Dr. Kerstin Strecker
Max-Planck Gymnasium
Theaterplatz 10
37073 Göttingen

Wie viel Programmierkompetenz braucht der Mensch?

In den GI-Bildungsstandards[1] für Informatik in der Sekundarstufe I, die 2008 herausgegeben wurden, findet man unter anderem den Bereich „Programmierung“. Sollte Informatik bundesweit ein Pflichtfach werden, dann werden (folgt man den Standards) alle Schülerinnen und Schüler auch im Bereich „Programmierung“ unterrichtet. In einer Klasse sitzen aber zukünftige Mediziner, Juristen, Bankangestellte oder Musiker. Warum sollten diese in Informatik unterrichtet werden und nicht z. B. in derselben Zeit einen Erste-Hilfe-Kurs belegen? Und wenn man sich schon für Informatik entscheidet, warum sollte die zukünftige Ärztin nicht nur im (sofort einsichtigen) Themenbereich Datenschutz unterrichtet werden, sondern auch noch programmieren? Wie kann der Bereich „Programmierung“ einen sinnvollen Beitrag zur Allgemeinbildung für alle unsere Schülerinnen und Schüler an allgemeinbildenden Schulen leisten? Wie viel Programmierkompetenz braucht der Mensch?

Wir werden versuchen in diesem Artikel Antworten auf diese Frage zu geben. Die erste Antwort wird inhaltlich orientiert sein, die zweite eher methodisch, wobei beide so eng miteinander verzahnt sind, dass nur eine gemeinsame Umsetzung sinnvoll erscheint.

1. Was soll denn programmiert werden?

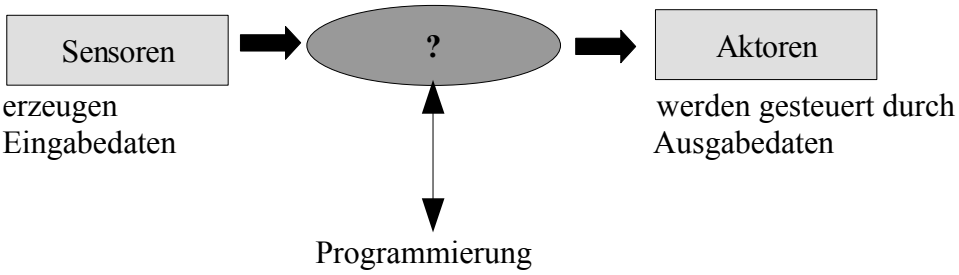
In seinem Werk „Allgemeinbildung und Mathematik“[2] beschreibt Heymann einen Anforderungskatalog an allgemeinbildenden Unterricht. Einer seiner Punkte darin ist der Punkt „Weltorientierung“. Heymann schreibt dazu: „Die Schüler sollen einen Überblick haben, die Erscheinungen um sich herum einzuordnen wissen, sie zueinander in Beziehung setzen können, über ihren engeren Erfahrungshorizont hinaus über die Welt „Bescheid wissen““ (siehe 2). Und Helmut Witten[3] schreibt dazu: „Es ist daher Aufgabe einer Weltorientierung durch informatische Bildung, die Informationstechnik in den alltäglichen Anwendungen sichtbar und verstehbar zu machen“ (siehe 3).

Mit anderen Worten: Informatik und auch eine Einheit “Programmierung“ kann allgemeinbildend sein, wenn sie die Informationstechnik in den alltäglichen Anwendungen sichtbar und verstehbar macht. Was ist aber Informationstechnik im Alltag der Schülerinnen und Schüler? Das sind sicher der eigene MP3-Player oder die technischen Geräte im Haushalt, wie Wäschetrockner, elektrische Zahnbürste oder Videorekorder. Das ist der Fahrscheinautomat im Bahnhof, der Strichcodescanner im Supermarkt, das Blutdruckmessgerät des Großvaters oder die automatische Sortiermaschine in großen Fabrikhallen.

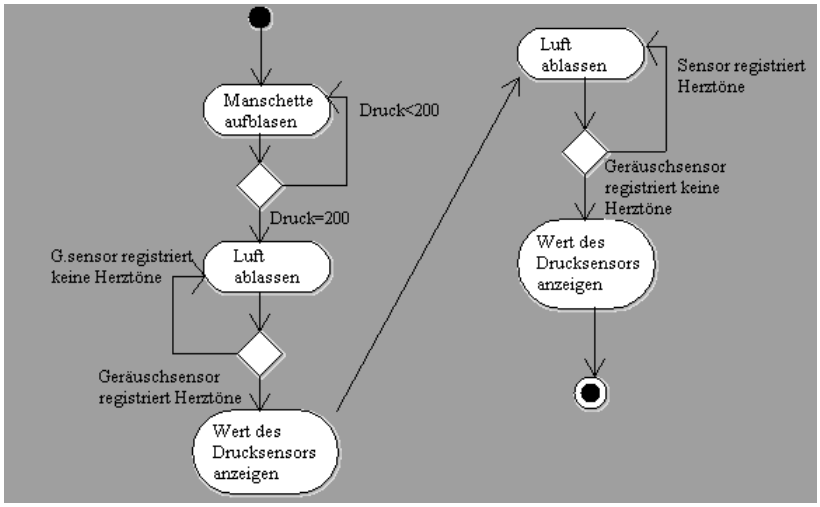
Betrachtet man diese ganzen technischen Systeme genauer, dann stellt man fest, dass sie alle aus einer Reihe von Sensoren und Aktoren bestehen. Der Feuchtesensor im Wäschetrockner liefert Daten über den Zustand der Wäsche, der Lichtsensor am Strichcodescanner liefert einen Hinweis darauf, ob ein Strich schwarz oder weiß ist, ein Geräuschsensor am Blutdruckmessgerät gibt gemeinsam mit einem Drucksensor Aufschluss über den systolischen oder diastolischen Blutdruckwert und ein Taster an der elektrischen Zahnbürste ist für das Ein- bzw. Ausschalten

1 http://www.sn.schule.de/~istandard/docs/bildungsstandards_2008.pdf, Zugriff: 11.7.2011
2 Hans Werner Heymann: „Allgemeinbildung und Mathematik“, Beltz Verlag, 1996
3 Helmut Witten: „Allgemeinbildender Informatikunterricht? Ein neuer Blick auf H.W. Heymanns Aufgaben allgemeinbildender Schulen“, in: Peter Hubwieser (Hrsg.): „Informatische Fachkonzepte im Unterricht“, INFOS 2003, 10. GI-Fachtagung Informatik und Schule, GI-Lecture Notes

derselben verantwortlich. Aktoren, die wir in diesen technischen Systemen finden ist z. B. der Motor, der die elektrische Zahnbürste bewegt oder die Wäsche im Trockner schleudert. Aktoren können außerdem Lampen, Summer, LED-Anzeigen oder auch Gebläse sein.
Damit das Gerät aber so funktioniert, wie es vom Bediener erwartet wird, müssen die Sensoren auf eine bestimmte Art und Weise mit den Aktoren verknüpft werden. Das technische System bestehend aus Sensoren und Aktoren muss so *konfiguriert* werden, dass die gewünschte Funktionalität erreicht wird. Dabei kann bei gleichen Sensoren und Aktoren ein Gerät, je nach Programmierung, den einen oder den anderen Zweck erfüllen.



Ein Blutdruckmessgerät beispielsweise besteht aus einem Geräuschsensor und einem Drucksensor, einem Gebläse für die Manschette, einem Ventil und einer Textanzeige. Die Konfiguration dieses Systems ergibt sich wie folgt:



Wenn wir eine Unterrichtseinheit „Programmierung“ dahingehend beschränken, dass wir den Schülerinnen und Schülern reale externe Sensoren und reale externe Aktoren zur Verfügung stellen und ein Programm eine Verarbeitung der durch die Sensoren erzeugten Eingabedaten ist mit dem Ziel, Ausgabedaten zu erzeugen, die Aktoren ansteuern, dann können wir technische Systeme der Lebenswelt der Schülerinnen und Schüler *rekonstruieren*. Dann hat die Programmierung oder Rekonstruktion den allgemeinbildenden Wert, Schülerinnen und Schüler die Funktionalität technischer Systeme ihrer Lebenswelt transparent und verstehbar zu machen. Dann können wir vielleicht sogar erreichen, dass ein so erzeugtes Verständnis technischer Systeme übertragen werden kann auf solche, die nicht Gegenstand des Unterrichts waren.

Betrachten wir die Programmierung, die Sensoren und Aktoren miteinander verknüpft einmal genauer. In der Realität werden solche Systeme aus dem Bereich „Steuern und Regeln“ oft mit Mealy-Maschinen (Endliche Automaten mit Ausgabe) programmiert. Das Programm muss also endlich viele Zustände annehmen können (bei imperativen Sprachen z. B. modelliert durch mehrere

boolesche Variablen) und in einem bestimmten Zustand bei einer Kombination von Über- oder Unterschreitungen eines oder mehrerer Sensorwerte einen oder mehrere Aktoren ansteuern und/oder den Zustand wechseln. In imperativen Sprachen bedeutet das, dass alle Bedingungen, seien sie im Kopf einer Schleife oder Eingangsbedingung einer Alternative, immer von den Sensoren abhängen, eventuell dem Zustand des Programms. Alle elementaren Anweisungen in einem Programmtext haben dann etwas mit Zustandswechsel oder der Ansteuerung von Aktoren zu tun. Damit vereinfacht sich für die Schülerinnen und Schüler die Programmstruktur enorm. In der obigen Abbildung erkennt man das z. B. daran, dass an den Rauten (Bedingungen) stets Über- bzw. Unterschreitungen von Sensorwerten abgefragt werden, in den Ovalen stets Ansteuerungen von Aktoren stehen.

Ein weiteres wichtiges Prinzip der Informatik wird nebenbei auch im wahrsten Sinne des Wortes greifbar: das EVA-Prinzip. Die Sensoren liefern die Eingaben. Sie stellen physikalisch gesehen eine „black box“ dar, die Verarbeitung liegt in den Händen der Schüler, die Ausgabedaten steuern die Aktoren, ebenfalls physikalisch gesehen „black boxes“. Eingaben und Ausgaben können nicht verwechselt werden, ein Feuchtesensor kann nichts „tun“, sowie ein Motor nichts „fühlen“ kann.

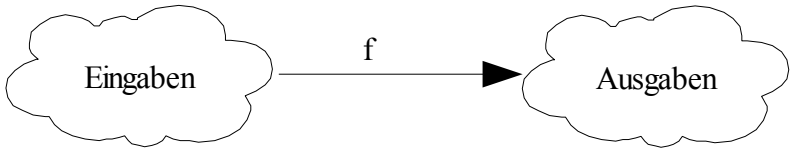
Um bei dem allgemeinbildenden Gedanken zu bleiben, muss die Rekonstruktion realer technischer Systeme mit Sensoren und Aktoren im Vordergrund stehen.
Die Funktionalität des Systems muss aber in einer Programmiersprache beschrieben werden und das von allen Schülerinnen und Schülern. Deshalb sollte bei der Wahl der Sprache darauf geachtet werden, dass das Erlernen der Sprache so einfach ist, dass es das eigentliche Ziel nicht durch z. B. eine unnötig komplexe Syntax überdeckt.
Wir brauchen also eine Sprache, die es den Schülerinnen und Schülern ermöglicht, ihre Zeit in die Rekonstruktion technischer Systeme und damit der Algorithmik zu investieren und nicht in das Erlernen einer Codierungsnotation für solche Algorithmen oder einer langwierigen Syntax-fehlersuche.
Weiterhin brauchen wir nicht notwendigerweise eine Sprache mit objektorientiertem Konzept. Objektorientierte Sprachen sind vor allem dann sinnvoll, wenn Daten gekapselt werden sollen. Hier beschäftigen wir uns mit einfachen technischen Systemen. Die Systeme sind intuitiv erfassbar, Ein- und Ausgaben schnell identifiziert. Eine längere Modellierungsphase, deren Produkte meist in objektorientierten Sprachen aufgegriffen werden, entfällt. Imperative Sprachen reichen für unsere Belange vollkommen aus.

Halten wir also zunächst fest:
Da die Schülerinnen und Schüler, die sich in der Oberstufe oder im späteren Berufsleben *nicht* mit Informatik beschäftigen in der Mehrzahl sind und wir somit Wert auf allgemeinbildende Aspekte in einer Unterrichtseinheit „Programmieren“ legen, sollten technische Systeme mit Sensoren und Aktoren konfiguriert werden mit dem Ziel, die Funktionalität technischer Systeme der Lebenswelt verstehbar zu machen. Dabei sind reale Sensoren und Aktoren notwendig und eine graphische Programmiersprache, die so „einfach“ zu erlernen ist, dass nicht das Erlernen der Sprache das eigentliche Ziel der Konfiguration technischer Systeme überdeckt. Durch welche Lernarrangements dieses im Unterricht umgesetzt werden kann, zeigt Abschnitt 3.

2. Warum denn selbst programmieren? Reicht es nicht aus, Programme lesen zu können?

Entwickeln wir selbst einen Algorithmus, dann haben wir ein klares Ziel vor Augen, welche Funktionalität das fertige Programm haben soll. Wir wollen z. B. dass ein Roboter einer schwarzen Linie folgt oder ein Summer Alarm gibt, wenn ein Einbrecher die Treppe hochgeht. Die Ausgabedaten sind durch die Problem- oder Aufgabenstellung gegeben und bekannt. Meist sind auch die Eingabedaten bekannt. Bei technischen Systemen gibt es in der Aufgabenstellung definierte Sensoren, die die Eingabewerte liefern können. Es bleibt die Frage, wie die Eingabewerte

so auf die Ausgabewerte abgebildet werden können, dass genau das passiert, was der Programmierer möchte. Es geht letztendlich mathematisch gesprochen darum, eine Funktion f zu finden, die die Eingabedaten so auf die Ausgabedaten abbildet, dass die gewünschte Funktionalität erreicht wird[4].



Machen wir einen kleinen Exkurs in die Mathematikdidaktik. In dem Werk „Mathematikunterricht entwickeln“[5] identifizieren die Autoren drei Komponenten der Aufgaben im Mathematikunterricht: den Ausgangszustand, die Transformation und den Zielzustand. In der Aufgabe:
„Gegeben sind zwei Funktionen $f = 3x - 4$ *und* $g = -2x + 6$. *Berechne den Schnittpunkt ihrer Graphen“* bilden die beiden Funktionen f und g den Ausgangszustand. Die Transformation ist die Lösung des zugehörigen linearen Gleichungssystems und das Ziel ist der Schnittpunkt $(2 \mid 2)$.
Nun kann in einer Aufgabe jede Komponente jeweils gegeben oder gesucht sein und so die Mathematikaufgabe klassifizieren.
Folgende Grafik stammt von Johanna Neubrand[6]:

	Art der Aufgabe	Ausgangs-zustand	Bearbeitung, Lösungsweg	Zielzustand
Aufgabenart 1 (verallgemeinerte Bestimmungsaufgaben)	Bestimmungsaufgabe	vorgegeben	gesucht	gesucht
	Umkehraufgabe	gesucht	gesucht	vorgegeben
Aufgabenart 2 (verallgemeinerte Grundaufgaben)	Grundaufgabe	vorgegeben	vorgegeben	gesucht
	Umkehraufgabe	gesucht	vorgegeben	vorgegeben
Aufgabenart 3 (Beweisaufgaben)	Beweisaufgabe	vorgegeben	gesucht	vorgegeben
Aufgabenart 4 (Reflexionsaufgaben)	Aufgabe über Aufgaben	vorgegeben	vorgegeben	vorgegeben
	Selbst eine Aufgabe bilden	gesucht	vorgegeben	gesucht

Vergleichen wir diese Klassifikation mit unserer Eingangsüberlegung zum Programmieren, dann gehören Programmieraufgaben zur Aufgabenart 3. Denn wie weiter oben beschrieben, sind Ausgangs- und Zielzustand bekannt und der Lösungsweg von den Eingabedaten zu den Ausgabedaten muss vom Programmierer gefunden werden.

Betrachtet man jetzt die verschiedenen Aufgabenarten, die tatsächlich im Mathematikunterricht vorkommen, so ergibt sich nach Neubrand (siehe 6) folgendes Bild:

4 Da in der Schule keine nichtdeterministischen Programme auftauchen, ist eine Funktion (und keine Relation) ausreichend.
5 Regina Bruder, Timo Leuders, Andreas Büchter: „Mathematikunterricht entwickeln. Bausteine für kompetenzorientiertes Unterrichten“, Cornelsen-Verlag, 2008
6 Johanna Neubrand: „Eine Klassifikation mathematischer Aufgaben zur Analyse von Unterrichtssituationen“, Verlag Franzbecker, 2002

N = 833	USA	Deutschland	Japan
Arithmetik			
verallgemeinerte Bestimmungsaufgaben	34 (42 %)	19 (66 %)	
verallgemeinerte Grundaufgaben	46 (57 %)	10 (34 %)	
Beweisaufgaben			
Reflexionsaufgaben	1 (1 %)		
gesamt (Arithmetik)	81 (100 %)	29 (100 %)	
Algebra			
verallgemeinerte Bestimmungsaufgaben	147 (80 %)	115 (93 %)	33 (69 %)
verallgemeinerte Grundaufgaben	36 (20 %)	6 (5 %)	13 (27 %)
Beweisaufgaben		1 (1 %)	
Reflexionsaufgaben		1 (1 %)	2 (4 %)
gesamt (Algebra)	183 (100 %)	123 (100 %)	48 (100 %)
Geometrie			
verallgemeinerte Bestimmungsaufgaben	146 (74 %)	80 (70 %)	25 (43 %)
verallgemeinerte Grundaufgaben	48 (24 %)	30 (26 %)	2 (3 %)
Beweisaufgaben		1 (1 %)	23 (40 %)
Reflexionsaufgaben	2 (1 %)	4 (4 %)	8 (14 %)
gesamt (Geometrie)	196 (100 %)	115 (100 %)	58 (100 %)
gesamt	460	267	106

Die Aufgabenart, die wir eben mit den klassischen Programmieraufgaben in Beziehung gesetzt haben, nämlich die hier genannten „Beweisaufgaben“, sind im herkömmlichen Mathematikunterricht kaum zu finden.

Wenn wir also von den Schülerinnen und Schülern fordern, dass sie *selbstständig* Algorithmen suchen, sich Lösungswege also selbst erschließen, dann fördern und fordern wir Kompetenzen von den Kindern, die im Mathematikunterricht u. a. wenig gefordert und gefördert werden.
Lassen wir die Schülerinnen und Schüler selbstständig Algorithmen entwickeln, dann erweitern wir ihre Kompetenzen um einem Aspekt, der in anderem Unterricht unterrepräsentiert ist.

Dann kann die eigenständige Algorithmensuche vielleicht auch dahingehend allgemeinbildend sein, dass hier Problemlösekompetenzen geschult werden, die anderswo wenig Beachtung finden und eine entscheidende Lücke geschlossen werden kann.
(Die Zahlen stammen aus dem Jahr 2002. Unter anderem durch die sogenannten „offenen Aufgaben“ hat sich allerdings auch der Mathematikunterricht weiter entwickelt. Das sollte an dieser Stelle nicht unerwähnt bleiben)

Beweisaufgaben kommen aber auch deshalb im Mathematikunterricht kaum vor, weil sie für Schülerinnen und Schüler so „schwierig“ sind. Nicht anders verhält es sich mit dem Programmieren. Wer schon einmal in der Sekundarstufe I eine Einheit „Programmierung“ so unterrichtet hat, dass die Schülerinnen und Schüler die Algorithmen eigenständig entwickeln mussten und nicht nur fertige Algorithmen abgetippt haben, der weiß, dass der dazu notwendige analytische Lösungszugang nicht allen Schülerinnen und Schülern leicht fällt. Durch die richtige Wahl der Lernumgebung ist es aber dennoch befriedigend möglich.
Im oberen Abschnitt haben wir bereits graphische Programmiersprachen erwähnt. An dieser Stelle fügen wir noch einen weiteren Aspekt hinzu:
Wenn nicht allen Schülerinnen und Schülern der analytische Zugang so leicht fällt, dann sollten wir einen zweiten Lösungszugang bieten. Wählt man Programmiersprachen, wie z. B. scratch, die einen Werkzeugkasten bieten, aus denen die Schülerinnen und Schüler sich die elementaren Anweisungen oder Kontrollstrukturen beliebig „zusammenklicken“ können, und führt weiterhin jede beliebige Kombination dieser Befehle zu einem ausführbaren lauffähigen Programm, dann können die Schülerinnen und Schüler ein Stück weit experimentell vorgehen. Sie können Befehle zusammensetzen, beobachten was passiert und wie sich das ausführbare Programm verhält. Dann können sie aus ihren Beobachtungen Rückschlüsse auf die Semantik ihres Programms ziehen und sich so Konstrukte erarbeiten.
Wenn wir also Sprachen verwenden, die den Schülern nur die Erzeugung syntaktisch korrekter

lauffähiger Programme erlauben, dann haben wir neben dem gewünschten analytischen Lösungszugang noch einen weiteren Lösungszugang. Durch den Werkzeugkasten kann Schülerinnen und Schülern die Entscheidung für einen Befehl oder eine Kontrollstruktur auch dadurch vereinfacht werden, dass sie die Befehle gegeneinander abwägen und einen Befehl wählen, weil sie andere ausschließen. Werden von jedem Schüler immer lauffähige Produkte erzeugt, dann beinhalten diese möglicherweise noch logische Fehler. Aber trotzdem „passiert" etwas, das System „funktioniert", wenn auch nicht zu 100%. Produktstolz unabhängig vom Urteil des Lehrers kann in diesem Zusammenhang nicht hoch genug bewertet werden.

Wie im ersten Kapitel gezeigt, vereinfacht sich die Programmstruktur bei der Programmierung technischer Systeme dadurch, dass Bedingungen mit der Über-/Unterschreitung von Sensorwerten und Anweisungen mit der Ansteuerung von Aktoren zusammenhängen. Zusammen mit graphischen Sprachen, die einen experimentellen Lösungszugang ermöglichen, wird die eigenständige Algorithmensuche überhaupt erst für alle Schülerinnen und Schüler *möglich*.

3. Umsetzungsmöglichkeiten:

Eckpunkte der oben beschriebenen Kapitel sind folgende:

- Schülerinnen und Schüler sollen technische Systeme mit realen Sensoren und Aktoren konfigurieren, um die Funktionsweise technischer Systeme der Lebenswelt zu durchdringen, womit ein Beitrag zur Allgemeinbildung geliefert wird.
- Schülerinnen und Schüler sollen die Algorithmen selbstständig entwickeln, um die dafür notwendigen Problemlösekompetenzen zu schulen, die im bisherigen Unterricht unterrepräsentiert sind. Dafür muss eine geeignete Sprache zur Verfügung stehen, die insbesondere Syntaxfehler ausschließt und einen zusätzlichen experimentellen Lösungszugang ermöglicht.

Umsetzbar ist dieses Konzept auf vielfältige Art und Weise. Es folgen einige kurz skizzierte Unterrichtsbeispiele, bei denen das beschriebene Konzept durch eine **Automatisierung „realer" Miniwelten** umgesetzt wurde.

Beispiel 1:

scratch und Lego-WeDo

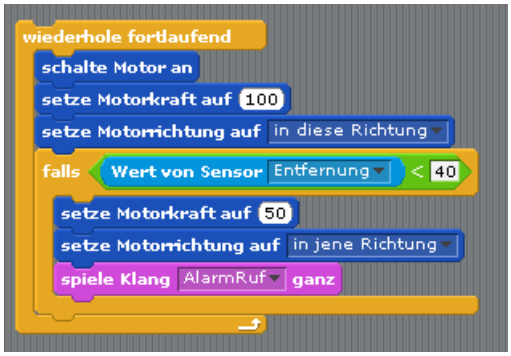
Die Programmierumgebung scratch[7] erfüllt alle Anforderungen an eine Programmierumgebung wie sie oben genannt wurden. Ohne zusätzliche Installation von Treibern können die Sensoren und der Motor von Lego-WeDo[8] angeschlossen werden.

Das System enthält einen Neigungssensor, einen Entfernungssensor und einen Motor. Hiermit lassen sich kleine Lego-Miniwelten automatisieren. Beispielsweise haben Grundschüler in einer

7 http://scratch.mit.edu/, Zugriff: 15.7.11

8 http://education.lego.com/en-gb/products/wedo/9580/, Zugriff: 15.7.2011

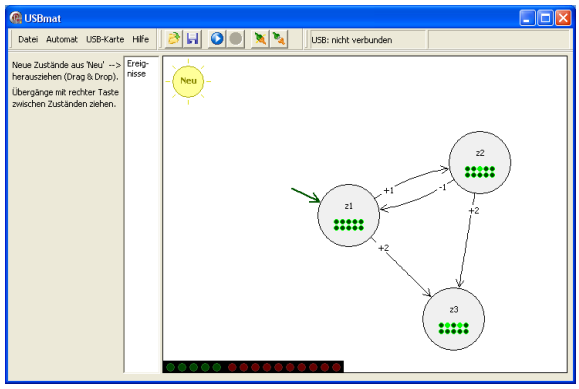
Legostadt mit Straßen, Häusern und einer Eisenbahn Schranken eingebaut. Wenn sich ein Zug nähert (Entfernungssensor), schließt der Motor die Schranke, usw. Ein weiteres Beispiel zur Automatisierung war (in einer durchgeführten Unterrichtseinheit in Klasse 5) die Ausstattung eines Forschungsschiffs zur Nordwestpassage. Bei starkem Seegang (Neigungssensor) gibt es Alarm und die Motoren werden gestoppt. Nähert sich ein Eisberg (Entfernungssensor) gibt es ebenfalls Alarm und die Motorrichtung wird umgekehrt, damit das Schiff nicht mit dem Eisberg kollidiert:



Beispiel 2:
vellemann-Board und USBomat

Das vellemann-Board[9] hat analoge und digitale Ein- sowie Ausgänge. Es kann mittels USB an den Rechner angeschlossen werden, die Installation einer dll ist notwendig. Sensoren und Aktoren können m.E. angeschlossen werden, jedoch ist eine weitere externe Spannungsversorgung in manchen Fällen notwendig. Programmiert werden kann das Ganze z. B. mit dem USBomat von H.Becker[10].

Hierbei handelt es sich um die Programmierung mit Moore-Maschinen. Die Oberfläche erfüllt alle Kriterien einer Sprache, wie sie im obigen Konzept gefordert wurde. In den Zuständen kann angegeben werden, welche Ausgänge im jeweiligen Zustand angesteuert werden sollen und an den Übergängen wird angegeben, auf welche Eingänge (Nummer des Eingangs, geschlossen oder offen) der Automat reagieren soll.

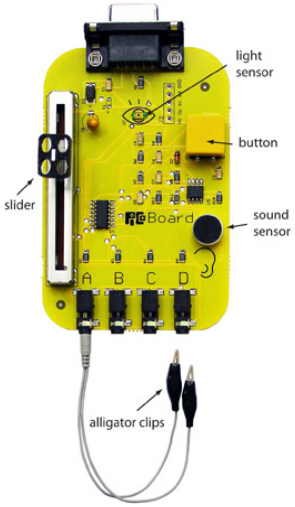


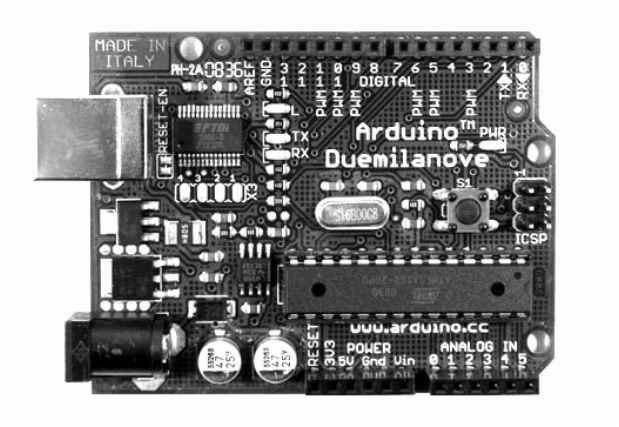
9 http://www.velleman.eu/distributor/products/view/?country=be&lang=en&id=351346, Zugriff: 15.7.11
10 http://hbecker.sytes.net/usbomat/, Zugriff: 15.7.11

Beispiel 3:

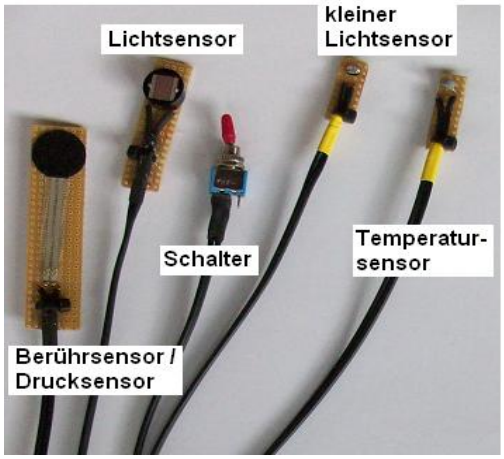
Picoboard, S4A und Arduino

Das Picoboard[11] hat einige integrierte Sensoren und ermöglicht es, vier weitere Sensoren anzuschließen. Physikalische Kenntnisse sind nicht notwendig. Mit einem zu installierenden Treiber kann es unter scratch verwendet werden. Hier kann man es übrigens auch sinnvoll in Kombination mit Lego-WeDo verwenden. Das Picoboard kann jedoch im Zusammenhang mit Automatisierungen nur als Eingabeboard verwendet werden. Als Ausgabeboard kann der Arduino[12] dienen, der unter einer scratch-Variante (S4A[13]) ebenfalls mit einer Sprache angesteuert werden kann, die unsere Anforderungen erfüllt.





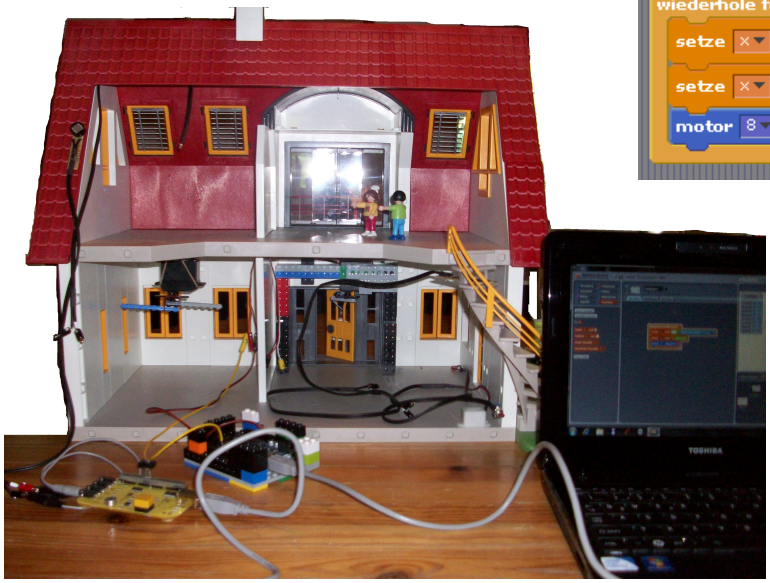
Eine Unterrichtssequenz, die mit dieser Technik durchgeführt wurde ist das PUMA-Projekt.
PUMA steht für **Pu**ppenhaus-Hausauto**ma**tion. Wir haben im Unterricht ein Puppenhaus der Firma playmobil verwendet und mit handelsüblichen Sensoren (Lichtsensor, Schalter, Drucksensor, Temperatursensor) und Aktoren (Motor, Summer, Glühlampe) ausgestattet.

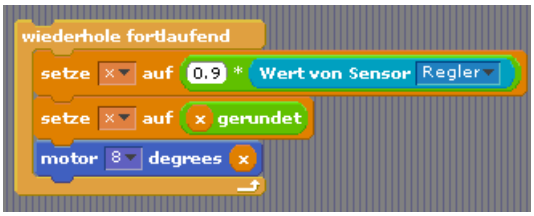


Die Schülerinnen und Schüler haben jeweils zu viert ein playmobil-Haus automatisiert. Eine Tür, die sich nach dem Klingeln öffnet und nach einiger Zeit, wenn die Bewohner im Haus sind, wieder automatisch schließt. Einen Deckenventilator, der an geht, wenn es zu heiß wird. Eine Markise, die man automatisch rein- und ausfahren kann. Lampen im Haus, die angehen, wenn es draußen zu dunkel wird, oder eine Alarmanlage, die ein akustisches Signal gibt, wenn ein Dieb die Treppe hoch geht, allerdings nicht, wenn dies ein Bewohner tut, der die Alarmanlage an- und ausschalten kann. Das hier angegebene Programm steuert z. B. einen Motor, der die Tür bewegen (auf- und zumachen) kann. Das geschieht hier mit dem Picoboard-Regler.

---

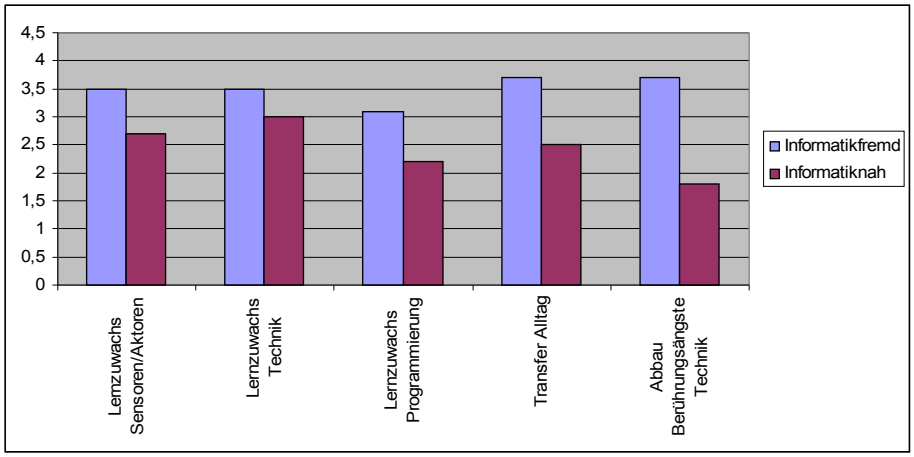
11 http://www.picocricket.com/picoboard.html, Zugriff: 15.7.2011
12 http://arduino.cc/en/Main/Hardware, Zugriff: 6.7.11
13 http://seaside.citilab.eu/scratch/arduino, Zugriff: 6.7.11



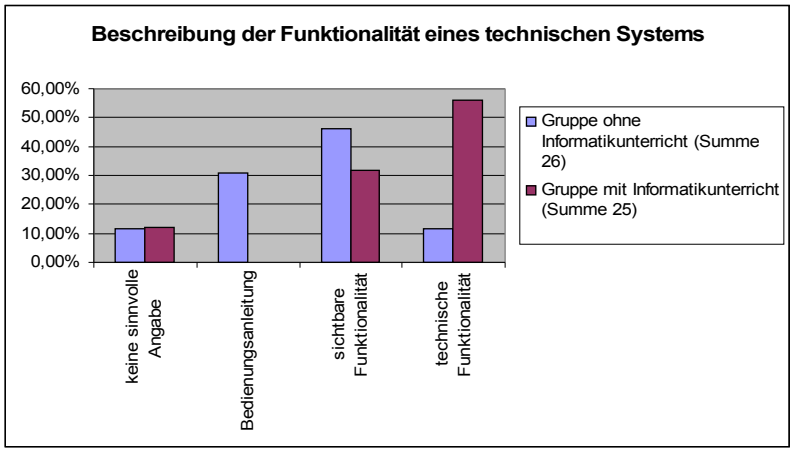


4. Evaluation:

Nach der Einheit PuMa wurden die 29 Schülerinnen und Schüler der Klasse angehalten, in einem anonymen Fragebogen ihren Lernzuwachs zu notieren. Die Gruppe an Schülern, die als Lieblingsfächer in der Befragung Mathematik und Physik angegeben hat, wurde als „informatiknah“ bezeichnet. Schülerinnen und Schüler mit den Lieblingsfächern Deutsch, Kunst, Sport, Religion,... als „informatikfremd“. Die „informatiknahen“ Schüler, die sich schon immer, auch in ihrer Freizeit, mit Technik beschäftigt haben, sind der Meinung, nicht ganz so viel Neues dazugelernt zu haben. Alle anderen jedoch sehen einen deutlichen Lernzuwachs in allen Bereichen. Da nur 29 Schülerinnen und Schüler befragt wurden, hat diese Grafik keinerlei Aussagekraft. Trotzdem ist es vielleicht ein Hinweis, dass mit dem oben beschriebenen Konzept *alle* Schülerinnen und Schüler, auch und gerade solche, die beruflich nichts mit Informatik zu tun haben werden, erreicht wurden.



Ob nun der Zweck, einen Beitrag zur Allgemeinbildung zu leisten, erfüllt wurde, kann ebenfalls nur im Ansatz beantwortet werden.
Die Gruppe wurde nach dem PuMa-Projekt aufgefordert, schriftlich die Funktionsweise eines Strichcodescanners zu erläutern. Dieser wurde natürlich nicht zuvor im Unterricht behandelt. Eine Vergleichsgruppe ohne Informatikunterricht hat dies ebenfalls getan. Die freie Antwort wurde in Kategorien zusammengefasst. „Sichtbare Funktionalität“ meint, dass beispielsweise beschrieben wurde, dass ein Taster die Aufnahme der weißen und schwarzen Striche auslöst und vom Computer an der Kasse Produkt und Preis dieser Codierung zugeordnet werden.



„Technische Funktionalität“ meint, dass die Schülerinnen und Schüler sich beispielsweise detaillierter darüber ausgelassen haben, dass im Inneren des Scanners genau so viele Lichtsensoren sitzen müssen, wie Striche auf dem Code sind. Sie haben beschrieben, dass die Werte der einzelnen Lichtsensoren so ausgewertet werden, dass weiß und schwarz erkannt werden durch einen Vergleich mit bestimmten Schwellwerten. Diese Codierung der schwarz-weiß-Abfolge muss eindeutig sein und durch mehrere Fallunterscheidungen eindeutig einem Produkt zugeordnet werden können. Produkt und Codefolge müssen vorher in den Programmtext eingegeben worden sein.

Auch wenn wiederum keine statistischen Aussagen möglich sind, kann diese Auswertung ein Indikator sein, dass die Schülerinnen und Schüler dieses Kurses in der Lage sind, ihr erworbenes Wissen im Rahmen der Automatisierung einer Miniwelt auf technische Systeme der Umwelt zu übertragen, die nicht Gegenstand des Unterrichts waren.

5. Fazit:

Die Frage: „Wie viel Programmierkompetenz braucht ein Mensch?“ wird hier gleichgesetzt mit der

Frage, wie man eine Unterrichtseinheit „Programmierung“ unter dem Aspekt der Allgemeinbildung sinnvoll für alle Schülerinnen und Schüler gestalten kann. Inhaltlich sollten technische Systeme der Lebenswelt im Vordergrund stehen. Die Verknüpfung von Sensorwerten und Programmzuständen zu einer definierten Ansteuerung von Aktoren ermöglicht die Rekonstruktion technischer Systeme. Im Unterricht kann dies insbesondere durch eine Automatisierung von (Puppen-)häusern, Legostädten, Schiffen, Lego-Bauernhöfen oder anderen Miniwelten erreicht werden.
Auch methodisch kann eine Einheit „Programmierung“ eine Lücke schließen, da bei einer eigenständigen Algorithmensuche Kompetenzen der Schülerinnen und Schüler gefordert und gefördert werden, die in anderem Unterricht unterrepräsentiert sind.
Allerdings kann, nach Meinung der Autorin, ein Unterricht im Programmieren für alle Schülerinnen und Schüler nur dann erfolgreich sein, wenn bei der Wahl der Programmierumgebung bestimmte Kriterien berücksichtigt werden. Die Sprache sollte Syntaxfehler ausschließen und ermöglichen, Algorithmen nach dem Baukastenprinzip aus intuitiv verständlichen Konstrukten zusammen zu setzen.