Schulung Blinkenlights

Komplett ohne irgendwas Blinkendes

Sebastian Köll, Jan Rüther

August 11, 2020

1 Vorwort

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2 Aufgabe 1...

To determine the atomic weight of magnesium via its reaction with oxygen and to study the stoichiometry of the reaction (as defined in **??**):

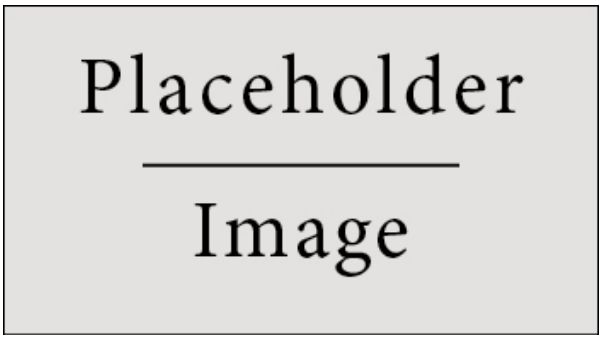


Figure 1: Figure caption.

2.1 Definitions

3 Experimental Data

4 Sample Calculation

5 Results and Conclusions

6 Discussion of Experimental Uncertainty

7 Answers to Definitions

8 Conways *game of life*

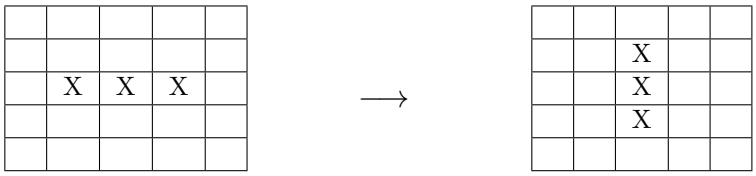
Im Jahr 1968 wurde von J. H. Conway an der Universität Cambridge das *game of life* erfunden und 1970 von M. Gardner im Scientific American einem breiten Publikum vorgestellt. Dabei handelt es sich um einen Algorithmus, der das Wachstum von fiktiven Lebewesen (Bakterien) simuliert. Infolge der interessanten Muster, die dabei entstehen, ist das game of life weit über Biologenkreise hinaus bekannt geworden. Es ist ein Beispiel für einen sogenannten zellulären Automaten. Schauplatz des *game of life* ist ein zweidimensionales Gitter aus Zellen, die entweder tot (' ') oder lebendig ('X') sind. Wie sich eine Zelle weiter entwickelt, hängt von ihren acht Nachbarn ab (diagonale Nachbarn zählen ebenfalls), und zwar gelten folgende Regeln:

1 Eine lebende Zelle überlebt in der nächsten Generation, wenn sie zwei oder drei Nachbarn hat. Sind es weniger bzw. mehr, so stirbt sie an Vereinsamung bzw. Überbevölkerung.

2 Eine tote Zelle wird immer dann in der nächsten Generation zum Leben

erweckt, wenn sie genau drei lebendige Nachbarn hat, ansonsten bleibt sie tot.

Die Zeit verstreicht dabei in diskreten Schritten, d.h. jede Zelle verharrt in ihrem zuvor eingenommenen Zustand, bis gewissermaßen bei einem Gongschlag alle gleichzeitig in den neuen Zustand übergehen. Anders ausgedrückt: Es wird für jede Zelle nachgeschaut, wie ihr Zustand und der ihrer Nachbarn zu einer bestimmten Zeit n ist und berechnet, wie ihr Zustand zur nächsten Zeit $n + 1$ sein wird. Hat man dies für alle Zellen getan, so werden alle gleichzeitig auf den neuen Zustand gesetzt.
Zur Veranschaulichung ist dies im Folgenden an einem einfachen Beispiel erläutert. Das *Spielfeld* besteht hierbei aus einem 5 mal 5 Zellen großen Feld.



Generation 0 Generation 1

Die 0-te Generation besteht aus einer Linie von drei belebten Feldern. Für die Folgegeneration geht unter Annahme der Regel 1 hervor, dass die beiden äußeren Zellen sterben werden, da diese weniger als 2 Nachbarn haben. Die mittlere Zelle überlebt, da sie 2 Nachbarn hat und somit nicht mehr als 3 oder weniger als 2.
Wird die Regel 2 angewandt, so entstehen zwei neu belebte Zellen über und unter der mittleren Zelle, da diese jeweils genau drei Nachbarn haben. Die restlichen Zellen bleiben unberührt.
Verschiedene Startmuster können hierbei zu unterschiedlichen "Populationsverläufen" führen.

8.1 Aufgabenbeschreibung

Es soll das *game of life* in Python selber Programmiert werden. Eine erste grobe Aufgabenaufteilung kann hierbei aus dem *EVA*-Prinzip abgeleitet werden:

Eingabe Als Eingabe wird hierbei das initiale Füllen des Spielfelds verstanden. Die 0-te Generation soll von Außen vorgegeben werden können. Die eingegebenen Daten müssen in einer Form vorliegen, dass die Verarbeitung sie für die Ermittlung der Nachfolgegenerationen verwenden kann.

Verarbeitung Die Verarbeitung ist der Motor des Spiel. Hierbei müssen die oben genannten Regeln umgesetzt werden und Generation für Generation ermittelt werden. Das Weiterschalten von den einzelnen Generationen kann über eine Zeit oder einen Tastendruck geschehen. Die ermittelten Daten sollen am Ende einer Generationsermittlung immer in der Form vorliegen, wie

sie auch von der Eingabe zur Verfügung gestellt wurden. In dieser Form werden sie der Ausgabe zur Verfügung gestellt.

Ausgabe Unter der Ausgabe ist die Darstellung des aktuellen Spielfelds samt "Bevölkerung" zu verstehen.

Vorab sollten sich jedoch einige Gedanken über die Rahmenbedingungen wie Größe des Spielfelds und Regeln für das Verhalten der Zellen am Rand des Spielfelds gemacht werden.
Mehr Informationen zum *game of life* gibt es mit Sicherheit im Internet! Im Folgenden werden wir uns mit der groben Vorgehensweise zum Implementieren des Programms beschäftigen.

8.2 Vorschläge für die Umsetzung

8.2.1 Datenformat einer Generation

Eine der wichtigsten Dinge die bei der Konzeption eines Programms beachtet werden müssen sind die Schnittstellen der einzelnen Module untereinander. Dies wird umso wichtiger, wenn mehrere Programmiere parallel an einem Projekt an unterschiedlichen Modulen arbeiten. Als Module können hierbei die einzelnen Schritte des EVA-Prinzips angesehen werden. Die Schnittstelle is in diesem Fall das weiterreichen der Daten. Wird hier nicht von Anfang an eine einheitliche Form gewählt, so führt dies beim Zusammenführen der Module zu einem großen Chaos und eine Menge zusätzliche Arbeit entsteht.
Was als Daten übergeben werden muss, ist im Grunde genommen nur das Spielfeld. Dieses besteht aus einzelnen Zellen, welche die Werte *tot* oder *lebendig* annehmen können. Da das Spielfeld eine Fläche ist, bietet sich eine zweidimensionale Datenstruktur an. Eine Umsetzung hierfür kann in Python ein Numpy Array sein:

```
import numpy as np
height = 4
width = 5
my_field = np.ones((height, width))
```

Das Codesnippet erzeugt ein Array der gewünschten Höhe und Breite, welches initial mit Nullen gefüllt ist:

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Um nun die zwei verschiedenen Zustände (*lebendig*, *tot*) anzuzeigen, sind verschiedene Darstellungen denkbar. Zum einen könnten tote Zellen mit Nullen und lebendige Zellen mit Einsen dargestellt werden. Denkbar wäre auch eine

boolsche Darstellung `True` für lebendig und `False` für tot oder eine Darstellung mit Hilfe von Strings:

0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
0	0	0	0	0

Darstellung durch Nullen und Einsen

False	True	False	True	False
False	False	True	False	False
False	True	False	True	False
False	False	False	False	False

Darstellung durch True oder False

tot	leb	tot	leb	tot
tot	tot	leb	tot	tot
tot	leb	tot	leb	tot
tot	tot	tot	tot	tot

Darstellung durch Strings

Vorab sollte in der Gruppe festgelegt werden, welches Datenformat ihr wählt. Die oben aufgeführten Formate sind nur Vorschläge. Falls ihr bessere Ideen habt, könnt ihr diese auch gerne umsetzen.

8.2.2 Eingabe der Generation 0

Für die Eingabe der nullten Generation sind verschiedene Möglichkeiten Denkbar. Ziel des Ganzen ist es, am ende der Eingabe ein Spielfeld in der Form vorliegen zu haben, wie es in 8.2.1 festgelegt wurde. Hier sollte die Möglichkeit bestehen, einzelne Zellen als *lebendig* zu initialisieren. Denkbar wäre auch das direkte Setzen verschiedener Formen.
Als simpelste Form der Eingabe ist hierbei sicher das hardgecodete initiale setzen des Spielfelds möglich. Denkbar wäre auch das einlesen einer *csv* Datei, in der das Spielfeld eingetragen ist. Ein Beispiel für eine solche Datei ist in dem Ordner **TODO** zu finden.
Etwas fortgeschrittenere Programmierer können hierbei auch auf eine grafische Oberfläche zurückgreifen, eine so genannte *GUI*.

8.2.3 Verarbeitung der einzelnen Generationen

8.2.4 Ausgabe / Anzeige einer Generation

Bei der Anzeige der aktuellen Generation kann im simpelsten Fall die Konsole von Python verwendet werden. Denkbar wäre hier die Darstellung in Textform:

6
7
8
9
0
1 X X
2 X X
3 X XX X
4 XX X X X XXXX X
5 XX X XX XXXX X
6 X X X X XX
7 X X XX X XX
8 X X X X
9 X
0
12345678901234567890123456789012345678901234567890123456789