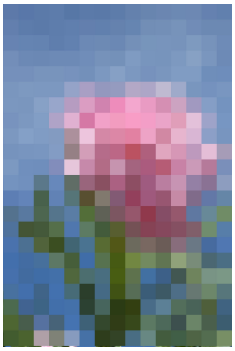
Friedrich-Schiller-Universität Jena
Institut für Informatik
Martin Mundhenk
https://www.complexity.uni-jena.de/Python1

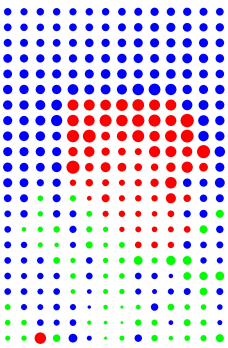
Aufgabenblatt 9, Abgabetermin 13.7.2020

Lösen Sie Aufgaben im Umfang von 15 Punkten.

Aufgabe 50: Bilder rastern **15 Punkte**

Schreiben Sie ein Programm, das Bilder rastert. Unten sehen Sie ein Foto und drei gerasterte Varianten davon. Bei allen Varianten wird das Bild in lauter Quadrate aufgeteilt, deren Breite in Pixeln von der Kommandozeile gelesen wird. Bei der ersten Variante wird für jedes Quadrat der Mittelwert aller Farben berechnet und das Quadrat wird komplett mit dieser Farbe gefüllt. Bei der zweiten Variante wird die Luminanz des Mittelwertes der Farben des Quadrats berechnet und das Quadrat wird mit einem schwarzen Kreis gefüllt, dessen Größe von der Luminanz abhängt. Bei der dritten Variante wird der Kreis nicht mit schwarz gefüllt sondern mit rot, grün oder blau. Es wird die Farbe gewählt, die beim Mittelwert der Farben des Quadrats den höchsten Wert unter den drei Farben hat.
Schreiben Sie ein Modul, das für jede Variante eine Funktion enthält. Sie erhalten 11 Punkte für eine Variante, und für jede weitere Variante 2 Punkte. Die Testfunktion des Moduls soll ein Bild einlesen und alle drei Varianten davon ausgeben.



Ich hatte Probleme beim Programmieren, da beim Setzen der Farbe eines Bildpunktes mit `pic.set(w,h,f)` der Rot-Wert und der Blau-Wert der Farbe `f` vertauscht wurden. Wenn das bei Ihnen auch passiert, wissen wir, dass es nicht Ihr Fehler ist. Ich habe das Bild letztlich mit `stddraw` gemalt.

Aufgabe 51: Diashow **15 Punkte**

Schreiben Sie ein Programm, das eine beliebige Anzahl von Bilddateinamen zeilenweise von standard input liest. Es soll die Bilder in einer Diashow (im Abstand von je ein paar Sekunden) anzeigen, wobei die Bilder mit einem zufällig ausgewählten Effekt ineinander übergehen sollen. Implementieren Sie dafür drei Effekte Ihrer Wahl. Z.B.:

Abdunkeln und Aufhellen Dunkelt das erste Bild bis zu Schwarz ab und hellt dann das zweite Bild auf.

Einschieben/Herausschieben Schiebt das zweite/erste Bild aus einer beliebigen Richtung in das/aus dem Gesamtbild ein/heraus.

Superposition Erhöht schrittweise einen Faktor α im Bereich von 0 bis 1 und zeigt für jeden dieser Faktoren das Gesamtbild an. Die Farbwerte jedes Pixels im Gesamtbild setzen sich zu $(1-\alpha)$ aus den Farbwerten des entsprechenden Pixels im ersten Bild und zu α aus den Farbwerten des entsprechenden Pixels im zweiten Bild zusammen.

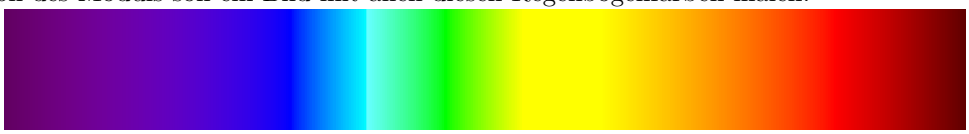
Auflösen Kopiert schrittweise eine zufällige Menge von Pixeln des zweiten Bilds an die entsprechenden Stellen im ersten Bild, bis das gesamte zweite Bild kopiert wurde.

Aufgabe 52: Regenbogenfarben **15 Punkte**

Bei Wikipedia beginnt der Eintrag über *Farbe* mit folgendem Satz.

> Eine Farbe ist ein durch das Auge und Gehirn vermittelter Sinneseindruck, der durch Licht hervorgerufen wird, genauer durch die Wahrnehmung elektromagnetischer Strahlung der Wellenlänge zwischen 380 und 760 Nanometer.

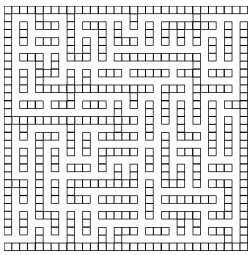
Schreiben Sie ein Modul mit einer Funktion, die eine Wellenlänge aus dem o.g. Bereich (und möglicherweise weitere Werte) als Parameter hat und das `Color`-Objekt für die Farbe dieser Wellenlänge zurückgibt. Die Testfunktion des Moduls soll ein Bild mit allen diesen Regenbogenfarben malen.



(Unter `http://www.physics.sfasu.edu/astro/color/spectra.html` gibt es ein FORTRAN-Programm, das weiterhelfen kann.)

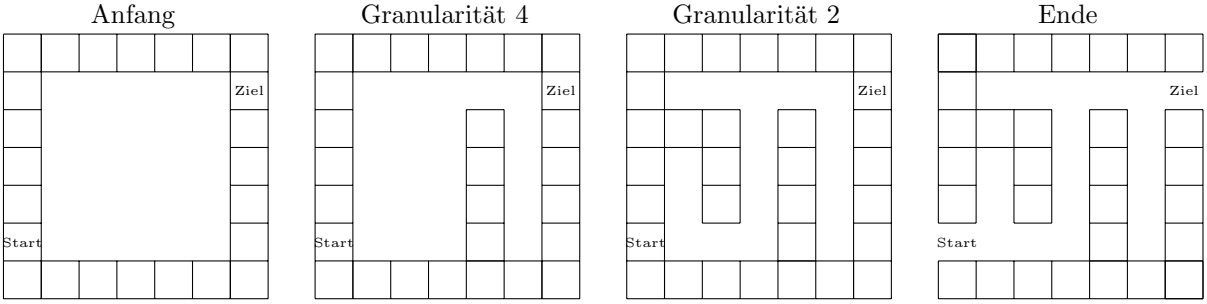
Aufgabe 53: Zufallslabyrinth erzeugen **15 Punkte**

In einer früheren Aufgabe haben wir versucht, zufällige Labyrinthe zu erzeugen. Dabei konnten wir nicht garantieren, dass das Labyrinth einen Weg vom Start zum Ziel besitzt. Jetzt wollen wir eine andere Methode verwenden, die Labyrinthe mit mehreren Lösungswegen erzeugt, wie z. B.:



Schreiben Sie ein Programm, das für vier als Kommandozeilenparameter gegebene positive ganze Zahlen m, n, $minLength$ und $maxLength$ ein zufälliges Labyrinth der Größe $2m+1$ mal $2n+1$ generiert und grafisch ausgibt. Dabei soll $maxLength$ größer oder gleich $minLength$ sein. Die Generierung der Wände soll eine Funktion `fillWithWalls(maze, minLength, maxLength, granularity, numWalls)` übernehmen, die `numWalls`-mal versucht, in dem Labyrinth `maze` zufällig Wände einzuzeichnen. Dies tut sie, indem zuerst eine Startposition aus zwei zufällig gewählten Vielfachen der Granularität gebildet wird, die im Labyrinth liegt. Also z. B. $(0,\ 3 \cdot granularity)$ oder $(4 \cdot granularity,\ granularity)$. Falls an der Startposition noch keine Mauer vorhanden ist, rät sie außerdem eine Richtung sowie eine Länge aus $minLength \cdot granularity + 1, (minLength + 1) \cdot granularity + 1, \ldots, maxLength \cdot granularity + 1$. Dann wird die entsprechende Mauer Block für Block in das Labyrinth eingefügt, bis die Länge, der Rand oder eine andere Mauer erreicht ist.

Ihr Hauptprogramm soll mit einem Labyrinth starten, das nur am Rand Wände hat. Dann soll das Labyrinthinnere durch mehrmaliges Aufrufen der Funktion `fillWithWalls` gefüllt werden. Experimentieren Sie mit verschiedenen Werten für die Granularität `granularity` und die Anzahl der Versuche `numWalls`. Die Granularität steuert, wie weit die möglichen Start- und Endpositionen der Wände auseinanderliegen. Fangen Sie mit einer hohen Granularität an, so entstehen erst große Räume mit langen Wänden. Diese können dann durch Aufrufe mit kleinerer Granularität gefüllt werden. Beachten Sie aber, nur *gerade* Werte für die Granularität zu verwenden – sonst können Wände generiert werden, die auf Kreuzungspunkten beginnen. Zum Schluss muss der Eingang bei $(0, 1)$ und der Ausgang bei $(2m, 2n-1)$ freigeräumt werden.



Generierung eines $2m+1$ mal $2n+1$ Zufallslabyrinths mit $m = n = 3$.