Schrittweise Ausführung[1]

Das Grafiksystem optimiert die Darstellung, indem es abwartet, bis das darzustellende Bild fertig generiert wird. Baut man daher in die Testanwendung Methodenaufrufe ein, werden diese erst vollständig bearbeitet, bevor das Grafikfenster angezeigt wird.

Schrittweise Ausführung: So geht's nicht!

```
    def TestAnwendung(self):
        # Allein fuer die Testanwendung:
        global stuhl1, stuhl2
        stuhl1=Stuhl()
        stuhl2=Stuhl()
        stuhl1.Zeige()
        stuhl2.Zeige()
        for i in range(100): stuhl1.BewegeHorizontal(1)
        for i in range(100): stuhl1.BewegeHorizontal(-1) # hebt das auf
```

Es wird also nur die Endstellung angezeigt, so dass sich die beiden Stuhlobjekte decken. Zum Testen kann man im ShellFrame **`stuhl2.BewegeHorizontal(10)`** aufrufen und sieht nun beide Stuhlobjekte.

Schrittweise Ausführung: Wie geht's?

Zur Lösung kann man auf die Klasse **`timer`** zugreifen. Dafür werden hier zwei Varianten gezeigt.

Lösung mit Bind

Im Projekt mit der schrittweisen Ausführung verwendet die App der Stuhlklasse in ihrer OnInit-Methode ihre Bind-Methode, um die Anwendung eines wx.Timer-Objektes an eine auszuführende Methode zu binden.

```
class StuhlApp(wx.App):
    def OnInit(self):
        self.__fenster = GrafikFenster(None, "Raumplaner-Grafik")
        ...
```

1 Projekt ***Raumplaner-schrittweise-Ausfuehrung***

```
        self.TestAnwendung()
        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.Drehe, self.timer)
        self.SchrittweiseDrehen()
        return True
```

Erläuterung:

- Es wird ein wxTimer-Objekt erzeugt, dessen owner das StuhlApp-Objekt ist,
- das Ereignis an die später zu definierende Methode Drehe gebunden
- und die Methode SchrittweiseDrehen aufgerufen.

Diese Methode definiert die Anzahl der Schritte und ruft dann **`self.timer`** mit der Unterbrechung in ms auf. Während **`self.timer`** die Bearbeitung das erstemal unterbricht, kann der derzeitige Zustand des Fensters dargestellt und bei den nachfolgenden Unterbrechungen jeweils aktualisiert werden.
Die Definition (**`Bind`**) kann auch in einer Methode erfolgen und die schrittweise Ausführung aus dem ShellFrame aufgerufen werden(mit **`app.SchrittweiseGehen()`**), wie die in die App ebenfalls eingebaute Methode zeigt:

```
    def SchrittweiseGehen(self):
        self.timer2 = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.GeheSchritt, self.timer2)
        self.anzahl = 50
        self.timer2.Start(100)

    def GeheSchritt(self, evt):
        stuhl.BewegeHorizontal(3)
        stuhl.BewegeVertikal()
        self.anzahl-=1
        if self.anzahl<1: self.timer2.Stop()
```

Lösung mit einer eigenen Klasse Schritte

Zunächst die Klasse[1]:

```
import wx

class Schritte(wx.Timer):
    '''Klasse fuer die schrittweise Ausfuehrung'''
    def __init__(self, owner, anzahlSchritte, schrittProzedur):
        wx.Timer.__init__(self, owner)
        self.anzahl = anzahlSchritte
        self.schrittProzedur=schrittProzedur

    def Notify(self):
        '''Diese jeweils aufgerufene Methode muss auch dafuer
        sorgen, dass irgendwann abgebrochen wird.'''
        self.schrittProzedur()
        self.anzahl-=1
        if self.anzahl<0:
            self.Stop()
```

Dem Konstruktor muss hier nicht nur der owner (die App) übergeben werden, sondern auch die Anzahl der Schritte und die Prozedur, die bei jedem Schritt ausgeführt werden soll[2].

1 Projekt `schritte.py`, dessen Klasse mit **`from schritte import Schritte`** eingebunden werden kann.
2 Das ließe sich auch noch so erweitern, dass Aufrufparameter für die Prozedur übergeben werden können.

Im betrachteten Projekt enthält die App für die Tischklasse die notwendigen Definitionen und Aufrufe:

```
    def SchrittProzedur(self):
        '''definiert, was in jedem Schritt zu tun ist'''
        tisch.BewegeHorizontal()
        tisch.BewegeVertikal(2)

    def Schrittweise(self):
        '''verwendet die eigenstaendige Klasse Schritte'''
        self.schritte=Schritte(self, 30, self.SchrittProzedur)
        self.schritte.Start(100)
```

Die Methode **`Schrittweise()`** kann dann am Schluss der **`OnInit`** – Methode (**`self.Schrittweise(self)`**) oder im ShellFrame (app.Schrittweise()) aufgerufen werden. Sie weist in diesem Beispiel 30 Schritte eine Schrittzeit von 100 ms an.

Alternative mit threads

Eine Alternative ist der Einsatz von threads[1] zusammen mit der time-Funktion sleep. Also zunächst:

```
import time
import thread
```

Ein Beispiel zeigt die folgende Testanwendung:

```
    def TestAnwendung(self):
        """Testanwendung mit threads"""
        self.tisch=Tisch(sichtbar=True)
        self.stuhl=Stuhl(sichtbar=True)
        self.sessel=Sessel(sichtbar=True)
        thread.start_new_thread(self.Action, (0.1, 30))

    def Action(self, sec, anzahl):
        """Allein fuer die Testanwendung:"""
        for i in xrange(anzahl):
            self.tisch.BewegeHorizontal(2)
            self.tisch.BewegeVertikal(3)
            self.stuhl.BewegeHorizontal(1)
            self.stuhl.BewegeVertikal(5)
            self.stuhl.Drehe(3)
            self.sessel.Drehe(10)
            time.sleep(sec)
```

Da der thread jeweils für sec Sekunden "schlafen" geschickt wird, kann das Grafiksystem in dieser Zeit jeweils den aktuellen Zustand darstellen. So entsteht ebenfalls der Eindruck einer Bewegung.

1 Auch im Deutschen wird meistens der englische Begriff verwendet statt *nebenläufige Prozesse*.