} - 1)$$

Beispiel:

	Bildgröße	Punkt 1	Punkt 2	Punkt 3	Punkt 4
verkleinertes Bild	16x14	(0 0)	(15 13)	(11 8)	(2 12)
Originalbild	48x42				
vergrößertes Bild	96x84	(0 0)	(95 83)	(11 8)	(2 12)
Originalbild	48x42				

Aufgabe

2. Fülle die Tabelle mit den Koordinaten der Punkte im Originalbild aus. Beachte, dass dabei  Kommazahlen herauskommen können.



Nearest Neighbor

Beim nearest Neighbor-Verfahren wird für die Farbe des neuen Pixel die Farbe des nächstgelegenen Pixels übernommen. Den nächsten Nachbarn (nearest neighbor) eines Pixels findet man, indem man diese Werte x_{alt} und y_{alt} auf ganze Zahlen rundet.

Aufgaben

3.  Gib an, welche Koordinaten die nächsten Nachbarn für die Pixel 1-4 im Originalbild hätten.

4. Erstelle eine Klasse `GeometrischeBildoperationen2`.

 Implementiere darin eine Methode `public Color naechsterNachbar(Color[][] pixel, double x, double y)`, die den nächsten Nachbarn zu den Koordinaten x und y bestimmt und dessen Farbe zurückgibt. Dabei sind x und y schon die Koordinaten im Originalbild.

Hinweis: Man rundet mit der Methode `Math.round(...)`. Leider kommt dabei bei Double-Zahlen ein Wert vom Typ `long` heraus. Daher muss dieser mit `(int)` noch in eine `int`-Zahl umgewandelt werden.

5.  Implementiere eine Methode `public Picture groesseAendern(Picture originalbild, int neueBreite, int neueHoehe)`, die unter Verwendung des nearestNeighbor-Algorithmus die Bildgröße anpasst.

Erstelle dazu ein Array für die neuen Pixel in der neuen Größe. Berechne dann für jeden neuen Pixel mit der oben angegebenen Formel die Koordinaten des Pixels im Originalbild und nutze dann die Methode `naechsterNachbar`, um die Farbe des neuen Pixels festzulegen.

Hinweis: Damit bei der Formel wirklich Kommazahlen herauskommen, muss Java angewiesen werden, die Zählvariablen der For-Schleifen in der Formel als `double`-Zahl zu interpretieren: `double altX = (double) x /`

Teste deine Methode mit verschiedenen Bildern. Verwende dabei auch "Testbild.png" und verkleinere die Breite auf 143. Untersuche, was mit den Elementen des Testbildes passiert.

Dieses Verfahren hat zwei Nachteile: Zum einen bekommen schräg verlaufende Linien durch die Rundungen „Knicke“. Manche Linien können ganz verschwinden. Zum anderen wird das Bild bei Vergrößerung pixelig.

Aufgaben

6.  Füge dem Bearbeiten-Menü der GUI einen neuen Menüpunkt `Größe ändern` hinzu. Erstelle ein Optionsfenster, in dem die alte Größe angezeigt und die neue Größe eingegeben werden kann. Für beides kann man `NumberFields` verwenden und bei den Anzeigefeldern die Eigenschaft "Editable" deaktivieren.

 Füge der Controller-Klasse die Action-Methode hinzu, die beim Auswählen des Menüpunktes das Optionsfenster anzeigt. Die Eingabefelder soll dabei direkt mit der alten Bildgröße gefüllt werden (Methode `setValue(...)`).

Füge eine `ChangeListener`-Methode für die `TextProperty` der Eingabefelder hinzu. In dieser Methode soll die Bildgröße mit der Methode `groesseAendern` angepasst werden, falls in den Eingabefeldern Werte größer Null eingegeben wurden.

Größe ändern

alte Größe x

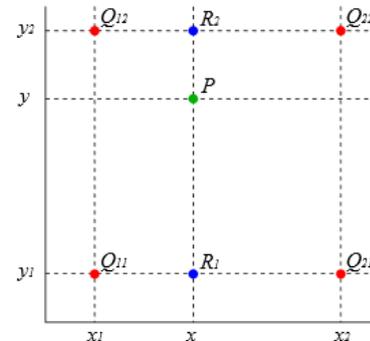
neue Größe x

Ausfuehren Abbrechen



Bilineare Interpolation

Rechnet man die Position eines neuen Pixels (P) in die Koordinaten im alten Bild um, fallen die Koordinaten in der Regel nicht genau auf ein Pixel des alten Bildes (Q_{11} bis Q_{22}). Statt das nächstgelegene Pixel zu nehmen, werden bei diesem Verfahren die vier umliegenden Pixel des alten Bildes in verschiedenen Gewichtungen herangezogen. Je näher P an einem Punkt Q liegt, desto stärker wird dieser gewichtet. Liegt P genau in der Mitte der vier Punkte, dann werden alle vier umliegenden Punkte mit dem Faktor $\frac{1}{4}$ gewichtet.



Wenn x die Differenz zwischen x_P und $x_{Q_{11}}$ ist und y die Differenz zwischen y_P und $y_{Q_{11}}$ ist, dann ergibt sich folgende Formel:

Abb. 1: Bilineare Interpolation (Lizenz: Public domain)

$$I_P = I_{Q_{11}} \cdot \underbrace{(1-x)}_{\text{Gewicht für } Q_{11}} \cdot \underbrace{(1-y)}_{\text{Gewicht für } Q_{11}} + I_{Q_{12}} \cdot \underbrace{(1-x)}_{\text{Gewicht für } Q_{12}} \cdot \underbrace{(y)}_{\text{Gewicht für } Q_{12}} + I_{Q_{21}} \cdot \underbrace{(x)}_{\text{Gewicht für } Q_{21}} \cdot \underbrace{(1-y)}_{\text{Gewicht für } Q_{21}} + I_{Q_{22}} \cdot \underbrace{(x)}_{\text{Gewicht für } Q_{22}} \cdot \underbrace{(y)}_{\text{Gewicht für } Q_{22}}$$

Beispiel:

P	Q_{11}	Q_{12}	Q_{21}	Q_{22}	g_{11}	g_{12}	g_{21}	g_{22}
(2,5 4,75)	(2 4)	(2 5)	(3 4)	(3 5)	1/8	3/8	1/8	3/8
(1,25 3,75)	(1 3)	(1 4)						
(4,75 0,5)								
(1,1 4,7)								

Aufgaben

7. Gib in der Tabelle an, welche Punkte Q_{11} bis Q_{22} um P herum liegen. Berechne die Gewichte g_{11} bis g_{22} .

8. Implementiere eine Methode `public Color bilineareInterpolation`

(`Color[][] pixel, double x, double y`), die die Farbe eines neuen Pixels mit der bilinearen Interpolation bestimmt.

Berechne dazu

- die Koordinaten x_1, y_1, x_2 und y_2 aus den Parametern x und y .

Hinweis: mit (`int`) kann man Nachkommastellen abschneiden.

Hinweis2: Achte darauf, dass x_1 und x_2 zwischen 0 und Breite-1 und y_1 und y_2 zwischen 0 und Höhe-1 liegen. Ermittle die Bildbreite bzw. Bildhöhe aus der Größe des Arrays. Andernfalls bekommt man `ArrayOutOfBoundsException`.

- die Differenzen dx und dy zwischen x und x_1 bzw. y und y_1 .

- die Gewichte g_{11}, g_{12}, g_{21} und g_{22}

Berechne dann die neue Intensität für jede Farbkomponente nach der oben angegebenen Formel und erstelle damit eine neue Farbe, die als Ergebnis zurückgegeben wird.

9. Passe die Methode `groesseAendern` so an, dass statt des nearest Neighbor-

Algorithmus die bilineare Interpolation verwendet wird.