INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Anreize zum Programmieren Lernen

Ideen und Ansätze für den motivierten Einstieg

David Li

Bachelorarbeit / Zulassungsarbeit

Betreuer Bachelorarbeit	Prof. Dr. François Bry
Betreuer Zulassungsarbeit	Christoph Krichenbauer
Abgabe Bachelorarbeit	9. März 2021

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 9. März 2021

.....
David Li

Zusammenfassung

Die Programmierung bietet durch ihre Vielzahl an Themen und aktuellen Entwicklungen grundsätzlich ein großes Begeisterungspotenzial. Leider finden sich aber in der Literatur immer wieder Berichte, bei denen der Einstieg in die Programmierung nicht gelingt und z.B. mit hohen Abbruchquoten verbunden ist. Das zugrundeliegende Problem ist meist ein Mangel an Motivation. Für den gelungenen Einstieg ist es daher wichtig, sich - neben der Vermittlung inhaltlicher Themen - auf die Motivation der Schüler:innen zu fokusieren.

Dafür gibt es bereits eine Vielzahl von Ideen und Ansätze. Allerdings existiert in der Literatur im Moment keine aktuelle Übersicht oder Zusammenfassung davon. Diese Arbeit hat daher das Ziel, eine aktuelle und umfassende Übersicht über verschiedene Anreize zum Programmieren Lernen zu geben. Die Darstellung ist aufgeteilt in drei große Kategorien: *Interesse wecken*, *Komplexität reduzieren* und *Unterstützung anbieten*. Innerhalb der Kategorien werden die Ideen und Ansätze nach Themen sortiert vorgestellt, wie z.B. virtuelle Welten, Mikrocontroller oder blockbasierte Programmierung. Es werden gängige Lernumgebungen wie Scratch oder BlueJ und ebenso einige neuere Entwicklungen behandelt.

Abstract

In principle, programming offers great potential for enthusiasm due to its wide range of topics and current developments. Unfortunately, however, there are repeated reports in the literature of students who do not succeed in getting started in programming (e.g. in courses with high drop-out rates). The underlying problem is usually a lack of motivation. For a successful start, it is therefore important to focus on the motivation of the students in addition to teaching them about the content.

There are already many ideas and approaches. However, there is currently no up-to-date overview or summary of these in the literature. This paper therefore aims to provide an up-to-date and comprehensive overview of different incentives for learning programming. The presentation is divided into three broad categories: *Arouse Interest*, *Reduce Complexity* and *Provide Support*. Within the categories, ideas and approaches are presented by topic, such as virtual worlds, microcontrollers or block-based programming. Common learning environments such as Scratch or BlueJ are covered, as well as some more recent developments.

Danksagung

An dieser Stelle ein sehr herzliches Dankeschön an Prof. François Bry: Offen, freundlich, und immer mit einer Idee, wie man den Text ein kleines bisschen besser schreiben kann. Seine Betreuung und Rückmeldungen haben die Arbeit in dieser Form erst möglich gemacht.

Vielen Dank auch an Christoph Krichenbauer für die Betreuung als Zulassungsarbeit und die Unterstützung bei der Ideenfindung.

Inhaltsverzeichnis

KAPITEL 1

Einleitung: Ein Funke, der das Feuer entfacht

Jedes Feuer beginnt mit einem kleinen Funken. So klein der Funke auch ist, er ist entscheidend dafür, dass das Feuer brennen kann. Er setzt damit eine Reaktion in Gange, die sich schließlich selbst erhält und nach außen hin Wärme abgibt.

Wenn man beobachtet, wie Menschen lernen, dann bekommt man manchmal den Eindruck, dass es dort ähnlich funktioniert. Es gibt Funken, die das Interesse wecken und Menschen dazu motivieren, sich mit einem Thema intensiv auseinanderzusetzen. Vielleicht ist es eine inspirierende Person oder ein Video, das eine faszinierende Idee vorstellt. Nachdem dieser erste Anreiz stattgefunden hat, findet man innerhalb des Themas immer mehr interessante Aspekte - man bleibt dran und kann schließlich selbst auf dem Gebiet einen Beitrag leisten.

Als Lehrerin[1] ist das natürlich der große Traum, dass so etwas gelingt - dass man Schülerinnen dazu anreizen kann, sich in ein Thema einzuarbeiten und sie dort dann selbstständig immer weiter ihr Interesse entwickeln können. Um dieses Ziel zu erreichen, braucht es als Lehrerin einiges an Vorbereitung und persönlichem Engagement. Eine wichtige Voraussetzung ist natürlich, dass man sich selbst für das Thema interessiert. Darüber hinaus sollte man auch daran Spaß haben, seine eigene Begeisterung an andere Menschen weiterzugeben. In dem Fall, wo das nicht schon auf diese Art und Weise gelingt, werden im Kapitel 2 einige

[1] Um den Lesefluss nicht zu beeinträchtigen wird hier und im folgenden Text zwar nur die weibliche Form genannt, stets aber die männliche und andere Formen gleichermaßen mitgemeint.

Grundlagen zur Motivationstheorie vorgestellt, die Tipps für den Einstieg geben können. Anschließend sollen diese Theorien auf das Thema der Programmierung angewandt werden. Das Kapitel 3 enthält dann eine Sammlung von ganz konkreten Ideen und Ansätzen.

Mittlerweile haben wir mithilfe von Software viel mehr Möglichkeiten, den Unterricht interessant und reizvoll zu gestalten. Auch im Bereich der Programmierung gibt es eine große Fülle an Software und Tools, die für diesen Zweck eingesetzt werden können. Es soll also nicht verwundern, wenn sich viele der Ideen und Ansätze aus dem Kapitel 3 in einer oder anderen Form der Software oder anderer Technologie bedienen. Im Kapitel 4 werden die Ergebnisse noch einmal reflektieren und die wesentlichen Vorteile zusammenfassen. Im Kapitel 5 wird schließlich das Thema der Programmierung verlassen und der Blick auf weitere Horizonte gelenkt.

Die vorliegende Arbeit wird in inhaltlich identischer Form sowohl als Bachelorarbeit als auch als Zulassungsarbeit eingereicht.

KAPITEL 2

Grundlagen

2.1 Motivationstheorien

Wenn man vor der Aufgabe steht, einen Unterricht zu einem Thema zu gestalten, konzentriert man sich häufig zuerst auf die inhaltlichen Aspekte: Welche Themen sollen vermittelt werden? Welche Erklärungen werden benötigt? Welche Aufgaben sollen gestellt werden? Wird allerdings der kognitive Bereich zu sehr fokussiert, dann kann es schnell passieren, dass der Unterricht schließlich scheitert: Die Schülerinnen zeigen wenig Bereitschaft, sich zu engagieren und der erwartete Erfolg bleibt aus. Was hier fehlt ist die Beachtung von psychologischen Aspekten. Man darf nicht vergessen, dass Unterricht stets auch eine Kommunikation mit Menschen ist. Diese Kommunikation soll auch die besonderen Eigenschaften von Menschen, wie eben die Motivation, berücksichtigen.

Motivation wird durch die Arbeit hindurch eine große Rolle spielen. In diesem Abschnitt wird das Thema aus einer theoretischen Sicht beleuchtet und es werden die verschiedenen Aspekte aufgezeigt, die bei der Motivation mitwirken. Dazu soll zu Beginn eine Definition von Motivation vorgestellt werden. Aus der Motivation leiten sich dann viele Teilaspekte ab. Einige ausgewählte Aspekte sollen davon nochmal näher besprochen werden. Dann wird schließlich ein wichtiges Modell zur Motivation vorgestellt. Es können leider nicht alle Aspekte im Detail besprochen werden, die angegebene Literatur kann als Startpunkt für eine

vertiefte Beschäftigung dienen.

Eine einfache und für unsere Anwendung ausreichende Definition von Motivation findet sich in der englischen Wikipedia (Wikipedia, 2021c):

> Motivation is a reason for actions, willingness, and goals.

Die Definition besagt also, dass Menschen zum Handeln bereit sind und auf Ziele hinarbeiten. Der Grund für diese Aktivität bezeichnen wir als Motivation. Um die Motivation zu untersuchen gibt es mehrere Ebenen, die wir betrachten müssen: Wir müssen uns anschauen, was die Menschen tun und auf welches Ziel dieses Tun ausgerichtet ist und schließlich ermitteln, aus welchen Gründen die Menschen handeln.

Wenn man nun versucht, die einzelnen Teile der Definition zu konkretisieren und mit Inhalten zu füllen, dann fällt schnell auf, dass man sich auf ganz unterschiedliche Aspekte fokussieren kann. Der Begriff der Motivation ist daher mit

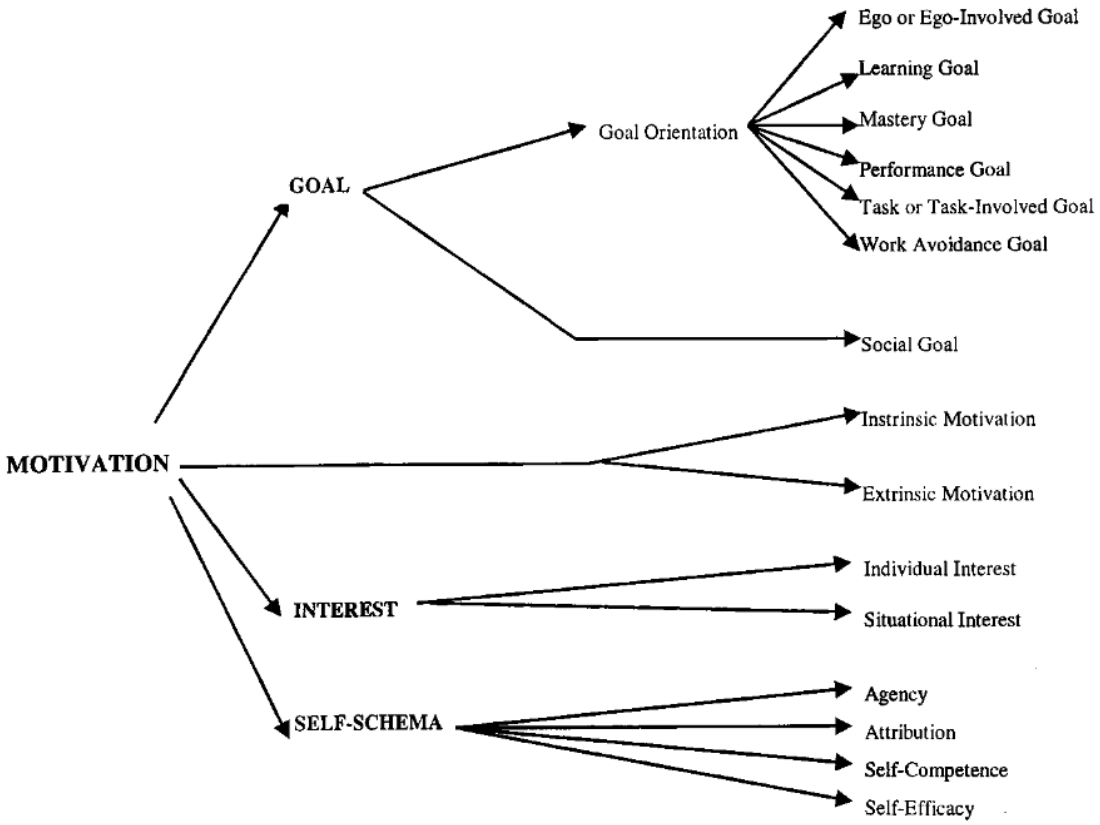


Abbildung 2.1: Übersicht verschiedener Aspekte von Motivation, Quelle: Murphy und Alexander (2000)

einer großen Vielfalt verbunden. Diese Begriffsvielfalt haben auch Murphy und Alexander (2000) festgestellt. In Abbildung 2.1 ist zu sehen, welche Teilaspekte die Autoren in der wissenschaftlichen Literatur wiedergefunden haben.

Wenn man sich für das Thema Motivation interessiert, dann gibt es sehr viele Facetten, mit denen man sich beschäftigen kann. Eine solche umfassende Beschäftigung ist sehr hilfreich für die Planung von Unterricht. Denn dadurch kann der Unterricht viel besser auf die psychologischen Bedürfnissen der Schülerinnen anpassen werden, was zum Beispiel der Langweile entgegenwirkt. Für den Anfang reicht es aber, wenn wir uns mit einigen grundlegenden Begriffen und Konzepten vertraut machen.

Betrachtet man die Quelle von Motivation, dann findet man häufig die Unterscheidung zwischen *intrinsischer* und *extrinsischer* Motivation (Hennessey u. a., 2015). Intrinsische Motivation ist dabei die Motivation, etwas aus der Sache an sich heraus zu tun. Eine Schülerin, die aus intrinsischer Motivation für eine Klassenarbeit lernt, tut das, weil ihr das Lernen Spaß macht und sie das Thema interessiert - und nicht, weil sie auf eine gute Note hofft. Falls man seine Motivation darin findet, ein Ziel außerhalb der Sache zu erreichen, wie z.B. eine Belohnung, dann spricht man von extrinsischer Motivation.

Wenn man die intrinsische Motivation zu einem Maximum steigert, dann wird ein Zustand erreicht, der üblicherweise als *flow* bezeichnet wird (Csikszentmihalyi, 1997). Durch die hohe Vertiefung und Leistungsfähigkeit dieses Zustands wird ihm eine große Bedeutung zugesprochen, allerdings sind auch die Voraussetzungen ziemlich hoch, in den flow zu kommen. Manchmal findet man auch die Ansicht, dass extrinsische Motivation einen negativen Einfluss hat. Das liegt darin, dass es Untersuchungen gibt die zeigen, dass extrinsische Motivation die intrinsische Motivation untergraben kann (Frey und Oberholzer-Gee, 1997). Dieses Phänomen tritt auf, wenn Menschen anfangs etwas freiwillig tun, aber durch einen äußeren Anreiz diese Freiwilligkeit verlieren. In der späteren Arbeit von Frey u. a. (2012) relativiert er aber seine Beobachtung und zeigt, dass dieser Effekt in beide Richtungen auftreten kann: Extrinsische Motivation kann intrinsische Motivation vermindern - oder fördern. Es ist also hilfreich, den gesamten Prozess im Blick zu behalten und auf beide Arten von Motivation zu achten.

Man kann seinen Blick auch darauf richten, wie Menschen ihren Erfolg oder Misserfolg zu erklären versuchen. Dieser Vorgang wird als Attribution bezeichnet, eine Einführung findet sich zum Beispiel in Malle (2011). Bei der Attribution wird häufig zwischen internalen und externalen bzw. stabilen und variablen

Faktoren unterschieden. Die Art der Attribution hat einen Einfluss auf die Motivation (Harvey und Martinko, 2009). Zum Beispiel ist es sehr ungünstig für die Motivation, wenn eine Schülerin ihren Misserfolg auf einen stabilen internalen Faktoren attribuiert. In diesem Fall sieht sich die Person als Versager. Eine andere Form der Attribution kann den Selbstwert besonders schützen: Dabei wird der Misserfolg einem externen, variablen Faktor zugeschrieben Eine schlechte Note wird also durch die zufällig ungünstigen Umstände erklärt, wie z.B. die Prüfung war unerwartet schwer.

In H. Heckhausen (1984) werden diese zwei Muster der Attribution als „deprimierend“ und „positiv“ beschrieben. Bei der positiven Attribution werden zusätzlich Erfolge auf einem stabilen, inneren Faktor zugeschrieben, wie die eigene Begabung. Das steigert nochmals den Selbstwert. Das Thema Attribution ist reichhaltig. Als Lehrerin hat man durch die Art des Unterrichts erheblichen Einfluss darauf, wie die Schülerinnen ihre Leistungen attribuieren. Im Idealfall kann ein vertieftes Wissen zu diesem Thema genutzt werden, um motivationsförderliche Muster zu etablieren.

Einen letzten Aspekt, den wir hier kurz besprechen wollen, sind Handlungsmotive. Während sich Motivation immer auf eine konkrete Situation bezieht, sind Motive eine dauerhafte Eigenschaft einer Person. Im Rahmen der Leistungsmotivation hat H. Heckhausen (1963) zwei Motivkategorien festgestellt: „Hoffnung auf Erfolg“ (HE) und „Furcht vor Misserfolg“ (FM). Menschen mit HE sind durch die Aussicht auf den Erfolg motiviert. Dazu passt zum Beispiel eine Selbstbeschreibung der Form „Ich will das schaffen!“. Bei FM hingegen wird versucht, ein negatives Ereignis abzuwenden, also „Ich darf keine Fehler machen!“.

Auf noch allgemeinerer Ebene kann man zwischen *Leistungsmotiven* (schwierige Aufgabe lösen), *Anschlussmotiven* (Teil einer Gruppe sein) und *Machtmotiven* (über Menschen bestimmen) unterschieden. Eine sehr detaillierte Behandlung dieser Motive findet in J. Heckhausen und H. Heckhausen (2010) statt.

Vielleicht mag dem einen oder anderem Leser jetzt der Kopf schwirren. Das Thema Motivation ist ein buntes Feld und man kann sich leicht in einzelne Teilaspekte verirren. Um nun das Kapitel abzurunden, soll als letztes ein einfaches Modell beschrieben werden, mit der man wie mit einer Formel die Motivation quasi „berechnen“ kann. Es handelt sich hierbei um das *Erwartung-mal-Wert-Modell* (Wigfield und Eccles, 2000). Wie unschwer zu erkennen, basiert dieses Modell auf zwei fundamentalen Variablen: dem subjektiven Wert einer Aufgabe und der Erwartung auf Erfolg. Die Motivation ist dann das Produkt aus diesen beiden

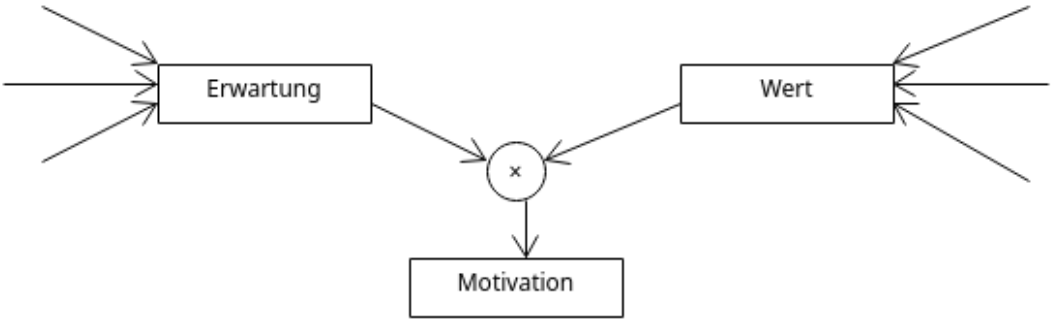


Abbildung 2.2: Grundstruktur des Erwartung-mal-Wert-Modells

Variablen. Es können also verschiedene Konstellationen auftreten. Zuerst einmal herrscht gar keine Motivation, wenn entweder die Erwartung oder der Wert null ist. Das passt gut zur Intuition: Man ist kaum geneigt, eine Aufgabe zu lösen, die unlösbar scheint - selbst wenn sie einen sehr hohen Wert. Umgekehrt hat man auch keine Motivation, eine Aufgabe zu lösen, die keinen Wert hat - selbst wenn sie leicht zu schaffen ist.

Für eine hohe Motivation ist es daher wichtig, dass sowohl Erwartung als auch der Wert hoch sind. Es gibt also zwei verschiedene Möglichkeiten, die Motivation zu steigern: Man kann einerseits versuchen, die Erwartung auf Erfolg zu erhöhen. Im Rahmen der Programmierung kann das beispielsweise dadurch erreicht werden, dass man eine weniger komplexe Programmierumgebung einsetzt. Hierbei ist aber zu beachten, dass die Erwartung sehr subjektiv ist. Die Lernumgebung sollte nicht nur faktisch einfacher zu bedienen sein, sondern sollte auch so von den Schülerinnen so wahrgenommen werden. Der zweite Weg besteht darin, den Wert zu erhöhen. Das kann beispielsweise dadurch gelingen, dass man die Aufgaben aus dem Lebenskontext der Schülerinnen aufgreift. Eine persönliche Identifikation führt häufig zu einer Erhöhung des Werts einer Aufgabe.

Mit diesem theoretischen Wissen ausgestatten können wir uns nun in den nächsten Abschnitten mit dem Thema der Programmierung beschäftigen.

2.2 Die zentrale Rolle von Motivation für das Programmieren Lernen

Dieser Abschnitt beschäftigt sich mit dem Zusammenhang zwischen Motivation und Programmieren Lernen. Zu Beginn werden dabei zwei verschiedene Herangehensweisen untersucht: Das Programmieren Lernen in informellen Umgebungen, sozusagen Programmieren als Hobby und Freizeit - und das Programmieren in formalen Umgebungen wie in der Schule oder Universität. Dabei wird schnell deutlich, dass Motivation einen großen Einfluss hat und damit ein Grundstein für erfolgreiches Lernen von Programmieren ist. Diese Einsicht mündet dann schließlich in einem neuen Rollenverständnis für Lehrerinnen.

Programmieren ist momentan stark im Hobby-Bereich vertreten. Das hat vor allem den Grund, dass Programmierung noch wenig im Unterricht behandelt wird. Und selbst wenn, dann nur auf einer sehr reduzierten Ebene. Dadurch liegt es nahe, dass man sich letztlich selbstständig mit der Informatik beschäftigt. Wenn man sich die Biographien der Gründer großer Technologiekonzerne anschaut, dann haben sich diese in der Regel ihr Wissen selbstständig erarbeitet. Aus welchen Gründen beschäftigen sich Menschen also mit Programmieren, oder anders gefragt: Was motiviert Menschen, sich in der Freizeit Programmieren beizubringen?

In Weigert (2017) beschreibt ein Journalist aus der persönlichen Erfahrung, wie er sich durch kleine Projekte anspornt. Guo (2017) beschreibt, was ältere Menschen motiviert, sich auch im fortgeschrittenen Alter das Programmieren beizubringen. Eine große Attraktivität ist, dass man sehr schnell Rückmeldung bekommt zu seiner Arbeit und die Ergebnisse leicht teilen kann. Der Journalist beschreibt beispielsweise, wie er das erste Mal ein eigenes Programm eingesetzt hat, um ein Diagramm für seine Arbeit zu generieren. Diese praktische Anwendung hat ihn sehr stark angespornt. Man kann diese Motivation zum Beispiel durch die drei großen Motive erklären, die im letzten Abschnitt angesprochen wurden: Das Machtmotiv wird aus dem letzten Beispiel schon deutlich, d.h. man kann real etwas bewirken. Durch die Beschäftigung mit Programmierung tritt man gleichzeitig auch in eine größere Gemeinschaft an Entwicklern bei und findet damit z.B. das Anschlussmotiv wieder. Und auch für das Leistungsmotiv gibt es beim Programmieren genug Möglichkeiten: Viele Herausforderungn ganz verschiedener Schwierigkeiten können angegangen werden.

Auch mit dem Erwartungs-mal-Wert-Modell kann eine hohe Motivation festgestellt werden. Aus den vorangehenden Überlegungen wird deutlich, dass Programmieren einen hohen Wert haben kann. Gleichzeitig gibt es mittlerweile immer mehr Systeme, die für Einsteiger zugeschnitten sind und damit die Erfolgserwartung stark steigern. Außerdem gibt es eine sehr große Fülle an Lehrmaterialien, die meist online und kostenlos verfügbar sind und mit denen man viele Projekte quasi Schritt-für-Schritt nachvollziehen kann.

Nach diesem Überlegungen würde man eigentlich davon ausgehen, dass beim Unterrichten von Programmieren nichts mehr schiefgehen kann. Es gibt viele Quellen an Motivation, so dass die Lehrerin eigentlich eine einfache Aufgabe hat. Die Realität in formalen Bildungseinrichtungen sieht leider anders aus. In der Forschung wird immer wieder berichtet, dass Programmieren ein Kurs ist mit einer vergleichsweisen hohen Abbruchsquote (Watson und Li, 2014). Kinnunen und Malmi (2006) stellten fest, dass fehlende Motivation ein großer Faktor für den Abbruch eines Kurses ist. Umgekehrt fanden Bergin und Reilly (2005) Anzeichen, dass eine höhere Motivation auch zu besseren Programmierleistungen führt. Während sich diese Studien besonders auf den universitären Bereich beziehen, wird Programmieren auch immer mehr in der Schule unterrichtet, siehe Szabo u. a. (2019). Die Forschung auf diesem Feld steckt leider noch in den Anfängen, aber es wäre fatal, wenn sich das Muster aus dem Tertiärbereich auch auf die Primar- und Sekundarstufe übertragen würde: Hohe Durchfallquoten aus Mangel an Motivation.

In Jenkins (2001) untersucht der Forscher und Dozent dieses Problem genauer. Er erkennt die wesentliche Rolle der Motivation für die Vermittlung von Programmierkenntnissen an und beschreibt:

> This paper argues that the primary role of a teacher of programming is not as a communicator of information, as in many other subjects or areas of computing. Rather, the teacher's main role is that of a motivator. The students must be motivated to engage in tasks that will make them learn, and it is the teacher's job to ensure this.

Dieser Grundgedanke bildet das Fundament für den weiteren Verlauf dieser Arbeit. Beim Programmieren Lernen ist es die wesentliche Aufgabe der Lehrerin, Motivation zu schaffen. Damit rücken neben den inhaltlichen Aspekten andere Themen mit in den Vordergrund. Diese sollen im nächsten Abschnitt besprochen werden.

2.3 Motivation im Unterricht steigern durch gezielte Anreize

Wie ist also ein guter Einstieg in die Programmierung zu gestalten? Die Erkenntnis aus dem letzten Abschnitt sagt aus, dass hierbei vor allem ein Fokus auf die Motivation wichtig ist. Die Lehrerin steckt in der Rolle eines Motivators. Ihre Aufgabe besteht im Kern darin, im Unterricht Anreize zu schaffen, durch die die Schülerinnen zur Beschäftigung mit Programmieren angeregt werden. Technische und inhaltliche Aspekte sollen diesem Ziel untergeordnet werden.

Um dieses Rollenverständnis auszufüllen, bedarf es einer ausgiebigen Recherche. Geeignete Anreize für den Unterricht zu finden ist keine einfache Aufgabe. Die Motivationstheorien zeigen, dass dafür einiges an psychologischen Feingefühl erforderlich ist. Die behandelten Themen sollen idealwerweise an die Lebenswelt der Schülerinnen angebunden werden, die Erwartung für den Erfolg erhöht werden und gleichzeitig auch ein Einsatz in einem Klassenzimmer mit bis zu 30 Schülerinnen möglich sein.

Um die Arbeit für angehende Lehrerinnen zu erleichtern, bietet diese Arbeit eine umfassende Sammlung von Anreizen für den Unterricht. Die Hauptzielgruppe ist dabei die Sekundarstufe, wobei der Großteil der Ansätze über alle Altersgruppen hinweg anwendbar. Eine Übertragung der Inhalte auf die Primarstufe oder den universitären Bereich ist also grundsätzlich möglich. Ganz allgemein brauchen natürlich alle Beispiel eine Anpassung auf die konkrete Situation und die konkrete Klasse.

Dem Autor ist keine solche Übersicht bisher in der Literatur bekannt. Es gibt allerdings viele einzelne Projekte und Lernumgebungen, die sich mit diesem Thema beschäftigen. Diese Lernumgebungen und ihre Literatur werden im nächsten Kapiteln an passender Stelle ausführlich behandelt. Die ähnlichsten Übersichten finden sich ganz allgemein zu Sprachen und Tools, wie z.B. Kelleher und Pausch (2005). Diese Quelle ist etwas veraltet und nicht speziell auf das Thema Motivation speziell ausgerichtet, dennoch war sie ein wichtiger Ausgangspunkt bei der Recherche.

Das nächste Kapitel wird also eine Übersicht geben über Anreize zum Programmieren lernen, mit denen ein motivierter Einstieg in die Programmierung gelingen kann. Viel Spaß und Motivation beim Lesen!

KAPITEL 3

Kategorien

Sobald man sich ein wenig auf die Suche macht, findet man eine sehr große Menge an Ideen und Anreize, die Programmierung mit Spaß und Motivation zu verbinden. Dieses Kapitel wird vollständig diesen Anreizen gewidmet sein. Um aber bei der Fülle an Ideen nicht die Übersicht zu verlieren, wird die Darstellung nach drei Kategorien gegliedert stattfinden. Die erste Kategorie *Interesse wecken* zielt dabei darauf ab, den Wert des Programmierens für die Schülerinnen zu erhöhen und an ihre Interessen anzuknüpfen oder neue Interessen aufzubauen. Die zweite Kategorie *Komplexität reduzieren* versucht, die Erwartung auf Erfolg zu steigern, indem unnötige Schwierigkeiten vermieden werden. Die dritte Kategorie *Unterstützung anbieten* vereinfacht den Inhalt nicht, aber versucht durch verschiedene Methoden die Schülerinnen zu unterstützen.

Eine besondere Art von Anreizen, die sehr häufig vorkommen, sind Lernumgebungen. Das sind Softwaresysteme, die mehrere Tools zum Programmieren - wie Editor, Interpreter oder Visualisierung - in einem Ganzen integrieren und mit bestimmten Inhalten verbinden. Diese Integration bietet ein besonderes Potenzial, das im Laufe des Kapitel sehr deutlich werden wird. Glücklicherweise haben sich viele Menschen mit dem Potenzial von Lernumgebungen befasst, sodass wir heute einige ausgereifte Ansätze zur Hand haben, die sich auch in einem Alltag im Klassenzimmer mit vielen Schülerinnen bewähren kann.

3.1 Interesse wecken

Beginnen wir also mit der Kategorie *Interesse wecken*. Ist in einer Schülerin das Interesse einmal geweckt, dann kann daraus eine sehr große Motivation für das Thema entstehen. In der Psychologie ist besonders das Modell von Hidi und Renninger (2006) beliebt, weil es einen starken Augenmerk auf die Entwicklung von Interesse legt und damit den Lehrerinnen sozusagen einen Fahrplan zur Hand gibt, wie ein gelungener Prozess aussehen kann. Abbilung 3.1 zeigt eine graphische Darstellung der vier Phasen.

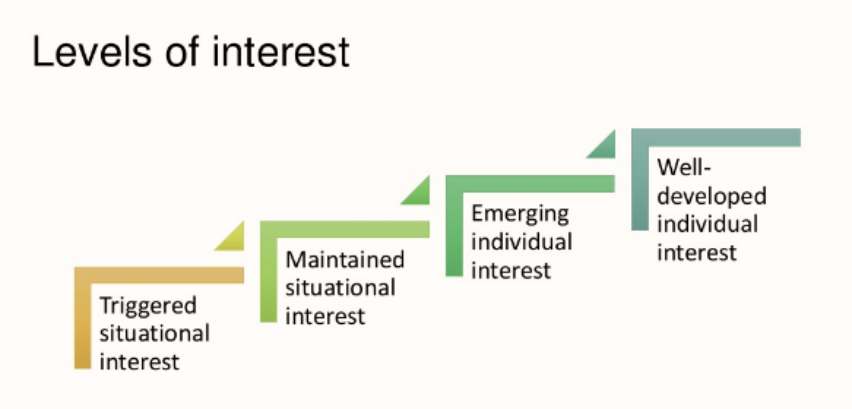


Abbildung 3.1: Darstellung zum Vierphasenmodell der Entwicklung von Interesse, Quelle: `https://www.slideshare.net/JessicaHarlan1/context-and-support-factors-in-elementary-and-middle-school-stem-programs/13`

Interesse beginnt sich zu entwickeln, wenn man in Situationen kommt, die z.B. etwas Überraschendes, Relevantes oder Intensives enthalten. Solche Auslöser sind meist von außen bereitgestellt und können von der Lehrerin im Unterricht vermittelt werden. Darauf aufbauend wird die zweite Phase erreicht, wenn die Lehrerin es schafft, diese Auslöser zu fokussieren und über einen längeren Zeitraum zu halten. Auch in dieser Phase kommt der Antrieb noch von außen. Das ändert sich dann in der dritten und vierten Phase. Dort verschmelzen die Anreize von außen mit inneren Antrieben und die Schülerinnen beginnen, sich selbstständig mit dem Thema zu beschäftigen.

Dieser Unterabschnitt möchte also untersuchen, welche Anreize geeignet sind, um Schülerinnen in die ersten beiden Phasen der Interessensentwicklung zu bringen. Sie sollen dabei gleichzeitig offen sein für eine weitere Entwicklung.

3.1.1 Agent in einer virtuellen Welt

Wer bin ich und was kann ich tun? Viele Lernumgebungen wecken das Interesse der Lernenden, indem sie auf diese beiden Fragen interessante Antworten geben. Die beiden klassischen Realisierungen dieser Frage lauten: Eine Schildkröte, die zeichnet - und ein Roboter, der Aufgaben löst. Beide sind schon vor mehr als 40 Jahren entwickelt worden und werden bis heute immer noch eingesetzt und neu interpretiert. Dieser Abschnitt möchte darstellen, aus welchen Quellen diese Ideen entstammen und die Gedanken zeigen, die sich die Originalautoren gemacht haben. Es wird dann versucht, die Entwicklung der letzten Jahrzehnte nachzuvollziehen und es werden Ergebnisse aus der Forschung vorgestellt. Zuletzt sei noch ein Beispiel gegeben für eine überraschende Wahl des Agents.

Eine ausgezeichnete Quelle, sich mit den Gedanken hinter der zeichnenden Schildkröte zu befassen, ist Papert (1980). Darin stellt Seymour Papert seine Vision vom Lernen dar, die sich auf den Konstruktivismus von Piaget stützt und er beschreibt die „Schildkrötengeometrie“ als eine Form der Geometrie, die sich besonders gut zum Lernen eignet. Erklären tut er das mit der Tatsache, dass Kinder ihr Vorwissen über den eigenen Körper auf die Schildkröte anwenden können.

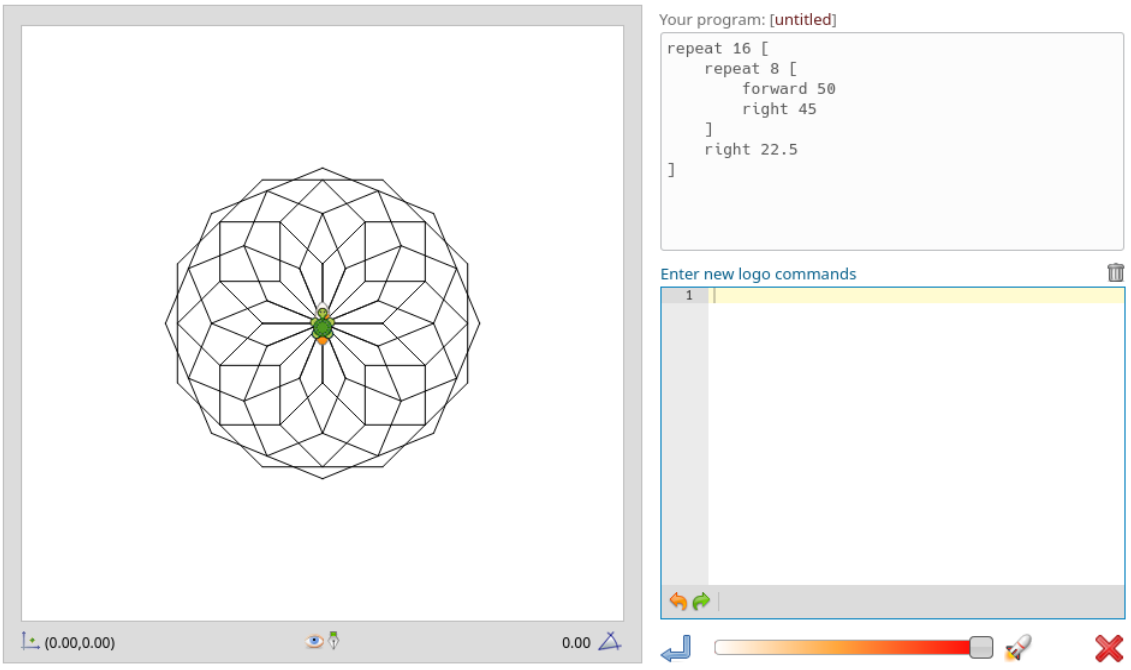


Abbildung 3.2: Die Schildkröte zeichnet ein Muster in LOGO
Quelle: `https://www.logointerpreter.com`

Sie können sich in den Agenten hineinversetzen und dadurch die Bewegungen vollständig nachvollziehen. Diese Anknüpfung an das Vorwissen ist für ihn ein zentraler Aspekt, weil es das Spielen und Experimentieren ermögliche. Das besondere an der Schildkrötengeometrie ist ihre algorithmische Art. Objekte können nur gezeichnet werden, indem man über die Programmiersprache LOGO der Schildkröte Befehle gibt. Das ist der neue Aspekt, den die Kinder lernen müssen. Durch den intuitiven Kontext aber sieht Papert das Potenzial, dass die Kinder mit der Sprache experimentieren können und beschreibt das als einen „aufregenden und spielerischen Prozess“ (Papert, 1980, S. 57).

Die Abbildung 3.2 zeigt ein komplexes geometrische Objekt und den zugehörigen Quellcode in LOGO. Wenn man in der unteren Zeile auf das Enter klickt, dann wird die Ebene zurückgesetzt und die Schildkröte fängt an, das Muster nach und nach zu zeichnen. Man kann der Schildkröte zuschauen und nachvollziehen, wie die einzelnen Zeilen des Programms zu den entsprechenden Bewegungen führt. Die hier gezeigte Lernumgebung ist über den Webbrowser aufrufbar. Dort finden sich auch viele weitere Beispiele und eine Dokumentation der Sprache. Diese Implementation ist sehr nahe am Original angelegt und entspricht weitestgehend der Umsetzung von Papert.

LOGO und die Schildkröte haben eine ganze Menge an Lernumgebungen inspiriert. Anpassungen lassen sich leicht durchführen: Die konkrete Umsetzung des Agenten lässt sich beliebig austauschen - sei es nun eine Schildkröte, ein Grashüpfer [1] oder eine niedliche Katze.[2] Auch die eingesetzte Sprache lässt sich austauschen, es gibt Implementation in Python,[3] Java[4] oder auch in einer blockbasierten Programmiersprache.[5] In Kelleher und Pausch (2005) sind außerdem 29 Lernumgebungen und Sprachen aufgelistet, die nach der Einschätzung der Autoren von LOGO beeinflusst wurden - und die wiederum weitere Lernumgebungen inspiriert haben. Einige empirische Untersuchungen sprechen dafür, dass der Einsatz von Turtle-Graphik in der Lage ist, die Motivation der Lernenden zu steigern, siehe z.B. Araujo, Bittencourt und Santos (2018) für eine Studie in der Sekundarstufe und Bakar, Mukhtar und Khalid (2020) für eine Studie an einer Universität. Insgesamt lässt sich sagen, dass die Schildkrötengeometrie eine grundlegende Idee ist, die sich heute in vielen Formen wiederfindet.

[1] Grasshopper: `https://grasshopper.app/`
[2] Scratch: `https://scratch.mit.edu/`
[3] Python Standard Library: `https://docs.python.org/3/library/turtle.html`
[4] Java Turtle Package: `https://sourceforge.net/projects/jturtle/`
[5] Pencil Code: `https://pencilcode.net/`

Einen anderen Ansatz verfolgt Richard Pattis. Im Vorwort zu Pattis (1981) beschreibt der Autor, dass viele Studierende Angst vor Computer haben. Daher hat er es sich zur Aufgabe gemacht, die Grundlagen der Programmierung in einem anderem Umfeld zu unterrichten, als den von Computern. Das ist eine der Anregungen, die schließlich zur Entwicklung von KAREL THE ROBOT führte. In dieser Lernumgebung übernimmt der Lernende die Kontrolle über einen „wortgetreuen“ Roboter, der sich in einer Kästchenwelt bewegt und diese wahrnehmen kann. Durch den Einsatz von Programmierkonzepten lassen sich Schritt für Schritt immer komplexere Probleme lösen. Nach dem Autor bietet der auf dem ersten Blick simple Ansatz durchaus Potenzial für reichhaltige Aufgabenstellungen.

Abbildung 3.3: Rätsel lösen mit Swift in der App Swift Playgrounds
Quelle: `https://mrjonesict.com/2017/02/03/getting-started/`

Ähnlich wie LOGO hat sich auch diese Idee weit verbreitet. Es gibt unzählige Implementationen in verschiedenen Programmiersprachen, einen Überblick bietet dazu Freiberger (2002). In Abbildung 3.3 ist die Lernumgebung Swift Playgrounds (`https://www.apple.com/de/swift/playgrounds/`) dargestellt. Das Ziel besteht darin, Diamanten zu sammeln. Das gelingt durch Programmierung des Roboters in der Sprache Swift. Diese Implementation zeichnet sich durch

ihre aufwendige graphische Gestaltung aus und die besondere Nutzung des Touchscreens. Ansonsten ist sie der Grundidee von 1981 ziemlich treu geblieben. Es gibt verschiedene Level mit Rätsel, die immer komplexere Sprachelemente einsetzen. Für bayerische Schulen gibt es das Programm Robot Karol (Freiberger, 2018).

Verschiedene Varianten von Agenten und virtuellen Welten untersucht Brusilovsky u. a. (1997), vor allem in Kombination mit Minisprachen (näheres dazu im Unterabschnitt 3.2.3). Diese beiden Ansätze lassen sich besonders gut kombinieren und ermöglichen einen intuitiven Einstieg in die Programmierung. Pelánek und Effenberger (2020) haben untersucht, wie sich allgemein Problemstellungen in einer solchen Welt darstellen lassen. Sie deuten an, dass es besser ist, mehrere Welten zur Verfügung zu haben, die jeweils einen Aspekt betonen, z.B. LOGO und KAREL. Außerdem sollte besonders auf die Komplexität der Aufgabenstellungen geachtet werden und sofern möglich diese mit Nutzungsdaten überprüft werden, wie z.B. Bearbeitungszeit oder Erfolgsrate.

Einen etwas überraschenden und nicht ganz alltäglichen Agenten bietet CargoBot (`https://twolivesleft.com/CargoBot/`). Hierbei übernimmt der Lernende die Programmierung eines Lastenkrans. Dieser kann sich auf einer Dimension bewegen und die Blöcke unter sich greifen. Die Programmierung findet graphisch statt. Neben einfachen Befehlen sich auch Bedingungen zugelassen sowie Rekursion. Das Konzept bietet eine erfrischende Abwechslung - auch für Profis.

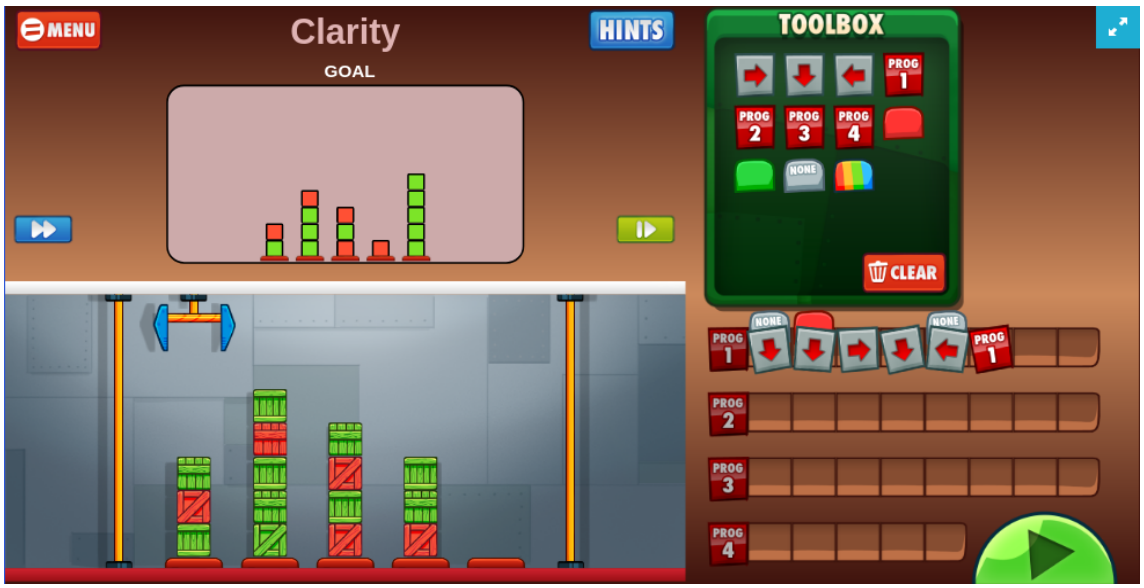


Abbildung 3.4: Ein Level von CargoBot, Bildschirmaufnahme von der Webversion, Quelle: `https://altermanchess.wixsite.com/cargobot`

3.1.2 Kunst und Multimedia

Viele Menschen verspüren das Bedürfnis, sich nach außen auszudrücken. In den meisten Programmierumgebungen, die nur textuelle Ein- und Ausgaben erlauben, sind die Möglichkeiten dafür eingeschränkt. Heute ist es aber technisch deutlich einfacher geworden, neben Text auch Bilder, Zeichnungen, Ton, Musik und Animationen zu bearbeiten und zusammenzustellen. Es gibt Lernumgebungen, die sich bewusst dafür einsetzen, diese neuen Medien für Lernende zugänglich zu machen und gleichzeitig mit der Programmierung zu verbinden. Die Kunstwerke, die so entstehen, können auch interaktiv sein und durch den Betrachter beeinflusst werden. Zwei große Projekte haben sich diese Aufgabe zum Ziel gesetzt und sollen in diesem Abschnitt vorgestellt werden. Außerdem gibt es zum Schluss noch witziges Beispiel mit einem dritten Projekt.

Das erste Projekt, mit dem wir uns hier befassen wollen, ist Scratch (Maloney u. a., 2010). In der neusten Version ist der Editor über den Browser aufrufbar. Als Besonderheit bietet der Editor neben einer blockbasierten Programmiersprache auch einen eingebauten Bildeditor und Soundrekorder. Mithilfe einer großen Mediensammlung können schnell und ohne großem Aufwand die ersten Projekte erstellt werden. Über eine Registrierung auf der Website lassen sich zudem die

Abbildung 3.5: Ein Animatic Music Video von otterstar auf der Scratch Website, Quelle: `https://scratch.mit.edu/projects/16136830/`

eigenen Projekte veröffentlichen und mit anderen Menschen teilen.

Um einen Eindruck davon zu bekommen, was Kinder in der Lage sind, mit diesen Werkzeugen zu schaffen, empfiehlt es sich sehr, sich das Projekt in Abbildung 3.5 anzuschauen. Es zeigt ein animiertes Musikvideo. Man erkennt gut, wie viel Arbeit in dieses Projekt geflossen ist und wie die Entwicklerin mit den ihr zur Verfügung stehen Mitteln etwas eindrucksvolles geschaffen hat.

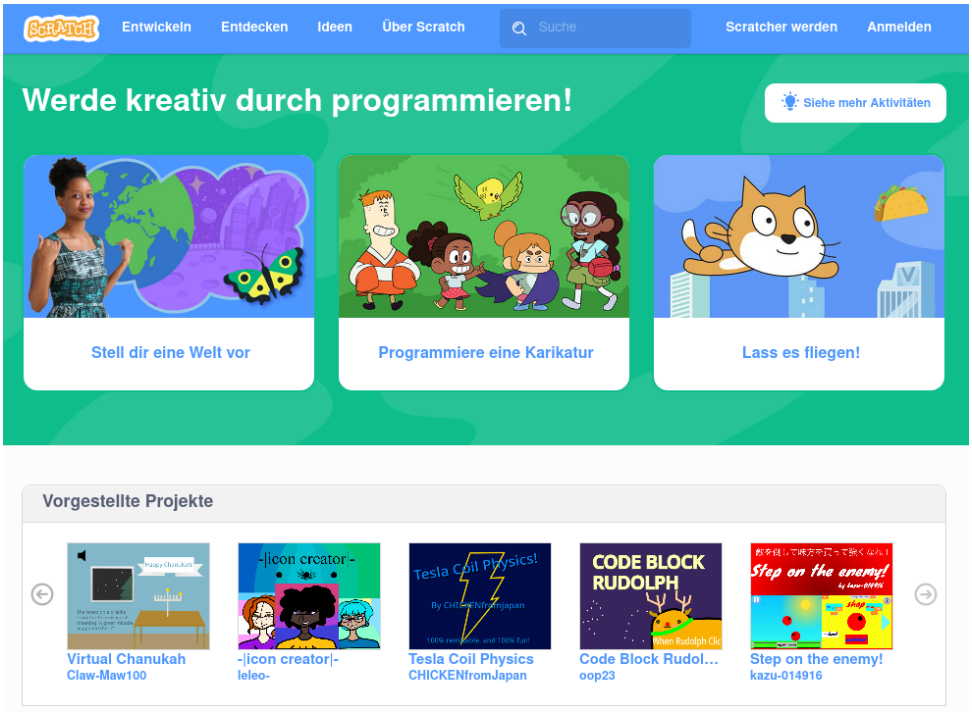


Abbildung 3.6: Die Startseite von Scratch mit vorgestellten Projekten, Dezember 2020, Quelle: `https://mit.scratch.edu`

Es lohnt sich, die Website von Scratch zu besuchen und sich ein wenig durch die vorgestellten Projekte zu klicken. In Kafai und Burke (2014) stellen die Autoren im großen Detail dar, welches Konzept sie hinter dieser Plattform sehen. Die Programmierung dreht sich hierbei nicht allein um das Lösen von Problemen, sondern vor allem darum, etwas zu erstellen, das man miteinander teilen kann und anderen Menschen zeigen kann. Das ermöglicht eben es, wie eingangs erwähnt, sich selbst auszudrücken. Krugel und Ruf (2020) stellten fest, dass die Lernenden ihre Erfahrungen mit Scratch stark mit Programmieren verbinden und es als positiv wahrnehmen, dass sie sehr viel ausprobieren konnten.

Widmen wir uns nun dem zweiten Projekt zu, dass die Programmierung mit Kunst und Multimedia verbinden möchte. In Reas und Fry (2007) wird die Pro-

grammierumgebung Processing (`https://processing.org/`) vorgestellt. Die Umgebung lässt sich auf den Computer installieren und stützt sich auf Java. Die Sprache wurde aber angepasst und vereinfacht. Insbesondere können graphische Objekte sehr leicht erstellt werden. Außerdem kann man direkt auf Mauszeiger zugreifen und auf Tasten reagieren. Das ermöglicht es, den Einstieg in die Programmierung mithilfe eines visuellen Ansatzes zu verfolgen. Die Studie von Greenberg, Kumar und Xu (2012) konnte eine positive Auswirkung auf die Motivation feststellen.

Über die Jahre hat sich die Landschaft verändert, die Programmierung findet immer mehr im Browser statt. Das hat zur Entwicklung von p5.js geführt (McCarthy, Reas und Fry, 2015, `https://p5js.org/`). Die wesentlichen Elemente von Processing bleiben erhalten, allerdings wurde die Basis auf JavaScript umgestellt. Abbildung 3.7 zeigt einen interaktiven Sketch. Durch die Bewegung mit der Maus können farbige Punkte gezeichnet werden, die langsam alle Farbtöne durchlaufen. Der gesamte Quellcode ist auf der rechten Seite zu sehen.

Das Beispiel zeigt, dass p5.js sehr stark für visuelle Programmierung optimiert ist. Es ist nicht notwendig, Klassen zu erzeugen. Diese können aber in Programmen eingesetzt werden. Das Grundgerüst besteht aus den zwei Funktionen `setup` und `draw`. Die Methode `setup` wird einmal am Anfang der Ausführung

```
let c = 0;

function setup() {
  createCanvas(windowWidth, windowHeight);
  background(255);
  colorMode(HSL);
  noStroke();
}

function draw() {
  // draw
  fill(c, 100, 50);
  ellipse(mouseX, mouseY, 20, 20);

  // update
  c = (c + 1) % 360;
}
```

Abbildung 3.7: Ein interaktiver Sketch in p5.js mit Quellcode, eigenes Programm, Quelle: `https://www.openprocessing.org/sketch/930942`

aufgerufen. Hier wird die Leinwand erzeugt. Die Methode `draw` wird in einem kleinen Intervall immer wieder aufgerufen und zeichnet pro Aufruf einen Kreis.

Auch zu Processing lassen sich im Internet eine große Zahl an Beispielen finden.[6] Diese sind teilweise sehr professionell gestaltet und es ist immer wieder unterhaltsam, sich die verschiedenen Kunstwerke anzuschauen.

Kommen wir zur dritten Projekt. Die beiden bisherigen Projekte beschränken sich weitestgehend auf zweidimensionale Ausgaben. Durch die technische Entwicklung lassen sich mittlerweile aber auch dreidimensionale Graphiken leicht erzeugen. Die Lernumgebung Alice (Cooper, Dann und Pausch, 2000) führt diese Idee aus. Die Lernumgebung erlaubt es, Objekte in einer dreidimensionalen Welt zu platzieren und zu bewegen. Man kann mit Sprechblasen Dialoge erstellen. Die Programmierung findet hier blockbasiert statt. In Kelleher, Pausch und Kiesler (2007) zeigt sich an einer Studie mit Mädchen, dass der Aspekt des Geschichtenerzählens eine deutliche Steigerung der Motivation bewirken konnte.

Abbildung 3.8: „... we all have to drink from that pond?“, programmiert mit Alice, Quelle: `https://www.youtube.com/watch?v=h_nWqSe1oKw`

Eine nette kleine Geschichte findet sich in Abbildung 3.8. Es empfiehlt sich, sich die Originalanimation unter dem Link direkt anzuschauen. Dieses Beispiel und die vorherigen Beispiele in diesem Abschnitt sollten einen ersten Eindruck davon gegeben haben, inwiefern sich Kunst und Multimedia mit der Programmierung verbinden lassen und was für Kunstwerke so entstehen können.

[6] Siehe OpenProcessing: `https://www.openprocessing.org/browse`

3.1.3 Mikrocontroller, Sensoren, LEDs

Viele Menschen betrachten ihren Laptop oder ihr Handy als eine Blackbox, deren wichtigste Funktion es ist, Software zum Laufen zu bringen. Welche Teile diese Blackbox besitzt und wie sie funktionieren interessiert im Alltag nur selten. Doch manchmal fragt man sich schon, wie „diese Kiste“ eigentlich funktioniert. Für solche neugierigen Menschen gibt es Ansätze, die sich zur Aufgabe machen, die Blackbox zu öffnen. Zwei solche Projekte möchte dieser Abschnitt beispielhaft vorstellen. Das erste Projekt bietet die Möglichkeit, eine Platine selbst zusammenzulöten, während das zweite Projekt mit der großen Vielfalt an Sensoren mehr Raum lässt für erfinderische Ideen.

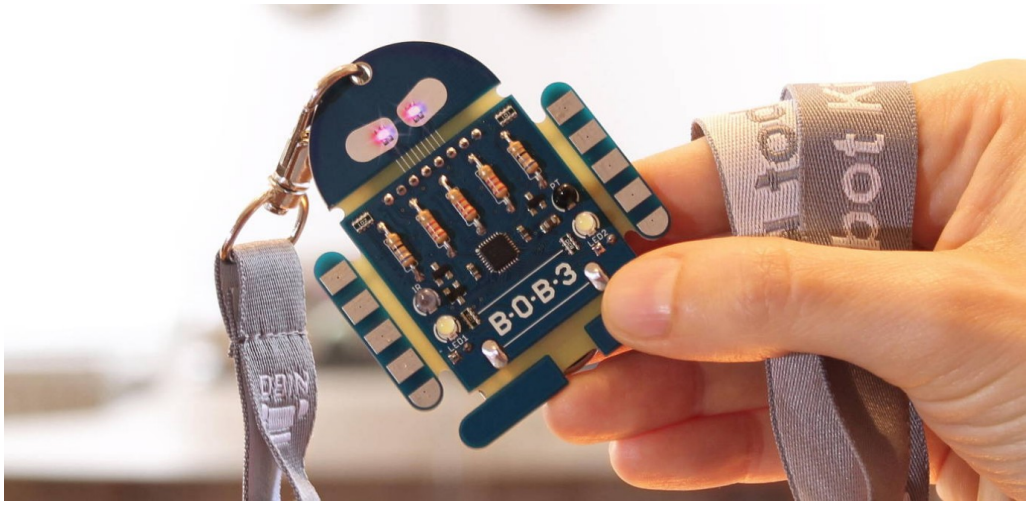


Abbildung 3.9: Der Minicomputer BOB3, Quelle: `https://www.bob3.org`

In Abbildung 3.9 ist das erste Projekt zu sehen. Der BOB3 Roboter (Bach, 2017) lässt sich als Bausatz kaufen und muss vor dem ersten Betrieb zusammengelötet werden. Für den Einsatz in der Schule lässt sich aber auch eine fertige Variante kaufen. Danach kann er programmiert und somit zum Leben erweckt werden. Die zentrale Einheit ist der Mikrocontroller, der sich in der Mitte des Roboters befindet. Auf diesem Mikrocontroller sind CPU, RAM, Flash-Speicher und Ein-/Ausgabe-Module integriert. An den Controller sind dann die Sensoren angeschlossen, in diesem Fall gibt es einen Temperatur-Sensor, einen Infrarot-Sensor und Tastsensoren. Zur Ausgabe gibt es vier LEDs, zwei Farbige in den Augen und zwei Weiße am Bauch.

Die Programmierung lernt man durch die integrierte Lernumgebung ProgBob. In Abbildung 3.10 ist eine kleine Beispielaufgabe zu sehen. ProgBob lässt

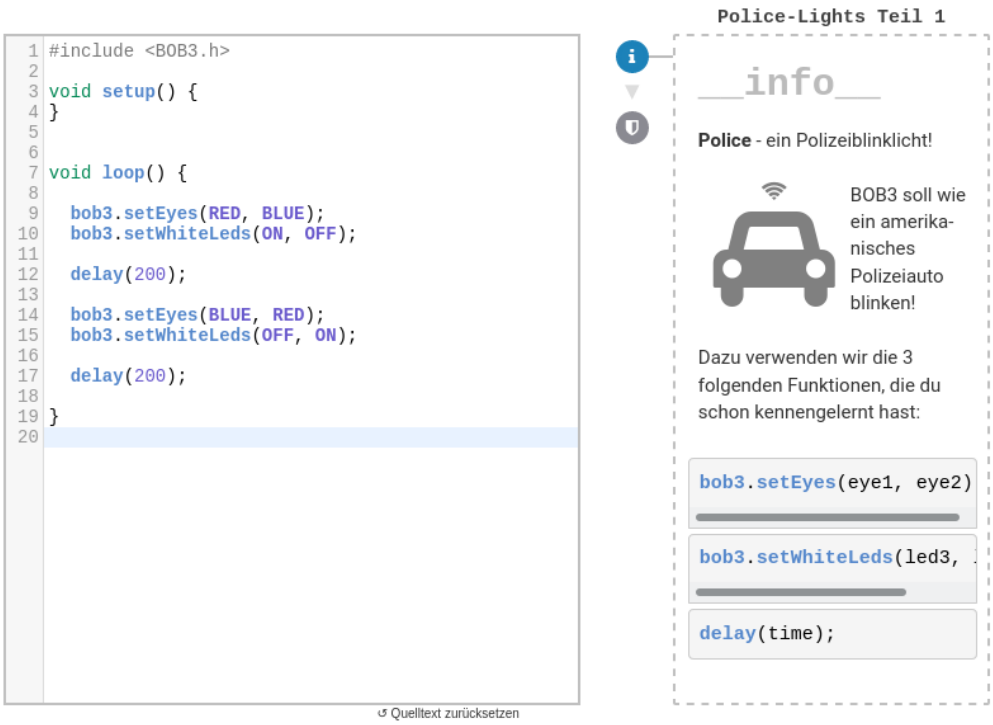


Abbildung 3.10: Polizei-Blicklicht programmieren für den BOB3 mit der Lernumgebung ProgBob, Quelle: `https://www.progbob.org`

sich über den Browser starten und führt die Schülerinnen Schritt für Schritt in die Programmierung mit C ein. Die Lernumgebung ist eng an Arduino (Badamasi, 2014) angelehnt. BOB3 möchte damit bewusst erreichen, dass man das erlernte Wissen später schnell auf die ganze Arduino-Reihe (Wikipedia, 2021b) übertragen kann. Das Grundgerüst eines Programms besteht aus der Funktion `setup`, welche einmalig am Anfang aufgerufen wird, und `loop`, welche in einer Schleife immer wieder aufgerufen wird. In dem gezeigten Beispiel sollen die Augen des BOB3 wie ein Polizeiauto blinken. Dazu schaltet man im entsprechenden Takt die LEDs an und aus bzw. setzt sie auf eine bestimmte Farbe.

Eine Evaluation von BOB3 und der Programmierumgebung findet durch Langer, Bergner und Schroeder (2019) statt und kommt zu durchaus positiven Ergebnissen. Im Fokus stand dabei vor allem die Mädchenförderung für den MINT-Bereich. Auch Bach (2017) berichtet von positiven Erfahrungen. Sie betont vor allem, dass man die Fähigkeiten der Kinder, in ihrem Fall aus der 5. Klasse, nicht unterschätzen soll. In diesem Alter sind sie schon zu sehr viel in der Lage und können sich autonom mit Technik beschäftigen.

BOB3 ist bei weitem nicht das einzige Projekt dieser Art. In den letzten Jahren gab es eine ganze Welle an Einplatinencomputer und Bausätze und es ist zu erwarten, dass regelmäßig neue Produkte auf den Markt kommen. Um den Rahmen nicht zu sprengen soll daher nur noch ein weiteres Projekt vorgestellt werden. Dieses Projekt wurde ausgewählt, weil es sich vor allem an deutschen Schulen großer Beliebtheit erfreut und in allen Teilen quelloffen ist. Es handelt sich hierbei um den Calliope mini (Wikipedia, 2021a).

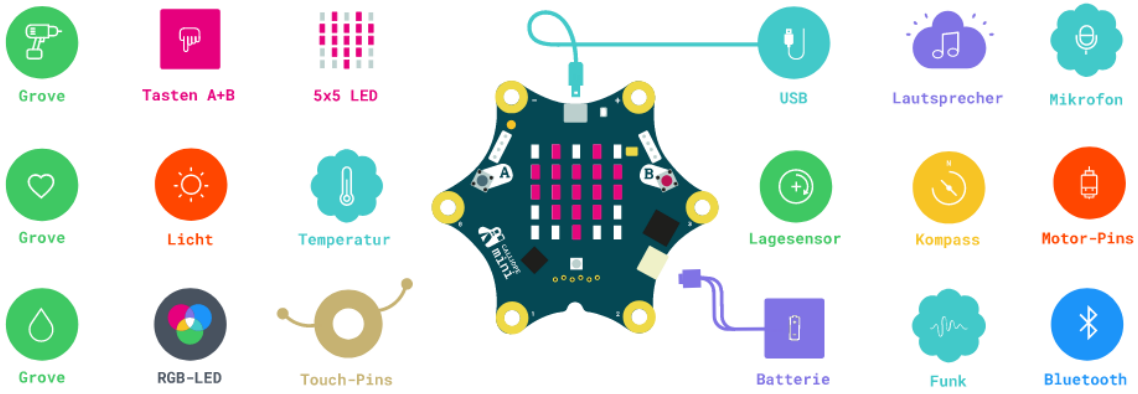


Abbildung 3.11: Übersicht über die Funktionen des Calliope mini, Quelle: `https://calliope.cc/calliope-mini/uebersicht`

Wenn man sich die Abbildung 3.11 anschaut, dann fällt auf, dass der Calliope mini einen deutlich größeren Funktionsumfang bietet als der BOB3. Der Calliope mini ist auch nur als fertige Platine zu kaufen. Der Funktionsumfang ist gleichzeitig auch die besondere Stärke des Calliope, weil man dadurch mehr Möglichkeiten hat für interessante Projekte. Eine Sammlung von 25 Programmen ist beigefügt, man kann sich die Programme unter `https://calliope.cc/calliope-mini/25programme` anschauen. Diese reichen von einem digitalen Würfel über Musikgeräte hin zu ganzen Minispielen. Die Programmierung kann über verschiedene Editoren stattfinden, es wird sowohl blockbasierte als auch textbasierte Programmierung unterstützt.[7]. Es gibt viele Unterrichtsmaterialen, die meist bereits für Grundschülerinnen geeignet sind.

Auch in der Forschung hat der Calliope mini Interesse geweckt. In Murmann (2018) findet sich eine großangelegte Studie, die den Calliope mini an verschiedenen Grundschulen einsetzte. Als Fazit steht geschrieben: „Heute, im Mai 2018, ziehen wir Bilanz. Es ist mit Blick auf die Umsetzbarkeit eines bildungswirk-

[7]MakeCode für Calliope: `https://makecode.calliope.cc/`

samen Grundschulunterrichts mit dem Calliope mini eine positive Bilanz.“ Als „Nebeneffekt“ konnten die Forscher nachweisen, dass die Kinder nach den Unterrichtseinheiten sicherer mit dem Computer umgehen konnten als davor.

Der Calliope mini ist nicht nur für Grundschülerinnen interessant. Durch die Vielzahl an Fähigkeiten eignet sich dieses Gerät auch hervorragend für das Maker Movement (Dougherty, 2012). In Halverson und Sheridan (2014) wird diese Idee auch explizit für die Schulbildung ausgearbeitet. Bei der Recherche nach interessanten Projekten ist dem Autor vor allem das Projekt in Abbildung 3.12 aufgefallen. Der Titel gibt an, dass man mit dem Calliope mini Längen messen kann. Aber mit welcher Methode? Der interessierte Leser kann gerne selber kurz überlegen, welcher Sensor sich dafür eignen könnte.

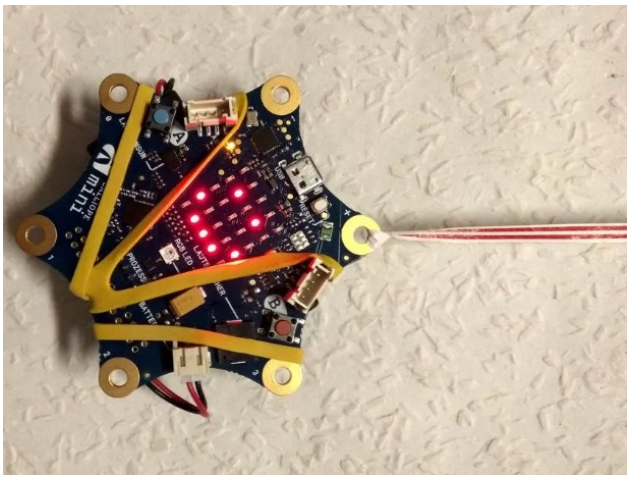


Abbildung 3.12: Mit dem Calliope mini Längen messen? Dieses Projekt beschreibt, wie das gehen kann, Quelle `https://www.hackster.io/Weja/calliope-weiss-wie-gross-du-bist-dce2a0`

Die Auflösung ist so einfach wie auch überraschend: Man misst mit dem Band eine Länge ab und lässt dann den Calliope mini wie ein Pendel schwingen. Weil die Schwingungsdauer nur von der Länge abhängt, kann der Calliope mini mit seinem Beschleunigungssensor die Schwingungsdauer messen und daraus die Länge herleiten. Dieses Beispiel soll nochmal verdeutlichen, was alles mit dieser Hardware möglich ist.

3.1.4 Das Internet der Dinge

Die Zeiten, wo ein Computer die Größe eines Raums hatte, sind vergangen. Mittlerweile hat selbst ein Handy mehr Rechenleistung als ein solcher Computer von damals. Aber nicht nur die Rechenleistung hat sich gesteigert. Die Geräte haben nun auch viel mehr Möglichkeiten, sich zu vernetzen und nutzen Protokolle wie WLAN, Bluetooth oder NFC, um mit anderen Geräten in der Nähe zu kommunizieren. Dieses Internet der Dinge hat großes Potenzial. Dieser Abschnitt möchte dieses Potenzial anhand von drei ausgewählten Beispielen verdeutlichen. Vor allem das letzte Beispiel zeigt, wie durch die Vernetzung von Geräten und Informationen auch sozialer Wandel vorangetrieben werden kann.

Abbildung 3.13: Kontaktlose Stoppuhr zum Händewaschen mit dem Calliope mini, Quelle: aliceabs (2020)

Das erste Beispiel ist eine Reaktion auf die Corona-Krise. Viele Menschen haben nicht die Gewohnheit, sich die Hände für mindestens 30 Sekunden zu waschen. Diese Gewohnheit ist aber in unserer aktuellen Krise ein wichtiges Mittel, um die Übertragung von Krankheiten zu reduzieren. Ein kleines Gerät soll dabei helfen: Der Calliope mini in Abbildung 3.13 ist so programmiert, dass er durch ein akustisches Signal, z.B. Pfeifen, aktiviert werden kann. Das startet eine Stoppuhr mit der empfohlenen Händewaschzeit. Nach Ablauf der Zeit erscheint ein Herz und der Calliope mini spielt eine Melodie ab.

Das Programm in der Form nutzt noch nicht alle Möglichkeiten des Calliope mini aus. Der Chip enthält zusätzlich noch ein Bluetooth-Modul. Das kann beispielsweise mit dem Handy gekoppelt werden. Eine App könnte dann die Informationen auswerten und z.B. anzeigen, wie oft man am Tag die Hände gewaschen hat oder ob man die Hände genügend lang gewaschen hat.

Eine andere Möglichkeit: Man kann sich Sensoren vorstellen, die darüber hinaus den CO2-Gehalt in der Luft messen und beim Unterschreiten einer Schwelle ein Signal zum Lüften geben. Ein Temperatursensor kann dann überprüfen, ob das Lüften tatsächlich stattfand. Diese Informationen können dann gesammelt und weiterverarbeitet werden, um z.B. Klassen zu identifizieren, die zu wenig lüften. Solche Projekte haben einen besonderen Motivationsfaktor, denn sie können wirksam in unseren Alltag eingreifen und ihn verändern.

Kommen wir zum zweiten Beispiel. Zum Einstieg soll dazu folgendes Szenario aus Shapiro und Tissenbaum (2019) zitiert werden:

> Davina, a seventh-grade girl, lives in a rural community. She loves her dog, Slayer, and wonders what he does all day. [...] Davina's friends have similar curiosities: Where do their dogs go? What other dogs do they hang out with? [...] With their teacher's help, they use the BBC Micro:bit (a small, low-cost microcontroller with built-in sensors and wireless communication) to create animal wearables that they can use to create a data set about their dogs' social and geo-spatial activities. They assign each dog a unique ID number and program each dog's Micro:bit to broadcast that unique ID every 30 seconds using the built-in short-range wireless radio. They also program each device to listen for broadcasts from other dogs' devices; each device will count up how many times it receives broadcasts from every other device. The next time they see their dogs, they attach the Micro:bits (in waterproof cases) to the dogs' collars, and send them out on their adventures. Then, when they see the dogs again, the students download their dogs' data sets to their laptops. They bring these data to class, and each creates a graph of how much time their dog hangs out with every otherdog. [...]

Für Davina ist die wesentliche Motivation die Neugier und ihr Interesse für ihren Hund, das sie vorantreibt. Aus der Sicht von Shaffer und Resnick (1999)

lässt sich eine große Authentizität wiederfinden. Insbesondere spiegeln die eingesetzten Technologien auch neuere Entwicklungen wieder: Die Systeme sind verteilt, es gibt viele Agenten, die miteinander kommunizieren. Für die Datenauswertung werden „klassische“ Computer und Laptops eingesetzt, während für die Datenerhebung spezielle Mikrocontroller zum Einsatz kommen. Man kann auch nicht mehr davon ausgehen, dass das gesamte System zu 100% zuverlässig funktionieren wird. Man muss mit Ausfällen von Teilsystemen rechnen, wenn z.B. ein Tracker defekt ist oder ein Signal nicht empfangen wird. In der Literatur werden diese Phänomene unter dem Begriff *eventual consistency* (Burckhardt, 2014) studiert.

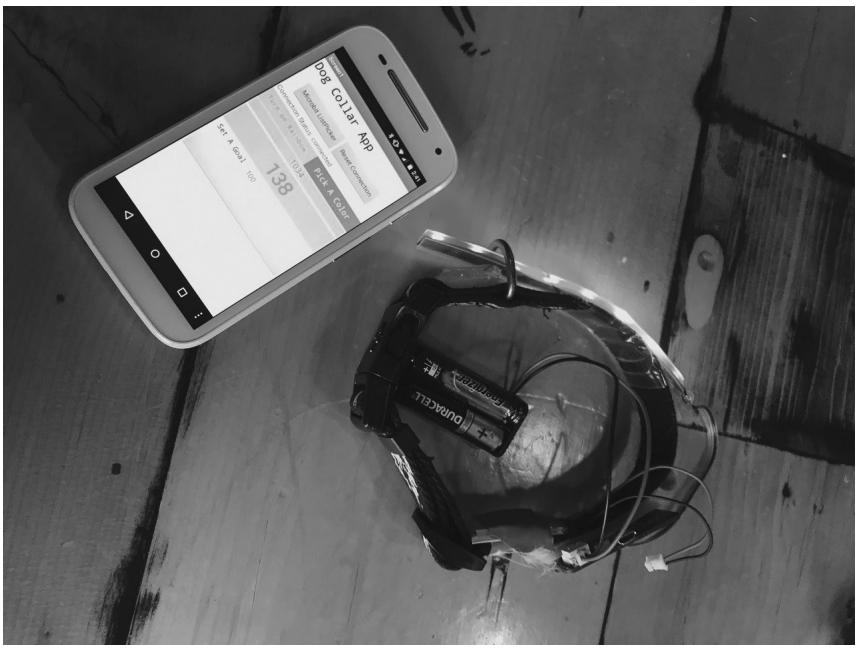


Abbildung 3.14: Realisierung des Hundetrackers aus dem Szenario, Quelle: Shapiro und Tissenbaum (2019)

Wenn wir uns die einzelnen Teile des Systems anschauen, dann können wir eigentlich feststellen, dass diese heute bereits verfügbar und für Schülerinnen zugänglich sind. Die Abbildung 3.14 zeigt eine konkrete Realisierung davon. Allerdings beklagen Shapiro und Tissenbaum (2019), dass es bisher nur sehr wenige Umsetzungen dafür in Schulen existieren. Das sollte nicht verwundern, wenn man bedenkt, wie lange das Schulsystem häufig braucht, um sich an Änderungen anzupassen. Trotzdem steht es jeder Lehrerin in ihrem Unterricht offen, sich auch mit neuen Entwicklungen zu beschäftigen und die Schülerinnen mit inter-

essanten und authentischen Projekten zu motivieren.

Das dritte und letzte Beispiel zeigt über den Rahmen des Unterrichts hinaus. Hier geht es um tatsächlichen Wandel und eine Verbesserung der Lebenswelt. Der Technovation Wettbewerb (`https://www.technovation.org/`) lädt Teilnehmer aus aller Welt ein, ein Projekt zu entwickeln und es vorzustellen. Es ist sehr interessant und inspirierend, sich die Einsendungen anzuschauen. In Abbildung 3.15 ist ein Beispiel gegeben. Korruption ist ein großes Problem in Kenia, v.a. auch im Bereich der Regierung. Um dieses Problem anzugehen, haben die Mädchen die App „Corrution Disrupter“ entwickelt. Diese bietet die Möglichkeit, Korruption zu melden und mit einem Ort zu verknüpfen. Auf einer Karte kann man dann sehen, welche Orte besonders korrupt sind. Das soll Aufmerksamkeit erregen und die Regierung zum Handeln auffordern. Außerdem enthält die App Informationen darüber, was Korruption ist und welche negativen Folgen sie hat.

Abbildung 3.15: Mädchen aus Kenia engagieren sich für sozialen Wandel, Quelle: Nairobi Tech Girls (2020)

Die Beispiele in diesem Abschnitt haben hoffentlich einen ersten Einblick dafür gegeben, was mit vernetzten Geräten alles möglich ist. Das Potenzial ist noch längst nicht ausgeschöpft und es ist zu hoffen, dass dieses Thema auch in der Schule mehr Einzug findet.

3.1.5 Rätsel- und Knobelaufgaben

Es gibt einige Menschen, die sich in der Freizeit gerne mit Rätsel- und Knobelaufgaben beschäftigen. Für solche Menschen kann es sehr motivierend sein, mithilfe solcher Rätsel in die Informatik einzusteigen. Besonders interessante Rätsel in der Informatik sind dabei Aufgabe, wo man etwas *hacken* muss, um an eine Lösung zu gelangen. Diese finden sich häufig in Format von „Capture the Flag“ (Chung und Cohen, 2014) Anwendung. Dabei gibt es eine große Bandbreite an Schwierigkeitsgraden, von relativ einfachen Einstiegsaufgaben bis hin zu extrem schwierigen Herausforderungen.

Es gibt einige Websiten, die solche Aufgaben anbieten, wie z.B. Projekt Euler (`https://projecteuler.net/`), welches sich besonders auf mathematisch-informatische Rätsel fokussiert. Für Aufgaben, die näher am Thema Hacking angelehnt sind, gibt es beispielsweise OverTheWire (`https://overthewire.org/wargames/`). Diese Website ist aber bereits anspruchsvoll und benötigt eine Menge an Vorwissen. Eine Variante davon (entwickelt vom Autor dieser Arbeit) heißt „Hack The Web“ (`https://hack.arrrg.de`) und ist bewusst schülerfreundlich gestaltet: Die Aufgaben sind auf deutsch verfügbar und erlauben einen sehr einfachen Einstieg. Man kann das gut in einer Doppelstunde mit einer Klasse spielen. Die besonders interessierten Schülerinnen können sich dann im Anschluss selbstständig an die komplexeren Aufgaben vorwagen.

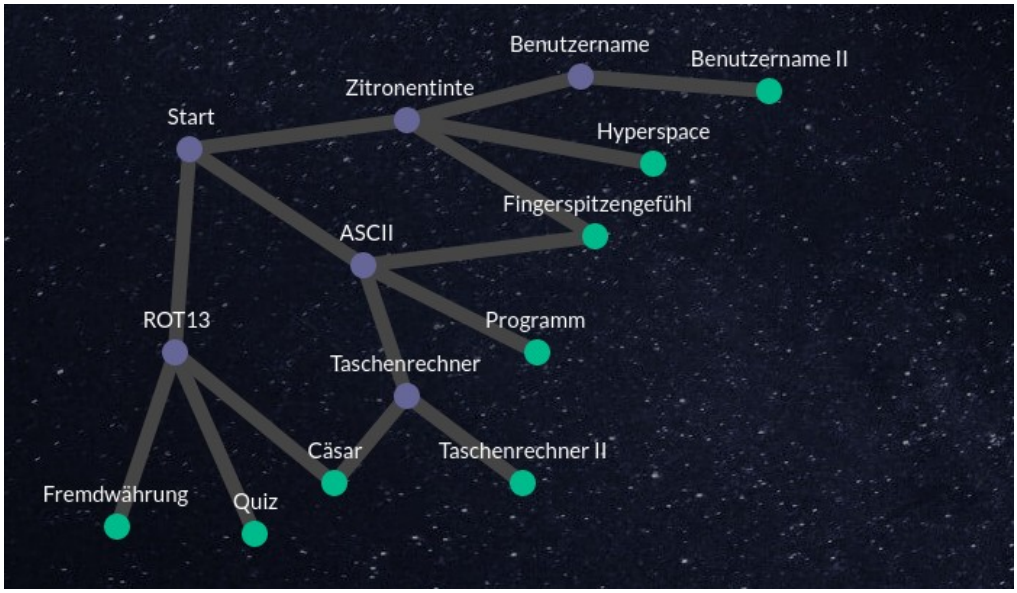


Abbildung 3.16: Karte von Hack The Web, Quelle: `https://hack.arrrg.de`

3.1.6 Computerspiele

Abbildung 3.17: Titelbild von Ozaria, Quelle: `https://www.ozaria.com/`

Computerspiele und Lernen - das mag vielleicht wie ein Gegensatz klingen. Zumal viele Kinder sehr gerne Computerspiele spielen, diese aber wohl nicht freiwillig viel Zeit mit Lernen verbringen würden. Computerspiele sind häufig so gestaltet, dass sie eine große Motivation ausüben können und man kann sich natürlich fragen, ob es nicht möglich ist, diese Motivation zu nutzen und mit ernsthaften Lerninhalten zu verbinden.

Dieser Abschnitt möchte ein aktuelles Projekt vorstellen, dass sich genau mit dieser Frage beschäftigt. In Abbildung 3.17 ist das Titelbild zu sehen von Ozaria (`https://www.ozaria.com/`), einem Spiel zum Programmieren Lernen. Die Website wirbt mit dem Slogan „Computer Science that Captivates“. Das erste Kapitel steht kostenlos zum Testen zur Verfügung. Die Entwickler sind die gleichen wie von CodeCombat (siehe Rosado, 2019), welche seit 2013 an Computerspielen zur Programmieren Lernen arbeiten. Im Folgenden sollen einige Eindrücke, die sich beim Testen des ersten Kapitels von Ozaria gefunden haben, vorgestellt und eingeordnet werden - auch im Vergleich zum Vorgängerprodukt oder anderen Ansätzen.

Zuerstmal beeindruckt die künstlerische Ausgestaltung: Man wird durch animierte Zwischenszenen und gesprochenen Dialogen in eine schön gestaltete Welt

eingeführt. Jedes Element bis hin zur Programmieroberfläche ist in einem einheitlichen Design eingebettet.

Abbildung 3.18: Ein Level von Ozaria, Quelle: `https://www.ozaria.com/`

In Abbildung 3.18 ist eine Programmieraufgabe zu sehen. Dieses unterscheidet sich auf dem ersten Blick wenig von vorherigen Versionen: Man hat rechts den Editor, die Ziele werden oben rechts angezeigt und die Ausführung lässt sich im linken Teil beobachten. Eine Neuerung ist die Zeitleiste unter dem Ausgabefenster: Damit lässt sich der Programmablauf vor- und zurückspulen. Falls man einen Fehler im Programm hat, muss man nicht, wie sonst üblich, jedes Mal das Programm von vorne starten und genau im richtigen Moment pausieren. Stattdessen kann man beliebig in der Ausführung hin und her springen.

Aus didaktischer Sicht setzt Ozaria sehr schnell einen Fokus darauf, größere Programme in Teilschritte aufzuteilen und diese getrennt zu bearbeiten. Ein interessantes Element ist dabei, dass in manchem Leveln spätere Ziele erst sichtbar werden, wenn vorherige Ziele erreicht wurden. Das schafft einen sehr natürlichen Bedarf danach, die Aufgabe schrittweise zu lösen. Schließlich sind die Aufgaben auch abwechslungsreich gestaltet. In vielen Leveln schreibt man Code, aber es gibt auch einige Level, wo man vorhandenen Code korrigieren muss oder einzelne Parameter anpasst. Insgesamt greift Ozaria einige neuere Ideen der Informatikdidaktik auf und schafft im Zusammenspiel mit einer schönen Spielerfahrung eine hohe Motivation, sich mit Programmierung zu befassen.

3.2 Komplexität reduzieren

Wenn wir uns an das Erwartung-mal-Wert-Modell der Motivation zurückerinnern, dann wird deutlich, dass sich der vorherige Abschnitt besonders darauf konzentriert hat, den Wert-Anteil zu steigern. Parallel dazu gibt es noch die Möglichkeit, die Erwartung auf Erfolg zu erhöhen. Mit diesem Aspekt wird sich dieser Abschnitt nun beschäftigen.

Wenn man sich den Vorgang des Programmierens anschaut, z.B. in der Sprache Java, dann lassen sich einige Hürden identifizieren: Zuerst einmal braucht es eine ordentliche Installation der richtigen Entwicklungspakete. Hier fangen die Schwierigkeiten schon an, das richtige Framework auszuwählen. Auch die Wahl des Editors kann herausfordernd sein: Die reduzierteste Art der Programmierung findet über die Konsole statt. Das setzt einen gewissen Grad an Vertrautheit mit den Systemwerkzeugen voraus. Meist wird man sich für eine Entwicklungsumgebung entscheiden. Endlich bei der eigentlichen Programmierung angekommen, muss man in Java viele Konzepte gleichzeitig anwenden, um überhaupt ein lauffähiges Programm zu erhalten: Es braucht Wissen über die Syntax, über die benötigten Bibliotheken zur Ein- und Ausgabe. Außerdem muss das Programm richtig typisiert und gleichzeitig objektorientiert programmiert sein.

Jedes dieser Aspekte bietet Fehlerquellen, die zu Frustration führen können. Erleben Schülerinnen zu viele frustrierende Momente, dann besteht die Gefahr, dass das Programmieren als etwas empfunden wird, dass schwierig ist - und was sich dann durch die gesunkene Erwartung auf Erfolg negativ auf die Motivation auswirkt.

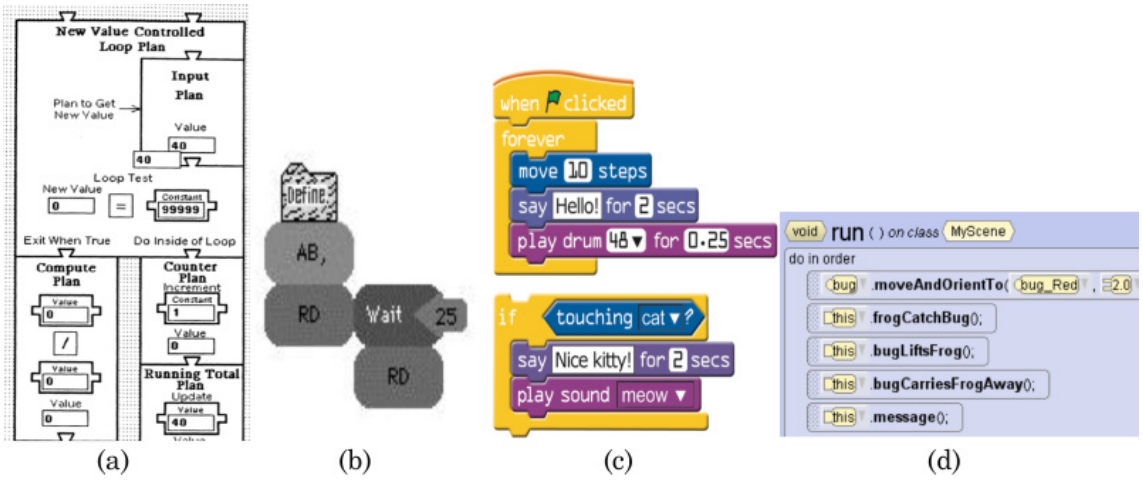
Die Anreize dieser Kategorie versuchen, technischen Fehlerquellen dieser Art möglichst früh aus dem Weg zu räumen und damit eine flüssigere Erfahrung zu bieten. Dabei werden teilweise sehr neue Wege gegangen, z.B. bei der blockbasierten Programmierung, die damit versucht, Syntaxfehler grundsätzlich zu vermeiden. Die gesamte Thematik rund um die Installation wird bei den meisten neueren Lernumgebungen dadurch vermieden, dass diese sofort im Browser lauffähig sind und damit die Installation entfällt. Weitere Ansätze, wie z.B. Minisprachen, versuchen außerdem, die Anzahl der Konzepte, die man für den Einstieg zu lernen braucht, zu senken und damit den *cognitive load* (Sweller, 2011) zu senken.

3.2.1 Blockbasierte Programmierung

Eine der ersten Hürden beim Programmieren lernen sind häufige Syntaxfehler, wie z.B. von Denny u. a. (2011) berichtet. Ein Programm, das nicht syntaktisch korrekt ist, wird nicht kompiliert und ausgeführt. Der Lernende muss zuerst diese Fehler beheben, bevor er eine Rückmeldung bekommt, ob sein Programm auch logisch richtig ist. Drosos, Guo und Parnin (2017) konnten nachweisen, dass Syntaxfehler mit Frustration korrelieren. Eine Lernumgebung, in der man sich keine Sorgen um den Syntax machen muss, sollte daher im Umkehrschluss für weniger Frustration sorgen und den Lernenden mehr Spaß bereiten.

Eine Umsetzung dieser Idee wird durch blockbasierte Programmiersprachen erreicht. Der erste Teil dieses Abschnitts zeigt, wie das gelingt und aus welchen Ideen Blocksprachen entstanden sind. Mittlerweile sind Blocksprachen weitverbreitet und im zweiten Teil seien einige gemeinsame Elemente einer typischen blockbasierten Lernumgebung vorgestellt. Im dritten Teil wird auf die empirische Forschung eingegangen, die einige der Vorteile bestätigen konnte, aber auch Grenzen aufzeigt. Zum Schluss möchte der Autor die Ergebnisse eines kleinen Selbstversuchs vorstellen.

Die Idee, einen Editor zu entwickeln, in dem der Syntax immer korrekt ist, wird bereits in Donzeau-Gouge u. a. (1980) vorgestellt. Der wesentliche Gedanke ist, dasss man als Benutzer nicht länger nur einzelne Zeichen, sondern gleich



(a) (b) (c) (d)

Abbildung 3.19: Historische blockbasierte Programmiersprachen: (a) BridgeTalk, (b) LogoBlocks, (c) Scratch (bis Version 2), (d) Alice, Quelle: Weintrop und Wilensky (2017)

größere Strukturen eingibt. Diese Strukturen sind Teile des Syntaxbaums und dadurch kann der Editor den Benutzer zielgerichteter unterstützen, z.B. indem er anzeigt, an welcher Stelle das Einfügen erlaubt ist und an welcher nicht. Besonders fruchtbar ist dazu die Verbindung mit einer graphischen Repräsentation. Hierbei werden der die Regeln der Syntax durch visuelle Elemente dargestellt und es wird dadurch intuitiver klar, welche Kompositionen erlaubt sind. Abbildung 3.19 zeigt einige historische Entwicklungsschritte. Die kleinsten Bauteile, die der Benutzer eingibt, sind nun keine Zeichen mehr, sondern Blöcke.

Abbildung 3.20: Eine typische blockbasierte Lernumgebung, eigenes Programm auf Code.org, Quelle: `https://studio.code.org/projects/infinity`

Große Universitäten, wie das MIT[8], und große Internetkonzerne haben viel Energie in die Entwicklung von blockbasierten Programmiersprachen und Lernumgebungen gesteckt, von denen einige auch schon in dieser Arbeit vorgestellt wurden. Abbilung 3.20 zeigt nochmal ein sehr typisches Beispiel. Die eigentliche Programmierung findet im Arbeitsbereich rechts statt. Die verfügbaren Blöcke sind im Blockmenü in der Mitte aufgelistet. Von dort aus können die Blöcke in den Arbeitsbereich gezogen werden. Um Blöcke zu löschen, kann man sie aus dem Arbeitsbereich wieder ins Menü ziehen. Die Farben der Blöcke geben an, von welchem Typ der Block ist, z.B. ein Befehl, ein Ereignishandler oder eine Kontrollstruktur. Links sieht man das Ergebnis der Ausführung des Programms.

[8]Projekt Scratch: `https://www.media.mit.edu/projects/scratch/overview/`

In diesem Beispiel begrüßen sich Elsa und Anna und bewegen sich ein wenig auf dem Bildschirm. Hinter diesem Aufbau stecken viele Gedanken, welche zum Beispiel in Fraser (2015) oder Maloney u. a. (2010) erklärt sind.

Die Verfügbarkeit von Blocksprachen hat auch die Forschung angeregt, die postulierten Vorteile zu untersuchen. Weintrop und Wilensky (2015) konnte tatsächlich feststellen, dass Schülerinnen die blockbasierte Programmierung als einfacher empfinden im Vergleich zur textbasierten Programmierung. Sie sagten, die Blöcke seien einfacher zu lesen und leichter zusammenzustellen, außerdem helfen die optischen Marker dabei zu erkennen, welche Kombinationen erlaubt sind. Das Menü erleichtert das Auffinden und reduziert den Merkaufwand. In einer Anschlussstudie stellten Weintrop und Wilensky (2017) auch eine Verbesserung des Lernerfolgs fest. Auch eine aktuelle Meta-Studie von Chiu und Tsuei (2020) stellt fest, dass blockbasierte Programmierung zu größeren Steigerungen in der Problemlösefähigkeit führt. Immer wieder wird auch berichtet, dass Kinder Blocksprachen als freundlicher empfinden (Weintrop und Wilensky, 2017) und weniger Vorbehalte haben. Die Stimmung im Klassenzimmer ist lockerer und die Kinder sind eher ermutigt, Sachen auszuprobieren.

Neben den vielen positiven Befunden haben die Studien auch ein paar Grenzen aufgezeigt. Weintrop und Wilensky (2015) haben ausführliche Interviews mit Schüler:innen durchgeführt. Sie stellten dabei fest, dass die Schülerinnen Blocksprachen als weniger mächtig wahrnehmen, sie die Bearbeitung als langsamer empfinden und teilweise als weniger authentisch sehen. Blockbasierte Programmierung wird als ein ausgezeichnetes Tool für den Einstieg in die Programmierung gesehen, aber für weiterführende Anwendungen ist der Übergang in die textbasierte Programmierung unerlässlich. Fraser (2015) schlägt vor, schon von Anfang an an eine „Exit strategy“ zu denken und sich Konzepte für einen Übergang zu überlegen. Einige Editoren (z.B. Droplet, `https://github.com/droplet-editor/droplet`) bieten mittlerweile die Möglichkeit, zwischen den verschiedenen Modalitäten zu wechseln. Blöcke und Text werden als äquivalente Formen desselben Programms gesehen.

Die Lernumgebungen entwickeln sich ständig weiter und es kommen immer wieder neue Anwendungen auf den Markt. Auch als Lehrerin ist man daher herausgefordert, sich mit neuen Lernumgebungen und Konzepten zu befassen - und manchmal bedeutet das, sich auch mal ins Unbekannte zu stürzen. Das ist vielleicht am Anfang etwas ungewohnt, aber man ist mit den Lernumgebungen gut aufgehoben und findet schnell Spaß am Programmieren.

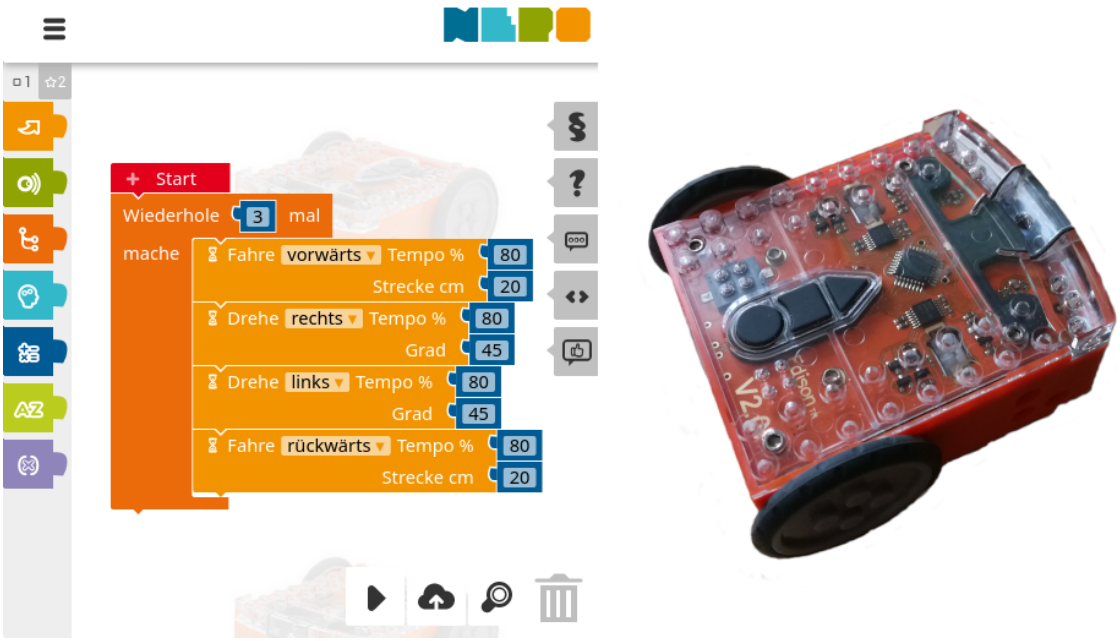


Abbildung 3.21: Den Edison-Roboter programmieren? Ein Selbstversuch mit Open Roberta Lab, Quelle: `https://meetedison.com/`

Für meinen Selbstversuch habe ich einen kleinen Roboter zur Hand genommen, der rechts in Abbildung 3.21 zu sehen ist. Diesen wollte ich mit Open Roberta Lab (`https://lab.open-roberta.org/`) programmieren, welches Unterstützung für diesen Roboter anbietet, womit ich aber noch nie gearbeitet habe. Das Laden der Lernumgebung ging sehr einfach, es ist ausreichend, die Adresse im Browser einzugeben und den Robotertyp auszuwählen. Die Funktionen sind auch über das Menü sehr leicht zu entdecken, alle Funktionen des Roboters - wie z.B. Sensoren, Motoren, Lautsprecher - haben ihre zugehörigen Blöcke. Nach wenigen Minuten ist das erste Programm zusammengeklickt. Auch die Übertragung auf den Roboter ging reibungslos, es genügte, den Anweisungen auf dem Bildschirm zu folgen. Und siehe da: der Roboter setzte sich in Bewegung.

Dieses kleine Beispiel zeigt, dass blockbasierte Sprachen es schaffen, die Barrieren für den Einstieg in die Programmierung zu senken. Das schafft Raum, sich mit anderen Aspekten der Programmierung zu befassen, wie z.B. dem Problemlösen und man erlebt schneller, dass man mit seinem Programm etwas in Bewegung setzt.

3.2.2 Programmieren im Browser

Bei vielen professionellen Programmierumgebungen ist die Installation und Eintrichtung ein großes und aufwendiges Unterfangen - mit zum Teil hohen Systemanforderungen. Das führt auf vielen Ebenen zu Schwierigkeiten: Zum einen verhindert es grundsätzlich den Einstieg in die Programmierung, falls kein passendes System vorhanden ist. Diese Lage ist besonders im Schulunterricht problematisch, weil man dort häufig an die Ausstattung der jeweiligen Schule gebunden ist und an die Verfügbarkeit von Computer bei den Schülerinnen Zuhause. Andererseits führen Fehler und Probleme bei der Installation schon vor der eigentlichen Programmierung zu Frustrationen, die sich schließlich negativ auf die Motivation auswirken können.

Gleichzeitig haben sich in den letzten Jahren die Internet-Browser enorm weiterentwickelt. Auch die Unterstützung über verschiedene Geräte hinweg verbessert sich ständig und damit können interaktive Anwendungen entwickelt werden, die gleichzeitig für Laptops, Smartphones und Tablets geeignet sind - ganz ohne Installation.

Lernumgebungen haben die Vorteile dieses Ansatzes sehr schnell aufgegriffen. Fast alle neueren Entwicklungen (Code.org, MakeCode, ArduinoIDE, ...) sind im Browser lauffähig - damit ist ein Start in die Entwicklung in wenigen Sekunden möglich. Bei älteren Lernumgebungen, die mit einer nativen Technologie gebaut sind, ist die Situation schwieriger. Bemerkswert ist hierbei zum Beispiel die Geschichte von Scratch, wie in DACH-Scratch-Wiki (2020) vorgestellt: In 2007 ist das Projekt entstanden, damals noch nativ in Smalltalk implementiert. Wenige Jahre später kommt die Version 2 heraus, die über das Flash-Plugin nun im Browser lauffähig ist. Allerdings hat das Plugin eine Installation benötigt und ist mittlerweile veraltet. Darauf hat Scratch reagiert und schließlich in der Version 3 in einem großen Unterfangen das gesamte Projekt in HTML5 neu implementiert. Damit ist Scratch nun vollständig im Browser lauffähig - und damit unabhängig von Betriebssystem und sogar Geräteform.

Man sollte aber auch bedenken, dass es trotz der vielen Vorteile auch manche Nachteil gibt: Das Programmieren im Browser setzt eine stabile Internetverbindung voraus. Das kann an sich wieder zu Problemen führen. Daher bietet Scratch 3.0 zum Beispiel weiterhin eine Offline-Version zur Installation an. Außerdem benötigen einige Webanwendungen einen leistungsstarken Server im Hintergrund (z.B. Glitch, `https://glitch.com/`), was wiederum zu Kosten führt.

3.2.3 Minisprachen

Beim Einstieg in die Programmierung wird man vor die Herausforderung gestellt, viele Konzepte gleichzeitig erlernen zu müssen, um überhaupt ein lauffähiges Programm zu erhalten. Das führt oft zu Überforderung. Um die Komplexität am Anfang zu reduzieren, gibt es schon seit längerer Zeit den Ansatz von Minisprachen, siehe Brusilovsky u. a. (1997). Dabei wird eine general purpose language auf eine domänenspezifische Sprache vereinfacht. Mithilfe dieser Minisprache können einzelne Konzepte geübt werden, bevor man sie mit anderen Konzepten verbindet.

Eine Neuauflage dieser Idee findet sich bei Hermans (2020). Hier wird insbesondere ein Vergleich mit dem Schreiben Lernen in der Grundschule gezogen: Auch das Schreiben von Sätzen lernt man in vielen einzelnen Schritten, die jeweils immer mehr Konzepte enthalten. In Abbildung 3.22 sieht man auf der linken Seite vier Versionen eines Satzes und auf der rechten Seite vier Versionen eines Python-Programms. Diese werden in der Lernumgebung Hedy (`https://www.hedycode.com/`) nacheinander den Schülerinnen vorgestellt. Wenn man mit einer einfachen Version vertraut ist, dann kann man zur nächsten Version springen. Das Projekt ist ganz frisch gestartet und wird täglich erweitert. Zum Beispiel sind bereits Übersetzungen für 6 Sprachen, inklusive deutsch, verfügbar. Man kann gespannt sein, in welche Richtung sich Hedy noch entwickeln wird.

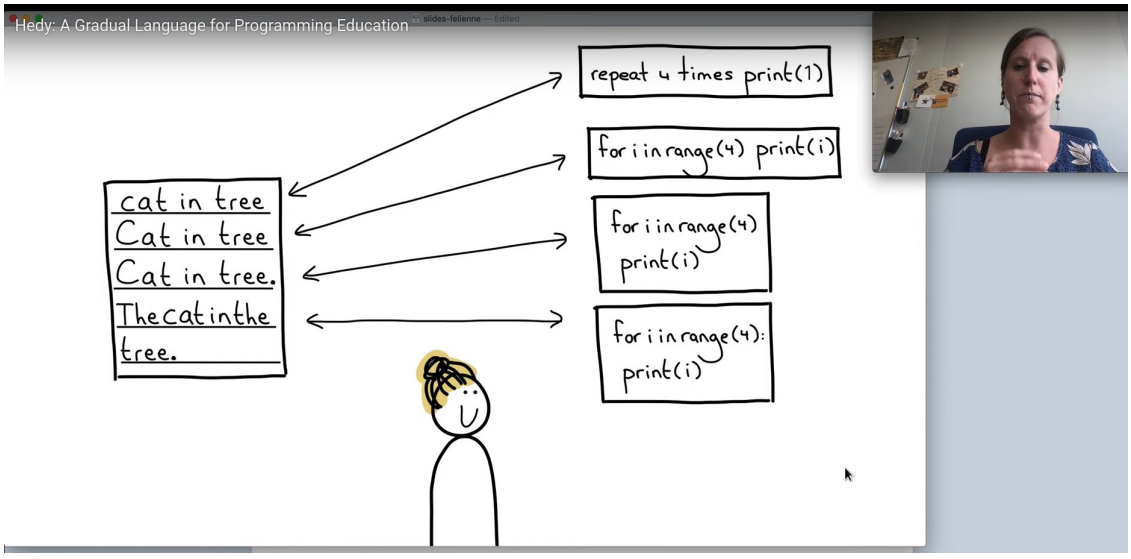


Abbildung 3.22: Vortrag zu Hedy, Quelle: `https://www.youtube.com/watch?v=DRB4ukdM7Cw`

3.3 Unterstützung anbieten

Die Ansätze aus der letzten Kategorie zielen darauf ab, die Programmierung zu vereinfachen, indem Komplexität reduziert wird. Es gibt aber viele Situationen, in denen gerade diese Komplexität Gegenstand des Unterrichts ist. Die blockbasierte Programmierung schafft es zum Beispiel gut, die Komplexität der Syntax zu reduzieren, aber man verpasst dadurch auch die Möglichkeit, Syntax zu lernen.

Die Ansätze dieser Kategorie versuchen also nicht, den inhaltlichen Teil der Programmierung zu verändern. Stattdessen wird versucht, über Unterstützungen die Arbeit zu erleichtern. Solche Unterstützungen sind nicht nur für das Programmieren Lernen von Bedeutung: Besonders auch im professionellen Umfeld finden sich eine große Zahl an Software und Tools, die die Arbeit mit der Programmierung erleichtern.

Die umfangreichsten Projekte dieser Art sind sicherlich professionelle Programmierumgebungen. Diese integrieren eine Vielzahl von Tools, wie z.B. Code-Editor, Compiler, Debugger und Laufzeitumgebung in einer Oberfläche. Idealerweise sind die einzelnen Tools aufeinander abgestimmt und miteinander vernetzt - wie z.B. die Möglichkeit, beim Debuggen den Code zu verändern und das Programm an der gleichen Stelle weiterlaufen zu lassen. Diese Vielzahl von Funktionen sind zwar für den produktiven Alltag sehr wertvoll, können aber Schülerinnen leicht überfordern. So findet sich in Chen und Marx (2005) ein Erfahrungsbericht, wo in den ersten Wochen des Lernens noch auf eine IDE verzichtet wird und diese erst nach und nach eingesetzt wird.

Für die Schule findet sich aber mittlerweile einige Lernumgebungen, die einzelne wichtige Funktionalität aus professionellen Entwicklungsumgebungen umsetzen, aber mit einer stark reduzierten Oberfläche. Ziel ist dabei, die Schülerinnen an den Vorteilen teilhaben zu lassen, ohne sie sofort zu überfordern. Solche Elemente sind zum Beispiel Visualisierungen, Code-Editoren mit Syntaxhervorhebung und Autovervollständigung oder interaktive Laufzeitumgebungen.

Über die Entwicklungsumgebung hinaus können auch weitere Bausteine die Motivation steigern, welche in dieser Kategorie auch nicht unerwähnt bleiben sollen: Gute Dokumentationen oder Videos erleichtern den Wissenserwerb. Es werden auch häufig Übungsaufgaben mit direktem Feedback eingesetzt. Schließlich gibt es Bemühungen, jungen Entwicklern eine Community zu geben, über die sie sich austauschen und vernetzen können und in der sie Hilfe finden.

3.3.1 Visualisierungen

Viele Lernumgebungen geben sich große Mühe, die Programmierung so anschaulich wie möglich zu gestalten. Visualisierungen spielen dabei eine zentrale Rolle, um die Lernenden zu unterstützen und ihnen im wahrsten Sinne des Wortes zu *zeigen*, was sie machen. Anhand von vier ausgewählten Beispielen soll in diesem Abschnitt dargestellt werden, wie das gelingt und welche Möglichkeiten es gibt.

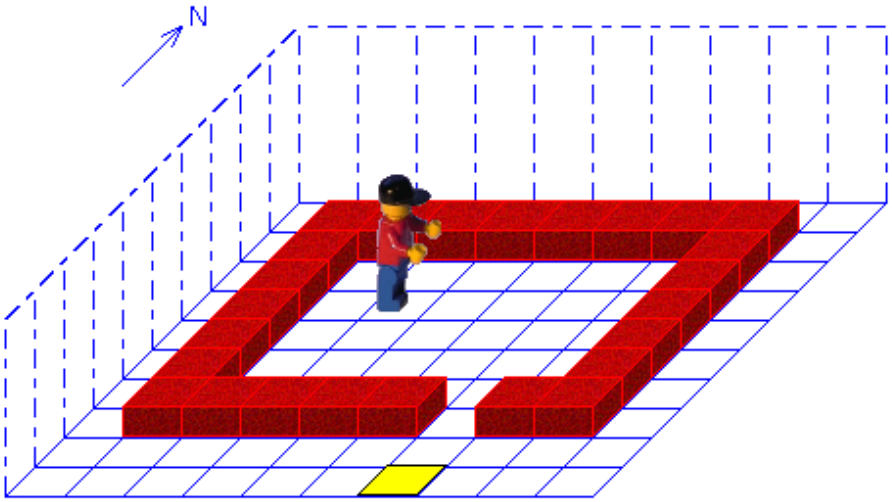


Abbildung 3.23: Robot Karol: Was ist wohl die Aufgabe? Quelle: Freiberger (2018)

Das erste Beispiel in Abbildung 3.23 zeigt eine Welt von Robot Karol (Freiberger, 2018). Ohne große Erklärungen lassen sich sofort einige Dinge bemerken: Karol befindet sich in einer Art „Raum“, aus dem es einen Ausgang gibt. Dieser Ausgang ist gelb markiert. Man kann vermuten, dass hier nach einem Programm gefragt ist, das Karol aus dem Raum hinausführt. Wir können noch nicht sagen, ob das Programm dabei nur für diesen Raum funktionieren soll oder allgemein für alle Räume. Auch ist noch nicht ersichtlich, wie Karol sich bewegen und die Welt wahrnehmen kann. Aber durch die Visualisierung der Welt alleine bekommen Schülerinnen schon ein ganz gutes Gefühl dafür, was sie hier erwartet. Auch können schon erste Überlegungen zur Strategie angestellt werden, z.B. sich an der Wand entlang zu tasten.

Robot Karol zeichnet sich dadurch aus, dass die Visualisierung ein sehr zentraler Aspekt der Lernumgebung ist. Über die Elemente auf dem Spielfeld hin-

aus gibt es im Programm keine weiteren Zustände und Variablen (bis auf den Call Stack). Man hat damit tatsächlich den gesamten Systemzustand im Blick und kann sich somit auf die Aufgabenstellung konzentrieren.

Kasurinen, Purmonen und Nikula (2008) stellten mit einer ähnlichen Visualisierung eine gesteigerte Motivation fest. Eine Steigerung der positiven Einstellung und der Vorstellungsfähigkeit kann laut Wu, Tseng und Huang (2008) erreicht werden, wenn man statt einer Roboter-Simulation einen realen Roboter einsetzt. Allerdings besteht beim Lernerfolg kein Unterschied.

Im zweiten Beispiel wird die Visualisierung eingesetzt, um das Konzept der Objektorientierung zu veranschaulichen. Hierbei betrachten wir die Lernumgebung BlueJ (Kölling u. a., 2003). Die Abbildung 3.24 zeigt auf der linken Seite das Hauptfenster von BlueJ und auf der rechten Seite einen Ausschnitt des Codes.

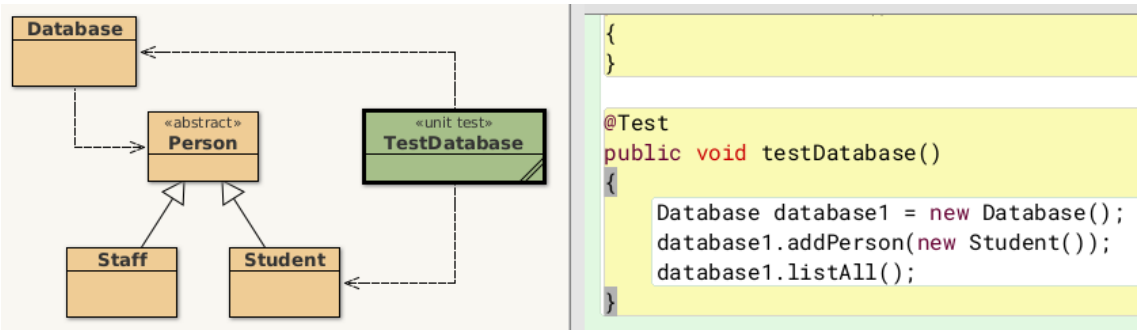


Abbildung 3.24: Dynamisches UML-Diagramm in BlueJ
Quelle: `https://www.bluej.org/`

Die Besonderheit hierbei liegt darin, dass im Hauptfenster die Klassenstruktur des Projekts als UML-Diagramm darstellt wird. Man kann auf den Quellcode der einzelnen Klassen zugreifen, indem man im Diagramm einen Doppelklick macht. Zusätzlich werden im Diagramm die Assoziationen und Vererbungen angezeigt. In diesem Beispiel ist zu sehen, dass Person eine Oberklasse zu Staff und Student ist. Die Database hat eine Referenz auf die Klasse Person und die TestDatabase bezieht sich auf Database und Student. Bemerkenswert ist, dass die Informationen des UML-Diagramms automatisch aus dem Quellcode ausgelesen werden. Die Eingabe des Code wie auf der rechten Seite der Abbildung genügt, dass die Lernumgebung die korrekten Beziehungen erfassen und darstellen kann.

Eine Evaluation von BlueJ durch Van Haaster und Hagan (2004) zeigt durchaus positive Ergebnisse, wobei aber durch das Fehlen einer Kontrollgruppe keine vergleichenden Aussagen getroffen werden können.

Wir kommen nun zum dritten Beispiel. Der Fokus hier liegt nun bei der Programmausführung. Der Artikel von Sorva (2013) bietet einen Überblick über die Herausforderungen, die Anfänger mit der Programmierung haben. Dabei wird vor allem auch deutlich, dass die Vorstellung einer geeigneten *notional machine* wesentlich ist bei der Erlernung der Programmiersprache. Die notional machine beschreibt eine abstrakte Vorstellung davon, wie der Computer ein Programm ausführt. Solche Maschinen unterscheiden sich von Programmiersprache zu Programmiersprache und werden häufig nur implizit behandelt. Sorva argumentiert, dass es sich lohnt, das Konzept der notional machine explizit im Unterricht zu behandeln.

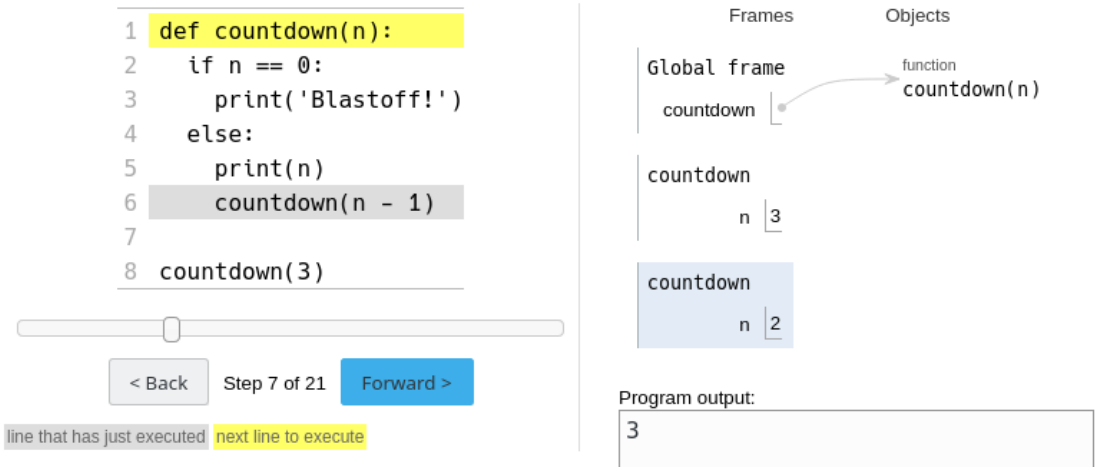


Abbildung 3.25: Visuelle Ausführung eines Programms in Python Tutor
Quelle: `https://cscircles.cemc.uwaterloo.ca/16-de/`

Eine Implementierung dieses Ansatzes durch Python Tutor (Guo, 2013) ist in Abbildung 3.25 zu sehen. Links ist ein Programm in Python zu sehen, rechts daneben eine Visualisierung der Ausführung. Diese Visualisierung hebt zwei Aspekte des Interpreters hervor, die meist nur implizit behandelt werden: Den Call-Stack und den Heap. Jeder Funktionsaufruf erhält einen eigenen Satz an Variablen, der in einer sog. Frame auf dem Call-Stack gespeichert wird. Variablen enthalten primitive Typen, die direkt gespeichert sind, oder Referenzen auf Objekte, die im Heap liegen. Diese zwei Teile des Interpreters zu kennen ist wichtig, um Konzepte wie Mutation von Objekten oder Rekursion zu verstehen. Die Abbildung zeigt ein Beispiel für eine Rekursion. Man sieht, dass gerade der zweite rekursive Aufruf stattgefunden hat. Man kann außerdem nach Belieben in der Ausführung des Programms vorwärts oder rückwärts gehen. Das Hilfsdoku-

ment zu Python Tutor[9] weist aber darauf hin, dass sich dieses Tool besonders für kleine, exemplarische Programme eignet und damit keine vollständige Entwicklungsumgebung ersetzt.

Visualisierungen zur Programmausführung sind meist für den Anfang des Tertiärbereichs entwickelt worden, können aber durchaus auch in der zweiten Hälfte der Sekundarstufe eingesetzt werden. Hier besteht noch Potenzial, die Werkzeuge der Universitätslehre an den Unterricht in der Schule anzupassen.

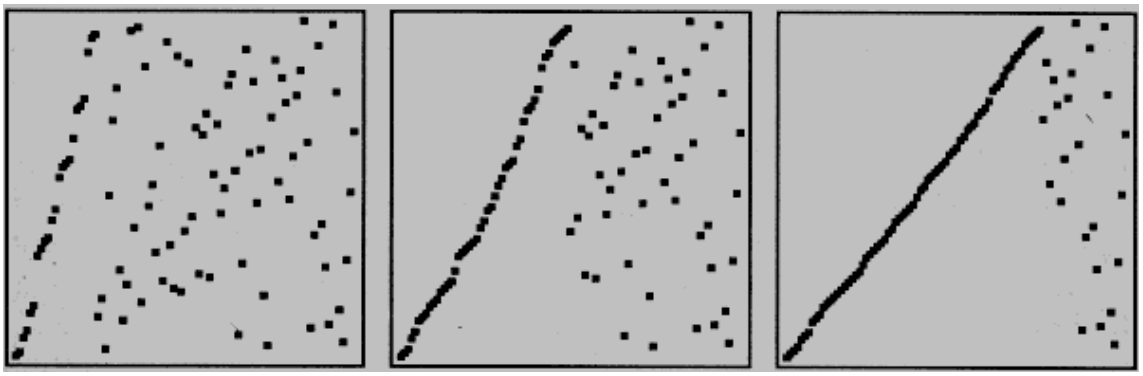


Abbildung 3.26: Visualisierung von Insertion Sort, Quelle: Sedgewick (1992)

Wir kommen zum letzten Beispiel einer Visualisierung. Diese geht in den meisten Fällen über den Inhalt der Sekundarstufe hinaus, ist aber sehr gut geeignet, um Anreize zu schaffen und einen Einblick in die Informatik zu geben. Die Abbildung 3.26 zeigt eine solche Visualisierung. Auch hier lassen sich Muster erkennen: Die Punkte sind am Anfang zufällig auf dem Feld verteilt, werden aber nach rechts hin immer mehr sortiert, bis sie zum Schluss eine gerade Linie bilden. Solche Bilder können in der Klasse als Startpunkte von Diskussionen verwendet werden. Man könnte zum Beispiel fragen, ob den Schülerinnen eine Struktur auffällt (die Punkte werden von links nach rechts sortierter). Zusammen mit Visualisierungen von anderen Sortieralgorithmen können grundsätzliche Vergleiche getätigt werden - ohne gleich den Algorithmus im Detail besprechen zu müssen. Dieser Zugang zu selbst komplexen Themen ist ein besonders Potenzial von Visualisierungen.

[9] Python Tutur unsupported features: `https://docs.google.com/document/d/13_Bc-l2FKMgwPx4dZb0sv7eMfYMHhRVgBRShha8kgbU`

3.3.2 Angepasster Editor

Beim Einsatz einer textbasierten Sprache ist der Editor die wichtigste Schnittstelle. Im professionellem Umfeld gibt es eine Vielzahl an Funktionen, die darauf abzielen, die Produktivität zu steigern. Die gleichen Werkzeuge sind auch in der Lage, Schülerinnen beim Einstieg in die Programmierung zu unterstützen und damit eine positive Wirkung auf die Motivation zu entfalten.

Dieser Abschnitt möchte einige wichtige Funktionen anhand des Editors von BlueJ vorstellen. In 3.27 ist eine Bildschirmaufnahme zu sehen. Darin lassen sich bereits einige Funktionen erkennen. Als erstes hilft es, wenn der Editor über eine sprachspezifische Syntaxhervorhebung verfügt. Diese stellt Schlüsselwerte in verschiedenen Farben dar, z.B. werden primitive Datentyp wie `int` in rot dargestellt. BlueJ geht darüber hinaus und hinterlegt Code-Blöcke jeweils mit einer Farbe um die Programmstruktur noch weiter hervorzuheben. Darüber hinaus bietet der Editor bei der Bearbeitung eine Autovervollständigung an. Diese erkennt am Kontext, welche Ergänzungen sinnvoll sind und zeigt sogar die Dokumentation an. Bei der Eingabe wird außerdem der Code im Hintergrund gleichzeitig immer wieder kompiliert. Dabei entstehende Fehler müssen nicht manuell überprüft werden, sondern werden gleich im Editor an der entsprechenden Stelle rot unterringelt. Das erleichtert noch einmal die Fehlersuche.

Es sei noch ein wichtiger Aspekt erwähnt: Es gibt natürlich noch ganz viele weitere Funktionen, die man häufig bei professionellen Entwicklungsumgebungen findet. Es gebietet sich aber, sich auf eine bestimmte Auswahl zu fokusieren, um die Schülerinnen nicht zu überfordern. Nähere Informationen, auch zu den Designkriterien von BlueJ, siehe Kölling (2015).

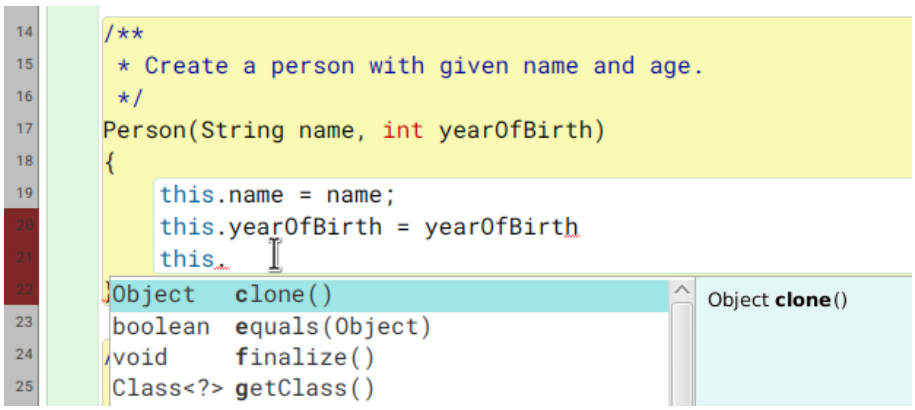


Abbildung 3.27: Angepasster Editor für Java in BlueJ

3.3.3 Interaktive Tutorials

Beim Lernen findet sich in der Literatur öfters der Gedanke, dass ein wesentlicher Baustein dafür Feedback ist, wie z.B. in Askew (2004) dargestellt. Auch intuitiv ist das einleuchtend: Mit einer Rückmeldung kann man besser einschätzen, ob man ein Thema gut verstanden hat und wo es noch Lücken gibt. Durch die Möglichkeiten von Software ist es heute gleichzeitig viel einfacher geworden, interaktive Anwendungen zu schreiben, die den Schülerinnen sehr viel Feedback geben können. In Abbildung 3.28 ist eine Übungsaufgabe eines solchen interaktiven Tutorials zu sehen.

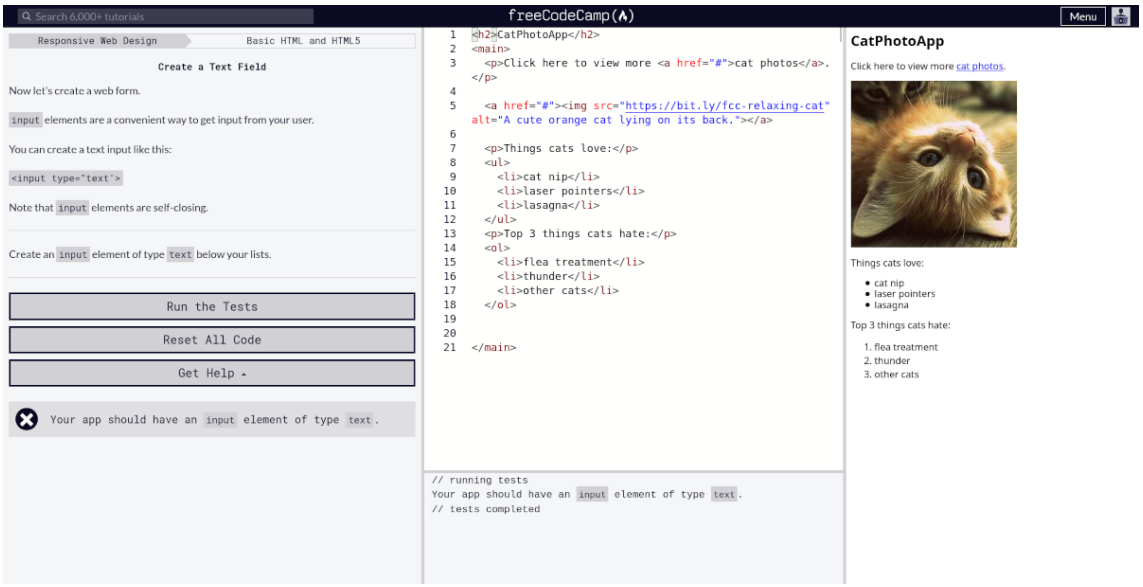


Abbildung 3.28: Eine Aufgabe auf freeCodeCamp, Quelle: `https://www.freecodecamp.com/`

Es gibt viele Anbieter für interaktive Tutorials, der Grundaufbau ist aber immer ähnlich: Man arbeitet sich im wesentlichen durch eine Abfolge von Aufgaben durch. Jede Aufgabe besitzt eine Reihe von Tests, die zum Absolvieren der Aufgabe erfüllt werden müssen. Meist ist schon ein Codegerüst vorgegeben, das dann noch angepasst werden muss. Je nach Anbieter sind die Fehlermeldung mehr oder weniger detailliert. Nach Erfüllung aller Tests kann man mit der nächsten Aufgabe weitermachen. In Malmi, Utting und Ko (2019) werden solche Tutorials als *E-Books* bezeichnet. Die Autoren merken allerdings an, dass es nur sehr wenig Forschung über die Wirksamkeit solcher Tutorials im Vergleich zu anderen Lernmethoden gibt.

3.3.4 Communities

Programmierung findet selten in Isolation statt, sondern man steht ständig mit verschiedenen Personen in direkter oder indirekter Verbindungen: Seien es die Nutzerinnen des Programms, die Entwicklerinnen einer eingesetzten Programmbibliothek oder andere Programmiererinnen, die vor den gleichen Problemen stehen. Viele dieser Verbindungen sind zwar privater Art, aber es gibt auch einige Plattformen, die solche Verbindungen nach außen sichtbar machen und damit Orte schaffen, an denen ein gemeinsamer Austausch und Zusammenarbeit stattfinden kann.

Solche Orte sind aus professionellen Umgebungen nicht mehr wegzudenken. Sei es etwa die Website Stack Overflow (`https://stackoverflow.com/`), welche sehr häufig eine der ersten Anlaufstellen bei Problemen ist, oder GitHub (`https://github.com/`), wo man sein Projekt und die Aufgaben verwalten und diskutieren kann. Eine Community, die auf den Bedarf von Schülerinnnen angepasst ist, findet sich auf der Website von Scratch (`https://scratch.mit.edu/`). Dort können Schülerinnen Projekte erstellen und teilen. Außerdem sind sie in der Lage, in andere Projekte reinzuschauen und diese zu „Remixen“, was nützlich ist, um neue Dinge zu lernen oder miteinander zusammen zu arbeiten, siehe Kafai und Burke (2014) für eine ausführliche Vorstellung.

Durch die organische und öffentliche Struktur von solchen Plattformen kann es schwierig sein, diese in den normalen Unterricht zu integrieren. Trotzdem lohnt es sich als Lehrerin, verschiedene Plattformen zu kennen, um gegebenenfalls interessierte Schülerinnen weitervermitteln zu können.

KAPITEL 4

Fazit

Wir sind nun am Ende dieser Ausarbeitung angekommen. Die vielen Ideen und Ansätze des letzten Kapitels geben eine erste Grundlage, um darauf aufbauend einen Unterricht zu gestalten, der sowohl inhaltliche als auch motivationale Aspekte beachtet. An dieser Stelle soll abschließend darüber nachgedacht werden, wieso es sich lohnt, Energie in die Motivation der Schülerinnen zu investieren. Was sind nun die Vorteile eines Einstiegs in die Programmierung mit vielen Anreizen? In diesem Kapitel sollen drei große Vorteile nochmal vorgestellt und erläutert werden.

4.1 Ein Unterricht, der Spaß macht

Bei einem Unterricht, der viele Anreize setzt, kann man viel eher davon ausgehen, dass dieser Spaß macht - sowohl für die Schülerinnen, als auch für die Lehrerin. Es gibt so viele Möglichkeiten, interessante und witzige Momente in den Unterricht einzubauen. Das tut nicht nur dem Lernerfolg gut, wie einige zitierte Studien zeigen, sondern schafft auch eine Atmosphäre, in der man als Lehrerin gerne unterrichtet. Dieser Aspekt ist nicht zu vernachlässigen, denn auch als Lehrerin braucht man Motivation für die tägliche Arbeit. Die aktive Beteiligung von Schülerinnen ist dafür ein starker Motivator und durch die Beteiligung entwickelt sich auch der Unterricht zu etwas Lebendigen. Man darf ja auch nicht vergessen,

dann man sich meist über ein ganzes Schuljahr regelmäßig sieht und dabei auch viel Zeit gemeinsam im Unterricht verbringt. Die Aussicht auf einen lebendigen Unterricht ist es sicherlich Wert, sich mit dem einen oder anderen Anreiz vertraut zu machen und ihn in den eigenen Unterricht einzubauen.

4.2 Die eigene Neugier erhalten

Es gibt weit mehr Ideen, als sie in dieser Arbeit vorgestellt worden sind. Damit ist auch die Einladung verbunden, sich selbst auf die Suche zu machen und nach weiteren Ideen zu suchen. Denn wenn man sich längere Zeit immer wieder mit demselben Thema beschäftigt, dann kann das ermüden. Immer wieder neue Ideen auszuprobieren macht nicht nur den Unterricht interessant, sondern hilft auch, als Lehrerin interessiert und neugierig zu bleiben. Viele Ideen in dieser Arbeit haben einen gewissen Überraschungseffekt. Viele Projekte stecken auch mitten in der Entwicklung. Man kann davon ausgehen, dass sich in den nächsten Jahren noch einiges entwickeln wird. Wenn man dieser Entwicklung aktiv begegnet, dann wird man kaum auf die Idee kommen, sie als Bedrohung zu sehen, sondern als eine Chance neue Dinge zu entdecken.

4.3 Für eine langfristige Wirkung

Die ständige Entwicklung in der Informatik bedeutet auch, dass Wissen sehr schnell veraltet. In dem Moment, wo die Schülerinnen ins Berufsleben einsteigen, werden sich die Programmiersprachen und Systeme wieder in eine neue Generation entwickelt haben. Faktisches Wissen, wie z.B. die Syntax einer bestimmten Version einer Programmiersprache, hält sich nur kurz. Die Vermittlung von Motivation und Begeisterung dagegen kann eine viel langfristigere Wirkung haben. Eine Einstellung, wie „Programmieren macht Spaß“ oder „Ich kann Sachen mit Software bauen“, halten sich über eine lange Zeit. Es kann der Anstoß sein, sich schon während der Schulzeit intensiver mit Informatik und Programmierung zu beschäftigen, oder sogar für eine bestimmte Studienwahl. Es kann aber auch Jahre später das Selbstbewusstsein geben, nebenbei noch ein wenig zu programmieren und zu gestalten. Auf jedem Fall ist das ein Ziel, für das sich die ganze Vorbereitung und Überlegung lohnt und auszahlt.

KAPITEL 5

Ausblick

Diese Arbeit hat einen klaren Fokus gehabt: Es ging um den Einstieg in die Programmierung und wie man diesen Einstieg - meist mithilfe geeigneter Lernumgebungen - mit Anreizen gestalten kann. Am Ende soll dieser Fokus nun etwas geweitet und noch um einige andere Themen ergänzt werden. Diese Themen können als Ausgangspunkt für ganz neue Ideen und Ansätze dienen.

5.1 Software verändert das Lernen

Die Digitalisierung verändert nicht nur den Informatikunterricht, sondern betrifft eigentlich jedes Fach in der Schule. Die Möglichkeiten sind weit gefasst und viele der Ideen und Ansätze in dieser Arbeit können auch in anderen Fächern eingesetzt werden. Geschichten zu schreiben und zu erzählen ist ein Thema des Deutschunterrichts. Dieses kann leicht mit Animationen und Bilder zum Leben erweckt werden, indem man zum Beispiel Scratch nutzt. Viele Grundkonzepte aus der Physik, wie der freie Fall oder die Gravitation, lassen sich mit wenigen Zeilen Code ebenfalls in Scratch als kleine Simulationen implementieren. Das ermöglicht einen ganz anderen Zugang zu den Gesetzmäßigkeiten als ein Unterricht, in dem nur die Formeln vorgestellt werden. Ebenso ist ein Tool wie Proceessing perfekt geeignet, um im Kunstunterricht eingesetzt zu werden.

Wenn man den Blick noch weiter öffnet, dann finden man noch viel mehr

Software, die den Lernprozess begleiten können: Seien es mathematische Tools wie WolframAlpha (`https://www.wolframalpha.com/`), oder verschiedene Sprachlernapps, oder die Leselernapp für Grundschüler: Sie nutzen alle das Potenzial, das Software zu bieten hat, wie multimediale Inhalte, sofortiges Feedback oder gezielte Vorschläge. Wir können davon ausgehen, dass wir noch längst nicht das volle Potenzial ausgeschöpft haben. In den nächsten Jahren werden uns sicherlich noch viele neue innovative Projekte begegnen.

5.2 Blick auf die Motivation schärfen

Um Motivation zu schaffen ist der Einsatz von Software nicht der einzige Weg. Beim Unterrichten hat man daneben noch viele weitere Methoden zur Hand, Inhalte mit Spaß und Begeisterung zu vermitteln. Programmieren muss nicht immer am Computer stattfinden. Hinter dem Begriff „Unplugged Programming“ (Nishida u. a., 2009) findet sich eine ganze Reihe von Unterrichtsentwürfen, die grundlegende Konzepte der Informatik vermitteln, ohne die Schülerinnen vor den Bildschirm zu setzen. Diese können als begleitende Inhalte eingesetzt werden. Es ist auch nicht ideal, wenn Schülerinnen die gesamte Zeit alleine arbeiten. Der Einsatz von Pair-Programming (Hanks u. a., 2011) kann hier Abhilfe schaffen.

Anreize zu Schaffen ist ein allgemeines Thema, das in allen Fächern eine Rolle spielt. Vielleicht besonders akut ist das im Falle des Fachs Mathematik. Viele Konzepte der Mathematik werden als kompliziert und befremdlich empfunden - die Situation ist gar nicht so unähnlich wie zur Programmierung. Hier können die Ideen aus dieser Arbeit als Ausgangspunkt diesen, auch in solchen Fächern Interesse zu schaffen, die Komplexität zu reduzieren oder Unterstützung anzubieten. In der Mathematik bietet es sich außerdem an, nach interessanten Fragestellungen zu suchen und darüber gemeinsam zu reden - dieses Konzept, mit weiteren Elementen ausgebaut, ist als Peer Instruction (Mazur, 2013) bekannt und konnte bereits erfolgreich in naturwissenschaftlichen Studiengängen eingesetzt werden.

5.3 Auf dem Weg zu neuen Unterrichtsformen

Bereits Papert hatte eine Vision des Unterrichts, in dem Schülerinnen aus eigenen Antrieb lernen und nicht durch das System Schule kaputt gemacht werden. Man muss dieser Haltung in dieser Radikalität nicht so zustimmen, aber es ist

gut, darüber nachzudenken, ob nicht etwas anderes möglich ist als wir heute in den Schulen sehen. Die neuen technischen Möglichkeiten und bewährte pädagogische Konzepte können zusammen den Ausschlag geben für einen Unterricht, der nicht nach dem Prinzip des Nürberger Trichters funktioniert, wo also die Lehrerin den Stoff die Schüler:innen „presst“, sondern bei der die Schülerinnen die Leitrolle beim Lernen übernehmen. Am Ende ist die Lehrerin nicht mehr Quelle des Wissens, sondern eine Begleiterin und Beraterin für das Lernen der Schülerinnen, das durch die vielen Anreize weitgehend selbstständig und motiviert stattfindet.

Literatur

aliceabs (2020). *Wash your hands! Timer zum Händewaschen fürs Calliope mini*. Hackster.io. URL: https://www.hackster.io/aliceabs/wash-your-hands-timer-zum-handewaschen-furs-calliope-mini-c0270b.

Araujo, Luis Gustavo J., Roberto A. Bittencourt und David M. B. Santos (2018). „An analysis of a media-based approach to teach programming to middle school students". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, S. 1005–1010.

Askew, Susan (2004). *Feedback for learning*. Routledge.

Bach, Katja (2017). *34C3 - Digitale Bildung in der Schule*. media.ccc.de. URL: https://www.youtube.com/watch?v=JsYyxr31SZc.

Badamasi, Yusuf Abdullahi (2014). „The working principle of an Arduino". In: *2014 11th international conference on electronics, computer and computation (ICECCO)*. IEEE, S. 1–4.

Bakar, Marini Abu, Muriati Mukhtar und Fariza Khalid (2020). „The Effect of Turtle Graphics Approach on Students' Motivation to Learn Programming: A Case Study in a Malaysian University". In: *International Journal of Information and Education Technology* 10.4.

Bergin, Susan und Ronan Reilly (2005). „The influence of motivation and comfort-level on learning to program". In:

Brusilovsky, Peter u. a. (1997). „Mini-languages: a way to learn programming principles". In: *Education and information technologies* 2.1, S. 65–83.

Burckhardt, Sebastian (2014). „Principles of eventual consistency". In:

Chen, Zhixiong und Delia Marx (2005). „Experiences with Eclipse IDE in programming courses“. In: *Journal of Computing Sciences in Colleges* 21.2, S. 104–112.

Chiu, Jen-I und Mengping Tsuei (2020). „Meta-Analysis of Children's Learning Outcomes in Block-Based Programming Courses“. In: *International Conference on Human-Computer Interaction*. Springer, S. 259–266.

Chung, Kevin und Julian Cohen (2014). „Learning obstacles in the capture the flag model“. In: *2014 {USENIX} Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*.

Cooper, Stephen, Wanda Dann und Randy Pausch (2000). „Alice: a 3-D tool for introductory programming concepts“. In: *Journal of computing sciences in colleges* 15.5, S. 107–116.

Csikszentmihalyi, Mihaly (1997). *Finding flow*. Bd. 131. New York: Basic Books.

DACH-Scratch-Wiki (2020). *Artikel zu Scratch*. Lifelong-Kindergarten-Group, MIT. URL: `https://de.scratch-wiki.info/wiki/Scratch` (besucht am 28. 02. 2021).

Denny, Paul u. a. (2011). „Understanding the syntax barrier for novices“. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, S. 208–212.

Donzeau-Gouge, Véronique u. a. (1980). *Programming environments based on structured editors: The MENTOR experience*. Techn. Ber. INRIA.

Dougherty, Dale (2012). „The maker movement“. In: *Innovations: Technology, governance, globalization* 7.3, S. 11–14.

Drosos, Ian, Philip J. Guo und Chris Parnin (2017). „HappyFace: Identifying and predicting frustrating obstacles for learning programming at scale“. In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, S. 171–179.

Fraser, Neil (2015). „Ten things we've learned from Blockly“. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE, S. 49–50.

Freiberger, Ulli (2002). *Karel: Eine Übersicht über verschiedene Entwicklungen, die auf der Idee von „Karel, the Robot“ basieren*. mebis – Landesmedienzentrum Bayern. URL: `https://mebis.bayern.de/wp-content/uploads/sites/2/2015/05/uebersicht.pdf` (besucht am 23. 11. 2020).

Freiberger, Ulli (2018). *Robot Karol*. mebis – Landesmedienzentrum Bayern. URL: `https://www.mebis.bayern.de/p/17864` (besucht am 23. 11. 2020).

Frey, Bruno S. u. a. (2012). „Crowding out and crowding in of intrinsic preferences“. In: *Reflexive governance for global public goods* 75, S. 78.

Frey, Bruno S. und Felix Oberholzer-Gee (1997). „The cost of price incentives: An empirical analysis of motivation crowding-out“. In: *The American economic review* 87.4, S. 746–755.

Greenberg, Ira, Deepak Kumar und Dianna Xu (2012). „Creative coding and visual portfolios for CS1“. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, S. 247–252.

Guo, Philip J. (2013). „Online python tutor: embeddable web-based program visualization for cs education“. In: *Proceeding of the 44th ACM technical symposium on Computer science education*, S. 579–584.

Guo, Philip J. (2017). „Older adults learning computer programming: motivations, frustrations, and design opportunities“. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, S. 7070–7083.

Halverson, Erica Rosenfeld und Kimberly Sheridan (2014). „The maker movement in education“. In: *Harvard educational review* 84.4, S. 495–504.

Hanks, Brian u. a. (2011). „Pair programming in education: A literature review“. In: *Computer Science Education* 21.2, S. 135–173.

Harvey, Paul und Mark J. Martinko (2009). „Attribution theory and motivation“. In: *Organizational behavior, theory and design in health care*, S. 143–158.

Heckhausen, Heinz (1963). *Hoffnung und Furcht in der Leistungsmotivation*. Bd. 6. Hain.

Heckhausen, Heinz (1984). „Attributionsmuster für Leistungsergebnisse – Individuelle Unterschiede, mögliche Arten und deren Genese“. In: *Metakognition, Motivation und Lernen*, S. 133–164.

Heckhausen, Jutta und Heinz Heckhausen (2010). *Motivation und Handeln (4., überarbeitete und erweiterte Auflage)*.

Hennessey, Beth u. a. (2015). „Extrinsic and intrinsic motivation“. In: *Wiley encyclopedia of management*, S. 1–4.

Hermans, Felienne (2020). „Hedy: A Gradual Language for Programming Education“. In: *Proceedings of the 2020 ACM Conference on International Computing Education Research*, S. 259–270.

Hidi, Suzanne und K. Ann Renninger (2006). „The four-phase model of interest development“. In: *Educational psychologist* 41.2, S. 111–127.

Jenkins, Tony (2001). „Teaching programming–A journey from teacher to motivator“. In: *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*.

Kafai, Yasmin B. und Quinn Burke (2014). *Connected code: Why children need to learn programming*. Mit Press.

Kasurinen, Jussi, Mika Purmonen und Uolevi Nikula (2008). „A Study of Visualization in Introductory Programming.“ In: *Ppig*, S. 19.

Kelleher, Caitlin und Randy Pausch (2005). „Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers“. In: *ACM Computing Surveys (CSUR)* 37.2, S. 83–137.

Kelleher, Caitlin, Randy Pausch und Sara Kiesler (2007). „Storytelling alice motivates middle school girls to learn computer programming“. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, S. 1455–1464.

Kinnunen, Päivi und Lauri Malmi (2006). „Why students drop out CS1 course?“ In: *Proceedings of the second international workshop on Computing education research*, S. 97–108.

Kölling, Michael (2015). „Lessons from the design of three educational programming environments: Blue, BlueJ and Greenfoot“. In: *International Journal of People-Oriented Programming (IJPOP)* 4.1, S. 5–32.

Kölling, Michael u. a. (2003). „The BlueJ system and its pedagogy“. In: *Computer Science Education* 13.4, S. 249–268.

Krugel, Johannes und Alexander Ruf (2020). „Learners' perspectives on block-based programming environments: code. org vs. scratch“. In: *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, S. 1–2.

Langer, Andrea, Nadine Bergner und Ulrik Schroeder (2019). „Mehr als Programmieren lernen - einen Mikrocontroller-Roboter zum Leben erwecken“. In: *Informatik für alle*. Hrsg. von Arno Pasternak. Bonn: Gesellschaft für Informatik, S. 380. DOI: 10.18420/infos2019-f8.

Malle, Bertram F. (2011). „Attribution theories: How people make sense of behavior“. In: *Theories in social psychology* 23, S. 72–95.

Malmi, Lauri, Ian Utting und Andrew J. Ko (2019). „Tools and Environments“. In: *The Cambridge Handbook of Computing Education Research*. Hrsg. von Sally A. Fincher und Anthony V.Editors Robins. Cambridge Handbooks in Psychology. Cambridge University Press, S. 639–662. DOI: 10.1017/9781108654555.022.

Maloney, John u. a. (2010). „The scratch programming language and environment“. In: *ACM Transactions on Computing Education (TOCE)* 10.4, S. 1–15.

Mazur, Eric (2013). „Peer instruction“. In:

McCarthy, Lauren, Casey Reas und Ben Fry (2015). *Getting Started with P5. js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Inc.

Murmann, Lydia (2018). „Calliope mini: Eine Explorationsstudie im pädagogisch-didaktischen Kontext-Abschlussbericht“. In:

Murphy, P. Karen und Patricia A. Alexander (2000). „A motivated exploration of motivation terminology“. In: *Contemporary educational psychology* 25.1, S. 3–53.

Nairobi Tech Girls (2020). *World Summit 2020 Technovation Girls Finalist Team Nairobi Tech Girls*. Technovation Girls. URL: https://www.youtube.com/watch?v=gUqJYQ9j5GI.

Nishida, Tomohiro u. a. (2009). „A CS unplugged design pattern“. In: *ACM SIGCSE Bulletin* 41.1, S. 231–235.

Papert, Seymour (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. USA: Basic Books, Inc. ISBN: 0465046274.

Pattis, Richard E. (1981). *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, Inc.

Pelánek, Radek und Tomáš Effenberger (2020). „Design and analysis of microworlds and puzzles for block-based programming“. In: *Computer Science Education*, S. 1–39.

Reas, Casey und Ben Fry (2007). *Processing: a programming handbook for visual designers and artists*. Mit Press.

Rosado, Jorge Iván Fuentes (2019). „Video Games to learn programming“. In: *Revista Educación en Ingeniería* 14.28, S. 119–123.

Sedgewick, Robert (1992). *Algorithmen in C*. Pearson Deutschland GmbH.

Shaffer, David Williamson und Mitchel Resnick (1999). „"ThickÄuthenticity: New Media and Authentic Learning“. In: *Journal of interactive learning research* 10.2, S. 195–216.

Shapiro, R. Benjamin und Mike Tissenbaum (2019). „New programming paradigms“. In: *The Cambridge Handbook of Computing Education Research*. Cambridge University Press, S. 606–636.

Sorva, Juha (Juni 2013). „Notional Machines and Introductory Programming Education“. In: *ACM Transactions on Computing Education* 13, 8:1–8:31. DOI: 10.1145/2483710.2483713.

Sweller, John (2011). „Cognitive load theory“. In: *Psychology of learning and motivation*. Bd. 55. Elsevier, S. 37–76.

Szabo, Claudia u. a. (2019). „Fifteen Years of Introductory Programming in Schools: A Global Overview of K-12 Initiatives“. In: *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, S. 1–9.

Van Haaster, Kelsey und Dianne Hagan (2004). „Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool.“ In: *Issues in Informing Science & Information Technology* 1.

Watson, Christopher und Frederick WB. Li (2014). „Failure rates in introductory programming revisited“. In: *Proceedings of the 2014 conference on Innovation & technology in computer science education*, S. 39–44.

Weigert, Martin (2017). *Programmieren lernen im Selbststudium: Der magische Meilenstein*. t3n Magazin. URL: https://www.mebis.bayern.de/p/17864 (besucht am 04. 01. 2021).

Weintrop, David und Uri Wilensky (2015). „To block or not to block, that is the question: students' perceptions of blocks-based programming“. In: *Proceedings of the 14th international conference on interaction design and children*, S. 199–208.

Weintrop, David und Uri Wilensky (2017). „Comparing block-based and text-based programming in high school computer science classrooms“. In: *ACM Transactions on Computing Education (TOCE)* 18.1, S. 1–25.

Wigfield, Allan und Jacquelynne S. Eccles (2000). „Expectancy–value theory of achievement motivation“. In: *Contemporary educational psychology* 25.1, S. 68–81.

Wikipedia (2021a). *Calliope mini*. Wikimedia Foundation. URL: https://de.wikipedia.org/wiki/Calliope_mini (besucht am 01. 02. 2021).

Wikipedia (2021b). *Liste von Arduino-Boards*. Wikimedia Foundation. URL: https://de.wikipedia.org/wiki/Liste_von_Arduino-Boards (besucht am 01. 02. 2021).

Wikipedia (2021c). *Motivation*. Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Motivation (besucht am 21. 01. 2021).

Wu, Cheng-Chih, I-Chih Tseng und Shih-Lung Huang (2008). „Visualization of program behaviors: Physical robots versus robot simulators“. In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer, S. 53–62.