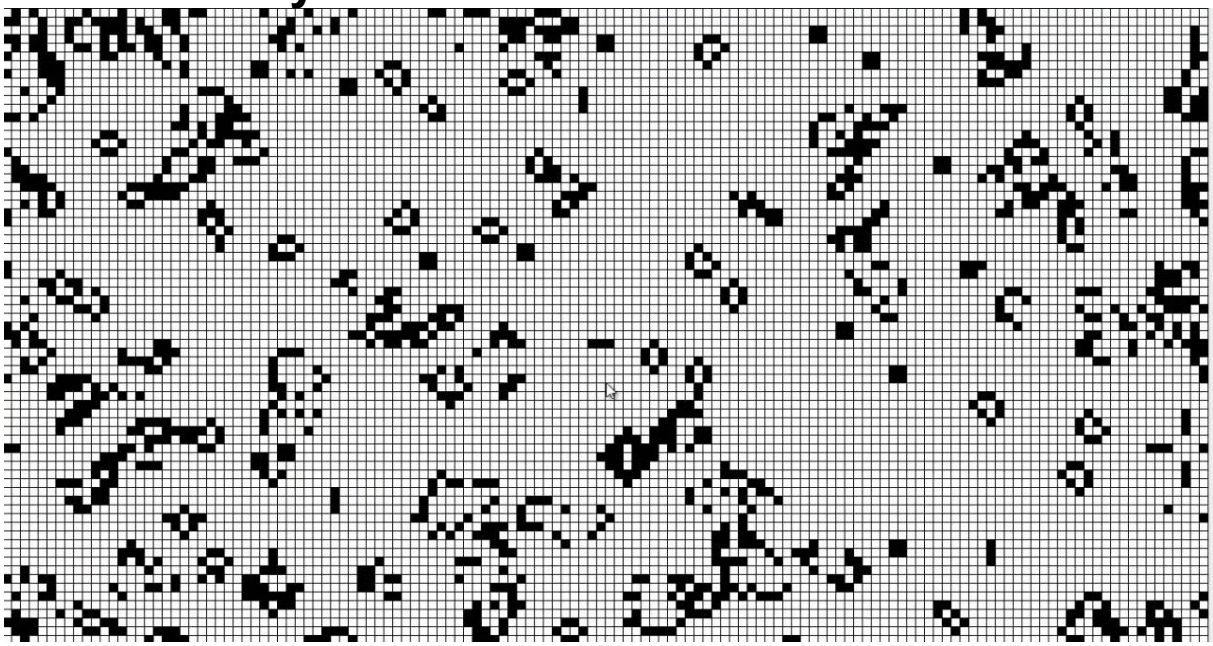
Modul 411: Datenstrukturen und Algorithmen entwerfen und anwenden

gibbix – Lernumgebung

Linux
Kubuntu

Inhalt AB411-05:

John Conway's Game of Life



Einfache Datenstrukturen

Vereinfachte FOR-Schleife

Die vereinfachte `for`-Schleifennotation benötigt beim Durchlaufen von Arrays keine Schranken und keine Manipulation von Zählvariablen.
Die Syntax sieht wie folgt aus:

```
for (<Datentyp> <Variablenname> : <Liste>) {
     <Anweisungen>;
}
```

Wir sagen also nicht mehr explizit, welche Indizes innerhalb der Schleife durchlaufen werden sollen, sondern geben eine Variable an, die jeden Index beginnend beim kleinsten einmal annimmt.

Schauen wir uns ein Beispiel an:

```
int[] werte = {1,2,3,4,5,6}; // literale Erzeugung
// Berechnung der Summe int summe = 0;
for (int x : werte)
     summe += x;
```

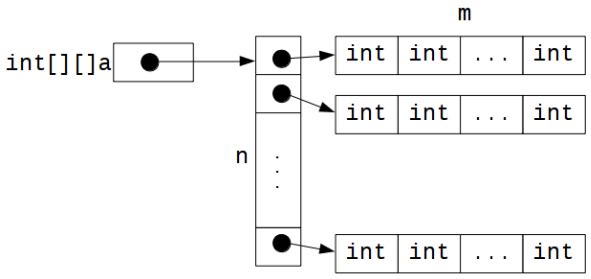
Damit lässt sich die `ArrayIndexOutOfBoundsException` elegant vermeiden.

Mehrdimensionale Arrays

Matrix

Für verschiedene Anwendungen ist es notwendig, Arrays mit mehreren Dimensionen anzulegen und zu verwenden. Ein Spezialfall mit der Dimension 2 ist die Matrix. Wir haben ein Feld von Elementen der Grösse `n x m`.
Wir erzeugen eine `n x m` Matrix, indem wir zum Array eine Dimension dazunehmen:

```
int[][] a = new int[n][m];
```



Wir können auch mehr Dimensionen erzeugen. Beispiel `(k x l x m x n)`:

```
int[][][][] a = new int[k][l][m][n];
```

Mehrdimensionale Arrays literal erzeugen

Bei der Matrix oder mehrdimensionalen Arrays können wir wieder die literale Erzeugung verwenden. Mal angenommen, wir wollen die folgende Matrix

$$M = \begin{bmatrix} 4 & 5 & 6 \\ 2 & -9 & -3 \end{bmatrix}$$

in unserem Programm als zweidimensionales Array definieren. Dann kann die literale Erzeugung sehr übersichtlich sein:

```
int[][] matrix = {{4, 5, 6},
                  {2,-9,-3}};
```

Dimensionierung auslesen und Matrix anzeigen

Jetzt wollen wir eine Funktion `showMatrix` schreiben, die ein zweidimensionales Array vom Datentyp `int` auf der Konsole ausgeben kann:

```
//  @author:        Ralph.Maurer@iet-gibb.ch
//  Compilation:    javac AB411_05_Matrix.java
//  Execution:      java -jar AB411_05_Matrix.jar
package ab411_05_matrix;

public class AB411_05_Matrix {

    public static void main(String[] args) {
        int[][] matrix = {{4, 5, 6},
                          {2,-9,-3}};
        for (int i=0; i<matrix.length; i++) {
            for (int j=0; j<matrix[i].length; j++){
                System.out.print(matrix[i][j]+"\t");
            }
            System.out.println();
        }
    }
}
```

In diesem Beispiel sehen wir gleich, wie wir auf die Längen der jeweiligen Dimensionen zurückgreifen können. In dem Beispiel haben wir die Länge 2 in der ersten und die Länge 3 in der zweiten Dimension. Die erste erfahren wir durch `matrix.length`
und die zweite erfahren wir von der Liste an der jeweiligen Stelle in der Matrix mit `matrix[i].length`
Als Ausgabe erhalten wir:

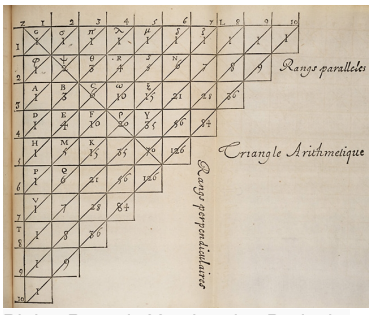
```
4   5   6
2  -9  -3
```

Die Ausgaben wurden in einer Zeile mit `\t` durch einen Tabulator räumlich getrennt.

Liste unterschiedlich langer Listen

Hier ein Beispiel, das die ersten vier Zeilen des Pascalschen Dreiecks zeigt (https://de.wikipedia.org/wiki/Pascalsches_Dreieck):

```
int[][] pascal = {{1},
                  {1,1},
                  {1,2,1},
                  {1,3,3,1},
                  {1,4,6,4,1}};
```



Blaise Pascals Version des Dreiecks

Da sich die Ausgabefunktion `showMatrix` jeweils auf die lokalen Listengrössen bezieht, ergeben die folgenden Zeilen die fehlerfreie Wiedergabe der Elemente:

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
```

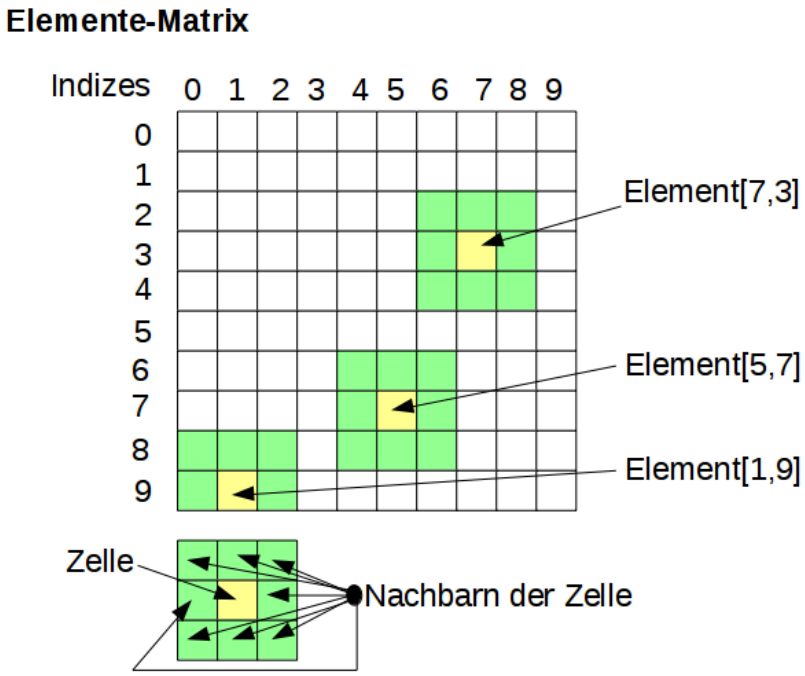
Wir können also problemlos mit mehrdimensionalen Listen unterschiedlicher Längen arbeiten.

John Conway's Game of Life

John Horten Conway ist ein Mathematiker der 1970 ein System entdeckte mit verblüffenden Ergebnissen.
Schauen Sie den Film:
/99_Div/AB411_05/Stephen Hawkings The Meaning of Life (John Conways Game of Life segment).mp4
Lesen Sie:
https://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens

Man stelle sich vor, eine Welt ist eine Matrix und bestünde nur aus 2 Dimensionen. Dies liesse sich wie ein grosses Gitter mit Zellen darstellen. Conway ging in seinem Game of Life von einer Matrix mit Billionen von Elementen aus. Wir arbeiten einfachheitshalber mit 10 Elementen:

Welt als Matrix



Jeder Eintrag, wir nennen ihn jetzt mal Zelle, kann innerhalb dieser Welt zwei Zustände annehmen: sie ist entweder lebendig oder tot. Jede Zelle interagiert dabei von Generation zu Generation mit ihren maximal 8 Nachbarn.

Die vier Spielregeln

Die Überführung der Welt zu einer neuen Generation, also die Überführung aller Zellen in ihren neuen Zustand, unterliegt den folgenden vier Spielregeln:

1. jede lebendige Zelle, die weniger als zwei lebendige Nachbarn hat, stirbt an Einsamkeit

2. jede lebendige Zelle mit mehr als drei lebendigen Nachbarn stirbt an Überbevölkerung

3. jede lebendige Zelle mit zwei oder drei Nachbarn fühlt sich wohl und lebt weiter

4. jede tote Zelle mit genau drei lebendigen Nachbarn wird wieder zum Leben erweckt

Die Idee besteht nun darin, eine konkrete oder zufällige Konstellation von lebendigen und toten Zellen in dieser Matrix vorzugeben. Das bezeichnen wir dann als die erste Generation. Die zweite Generation wird durch die Anwendung der vier Regeln auf jede der Zellen erzeugt. Es wird geprüft, ob Zellen lebendig bleiben, sterben oder neu entstehen.
Wird ein solches System beispielsweise mit einer zufälligen Konstellation gestartet,

so erinnert uns das Zusammenspiel von Leben und Tod z.B. an Bakterienkulturen in einer Petrischale, daher der Name "Spiel des Lebens".

Auftrag: Conways' Game of Life programmieren

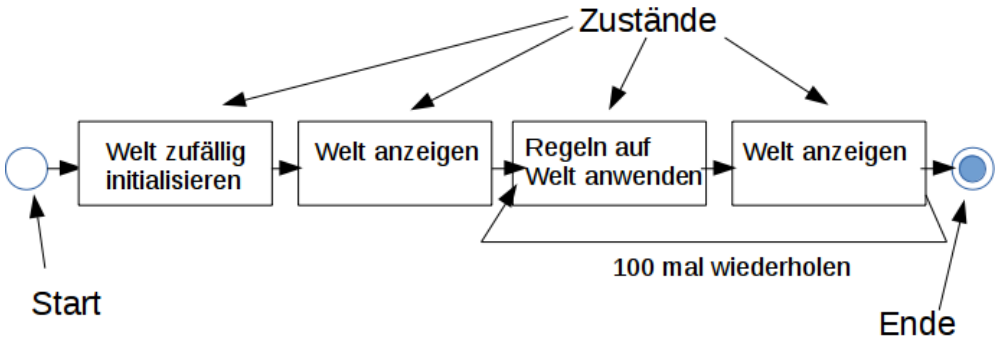
Eine ausführliche Version von Conway's Game of Life in Java hat Edwin Martin programmiert.
http://www.bitstorm.org/gameoflife/
Testen Sie die Version als ausführbare Datei /99_Div/AB411_05/GameOfLife.jar.

Erster Projektentwurf

Unsere Aufgabe wird es nun sein, eine sehr einfache Version von Conways Game of Life zu implementieren. Schön wäre auch eine Generationsfolge, damit wir die sich ändernde Welt visuell verfolgen können.

Grobentwurf – endlicher Automat

Identifizierte Teilaufgaben sind in einem endlichen Automaten dargestellt:



Teilaufgaben ausformuliert

- Grobarchitektur des Programms festlegen: Interaktion der Funktionen, Austausch der Daten und Programmschleife.
- Datentyp für die zweidimensionale Weltmatrix definieren, dabei Lösungen für die Randproblematik finden.
- Variable Programmparameter festlegen: Weltmatrixgröße, Verteilung der lebendigen und toten Zellen in der Startkonfiguration.
- Funktion `initWelt` implementieren, die eine Welt erzeugt und entsprechend der vorgegebenen Verteilung füllt und zurückgibt.
- Funktion `zeigeWelt` implementieren, die die aktuelle Welt auf der Konsole ausgibt.
- Funktion `wendeRegelnAn` implementieren, die eine erhaltene Generationsstufe der Welt nach den Spielregeln in die nächste Generationsstufe überführt und das Resultat zurückgibt.

Welt und Randproblematik

In der `main`-Funktion werden wir eine zweidimensionale Matrix `welt` anlegen. Für die möglichen Zellzustände ist es naheliegend, eine Matrix vom Typ `boolean` zu verwenden. Eine tote Zelle wird durch `false` und eine lebendige Zelle entsprechend durch `true` repräsentiert.
Es gibt nun verschiedene Möglichkeiten mit der Randproblematik umzugehen.
Wenn wir also die acht Nachbarn eines Weltelements erfragen, wollen wir gerne auf eine `IndexOutOfBoundsException` verzichten. Trotzdem soll die Abfrage der Nachbarn direkt und nicht abhängig von der jeweiligen Position sein. Ein legitimer

Weg ist es statt eine 10x10 Matrix eine 12x12 anzuwenden.

Festlegung der Parameter

Die Welt-Matrix

Eine Welt der Dimension 10x10 sollte für den Einstieg genügen. Wir werden für den zusätzlichen Rand insgesamt also eine 12x12 Matrix benötigen.

Lebendige oder tote Zelle

Für die Startkonstellation wünschen wir uns 60% lebendige Zellen mit einer zufälligen Verteilung.

Grundgerüst des Programms

```
//  @author:       Ralph.Maurer@iet-gibb.ch
//  Compilation:   javac AB411_05_GameOfLife.java
//  Execution:     java -jar AB411_05_GameOfLife.jar
package ab411_05_gameoflife;

public class AB411_05_GameOfLife {

// global definierte Konstanten für die beiden Dimensionen
final static int DIM1 = 12;
final static int DIM2 = 12;

// liefert eine zufällig initialisierte Welt
public static boolean[][] initWelt() {
       ...}

// gibt die aktuelle Welt aus
public static void zeigeWelt(boolean[][] welt) {
       ...}

// wendet die 4 Regeln an und gibt die
// Folgegeneration wieder zurück
public static boolean[][] wendeRegelnAn(boolean[][] welt){
       ...}

public static void main(String[] args) {
    boolean[][] welt = initWelt();
    System.out.println("Startkonstellation");
    zeigeWelt(welt);
    for (int i=1; i<=100; i++){
        welt = wendeRegelnAn(welt);
        System.out.println("Generation "+i);
        zeigeWelt(welt);
     }
}
```

Tipp 1: Zufällige Initialisierung der Welt

Angenommen, wir wollen einen Zufallswert erzeugen, der in 60% der Fälle `true` liefert:

```
if (Math.random() > 0.4) // Zufallszahl im Intervall [0,1)
welt[x][y] = true; // 60% true
else
welt[x][y] = false; // 40% false
```

Effizienter wäre:

```
welt[x][y] = Math.random() > 0.4;
```

Tipp 2: Zufällige Initialisierung der Welt

Eine Hauptinformation für die Entscheidung, welche der Regeln greift, ist die Anzahl der vorhandenen Nachbarn. Es lohnt sich darüber nachzudenken, ob wir dafür eine eigene Funktion anbieten wollen.
Für die Nachbarschaftsberechnung einer Zelle benötigen wir die Weltmatrix sowie die Position mit x- und y-Koordinaten. Dann müssen wir in zwei Schleifen nur noch die lebenden Zellen aufsummieren und gegebenenfalls die Zelle selbst nochmal abziehen:

```
public static int anzNachbarn(boolean[][] welt, int x, int y) {
  int ret = 0;
  for (int i=x-1; i<=x+1; ++i)
    for (int j=y-1; j<=y+1; ++j)
      if (welt[i][j])
        ret += 1;

// einen Nachbarn zuviel mitgezählt?
if (welt[x][y])
  ret -= 1;

return ret;
}
```

Wenn uns die Nachbarschaftsfunktion zur Verfügung steht, können wir den Abschnitt zu den Spielregeln sehr übersichtlich gestalten.

Viel Erfolg!

Ausblick

Wer jetzt denkt, dass Conway's Game of Life nur reine Spielerei ist, der täuscht sich. An dieser Stelle muss kurz erwähnt werden, dass dieses Modell aus Sicht der Berechenbarkeitstheorie ebenso mächtig ist, wie es beispielsweise die Turingmaschine und das Lambda-Kalkül sind. Informatikstudenten, die sich im Grundstudium mit dieser Thematik intensiv auseinandersetzen müssen, wissen welche Konsequenz das hat.

Für den interessierten Leser gibt es zahlreiche Webseiten und Artikel zu diesem Thema, eine Webseite möchte ich allerdings besonders empfehlen.
http://www.conwaylife.com/

Aber Vorsicht: Conway's Game of Life kann süchtig machen!