Schildkrötengrafik zeitgemäß

Mit dreidimensionaler Turtle-Grafik genetisch zur objektorientierten Programmierung

von Reinhard Oldenburg, Magnus Rabel und Jan Schuster

Schildkrötengrafik (*turtle graphics*) als Komponente der didaktischen Programmiersprache LOGO wurde und wird vielfach eingesetzt, um Grundlagen der Algorithmik zu vermitteln. Im folgenden Beitrag wird beschrieben, wie die Verlagerung der Schildkröte in den dreidimensionalen Raum die grundlegenden Ideen auch im heutigen Kontext für Schülerinnen und Schüler motivierend machen kann.

Turtle-Grafik

Die Idee der Turtle-Grafik – also einer durch einfache Befehle steuerbaren Figur, die sich in einer virtuellen Welt bewegen und Linien zeichnen kann – ist eine der einflussreichsten in der Geschichte der Didaktik der Informatik, und es gibt eine reiche Literatur dazu (vgl. u.a. Abelson/diSessa, 1981; Hromkovič, 2009). Die ursprünglich an die Sprache LOGO gekoppelte Turtle (vgl. Papert, 1980) wurde auch in vielen anderen Sprachen (DELPHI, JAVA usw.) realisiert und avancierte zum Inbegriff einer Mikrowelt, die zu spezialisierten und fortgeschrittenen Mikrowelten wie SCRATCH, NETLOGO und KARA weiterentwickelt wurde.

Die Vielzahl der Projekte und Unterrichtskonzepte, in denen mit der Turtle-Grafik gearbeitet wurde, belegt die didaktische Nützlichkeit des Konzepts. Selbst in der Mathematikdidaktik wurden umfangreiche Belege gesammelt, die positive Auswirkungen, insbesondere auf die geometrische Begriffsbildung belegen (vgl. Blume/Heid, 2008).

Projekte zur Erweiterung der Turtle in den Raum hat es auch schon gegeben (vgl. Paysan, 1999; Wolfram Demonstrations Project; ELICA; NETLOGO 3D), aber sie haben bisher deutlich weniger Zuspruch gefunden als die zweidimensionale Turtle. Das mag daran liegen, dass die meisten dieser Projekte bei LOGO als Programmiersprache geblieben sind und damit unter der kleiner werdenden Akzeptanz dieser Sprache (u.a. wegen ihrer fehlenden Möglichkeiten zur objektorientierten Programmierung) leiden.

Motivation

Im Rahmen des Schulversuchs *Genetischer Informatikunterricht* (vgl. Schuster, 2011, und Oldenburg/Rabel/Schuster, 2012) haben wir nach einer geeigneten Umgebung für den Einstieg in die Algorithmik gesucht. Dabei sollte die Perspektive auf die OOP eröffnet oder zumindest nicht verbaut werden. Die klassische Turtlegrafik war eine Option, aber bei Schülerinnen und Schülern in der Sekundarstufe II erschienen uns die damit erstellbaren Bilder nicht hinreichend motivierend zu sein. Da die Lernenden Programme mit dreidimensionalen Darstellungen (z.B. Computerspiele) schon kennen, lag es nahe, diesen Anknüpfungspunkt zu verwenden. Es wurde also eine Bibliothek für die im Schulversuch eingesetzte Sprache PYTHON entwickelt, mit der eine Turtle gesteuert werden kann.

Möglichkeiten der 3-D-Turtle

Der folgende Programmtext lädt die Bibliothek, erzeugt eine 3-D-Turtle und führt einige ihrer Methoden aus, die zusammen eine unterbrochene Linie und ein Quadrat erzeugen. Das Programm kann interaktiv über die Konsole ausgeführt oder aus einer Datei gestartet werden.

```
from VTools import *
t = Turtle3D()
t.forward(1)
t.penUp()
t.forward(1)
t.penDown()
t.forward(1)
t.turnRight(90)

t.setColor(color.blue)
for i in range(4):
  t.turnUp(90)
  t.forward(3)
t.forward(2)
```

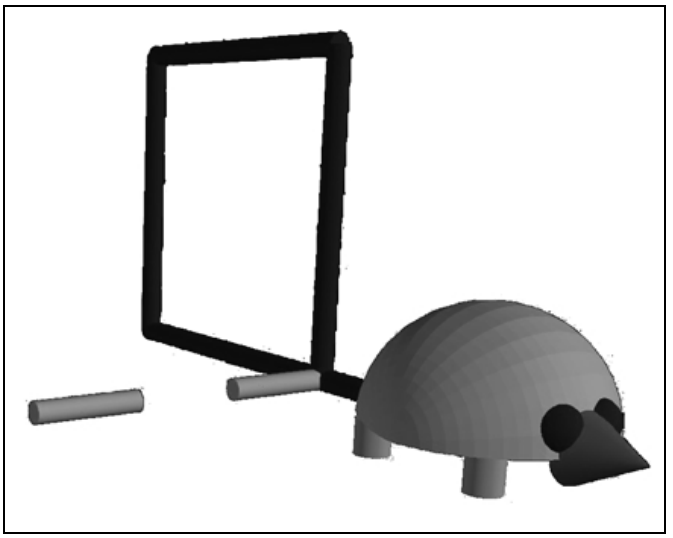

Bild 1: Die 3-D-Turtle zeichnet ein Quadrat.

Das Ergebnis dieses Programms wird im Bild 1 gezeigt. Erfreulicherweise reichen sehr wenige Befehle zur Steuerung der Turtle aus. Alle Befehle in der Tabelle (siehe nächste Seite) sind Methoden, müssen also auf ein Turtle3D-Objekt angewendet werden.

Algorithmik mit der 3-D-Turtle

Nachdem die Schülerinnen und Schüler sich mit der grundlegenden Funktionsweise vertraut gemacht hatten, konnten sie frei zeichnen. Dabei sind viele komplexe Zeichnungen entstanden. Ein Schüler ließ beispielsweise ein ganzes Haus zeichnen. Im Programmtext dazu wiederholen sich viele ähnliche Abschnitte, etwa:

```
t.forward(10)
t.turnRight(90)
t.forward(10)
t.turnRight(90)
t.forward(10)
t.turnRight(90)
```

Wenn solche Wiederholungen auftreten, stellt sich von selbst die Frage, ob das nicht schneller geht, ob man also nicht automatisch Befehle wiederholen lassen kann. Damit ist ein genetischer Weg zur *Schleife* beschritten: Das Konzept der Schleife beantwortet genau die sich aus der Erfahrung der Schülerinnen und Schüler ergebende Frage.

Auch das elementare Konzept der *Referenz durch Variablen* ergibt sich schnell: Der Wunsch, einen Teil der Zeichnung anders zu dimensionieren (z.B. die Seitenlänge eines Würfels) führt zu einer fehleranfälligen Suche, wo überall Änderungen durchzuführen sind. Es ist viel besser, die Abmessungen vorab festzulegen, indem man Variablen definiert, deren Wert die Größe bestimmter Bereiche festlegt. Damit ist die Zeichnung durch Referenz auf vorgegebene Zahlen parametrisiert und kann schnell mit anderen Werten wiederholt werden.

Aus der Erkenntnis, dass bestimmte grundlegende Operationen, z.B. das Zeichnen eines Rechtecks, immer wieder auftreten, ergibt sich der Wunsch nach *prozeduraler Abstraktion*.

```
def rechteck(a,b):
  for i in range(2):
    t.forward(a)
    t.turnDown(90)
    t.forward(b)
    t.turnDown(90)
```

Analog sind ebene regelmäßige Vielecke, Kreise oder räumliche Winkel und Sterne gute Kandidaten für wiederverwendbare Formen. Die folgende Funktion lässt sich nutzen, um sehr einfach einen Zylinder zu erzeugen:

```
def zylinder(radius, hoehe, n = 150):
  for i in range(n):
    t.backward(radius)
    rechteck(2*radius, hoehe)
    t.forward(radius)
    t.turnLeft(360.0/n)
```

Solche Funktionen (bzw. Prozeduren – je nach Sprachgebrauch) zeigen auch eine wichtige Rolle von Funktionen: Es handelt sich um *Konstruktoren*. Sie unterscheiden sich nur insofern von vorgegebenen Konstruktoren wie Turtle3D(), dass letztere das konstruierte Objekt zurückliefern, sodass man später darauf verweisen kann. Dies ermöglicht bei der Turtle u.a., mit mehreren Schildkröten zu arbeiten.

Die obige Art der Funktionsdefinition hat jedoch noch einen gravierenden Nachteil, der sich zeigt, sobald man mehr als eine Turtle benutzt: Es wird nämlich immer auf die gleiche Turtle verwiesen; die Wiederverwendbarkeit des Programmtextes ist dadurch eingeschränkt. Man sollte besser wie folgt formulieren:

```
def rechteck(dieseTurtle, a, b):
  for i in range(2):
    dieseTurtle.forward(a)
    dieseTurtle.turnDown (90)
    dieseTurtle.forward(b)
    dieseTurtle.turnDown(90)
```

Für mehrere Turtles gibt es eine Reihe sinnvoller Anwendungen: Eine Fläche kann schraffiert werden, indem zwei Turtles längs der Ränder laufen und die „Schraffierturtle“ immer zwischen ihnen hin und her geht. Eine weitere Möglichkeit stellen etwa Verfolgungsprobleme dar. Dazu lässt sich leicht eine Tastatursteuerung der Turtles realisieren (ein Beispiel für ein solches Spiel findet sich auf der Homepage des Schulversuchs *Genetischer Informatikunterricht*.

OOP mit der 3-D-Turtle

Die Turtle leistet auch eine Vorbereitung auf Konzepte der *objektorientierten Programmierung*. Jede Turtle besitzt *Attribute* (Stiftfarbe, Stiftposition usw.) und eigene Methoden (gehe vorwärts, gehe rückwärts

Tabelle: Turtle-Befehle.

Methode	Bedeutung: Die Turtle …	Beispiel
`forward(a)`	geht a Einheiten vorwärts	`t.forward(5)`
`backward(a)`	geht a Einheiten rückwärts	`t.backward(5)`
`turnLeft(w)`	dreht sich um die angegebene Gradzahl nach links	`t.turnLeft(45)`
`turnRight(w)`	dreht sich um die angegebene Gradzahl nach rechts	`t.turnRight(45)`
`turnUp(w)` `turnDown(w)`	dreht sich um die angegebene Gradzahl nach oben bzw. unten	`t.turnUp(45)`
`penUp()` `penDown()`	schaltet den Stift aus bzw. an. Das Beispiel rechts zeichnet eine unterbrochene Linie	`t.forward(30)` `t.penUp()` `t.forward(30)` `t.penDown()` `t.forward(30)`
`setVisible(b)`	Wenn b *true* ist, ist die Turtle sichtbar	`t.setVisible(False)`
`setAnimated(b)`	Wenn b *true* ist, bewegt sich die Turtle langsam animiert. Ist b *false*, so erscheint sofort das Ergebnis der Zeichnung	`t.setAnimated(False)`
`setColor(farbe)`	setzt die Farbe, z.B. *color.red*, *color.blue* etc. oder als RGB-Werte mit Komponenten aus 0..1	`t.setColor(color.yellow)` `t.setColor((1,0,0))` `t.setColor((0.5,0.2,0))`
`setThickness`	setzt die Strickdicke; Standardwert ist 0.1	`t.setThickness(0.5)`
`goto((x,y,z))`	geht zu (x,y,z)	`t.goto((1,5,-3.2))`
`lookdir(richtung)`	Turtle blickt in Richtung des Vektors	`t.lookdir((0,1,0))`

usw.). Diese werden von Beginn an in der Schreibweise verwendet, die auch in der späteren OOP relevant ist. Um eine Verwechslung von Klassen und Objekten zu vermeiden, sollte früh die Möglichkeit genutzt werden, mehr als eine Turtle zu erzeugen. Wenn dies frühzeitig geschieht, verstehen die Schülerinnen und Schüler zum einen, warum man (übrigens anders als bei LOGO) die individuelle Turtle (hier: t1) vor dem Punkt immer angegeben muss, und sie lesen t1.penDown() als Anweisung an ein Objekt t1, seinen Zustand zu ändern.

Im Unterricht gab es den Schülerwunsch, eigene Funktionen für die Turtle genauso anzuwenden wie die eingebauten – also auch mit der Punktnotation. Das ermöglicht im Prinzip schon an dieser Stelle einen genetischen Weg in die Objektorientierung, der folgendermaßen laufen könnte: Ausgangspunkt ist die Implementation irgendeiner Funktion, die etwas Sinnvolles mit einer Turtle macht, z.B. einen Kreis (oder wie oben ein Rechteck) zu zeichnen:

```
from VTools import *
from math import pi
t = Turtle3D()
t.setAnimated(False)

def kreis(dieTurtle, radius):
  n = 100
  for i in range(n):
    dieTurtle.turnLeft(360.0/n)
    dieTurtle.forward(2*pi*radius/n)

kreis(t, 5)
kreis(t, 8)
```

In Analogie zu den „eingebauten“ Turtle-Befehlen erwartet man eigentlich, dass man auch t.kreis(5) schreiben kann. Der Weg, diesem Schülerwunsch zu entsprechen, erfordert nur, dass man exakt(!) den gleichen Programmtext wie oben in eine Klasse verpackt, die von Turtle3D erbt:

```
class KlugeTurtle(Turtle3D):
  def kreis(dieseTurtle, radius):
    n = 100
    for i in range(n):
      dieseTurtle.turnLeft(360.0/n)
      dieseTurtle.forward(2*pi*radius/n)

t2 = KlugeTurtle()
t2.kreis(10)
```

In PYTHON wird das erste Argument einer Methode per Konvention self genannt, aber gegen diese Konvention wird hier so verstoßen, dass es verständnisfördernd ist: Die Lernenden können erkennen, dass das erste Argument für das Objekt steht, auf das die Methode angewendet wird.

Bemerkenswerterweise kommt so eine der anspruchsvollsten Aspekte der OOP, nämlich die *Vererbung*, schon am Anfang vor. Trotzdem kann man von einem genetischen Vorgehen sprechen, weil sich die Entwicklung aus dem Fragehorizont der Schülerinnen und Schüler ergibt.

Außerdem kann das Beispiel als Muster für den Erwerb von Grundvorstellungen der OO gelten. Eine bekannte Grundvorstellung ist die des *Bauplans*: Eine Klasse ist der Bauplan, der angibt, wie etwas zu konstruieren ist; und der Aufruf des Konstruktors löst den Bau gemäß Bauplan aus. Mit dieser Grundvorstellung (vgl. Rabel, 2011) lassen sich vor allem die Attribute erklären. Die Methoden passen dagegen nicht so gut zur Vorstellung „Klasse als Bauplan“. Die Schildkröte erweitert die Sicht von passiven technischen Objekten (für die Baupläne üblicherweise verwendet werden) hin zu Lebewesen, die neben dem Bauplan auch ein Verhaltensrepertoire (Methoden) besitzen, das sich durch Dazulernen erweitern lässt.

Fazit

Die Erfahrung im Unterricht zeigt, dass die 3-D-Turtle eine gewisse Faszination auf Schülerinnen und Schüler der gymnasialen Oberstufen ausübt. Diese kann kurzzeitig sogar noch gesteigert werden, indem man den *Stereomodus* einschaltet, wodurch die Darstellung automatisch für Farbfilter-3-D-Brillen optimiert wird. So sind die Lernenden mehr als ausreichend motiviert, sich in einem ersten Anlauf den Schwierigkeiten der Algorithmik zu stellen. Natürlich müssen Konzepte wie *Schleife* und *Konstruktor* danach dekontextualisiert werden, um transferierbar zu werden. Aber ein anregender Einstieg ist auf diese Weise geschafft.

Prof. Dr. Reinhard Oldenburg
Universität Augsburg
Universitätsstraße 14
86159 Augsburg

E-Mail: reinhard.oldenburg@math.uni-augsburg.de

StD Magnus Rabel
Kaiserin-Friedrich-Gymnasium
Auf der Steinkaut 1–15
61352 Bad Homburg v. d. H.

E-Mail: m.rabel@kaiserin-friedrich.de

StR Jan Schuster
Goethe-Universität Frankfurt am Main
Institut für Didaktik der Mathematik
und der Informatik
Robert-Mayer-Straße 6–8
60325 Frankfurt

E-Mail: schuster@math.uni-frankfurt.de

Literatur und Internetquellen

Abelson, H.; diSessa, A.: Turtle Geometry – The Computer as a Medium for Exploring Mathematics. Cambridge (USA, MA): MIT Press, 1981.

Blume, G.W.; Heid, M.K.: Research on Technology and the Teaching and Learning of Mathematics. Vol. 1: Research Syntheses. Vol. 2: Cases, and Perspectives. Charlotte (USA, NC): Information Age Publishing, 2008.

ELICA:
http://www.elica.net/

Hromkovič, J.: Einführung in die Programmierung mit LOGO – Lehrbuch für Unterricht und Selbststudium. Wiesbaden: Vieweg+Teubner, 2009.

NetLogo und NetLogo 3D:
https://ccl.northwestern.edu/netlogo/

Oldenburg, R.; Rabel, M.; Schuster, J.: A Turtle's genetic path to Object Oriented Programming. In: Chr. Kynigos, J. E. Clayson, N. Yiannoutsou (Hrsg.): Constructionism 2012 – Theory Practice and Impact – Conference Proceedings, August, 21–25, Athens Greece. Athen (Griechenland): National & Kapodistrian University of Athens, The Educational Technology Lab, 2012, S. 74–82.

Papert, S.: Mindstorms – Children, Computers, and Powerful Ideas. New York: Basic Books, 1980 [deutsch: Mindstorms – Kinder, Computer und neues Lernen. Basel; Boston; Stuttgart: Birkhäuser, 1982].

Paysan, B.: "Dragon Graphics" – Forth, OpenGL und 3D-Turtle-Graphics. 22. April 1999.
http://bernd-paysan.de/dragongraphics.pdf

Rabel, M.: Grundvorstellungen in der Informatik. In: M. Weigend, M. Thomas, F. Otto (Hrsg.): Informatik mit Kopf, Herz und Hand – Praxisbeiträge zur INFOS 2011, 14. GI-Fachtagung Informatik und Schule, 12. bis 16. September 2011 in Münster. Münster: ZfL-Verlag, 2011, S. 61–70.

Schulversuch „Genetischer Informatikunterricht" – Informationen und Materialien:
http://myweb.rz.uni-augsburg.de/~oldenbre/geniu.html

Schuster, J.: Ein genetischer Zugang zum Programmieren mit CGI-Skripten in Python. In: M. Thomas (Hrsg.): Informatik in Bildung und Beruf. INFOS 2011 – 14. GI-Fachtagung Informatik und Schule, 12.–15. September 2011 in Münster. Reihe „GI-Edition Lecture Notes in Informatics", Band P-189. Bonn: Köllen Verlag, 2011, S. 227–236.

Wolfram Demonstrations Project – 3D Flying Pipe-Laying Turtle:
http://demonstrations.wolfram.com/3DFlyingPipeLayingTurtle/

Alle Internetquellen wurden zuletzt am 15. September 2015 geprüft und können auch aus dem Service-Bereich des LOG IN Verlags (http://www.log-in-verlag.de/) heruntergeladen werden.