Objektorientierte Methoden mit Objekt-Turtles

1. Ein Plädoyer für objektorientierte Methoden im Informatikunterricht

Objektorientierte Methoden spielen beim modernen Softwareentwurf eine wichtige Rolle. Im Gegensatz zu einer weit verbreiteten Meinung eignen sie sich nicht nur zur Behandlung komplexer Probleme, sondern entsprechen einer grundsätzlich neuen Art der Problemlösung - man spricht mit Recht von einem Paradigmenwechsel. Der Informatikunterricht an allgemeinbildenden Schulen war in den letzten 20 Jahren einem ständigen Wandel ausgesetzt und hat bis heute Mühe, sich im Fächerkanon zu etablieren. Oft wird argumentiert, der ständige Wandel zeige, dass in der Informatik, im Gegensatz zu den klassischen Schulfächern, keine fundamentalen Prinzipien und Ideen existieren. Diese Kritik mag teilweise berechtigt sein, wenn man etwa an die rasche Veränderung der Ziele und Inhalte des Informatikunterrichts der vergangenen Jahre denkt, von der hardwarenahen, über die algorithmische zur anwendungsorientierten Informatik. Unbestritten gibt es auch in der Informatik Grundlagen von allgemeiner Bedeutung. So haben das von Neumannsche Prinzip der sequentiellen Maschine, das Prinzip von Böhm und Jacopini über die Programmstrukturen trotz Parallelrechner und logischer Programmiersprachen die Zeit überdauert. Es ist anzunehmen, dass die objektorientierten Methoden (im folgenden *OO-Methoden* genannt) von ebenso grundsätzlicher Bedeutung sein werden. Der Grund für den fundamentalen Charakter dieser neuen Methoden mag auch daran liegen, dass sie weitgehend einer natürlichen Betrachtungsweise entsprechen. So weiss man aus der pädagogischen Forschung dass bereits Kleinkinder ihr Umfeld als eine Objektwelt auffassen, in der die Objekte spezifische Eigenschaften (Zustände und Fähigkeiten, im Sinn der objektorientierten Programmierung Attribute und Methoden) aufweisen.

In letzter Zeit fordern deshalb fortschrittliche Lehrpersonen, Fachwissenschafter und Lehrplangestalter, die Objektorientierung bereits im Anfängerunterricht einzuführen. So schreibt Baumann in *Informatische Bildung in Deuschland*: "Im Gegensatz zu [...], der die (abwegige) Meinung vertritt, Objektorientierung sei für die Schule zu schwierig, sollte dieser Aspekt von Beginn an in den Informatikunterricht einfliessen" [1]. Hier sind vor allem die Didaktiker gefordert, einen praktikablen Weg aufzuzeigen, was in keineswegs eine leichte Aufgabe ist. So genügt es nicht zu fordern, im Informatikunterricht statt einer klassischen Programmiersprache eine objektorientierte (wie Java, Eiffel oder Smalltalk), oder eine mit objektorientierten Erweiterungen (wie C++ oder ObjektPascal) einzusetzen. Für die Einführung von OO-Methoden ist dies nur eine notwendige, aber keineswegs eine hinreichende Bedingung. Welches ist aber die geeignete Unterrichtsmethodik?

Wie wir aus einer langjährigen pädagogischen Forschung wissen, eignet sich gerade in mathematisch-naturwissenschaftlichen Bereich, zu dem ich die Informatik zähle, das exemplarische Prinzip nach Wagenschein [2] hervorragend. Für den Informatikunterricht müssen wir also nach exemplarisch geeigneten Objekten Ausschau halten. Abhängig von der Zielsetzung des Unterrichts gibt es mehrere Kandidaten: Für eine einfach Betrachtung eignen sich beispielsweise Elemente einer graphischen Bildschirmoberfläche, wie Fenster oder Schaltflächen. Im betriebswirtschaftlich orientieren Unterricht können sehr wohl Datenbanken als Objekte aufgefasst werden. Ueber das rein Begriffliche hinaus, eignen sich aber diese Beispiele kaum, da ein praktischer Umgang am Rechner kompliziert ist. Ein hinreichend einfaches und den Schüler und insbesondere auch die Schülerin ansprechendes Objekt ist die Turtle. Wir erinnern uns dabei an die Hoffnungen, ja Euphorie, welche sowohl die pädagogische Fachwissenschaft, wie die Schulpraktiker in den Achtzigerjahren auf Grund der Ideen von Papert [3] erfasst hat. In letzter Zeit ist es aber still geworden um die Logo-Turtle. Dies ist nicht auf einen Misserfolg der von Papert geforderten Unterrichtsmethoden,

beispielsweise das "Learning by Doing" oder das "Konstruieren von Mikrowelten" zurückzuführen, sondern vielmehr auf das Fehlen eines adäquaten Unterrichtsumfeldes in den neuen Schulstrukturen. Mancherorts wurde der Informatikunterricht, insbesondere das Programmieren, stark reduziert oder gänzlich aufgehoben.

Wie unsere Erfahrung zeigt, eignet sich die Turtle für eine Einführung in die OO-Methoden hervorragend. So besitzt die Turtle einerseits Attribute, wie eine *Lage*, eine *Blickrichtung*, eine *Form*, eine *Farbe*, eine *Spurfarbe*, usw. und andererseits Methoden, wie *gehe vorwärts*, *gehe rückwärts*, *drehe links*, *drehe rechts*, usw. Allerdings verfügt die Programmiersprache Logo über keine Mittel, mit diesem Objekt aus der Sicht der OO-Methoden umzugehen: Sie ist typenlos und tut sich bereits mit dem Variablenbegriff schwer. Objekte müssen sich in beliebiger Zahl erzeugen, verwenden und wieder vernichten lassen. Es ist aber keine Implementierung von Logo bekannt, welche im gleichen Grafikfenster mehrere Turtles mit ihren möglichen Ueberdeckungen zulässt.

Trotz unserer ehemals grossen Begeisterung für die Programmiersprache Logo, haben wir uns aus einem anderen Grund entschieden, Logo nicht mit OO-Methoden zu ergänzen und auf eine moderne Umgebung zu portieren. Logo, ein Abkömmling von LISP, wurde schon immer als eine sehr exotische Sprache betrachtet. Lehrpersonen wurden oft nicht nur von Fachinformatikern, sondern auch von Kollegen anderer Fächer, wie der Mathematik und Physik belächelt und mit der Frage konfrontiert, ob man Logo auch tatsächlich für relevante algorithmische Probleme aus der Mathematik oder Physik, etwa für numerische Integrationen oder Simulationen einsetzen könne. Wir mussten zugeben, dass dies "im Prinzip" wohl möglich sei, aber dass Logo dafür doch zu wenig geeignet ist. Wir haben uns daher entschieden, die Objekt-Turtle in einer der bekannten modernen Programmiersprachen zu implementieren. Dabei sollte es aber gemäss unseren eigenen Vorgaben möglich sein, mit ebenso wenig Aufwand an technischem Rüstzeug und Hintergrundwissen ein Programm zu schreiben wie früher mit Logo. Wir haben uns vorgenommen, in Zusammenarbeit mit einer Softwarefirma eine Programmbibliothek (eigentlich eine Klassenbibliothek) zu entwickeln, welche es dem Anfänger ermöglicht, bereits in der ersten Unterrichtsstunde Windows-fähige OO-Programme zu schreiben, ohne dass dafür Kenntnisse über Windows-Programmierung nötig sind. Die Bibliothek haben wir *Champ* genannt, womit angedeutet wird, dass sie eine Unterstützung für zukünftige Informatik-Champions sein soll.

Unsere Vorgaben führten zu zwei wichtigen Entscheiden: In den meisten Einführungen in die OO-Methoden werden zuerst die Objekte mittels Klassendefinitionen konstruiert und im folgenden verwendet. Klassendefinitionen sind aber bekanntlich sowohl syntaktisch wie vom Begrifflichen her kompliziert und dem Anfänger nicht zuzumuten. Aus diesem Grund gehen wir umgekehrt vor: wir *verwenden* vordefinierte Klassen über eine längere Zeit bevor wir Klassen konstruieren oder ableiten. Wir glauben, dass dies der richtige Weg in einem *anwenderorientierten* Informatikunterricht ist, da er auch weitgehend den Tendenzen in der professionellen Informatik entspricht, wo immer mehr wiederverwendbare vordefinierte Klassen eingesetzt werden.

Der zweite Entscheid befasst sich mit der Programmierung unter einer grafischen Benützeroberfläche. Die Schwierigkeiten, die sich dabei stellen, sind selbst für den professionellen Informatiker enorm und die Hilfsmittel entsprechend kompliziert. So ist es unseres Erachtens zu riskant, den Einstieg in die Informatik mit ereignisgesteuerter API-Programmierung oder OO-Sprachen, welche gleich zu Beginn Klassenableitungen erfordern, zu wagen (MFC, OWL, Java, Smalltalk). Als Alternative stehen zwar noch Programmiersysteme zur Verfügung, welche vom *Formularentwurf* ausgehen (Visual Basic,

Delphi). Dieses Vorgehen, wie die dabei zum Einsatz kommenden Sprachversionen, sind aber unseres Erachtens zu wenig fundamental und zu stark produkttypisch.

Die langjährigen guten Erfahrungen bei der Einführung von Programmiersprachen unter einem textorientierten Betriebssystem (beispielsweise Pascal unter DOS) haben uns geradezu herausgefordert. Wir möchten der Lehrkraft eine Programmierumgebung in die Hand geben, damit sie unter Windows ebenso einfach programmieren kann wie früher unter DOS (oder auf dem Commodore). Insbesondere soll es möglich sein, rein sequentielle Programme ohne Ereignissteuerung zu schreiben. Die komplizierteren Elemente einer GUI-Oberfläche, welche Ereignisse auslösen (beispielsweise Dialoge, Menüs, Maus), sollen erst in einer späteren Phase hinzugefügt werden können.

2 Ein Plädoyer für die Turtlegrafik unter C++

Von Anfang an war klar, dass der volle Befehlssatz der Logo-Turtle derart implementiert werden muss, dass dieser nicht als ein künstlich aufgepfropfter Zusatz zur Programmiersprache, sondern vielmehr als Bestandteil der Sprache selbst empfunden wird. Eine weitere Forderung bestand darin, eine Programmiersprache einzusetzen, die eine breite Akzeptanz in der professionellen Welt besitzt. Insbesondere soll das Erlernte auch im weiterführenden Informatikunterricht sowie in anderen Fächern, etwa der Mathematik und den Naturwissenschaften direkt einsetzbar sein. Obschon wir aus Tradition eher aus der Welt von LISP/Logo sowie Basic/Pascal/Modula hervorgingen und die vielen didaktischen Vorwürfe an die Programmiersprache C kannten, ja teilweise selbst erhoben, haben wir uns für C++ entschieden. Diese Sprache ist zwar aus C hervorgegangen, hat aber die meisten Schwächen von C überwunden. Zwar kann man in C++ immer noch in alter schlechter C-Manier programmieren, falls man alle Warnungen des Compilers in den Wind schlägt. Die Sprache ist aber so reich, dass sich mit ihr praktisch alle modernen Gesichtspunkte der Programmierung exemplarisch aufzeigen lassen. Es ist klar, dass C++ durch diese Universalität eine umfangreiche, komplizierte Sprache geworden ist. Mit didaktischem Geschick kann aber jeweils gerade nur derjenige Teil ins Blickfeld gerückt werden, der für die Erläuterung eines Unterrichtsgegenstandes notwendig ist. Ist man beispielsweise der Meinung, dass im Sinne von Java der Zeigerbegriff entbehrlich, ja gefährlich ist, so führt man ihn eben gar nicht ein. Statt durch eine Abmagerung der Sprache Einschränkungen zu erzwingen, kann der Programmierer diejenigen Sprachkonstrukte verwenden, die optimal der Problemstellung angepasst sind. C++ erfüllt aber auch wachsende Ansprüche und wird an vielen höheren Lehranstalten und im kommerziellen Bereich intensiv eingesetzt, kurz "C++ ist eine Sprache, mit der man wachsen kann" [4].

Die Implementierung der Turtlegrafik auf die geforderte harmonische Art ist in C++ besonders einfach möglich, da C++ als erweiterbare Sprache konzipiert ist und eine einfache Include-Anweisung genügt, um eine Vielzahl von Spracherweiterungen (Datenstrukuren und Algorithmen) zugänglich zu machen.

Im folgenden wird gezeigt, mit welchem methodischen Vorgehen bereits in der ersten Unterrichtsstunde ein objektorientiertes Programm entwickelt werden kann, das der Anfänger in allen Einzelheiten versteht. Dabei zeigt sich, dass die Syntax von C++ selbst beim Anfänger eine gute Akzeptanz besitzt, falls man bereits früh auf eine einheitliche und saubere Bildschirmgestaltung des Sourceprogramms Wert legt. Gerade der Anfänger hat sich aber an den *Leitsatz* zu halten, dass Programme nicht nur richtig laufen, sondern auch schön geschrieben sein müssen.

In der Unterrichtsstunde werden vorerst einige Grundlagen der OO-Methoden besprochen, insbesondere

Ein Objekt besitzt:
- eine Identität (einen Typ, d.h. eine Klassenzuhörigkeit und einen Namen)
- Attribute (d.h. einen inneren Zustand)
- Methoden (d.h. ein mögliches Verhalten)

Für die Turtle heisst dies:
- Klassenzugehörigkeit zur Klasse *Turtle*
- Attribut *Farbe*, Attribut *Position*, usw. (sichtbar auf Bildschirm)
- Farbe setzen: *setColor(color)*, Position setzen *setX(x)*, *setY(y)*, *setPos(x, y)*, Bewegungen ausführen: *forward(steps)*, *back(steps)*, *left(angle)*, *right(angle)*, usw.

(Wie in der OO-Programmierung üblich, können *aus Sicherheitsgründen* von aussen nicht direkt auf die Attribute des Objekts zugegriffen werden, sondern nur über die entsprechenden Methoden.)

Zum Verständnis des ersten Programms sind noch folgende Zusatzkenntnisse nötig:
- das Hauptprogramm ist eine Funktion *gmain()* und gibt keinen Wert zurück (Rückgabetyp void)
- *#define OBJECT_TURTLE* bewirkt, dass *ginit()* ein Grafikfenster initialisiert, dessen Koordinaten der Turtlegrafik angepasst sind
- *getch()* bewirkt ein Warten auf ein Tastaturzeichen
- *gend()* schliesst das Grafikfenster

Im ersten Programm (Listing 1) wird die Turtle mit dem Namen toby *erzeugt*, dann ihr Farbattribut *gesetzt* und ihr schliesslich *befohlen*, sich vorwärts zu bewegen. Die Kompilation, Bindung und Ausführung geschieht in der Entwicklungsumgebung mit einem einzigen Knopfdruck und erzeugt eine Grafik in einem üblichen Windows-Fenster (Bild 1).

```
#define OBJECT_TURTLE
#include <champ.h>

void gmain ()
{
   ginit();
   Turtle toby;
   getch();

   toby.turtleColor( RED );
   getch();

   toby.forward( 150 );
   getch();

   gend();
}
```

Listing 1: Das erste Programm

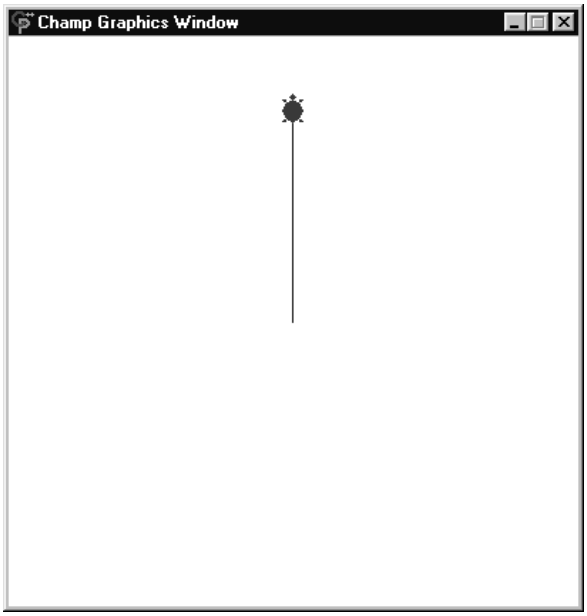

Bild 1: Das Resultat von Listing 1

Im zweiten Programm (Listing 2 und Bild 2) wird die Stärke der OO-Methode offensichtlich: ohne Schwierigkeiten lassen sich zwei Turtles tina und toby erzeugen, die sich gemeinsam treffen. Um den Vorgang genauer beobachten zu können, wird mit speed(100) die Bewegungsgeschwindigkeit der Turtles (rechnerunabhängig) verkleinert.

```
#define OBJECT_TURTLE
#include <champ.h>

void gmain ()
{
   ginit();

   Turtle tina;
   tina.speed( 100 );
   tina.setY( 150 );
   tina.left( 180 );

   Turtle toby;
   toby.turtleColor( RED );
   getch();

   tina.forward( 130 );
   getch();

   gend();
}
```

Listing 2: Das zweite Programm

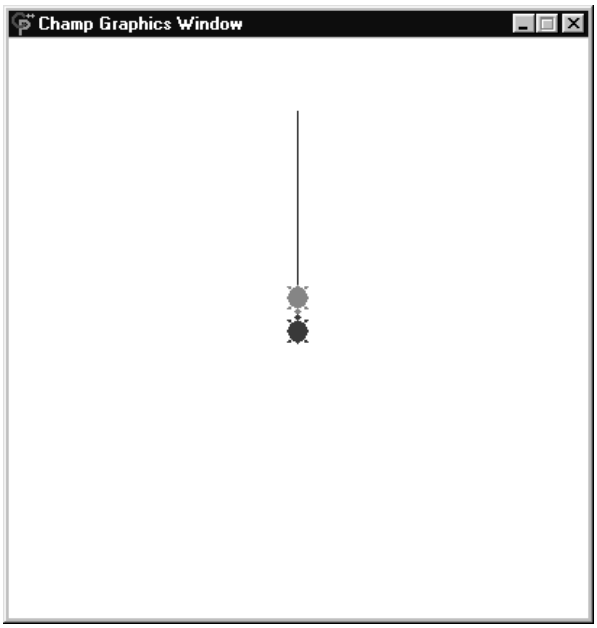

Bild 2: Das Resultat von Listing 2

C++/Champ ist ein professionelles Entwicklungswerkzeug, das sich problemlos in weiterführenden Kursen und zur Herstellung von Lern- und Simulationsprogrammen einsetzen lässt. Es stehen Klassen für Koordinatengrafik in mehreren Fenstern, für Dialoge und Menüs, für Timer, serielle und parallele Schnittstellen, zur Dateibehandlung, usw. zur Verfügung. Programme in Basic oder Pascal lassen sich sehr einfach in die Windows-Umgebung portieren, beispielsweise das im Buch "Bausteine des Chaos" notierte Basic-Programm zur Erzeugung eines Sierpinski-Dreiecks (Listing 3 und Bild 3).

```
DEFINT x, y
FOR y = 0 TO 255
   FOR x = 0 TO 255
      IF (x AND y) = 0 THEN PSET (x+30, y+30)
   NEXT x
NEXT y
END
```

Listing 3: BASIC Programm aus Peitgen et al., *Bausteine des Chaos*

```
#include <champ.h>

void gmain ()
{
   CPWindow w;

   for ( int y = 0; y <= 255; y++ )
      for ( int x = 0; x <= 255; x++ )
        if ( ( x & y ) == 0 )
           w._putPixel( x + 30, y + 30, BLACK );

   getch();
}
```
Listing 4: Portierung von Listing 3 in C++/Champ

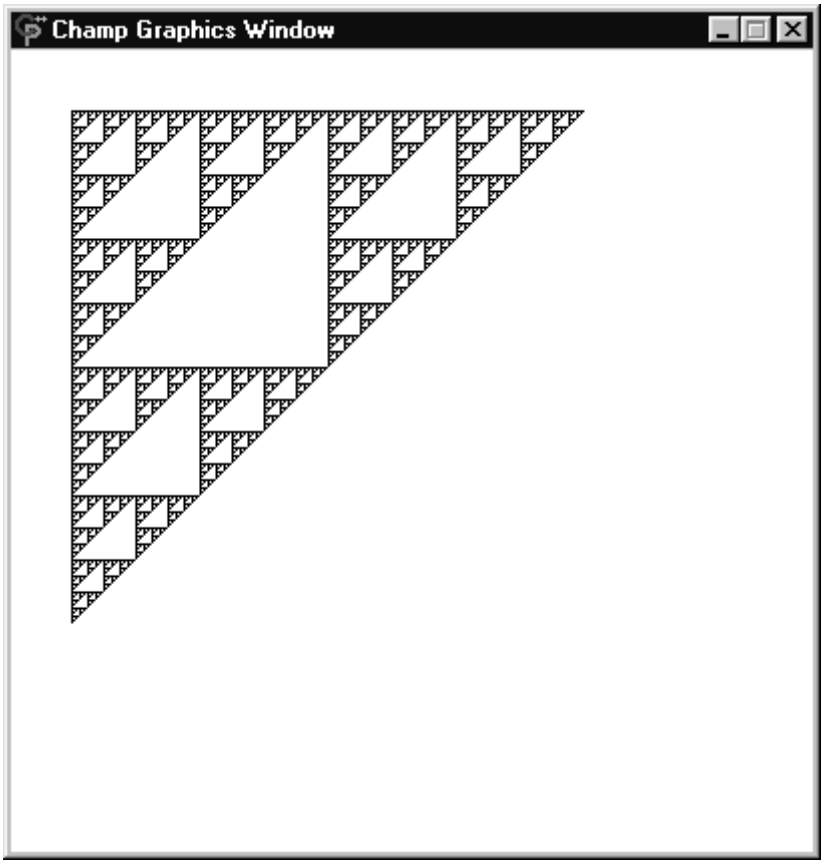

Bild 3: Das Resultat von Listing 4

3 Erfahrungen

Die mehr als zweijährige Erprobung der beschriebenen Unterrichtsmethoden sowohl in der Lehramtsausbildung der Universität Bern wie an mehreren Gymnasien zeigt durchaus ermutigende Resultate. Selbst Skeptiker gegenüber der alten C-Schule, sowie Basic- und Pascal-Freaks konnten davon überzeugt werden, dass das Erlernen von C++/Champ mit einem vernünftigen Zeitaufwand möglich ist, falls man auf der didaktischen Literatur aufbaut und die komplizierten Standardwerke über C++ und Windows-Programmierung nur zum vertieften Verständnis heranzieht. In Zeiten des allgemeinen Abbaus der Programmierung an

den Schulen ist es kaum angebracht, einen Streit über Programmiersprachen zu führen (etwa zwischen C++ und Java), sondern die Wahl hat pragmatisch zu erfolgen, im Sinn: *Wie kriege ich in der kürzesten Zeit und mit dem kleinsten Aufwand die besten Voraussetzungen für einen modernen, d.h. objekt- und grafikorientierten Informatikunterricht an allgemeinbildenden Schulen?*

Anschrift des Verfassers:
Prof. Dr. Aegidius Plüss
Uni Bern, Fachdidaktik Informatik
Bremgartenstrasse 133
CH-3012 Bern, Schweiz
E-Mail: pluess@sis.unibe.ch

Lehrgänge und didaktische Berichte sind zum Download erhältlich bei:
www.clab.unibe.ch/champ

Die freie Evalutions-Version von Champ ist zu beziehen bei:
www.salvisberg.com

[1]Baumann R., "Fundamentale Ideen der Informatik - gibt es das?" in Koerber, Peters (Hrg.), *Informatische Bildung in Deutschland*, Log In Verlag 1998
[2]Wagenschein M., *Verstehen lernen*, Klett 1970
[3]Papert S., *Mindstorms - Kinder, Computer und Neues Lernen*, Birkhäuser 1982
[4]Stroustrup B., *Die C++ Programmiersprache*, Addison-Wesley-Longman 1998