Grafik bei Python einsetzen

Im Projekt zur Entwicklung eines Einrichtungsplaners (*Raumplaner*)[1] wird von mir standardmäßig das Grafiktool wxPython eingesetzt. Wie der nachfolgende Text zeigen soll, gibt es dafür einige gute Gründe.

wxPython

Ein wichtiger Grund ist die Möglichkeit mit wxPython eine Arbeitsweise entsprechend der Vorgaben von BlueJ bei Java zu ermöglichen.

- Es gibt entsprechend der Klasse **`GraphicPath`** von Java im **`GraphicContext`** von wxPython die Methode **`gc.createPath()`**, die einen solchen Pfad erzeugt, der sich in gleicher Weise handhaben lässt (Elemente hinzufügen usw, insbesondere aber die Möglichkeit Affine Transformationen (*hier verwendet Verschiebung und Drehung*) für alle diese Objekte nutzen zu können.
- Mit dem ShellFrame von wxPython gibt es eine Oberfläche, die (*ähnlich wie bei BlueJ*) eine Kommunikation mit laufenden Anwendungen ermöglicht.

tkinter

Ein wichtiger Grund, der für das Grafiktool tkinter spricht: Es ist bei jeder Standard-installation von Python dabei[2]. Nachteilig ist, dass die Dokumentation von tkinter nicht so gut ist, wie die von wxPython. Das betrifft leider gerade die Klasse **`Canvas`** (Zeichenfläche), die wir für unsere Grafikdarstellung benötigen. Einige Kleinigkeiten erleichtern allerdings das Arbeiten mit dieser Grafik.

- Zu jedem Grafikobjekt, das der Zeichenfläche hinzu gefügt wird, wird eine **`ID`** erzeugt und es kann ein **`tag`** (*ein String*) zugeordnet werden, über das es beim Aufruf von vorgegebenen Konfigurationsmethoden angesprochen werden kann. Dabei ist es auch möglich, das **`tag`** mehreren Objekten zuzuordnen, so dass darüber ein Gruppenzugriff möglich wird.
- Bei der Verarbeitung von Mausaktionen kann man eine Methode nutzen, bei der das nächst gelegene Objekt angesprochen wird, so dass die Bestimmung eines Objekts zu einem Ereignis nicht selbst programmiert werden muss.

Das von Benjamin Moll entwickelte Projekt zum Möbelplaner auf der Basis von tkinter ist durchaus für den Unterricht nutzbar, weicht in einigen Konzepten aber leicht von meinem Projektvorschlag ab. Bei seinem Einsatz müsste also einiges Material überarbeitet werden.
Nachteil:

- Durch das Fehlen der Transformationen ist es zwar möglich Objekte einfach zu verschieben. Drehungen sind allerdings nicht möglich. Man kann sie teilweise natürlich selbst programmieren, was bei einigen Figuren auch gelingt. Bei Ellipsen ist das mit einem erzeugten Ellipsenobjekt jedoch nicht möglich, so dass man zwar noch die Drehung um 90° mit vertretbarem Aufwand hin bekommt, andere Drehwinkel jedoch nicht.

1 Die Wahl dieses Kontextes ist natürlich beliebig. Allerdings gibt es für die Entwicklung eines Grafiksystems als Kontext einen wesentlichen Grund: Die Schülerinnen und Schüler können in der Regel direkt die Auswirkung von Arbeiten am Programm sehen und können so selbstständiger kontrollieren und korrigieren.
2 Das ist auch der Grund, weshalb ich selbst einige Softwareprojekte mit tkinter entwickelt habe.

Gemeinsames

Grundsätzlich gemeinsam sind die grundlegenden Konzepte objektorientierter Software, wie sie von Python umgesetzt werden. Beispiele also:

- Genau ein Konstruktor (`__init__`in jeder Klasse statt overload (mehrere mit unterschiedlicher Signatur) und entsprechend nur genau eine Methode zu jedem Namen.
- Beide mit der Möglichkeit eine variable Anzahlen von Parametern, vordefinierte Parameterwerte und Schlüsselwort-Parameter anbieten zu können