

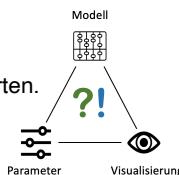
GAME OF LIFE – CHEAT SHEET

Simulationen

Simulationen gehen von einer Fragestellung aus und versuchen diese zu beantworten.

3 Bestandteile:

- Modell (vereinfachtes Abbild der Wirklichkeit)
- Parameter (können verändert und ihr Einfluss auf das Geschehen beobachtet werden)
- Visualisierung (graphische Darstellung; optional)



Game of Life

MODELL

Das Game of Life simuliert eine Welt über die Zeit. Die Welt besteht aus einem zweidimensionalen Raster von Zellen.

Jede Zelle kann **zwei Zustände** annehmen:

- 1: lebend
- 0: tot

Jede Zelle hat **8 Nachbarzellen**:

1	2	3
8	■	4
7	6	5

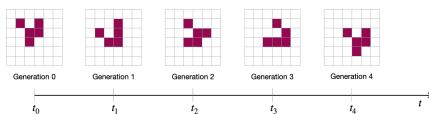
Der **Zustand** einer Zelle **zum Zeitpunkt $t+1$** ermittelt sich aus dem Zustand der Zelle und den Zuständen ihrer Nachbarzellen zum Zeitpunkt t .

Regeln

Zustand Zelle	Zeitpunkt t		Zeitpunkt $t+1$
	Anzahl lebender Nachbarn	Zustand Zelle (neu)	
< 2	(Unterpopulation)		
> 3	(Überpopulation)		
$\in \{2,3\}$	(Überleben)		
tot	3 (Geburt)	lebend	

Eine **Generation** ist der **Zustand der Welt** zu einem **Zeitpunkt t** .

Neue Generationen werden zyklisch berechnet. Die Länge dieser Zyklen ist die **Ticklänge**, die Zeitpunkte werden **Ticks** genannt.



Umsetzung in Python

Abbildung der Welt als 2D-Array (Matrix)

Die **Welt** wird in einem **2-dimensionalen Array** (auch **Matrix** genannt) abgebildet.

Arrays (in Numpy)

Arrays als Datentyp sind nicht in Python integriert. Um Arrays zu verwenden wird eine **Bibliothek** benötigt, z.B. [NumPy](#) (Numerical Python), die mathematische Berechnungen mit mehrdimensionalen Arrays (Matrizen) erlaubt.

`import numpy as np`

Eindimensionale Arrays sind sehr ähnlich wie Listen:

- **Definition** eines Arrays mit `np.array`:
`my_array = np.array([])`
`my_array = np.array([1,2,3,4])`
- Ihr Array hat den **Typ**
`type(my_array) -> numpy.ndarray`
`ndarray steht für n-dimensional array`
- **Zugriff** auf das erste Element eines Arrays:
`my_array[0] -> 1`
- Zugriff auf Teilkreiche von Arrays mittels **Teilbereichsoperator**: `[von: bis und ohne: Schrittgröße]`

In **zweidimensionalen Arrays** ist jede Zeile ein Array.

- **Definition**:
`array_2d = np.array([[0,1,0],[1,1,1]])`
`-> [[0, 1, 0],`
`[1, 1, 1]]`
- **Leere Welt erstellen: Matrix von Nullen**
`new_world = np.zeros((4,8), int)`
- **Dimension n.m.:** Tupel (n Zeilen, m Spalten)
- **Datentyp**
- **Zugriff** auf das Element (1,2) eines Arrays:
`array_2d[1][2] -> 1`
- **Konstellation in die Welt einfügen**:
Teilbereich ersetzen:
`new_world[1:3,2:5]=array_2d`

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	0	0	0	0	0

August 2021 • Alessandra Donzé

- **Zufällige Welt** (Matrix mit zufälligen Werten) im Bereich [Minimum, Maximum] erstellen z.B. [0,2], für zufällige Werte aus {0,1}:

`np.random.randint(0, 2, (20, 40))`

Minimum Maximum (exklusiv) Dimension: Tupel (Zeilen, Spalten)

Dimension:

`shape` liefert den Tupel (Zeilen, Spalten)
`dimension = new_world.shape -> (4, 8)`
`height = new_world.shape[0] (Anz. Zeilen)`
`width = new_world.shape[1] (Anz. Spalten)`

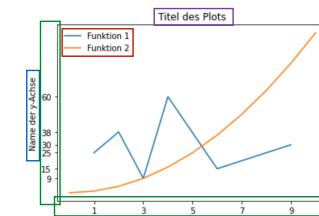
Iteration (Schleife) über alle Zellen:

```
np.ndindex liefert alle Koordinatenpaare (i,j) einer Matrix der Dimension (Zeilen, Spalten)
for index in np.ndindex(dimension):
    print(index) ->(0,0), (0,1), ..., (3,7)
oder anstelle von index (i,j):
for (i,j) in np.ndindex(dimension):
    print((i,j)) ->(0,0), (0,1), ..., (3,7)
```

VISUALISIERUNG

Eine graphische Darstellung (Visualisierung) ist für eine Simulation nicht zwingend nötig, aber eine gute Visualisierung trägt zu einem besseren Verständnis bei.

Die Bibliothek [pyplot aus matplotlib](#) erlaubt die graphische Darstellung (Plots).
`import matplotlib.pyplot as plt`



Welt darstellen: Matrix plotten

Matrix plotten (graphisch darstellen):

`plt.imshow(new_world, cmap=plt.cm.Blues)`

Farbschema:
Argument der Funktion imshow spezifizieren mit `cmap=...`

Beschriftung der x- und y-Achsen

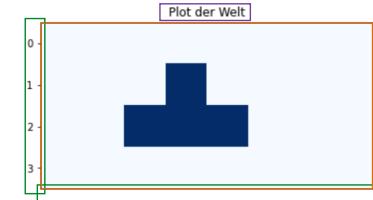
`plt.xticks([1,2,5,7])` -> festgelegte Werte
Für Zeilen- und Spalten-Indizes:
`plt.xticks(rang(0, new_world.shape[1]))`
`plt.xticks(range(0, new_world.shape[0]))`
falls **keine** Beschriftung gewünscht:
leere Arrays übergeben:
`plt.xticks([]), plt.yticks([])`

Titel

`plt.title('Plot der Welt')`

Plot anzeigen

`plt.show()`



Beispiel Funktionsplots

- Werte für x und y definieren:

`x=[1,2,3,4,6,9]`
`y=[25,38,9,60,15,30]`

- Plot definieren

`plt.plot(x,y, label='Funktion 1')`

- Werte für einen zweiten Plot definieren:

`x1=np.arange(0,11,1)`
`y1=x1**2 (x1^2)`

- Zweiten Plot definieren

`plt.plot(x1,y1, label='Funktion 2')`

- Achsennamen

`plt.xlabel('Name der x-Achse')`
`plt.ylabel('Name der y-Achse')`

- Achsenbeschriftung:

`plt.xticks(np.arange(1,11,step=2))`
`plt.yticks(y)`

- Titel des Plots

`plt.title('Titel des Plots')`

- Legende

`plt.legend()`

- Plot anzeigen

`plt.show()`

Werte verteilen:

`np.arange(von, bis, Schrittweite)`