CrePes - Warum und wie Schüler ab 8 Jahren Programmieren lernen sollten

Markus Esch, Patrick Gratz, Steffen Rothkugel
Mobile Computing and Communication Research Lab
Faculté des Sciences, de la Technologie et de la Communication
Université du Luxembourg
L-1359 Luxembourg
{markus.esch, patrick.gratz, steffen.rothkugel}@uni.lu

Jörg Jakoby, Ingo Scholtes, Peter Sturm
Systemsoftware und Verteilte Systeme
Fachbereich Informatik
Universität Trier
D-54286 Trier
{jakoby, scholtes, sturm}@syssoft.uni-trier.de

Abstract: In diesem Artikel beschreiben wir eine komfortable und kindgerechte Lernumgebung welche den Erwerb grundlegender Programmierfertigkeiten unterstützt. Diese ist auf der Grundlage mehrjähriger Erfahrungen in der Vermittlung mathematisch-logischer sowie programmiertechnischer Konzepte an Schüler im Alter zwischen 8 und 13 Jahren innerhalb unserer Arbeitsgruppen entstanden. Neben einer Beschreibung einiger, der Software zugrunde liegender Konzepte berichten wir zudem von unseren Erfahrungen in deren Einsatz in unterschiedlichsten Lernsituationen.

1 Einleitung

Obwohl die Computerprogrammierung im Alltag kaum eine Rolle spielt und meist vollständig dedizierten Spezialisten vorbehalten bleibt, finden sich ihr inhärente Konzepte wie analytisches und algorithmisches Denken doch in vielen Bereichen des täglichen Lebens wieder. In den letzten Jahren setzt sich daher zunehmend die Erkenntnis durch, Programmierung nicht mehr nur als notwendiges Mittel zur Softwareentwicklung zu betrachten, sondern stattdessen als Teil der allgemeinen mathematisch-logischen Ausbildung einer breiten Öffentlichkeit zugänglich zu machen [2]. Im Rahmen eines von verschiedenen Seiten geförderten Projekts versuchen unsere Arbeitsgruppen daher seit Anfang 2003 Jugendlichen zwischen 8 und 12 Jahren grundlegende Programmierkonzepte zu vermitteln. Um dies auch Schülern der anvisierten Altersgruppe zu ermöglichen, kamen zu Beginn des Projekts Ansätze zum Einsatz, welche die Computerprogrammierung von allen für Schüler unnötigen technischen Details befreien. Während dies im Allgemeinen recht gut gelang, entstanden im täglichen Einsatz dieser Programmierumgebungen jedoch nach und

nach Ideen für weitergehende Unterstützungstechniken, die keine der bis dato verfügbaren Ansätze bot. Auf Basis dieser Ideen entwickelten wir im Verlauf der letzten vier Jahre die auf dem "Turtle Graphics"-Prinzip [1] beruhende Programmierumgebung CrePes, welche einen ausgeprägt interaktiven und spielerischen Charakter sowie eine hinsichtlich der gewonnenen Erfahrungen erweiterte Funktionalität bietet. Die dank unterschiedlicher Förderungen zur Verfügung stehenden Mittel konnten dabei nachhaltig investiert werden, um ein abwechslungsreiches Repertoire vorgefertigter Unterrichtseinheiten zu schaffen. Diese stehen Lehrenden und Lernenden gleichermaßen in Form eines Lernportals zur freien Verfügung und erlauben eine einfache und bedarfsgerechte Unterrichtsgestaltung. Im Rahmen dieses Artikels möchten wir diese Programmierumgebung sowie deren zugrunde liegenden Konzepte vorstellen und über unsere in den vergangenen fünf Jahren gewonnenen Erfahrungen im Einsatz der Software an Schulen und bei einigen anderen Gelegenheiten berichten.

2 Das CrePes-Projekt

Wie bereits einleitend erwähnt, ist es zur Vermittlung programmiertechnischer Fertigkeiten in Schulen zunächst notwendig, Computerprogrammierung von technischen Details zu befreien und Programmabläufe auf einfache Art und Weise zu visualisieren. Über die Wahl der dafür zu nutzenden Methoden gibt es jedoch zum Teil recht unterschiedliche Auffassungen. Um die Programmausführung zu veranschaulichen ist es zunächst notwendig, diese in der Vorstellung des Nutzers vom Rechnermodell mit all seinen technischen Details abzukoppeln. Hierzu können unterschiedliche Visualisierungsmethoden verwendet werden. Die Mehrzahl der existierenden visuellen Programmierumgebungen stehen in der Tradition von "Turtle Graphics" [1], eines ursprünglich 1976 in LOGO implementierten Zeichensystems, bei dem eine mit einem Zeichenstift versehene virtuelle Schildkröte über den Bildschirm bewegt wird, um zweidimensionale Strukturen zu zeichnen. Obwohl die meisten der verfügbaren Lösungen auf diesem bewährten Konzept beruhen, unterscheiden sie sich in der Wahl der Programmierparadigmen. So nutzt beispielsweise das von der ETH Zürich entwickelte Kara endliche Automaten [2], wohingegen Vertreter wie Richard Pattis' "Karel the Robot" [3] oder der "Hamster-Simulator" der Universität Oldenburg [4] das Programmiersprachen-Paradigma verwenden. Während Karel the Robot eine proprietäre Programmiersprache Karel++ einführt, verwendet der Hamster-Simulator Java. Ein weiteres, in diesem Kontext recht weit verbreitetes Programmiermodell sind Flußdiagramme, eingesetzt beispielsweise in Raptor, welches von der US Air Force entwickelt wurde [5]. Eine andere Herangehensweise stellt GNOME dar [6]. Hier editiert der Nutzer direkt den abstrakten Syntaxbaum, eine Datenstruktur welche üblicherweise in Compilern für die interne Repräsentation eines Programms zum Einsatz kommt. Einen vollständig anderen Ansatz als "Turtle Graphics" verwendet das vom Rochester Institute of Technology entwickelte "Muppets". Bei diesem, in der Hauptsache auf Informatik-Studenten höheren Semesters ausgerichteten Projekt, können Aussehen und Verhalten dreidimensionaler Objekte in einer virtuellen Online-Welt programmiert werden. Der Fokus liegt hier auf der Mächtigkeit der Programmierschnittstelle und auf der Möglichkeit des kollaborativen Lernens in einer immersiven Umgebung [7].

2.1 Die Software

Für die Entwicklung der CrePes Umgebung haben wir uns für das bewährte "Turtle Graphics" Konzept sowie für die Nutzung des Programmiersprachenparadigmas entschieden. Den Hauptvorteil dieser Wahl sehen wir im Angebot eines möglichst einfachen Migrationspfads hin zu realen Programmierumgebungen. Zudem hatten zu diesem Zeitpunkt bereits erste Erfahrungen mit anderen auf Programmiersprachen beruhenden Umgebungen gezeigt, dass auch Schüler der anvisierten Altersgruppe kaum Probleme mit diesem Ansatz haben. Die im Lauf der letzten Jahre entstandene Software bietet eine integrierte Entwicklungsumgebung (IDE), welche auf die speziellen Bedürfnisse von Jugendlichen ab acht Jahren zugeschnitten ist. Sie ermöglicht das Erstellen von Programmen, welche eine virtuelle Spielfigur (z.B. einen Roboter) über ein wahlweise zwei- oder dreidimensionales Spielfeld steuert, um bestimmte Aufgaben zu lösen. Die Spielfigur ist dabei mit einer Reihe grundlegender Funktionen ausgestattet welche es ihr erlauben z.B. sich ein Feld vorwärts zu bewegen oder zu drehen, Objekte aufzunehmen bzw. abzulegen oder Schalter an- bzw. auszuschalten. Das Spektrum der möglichen Aufgaben reicht hierbei von einfachen Übungen wie "Steuere die Figur durch ein Labyrinth" bis hin zu deutlich komplexeren Herausforderungen wie "Sortiere eine Folge von Batteriestapeln der Größe nach". Während vielen existierende Umgebungen ein ähnlicher Ansatz zugrunde liegt, stellt die große Erweiterbarkeit sowie die Möglichkeit der Interaktion mit und zwischen Objekten des Spielfeldes eine besondere Funktionalität dar. Dabei erlaubt ein spezieller Editor das Erstellen eigener Spielobjekte sowie die Definition von Interaktionen zwischen Objekten auf dem Spielfeld. So kann z.B. durch das Betätigen eines bestimmten Schalter-Objekts das Öffnen einer Tür bewirkt werden. Zur Programmierung der Spielfigur kommt eine Teilmenge der modernen Programmiersprache C# zum Einsatz. Die eigentliche Programmiersprache wurde dabei um die vorgenannten grundlegenden Funktionen der Spielfigur erweitert. Zur Lösung der Aufgaben können diese dann zusammen mit bereits in C# vorhandenen Kontrollstrukturen verwendet werden. Folgender Abschnitt beschreibt die wesentlichen Komponenten der kindgerechten Entwicklungsumgebung CrePes. Diese lässt sich dabei in drei Komponenten gliedern (siehe Abbildung 1):

- Ein Code-Editor der mit komfortablen Eigenschaften wie z.B. Codevervollständigung und Hervorhebung der Programmiersprachensyntax ausgestattet ist
- Ein Spielfeld, auf dem die programmierbare Spielfigur, Barrieren, vordefinierte sowie selbst erstellte Objekte mit frei definierbaren Eigenschaften platziert werden können
- Ein Ein-/Ausgabefenster welches eine textbasierte Interaktionsmöglichkeit zur Programmlaufzeit bietet, die Ausgabe von Fehler- und Fortschrittsinformationen des Compilers ermöglicht und eine Beschreibung der zu lösenden Aufgabe anzeigt

Ein wesentliches Augenmerk bei der Entwicklung der Software wurde auf deren Praxistauglichkeit im Einsatz an Schulen aufgrund der dort häufig wenig leistungsfähigen Hardware sowie eingeschränkten Benutzerrechten gelegt. Aus diesem Grund besteht die Software aus einer schlanken Basisinstallation sowie einer Plugin-Schnittstelle, welche eine

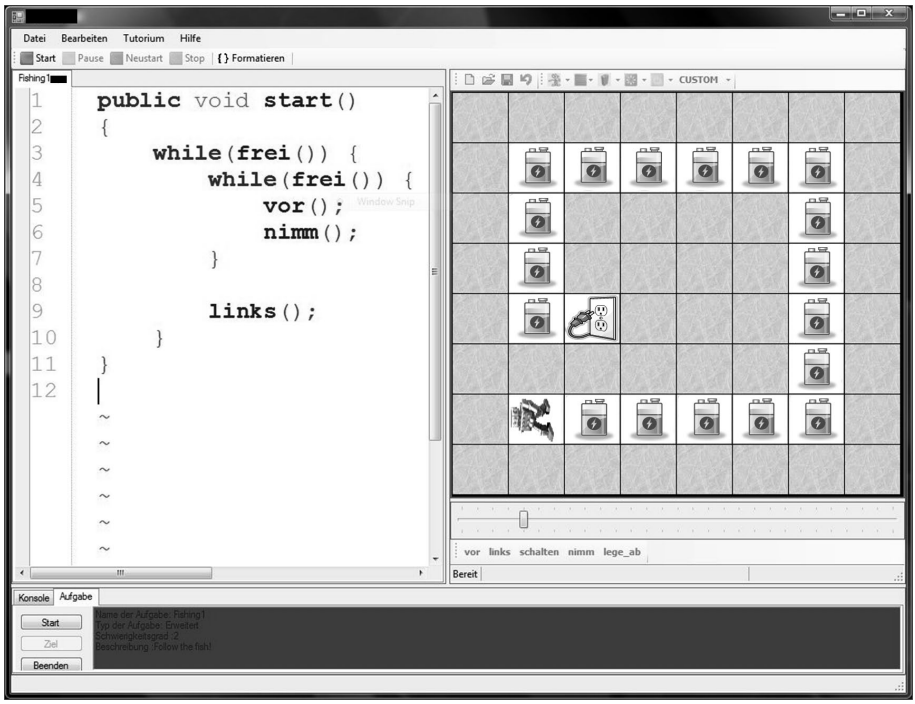

Abbildung 1: Die IDE der CrePes Software

an vorhandene Ressourcen angepasste Erweiterung um optionale Funktionalitäten bietet. So existiert beispielsweise ein Zusatzmodul, welches eine dreidimensionale Darstellung des Spielfelds samt Roboter und Spielobjekten ermöglicht. Weitere Plugins erlauben zudem die Konstruktion eigener dreidimensionaler Spielobjekte und Spielfiguren, wodurch sie Software auch um kreative Einsatzmöglichkeiten erweitert wird.

2.2 Unterstützung bei der Aufgabenerstellung

Der häufigste Einsatz der Software im Unterricht sieht vor, dass die Schüler bestimmte Aufgaben auf einem Spielfeld lösen. Solche Aufgaben können beispielsweise aus der Angabe eines Programms zur automatischen Verfolgung einer Objektspur oder zum Finden eines Weges durch ein Labyrinth sein. Die hauptsächliche hinter der Entwicklung der Software stehende Motivation war eine weitgehende Unterstützung des Lehrenden bei der Erstellung von Aufgaben sowie eine automatisierte Kontrolle erstellter Lösungen zu erreichen. Hierzu wurde eine spezielle Aufgabenabstraktion konzipiert. Eine Aufgabe besteht aus der Definition eines Spielfelds sowie einer Aufgabenbeschreibung. Um die Erstellung von Aufgaben für Lehrende möglichst einfach zu gestalten, bietet die Software einen inte-

grierten Aufgabeneditor. Dabei gibt es zwei grundsätzlich verschiedenartige Aufgabentypen: Einfache sowie erweiterte Aufgaben. Einfache Aufgaben werden lediglich durch die Angabe zweier Spielfelder definiert, einem Start- und einem Ziel-Spielfeld. Die Aufgabe für den Lernenden besteht nun darin, ein Programm zu schreiben, welches das Spielfeld vom Start- in den Zielzustand überführt. Eine wichtige Erfahrung im Einsatz dieses Aufgabentyps im Unterricht war, dass Schüler im Allgemeinen dazu neigen Lösungen für das spezifisch vorliegende Spielfeld zu erstellen. Für Variationen der Aufgabenstellung denen das gleiche Problem zugrunde liegt sind diese Lösungen meist ungeeignet. So kann man zum Beispiel für eine Aufgabe, bei der der Roboter ein Labyrinth durchlaufen soll recht einfach eine Lösung angeben, die für ein konkretes Labyrinth korrekt ist. Deutlich komplexer gestaltet sich jedoch die Angabe einer Lösung, die für jedes beliebige Labyrinth gleicher Struktur funktioniert. Unserer Erfahrung nach fällt es Schülern der anvisierten Altersgruppe meist schwer, das einer Aufgabe zugrunde liegende allgemeine Problem zu erkennen und eine entsprechend generisch funktionierende Lösung zu erarbeiten. Um dies speziell zu fördern wurde daher ein erweiterter Aufgabentyp entwickelt, bei dem das Spielfeld vor jedem neuen Programmdurchlauf anhand eines zu erkennenden Prinzips neu erzeugt wird. Zur Erstellung solcher erweiterter Aufgaben ist die Angabe von Programmcode notwendig, welcher beim Laden der Aufgabe ein zufälliges Spielfeld nach einem bestimmten Muster erstellt, auch hierfür steht dem Lehrenden wieder eine Unterstützung in Form eines Editors zur Verfügung. Auf diese Weise können auf einfache Art und Weise ganze Aufgabenklassen definiert werden. Jede konkrete von der Software erzeugte Aufgabe ist eine Instanz dieser Aufgabenklasse. Lösungen erweiterter Aufgaben werden automatisch von der Software auf Korrektheit geprüft. Dabei wird eine Lösung nur dann als korrekt erkannt, falls sie für eine zufällige Stichprobe mehrerer Aufgabeninstanzen eine frei definierbare Zielbedingung erfüllt. Abbildung 2 zeigt beispielhaft vier unterschiedliche zufällig erzeugte Instanzen einer solchen erweiterten Aufgabe, deren Ziel das Laufen entlang variabel hoher Hürden ist. Im vorliegenden Beispiel ist leicht ersichtlich, wie sich der Schwierigkeitsgrad einer einfachen Lösung von dem einer generischen, für alle Instanzen funktionierenden Lösung unterscheidet. Während das Erstellen einer einfachen Aufgabe mit Hilfe des Aufgabeneditors sehr intuitiv per Drag-and-Drop und somit auch für die Schüler selbst möglich ist, können erweiterte Aufgaben in der Regel nur von Lehrenden erstellt werden, da dies die Angabe von Programmcode erfordert.

2.3 Das Lernportal

Ein wichtiger Vorteil der oben beschriebenen Aufgabenabstraktion ist die Möglichkeit des Selbststudiums, da die Korrektheit einer angegebenen Lösung von der Software selbst bewertet werden kann. Um Schüler beim selbstständigen Lernen zu unterstützen, wurde eine Vielzahl so genannter Tutorien entwickelt. Diese bestehen aus einer Menge inhaltlich zusammenhängender Aufgaben sowie einem erklärenden Text, welcher jeweils ein bestimmtes Programmierkonzept erläutert. Jedes Tutorium wird dabei durch eine Geschichte oder einen bestimmten thematischen Kontext motiviert. So existiert zum Beispiel eine Reihe von Tutorien, welche die Geschichte eines fiktiven Roboters erzählen und in denen die

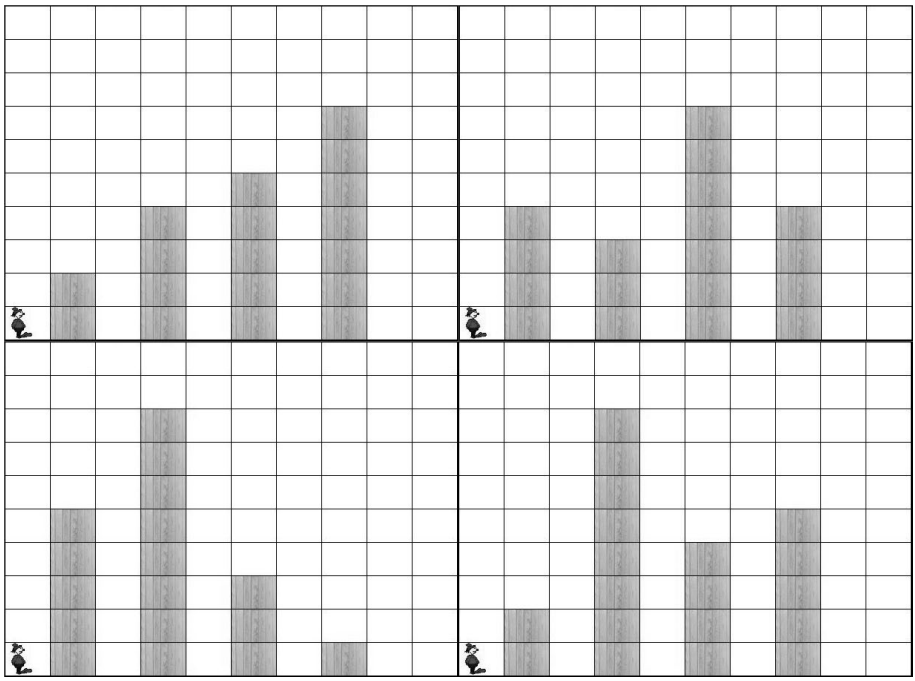

Abbildung 2: Beispiel für vier unterschiedliche Instanzen einer erweiterten Aufgabe

Schüler diesem helfen müssen verschiedene Abenteuer zu bestehen. Im Verlauf dieser zusammenhängenden Tutorien werden dann nach und nach komplexere Konzepte vermittelt und in Form von Aufgaben deren korrekte Anwendung getestet. Tutorien sind gemäß Ihres Schwierigkeitsgrades und des notwendigen Vorwissenens sortiert. Auf diese Weise ist es möglich, ohne Vorkenntnisse beginnend, durch das Bearbeiten aufeinander folgender Tutorien sämtliche im Rahmen des CrePes Projekts vermittelten Programmierkonzepte selbstständig zu erlernen. Eine große Zahl der in den vergangenen Jahren entwickelten Tutorien sowie weiterführende Informationen wurden im Rahmen einer Förderung in Form eines Lernportals zusammengefasst. Hier finden Lehrer all das, was sie zum Einsatz der Software im Unterricht benötigen. Schülern wird durch das breite Angebot vorgefertigter Unterrichtseinheiten eine eigenständige Einarbeitung in grundlegende Programmierkonzepte ermöglicht. Abbildung 3 zeigt einen Auszug dieser Website.

3 Erfahrungen

Ersten Erfahrungen im Kontext des CrePes Projekts wurden von 2003 bis 2006 im Verlauf eines durch eine Elterninitiative veranstalteten Kurses für hochbegabte Kinder gesammelt [9]. Im Rahmen dieses Kurses ergaben sich nach und nach die Anforderungen für die

Abbildung 3: Das Webportal

im vorangegangenen Abschnitt vorgestellte Software. Seit 2005 veranstalten unsere Arbeitsgruppen regelmäßig singuläre mehrtägige Programmier-Workshops sowie weitere regelmäßig stattfindende Lehrveranstaltungen, beispielsweise wöchentlich in Form einer Arbeitsgemeinschaft in Kooperation mit einem örtlichen Gymnasium. Im jährlichen Rhythmus wird zudem ein dreitägiger Workshop an der Universität durchgeführt. Darüber hinaus wurden diverse andere Kurse und Veranstaltungen angeboten, die Kindern die Möglichkeit gaben sich unter Betreuung mit der Software zu beschäftigen. Das Alter der Teilnehmer all dieser Kurse lag zwischen 8 und 13 Jahren, mit zum Teil recht großen Altersunterschieden von drei und mehr Jahren innerhalb einer Gruppe. All diesen Programmierkursen lag ein sehr zurückhaltendes Lehr- und Lernprinzip zugrunde, bei dem sich der Lernerfolg primär katalytisch einstellen sollte. Ein besonderer Vorteil des außerschulischen Charakters all dieser Veranstaltungen ergibt sich aus der Tatsache, dass es keinen einzuhaltenden Lehrplan gibt. Ebenfalls wurde nicht vorausgesetzt, dass die Schüler am Ende einen fest

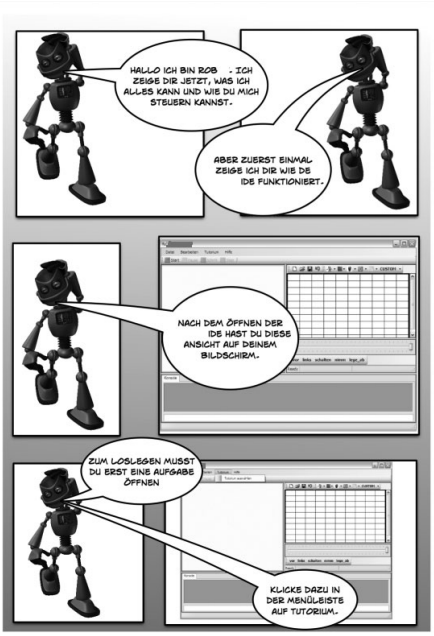

Abbildung 4: Comic zur Einführung der Software

definierten Wissensstand erreichen. Alle angebotenen Lehrveranstaltungen haben einen ähnlichen Ablaufplan. Dieser sieht die Vermittlung grundlegender algorithmischer Kenntnisse in verschiedenen Stufen vor. Nach einer grundlegenden Einführung in die Funktionsweise der Software, welche häufig in Form von Comics erfolgte (siehe Abbildung 4) sieht die erste Stufe eine Beschäftigung mit grundlegenden Befehlen (wie z.B. vor(), links(), nimm(), lege ab(), schalte()) vor. In der Regel erfolgte der erste Kontakt mit der Software in Form einer Aufgabe, die eine ungebräuchliche Verwendung der IDE vorsah, wie z.B. “Wer malt das schönste Bild auf dem Spielfeld”. Dies ist mithilfe verschieden geformter und gefärbter Spielobjekte möglich, die von den Schülern per Maus auf dem Spielfeld platziert werden können. Nach dem Lösen einfachster Aufgaben mit Hilfe einfacher Befehlssequenzen folgt die zweite Stufe, welche die Verwendung von while-Schleifen sowie dazu benötigte Bedingungen vorsieht. Diese Kontrollstruktur erlaubt die Wiederholung einfacher Befehle bis zum Eintreffen einer bestimmten Bedingung, beispielsweise das Bewegen der Spielfigur bis zum Erreichen einer Barriere. Die recht frühe Einführung von while-Schleifen erschließt sich durch die Tatsache, dass dies eine große Auswahl intuitiv verständlicher Aufgaben erlaubt. Im vorgenannten Beispiel ist den Schülern beispielsweise recht leicht vermittelbar, dass die Wiederholung des Befehls vor() bis zum Erreichen der Barriere einfacher und natürlicher ist, als ein Abzählen der Schritte und ein mehrfaches Schreiben des Befehls. Zudem bietet der Einsatz von Endlosschleifen die Möglichkeit eine beliebte Form Aufgaben zu stellen, in denen beispielsweise die Bewegung des Roboters während der Laufzeit eines Programms durch das interaktive Setzen von Barrieren

gesteuert werden soll. Eine dritte Stufe hatte die Einführung von Prozeduren und der if-Anweisung zum Inhalt. Das Prozedur- bzw. Unterprogrammkonzept wird dabei zunächst in einer Form vermittelt, in der sich die Schüler diese als selbst erweiterbare Funktionen der Spielfigur vorstellen können. Dies geschieht in der Regel durch die Erstellung eines eigenen Befehls “rechts()”, welcher aus didaktischen Gründen nicht Teil der standardmäßig vorhandenen Funktionen ist. In Lehrveranstaltungen in denen genügend Zeit zur Verfügung stand wurde noch eine vierte Stufe vermittelt, welche den Umgang mit Variablen sowie Rekursion zum Inhalt hatte.

Neben typischen Aufgaben in denen ein Programm zum Erreichen eines bestimmten Ziels erstellt werden muss, werden immer wieder auch alternative Aufgabentypen getestet. So haben sich beispielsweise Aufgaben bewährt, in denen es notwendig ist, die Funktionsweise eines vorgegebenen sehr komplexen Programms zu ergründen. Dies kann beispielsweise durch Manipulation des Spielfelds und wiederholtes Testen des Programms unter veränderten Anfangsbedingungen erfolgen. Zudem hat sich das Konzept bewährt die Teilnehmer ab einem gewissen Kenntnisstand selbst mit der Erstellung von Aufgaben sowie einer dazu passenden Musterlösung zu beschäftigen, welche dann wechselseitig gestellt werden können. Die veranstalteten ganztägigen Workshops brachten zudem die Erfahrung, dass die Lösung von Aufgaben nach spätestens drei Stunden die Konzentrationsfähigkeit und die Motivation der Schüler beeinträchtigt. Zur Auflockerung wurden in diesen Workshops daher immer wieder auch andere Aufgaben integriert, die den Schülern eine Pause von den Programmieraufgaben gönnten. Zu diesen Aufgaben zählten z.B. ein Informatik-Quiz zu dessen Lösung eine Web-Recherche notwendig war, die Erstellung eines eigenen dreidimensionalen Roboters oder auch eine Bastelaufgabe, wie das Basteln von Spielfiguren, Spielfeld und Spielobjekten. Zur weiteren Auflockerung wird immer wieder auch das Entschlüsseln von Geheimtexten oder eine Schnitzeljagd verwendet, die sowohl real auf dem Universitätsgelände als auch in Form einer Online-Schnitzeljagd im Internet stattfand.

Des Weiteren stellte sich erwartungsgemäß heraus, dass das Verständnis der Schüler stark variiert. Eine bemerkenswerte Erkenntnis war zudem, dass dies in den wenigsten Fällen auf das unterschiedliche Alter der Teilnehmer zurückzuführen war. Häufig wiesen Schüler, die zum Teil drei bis vier Jahre jünger waren als andere Teilnehmer das beste Verständnis verschiedener Programmierkonzepte auf. Aufgrund einer in der Regel ausgesprochen guten Betreuungssituation kann durch diese starke Variation verursachten möglichen Problemen jedoch meist vorgebeugt werden, indem ab einem gewissen Zeitpunkt jedem Schüler auf seinen individuellen Wissensstand zugeschnittene Aufgaben zugeteilt werden. Dank des Fehlens fester Lehrpläne und Lernziele wird zudem ein bewusst langsames Lerntempo anvisiert und durch das wiederholte Stellen von Aufgaben gleichen Schwierigkeitsgrads eine Vertiefung des Gelernten gefördert. Dank der damit verbundenen Erfolgserlebnisse erhöht dies zudem den Spaß der Schüler an den Aufgaben.

4 Zusammenfassung

Im Lauf der letzten Jahre hat sich die CrePes Programmierumgebung im Einsatz in unterschiedlichen Lernszenarien bewährt. Eine Schlüsselrolle nimmt dabei die Möglichkeit ein, generisches Denken und eine Abstraktion von konkreten Probleminstanzen hin zu Problemklassen zu erzwingen. In diesem Kontext hat sich in unseren Augen ganz besonders die Möglichkeit der Definition erweiterter Aufgaben bewährt. Dieser erzwingen generische, auf ein bestimmtes abstraktes Problemmuster zugeschnittene Lösungen und stellen damit einen interessanten Mehrwert gegenüber anderen existierenden Ansätzen dar. Die in den letzten Jahren gewonnenen Erfahrungen bestätigen zudem die Vermutung, dass der ungezwungene Zugang von Kindern und Jugendlichen zu Computern auch zur Vermittlung grundlegender algorithmischer Grundfertigkeiten in sehr jungem Alter genutzt werden kann. Die sich daraus ergebenden Chancen sehen wir dabei weniger in den erworbenen programmiertechnischen Fertigkeiten als in der generellen Stärkung der analytisch-kognitiven Fähigkeiten der Kursteilnehmer. Hier sind insbesondere die positiven Erfahrungen mit der Erstellung eigener Aufgaben durch die Kursteilnehmer hervorzuheben. Gemäß des Prinzips "docendo discimus" bietet das damit verbundene Schlüpfen der Schüler in die Rolle des Lehrenden ganz besondere Möglichkeiten analytischen Fähigkeiten weiterzuentwickeln und spielerisch die Natur unterschiedlichster algorithmischer Probleme zu erkunden. Im täglichen Unterrichtseinsatz bietet die Software CrePes dem Lehrenden auf zweierlei Arten Unterstützung an. Zum einen ist mit dem Lernportal eine große Menge vorgefertigter Unterrichtseinheiten verfügbar, zum anderen steht mit dem Aufgabeneditor eine komfortable Unterstützung bei der Erstellung eigener Lehreinheiten zur Verfügung. Die kostenlos erhältliche Programmierumgebung ist auf allen Microsoft Windows Betriebssystemen ab Windows 98 lauffähig. Vorausgesetzt wird lediglich eine installierte Microsoft .NET 2.0 Laufzeitumgebung, welche ebenfalls frei erhältlich ist. Die Software wurde insbesondere im Hinblick auf die an Schulen vorherrschende technische Infrastruktur entwickelt und stellt lediglich moderate Anforderungen an die zur Verfügung stehende Hardware. Die unbeaufsichtige und automatisierte Installation der Software auf einer großen Menge von Computern ist mittels eines MSI-Pakets ebenfalls problemlos möglich.

Literatur

[1] Lieberman, H.: The TV Turtle - A Logo Graphics System for Raster Displays. In The papers of the ACM symposium on Graphic languages, pages 66-72, Florida, 1976

[2] Hartmann, W.; Nievergelt, J.; Reichert, R.: Kara, finite state machines, and the case for programming as part of general education In Proceedings of the 2001 IEEE Symposia on Human-Centric Computing Languages and Environments, pages 135141. Stresa, Italy, September 2001.

[3] Pattis, R. E.: Karel the Robot: A Gentle Introduction to the Art of Programming, John Wiley & Sons, Inc., New York, NY, USA, 1994

[4] Boles D.: Programmieren spielend gelernt mit dem JAVA Hamster Modell, Teubner Verlag, Wiesbaden, 2002

[5] Carlisle, M. C.; Wilson, T. A.; Humphries J.W.; Hatfield S.M.: Raptor, a visual programming environment for teaching algorithmic problem solving, In Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '05), pages 176-180, St. Louis, USA, February 2005

[6] Garlan, D.B.; Miller, P.L.; GNOME: An Introductory Programming Environment Based on a Family of Structure Editors, SIGSOFT Softw. Eng. Notes 9, pages 65-72, May 1984

[7] Phelps, A.M.; Bierre, K.J.; Parks, D.M.: Course design & learning enhancement: MUPPETS: Multi-User Programming Pedagogy for Enhancing Traditional Study, In Proceedings of the 4th Conference on information Technology Curriculum, pages 100-105, Lafayette, USA, October 2003

[8] Brusilovksy, P.; Calabrese, E.; Hvorecky, J.; Kouchnirenko, A.; Miller, P.: Mini-languages: A Way to Learn Programming Principles. Education and Information Technologies 2, pages 65-83, Jan. 1998

[9] Peter Sturm: Eine Computer-AG für hochbegabte Kinder, erschienen in: "Bildungsentwicklungsland Deutschland?", Herausgeber K. Zey-Wortmann, D. Dietrich, M. Reinsch, Bertuch-Verlag, 2005, Seite 49ff