PUCK – ein Sommernachtstraum

Ein Open-Source-Projekt für die Sekundarstufe I

von Lutz Kohl, Gabor Meißner und Harald Schmidt

Doch wer dich freundlich grüßt, dir Liebes tut,
Dem hilfst du gern, und ihm gelingt es gut.

William Shakespeare (1564–1616)
„Ein Sommernachtstraum", 2. Aufzug, 1. Szene

PUCK ist ein visuelles System, mit dem es möglich ist, grundlegende Strukturen der Programmierung zu erlernen. Bausteine können zu Programmen verknüpft werden. Da Syntaxfehler durch die visuelle Konstruktion nicht möglich sind, können die erstellten Programme stets direkt ausprobiert werden. So können einfache Anweisungen, Kontrollstrukturen, das Variablenkonzept und schließlich Prozeduren mit Parametern schrittweise erarbeitet werden.

PUCK ist ein Open-Source-Projekt, das allen Interessierten kostenfrei zur Verfügung steht. Entwickelt wurde PUCK nach den Anforderungen von Informatiklehrern in Thüringen, wo im Zentralabitur OBERON-2 eingesetzt wird. In Anlehnung an Shakespeares *Sommernachtstraum*, in dem Oberon der König der Elfen ist, wurde das System nach dem Gehilfen Oberons „Puck" genannt.

Wenn eine Programmiersprache in der Schule eingesetzt werden soll, müssen verschiedene Kriterien erfüllt sein. So fordert Fothe u. a., dass geeignete Unterrichtsmaterialien vorliegen müssen (vgl. Fothe, 2006, S. 6–7). Für das Programmiersystem PUCK wurden deshalb bereits erste Materialien zum selbstständigen Einarbeiten entwickelt (vgl. Kohl, 2006). Auch in diesem Beitrag wird zunächst erprobter Unterricht vorgestellt (Kapitel „Die Unterrichtskonzeption") und anschließend das Programmiersystem PUCK selbst (Kapitel „Das Programmieren mit PUCK").

Die Unterrichtskonzeption

Die Idee

Wichtig für den Einsatz eines neu entwickelten Programmiersystems für die Schule ist vor allem, dass dies auch in der Unterrichtspraxis erprobt wird und die damit gewonnenen Ergebnisse direkt in die Weiterentwicklung einfließen. Deshalb soll hier zunächst einiges aus den gewonnenen Unterrichtserfahrungen vorgestellt werden. Grundlage hierfür war eine Projektarbeit, die im Rahmen einer Lehrveranstaltung zur Informatikdidaktik an der Friedrich-Schiller-Universität Jena angefertigt wurde (vgl. Meißner/Schmidt, 2007). Die Konzeption besteht aus fünf 45-minütigen Einheiten, die Module genannt werden und an die der Anspruch gestellt wird, dass mit ihnen ein methodisch moderner Informatikunterricht für die Sekundarstufe I realisiert werden kann. Elemente der Unterrichtsreihe wurden in einer 8. Klasse des Carl-Zeiss-Gymnasiums Jena erprobt. Die entwickelten Unterrichtsmaterialien stehen kostenfrei zum Herunterladen zur Verfügung (siehe Literatur und Internetquellen – „Puck Unterrichtskonzeption").

Die Lernziele und die Voraussetzungen

Die Lernziele dieser Unterrichtskonzeption sind in Fach-, Methoden-, Sozial- und Selbstkompetenzen gegliedert. Die fachlichen Kompetenzen, die von den Schülerinnen und Schülern im Unterricht erworben werden sollen, bestehen aus einer grundlegenden Beherrschung von ausgewählten Programmierelementen sowie in einem sicheren Umgang mit Begriffen, die mit der Programmierung unmittelbar zusammenhängen. Somit kann bereits frühzeitig an die fundamentale Idee der Algorithmisierung herangeführt werden (vgl. Schubert/Schwill, 2004 S. 89 f.).

Die von den Schülerinnen und Schülern zu erwerbenden methodischen Kompetenzen sind:

- die kritische Auseinandersetzung mit einem Begriff der Informatik – insbesondere der Programmierung – und dessen gesellschaftlicher sowie schulischer Bedeutung (Modul 1),
- der Umgang mit einem Programmiersystem (PUCK) und das Entwickeln von einfachen Programmen (Module 2 und 3)

Inhalt	Material (M)
Modul 1	
Lehrervortrag über den Begriff „Programmierung“	M1: Präsentation zum Lehrervortrag „Programmierung“
Vorbereitung und Durchführung der Gruppenphase	M2-1: Pro-Text M2-2: Kontra-Text
Podiumsdiskussion	
Ergebnissicherung durch Gruppe 3 und Lehrkraft	
Modul 2	
Tägliche Übung mit einem Lückentext	M3: Lückentext
Lehrervortrag zur Einführung in PUCK	M4: Was ist PUCK?
Die Lehrkraft stellt das Vorgehen bei der Programmierung mit Entwurf, Implementierung und Test eines Programms zur Body-Mass-Index-Berechnung (Berechnung von Normal- oder Übergewicht) vor. Die Schülerinnen und Schüler vollziehen die Vorgehensweise Schritt für Schritt am PC nach.	M5: Präsentation und Screenshot zum BMI-Programm
Die Ergebnisse werden durch eine Schülerin oder einen Schüler zusammengefasst.	
Modul 3	
Eine Schülerin oder ein Schüler erklärt mithilfe der Lehrkraft, wie man mit PUCK ein Programm schreibt und wie die Oberfläche zu bedienen ist.	
Lehrervortrag über Ein- und Ausgabe, Ausdrücke sowie Variablen	M6: Präsentation für Eingabe, Ausgabe und Ausdrücke
Implementierung eines Programms zur Berechnung des Volumens und der Oberfläche eines Quaders in Partnerarbeit	
Vorführung und Diskussion der Ergebnisse einer Gruppe	
Modul 4	
Wiederholung der Ergebnisse der letzten Stunde durch die Vorführung eines Schüler-Programms	
Lehrervortrag über die Verwendung der Dokumentation am Beispiel des *Line*-Bausteins	
Programmierung unter Verwendung des *Rect*-Bausteins in Einzelarbeit mithilfe der Dokumentation	
Zwei Schülerinnen oder Schüler stellen ihre Ergebnisse vor.	
Modul 5	
Zur Wiederholung wird eine weitere Suchaufgabe unter Verwendung der Dokumentation gestellt. Diesmal wird aber das Programm nicht implementiert, sondern lediglich entworfen.	
Das Beschreibungsspiel wird vorgestellt, und die Gruppen werden eingeteilt.	M7: Begriffe für das Beschreibungsspiel
Durchführung des Spiels	
Auswertung	
Alternative zum Modul 5	
Leistungskontrolle	M8: Leistungskontrolle M9: Erwartungshorizont

Tabelle 1: Inhalte und verwendete Materialien der einzelnen Module.

▷ sowie das effiziente Arbeiten mit einer Programmdokumentation am Beispiel der PUCK-Hilfe (Modul 4).

In der gesamten Unterrichtskonzeption sollen das strukturierte Programmieren mit Entwurfs-, Implementierungs- und Evaluationsphase sowie die Gliederung von Programmen in Eingabe, Verarbeitung und Ausgabe berücksichtigt werden.

Sozialkompetenzen sollen durch die Arbeit in unterschiedlichen Sozialformen geübt werden, insbesondere werden die Auseinandersetzung mit anderen Positionen sowie das zielorientierte Arbeiten in Gruppen geschult (Module 1 und 3). Damit soll auch verdeutlicht werden, dass die Entwicklung von Software in der Praxis teamorientiert ist.

Die Phase des Testens und der Evaluation der erstellten Programme ermöglicht eine Reflexion über die eigene Arbeit und einen Vergleich mit anderen Leistungen und trainiert so Selbstkompetenzen.

Für die Unterrichtsreihe wird wenig Fachspezifisches vorausgesetzt. Die Schülerinnen und Schüler sollten grundlegende PC-Kenntnisse haben und bereits mit Programmen, die das Drag-and-Drop-Prinzip verwenden, gearbeitet haben.

Die Unterrichtsinhalte

Modul 1

Der Einstieg in die Programmierung wird mit einem einleitenden Lehrervortrag realisiert. Hierbei wird auf eine technische Einführung etwa unter Verwendung einer Sprachhierarchie zugunsten einer informationsorientierten Einleitung mit dem EVA-Prinzip verzichtet. Im Hauptteil der Stunde wird eine Podiumsdiskussion vorbereitet und durchgeführt. Die zu diskutierende Fragestellung soll sein, ob es für Schülerinnen und Schüler sinnvoll ist, das Programmieren zu erlernen. Ziel ist also eine Auseinandersetzung mit der sozialen, persönlichen, arbeitsweltlichen und informatischen Bedeutung des Programmierens. Letztlich sollte die Debatte zum Programmieren motivieren.

Für die Podiumsdiskussion wird die Klasse in drei Gruppen aufgeteilt. Ein

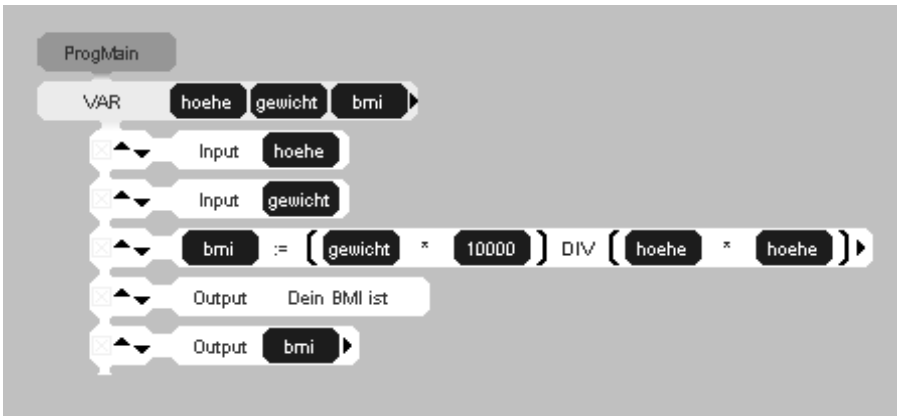


Bild 1: Berechnung des Body-Mass-Index in PUCK formuliert.

Team bearbeitet die Pro- und ein anderes Team die Kontra-Position. Die dritte Gruppe ist für die Vorbereitung und Leitung der Diskussion sowie für eine objektive Ergebnissicherung verantwortlich. Die Erarbeitung der Argumente für oder gegen die Programmierung im Unterricht wird anhand von Texten vorgenommen, die den Schülerinnen und Schülern als Materialien zur Verfügung gestellt werden (vgl. Tabelle 1, vorige Seite). Nach der Gruppenphase werden Vertreterinnen und Vertreter der Gruppen gebeten, vor der Klasse über die Fragestellung zu diskutieren. Die Moderation übernimmt dabei eine Schülerin oder ein Schüler aus dem dritten Team. Abschließend werden einige Schülerinnen und Schüler zur Auswertung aufgefordert, ihre eigene Meinung zu formulieren.

Modul 2

Die Module 2 bis 4 gehören zur Erarbeitungsphase des Projekts; sie beginnen mit einer täglichen Übung und enden mit einer Zusammenfassung zur Stabilisierung und Strukturierung der Unterrichtsinhalte. Die tägliche Übung des zweiten Moduls besteht aus einem Lückentext, der die Inhalte der letzten Stunde aufgreift. Anschließend wird in einem kurzen Lehrervortrag die Programmiersprache PUCK präsentiert.

Hauptinhalt der Unterrichtseinheit ist das Vorstellen der PUCK-Oberfläche anhand eines einfachen Beispielprogramms. Dazu wird eine Body-Mass-Index-Berechnung entworfen, implementiert und getestet. Im Vordergrund steht allerdings das Bedienen der Programmoberfläche durch reproduktive Anwendung; die Lehrkraft präsentiert also das Programm mit einem Beamer, und die Schülerinnen und Schüler vollziehen es direkt am Computer nach.

Der *Body-Mass-Index* (BMI) – auch als Körpermasse-Index (KMI), Kaup-Index oder Körpermassenzahl (KMZ) bezeichnet – ist eine Maßzahl für die Bewertung des Körpergewichts eines Menschen im Verhältnis zum Quadrat seiner Größe:

$$\text{BMI} = \frac{\text{Gewicht in Kilogramm}}{(\text{Körpergröße in Meter})^2}$$

Da Übergewicht ein weltweit zunehmendes Problem darstellt, wird die Körpermassenzahl vor allem dazu verwendet, auf entsprechende Gefährdungen hinzuweisen. Hat der BMI einen Wert zwischen 20 bis 25, gilt dies – je nach Alter – als Normalgewicht, darüber liegende Werte als Übergewicht.

Modul 3

In der dritten Einheit sollen dann zentrale Programmierelemente und -konzepte vermittelt werden. Dies sind die Bausteine zur Ein- und Ausgabe, die Wertzuweisung sowie das Variablenkonzept. Diese Elemente werden zuerst in einem Lehrervortrag präsentiert. Anschließend wenden die Schülerinnen und Schüler ihr erlerntes Wissen bei der Implementierung eines Programms zur Berechnung des Volumens und der Oberfläche eines Quaders in Partnerarbeit an. Die Ergebnisse werden dann von den Schülerinnen und Schülern vor der Klasse vorgestellt und diskutiert.

Modul 4

Im vierten Teil der Unterrichtsreihe sollen die Schülerinnen und Schüler angeleitet werden, einfache Programme unter Verwendung der PUCK-Dokumentation zu erstellen. Die Lehrkraft präsentiert dazu vorab die Dokumentationsseite des *Line*-Bausteins (zum Zeichnen oder Löschen einer Linie) und entwickelt mit dieser Hilfe ein Beispielprogramm. Die Schülerinnen und Schüler sollen dies in ähnlicher Weise am *Rect*-Baustein (zur Ausgabe eines Rechtecks oder einer Ellipse) nachvollziehen. Im Vordergrund dieser Übung stehen dabei nicht die Bausteine und ihre Nutzung in Programmen, sondern das selbstständige, produktive Erarbeiten von Inhalten unter Verwendung der Programmdokumentation.

Modul 5

Das fünfte Modul dient vor allem der Ergebnissicherung und der Lernerfolgskontrolle. Dafür werden die wichtigsten Begriffe zur Programmierung abschließend in spielerischer Form thematisiert. Hierzu wird ein Beschreibungsspiel verwendet, das dem bekannten Gesellschaftsspiel „Activity“ ähnelt. Die in Tabelle 2 (nächste Seite) aufgeführten Begriffe aus der Unterrichtsreihe werden auf Karteikarten festgehalten. Sie sollen mit Mimik und Gestik, durch Zeichnungen oder verbal dargestellt werden. Damit lehnt sich das Beschreibungsspiel an die Repräsentationsmodi von Jerome Bruner an (vgl. Bruner, 1974, S.49). Die Klasse wird in zwei Gruppen aufgeteilt, wobei abwechselnd aus jedem Team eine Schülerin oder ein Schüler eine Minute Begriffe präsentiert. Die jeweilige Gruppe bekommt für jeden erratenen Begriff einen Punkt. Mit dieser Methode setzen sich die Schüler aktiv mit den Unterrichtsinhalten der letzten Stunden auseinander. Durch die Beschreibungen vernetzen sie Begriffe und Erklärungsmuster. Sie vollziehen einen horizontalen und vertikalen Transfer der Sachverhal-

te, indem sie Begriffe mit ähnlichen Begriffen (etwa: „Gegenteil von Output ist Input“) erklären oder sie in anderen Zusammenhängen stellen („ein Fenster kann ich ... öffnen → Datei öffnen“). Alternativ kann die Unterrichtsreihe auch mit einer Leistungskontrolle abschließen. Für beide Möglichkeiten werden konkrete Materialien bereitgestellt.

Fazit

Mit der Unterrichtskonzeption „Einstieg in die Programmierung mit PUCK“ soll eine Möglichkeit aufgezeigt werden, wie der Zugang zum meist sehr komplexen Thema „Programmierung“ schülergerecht sowie methodisch vielfältig realisiert werden kann. Durch die Unterteilung des Unterrichts in reproduktive und produktive Arbeit ergibt sich eine stetige Erweiterung der Schüleraktivität vom einfachen Nachahmen über das Kennen, Können, Wissen hin zum beginnenden, eigenständigen, reflektierten Handeln im Modul 4.

Das Programmiersystem PUCK hat sich schon in einzelnen Projekten an Thüringer Schulen bewährt. Die vorgestellten Unterrichtsmethoden werden häufig in anderen Fächern eingesetzt, wobei sie auch für den Informatikunterricht viele Möglichkeiten bieten. Nicht zuletzt sei hier auch dem Carl-Zeiss-Gymnasium Jena für die Möglichkeit gedankt, Teile der Unterrichtskonzeption zu testen.

Verbale Beschreibung	Zeichnung	Mimik und Gestik
• Programmierung • Name • Integer • Löschen • Zusammenfassen • Befehl • Beschreibung • Entwurf • Implementierung • Test • Wertzuweisung • Operator • DIV • MOD • Dokumentation • Datei • Speichern • Ausführen • Variable • Ganzzahl	• PUCK • Programm • Eingabe • Einhängen • Output • Input • Verarbeitung • EVA-Prinzip • Multiplikation • Text • Line • Öffnen • Koordinaten	• Drag and Drop • Suchen • Rect • Baustein • Ellipse • Hilfe

Tabelle 2: Begriffe für das Beschreibungsspiel.

Das Programmieren mit PUCK

Programmieren ohne Syntaxfehler? Geht das? Ja, mit PUCK!

Die Programme werden aus Bausteinen per *Drag and Drop* (Ziehen und Ablegen) zusammengesetzt. Während des Erstellens kann ein Programm jederzeit ausprobiert werden, da die Bausteine nur syntaktisch korrekt kombiniert werden können (vgl. Kohl, 2005).

Das System ist ein Open-Source-Projekt – jeder kann somit sehen, wie es aufgebaut ist. Im Folgenden sollen nach einer kurzen Einführung in das Programmieren mit PUCK verschiedene, speziell für den Einsatz in der Schule entwickelte Eigenschaften des Systems erläutert werden.

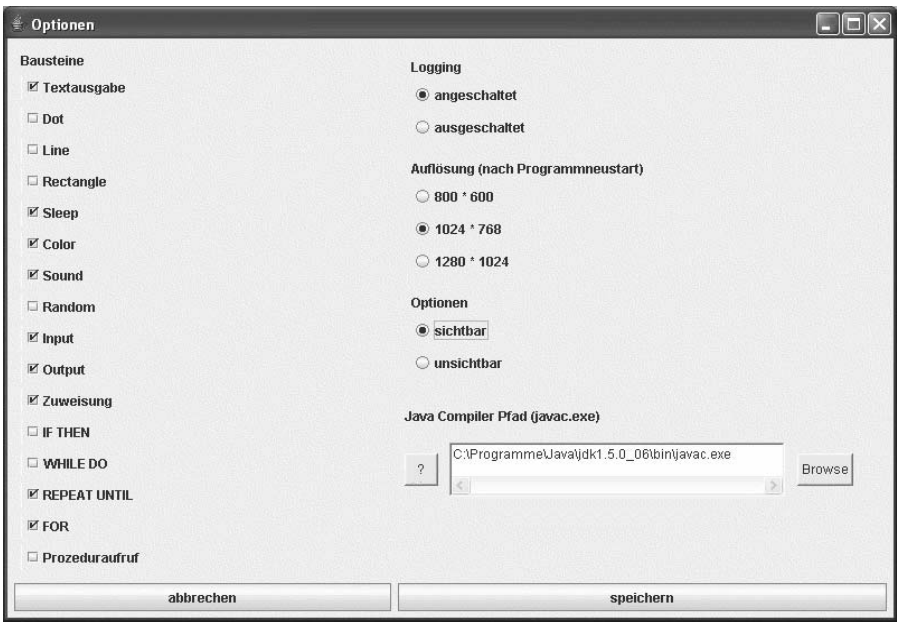


Bild 2: Die verschiedenen Einstellmöglichkeiten im Setup-Programm von PUCK.

Die Installation von PUCK

Vorgehensweise

▷ Da PUCK-Programme in JAVA-Dateien übersetzt werden, ist es nötig, eine aktuelle JAVA Programmierumgebung (JDK – JAVA Development Kit) zu installieren.

▷ Darüber hinaus wurde das PUCK-System selbst in JAVA entwickelt. Um PUCK auszuführen, muss deshalb eine aktuelle JAVA-Laufzeit-

umgebung (JRE – JAVA Runtime Environment) installiert sein. Diese ist bei der Installation der JAVA-Programmierumgebung enthalten.

- ▷ Nach der Installation von JAVA wird der PUCK-Ordner entpackt und auf die Festplatte kopiert.
- ▷ Anschließend wird das Setup-Programm ausgeführt, und der Ort der Datei *javac.exe* (diese befindet sich im Unterordner *bin* des zuvor installierten JDK) wird nach einem Klick auf den *Browse-Button* ausgewählt.
- ▷ Außerdem können die Optionen auf *sichtbar* geschaltet werden, damit während der Arbeit mit PUCK Änderungen an den Einstellungen vorgenommen werden können.
- ▷ Die Bausteine *Sleep*, *Color*, *Sound*, *REPEAT-UNTIL* und *FOR* werden für das erste Programm aktiviert.
- ▷ Durch einen Klick auf den *speichern*-Button schließt sich das Fenster, und die eingestellten Optionen werden in die Datei *PuckOptions.opt* geschrieben, auf die dann auch das PUCK-Programm zugreift.
- ▷ Anschließend kann die Datei *Puck.jar* gestartet werden.

Aufgabe „Teezubereitung“

Situation

Susis Mutter trinkt gern Tee. Verschiedene Teesorten haben unterschiedliche Ziehzeiten. Oft vergisst Susis Mutter den Teebeutel rechtzeitig aus ihrer Tasse zu nehmen und ärgert sich anschließend, weil ihr Tee nicht den gewünschten Geschmack hat. Susi hat die Idee ein Computerprogramm zu schreiben, das ihrer Mutter hilft.

Algorithmus

Ein von Susi entwickelter Algorithmus, der ihre Mutter bei der Teezubereitung unterstützen soll, ist angegeben (siehe Bild 3).

Der vorgegebene Algorithmus ist zu analysieren, in ein Computerprogramm zu übertragen und zu testen.

Änderungen

Susis Mutter hat das Programm ausprobiert und ist begeistert. Sie hat allerdings noch zwei Änderungswünsche:

- ▷ Während des Countdowns soll der Hintergrund Rot eingefärbt sein.
- ▷ Sie möchte bei dem Programm Minuten- oder Sekundenwerte vorgeben können.

Das Programm ist also derart zu verändern, dass es den Anforderungen von Susis Mutter genügt. Anschließend soll es wieder getestet werden.

Teezubereitung mit PUCK

Im Folgenden soll an der Aufgabe „Teezubereitung“ gezeigt werden, wie ein einfaches PUCK-Programm erstellt wird.

Beim Starten des PUCK-Systems wird der Benutzer aufgefordert, einen Namen für das zu entwickelnde Programm einzugeben. Hier bietet sich der Aufgabenname „Teezubereitung“ an. Anschließend öffnet sich die Programmieroberfläche, die in drei Bereiche unterteilt ist. Links befindet sich die Bausteinquelle, in der sich die Standardbausteine sowie die selbstdefinierten Prozeduren befinden. Auf der rechten Seite ist unter einer bausteinspezifischen Attributtabelle ein Textfeld, in dem der Entwurf, Ideen, Kommentare, Erläuterungen und Reflexionen festgehalten werden können. In der Mitte des Bildschirms ist der Arbeitsbereich, in dem sich schon ein Modul-Baustein mit einer Prozedurdeklaration befindet. Auch die Prozedur *ProgMain*, die beim Start eines PUCK-Programms als erstes aufgerufen wird, ist schon vorhanden (siehe Bild 4, nächste Seite).

Für die Entwicklung eines Programms ist ein Entwurf hilfreich. Dieser kann auf Papier oder – wie im Bild 4 zu sehen ist – im Textfeld auf der rechten Seite des PUCK-Fensters notiert werden.

Als erstes werden zwei Variablen angelegt, indem in *ProgMain* rechts des Wortes VAR das kleine schwarze Dreieck mit der linken Maustaste angeklickt wird. Es entsteht eine Variable *a*. Beim nächsten Klick auf das Dreieck entsteht die Variable *b*. Nachdem *a* mit der linken Maustaste angeklickt wurde, öffnet sich auf der rechten Seite des Bildschirms die Attributtabelle der

PROGRAMM Teezubereitung

- Lege die Variablen *sekunden* und *lauf* vom Typ *integer* an.
- Fordere den Benutzer mit „Wie viel Sekunden soll der Tee ziehen?“ auf, eine Zahl einzugeben.
- Speichere die eingegebene Zahl in die Variable *sekunden*.
- Mache Folgendes so lange, bis (*sekunden = 0*) wahr ist:
  - Gib den Wert von *sekunden* aus.
  - Gib den Text „Sekunden“ aus und wechsele in die nächste Zeile.
  - Warte mit der Abarbeitung des Programms 1 Sekunde.
  - Weise der Variablen *sekunden* den Wert von *sekunden–1* zu.
- Gib den Text „Der Tee ist fertig!“ aus.
- Setze die Hintergrundfarbe auf Grün.
- Lasse den Wert der Variablen *lauf* von 0 bis 24 in Schritten der Größe 1 laufen und mache Folgendes:
  - Spiele einen Ton (Tonhöhe: *lauf* Tondauer: *2* Instrument: *xylophon*)

Bild 3:
Ein erster Algorithmus zur Teezubereitung.

Bild 4: Die PUCK-Oberfläche beim Beginn des Lösens der Aufgabe „Teezubereitung“ (hier mit Kurzerklärung).

Variablen, in der Datentyp und Name verändert werden können. Die Variablen werden hier in *sekunden* und *lauf* umbenannt. Die Veränderungen an den Variablen müssen mit der Return-Taste oder einem Klick auf *übernehmen* bestätigt werden. Die Datentypen der Variablen werden in PUCK durch Farben visualisiert. Die Variablen sollen vom Datentyp integer, also blau sein.

Für die Eingabe einer Zahl wird der *Input*-Baustein verwendet. Er wird bei gedrückter linker Maustaste per Drag and Drop von der Bausteinquelle an einen freien Platz im Arbeitsbereich gezogen. Anschließend wird er mit der weißen puzzleartigen Anschlussstelle am unteren Ende von *ProgMain* verbunden. Der *Input*-Baustein enthält ein rotes Feld, das signalisiert, dass an der entsprechenden Stelle noch etwas fehlt. Hier muss nach einem Klick mit der rechten Maustaste die Variable *sekunden*, in der der eingegebene Wert gespeichert werden soll, ausgewählt werden. Um dem Benutzer mitzuteilen, was einzugeben ist, kann nach einem Linksklick auf das Wort *Input* in der Attributtabelle noch die Beschreibung „Wie viele Sekunden soll der Tee ziehen?“ angegeben werden.

Um weitere Bausteine mit der Prozedur *ProgMain* zu verbinden, ist es nötig, noch einige Anschlussstellen hinzuzufügen. Dies geschieht, indem eines der schwarzen Dreiecke in der weißen Anschlussstelle mit der linken Maustaste angeklickt wird. Dadurch entsteht – je nach Pfeilrichtung – eine weitere Anschlussstelle oberhalb oder unterhalb der bereits vorhandenen.

Anschließend wird der *REPEAT-UNTIL*-Baustein unterhalb des *Input*-Bausteins angefügt. Nach einem Klick mit der rechten Maustaste auf den grauen Boolean-Ausdruck *TRUE* wird *Vergleich* ausgewählt. Hierdurch entsteht als Abbruchbedingung für die Schleife ein Vergleich von zwei integer-Ausdrücken. Nun muss auf der linken Seite des Gleichheitszeichens noch die Variable *sekunden* mit einem Rechtsklick ausgewählt werden.

Der Schleifenrumpf des *REPEAT-UNTIL*-Bausteins besteht aus einer nach rechts versetzten Anschlussstelle, die wie oben beschrieben erweitert werden kann. Mit ihr wird ein *Output*-Baustein (Beschriftung *Output*, rechts ein blauer integer-Ausdruck) verbunden, bei dessen Ausdruck nach einem Rechtsklick die Variable *sekunden* ausgewählt wird. Außerdem wird darunter noch ein *Textausgabe*-Baustein (Beschriftung *Output*, Farbe weiß) angefügt, bei dem nach einem Linksklick in der Attributtabelle der Text „Sekunden“ eingegeben werden kann. Durch diese beiden Bausteine wird also bei jedem Schleifendurchlauf der aktuelle Wert der Variablen *sekunden* gefolgt von dem Wort „Sekunden“ ausgegeben und anschließend in eine neue Zeile gewechselt.

Des Weiteren wird im Schleifenrumpf ein *Sleep*-Baustein benötigt, bei dem die Dauer der Programmunterbrechung in der Attributtabelle von 0 in 100 hundertstel Sekunden geändert wird. Am Ende des Schleifenrumpfes muss noch mit einem *Wertzuweisungs*-Baustein (rotes Feld gefolgt von :=) der Variablen *sekun-*

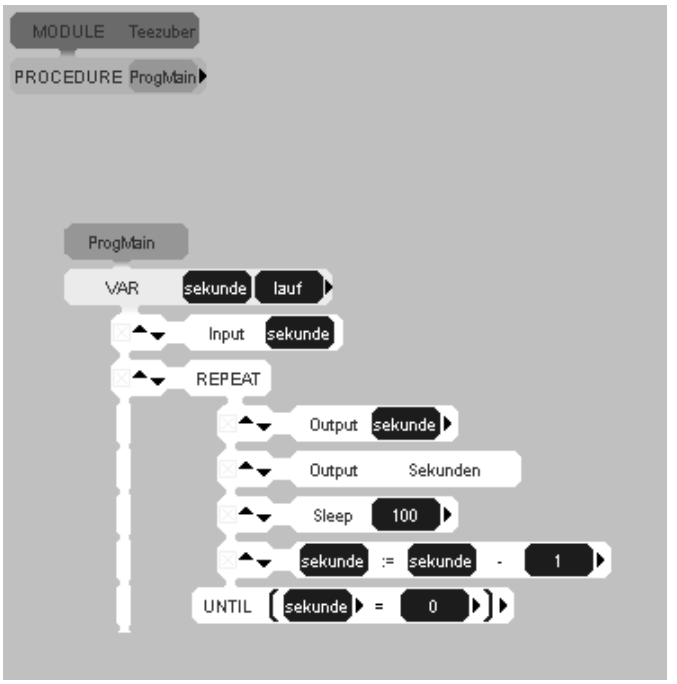


Bild 5:
Die Entwicklung des Programms „Teezubereitung“.

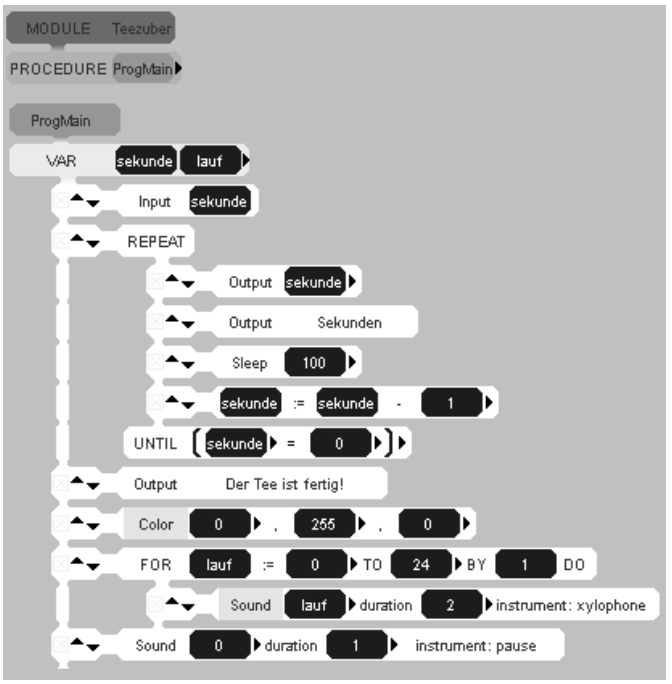


Bild 6: Das fertige Programm „Teezubereitung“.

den der Wert von *sekunden – 1* zugewiesen werden. Hierfür wird zuerst die Variable *sekunden* auf der linken Seite des Wertzuweisungsoperators := ausgewählt. Anschließend wird rechts die Variable *sekunden* gewählt. Nun wird der Ausdruck durch einen Linksklick auf das schwarze Dreieck um einen Operanden erweitert. Nachdem das Rechenzeichen in ein Minus geändert wurde, wird die *0* in der Attributtabelle durch eine *1* ersetzt (siehe Bild 5, vorige Seite).

Nach dem Fertigstellen der Schleife kann das Programm über das Menü *Programm → Programm ausführen* schon einmal getestet werden.

Zur Vervollständigung müssen unter dem *REPEAT-UNTIL*-Baustein noch eine Textausgabe mit dem Text „Der Tee ist fertig!“ und ein *Color*-Baustein angefügt werden. Nach einem Linksklick auf das Wort *Color* kann in der Attributtabelle das Setzen der *Hintergrundfarbe* angeklickt werden. Anschließend werden noch die Rot-, Grün- und Blauwerte der Farbe angegeben. Für die Farbe Grün muss nur beim zweiten Farbwert in der Attributtabelle eine *255* eingegeben werden.

Als letztes soll noch eine Folge von Tönen abgespielt werden. Dies wird mithilfe einer *FOR*-Schleife realisiert. Nachdem der *FOR*-Baustein in den Arbeitsbereich gezogen und mit *ProgMain* verbunden wurde, enthält er ein rotes Feld. Hier muss nach einem Klick mit der rechten Maustaste die integer-Variable *lauf* ausgewählt werden. Rechts der ausgewählten Variable befinden sich noch Startwert, Endwert und Schrittweiteneinstellungen der Schleife. Der Endwert *24* wird in der Attributtabelle eingestellt. Schließlich wird im Schleifenrumpf der *FOR*-Schleife ein *Sound*-Baustein angefügt, bei dem Tonhöhe und Tondauer mit je einem integer-Ausdruck eingestellt werden können. Als Tonhöhe wird die Variable *lauf* ausgewählt. Die Tondauer *2* ist ausreichend. Als Instrument kann nach einem Rechtsklick das *Xylophon* ausgewählt werden.

Nach einem Linksklick auf *Sound* kann in der Attributtabelle zwischen *puffern* und *spielen* gewählt werden. Gepufferte Töne werden in einer Liste gespeichert. Diese Liste mit Tönen wird erst dann abgespielt, wenn ein *Sound*-Baustein abgearbeitet wird, bei dem in der Attributtabelle *spielen* ausgewählt wurde.

Um Pausen zwischen den Tönen zu vermeiden, wird innerhalb der Schleife *puffern* gewählt, nach der Schleife noch ein *Sound*-Baustein eingefügt, bei dem in der Attributtabelle *spielen* gewählt wird (siehe Bild 6).

Die in der Aufgabe gestellten Modifikationsaufträge seien den interessierten Lesern überlassen. Bei der Auswahl zwischen Minuten und Sekunden bietet es sich an, einen *Input*-Baustein mit einer *Boolean*-Variablen zu verwenden.

Eine ausführliche Einführung in das PUCK-System mit fünf Beispielen kann im Übrigen bei Kohl (2006) nachgelesen werden.

Die Bausteine

Wenn Schülerinnen und Schüler den Informatikunterricht in der Sekundarstufe I beginnen, haben sie meist bereits gewisse Vorkenntnisse. Sie haben schon im Internet gesurft, Texte verarbeitet, sich mit diversen Spielen die Zeit vertrieben und oft noch vieles mehr. Damit im Informatikunterricht nicht nur mit Kontrollstrukturen, Zahlen und Textausgaben operiert wird, wurden in PUCK verschiedene Bausteine zur Verfügung gestellt, die die Kreativität und die Motivation der Schüler wecken sollen.

Mithilfe der Bausteine *Dot*, *Line* und *Rect/Ellipse* können Punkte, Linien, Rechtecke und Ellipsen gezeichnet werden. Mit dem *Color*-Baustein werden Vorder- und Hintergrundfarbe eingestellt.

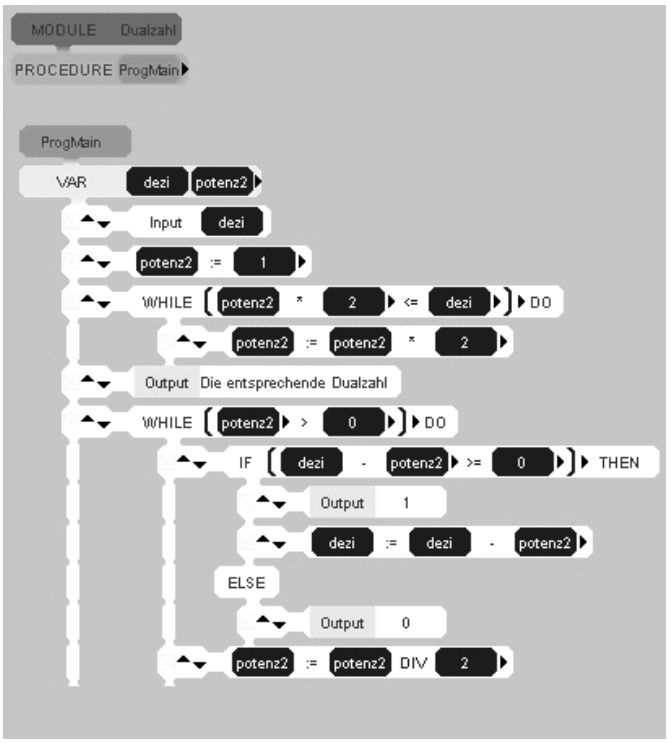
Beim *Sound*-Baustein können die Schüler die Tonhöhe, die Tondauer und ein Instrument wählen. Bei der Tonhöhe wird keine Frequenz angegeben, sondern eine Zahl, die für eine Note bzw. eine Taste des Klaviers steht. So können Notenblätter einfach in Programme übersetzt und dann vom Computer vorgespielt werden.

Mithilfe des *Random*-Bausteins kann einer Integer- oder einer Booleanvariablen ein zufälliger Wert zugewiesen werden. Bei Integervariablen wird die obere Grenze des Bereichs angegeben und eine Zahl zwischen *1* und dieser Obergrenze zufällig gewählt. Mit der Obergrenze *6* kann also einer Variablen eine Würfelzahl zugewiesen werden. Bei Booleanvariablen wird die Wahrscheinlichkeit angegeben, mit der *TRUE* gewählt werden soll. Bei der Wahl *80* wird also mit 80 %iger Wahrscheinlichkeit *TRUE* zugewiesen und mit 20 %iger *FALSE*.

Der *Sleep*-Baustein ist für Verzögerungen zuständig. So können Animationen, bestehend aus einzelnen Grafiken, ausgegeben werden.

Komplexitätsberechnung für Programme

Schülerlösungen zu beurteilen und zu vergleichen, ist oft nicht einfach. Programme mit unterschiedlicher



Anweisungs-komplexität	Ausdrucks-komplexität	Daten-komplexität
1	0	1
1	0	1
3	1,5*(2+1)	2
1,5*1	2	1
1	0	0
3	1,5*1	1
1,5*3	1,5*(1+1)	2
2,25*1	0	0
2,25*1	1	2
2,25*1	0	0
1,5*1	2	1
Summe		
23,25	14	11
Gesamtkomplexität: 48,25		

Bild 7: Komplexitätsberechnung am Beispiel „Umrechnen von Dezimal- in Dualzahlen“.

Länge lösen oft ein und dasselbe Problem auf unterschiedliche Art und Weise. Aber die Lesbarkeit eines Programms kann meist nicht nur an der Länge des Quelltextes festgemacht werden. Im PUCK-System wurde ein von Peter Rechenberg entwickeltes Verfahren zur Messung der Lesbarkeit, Verstehbarkeit und Änderbarkeit von Programmsystemen implementiert (vgl. Rechenberg, 1986). Bei diesem Komplexitätsmaß handelt es sich um die Summe von Anweisungs-, Ausdrucks- und Datenkomplexität eines Programms.

Die Anweisungskomplexität ergibt sich aus der Summe aller im Programm verwendeten Anweisungen, wobei jede Anweisung eine Punktzahl zugeordnet bekommt, die noch mit einem Verschachtelungsfaktor (1,5 pro Schachtelung) multipliziert wird. Einfache Anweisungen (Wertzuweisung, *Sleep*, Textausgabe usw.) bekommen 1 Punkt, Kontrollstrukturen (IF-THEN-ELSE, WHILE, REPEAT, FOR) erhalten 3 Punkte, und Prozeduraufrufe bekommen 1 Punkt plus 1 Punkt je Parameter.

Bei der Ausdruckskomplexität werden die Operatoren in den verwendeten Ausdrücken betrachtet. Die Relationsoperatoren <, ≤, >, ≥, = und ≠ sowie + und – bekommen den Wert 1. DIV, * und NOT bekommen den Wert 2. AND, OR und MOD bekommen den Wert 3. Bei jedem Klammerpaar wird die Ausdruckskomplexität des geklammerten Ausdrucks mit dem Klammerungsfaktor 1,5 multipliziert.

Bei der Datenkomplexität wird das Auftreten von selbstdefinierten Namen für Variablen, Parameter und Prozeduren in Anweisungen gemessen. Das Vorkommen von lokalen Variablen und Parametern in einer Anweisung hat jeweils die Datenkomplexität 1. Der Aufruf von Prozeduren hat die Datenkomplexität 2. Für jede Anweisung werden die Punkte der verwendeten Namen addiert. Wenn Namen innerhalb einer Anweisung mehrmals vorkommen, so fließen sie nur einmal in die Berechnung ein (siehe Bild 7).

Das Beispiel des Umrechnens von Dezimal- in Dualzahlen illustriert die Berechnung der Komplexität eines Programms. Mithilfe dieser vom PUCK-System automatisch berechneten Komplexitäten kann durch Lehrkräfte und Schülerinnen und Schüler eingeschätzt werden, wie kompliziert eine Lösung ist. Somit kann dieses Maß als Ausgangspunkt zur Diskussion über Merkmale von Computerprogrammen genommen werden.

Lehrermodul

PUCK enthält ein Setup-Programm (siehe Bild 2, Seite 42), mit dem das System für die Schülerinnen und Schüler konfiguriert wird. Dort kann vorgegeben werden, welche Bausteine in der visuellen Programmiersprache zur Verfügung stehen. Der in PUCK vorhandene Menüpunkt *Optionen* kann im Setup unsichtbar geschaltet werden. Somit können die Schülerinnen und Schüler an den vorhandenen Einstellungen nichts mehr ändern und wirklich nur die gegebenen Bausteine einsetzen.

Außerdem gibt es noch ein PUCK-Lehrermodul, bei dem ein weiterer Menüpunkt zur Verfügung steht. Unter *Aufgabenkonstruktion* gibt es die Möglichkeit, einen Pseudoquelltext zum aktuellen Programm zu generieren. Dieser kann bei der Konstruktion von Aufgaben oder zur Unterstützung schwächerer Schüler eingesetzt werden (siehe Bild 8, nächste Seite).

Um Lehrerinnen und Lehrern das Entwickeln von Arbeitsblättern zu erleichtern, können Screenshots des aktuellen Arbeitsbereichs erstellt werden, die im Unterverzeichnis *screenshots* fortlaufend nummeriert gespeichert werden.

Außerdem ist es möglich, ein Beispiel des aktuellen Programms zu erzeugen. Dieses Beispiel kann dann – wie jede andere PUCK-Datei – per Doppelklick ausge-

Bild 8: Der von PUCK generierte Pseudoquelltext des Programms „Umrechnen von Dezimal- in Dualzahlen“.

```
PROGRAMM Dualzahlen

  Prozedur ProgMain()
     Lege die Variable dezi vom Typ Integer an.
     Lege die Variable potenz2 vom Typ Integer an.
     Fordere den Benutzer mit "Bitte geben Sie eine Dezimalzahl ein:" auf,
                                  eine  Zahl  einzugeben.
     Speichere die eingegebene Zahl in die Variable dezi.
     Weise der Variablen potenz2 den Wert von 1 zu.
     Solange wie (potenz2*2  <=  dezi) wahr ist, mache Folgendes:
       Weise der Variablen potenz2 den Wert von potenz2*2 zu.
     Gib den Text "Die entsprechende Dualzahl ist: " aus.
     Solange wie (potenz2  >  0) wahr ist, mache Folgendes:
       Wenn die Bedingung (dezi-potenz2  >=  0) wahr ist, mache Folgendes:
         Gib den Text "1" aus.
         Weise der Variablen dezi den Wert von dezi-potenz2 zu.
       Wenn die Bedingung (dezi-potenz2  >=  0) falsch ist, mache Folgendes:
         Gib den Text "0" aus.
       Weise der Variablen potenz2 den Wert von potenz2 DIV 2 zu.
```

führt werden. Ein Öffnen des Beispiels mit PUCK ist aber nicht möglich. Somit kann die Lehrkraft ein funktionierendes Programm vorgeben, das von den Schülern rekonstruiert werden muss.

Logging

In PUCK gibt es eine Logging-Funktion, die die ausgeführten Schritte, die Uhrzeit, die Komplexität, die verwendeten Bausteine und weitere Informationen in der Tabelle log/LogFile.csv einträgt und bei jedem Ausführen das aktuelle Programm im Unterordner log/$Datum$ speichert. Somit sind die Lernfortschritte der Schülerinnen und Schüler für die Lehrkraft nachvollziehbar. Außerdem können so alle Programme, die ausgeführt wurden, wiederhergestellt werden.

Weiterentwicklung

Das PUCK-System wurde mit den Quelltexten unter der GNU General Public License (GPL) bei *SourceForge.net* veröffentlicht.

Eine Erprobung des PUCK-Systems aus der Perspektive der Bildungsstandards Informatik im Sinne eines kompetenzorientierten Informatikunterrichts erfolgt in einem ersten Durchlauf seit August 2006. Informatiklehrerinnen und -lehrer aus Deutschland, Österreich und der Schweiz, die an der Teilnahme am geplanten zweiten Durchlauf im Zeitraum vom August/September 2007 bis Februar 2008 interessiert sind, wenden sich bitte an den erstgenannten Autor. In der Erprobung soll das PUCK-System im Informatikunterricht der Sekundarstufe I (Klassenstufen 8 bis 10) eingesetzt werden. Dafür werden ein Kompetenzmodell und eine Aufgabensammlung bereitgestellt.

Dipl.-Inform. Lutz Kohl
Gabor Meißner
Harald Schmidt
Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Abteilung Didaktik der Mathematik und Informatik
Casio-Stiftungsprofessur
Ernst-Abbe-Platz 2
07743 Jena

E-Mail: lutz.kohl@uni-jena.de
Puck-Homepage: http://www.uni-jena.de/puck.html

Literatur und Internetquellen

Bruner, J. S.: Entwurf einer Unterrichtstheorie. Düsseldorf: Schwann, 1974.

Fothe, M.: Unterricht – bald nur noch mit Computer? In: informatica didactica, 8. Jg. (2006), Ausgabe 7, 12 Seiten.
http://www.informatica-didactica.de/InformaticaDidactica/Fothe2006.pdf
[Stand: April 2007]

GPL – GNU General Public License (Deutsche Übersetzung):
http://www.gnu.de/gpl-ger.html [Stand: April 2007]

JDK – Java SE Downloads:
http://java.sun.com/javase/downloads/index.jsp [Stand: April 2007]

Kohl, L: Puck – eine visuelle Programmiersprache für die Schule. In: Friedrich, St. (Hrsg.): Unterrichtskonzepte für informatische Bildung. INFOS 2005 – 11. GI-Fachtagung Informatik und Schule. Reihe „GI-Edition LNI – Lecture Notes in Informatics“, Band P-60. Bonn: Köllen Verlag, 2005, S. 309–318.

Kohl, L.: Puck – eine visuelle Programmiersprache für die Schule. In: informatica didactica, 8. Jg. (2006), Ausgabe 7, 11 Seiten.
http://www.informatica-didactica.de/InformaticaDidactica/Kohl2006.pdf
[Stand: April 2007]

Meißner G.; Schmidt H.: Einführungsunterricht zur Programmierung mit Puck. Jena: Unveröffentlichtes Manuskript, 2007.

PUCK bei SourceForge.net:
http://sourceforge.net/projects/puck [Stand: April 2007]

PUCK Unterrichtskonzeption:
http://www.uni-jena.de/puck_unterrichtskonzeption.html
[Stand: April 2007]

Rechenberg, P.: Ein neues Maß für die softwaretechnische Komplexität von Programmen. In: Informatik – Forschung und Entwicklung, 11. Jg. (1986), H. 1, S. 26–37.

Schubert, S.; Schwill, A.: Didaktik der Informatik. Heidelberg; Berlin: Spektrum Akademischer Verlag, 2004.