informatica didactica 7 (2006) - http://www.informatica-didactica.de

Puck - eine visuelle Programmiersprache für die Schule

Lutz Kohl

Abteilung für Didaktik
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, 07743 Jena
E-Mail: Lutz.Kohl@uni-jena.de
WWW: http://www.uni-jena.de/puck.html

Zusammenfassung[1]**:** Im Rahmen der Diplomarbeit des Autors wurde an der Friedrich-Schiller-Universität Jena eine visuelle Programmiersprache für den Einsatz in Schulen entwickelt. In einer vorangegangenen Studienarbeit wurden hierfür theoretische Grundlagen gelegt und bestehende Systeme gesichtet. Durch Interviews mit Lehrern sind Anforderungen an eine solche Sprache zusammengetragen worden. Ziel war es, ein System zu entwickeln, das Anfängern einen leichten Einstieg in das Programmieren ermöglicht und Syntaxfehler vermeidet. Das Ergebnis der Arbeiten – das Puck-System – wird in diesem Beitrag vorgestellt. Außerdem wird eine erste Erprobung in der Schule beschrieben.

Abstract: In the author's diploma thesis a visual programming language was developed for learning to program at school. The basics of visual programming and existing systems had been presented in a previous work at Friedrich Schiller University Jena. The requirements for the system to be developed were found via interviews with computer science teachers. The goal was to create a visual programming environment which is easy to handle and prevents students from making syntax-errors. The result of the work - the Puck-System - is introduced in the article. Furthermore a first evaluation of Puck at school is described.

[1] Dieses Dokument besteht zu großen Teilen aus der INFOS-05 Publikation [Ko05] des Autors. Einige Textstellen und Screenshots wurden aktualisiert, außerdem wird auf weitererführendes Material in der HTML-Fassung des Artikels hingewiesen.

1 Einleitung

Stellung und Bedeutung des Programmierens im Informatikunterricht an allgemeinbildenden Schulen werden innerhalb der Fachdidaktik Informatik immer wieder neu diskutiert. Eines scheint aber klar zu sein: Das Programmieren zu lernen ist nicht einfach. Gerade durch die vielfältigen Syntaxregeln erscheinen dem Anfänger die Hürden oft unüberwindbar. Durch visuelle Programmierung[2] ist es mit Hilfe von Bausteinen, die aufgrund ihrer Form nur in richtiger Art und Weise miteinander verbunden werden können, möglich, Syntaxfehler weitestgehend zu vermeiden (vgl. [Ke03]).

Somit wird der anfängliche „Schock“ bei Programmierneulingen (vgl. [Ro03]), der dadurch entsteht, dass in kurzer Zeit verschiedene schwierige Aufgaben zu bewältigen sind[3], entschärft, indem das Einhalten der Syntax von einem Programm übernommen wird.

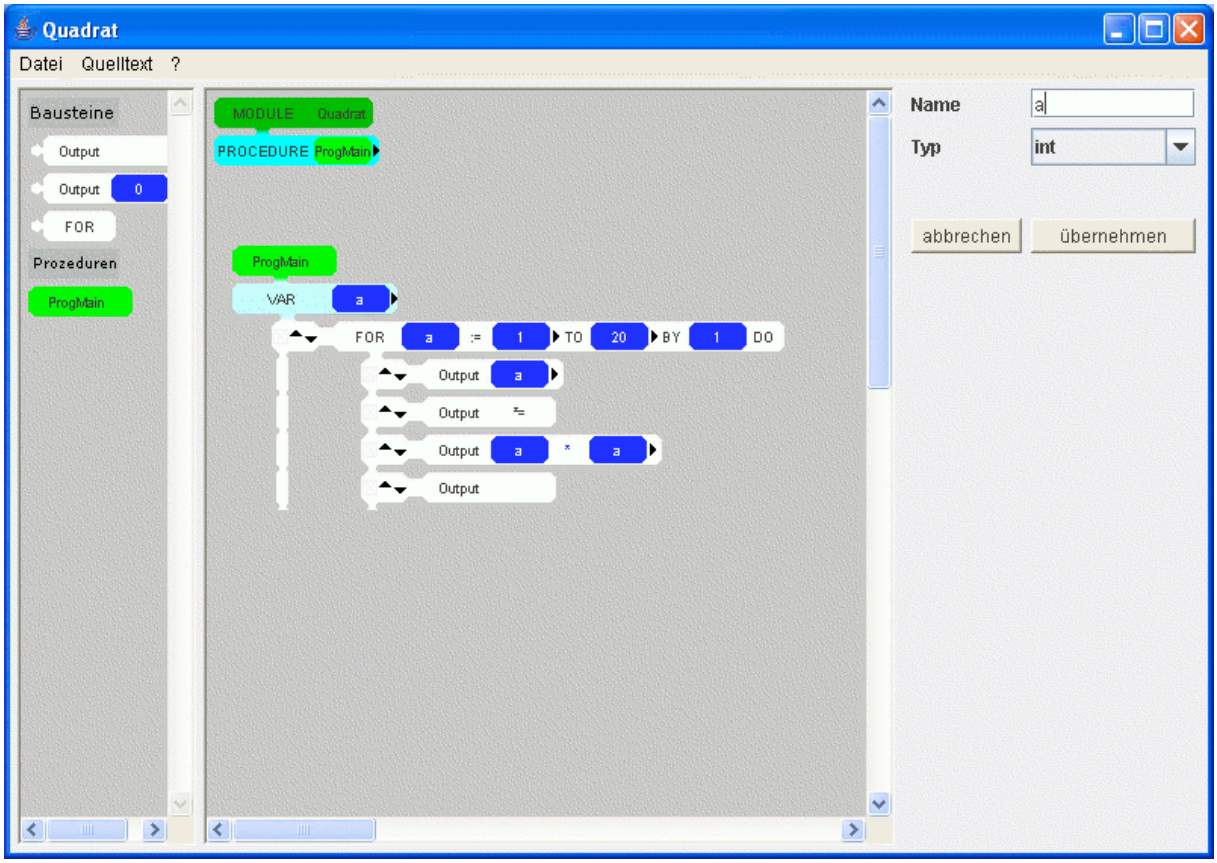


Abbildung 1: Ein mit Puck-Programm zur Berechnung der ersten 20 Quadratzahlen

[2] Mit visuellen Programmiersprachen sind im Folgenden solche Sprachen gemeint, bei denen ein Programm vollständig mit Hilfe von visuellen Elementen entwickelt wird. Auf Systeme, die eine einfache visuelle Erstellung der Oberfläche ermöglichen, bei denen die Funktionalität aber textuell entwickelt werden muss, wird in diesem Beitrag nicht näher eingegangen.

[3] In [Ro03] sind folgende schwierige Aufgaben für Programmieranfänger angegeben: generelle Orientierung, wofür Programme da sind und was man mit ihnen machen kann; ein imaginärer Maschinenbegriff: Ein Computermodell, das mit der Programmausführung verbunden ist; Notation des Programms, das heißt Syntax und Semantik einer Programmiersprache; Schemas und Pläne zum Lösen von Aufgaben; praktische Fähigkeiten wie Planen, Entwickeln, Testen, Fehlersuchen.

Der größte Nachteil visueller Programmiersprachen, komplexe Inhalte vergleichsweise unübersichtlich darzustellen, tritt bei von Anfängern konstruierten Programmen nicht auf, da diese nur einen geringen Umfang haben[4]. In einer vom Autor an der Friedrich-Schiller-Universität Jena verfassten Diplomarbeit wurde nach einer vorherigen Erhebung von Anforderungen eine visuelle Programmiersprache mit einem dazugehörigen Programmiersystem für den Einsatz in Schulen entwickelt. Mit Hilfe dieses Systems können Anfänger ein Programm per Drag and Drop konstruieren. Im Verlauf des Lernprozesses werden den Schülerinnen und Schülern – je nach Konfiguration durch den Lehrer – nur bestimmte Bausteine zur Verfügung gestellt. Dadurch bekommt ein Anfänger die Möglichkeit, das Programmieren schrittweise, also Anweisung für Anweisung, zu erlernen. Die visuell erstellten Programme können stets direkt ausgeführt werden.

2 Entwicklung von Anforderungen

Ein Ziel der Studienarbeit des Autors war es, Anforderungen an eine visuelle Programmiersprache, die an Schulen eingesetzt werden kann, zu erheben (vgl. [Ko04a]). Zunächst wurden verschiedene Systeme analysiert. Lehrer stellen neben Schülerinnen und Schülern die Zielgruppe für ein visuelles Programmiersystem dar und bestimmen auf Grundlage des Lehrplans über den Einsatz eines Werkzeuges im Unterricht. Außerdem besitzen sie wertvolle Erfahrungen, die für die Entwicklung einer Programmiersprache für den Einsatz in Schulen nützlich sind. Deshalb erwies es sich als sinnvoll, die Meinungen von Lehrern zu erheben. In einem ersten leitfadenorientierten Interview wurden diese nach ihrer Unterrichtspraxis und ihren Forderungen an eine visuelle Programmiersprache für den Einsatz in Schulen befragt. Da den Interviewpartnern meist keine visuellen Programmiersysteme bekannt waren, wurden ihnen die Beispiele LabView [LabV], LogoBlocks [LoB] und Agentsheets [Age] vorgestellt. Dadurch wurde abgesichert, dass sie eine Einführung in die Möglichkeiten der visuellen Programmierung erhielten. Die Ergebnisse der Befragung wurden zu einem Anforderungsdokument zusammengefasst. Dieses wurde den Lehrern in einem zweiten Interview vorgestellt und konkretere Fragen wurden ausführlicher diskutiert.

Die Anforderungen der Lehrer waren auf die Umsetzung des Lehrplans ausgerichtet. So wurden die Erwartungen formuliert, dass sich die visuelle Programmiersprache am imperativen Programmierparadigma orientieren und auf das Arbeiten mit textuellen Sprachen vorbereiten soll. Auf die im Unterricht zu vermittelnden Konzepte wie Variablen, Anweisungen, Rekursion sowie Globalität und Lokalität wurde viel Wert gelegt. Außerdem wurde gefordert, dass zu einem visuell erstellten Programm der Oberon-2- bzw. Pascal-Quelltext generiert werden kann. Des Weiteren wurden eine Hilfe und eine Aufgabensammlung gewünscht. Die visuelle Programmiersprache sollte nach Möglichkeit den Schulen im Internet kostenlos zur Verfügung gestellt werden. Für die Entwicklung von schulisch geeigneter Software wird in [Fo04] außerdem gefordert, besonderen Wert auf die sorgfältige Ausgestaltung der „Schüler-Computer-Schnittstelle“ zu legen. Im Rahmen der Diplomarbeit des Autors wurde ein diesen Anforderungen entsprechendes System entwickelt.

[4] Vor- und Nachteile visueller Programmiersprachen sind in [Sc98] und [Ko04a] ausführlicher zusammengestellt.

3 Fragen zu Puck

Was ist Puck?

Puck ist eine visuelle Programmiersprache mit dem zugehörigen visuellen Programmiersystem, das es dem Benutzer ermöglicht, mit Hilfe von Bausteinen einfache Programme zu erstellen. Somit können Anfänger die Verwendung von Variablen, Anweisungen und Prozeduren erlernen, ohne gleich mit der komplexen Syntax einer Programmiersprache konfrontiert zu werden. Nach dem Erlernen der Programmierung mit Puck kann auf textuelle Programmierung umgestiegen werden. Dies wird dadurch erleichtert, dass sich der Nutzer zum aktuellen visuellen Programm auch stets den Quelltext ansehen kann.

Was kann Puck?

In der visuellen Programmiersprache Puck können typische Programme des Anfangsunterrichtes entwickelt werden. Mit zwölf Anweisungsbausteinen (Zuweisung, Input, Output, Textausgabe, IF THEN, WHILE DO, REPEAT UNTIL, FOR, Prozeduraufruf, Dot, Line und Color) ist es möglich, die Grundstrukturen imperativer Programmierung kennen zu lernen. Innerhalb der Anweisungen können nur vorher deklarierte Variablen verwendet werden. Als Datentypen stehen Integer und Boolean zur Verfügung[5]. Es ist möglich, Prozeduren zu erstellen, denen eine beliebige Anzahl an Wert- und Referenzparametern übergeben werden kann. Somit können auch rekursive Programme, zum Beispiel zum Lösen des Problems der Türme von Hanoi oder zum Berechnen von Fibonacci-Zahlen implementiert werden. Mit Dot, Line und Color wurden drei Anweisungsbausteine in das Puck-System aufgenommen, die es ermöglichen, motivierende Fadengrafiken zu erstellen. Es wurde bewusst auf weitere Datentypen, komplexe Anweisungen und die Verwendung von Funktionen verzichtet, um eine Eignung der visuellen Programmiersprache für den Anfangsunterricht sicherzustellen.

Durch Puck wird gewährleistet, dass ein Programm zu jeder Zeit der Entwicklung ausführbar ist. Somit kann es nach jedem Schritt getestet werden. Schülerinnen und Schüler haben so die Möglichkeit während des Programmierens stets Feedback zu erhalten. Außerdem kann der Benutzer zu jedem Programm den Quelltext in der Sprache Oberon-2 generieren und anzeigen lassen[6]. Weiterhin ist es möglich eine ausführbare JAR-Datei zu erzeugen, die ohne das Puck-System gestartet werden kann.

Weshalb Puck?

„Herr Lehrer, ich hab da so eine komische Fehlermeldung.“ „Du hast dort ein Semikolon vergessen.“ „Wenn du eine Variable verwendest, musst du sie vorher deklarieren.“ „Du hast dort das Schlüsselwort ‚WHILE’ falsch geschrieben.“ „In deinem Programm ist ein ‚END’ zu wenig.“

---

[5] Die verwendeten Datentypen reichen für eine Einführung in die Programmierung aus. In [Fo02] werden die Grundbegriffe imperativer Programmierung mit der Programmiersprache Python in einem ersten Kapitel auch unter ausschließlicher Verwendung der Datentypen Integer und Boolean erklärt.

[6] Die Quelltextgenerierung wurde exemplarisch für die Programmiersprache Oberon-2 implementiert, da diese an vielen Thüringer Schulen eingesetzt wird. Der Oberon-2-Quelltext kann mit der Programmierumgebung Pow! geöffnet, kompiliert und ausgeführt werden. Es ist möglich, das System so zu modifizieren, dass textueller Code für eine andere Programmiersprache erzeugt wird.

Wer einen dieser Sätze schon einmal im Unterricht gehört oder gesagt hat, wird die Vorteile von Puck zu schätzen wissen. Nun mag der Leser entgegnen, dass die dargestellten Probleme beim Übergang zu einer textuellen Sprache trotzdem auftreten können. In der Tat sind dazu weitere Untersuchungen erforderlich. Der Autor ging bei seiner Arbeit aber davon aus, dass es einfacher ist, eine abstrakte textuelle Programmiersprache zu erlernen, wenn die verwendeten Konstrukte bereits bekannt sind. Außerdem können die Schülerinnen und Schüler schon während der Arbeit mit Puck durch den generierten Quelltext auf die richtigen Schreibweisen vorbereitet werden.

Des Weiteren erscheint der Zugang zur Informatik mit Hilfe von Bausteinen, die zu einem Programm zusammengefügt werden, für Schülerinnen und Schüler motivierender als das Erstellen eines Textes, der einer großen Menge von syntaktischen Regeln entsprechen muss. Puck ist ein Open-Source-Projekt und kann so problemlos erweitert werden. Es besteht durchaus die Möglichkeit, dass besonders interessierte Schülerinnen und Schüler im Rahmen von Projektarbeiten zum Beispiel zusätzliche Bausteine entwickeln[7].

4 Ein Beispiel

Ein rechteckiges Grundstück soll mit quadratischen Steinplatten gleicher Größe lückenlos gepflastert werden. Sowohl die gesamte Fläche als auch die Steinplatten haben ganzzahlige Seitenlängen. Ermitteln Sie die Steinplattenmaße so, dass für das Pflastern der Fläche möglichst große Platten verwendet werden können.

Bei genauerer Betrachtung diese Aufgabe stellt sich heraus, dass der größte gemeinsame Teiler der Seitenlängen des Grundstücks gesucht ist. Dieser kann geometrisch ermittelt werden: Von einem Rechteck wird stets ein Quadrat mit den Seitenlängen der kürzeren Seite des Rechteckes abgezogen. Anschließend wird nur das entstehende Rest-Rechteck betrachtet und mit diesem wird wieder genau so verfahren. Gibt es kein Rest-Rechteck, so hat das als letztes abgezogene Quadrat das gesuchte Plattenmaß (vgl. Abbildung 3).

Das in Abbildung 2 dargestellte Programm „RechteckPflastern“ zeigt eine mögliche Lösung der Aufgabe mit einer Darstellung des beschriebenen geometrischen Prozesses.

[7] Puck ist unter der URL http://www.uni-jena.de/puck.html verfügbar. Das Programm ist in Java implementiert. In dem ausführbaren Archiv SetuPuck.jar sind ca. 100 Klassen mit den zugehörigen Quelltexten enthalten. Das Entwickeln eines Bausteines für das Puck-System ist in [Ko04b] exemplarisch an einem Beispiel dargestellt.

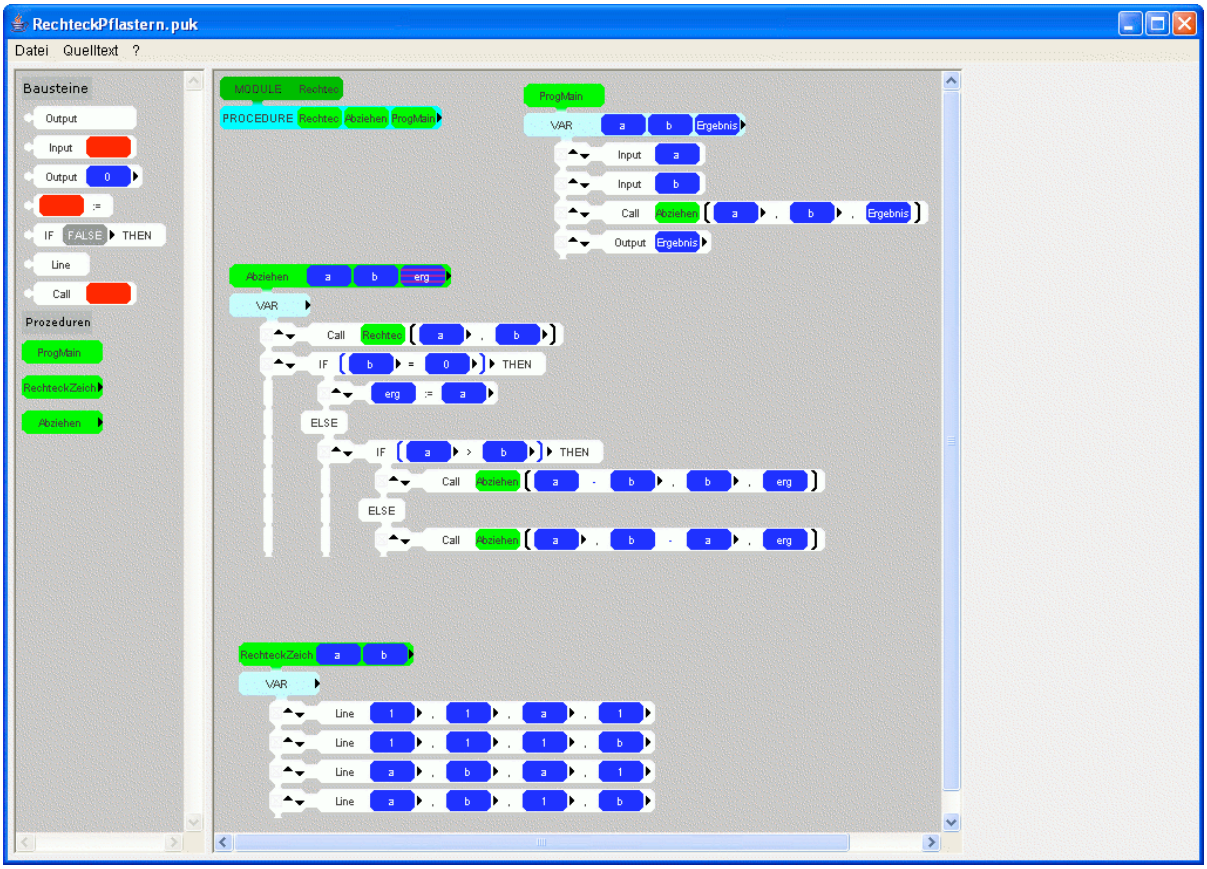


Abbildung 2: Das Puck-Programm zum Pflastern eines Rechteckes

Das entwickelte visuelle Programm kann über die Menüleiste (*Quelltext - Programm ausführen*) direkt ausgeführt werden. Die Ausgaben des Programms bei den Eingabewerten 308 und 66 sind in Abbildung 3 dargestellt.

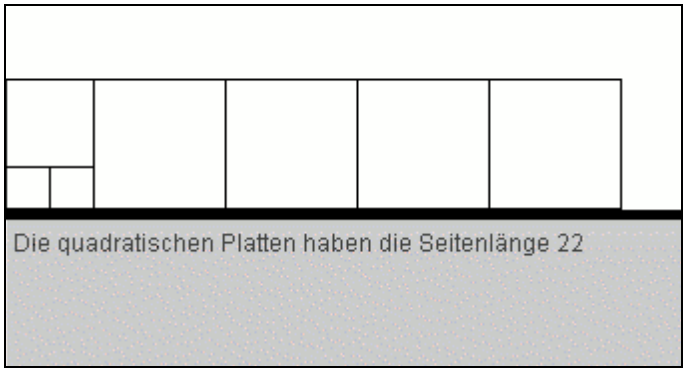


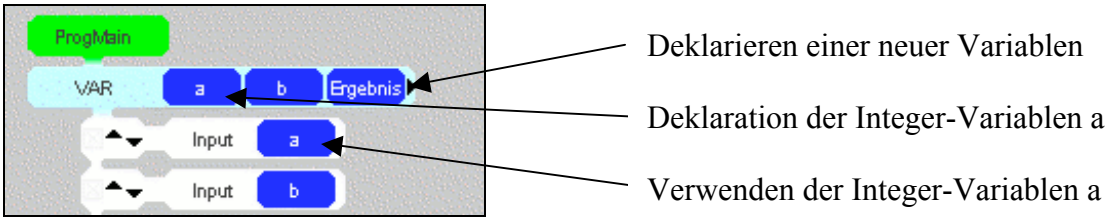
Abbildung 3: Ausschnitt aus der Ausgabe des Programms RechteckPflastern
oben: der geometrische Lösungsprozess; unten: Ausgabe des Ergebnisses

An diesem Beispiel sollen nun einige Eigenschaften des Puck-Systems erläutert werden.

Bausteine

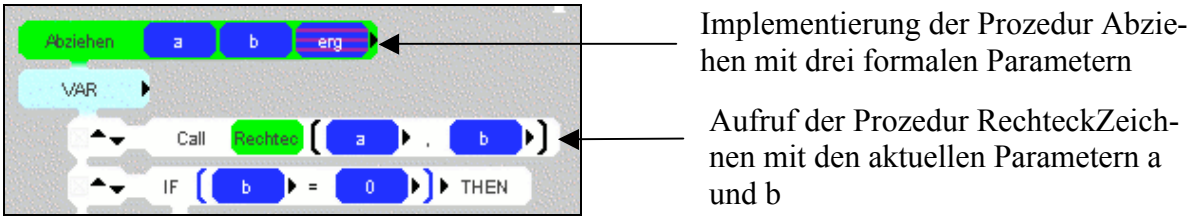
Für die Lösung der oben genannten Aufgabe wurden nur die Bausteine Input, Output, Zuweisung, IF THEN, Prozeduraufruf und Line benötigt. Alle anderen Bausteine wurden im Programm unter dem Punkt Optionen ausgeblendet[8].

Variablendeklaration



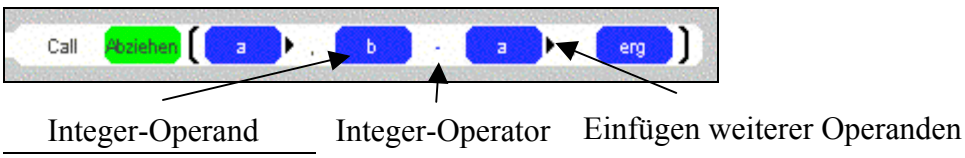
Variablen werden deklariert, indem das kleine schwarze Dreieck rechts neben dem Schlüsselwort VAR angeklickt wird. Sie werden fortlaufend mit kleinen Buchstaben des Alphabetes benannt. Name und Datentyp können in der Attributtabelle, die sich beim Anklicken der Bausteine jeweils auf der rechten Seite des Bildschirms befindet, verändert werden. Es wird nach jeder Veränderung überprüft, ob ein Variablenname im gegebenen Kontext bereits vergeben wurde bzw. ob er den Namenskonventionen entspricht. Variablen vom Typ Integer sind dunkelblau, Variablen vom Typ Boolean sind grau dargestellt.

Prozeduraufruf



Bei einem Prozeduraufruf müssen die aktuellen mit den formalen Parametern in Anzahl, Typ und Reihenfolge übereinstimmen. Um dies zu garantieren, wurde im Puck-System durchgängig mit dem Observer-Entwurfsmuster (vgl. [Ga97]) gearbeitet. Wenn der Benutzer die Signatur einer Prozedur verändert, indem er zum Beispiel einen weiteren Wertparameter vom Typ Integer hinzunimmt, so wird vom System in jeder Anweisung, in der diese Prozedur aufgerufen wird, ein Integer-Ausdruck eingefügt, der der Prozedur als aktueller Parameter übergeben wird. Auch das Verändern des Namens einer Prozedur oder einer Variablen wirkt sich nicht nur auf die Deklaration, sondern auch auf jede Verwendung derselben im gesamten Programm aus. Referenzparameter werden in Puck durch eine Schraffierung gekennzeichnet.

Erstellen von Ausdrücken

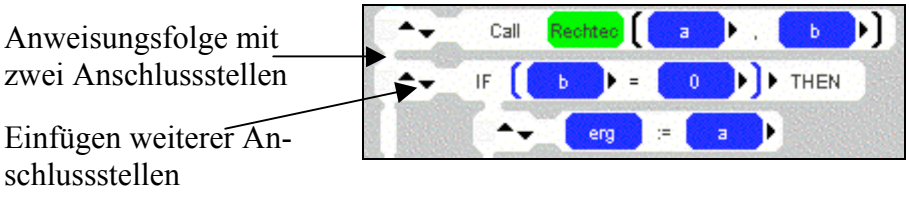


[8] Der Lehrer hat zusätzlich zu dem Menüpunkt „Optionen“ die Möglichkeit, mit einem Setup-Programm das Puck-System jede Unterrichtsstunde neu einzustellen und für die Schülerinnen und Schüler so die zu verwendenden Bausteine festzulegen. Außerdem kann der Menüpunkt „Optionen“ für die Schülerinnen und Schüler unzugänglich geschaltet werden, so dass diese nichts an der aktuellen Einstellung des Systems verändern können.

Ausdrücke werden in verschiedenen Anweisungen verwendet und können innerhalb des Puck-Systems immer in der gleichen Art und Weise modifiziert werden. Ein Ausdruck innerhalb eines Anweisungsbausteins besteht zunächst aus einem Wert[9], der in der Attributtabelle verändert oder nach einem Rechtsklick durch eine im aktuellen Kontext gültige Variable des richtigen Typs ersetzt werden kann[10]. Sollen zu einem Ausdruck noch weitere Operatoren und Operanden hinzugefügt werden, so kann dies durch einen Linksklick auf das kleine Dreieck an der rechten Seite des Ausdrucks geschehen. Daraufhin werden immer ein Operator und ein Operand in den Ausdruck eingefügt. Die je nach Datentyp möglichen Operatoren können über ein Kontextmenü ausgewählt werden. Nach jeder Aktion entsteht stets ein korrekter Ausdruck des jeweiligen Typs.

Klammern werden paarweise um einen Operanden gesetzt und können anschließend einzeln verschoben werden, wenn ein Verschieben in die jeweilige Richtung möglich ist. Öffnende und schließende Klammern werden stets gemeinsam gelöscht.

BEGIN-END-Strukturen



Die Grenzen eines Blockes werden in textuellen Programmiersprachen meist durch geschweifte Klammern, durch BEGIN-END-Strukturen oder durch Einrückung gekennzeichnet. In Puck wird ein Block durch eine Anweisungsfolge charakterisiert. Diese besteht anfänglich aus einer einzelnen Anschlussstelle für einen Anweisungs-Baustein. Klickt der Benutzer eines der beiden Dreiecke der Anschlussstelle an, so entsteht oberhalb bzw. unterhalb eine weitere Anschlussstelle, die wieder ebenso erweitert werden kann. Wenn zum Beispiel ein IF-THEN-Baustein an eine Anschlussstelle angehängt wird, so beinhaltet dieser zwei nach rechts versetzte Anweisungsfolgen, die den Block des THEN- und des ELSE-Zweigs beschreiben. Um Überschneidungen zu verhindern, werden im übergeordneten Block automatisch leere Anschlussstellen eingefügt.

Weitere Beispiele

Im Anhang der HTML-Fassung dieses Artikels ist eine Einführung in die Puck-Programmierung mit fünf Beispielen enthalten[11]. In dieser Einführung werden die Installation und die Bedienung von Puck ausführlicher erläutert. Außerdem werden Variablen, das EVA-Prinzip[12], verschiedene Kontrollstrukturen sowie Prozeduren mit Parametern erklärt.

---

[9] Der Wert ist bei Integer 0 und bei Boolean je nach Anweisung TRUE oder FALSE. Boolean-Ausdrücke in Schleifen sind so initialisiert, dass eine Schleife höchstens einmal durchlaufen wird.
[10] In einen Boolean-Ausdruck kann der Vergleich zweier Integer-Ausdrücke als Operand eingefügt werden. Der Vergleichsoperator kann über ein Kontextmenü ausgewählt werden und die Integer-Ausdrücke können verändert werden.
[11] Vgl. [Ko06].
[12] Eingabe→Verarbeitung→Ausgabe

5 Eine erste Erprobung

Das Puck-System wurde im ersten Halbjahr des Schuljahres 2004/05 am Adolf-Reichwein-Gymnasium Jena in zwei Grundkursen Informatik der 11. Klasse eingesetzt. Der Lehrer wurde in die Bedienung des Systems eingewiesen, gestaltete seinen Unterricht aber selbstständig. Am Ende des Halbjahres wurde die Verwendung des Systems mit Hilfe eines schriftlichen Erfahrungsberichtes des Lehrers, eines Interviews und der Erkenntnisse aus Unterrichtshospitationen[13] ausgewertet. Der Unterricht untergliederte sich in drei Phasen. In der ersten Phase arbeiteten die Schülerinnen und Schüler nur mit Puck. Der Übergang zur textuellen Programmierung mit dem Pow!-System [Pow] wurde in einer zweiten Phase vollzogen. In der dritten Phase arbeiteten die Schülerinnen und Schüler ausschließlich mit textuellen Oberon-2-Programmen.

Zum Einstieg in das Programmieren wurden Punkte, Linien und Fadengrafiken gezeichnet. Im Verlauf der ersten Phase wurden den Schülerinnen und Schülern die benötigten Bausteine schrittweise zur Verfügung gestellt. Dies wurde vom Lehrer als wichtiges methodisches Mittel für den Anfangsunterricht erkannt, denn die Schülerinnen und Schüler wurden dadurch nicht mit einer Vielzahl von Befehlen konfrontiert, sondern konnten, je nach Problem, nur mit den vorgegebenen Anweisungen arbeiten. Am Anfang einer Doppelstunde waren die Computer gesperrt. Das jeweils zu bearbeitende Problem wurde im Klassenverband analysiert, in Teilprobleme zerlegt und mögliche Schwierigkeiten, wie zum Beispiel der Datenfluss zwischen den Programmteilen, wurden besprochen. Struktogramme wurden im Unterrichtsgespräch an der Tafel erstellt.

Anschließend entwickelten die Schülerinnen und Schüler in Einzelarbeit das Programm im Puck-System. Durch die ausführliche Vorarbeit entstanden nur wenige semantische Fehler. Durch die Verwendung der visuellen Programmiersprache waren syntaktischen Fehler praktisch ausgeschlossen. Somit hatten nahezu alle Schülerinnen und Schüler jeweils am Ende einer Doppelstunde ein funktionstüchtiges Programm.

Nach Meinung des Lehrers konnten die Schülerinnen und Schüler in der ersten Phase selbstständiger als bisher arbeiten und auch schneller mit neuen Befehlen umgehen, da die Syntax nicht im Mittelpunkt stand. Außerdem wurde mehr Kreativität und Experimentierfreude verzeichnet.

Wann genau der Übergang vom Erstellen der Programme mit Puck hin zum direkten Entwickeln des Oberon-2-Quelltextes stattfindet, wurde den Schülerinnen und Schülern in der zweiten Phase freigestellt. Dadurch hatte der Lehrer mehr Zeit für die jeweiligen Umsteiger. Während einige Schülerinnen und Schüler noch lange mit Puck arbeiteten, hatten sich andere schnell an die textuelle Programmierung gewöhnt. Das Verändern des von Puck generierten Quelltextes war für viele Schülerinnen und Schüler ein wichtiger Zwischenschritt. Der Lehrer empfand das Kompilieren der Programme im Pow!-System als wichtig, da sich die Schülerinnen und Schüler so von Anfang an mit der Oberfläche der Oberon-2-Entwicklungsumgebung vertraut machen konnten[14].

[13] Der Autor hatte die Möglichkeit, in vier Doppelstunden, in denen das Puck-System verwendet wurde, zu hospitieren.
[14] Während der Erprobung war es noch nicht möglich Puck-Programme direkt auszuführen. Die Schülerinnen und Schüler speicherten den generierten Oberon-Quelltext in eine Datei. Diese Datei wurde anschließend mit der Programmierumgebung POW! kompiliert und ausgeführt.

Auf den direkten Umgang mit dem Puck-System gab es keine Zensuren. Die Leistungskontrollen bezogen sich in den ersten zwei Phasen ausschließlich auf Problemanalysen und das Erstellen von Struktogrammen. Erst nachdem alle Schülerinnen und Schüler den Wechsel von Puck zu Pow! vollzogen hatten, begann die dritte Phase, in der auch der Quelltext mit in die Leistungsbewertung aufgenommen wurde.

Nach Meinung des Lehrers bietet das Puck-System eine gute Möglichkeit, erste Programmiererfolge zu erlangen und die Komplexität des Anfangsunterrichts zu vermindern. Im Schuljahr 2005/06 wird Puck im Adolf-Reichwein-Gymnasium Jena wieder eingesetzt. An dieser Stelle möchte sich der Autor bei dem unterrichtenden Lehrer Herrn Gert Stamm für die Kooperation bedanken.

6 Resümee

Das in diesem Artikel vorgestellte Puck-System zeigt exemplarisch, wie visuelle Programmiersprachen für den Unterricht entwickelt werden können. Das Einbeziehen der Lehrerschaft in die Anforderungsanalyse hat sich als sinnvoller Schritt herausgestellt. Hierdurch konnten konsequente Zielorientierung sowie Akzeptanz in der Schule sichergestellt werden. Durch das Ausnutzen der Möglichkeiten der Visualisierung können Syntaxfehler zumindest im Anfangsunterricht nahezu vollständig vermieden werden. Obwohl eine erste Erprobung des vorgestellten Systems positiv verlaufen ist, stehen genauere Untersuchungen des Unterrichtsverlaufes noch aus. Als weitere Forschungsfragen stellten sich heraus:

- Wie gestaltet sich ein gesamtes Curriculum bei Einbeziehung visueller Programmiersprachen?
- Erzeugen visuelle Programmiersprachen ein eingeschränktes mentales Modell, das dem sprachlich abstrakten Arbeiten mit textuellen Sprachen entgegen steht?
- Ist es möglich, eine visuelle Sprache zu entwickeln, die so flexibel ist, dass sie den Anforderungen des Informatikunterrichts in der Schule gerecht wird und somit textuelle Sprachen im Unterricht ersetzen kann?

Literaturverzeichnis

[Age] http://www.agentsheets.com/index.html (Stand: Dezember 2006)

[Fo02] Fothe, M.: Problemlösen mit Python. Thüringer Institut für Lehrerfortbildung, Lehrplanentwicklung und Medien (Hrsg.): Reihe Materialien, Heft 72, Bad Berka, 2002. http://www.uni-jena.de/data/unijena_/faculties/minet/casio/Publikationen/python.pdf (Stand: Februar 2006)

[Fo04] Fothe, M.: Unterricht – bald nur noch mit Computer? (Antrittsvorlesung), Jenaer Schriften zur Mathematik und Informatik, Math/Inf/13/04, Jena, 2004. http://www.informatica-didactica.de/InformaticaDidactica/Fothe2006.pdf (Stand: Mai 2006)

[Ga97] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. 11. Auflage, Reading, 1997.

[Ke03] Kelleher, C.; Pausch, R.: Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. Pittsburgh, 2003. http://reports-archive.adm.cs.cmu.edu/anon/2003/CMU-CS-03-137.pdf (Stand: Februar 2006).

[Ko04a] Kohl, L.: Konzepte der visuellen Programmierung und ihrer Einsatzmöglichkeiten an Schulen. (Studienarbeit), 2004. http://www.uni-jena.de/Lutz_Kohl.html (Stand: Februar 2006)

[Ko04b] Kohl, L.: Entwurf und Implementierung einer visuellen Programmiersprache für den Einsatz in Schulen. (Diplomarbeit), 2004.
http://www.uni-jena.de/Lutz_Kohl.html (Stand: Februar 2006)

[Ko05] Kohl, L.: Puck - eine visuelle Programmiersprache für die Schule. In: S. Friedrich (Hrsg.): Unterrichtskonzepte für informatische Bildung, INFOS 2005, 11. GI-Fachtagung Informatik und Schule, 28.-30. September 2005 an der TU Dresden, Gesellschaft für Informatik, 2005, S. 309-318.

[Ko06] Kohl, L.: Mit Puck einfach Programmieren lernen – Eine Einführung mit fünf Beispielen –. Jenaer Schriften zur Mathematik und Informatik, Math/Inf/01/06, Jena, 2006.
http://www.minet.uni-jena.de/preprints/kohl_06/Lutz%20Kohl%20Mit%20Puck%20einfach%20Programmieren%20lernen.pdf
(Stand: Februar 2006)

[LabV] http://www.ni.com/labview/d/ (Stand: Dezember 2006)

[LoB] http://artdecom.mesh.de/projekte/werkzeuge/software/microcomputer-sensorik_familie/cricket/logo-blocks-for-crickets.html (Stand: Dezember 2006)

[Pow] http://www.fim.uni-linz.ac.at/pow/Download.htm (Stand: Dezember 2006)

[Ro03] Robins, A.; Rountree, J.; Rountree, N.: Learning and Teaching Programming: A Review and Discussion. In: Computer Science Education, 2003, Vol. 13, Nr. 2; S. 137-172.

[Sc98] Schiffer, S.: Visuelle Programmierung: Grundlagen und Einsatzmöglichkeiten. Bonn, 1998.