

Künstliche Intelligenz kاپieren und programmieren

Teil 2: KI als Spielgegner

Michael Weigend
Universität Münster



mw@creative-informatics.de
www.creative-informatics.de
2024



Materialien bei GitHub:
<https://github.com/mweigend/ki-workshop>

Tag 1

Zeit	Thema	Inhalte
9.00	Denkende Maschinen	Pädagogische Konzepte, Einstieg in Python, Chatbots und Assistenzsysteme
11.15	KI als Spielgegner	Modellieren mit Listen, Nim-Spiel mit KI als Gegner
12.30	<i>Mittagspause</i>	
13.30	Klassifizieren	Entscheidungsbaum, k-Means-Clustering
14.45	Lernen	Lernfähiger Währungsrechner, Wegsuche, Fußgänger erkennen
16.00	<i>Ende</i>	

Nimrod – der erste elektronische Computer in Deutschland (1951)

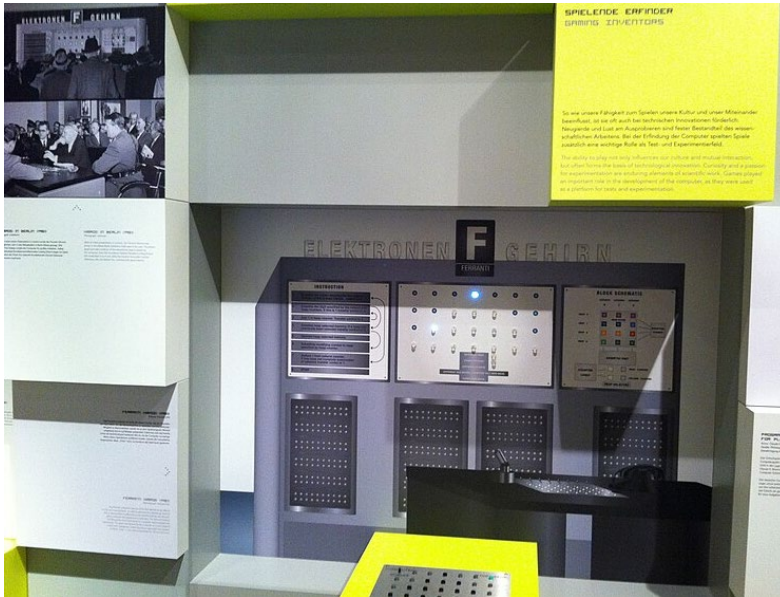


Foto: cc-by Chuck Smith, in Wikimedia Commons

Nachbau im Computerspiele-Museum Berlin



480 Röhren
Fest verdrahtete
Schaltung

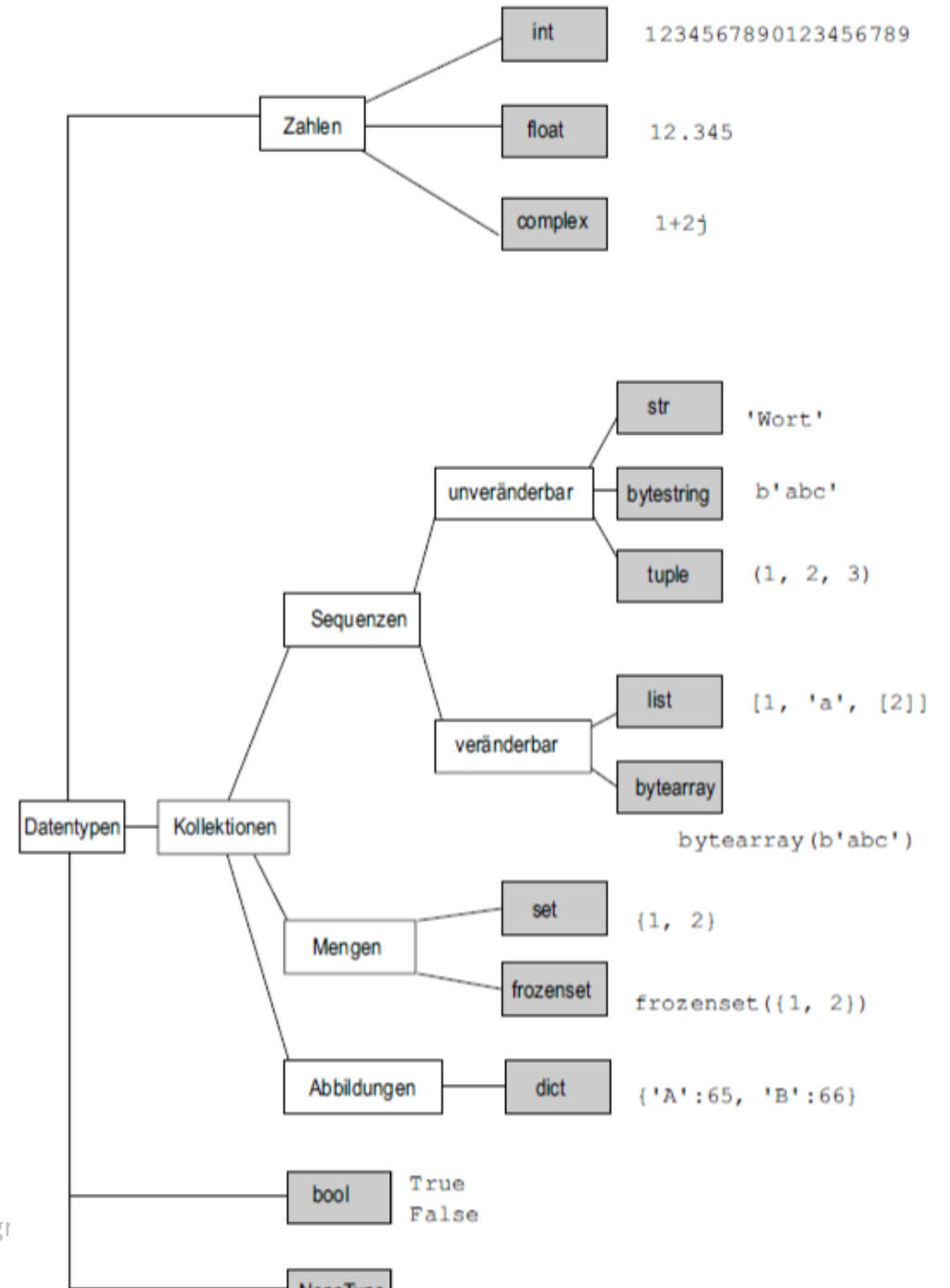
<https://www.heise.de/news/Vor-50-Jahren-fing-alles-an-das-erste-Elektronenhirn-in-Deutschland-51722.html>

Das Nim-Spiel

- Zug: Aus einer Zeile ein oder mehrere Streichhölzer nehmen
- Wer das letzte Hölzchen nimmt, gewinnt.



Modellieren mit Listen und Tupeln



Menge

- Jedes Element kommt nur einmal vor
- Keine Reihenfolge



$$A = \{1, 4, 9, 16, 25\}$$

Sequenz

-2

-1



Index

0

1

2

3

4

Sequenz

Veränderbar:
Liste

```
namen = ['Schall', 'Rauch']
```

```
Tel = ['0223 788834',  
       '0201 566722',  
       '0224 66898',  
       '0201 899933',  
       '0208 33987']
```

Eine Python-Zeile

Unveränderbar:
Tupel, String

```
person = ('Mozart', 'Amadeus')  
V = (15, 'mL')
```

```
name = 'Mozart'
```


Gemeinsame Operationen für Sequenzen

Operation	Ergebnis
<code>x in s</code>	True, wenn ein Element mit dem Wert von <code>x</code> in der Sequenz <code>s</code> enthalten ist, und False sonst.
<code>x not in s</code>	False, wenn ein Element mit dem Wert von <code>x</code> in der Sequenz <code>s</code> enthalten ist, und True sonst.
<code>s + t</code>	Konkatenation der beiden Sequenzen <code>s</code> und <code>t</code>
<code>s * n</code> , <code>n * s</code>	<code>n</code> Kopien der Sequenz <code>s</code> werden hintereinandergehängt.
<code>s[i]</code>	Das <code>i</code> -te Element der Sequenz <code>s</code>
<code>s[i:j]</code>	Ein Ausschnitt (<i>slice</i>) von <code>s</code> , der vom <code>i</code> -ten bis zum <code>j</code> -ten Element (nicht einschließlich) geht
<code>len(s)</code>	Die Länge der Sequenz <code>s</code>
<code>min(s)</code>	Das kleinste Item der Sequenz <code>s</code>
<code>max(s)</code>	Das größte Element der Sequenz <code>s</code>

Welche dieser Operationen funktionieren auch bei Mengen?

Wie kann man Listen erzeugen?

Aufzählen:

```
s1 = [17, 2, 900]
```

Mit `list()` aus anderen Kollektionen gewinnen

```
s2 = list('Guido')
```

```
s3 = list(range(3))
```

```
['G', 'u', 'i', 'd', 'o']
```

```
[0, 1, 2]
```

Wie kann man Listen erzeugen?

In einer Schleife Schritt für Schritt aufbauen

```
quadratzahlen = []  
for i in range(4):  
    item = i**2  
    quadratzahlen.append(item)
```

```
[0, 1, 4, 9]
```

List Comprehension (Listen-Abstraktion)

```
quadratzahlen = [i**2 for i in range(4)]
```

List Comprehension als Filter

[Ausdruck **for** *Element* **in** *Kollektion* **if** *Prädikat*]

```
[x for x in [0, 4, 9, 15, 0, 1] if x < 2]
```

```
[0, 0, 1]
```

Operationen für Listen

Methodenaufruf

Operation	Ergebnis
<code>s[i] = x</code>	Das Element mit Index <code>i</code> wird durch <code>x</code> ersetzt.
<code>s[i:j] = [a₁, ..., a_k]</code>	Die Elemente mit den Indexen <code>i</code> bis <code>j</code> werden durch die Elemente der Liste <code>[a₁, ..., a_k]</code> ersetzt.
<code>s.append(x)</code>	An die Liste <code>s</code> wird als neues Element <code>x</code> angehängt.
<code>s.count(x)</code>	Zurückgegeben wird die Anzahl der Listenelemente mit dem Wert <code>x</code> .
<code>del s[i]</code>	Das Element mit Index <code>i</code> wird aus der Liste entfernt. Damit wird die Länge der Liste um eins verringert.
<code>del s[i:j]</code>	Die Elemente mit den Indexen <code>i</code> bis <code>j</code> werden gelöscht.
<code>s.extend(t)</code>	Die Liste <code>s</code> wird um die Elemente der Sequenz <code>t</code> verlängert.
<code>s.index(x)</code>	Zurückgegeben wird der kleinste Index <code>i</code> mit <code>s[i] == x</code> .
<code>s.insert(i, x)</code>	Falls <code>i >= 0</code> , wird das Objekt <code>x</code> vor dem Element mit dem Index <code>i</code> eingefügt.
<code>s.pop()</code>	Das letzte Listenelement wird aus <code>s</code> entfernt und der Wert zurückgegeben.
<code>s.remove(x)</code>	Das erste Element mit dem Wert <code>x</code> wird aus der Liste <code>s</code> entfernt.
<code>s.reverse()</code>	Die Reihenfolge der Elemente wird umgekehrt.
<code>s.sort()</code>	Die Elemente der Liste werden aufsteigend sortiert.

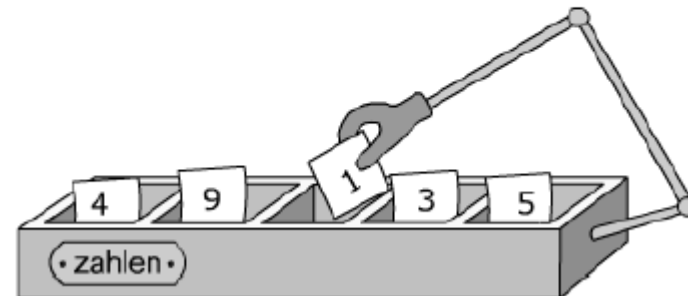
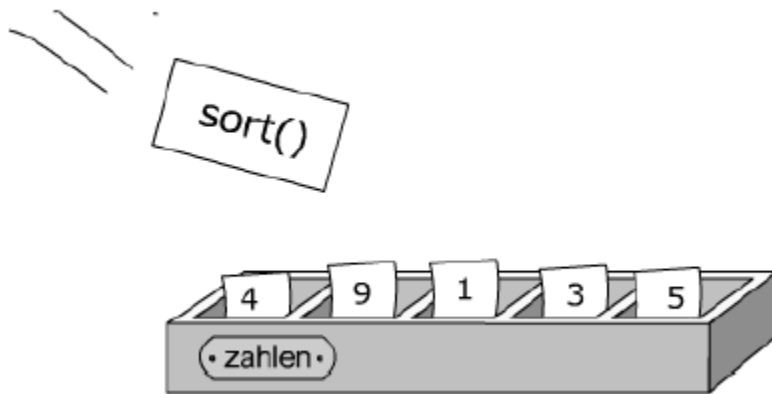
Listen als Objekte

```
>>> zahlen = [4, 9, 3, 1, 5]
>>> zahlen.sort()          # Aufruf der Methode sort()
>>> zahlen
[1, 3, 4, 5, 9]
```

Eine Methode ist eine Funktion, die zu einem Objekt gehört („Objektfunktion“).

Aufruf:

objekt.methode(...)



Ein list-Objekt empfängt eine Botschaft und führt einen Auftrag aus.

Modellieren



Was bedeutet 0 bzw. 1?

(0, 1, 0, 0, 1, 0)

Modellieren

```
((plastic, plastic, plastic), (metal, metal))
```



Modellieren

```
b = (a, a, a, a, a, a, a)  
deckel = (b, b)
```



Übung 2.1 Listen

1. Was schreibt das folgende Skript auf den Bildschirm?

```
liste = ["mond", "stoff", "treib", "raum", "schiff"]
print(liste[0])
print(liste[2] + liste[1])

print(liste[-2] + liste[-1])
for wort in liste:
    if wort[0] == "s":
        print(wort)

for wort in liste:
    print(wort[1])

liste = liste + ["gestein"]
print(liste[0] + liste[-1])
```


Übung

2. Vervollständigen Sie die folgende Tabelle!

List Comprehension	Liste
<code>[a**2 for a in range(5)]</code>	<code>[0, 1, 4, 9, 16]</code>
	<code>[(0, 1), (1, 2), (2, 3)]</code>
<code>[(b, a) for (a, b) in [(1, 2), (3, 4), (5, 6)]]</code>	
<code>[w[0] for w in ['Python', 'WWU']]</code>	
<code>['klein' if x < 10 else 'groß' for x in [3, 2345, 11, 10]]</code>	

Lösung 1

Was schreibt das folgende Skript auf den Bildschirm?

```
liste = ["mond", "stoff", "treib", "raum", "schiff"]
print(liste[0])
print(liste[2] + liste[1])

print(liste[-2] + liste[-1])
for wort in liste:
    if wort[0] == "s":
        print(wort)

for wort in liste:
    print(wort[1])

liste = liste + ["gestein"]
print(liste[0] + liste[-1])
```

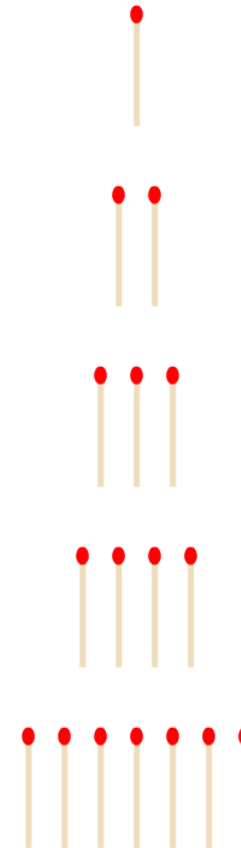
mond
treibstoff
raumschiff
stoff
schiff
o
t
r
a
c
mondgestein

Lösung

List Comprehension	Liste
<code>[a**2 for a in range(5)]</code>	<code>[0, 1, 4, 9, 16]</code>
<code>[(a, a+1) for a in range(3)]</code>	<code>[(0, 1), (1, 2), (2, 3)]</code>
<code>[(b, a) for (a, b) in [(1, 2), (3, 4), (5, 6)]]</code>	<code>[(2, 1), (4, 3), (6, 5)]</code>
<code>[w[0] for w in ['Python', 'WWU']]</code>	<code>['P', 'W']</code>
<code>['klein' if x < 10 else 'groß' for x in [3, 2345, 11, 10]]</code>	<code>['klein', 'groß', 'groß', 'groß']</code>

Programmierung des Nim-Spiels

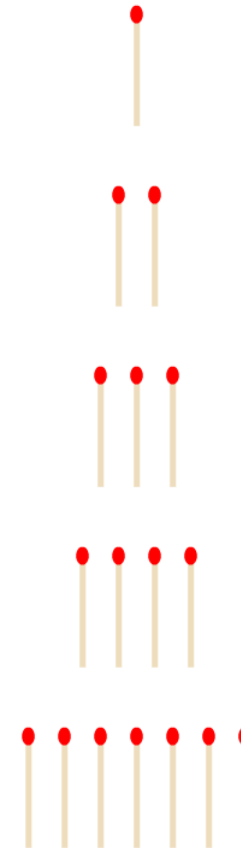
- Zug: Aus einer Zeile ein oder mehrere Streichhölzer nehmen
- Wer das letzte Hölzchen nimmt, gewinnt.



Modell für die Streichhölzer des Nim-Spiels

```
streichhölzer = [[1],  
                 [1, 1],  
                 [1, 1, 1],  
                 [1, 1, 1, 1],  
                 [1, 1, 1, 1, 1, 1, 1, 1]]
```

```
streichhölzer = [1, 2, 3, 4, 7]
```



1. Iteration

```
from random import randint
```

```
def ausgabe():  
    for i in streichhölzer:  
        print(i)
```

Definition einer Funktion

```
def mensch_zieht():  
    input()
```

```
def ki_zieht():  
    pass
```

Leere Anweisung
(macht nichts)

```
def runde():  
    mensch_zieht()  
    ausgabe()  
    if sum(streichhölzer) == 0:  
        print("Du hast gewonnen!")  
    else:  
        ki_zieht()  
        ausgabe()  
        if sum(streichhölzer) == 0:  
            print("Du hast verloren!")
```

```
streichhölzer = [1, 2, 3, 4, 7]  
ausgabe()  
while sum(streichhölzer) != 0:  
    runde()
```

Aufruf einer Funktion

2. Iteration

```
from random import randint

def ausgabe():
    for i in streichhölzer:
        print(i)

def mensch_zieht():
    reihe = int(input("Reihe (1-5): ")) - 1
    anzahl = int(input("Wieviele Streichhölzer nimmst du? "))
    streichhölzer[reihe] -= anzahl

def ki_zieht():
    ok = False
    while not ok:
        reihe = randint(0, 4)
        if streichhölzer[reihe] > 0:
            streichhölzer[reihe] -= 1
            ok = True
    print("Ich nehme 1 Streichholz aus Reihe", reihe + 1)
```

```
1
2
3
4
7
Reihe (1-5):
```

Übung 2.2

Verbessern Sie das Starterprojekt `nim_1.py`

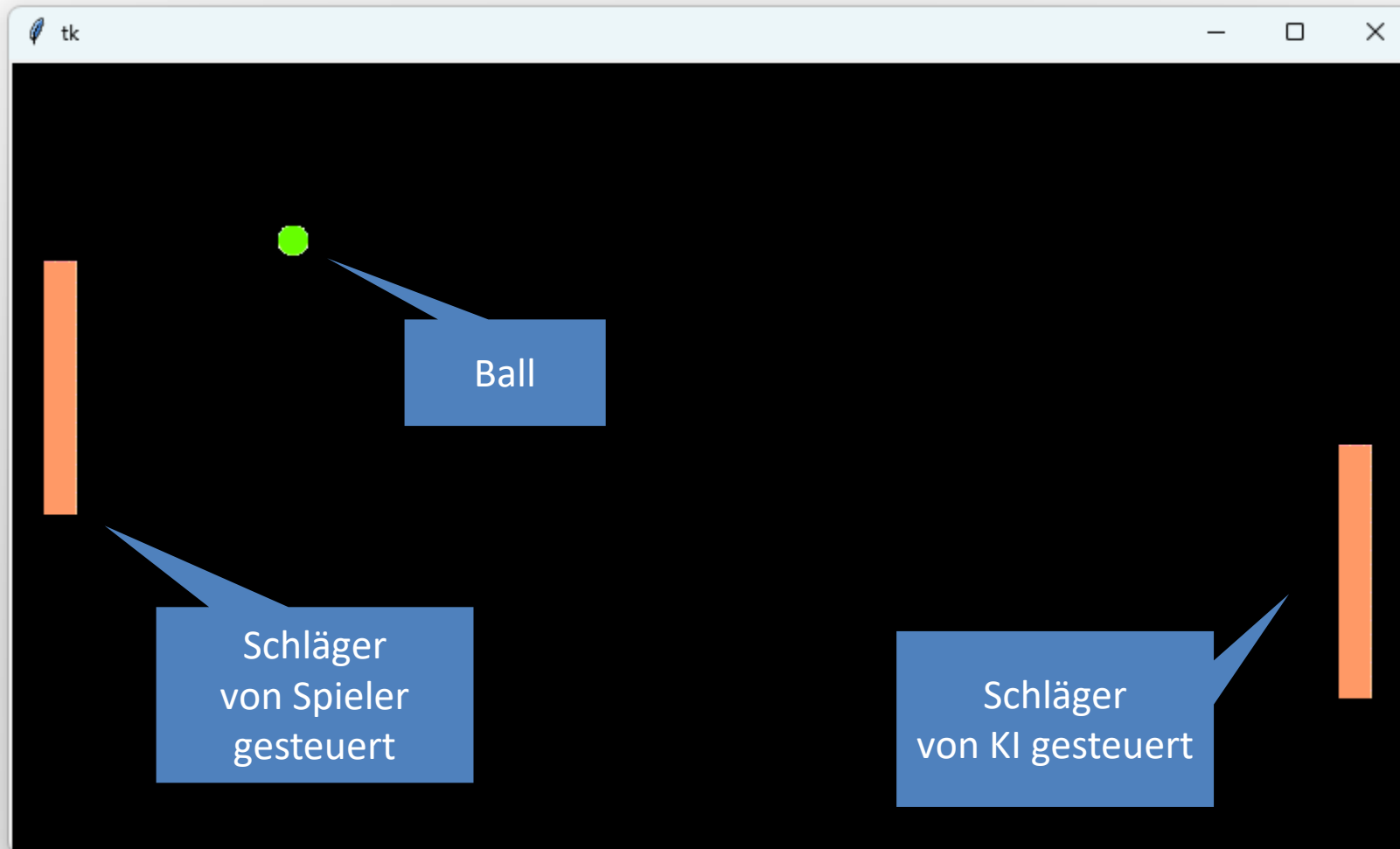
- Machen Sie das Verhalten der KI menschenähnlicher.
- Verbessern Sie die Benutzungsschnittstelle. Sorgen Sie dafür, dass ein Spieler keine ungültigen Eingaben machen kann.

Weitere Ideen:

https://docs.google.com/document/d/140INsIEWA_AwUwtpVMCZqtH7_eOkU8rmvfrgWqE5M3E/edit?usp=sharing

Pong(*)

Anwendungsfenster



Die Tick-Metapher



Bei jedem Tick ändert sich der Zustand der Spielelemente.

Der Ball

```
from tkinter import *
import random
```

```
STEP = 50    # Millisekunden
```

```
class Ball:
    def __init__(self, canvas):
        self.canvas = canvas
        self.img = PhotoImage(file='ball.gif')
        self.iD = self.canvas.create_image(0, 0,
                                           image=self.img)

        self.start()

    def start(self):
        x = int(self.canvas['width'])/2
        self.canvas.coords(self.iD, x, 1)
        self.vy = 3
        self.vx = random.choice([-3, 3])

    def bounce(self):
        if self.vx > 0:
            self.vx = -self.vx -1
        else:
            self.vx = -self.vx +1
        self.canvas.move(self.iD, self.vx, self.vy)
```

```
    def tick(self):
        x, y = self.canvas.coords(self.iD)
        if 0 < int(x) < int(self.canvas['width']):
            self.canvas.move(self.iD, self.vx, self.vy)
            x, y = self.canvas.coords(self.iD)
            if not(0 < int(y) < int(self.canvas['height'])):
                self.vy = - self.vy
        else:
            self.start()
```

Schläger vom Spieler gesteuert

```
class Bat:
    def __init__(self, canvas, ball, x):
        self.canvas, self.ball = canvas, ball
        self.img = PhotoImage(file='bat.gif')
        self.iD = self.canvas.create_image(x, 0,
                                           anchor=NW, image=self.img)

    def up(self, event):
        self.canvas.move(self.iD, 0, -5)

    def down(self, event):
        self.canvas.move(self.iD, 0, 5)

    def tick(self):
        x1, y1, x2, y2 = self.canvas.bbox(self.iD)
        if self.ball.iD in self.canvas.find_overlapping(x1, y1, x2, y2):
            self.ball.bounce()
```



Eventhandler

Schläger von KI gesteuert

```
class BatAI:
    def __init__(self, canvas, ball, x):
        self.canvas, self.ball, = canvas, ball
        self.img = PhotoImage(file='bat.gif')
        self.iD = self.canvas.create_image(x, 220,
                                           anchor=NW, image=self.img)

    def tick(self):
        x1, y1, x2, y2 = self.canvas.bbox(self.iD)
        ballX, ballY = self.canvas.coords(self.ball.iD)
        middle = (y1 + y2) // 2

        # Schläger bewegen (hier fehlt etwas)

        if self.ball.iD in self.canvas.find_overlapping(x1, y1, x2, y2):
            self.ball.bounce()
```

Bounding Box des Schlägers

Höhe des Mittelpunkts des
Schlägers

Anwendungs- fenster

Tastaturevents werden an
Methoden gebunden

Gameloop

```
class Pong:
    def __init__(self):
        self.window = Tk()
        self.canvas = Canvas(master=self.window, bg="black",
                               width=800, height=450)

        self.canvas.pack()
        self.ball = Ball(self.canvas)
        self.leftBat = Bat(self.canvas, self.ball, 20)
        self.rightBat = BatAI(self.canvas, self.ball, 760)
        self.window.bind('<KeyPress-Up>', self.leftBat.up)
        self.window.bind('<KeyPress-Down>', self.leftBat.down)
        self.window.after(STEP, self.tick)
        self.window.mainloop()

    def tick(self):
        self.ball.tick()
        self.leftBat.tick()
        self.rightBat.tick()
        self.window.after(STEP, self.tick)
```

Pong()

Übung 2.3 *

Verbessern Sie das Starterprojekt `pong_aufgabe.py`.

Implementieren Sie eine einfache KI, die den rechten Schläger bewegt, so dass man glauben kann, er würde von einem Menschen gesteuert.

Tipp: Sorgen Sie dafür, dass der Schläger sich auf die aktuelle Höhe des Balls zubewegt.

Lösung 2.2 Nim

Sichere Eingabe

```
def mensch_zieht():
    ok = False
    while not ok:
        try:
            reihe = int(input("Reihe (1-5): ")) - 1
            anzahl = int(input("Wieviele Streichhölzer nimmst du? "))
            if 0 <= reihe <= 4:
                if anzahl < 1:
                    print("Mindestens 1 Streichholz ziehen!")
                elif anzahl <= streichhölzer[reihe]:
                    streichhölzer[reihe] -= anzahl
                    ok = True
                else:
                    print("Nicht genügend Streichhölzer in der Reihe!")
            else:
                print("Nummer der Reihe muss zwischen 1 und 5 liegen")
        except:
            print("Bitte Zahlen eingeben!")
```

Zufällige Anzahl

```
def ki_zieht():
    ok = False
    while not ok:
        reihe = randint(0, 4)
        if streichhölzer[reihe] > 0:
            n = randint(1, streichhölzer[reihe])
            streichhölzer[reihe] -= n
            ok = True
    if n == 1:
        print("Ich nehme 1 Streichholz aus Reihe", reihe + 1)
    else:
        print("Ich nehme", n, "Streichhölzer aus Reihe", reihe + 1)
```

Lösung 2.3

```
class BatAI:

    def tick(self):
        x1, y1, x2, y2 = self.canvas.bbox(self.iD)
        ballX, ballY = self.canvas.coords(self.ball.iD)
        middle = (y1 + y2) // 2
        if middle > ballY:
            self.canvas.move(self.iD, 0, -5)
        else:
            self.canvas.move(self.iD, 0, 5)
        if self.ball.iD in self.canvas.find_overlapping(x1, y1, x2, y2):
            self.ball.bounce()
```


Rückblick

- Daten können in Kollektionen zusammengefasst werden. Dazu gehören Sequenzen (Strings, Listen und Tupel) und Mengen.
- In einer Sequenz haben die Elemente (Items) eine bestimmte Reihenfolge und können über den Index angesprochen werden.
- Es gibt gemeinsame Operationen für Sequenzen (`in`, `len()`, `sum()`, `max()`, ...).
- Mit `list()` kann aus einer beliebigen Sequenz eine Liste gewonnen werden.
- Eine Liste ist eine veränderbare Sequenz, d.h. Elemente können geändert, gelöscht oder hinzugefügt werden.
- Mit Listen und Tupeln können Strukturen der Wirklichkeit modelliert werden. Die Streichhölzer des Nim-Spiels können z.B. durch eine Liste von Zahlen modelliert werden.
- Eine KI kann einen menschlichen Gegner in einem Spiel simulieren (z.B. Nim oder Pong).