Informatische Bildung in Schulen
Jahrgang 2 • Ausgabe 1

ibis

www.informatischebildung.de

GESELLSCHAFT FÜR INFORMATIK
FA IBS · FG DDI

Online-Ausgabe

März 2024

ISSN (Print): 2941-7538 • ISSN (Online): 2941-7546

Impressum

Bezug der Zeitschrift

Die Zeitschrift ibis erscheint kostenfrei online unter www.informatischebildung.de.

Alle Beiträge stehen dort und zusätzlich in der *Digitalen Bibliothek* der *Gesellschaft für Informatik* unter dl.gi.de als Open-Educational-Ressources zur Verfügung.

Zusätzlich werden zu verschiedenen Anlässen Printausgaben produziert, z. B. zur *20. GI-Fachtagung „Informatik und Schule“ 2023*, die vor Ort an der Veranstaltung bezogen werden können.

Beitragseinreichung

Beiträge zur ibis können jederzeit online über die Webseite www.informatischebildung.de eingereicht werden. Dort finden Sie auch den Beitragsaufruf und alle Informationen zur Einreichung.

Für Fragen steht die Redaktion gerne zur Verfügung.

Bibliographische Angaben

ISSN (Print): 2941-7538
ISSN (Online): 2941-7546

DOI-Namensraum: 0.18420/ibis-*JG-NR-ART* (JG: Jahrgang, NR: Heftnummer, ART: Artikelnummer)

Herausgeber

Gesellschaft für Informatik e. V.
Fachausschuss Informatische Bildung an Schulen
Fachgruppe Didaktik der Informatik

Wissenschaftszentrum, Ahrstraße 45, 53175 Bonn

Redaktion

Dr. Andreas Grillenberger *(verantwortlicher Redakteur)*
Prof. Dr. Tilman Michaeli
Prof. Dr. Sandra Schulz
Dr. Peer Stechert

www.informatischebildung.de
info@informatischebildung.de

Beirat

Alisch, Sven · Anthes, Jaqueline · Arnold, Peter · Bergner, Nadine · Brinkmeier, Michael · Burk, Steffen
Bockelberg, Stefanie · Batur, Fatma · Diethelm, Ira · Dorn, Julian · Gallenbacher, Jens · Grillenberger, Mareen
Hempel, Tino · Hennecke, Martin · Hielscher, Michael · Hildebrandt, Claudia · Kastl, Petra · Otto, Torsten
Pöhner, Nicolai · Rau, Thomas · Romeike, Ralf · Rücker, Michael · Schmalfeldt, Thomas · Strecker, Kerstin

Autorinnen und Autorinnen dieser Ausgabe

Andres, Daniela · Bartoschek, Thomas · Bovermann, Klaus · Braune, Gert · Hennecke, Martin
Joachim, Silvia · Komm, Dennis · Michaeli, Tilman · Moenig, Jens · Mühling, Andreas
Özkaya, Ilhan Talha · Pesch, Mario · Rau, Thomas · Witte, Verena

Gestaltung und Satz

Dr. Andreas Grillenberger

Druck

Reine Onlineausgabe

Bildquellen

Titelbild von macrovector_official auf Freepik
Grafiken in den Artikeln: Bildquelle gem. Angabe an den Grafiken

Editorial

Liebe Leserinnen und Leser,

zur ersten Ausgabe der der Zeitschrift „Informatische Bildung in Schulen – ibis“ haben wir im allgemeinen viel positives aber auch kritisches Feedback erhalten, das wir nutzen um insbesondere unsere Prozesse weiterzuentwickeln. Dafür vielen Dank!

Mittlerweile ist die ibis in einigen der relevanten Onlineportale indiziert und somit gut auffindbar, was einerseits für uns ein gutes Zeichen ist und die Akzeptanz der Zeitschrift erhöht, insbesondere aber auch für alle Autorinnen und Autoren sicherlich ein Zeichen der Wertschätzung ist, da so ihre Arbeit möglichst gut sichtbar wird. Alle Beiträge werden derzeit mindestens in der Zeitschriftenbibliothek zlb, bei Google Scholar, im Fachportal Pädagogik und der DBLP indiziert sowie über unsere Webseite, die GI Digital Library, die Deutsche Nationalbibliothek und seit kurzem über das offene Medienportal der Länder „MUNDO“ und somit auch in vielen Medienportalen der deutschen Bundesländer bereitgestellt.

Nun heissen wir Sie herzlich willkommen zur zweiten Ausgabe der ibis. In dieser Ausgabe präsentieren wir eine Auswahl an Artikeln, die die Vielfalt und Tiefe der informatischen Bildung in der Schullandschaft widerspiegeln. Von der praktischen Anwendung von Python und Jupyter-Notebooks im Unterricht über kritische Reflexionen zur Tradition des Informatikunterrichts bis hin zu innovativen Ansätzen in der Vermittlung generativer künstlicher Intelligenz – diese Ausgabe bietet Einblicke in aktuelle Entwicklungen und Methoden, die Lehrkräfte inspirieren und Schülerinnen und Schüler begeistern sollen.

K. Bovermann führt uns durch die Welt der Graphen, indem er zeigt, wie diese mithilfe von Python und Jupyter-Notebooks für Schülerinnen und Schüler erfahrbar gemacht werden können. T. Michaeli regt mit seiner polemischen Grabrede zu Robot Karol zum Nachdenken über Traditionen des Informatikunterrichts an. J. Mönig nimmt uns mit auf eine Reise in die Welt der generativen künstlichen Intelligenz mit Snap!GPT, während das Team um Özkaya, Witte, Bartoschek und Pesch die senseBox vorstellt, die Physical Computing auf innovative Weise vermittelt. D. Andres, S. Joachim und M. Hennecke stellen uns anhand eines Beispieldatensatzes ein Konzept vor, den K-Means-Algorithmus unplugged und plugged zu thematisieren. T. Rau erklärt Schritt für Schritt die Einführung in Reinforcement Learning in Schulprojekten, und Braune sowie Mühling erläutern, wie Programmieren mit einem genetischen Ansatz erlernt werden kann. Im letzten Beitrag dieser Ausgabe gibt uns D. Komm einen beispielorientierten Einblick in die Informatik am Schweizer Gymnasium sowie in dortige aktuelle Entwicklungen.

Wir freuen uns bereits auf die nächste Ausgabe im Sommer und laden Sie herzlich ein, sich mit Ihren Beiträgen zu beteiligen. Besonders interessiert sind wir an weiteren Beiträgen in der Kategorie „Aus der Wissenschaft für die Praxis“, die den Brückenschlag zwischen theoretischer Forschung und praktischer Anwendung im Schulalltag suchen. Alle Informationen zur Beitragseinreichung sind auf der Webseite www.informatischebildung.de verfügbar - gerne stehen wir auch für Fragen per E-Mail unter info@informatischebildung.de zur Verfügung.

Wir danken allen Autorinnen und Autoren für ihre wertvollen Beiträge, allen Mitgliedern unseres Beirates für ihre Arbeit und die wertvollen Hinweise zu den Beiträgen und Ihnen, liebe Leserinnen und Leser, für Ihr Interesse und Engagement.

Viel Freude beim Lesen wünschen

Andreas
Peer
Sandra
Tilman

Inhalt

Graphen im Unterricht

Mit Hilfe von Python und Jupyter-Notebooks

Bovermann, K.
BR Detmold

DOI: 10.18420/ibis-02-01-02

Zusammenfassung

Graphen spielen eine große Rolle in vielen Alltagssituationen sowie in der wissenschaftlichen Informatik. Sie sind Teil der Informatik-Lehrpläne (mindestens der Leistungskurse) der gymnasialen Oberstufe. Beispielsweise findet man das Thema im schulinternen Lehrplan von NRW[1]:

1. Analyse von Graphen in verschiedenen Kontexten

(a) Grundlegende Begriffe (Graph, gerichtet – ungerichtet, Knoten, Kanten, Kantengewicht)

(b) Aufbau und Darstellung von Graphen anhand von Graphenstrukturen in verschiedenen Kontexten (Adjazenzmatrix, Adjazenzliste)

2. Die Datenstruktur Graph im Anwendungskontext unter Nutzung der Klassen Graph, Vertex und Edge.

(a) Erarbeitung der Klassen Graph, Vertex und Edge und beispielhafte Anwendung der Operationen

(b) Bestimmung von Wegen in Graphen im Anwendungskontext (Tiefensuche, Breitensuche)

(c) Bestimmung von kürzesten Wegen in Graphen im Anwendungskontext (Backtracking, Dijkstra).

(d) Bestimmung von minimalen Spannbäumen eines Graphen im Anwendungskontext.

Die im Folgenden aufgeführten Teile dieses Artikels sind entstanden, um Graphen im Unterricht für die Schülerinnen und Schüler erfahrbar zu machen, zum Beispiel unter Nutzung von Python und Jupyter-Notebooks. Dazu wurden entwickelt:

- zwei Python-Bibliotheken, mit denen die Verwaltung von ungerichteten Graphen einfach gelingt.
- eine für die Schülerinnen und Schüler geeignete sehr kurze Einführung in die Begriffswelt von Graphen.
- ein Jupyter-Notebook, das den Schülerinnen und Schülern als Arbeitsmaterial vorgelegt wird.

[1] siehe z. B. https://www.schulentwicklung.nrw.de/lehrplaene/lehrplannavigator-s-ii/gymnasiale-oberstufe/informatik/hinweise-und-beispiele/schulinterner-lehrplan/uv_lk_q1_v.html

Einführung

An vielen Stellen des Alltags begegnen uns sogenannte *Graphen*:

- Das Netz des öffentlichen Nahverkehrs bildet einen Graph; die einzelnen Haltestellen (*Knoten*) sind durch Linien (*Kanten*) miteinander verbunden.
- Mit Hilfe von Graphen kann ein soziales Netzwerk die Beziehungen der Menschen modellieren, indem die Personen (Knoten), die miteinander befreundet sind, mit Kanten verbunden werden.

Schülerinnen und Schüler sollten im Informatikunterricht der Sekundarstufe 2 (besonders sicherlich in einem Leistungskurs) diese wichtige Datenstruktur kennenlernen und grundlegende Algorithmen verstehen, entwickeln und implementieren, wie z. B.:

- Tiefen- und Breitensuche in einem Graphen
- Euler-Weg und Hamilton-Weg
- Kürzeste Wege; der Dijkstra-Algorithmus
- Spannbäume; Algorithmus von Kruskal oder Algorithmus von Prim
- MaxFlowProblem; Ford-Fulkerson-Algorithmus
- TravellingSalesmanProblem; z. B. genetische oder andere approximative Algorithmen

Im Rahmen der Unterrichtseinheit *Graphen* sollten die Grundbegriffe der Graphentheorie vorgestellt werden. *Euler-Wege* und die damit verbundenen Probleme, Sätze und Methoden können dort gute Dienste leisten.

Das Problem des *Kürzesten Weges* ist sicherlich besonders gut geeignet, um das Thema *Graphen* zu motivieren. Im Rahmen dieses Artikels wird die didaktisch-methodische Aufbereitung und Präsentation des *Dijkstra-Algorithmus* unter Nutzung von Python und Jupyter-Notebooks thematisiert.

Alle hier vorgestellten Materialien wurden zum Beispiel in einem Projekt im Rahmen der SMIMS[2] erfolgreich eingesetzt. Die Schülerinnen und Schüler benutzten dabei das beigefügte Jupyter-Notebook sowie die Python-Bibliotheken (siehe dazu später). Neben der Implementation des Dijkstra-Algorithmus entstanden dabei sogar Jupyter-Notebooks[3] zu aktuellen Forschungen über das TSP.

Die Programmiersprache *Python*

Als Programmiersprache bietet sich *Python* an:

[2] siehe unter http://www.smims.nrw.de
[3] erhältlich auf Nachfrage beim Autor

- Die Syntax ist besonders einfach, so dass ein Algorithmus, der bereits umgangssprachlich in einer Art Pseudocode entwickelt wurde, leicht implementiert werden kann.
- Die Semantik der Basiskonzepte ist leicht verständlich.
- Ein Python-Programm erfordert weniger Code (als Java), daher kann ein Programm schneller und verständlicher erzeugt werden.
- Programmbausteine können einfach getestet werden (Vorteil einer Interpretersprache); damit unterstützt Python das sog. *Prototyping*.
- Python stellt eine Unmenge von Bibliotheken bereit, die es gestatten, sich bei der Implementation von Algorithmen auf die wesentlichen Aspekte konzentrieren zu können, ohne viele technische Details kennen zu müssen.
- Die Installation von Python sowie von schüler- bzw. schulgerechten IDEs (wie z. B. Jupyter, s.u.) auf allen Betriebssystemen und allen Geräten gelingt problemlos.

Zwar ist zur Zeit in den Schulen die Programmiersprache Java weit verbreitet. Doch in den Lehrplänen (mindestens dem Lehrplan in NRW) wird keine Programmiersprache vorgeschrieben, und der Informatikunterricht ist sicher kein Programmierkurs, sollte also nicht den Eindruck hinterlassen, dass das Erlernen einer bestimmten Programmiersprache im Mittelpunkt stünde. Ziel des Informatikunterrichts sollte u.a. darin bestehen, wichtige zentrale Ideen und Strukturen aus Informatik zu vermitteln.

Dabei benötigt man sicherlich auch(!) eine Programmiersprache, damit z. B. Algorithmen implementiert werden können und ihre Funktionalität überprüft werden kann. Es dient nicht zuletzt der Motivation, wenn das lauffähige Programm das gewünschte Ergebnis produziert.

Die Rolle von Jupyter-Notebooks

Als Programmierumgebung können Juypter-Notebooks (JNB)[4] genutzt werden. Sie sind webbasiert und ermöglichen einen interaktiven Umgang der Schülerinnen und Schüler mit den - über die JNB direkt zur Verfügung gestellten - Lernmaterialien.

Im schulischen Kontext enthält ein JNB Lehrtexte, Bilder, Links als auch ausführbaren Python-Code. JNBs können mehrere Zwecke erfüllen:

1. Ein JNB ist ein vorbereitetes digitales *dynamisches Arbeitsblatt:*
   es ist ein Tutorial mit Erklärtexten und -bildern, vorbereitetem Code, Aufgaben...
2. Ein JNB ist ein sog. *Computational Essay:*
   es ist ein digitales Notizbuch, mit dem die lesende Person interagieren kann.
3. Ein JNB ist ein sog. *Worked Example:*
   es enthält lauffähige und veränderbare Programme.

[4] https://jupyter.org

Typischerweise enthält ein JNB Lehrtexte, die wie ein traditionelles Schulbuch die fachlichen Grundlagen vermitteln. Anders jedoch als ein Schulbuch kann ein JNB dynamische Bilder, anklickbare Links, Videos, … enthalten. Mit Hilfe von *MarkDown*[5] (eine Art HTML-Light) können diese Inhalte formatiert werden.

Daneben findet sich in einem JNB ausführbarer Python-Code. Schülerinnen und Schüler lernen so die Syntax und Semantik von Python, können den Code interaktiv verändern und erweitern, verbessern und vervollständigen.

Als Entwicklungsumgebung ist der *Anaconda-Navigator*[6] mit den *Jupyter-Notebooks* besonders geeignet:

- kostenlos
- leicht zu installieren
- Browser-basiert; in allen gängigen Browsern lauffähig
- für alle Plattformen verfügbar

Bei der Installation von Anaconda wird auch automatisch (in der Regel die neueste Version von) Python installiert.

Jupyter-Notebooks sind spezielle Dateien, die mit dem Jupyter-System geöffnet werden können. Sie haben die Dateiendung *ipynb*.

Die Python-Bibliothek *networkx*

Um in Python Graphen zu verwalten, kann man z. B. die Bibliothek *networkx* nutzen. Dabei handelt es sich um eine im Internet zugängliche Bibliothek[7]. Dort findet man auch eine sehr umfangreiche Dokumentation aller verfügbaren Funktionen.

Mit Hilfe dieser Bibliothek und der dort bereitgestellten Klasse *Graph* ist es u.a. möglich, ungerichtete Graphen zu verwalten.

Details zu den Bibliotheken *attrDirGraph* und *simpleGraph*

Die oben erwähnte Bibliothek *networkx* ist zwar sehr leistungsstark, jedoch aus Sicht des Informatikunterrichts an Schulen komplex und für Schülerinnen und Schüler gerade bei der Einführung in das Thema *Graphen* aus meiner Sicht nur bedingt geeignet. Deswegen wurden zwei Bibliotheken entwickelt, die alle Klassen und Funktionen aus der erwähnten Bibliothek erben, jedoch für Lernende einen einfachen Umgang mit den notwendigen Funktionen erlauben.

Beide Bibliotheken können von GitHub heruntergeladen werden:
https://github.com/klausNetSchulbuch/GraphWithPython.git

[5] siehe z. B. https://markdown.de/
[6] siehe z. B. https://www.anaconda.com/download
[7] *https://networkx.org/*

Die Bibliothek *attrDirGraph* stellt eine Python-Klasse bereit, mit der ungerichtete Graphen verwaltet werden können. Jeder Knoten und jede Kante eines solchen Graphen kann beliebig viele, vom Benutzer frei zu benennende Attribute erhalten. Insbesondere kann jede Kante ein Gewicht haben, so dass dann der Graph (im Sinne der Graphentheorie) ein gewichteter Graph[8] ist.

Ein kurzes Beispiel kann den Umgang mit dieser Bibliothek verdeutlichen:

```
In [1]: import attrDirGraph as adg

        g = adg.attrDirGraph()

        g.fuegeKnotenHinzu("Berlin")
        g.fuegeKnotenHinzu("München")
        g.fuegeKanteHinzu("Berlin", "Potsdam")
        g.setKnotenAttribut("Berlin", "Farbe", "blau")

        print(g.alleKnoten())
        print(g.alleKanten())
        print(g.alleNachbarknoten("Berlin"))
        print(g.getKnotenAttribut("Berlin", "Farbe"))
```

```
['Berlin', 'München', 'Potsdam']
[('Berlin', 'Potsdam')]
['Potsdam']
blau
```

Abbildung 1: Umgang mit der Bibliothek

Besonders in Zusammenhang mit dem zu implementierenden Dijkstra-Algorithmus benötigt man spezielle Markierungen der Kanten und Knoten. Dazu stellt die Bibliothek *simpleGraph* weitere Vereinfachungen zur Verfügung:

```
In [2]: import simpleGraph as sg

        g = sg.simpleGraph()

        g.fuegeKanteHinzu("B", "K")
        g.markiereKnoten("B", "gross")
        g.besucheKnoten("K")

        if g.knotenIstBesucht("K"):
            print("Ist besucht")

        print(g.getKnotenMarke("B"))
```

```
Ist besucht
gross
```

Abbildung 2: Vereinfachungen durch simpleGraph

Eine Dokumentation aller Funktionen dieser Klassen ist in den Bibliotheksdateien in Form von Python-Kommentaren zu finden. Der Umgang mit den beiden Klassen kann optimal bei der Benutzung des Jupyter-Notebooks erfahren werden.

[8] In diesem Sinne ist ein gewichteter Graph nur ein Sonderfall eines attributierbaren Graphen.

Anmerkung zu den *assert-Statements* in den Bibliotheken

Alle Funktionen in den beiden Bibliotheksklassen sind mit sog. *assert-Statements* abgesichert. Wenn man beispielsweise ermitteln möchte, ob der Knoten "H" bereits besucht wurde, dieser Knoten jedoch in dem Graphen nicht enthalten ist, wird eine aussagekräftige Fehlermeldung erzeugt, die das Programm beendet oder mit einer entsprechende Aktion behandelt werden kann. Diese Fehlerprüfungen sind jedoch sehr zeitintensiv. Wenn man sicher ist[9], dass solche Situationen niemals auftreten, kann man auf diese Prüfungen verzichten.

Dazu gibt es beide Bibliotheken als unsafe-Versionen, unter dem Namen *attrDirGraphUnsafe* bzw. *simpleGraphUnsafe*.

Die beigefügten Dateien

Die in diesem Artikel angesprochenen Bibliotheken und alle für die Arbeit mit dem JNB notwendigen Dateien finden Sie unter der Adresse
https://github.com/klausNetSchulbuch/GraphWithPython

Die Nutzung der Bibliotheken wird im Jupyter-Notebook direkt demonstriert.

4. In dem Verzeichnis *GraphBib* findet man die Bibliotheken.
5. Im dem Verzeichnis *Daten* sind einige csv-Dateien, in denen Graphen beschrieben sind und in Jupyter-Notebooks eingelesen werden können.
6. Das Jupyter-Notebook *Dijkstra-Lernen.ipynb* ist zentral für den Einsatz im Unterricht. Mit diesem Notebook können Schülerinnen und Schüler das Dijkstra-Verfahren kennenlernen. Dieses Notebook verdeutlicht den Ablauf der Unterrichtsreihe. Es verdeutlicht die Idee, die Jupyter-Notebooks charakterisieren (*dynamisches Arbeitsblatt, Computational Essay, Worked Example).*
7. Die Datei *Dijkstra-Lernen.pdf* ist ein PDF-Export des gleichnamigen Jupyter-Notebooks.
8. Die Datei *Einfuehrung.pdf* bietet einen sehr kurzen Abriss der Graphentheorie.

Lizenz

Kontakt

Klaus Bovermann
BR Detmold
E-Mail: klaus.bovermann@uni-bielefeld.de

[9] Das Jupyter-Notebook *Dijkstra-Loesen.ipynb* findet automatisch kürzeste Wege. Nutzt man die sicheren Versionen der Bibliotheken, benötigt die Suche je nach Rechner bis zu 20 Minuten!

Robot Karol: Eine Grabrede

Michaeli, T.
TU München

DOI: 10.18420/ibis-02-01-03

Dieser Text nutzt bewusst Stilmittel der Polemik, um auf Traditionen des Informatikunterrichts aufmerksam zu machen, die aus Sicht des Autors hinterfragt werden sollten.

Liebe Trauergemeinde,

wir haben uns heute hier versammelt, um dich, lieber Robot Karol, zur letzten Ruhe zu geleiten. Zunächst wollen wir dazu gemeinsam auf dein Leben zurückblicken.

Aufgewachsen bist du in einer sehr großen Familie: Dein ältester Bruder, *Karel the Robot*, hat bereits 1981 das Licht der Welt erblickt. Er konnte mit einfachen textbasierten Befehlen in einer 2D-Rasterwelt bewegt werden und *Beeper* aufheben und ablegen, um Anfängerinnen und Anfänger die Grundlagen der Programmierung zu vermitteln (Pattis, 1981). Daneben hast du zahlreiche weitere Geschwister wie etwa Niki den Roboter, Karol++, oder Java Karol sowie Cousins und Cousinen wie Kara, den Marienkäfer, den JavaHamster oder weitere Miniwelten. Als deutsche Variante hast du schließlich 2001 die Beschränkung auf zwei Dimensionen hinter dir gelassen und den dreidimensionalen Raum erobert, um dort neben Marken nun auch eifrig Ziegel zu legen, zu stapeln und aufzuheben.

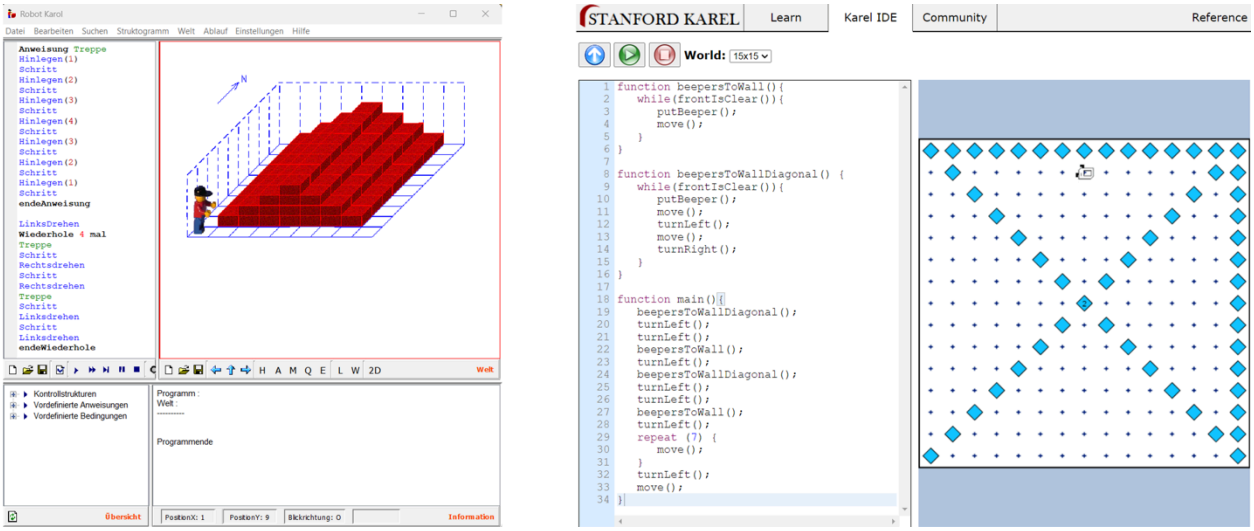


Abbildung 1: Karol und Karel (Screenshots)

Stammvater deiner Familie ist Logo, die erste Programmiersprache für Kinder, die Cynthia Solomon, Wallace Feurzeig, Daniel Bobrow und Seymour Papert 1967 entwickelt haben[1]. Wichtig war ihnen, Lernenden *Gegenstände-mit-denen-man-denkt* (engl: “Objects to think with”) an die Hand zu geben (Papert, 1982), wie etwa durch die sogenannte *Turtle-Grafik* als wohl bekanntesten Teil dieser (viel mächtigeren und umfangreicheren) Programmiersprache: Mit Befehlen wie *forward(länge), back(länge), right(winkel), left(winkel)* können eine (oder mehrere) kleine Schildkröten über den Bildschirm bewegt werden, die auf Wunsch farbige Spuren hinter sich herziehen. Auf diese Art und Weise veranschaulicht die Bewegung der *Turtle* den Programmablauf und es lassen sich etwa einfache geometrische Figuren wie Dreiecke oder Quadrate umsetzen. Aus diesen einfachen Bausteinen können größere Kunstwerke kombiniert werden, genauso lassen sich aber auch komplexe Figuren wie Fraktale oder Hilbertkurven realisieren. Diese Idee von Logo und der *Gegenstände-mit-denen-man-denkt* (virtuell wie physisch) begründet einen großen Stammbaum, von Miniwelten wie Karel über Scratch bis hin zu LEGO Mindstorms (benannt nach dem gleichnamigen Buch von Seymour Papert).

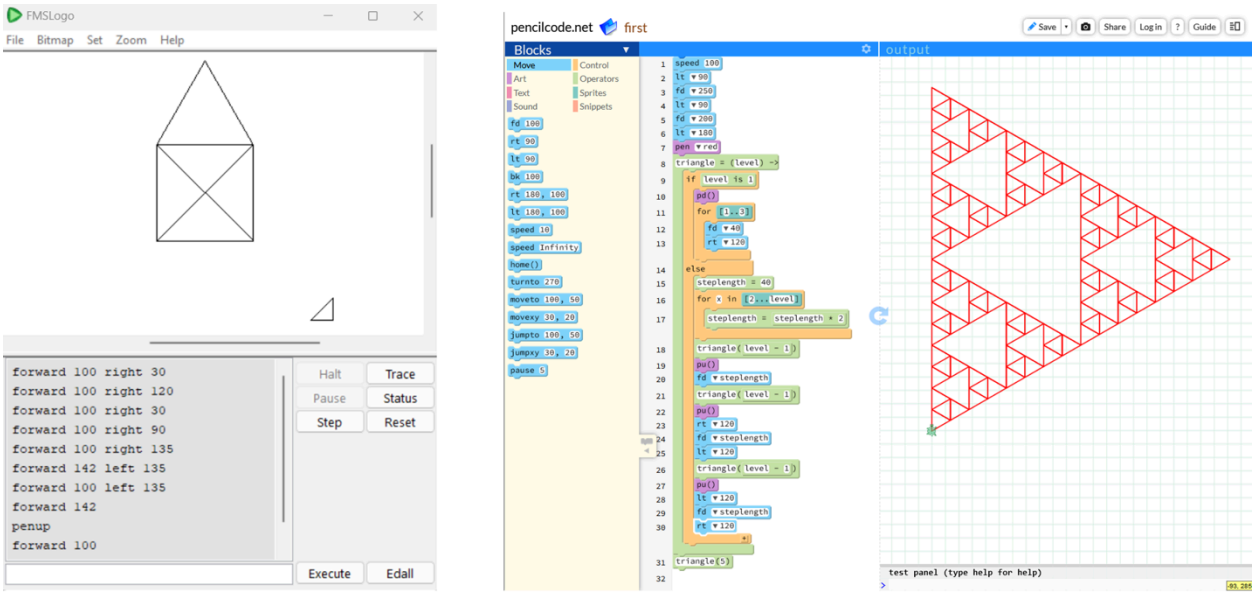


Abbildung 2: Turtlegrafik in Logo und PencilCode (Screenshots)

Nun aber zurück zu deinem Leben: In deutschen Schulen (insbes. an bayerischen Gymnasien) hast du, lieber Robot Karol, lange Jahre Schülerinnen und Schüler mit deinem reduzierten Sprachumfang und – ganz Logo-Nachkomme – vielfältiger Unterstützung beim Debugging an grundlegende algorithmische Konzepte herangeführt. Zu einer Zeit, in der informatische Bildung in Schulen noch in den Kinderschuhen steckte, vor “Bildungsstandards Informatik”, vor der langsamen aber konstanten Verbreitung des Pflichtfachs, warst du eines der wenigen Werkzeuge, um Schülerinnen und Schüler zielgruppengerecht Programmieren beizubringen

[1] Zum Nachlesen der Geschichte von Logo siehe etwa Solomon et al., 2020.

und hast damit einen wichtigen Beitrag für informatische Bildung geleistet. Manche kamen sogar auf die Idee, dich für die Objektorientierung zu missbrauchen, obwohl du in deiner Welt – ganz alleine – das einzige (potentielle) Objekt bist und kein Zweck der Modellierung (als zentraler Bestandteil informatischer Modellbildung) erkennbar ist. Bis heute bist du damit häufig die Primärerfahrung von Schülerinnen und Schülern zu "Informatikunterricht"[2], insbesondere in Bayern, und eine der wenigen Erinnerungen, die auch nach dem Schulabschluss noch Bestand haben: "Achja, Informatik, das war das mit diesem nutzlosen Roboter".

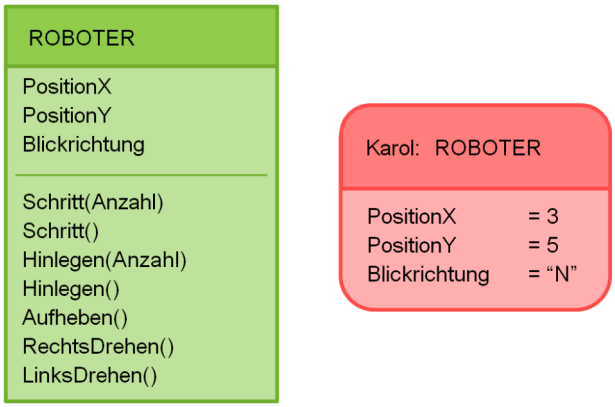


Abbildung 3: Exemplarische Darstellung eines OO-Karol ohne Zweck der Modellierung wie etwa in Schulbüchern üblich

Nun ist es aber an der Zeit – nach getaner Arbeit – die wohlverdiente ewige Ruhe zu genießen. Schließlich ermöglichen seit bereits mindestens 15 Jahren viele deiner Nichten und Neffen eine neuere, aufregendere, und – um direkt zu sein – einfach bessere Einführung in die Programmierung.

Dabei geht es nicht darum, dass du ein textbasiertes Programmierwerkzeug bist – auch wenn für den Programmiereinstieg natürlich viele gute Gründe (insbes. das Wegfallen der Hürde *Produktwissen Syntax*) für visuelle und blockbasierte Ansätze sprechen. Es geht auch nicht darum, ob du im Browser verwendet werden kannst, anstatt via vorsintflutlich anmutender ausführbarer JAR-Datei und Optik der Benutzeroberfläche. Seit ein paar Jahren läufst du ja auch online, sogar in einer Variante mit Blöcken.

Vielmehr geht es um das Bild der Informatik, von dem du geprägt bist und das du in den Köpfen unserer Schülerinnen und Schüler erzeugst. Du bist ein *Teaching System* (Kelleher & Pausch, 2005), das – anstatt ein Fenster in die bunte Welt der Informatik zu eröffnen – nur darauf abzielt, Lernenden "korrektes Programmieren" beizubringen. Dazu müssen viele kleine, konstruierte und künstliche Problemchen und Puzzles gelöst werden, wie z.B., dich in einem Zimmer mit nur einem Ausgang die Türe finden oder bestimmte vorgegebene Bauwerke aus Ziegeln

[2] Galt auch für den Autor an einem fränkischen Gymnasium; zusammen mit irgendwelchen Kästen, die mal "abgerundete", mal "spitze" Ecken hatten, ohne dass es Ursache oder Zweck für diesen Unterschied zu geben schien, und die bis heute ungeklärte Frage, was zur Hölle "BEZEICHNER" sein sollen.

errichten zu lassen (also eben gerade keine Probleme zu lösen). Und von dem Bild, das wir ausgerechnet mit einem Roboter als Sinnbild von “Technik” und stereotypischer “Männerdomäne” erzeugen, wollen wir lieber gar nicht erst anfangen. Gerade für den Erstkontakt mit Informatik ist es aber von größter Bedeutung, erfahrbar zu machen, was Informatik ist: Und das ist eben **nicht** das technische Lösen künstlicher, bedeutungsloser, kleiner und langweiliger Aufgaben und vermeintlicher Probleme, sondern **aktives und kreatives Gestalten!**

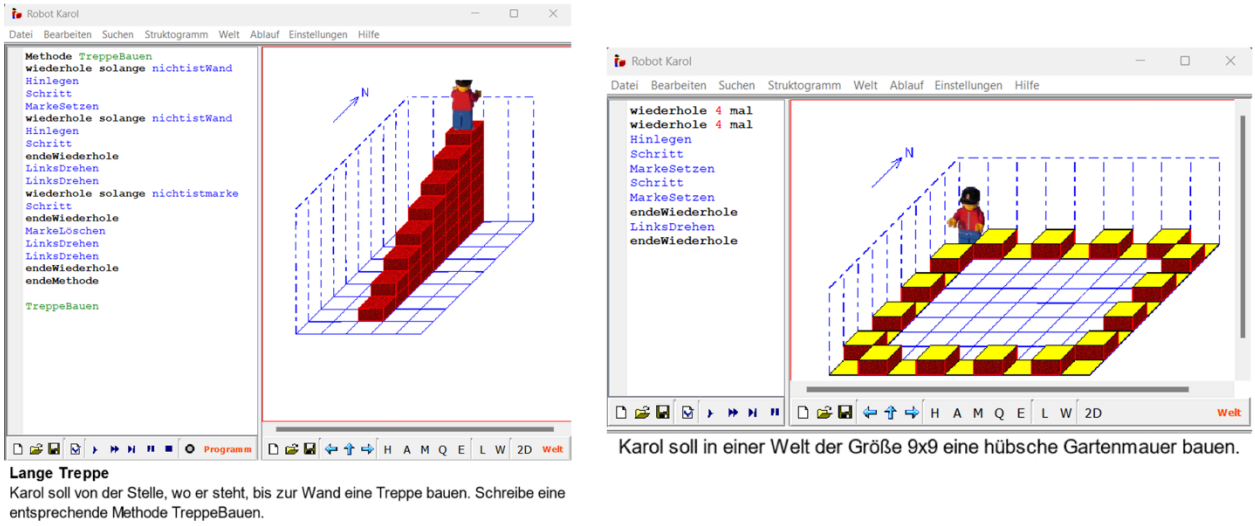


Abbildung 4: Typische Aufgaben (keine Probleme) in Robot Karol (Screenshots)

In diesem Sinne ist es genauso wenig das Ziel informatischer Allgemeinbildung, zukünftige Entwicklerinnen und Entwickler für das “korrekte Programmieren” vorzubereiten: Stattdessen wollen wir jeden und jede dazu befähigen, die digitale Welt zu verstehen und mitzugestalten (GI, 2016). Das bedeutet, Schülerinnen und Schüler zu *empowern*, mit und durch Informatik bzw. der Programmierung als unserem zentralen Gestaltungsmittel, **persönlich bedeutungsvolle** Ideen zu verwirklichen, eigene Spiele oder Animationen zu gestalten, interaktive Geschichten zu erzählen oder Simulationen zu erstellen (Romeike, 2007; Resnick et al., 2009). Dabei und dafür werden die notwendigen informatischen bzw. algorithmischen Konzepte erlernt und (echte) Probleme gelöst. Die Bedeutung des Erschaffens und Gestaltens persönlich bedeutungsvoller Artefakte für Lernen wird auch durch die Lerntheorie des *Konstruktionismus* betont, die in der informatischen Bildung lange etabliert ist und von Seymour Papert wie folgt zusammengefasst wird: “Die Idee, dass Lernen am effektivsten ist, wenn es Teil einer Aktivität ist, die der Lernende als Schaffen eines persönlich bedeutungsvollen Produkts wahrnimmt”[3] (Papert, 1987).

Dafür sind Werkzeuge nötig, die das Gestalten in entsprechenden **Kontexten** ermöglichen. Und gerade für ein Pflichtfach ist die Orientierung an geeigneten Kontexten zentral, um Interesse

[3] Eigene Übersetzung aus dem Englischen.

und Motivation von Schülerinnen und Schülern zu wecken (Koubek et al., 2009). Diese Bedeutung und Möglichkeiten der Kontextualisierung für die informatische Bildung werden etwa durch den Ansatz *Media Computation* (Guzdial, 2013; Mönig & Hügle, 2021) eindrucksvoll belegt, gerade auch für die Ansprache "informatikferner" Zielgruppen, die sich vorher nicht unbedingt vorstellen konnten, programmieren zu wollen. Es geht aber nicht nur um Motivation und Interesse: Die Wahl geeigneter Kontexte ist elementare Voraussetzung informatischer Allgemeinbildung, da sie (1) das Wesen der Informatik als Grundlage der Digitalisierung in allen Lebensbereichen verdeutlicht, (2) die Dekonstruktion und damit Entmystifizierung von Informatiksystemen erlaubt (Strecker, 2014; Magenheim, 2000) und (3) die Analyse der Wirkung von Informatik hinsichtlich verschiedener Dimensionen auf Individuen und Gesellschaft (gesellschaftliche-kulturelle Perspektive des Dagstuhl-Dreiecks) überhaupt erst ermöglicht.

Während schon die Turtle-Grafik (informatische) Kunst als Kontext und Zugang ermöglicht, bist du, lieber Karol, nunmal ein Roboter, der Ziegel legen kann und mit dem sich keinerlei Projekte mit Bezug zur Lebenswelt, Interessen der Schülerinnen und Schüler, zur Entmystifizierung von Informatiksystemen oder gar zu gesellschaftlich-kulturellen Aspekten von Informatik umsetzen lassen. Man vergleiche das nur mit den Möglichkeiten, die uns Werkzeuge wie Scratch, Snap! und Co. an sich, oder auch in Kombination mit Ansätzen des Physical Computing und der Verbindung in die echte Welt bieten!

Das sind keine neuen Feststellungen: Schon Seymour Papert, der Logo inklusive Turtle-Grafik als konstruktionistischen Ansatz anno dazumal entwickelt hat, formulierte Anforderungen an Programmiersprachen für Kinder: Diese sollen sowohl *Low Floors* (geringe Einstiegshürden), *Wide Walls* (vielfältige unterschiedliche Arten von umsetzbaren Projekten) als auch *High Ceilings* (die Möglichkeit, auch komplexe und anspruchsvolle Projekte zu realisieren) ermöglichen (Resnick et al., 2009). Zwar können wir die Low Floors für dich festhalten, wie bereits oben ausführlich beschrieben, hast du aber nicht mal im Ansatz Wide Walls. Genauso wenig können wir mit dir im Sinne von *High Ceilings* komplexe Projekte umsetzen – im Gegensatz zu sowohl dem antiken Logo (jenseits der Turtle) als auch bereits genannten dir überlegenen Werkzeugen, mit denen wir bis zum Abitur oder gar an Universitäten erfolgreich arbeiten (Strecker, 2015; Garcia et al., 2015).

Auch die Wissenschaft weiß das schon lange und belegt durch empirische Forschungsergebnisse, dass du mit deinen Nachfahren nicht mithalten kannst. So zeigen Untersuchungen etwa, dass Schülerinnen und Schüler bei einem "kreativen" Ansatz (mit Scratch) vs. einem "problemlösenden" (mit Karol), signifikant mehr Spaß, Interesse, Kreativität und Experimentierfreudigkeit berichten (Romeike, 2007). Auch haben ausgerechnet bayerische Studienergebnisse für die Verwendung von Scratch eine signifikant höhere intrinsische Motivation und Leistung im Vergleich zu Robot Karol festgestellt (Ruf et al., 2014). International werden diese Ergebnisse für den Vergleich textbasierter und blockbasierter Programmierung im Anfangsunterricht bestätigt (Weintrop & Wilensky, 2017). Und auch für den Übergang zu "korrekter" Programmierung (als Ziel von Karol) in professionell genutzten Programmiersprachen wie Java, JavaScript oder

Python belegt empirische Forschung den erfolgreichen Transfer von kontextualisiertem Unterricht mit Scratch (etwa Grover, 2021), der auch von Studierenden bestätigt wird (Malan & Leitner, 2007). Während also in so manchen Lehrerzimmern oder Fortbildungsveranstaltungen Diskussionen wie Karol vs. Scratch (oder vielleicht ja auch – etwas weiter gefasst – "Text vs. Blöcke") im Anfangsunterricht immer noch ein heißes Thema sind, ist dieser Diskurs in der Wissenschaft Didaktik der Informatik schon seit vielen Jahren weitgehend abgeschlossen – mit einem eindeutigen Ergebnis.

Genauso wenig wie wir heute noch mit Logo arbeiten, solltest du als Werkzeug der achtziger Jahre also noch für die Einführung in die Programmierung im Informatikunterricht herhalten müssen. Mit unserem gemeinsamen Rückblick auf dein erfülltes Leben verabschieden sich nun hoffentlich auch deine letzten verbliebenen treuen Fans und langjährigen Wegbegleiter von dir; vielleicht ja sogar die Autorinnen und Autoren bayerischer Informatik-Schulbücher…

So wollen wir diesen Anlass also nicht mit Trauer über dein Ableben beschließen, sondern vielmehr mit Freude über die sich vollzogene Weiterentwicklung informatischer Bildung und die tollen Möglichkeiten, die Werkzeuge wie Scratch, Snap! usw. (samt zahlloser begleitenden Unterrichtsmaterialien) für Informatikunterricht und Schülerinnen und Schüler heutzutage bieten.

Insofern: Danke für deine Leistung, nun ruhe in Frieden – es ist WIRKLICH Zeit!

Ein besonderer Dank gilt Ralf Romeike für die Grundlage und Ausschärfung der Argumentation sowie den weiteren kritischen Testlesern Peter Brichzin, Sven Baumer und Franz Jetzinger für die vielen wertvollen Anregungen zur Darstellung.

Quellen

Garcia, D., Harvey, B., & Barnes, T. (2015). The beauty and joy of computing. *ACM Inroads*, *6*(4), 71-79.

Gesellschaft für Informatik (2016). Dagstuhl-Erklärung: Bildung in der digitalen vernetzten Welt, gi.de/fileadmin/GI/Hauptseite/Themen/Dagstuhl-Erkla__rung_2016-03-23.pdf,

Grover, S. (2021). Teaching and assessing for transfer from block-to-text programming in middle school computer science. *Transfer of learning: Progressive perspectives for mathematics education and related fields*, 251-276.

Guzdial, M. (2013). Exploring hypotheses about media computation. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 19-26).

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR)*, *37*(2), 83-137.

Koubek, J., Schulte, C., Schulze, P., & Witten, H. (2009). Informatik im Kontext (IniK)–Ein integratives Unterrichtskonzept für den Informatikunterricht. *Zukunft braucht Herkunft–25 Jahre» INFOS–Informatik und Schule «*.

Magenheim, J. (2000). Informatiksystem und Dekonstruktion als didaktische Kategorien-Theoretische Aspekte und unterrichtspraktische Implikationen einer systemorientierten Didaktik der Informatik. *GI-Tagung Informatik-Ausbildung und Beruf*.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM Sigcse Bulletin*, *39*(1), 223-227.

Mönig, J. & Hügle, J. (2021): Von Media Computation zu Data Science in SNAP!. LOG IN: Vol. 41, No. 1. Berlin: LOG IN Verlag. PISSN: 0720-8642. pp. 74-81.

Papert, S. (1982). Mindstorms: Kinder, Computer und Neues Lernen. Springer Basel AG.

Papert, S. (1987). Constructionism: A New Opportunity for Elementary Science Education. NSF Award Abstract #8751190. https://www.nsf.gov/awardsearch/showAward?AWD_ID=8751190

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

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.

Romeike, R. (2007). Applying creativity in CS high school education: criteria, teaching example and evaluation. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88* (pp. 87-96).

Romeike, R. (2007). Animationen und Spiele gestalten-Ein kreativer Einstieg in die Programmierung. LOG IN, 27(146/147), 36-44.

Ruf, A., Mühling, A., & Hubwieser, P. (2014). Scratch vs. Karel: impact on learning outcomes and motivation. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 50-59).

Strecker, K. (2014): Kontextbezogene Aufgaben. LOG IN: Vol. 34, No. 1. DOI: 10.1007/s40569-014-0012-6. LOG IN Verlag GmbH. pp. 88-95

Strecker, K. (2015): Grafische Programmiersprachen im Abitur. Informatik allgemeinbildend begreifen. In *16. GI-Fachtagung Informatik und Schule - Informatik allgemeinbildend begreifen (INFOS15).* Bonn: Gesellschaft für Informatik e.V. (pp. 293-300).

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., ... & Silverman, B. (2020). History of logo. *Proceedings of the ACM on Programming Languages*, *4*(HOPL), 1-66. https://dl.acm.org/doi/10.1145/3386329

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1-25.

Lizenz

Kontakt

Prof. Dr. Tilman Michaeli

Didaktik der Informatik, TU München

tilman.michaeli@tum.de

Snap!GPT – Bausteine für generative künstliche Intelligenz

Mönig, J.
SAP, Walldorf

DOI: 10.18420/ibis-02-01-04

Zusammenfassung

„Und dann?“

Um diese Frage geht es bei generativer künstlicher Intelligenz. Darum, was als nächstes kommen könnte: Das nächste Wort in einer Geschichte, der nächste Ton in einem Lied, der nächste Strich in einer Zeichnung. In diesem Beitrag zeige ich, wie man ein „Next Token Prediction“ System in Stil von ChatGPT in Snap! selber programmieren kann. Gibt man ihm ein paar Geschichten, dann lernt es, eigene Text zu schreiben. Gibt man ihm ein paar Kinderlieder, dann improvisiert es eigene Melodien, und zeichnet man ihm etwas vor, dann versucht es, selbst etwas zu kritzeln.

Man selbst lernt bei dieser Aktivität, Wörter, Musiknoten und Striche zu kodieren, in einem Datenmodell zu strukturieren, und kontextbezogen abzufragen. Das lässt sich mit zwei eigenen Funktionsblöcken, ein paar Schleifen, Variablen und Bedingungen bewerkstelligen, und eignet sich z.B. für den Informatikunterricht ab der 7. Klasse.

Einleitung

Im letzten IBIS-Heft hat Michael Hielscher mit SoekiaGPT ein auf n-Grammen basierendes didaktisches Sprachmodell vorgestellt. Aus 25 Grimm’schen Märchen erzeugt es ähnlich lautende, aber unsinnige Texte. Seine Idee, Funktionsweisen und Eigenschaften generativer künstlicher Intelligenz mit simplen Markov-Ketten und dem Halluzinieren von Märchen (sic!) zu erklären, begeistert mich. Ich habe mich gefragt, ob es möglich ist, das Prinzip noch weiter zu vereinfachen, damit Lernende so etwas wie ChatGPT selber programmieren können, gar in einer grafischen Programmiersprache wie Snap!, an der ich mitentwickle.

Entstanden ist ein kleines Programm mit nur zwei eigenen Blöcken und wenigen Skripten. Man kann damit lustige Texte erzeugen, und nebenbei einiges über das Programmieren mit größeren Datenmengen lernen. Und wenn man die Bausteine ein bisschen anders zusammenfügt, passiert sogar etwas Erstaunliches: Dann lernt das Programm, auch andere Reihen, die nicht aus Wörtern bestehen, fortzusetzen, und damit z.B. Musik oder Bilder zu machen.

Das Snap!GPT Projekt hat drei Teile: Es werden Beispieldaten (hier: Märchen) geladen und zerlegt, daraus entsteht ein Sprachmodell, eine Art Datenbank. Anschließend wird eine Funktion entwickelt, die aus dem Datenmodell einen Eintrag (hier: ein Wort) heraussucht, der eine bestehende Reihe (hier: vorhergehende Worte) plausibel fortsetzt. Zum Schluss kann das Ganze noch interaktiv für Benutzer gestaltet werden.

Snap! ist eine grafische, blockbasierte Programmiersprache, die gemeinsam von der Universität Berkeley und SAP für den Informatikunterricht an Schulen und Universitäten entwickelt wird. In den USA wird damit das vom College Board zertifizierte „Beauty and Joy of Computing“ Curriculum unterrichtet. Auch in Deutschland gibt es eine wachsende Community. Snap! verwendet dieselbe Baustein-Grammatik wie Scratch, das erleichtert den Einstieg für Lernende, die schon mal mit Scratch programmiert haben. Gleichzeitig sind in Snap! viele Funktionen schon eingebaut, die beim Erstellen eines statistischen Sprachmodells hilfreich sind. Dadurch bleibt das hier vorgestellte Projekt von Umfang her überschaubar. Snap! läuft im Web Browser, ist kostenlos und quelloffen.

Daten analysieren

Dateien verschiedener gebräuchlicher Formate (txt, csv, json) kann man mit der Maus in das Snap!-Fenster ziehen, um sie zu laden. Oder man wählt sie im Dateimenü mit „Importieren“ aus. Snap! legt eine globale Variable mit dem Dateinamen an und weist ihr deren Inhalt als Wert zu. Strukturierte Daten (z.B. csv oder json) werden dabei automatisch in Listen- bzw. Tabellenform gebracht, reine Texte bleiben unverändert am Stück. Weil mir Hielschers Beispiel mit den Märchen so gut gefällt, habe ich mir ebenfalls ein paar Webseiten mit Märchen der Brüder Grimm herausgesucht, und davon 30 Geschichten als txt-Datei heruntergeladen, bzw. aus dem Browser in lokale Textdateien kopiert. Die Einzeltexte habe ich zu einer einzigen großen Datei zusammengefügt, und diese in Snap! geladen.

Mit dem „trenne” Block kann ein Text in verschiedene Einzelteile zerlegt werden. Ich habe die gesammelten Märchen nach „Wort” getrennt, um eine große Liste mit der Reihenfolge aller Wörter zu erhalten (siehe Abb. 1).

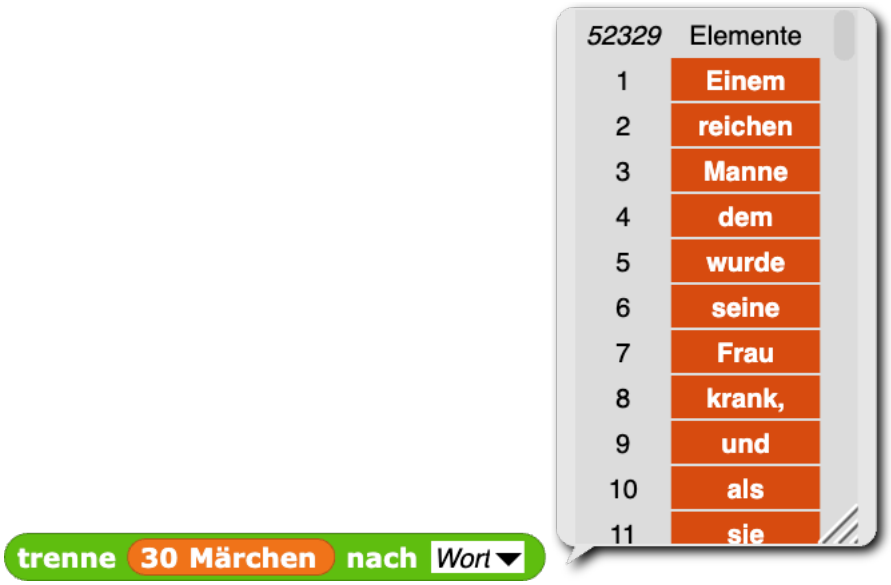


Abbildung 1: Mit dem „trenne“ Block in Snap! aus einem Text eine Wörterliste gewinnen

Der Clou des Sprachmodells ist die Zerlegung dieser langen Wörterliste in mehrere Versionen mit vielen kleinen Einträgen bestehend aus Wörterpaaren, Tripeln, Vierer- und Fünferketten, sogenannten n-Grammen. Dafür bietet sich eine Funktion mit zwei Parametern an, „n” für die gewünschte Länge der zu bildenden Ketten, und „Korpus” für die Ausgangsliste aller aufeinanderfolgenden Wörter. In Snap! habe ich dafür einen kleinen Funktionsblock definiert (siehe Abb. 2).

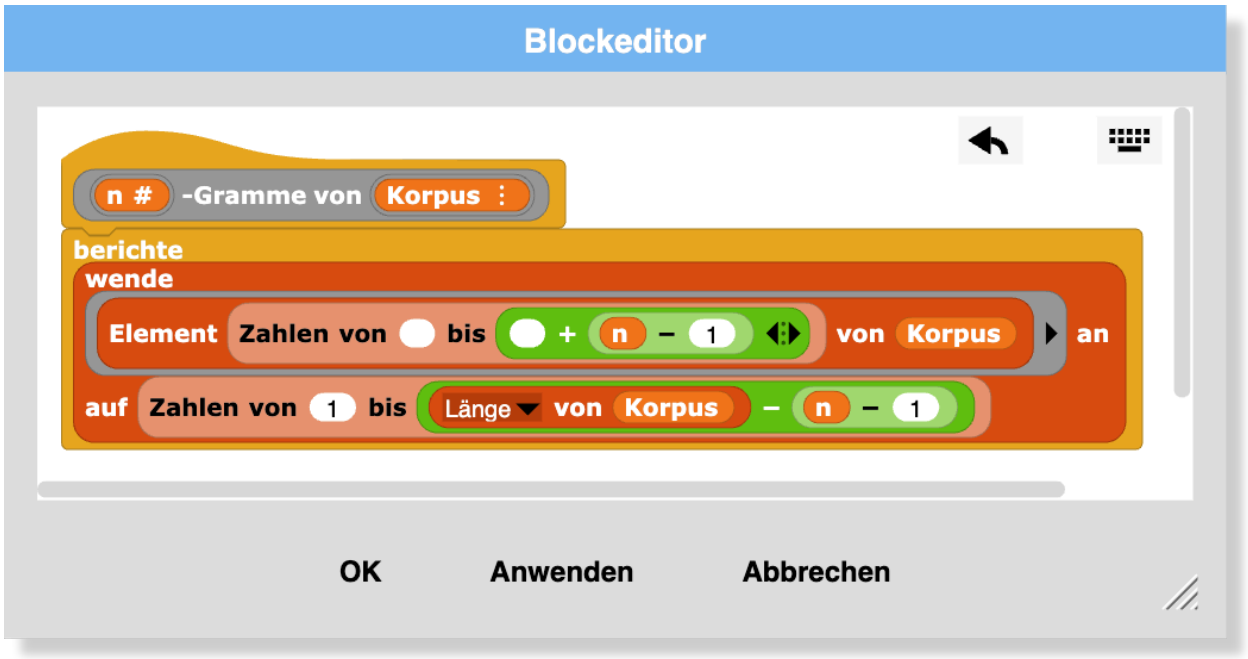


Abbildung 2: Beispiel für eine benutzerdefinierte „n-Gramme“ Funktion in Snap!

Man kann n-Gramme in Snap! auf verschiedene Weisen implementieren, z.B. imperativ mit Befehlen und sogar hyperdimensional mit Vektoroperationen. Ich habe mich für eine funktionale Variante entschieden, weil sie am expressivsten ist, d.h. am wenigsten Blöcke braucht, und die Aufgabe recht schnell erledigt. Wer etwas längere Laufzeiten in Kauf nimmt, kann natürlich genauso gut mit Schleifen und Variablen arbeiten.

Als nächstes sollen n-Gramme für Wörterketten aus einem bis fünf Gliedern gebildet, und in einem Sprachmodell gespeichert werden. Dazu habe ich eine weitere globale Variable namens „Modell" angelegt. Ihr soll eine Liste zugewiesen werden, deren Elemente jeweils das Resultat der entsprechenden „n-Gramme" Funktion sind. An erster Stelle steht eine Liste aller Einzelwörter, an zweiter Stelle alle Wörter-Paare, das dritte Element ist eine Liste aller Dreier-Ketten, und so weiter. Das kann man z.B. mit einer Schleife und einer Laufvariable erreichen. Ich habe mich wiederum für ein kurzes funktionales Skript entschieden (siehe Abb. 3).

Abbildung 3: Das fertige Snap! Skript für das „Trainieren" eines statistischen Sprachmodells

Es reicht, wenn dieses Skript einmal mit der Maus angeklickt und ausgeführt wird. Dann dauert es eine (hoffentlich nur kurze) Weile, und das Sprachmodell ist einsatzbereit. Um das zu überprüfen, kann das Modell mit dem „Element" Block für verschiedene n-Werte von 1 bis 5 abgefragt werden. Es sollte jeweils eine Liste bzw. Tabelle angezeigt werden, deren Spaltenanzahl mit ihrer Position im Modell übereinstimmt (siehe Abb. 4).

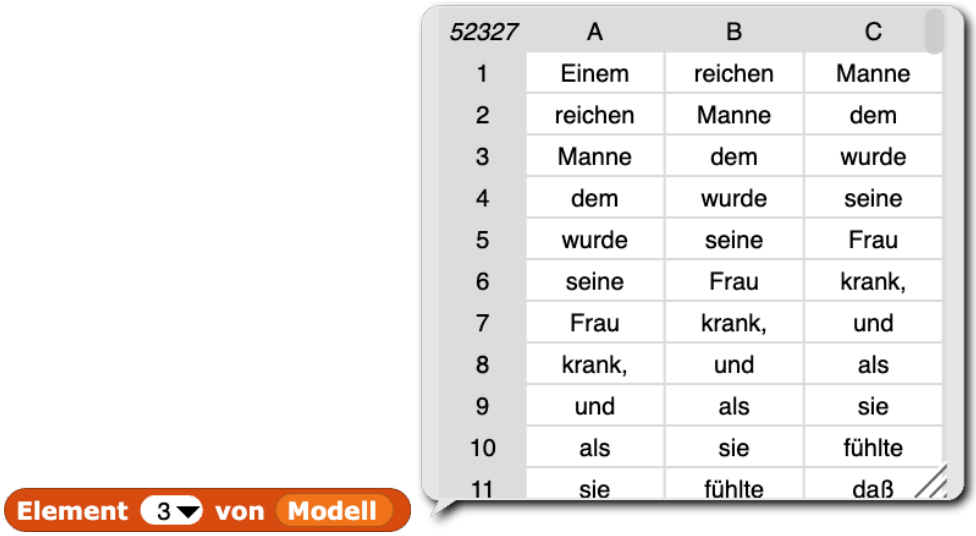


Abbildung 4: Inspizieren des Sprachmodells in Snap!
Die Anzahl der Spalten sollte dem jeweiligen Index entsprechen

Üblicherweise wird nun die Häufigkeit jeder Wortfolge ermittelt und im Modell gespeichert. Zur Vereinfachung habe ich auf diesen Schritt verzichtet, und schreibe stattdessen eine mehrfach angetroffene Wortfolge auch mehrfach ins Modell. Wenn später ein Text zufällig fortgesetzt werden soll, werden häufigere Wortfolgen auch mit einer höheren statistischen Wahrscheinlichkeit gezogen. Hier bietet sich an, die Wahrscheinlichkeitsberechnung aus der Mathematik im Unterricht aufzugreifen.

Tipp: Eine Variable als "nicht persistent" markieren

Wenn man ein Snap! Projekt speichert, dann bleiben auch die Werte aller globalen Variablen erhalten. Das hat den Vorteil, dass man z.B. die Datei mit den Märchen nicht jedes Mal wieder neu importieren muss, wenn man an dem Projekt weiterarbeitet. Ebenso ist es hier mit dem Modell aus n-Grammen. Allerdings kann es sein, dass dadurch das Projekt so große Datenmengen anhäuft, dass es das Speicherlimit für die Cloud übersteigt. Um das zu verhindern können einzelne Variablen, hier z.B. das Modell, in der Palette in ihrem Kontextmenü als "nicht persistent" gekennzeichnet werden. Natürlich muss man dann beim nächsten Öffnen des Projekts wieder das Skript mit den n-Grammen (Abb. 3) ausführen, um das Model wiederherzustellen.

Eine Reihe fortsetzen

Als nächstes geht es darum, angefangene Sätze mithilfe des Sprachmodells zu vervollständigen. Konkret heißt das, dass im Modell nach einem Wort gesucht wird, das man einer Liste aus vorherigen Wörtern hinzufügen kann, ohne dass dabei Kauderwelsch entsteht. Das ist eigentlich eine schwierige Sache, schließlich muss man dafür viel über Sprache, Grammatik, Satzteile, Fälle, Formen und Ausnahmen wissen, und sich auch mit Satzzeichen und Groß- und Kleinschreibung auskennen. Von alledem weiß das Modell jedoch nichts. Es hat aber einen großen Fundus an Wörtern, die schon mal hinter anderen Wörtern gestanden haben. Diese Korrelationen genügen, um einigermaßen echt klingende Sätze zu bilden.

Der Markov-Kette-Textgenerator-Algorithmus nimmt die jeweils letzten „n" Wörter einer angefangenen Geschichte als Kontext, und sucht im Modell zuerst alle Einträge heraus, die genau ein Wort länger sind als der Kontext. Von diesen Einträgen behält er diejenigen, die bis auf das letzte Wort dem Kontext entsprechen. Aus den so ermittelten Kandidaten wird einer zufällig ausgewählt, und dessen letztes Wort als Ergebnis zurückgegeben. Findet sich kein Kandidat, wird die Suche so lange mit einem jeweils kürzeren Kontext wiederholt, bis schließlich ein zufälliges Wort aus der 1-Gramm Liste zurückgegeben wird.

Üblicherweise wird die Kontextlänge für die Suche hin und wieder zufällig gewählt, damit das System „kreativer" wird, und nicht passagenweise die verhackstückten Originaltexte wieder zusammensetzt. Ich habe hier auf dieses Detail verzichtet, damit das Nachprogrammieren noch etwas einfacher wird. Außerdem finde ich es interessant, wenn man aus den erzeugten Sätzen noch ihren Ursprung erkennen kann, d.h. welchem Märchen sie entnommen wurden.

Für die Implementierung als Funktionsblock in Snap! habe ich dieses Mal eine klassische imperative Variante gewählt (siehe Abb. 5).

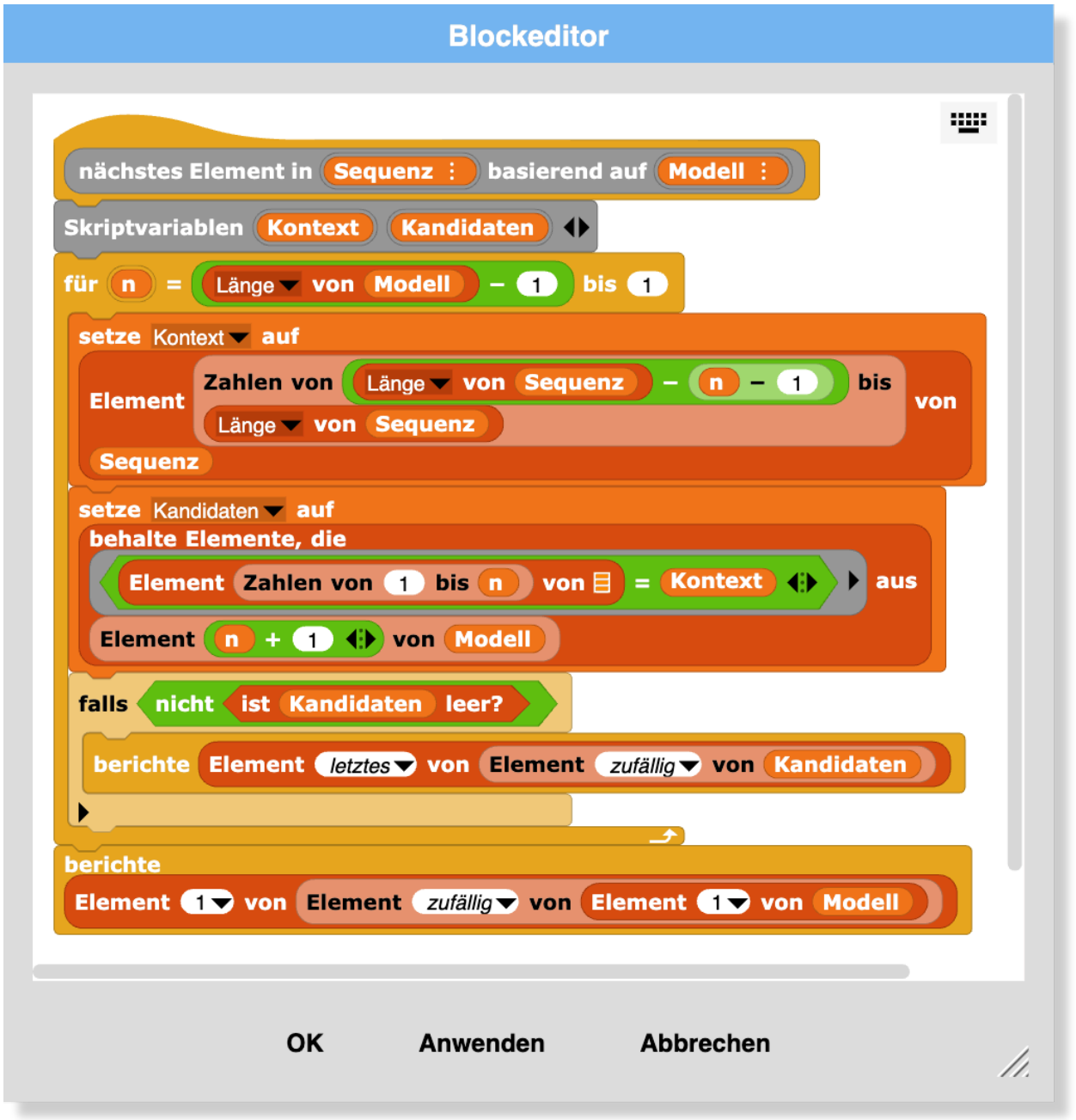


Abbildung 5: Beispiel für eine benutzerdefinierte Abfragefunktion des vereinfachten Markov-Ketten Textgenerators in Snap!

Man kann diesen Block testen, indem man ihm eine Liste aus ein paar Worten und ein Datenmodell übergibt, und dann durch Mausklick ausführt (siehe Abb. 6).

Abbildung 6: Testen des Abfrageblocks in Snap! anhand einer Beispiel-Liste

Tipp: Texte und Listen vergleichen

Snap! unterstützt das Modellieren von Projekten rund um Sprache, indem Listen und Tabellen auf ihre strukturelle Gleichheit getestet werden können. Zwei oder mehrere Listen können unabhängig von ihrem Rang (der Tiefe ihrer Verschachtelung) direkt auf Gleichheit aller ihrer Elemente getestet werden, ohne dass man dafür, wie in anderen Programmiersprachen, die Elemente einzeln anfassen muss. Außerdem ignoriert Snap! beim Vergleichen von Texten standardmäßig die Groß- und Kleinschreibung. Beides folgt dem Vorbild der Programmiersprache LOGO, in der es ursprünglich genau um ebensolche wort- und satzbasierten Sprachprojekte ging.

Ein interaktives Programm gestalten

Aus den vorhandenen Bausteinen kann man jetzt schon ein kleines Programm gestalten, das den Benutzer einbezieht. Dazu habe ich eine weitere globale Variable namens „Geschichte" angelegt, und mir folgende Benutzerführung überlegt: Wenn die grüne Flagge angeklickt wird, soll der Benutzer aufgefordert werden, die ersten Worte eines neuen Märchens einzugeben. Das Programm generiert daraus eine Wörterliste, weist sie der „Geschichte"-Variable zu, ergänzt die Geschichte um das nächste Wort, und gibt sie als Fließtext aus. Jedes Mal, wenn der Benutzer die Leertaste drückt, wird ein weiteres Wort der Geschichte hinzugefügt, und die Ausgabe aktualisiert.

Das fertige Programm besteht aus vier kleinen Skripten, dem oben beschriebenen Modell-Skript (Abb. 3), und einem interaktiven Textgenerator (siehe Abb. 7).

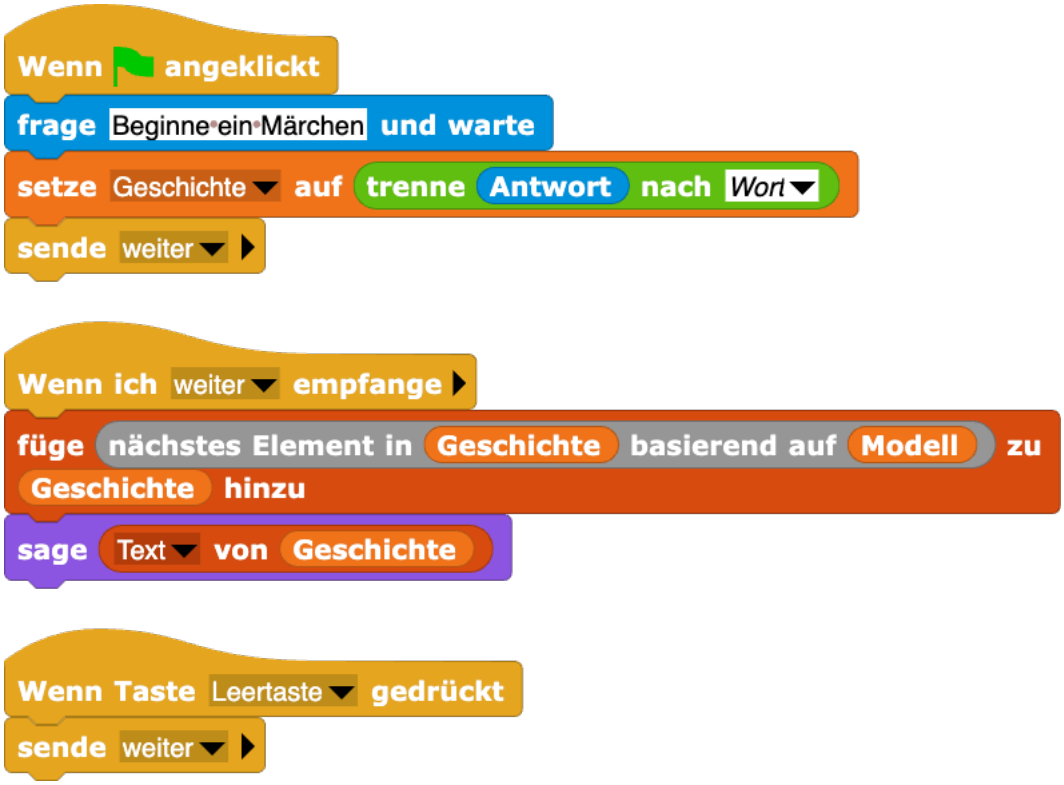


Abbildung 7: Der fertige Textgenerator in Snap!

Wenn man es ausprobiert und auf die Aufforderung, ein neues Märchen zu beginnen, z.B. „Es war einmal“ eingibt, dann entstehen durch wiederholtes Drücken der Leertaste jedes Mal andere Sätze und Geschichten, wie z.B. diese hier:

```
Es war einmal ein König, der hatte eine Frau mit goldenen Haaren, und sie war so unbarmherzig daß sie die arme Rapunzel in eine Wüstenei brachte, wo sie in großem Jammer und Elend leben mußte.
```

Tipp: Eine Wörterliste als Texte ausgeben

Weil das Sprachmodell auf einzelnen Wörtern beruht, ist der Wert der Variable "Geschichte” ebenfalls eine Wörterliste. Um diese als Text auszugeben, müssen alle Wörter miteinander verbunden und dazwischen Leerzeichen eingefügt werden. Das kann man in Snap! auf verschiedene Arten bewerkstelligen. Die einfachste Methode ist, den “Länge von Liste” Block zu nehmen, und anstatt “Länge” die Auswahl auf “Text” zu setzen.

Tipp: Die Bühne etwas “sagen” lassen

In Snap! hat jedes Objekt einen “sage” Block, mit dem es Informationen in einer Sprechblase ausgeben kann, auch die Bühne. Die Sprechblase der Bühne ist größer als die der anderen Objekte und kann einen längeren Ausschnitt der Geschichte anzeigen. Aus allen Sprechblasen kann man durch Rechtsklick den Inhalt als Datei exportieren, z.B. wenn man die erzeugte Geschichte jemandem senden möchte.

Das Modell anpassen

Manchmal kommt es vor, dass das Programm ein Märchen mehr oder weniger ganz abschreibt, oder aber, dass es in eine Art Schleife fällt, und einen Absatz ständig wiederholt. Professionelle Sprachmodelle bieten Einstellungen an, um solche und andere Ungeschicklichkeiten zu vermeiden. Sind die erzeugten Texte zu nah am Original, dann kann man den Zufallsgrad heraufsetzen, mit dem das jeweils nächste Wort bestimmt wird. Das Programm wird dann kreativer, aber auch weniger zuverlässig. In meinem kleinen Snap!GPT Programm lässt sich ähnliches erreichen, wenn man die n-Gramm-Listen im Modell anstatt mit den Zahlen von 1 bis 5 nur bis zur Länge von 2 oder 3 erstellt. Dann verringert sich das sog. Kontextfenster, und der Textgenerator wird abwechslungsreicher, aber auch „fahriger“, und seine Sätze ergeben manchmal keinen Sinn mehr. Auf jeden Fall lohnt es sich, mit diesem Parameter zu spielen, und verschiedene Varianten auszuprobieren.

Häufige Wiederholungen kann man in professionellen Textgeneratoren dadurch verringen, dass man einen „Bestrafungswert“ für Doppelungen innerhalb eines bestimmten Bereichs festlegt. In Snap!GPT könnte man z.B. nur Ergebnisse zulassen, die noch nicht innerhalb der letzten 10 oder 20 Wörter enthalten sind. Ähnliche „Bestrafungsgewichte“ kann man problematischen Wörtern und Passagen erteilen, damit z.B. in Gute -Nacht Geschichten für Kinder keine Schimpfworte vorkommen, und allgemein verbotene Inhalte, z.B. Beleidigungen, rassistische, oder pornografische Formulierungen unterbleiben. So eine Zensur kann z.B. mit einer Stoppwortliste bewirkt werden.

Mich hat – aus didaktischer Sicht positiv - überrascht, dass sich solche aus der gesellschaftlichen Diskussion über generative künstliche Intelligenz bekannten Probleme schon in kleinem Rahmen zeigen. Das gibt Anlass zu Reflexion und Diskussion, nicht nur in Informatikunterricht.

Transfer: Das Prinzip verallgemeinern

Einen Textgenerator selbst zu programmieren macht Spaß und regt zum Nachdenken an. Seine Kreationen sind einerseits verblüffend realistisch, andererseits kompletter Nonsens, durchsetzt mit eindeutig plagiierten Passagen. Man kann anstelle von Märchen andere Vorlagen verwenden, und dann Texte „im Stil” dieser anderen Quellen generieren. Das funktioniert sogar (natürlich!) mit fremdsprachigen Texten. In Snap! kann man beliebig viele Dateien als Variablen

importieren, und im Projekt z.B. diejenige Variable ersetzen, aus der das Sprachmodell erstellt wird. Man kann auch mehrere Modelle mit verschiedenen Inhalten im gleichen Projekt verwalten, und jeweils auswählen, aus welchem das nächste Element ermittelt werden soll.

Eine spannende Erkenntnis ist, dass es überhaupt nicht auf den Inhalt der verarbeiteten Quelldaten ankommt, sondern nur darauf, dass sie in einer bestimmen Reihenfolge sind. Solche Sequenzen gibt es zuhauf, nicht nur in Texten, sondern überall, wo „Listen" im weitesten Sinne Verwendung finden. Einkaufslisten: Leute, die dies gekauft haben, kaufen häufig auch jenes. Playlists: Wer sich das anhört oder ansieht, ist vielleicht auch an jenem interessiert. Krankheitsverläufe: Wer diese Symptome hat leidet oft auch unter jenen. Spielpartien: Nach diesem Schachzug haben erfolgreiche Spieler oft jenen gewählt. Die Bausteine, mit denen Beispieldaten in n-Gramme zerlegt, und unvollständige Reihen plausibel ergänzt werden, müssten folglich auf beliebige andere Zusammenhänge anwendbar sein.

Buchstaben statt Wörter

Statt nach Wörtern kann man die Märchen auch nach Buchstaben trennen. Dazu muss man nur in den beiden „trenne" Blöcken im Modellskript (Abb. 3) und im grüne-Flagge Skript (Abb. 7) die Option „Buchstaben" auswählen. Außerdem muss man den roten „Text von Liste" Block im „Wenn ich 'weiter' empfange" Skript gegen einen grünen „verbinde" Block austauschen, weil ansonsten Leerzeichen zwischen den Buchstaben eingefügt werden. Jetzt einmal das Modell-Skript (Abb. 3) anklicken damit es neu auf der Basis von Buchstaben angelegt wird - das dauert jetzt etwas länger, weil es viel mehr Buchstaben als Wörter gibt -, und dann generiert das Programm - langsam - plausible Wörter. Man kann das deutlich schneller machen, wenn man weniger Quelldaten verwendet. 30 Märchen reichen gerade so aus, um realistische Sätze zu bilden, für plausible Wörter genügt aber schon ein einziges Märchen, oder ein einziger Zeitungsartikel. Dann ist das Sprachmodell viel kleiner und schneller zu durchsuchen.

Musik statt Sprache

Mich hat interessiert, ob sich das Prinzip des „rate mal, was als nächstes kommen könnte" nicht nur sichtbar, sondern auch hörbar machen lässt. Dazu habe ich - in recht mühsamer Kleinarbeit – in einem anderen Snap! Projekt 20 Kinderlieder von „Hänschen klein" über „Fuchs, du hast die Gans gestohlen" zu „Schneeflöckchen, Weißröckchen" als Midi-Noten aufgeschrieben. Ich habe darauf geachtet, immer dieselbe Tonart (C-Dur) zu verwenden, damit die Melodien untereinander ähnlicher sind. Zum Abspielen und Ausprobieren der Melodien kann man in Snap! eine Schleife mit dem „spiele Note" Block verwenden. Der „spiele Note" Block hat zwei Eingaben, die Tonhöhe und die Anzahl der Schläge, wie lange die Note dauern soll. Für jede Note braucht es deshalb eine Liste aus diesen beiden Werten. Ein Lied als Abfolge von Noten wird folglich zu einer zweidimensionalen Liste, bzw. zu einer zweispaltigen Tabelle (siehe Abb. 8).

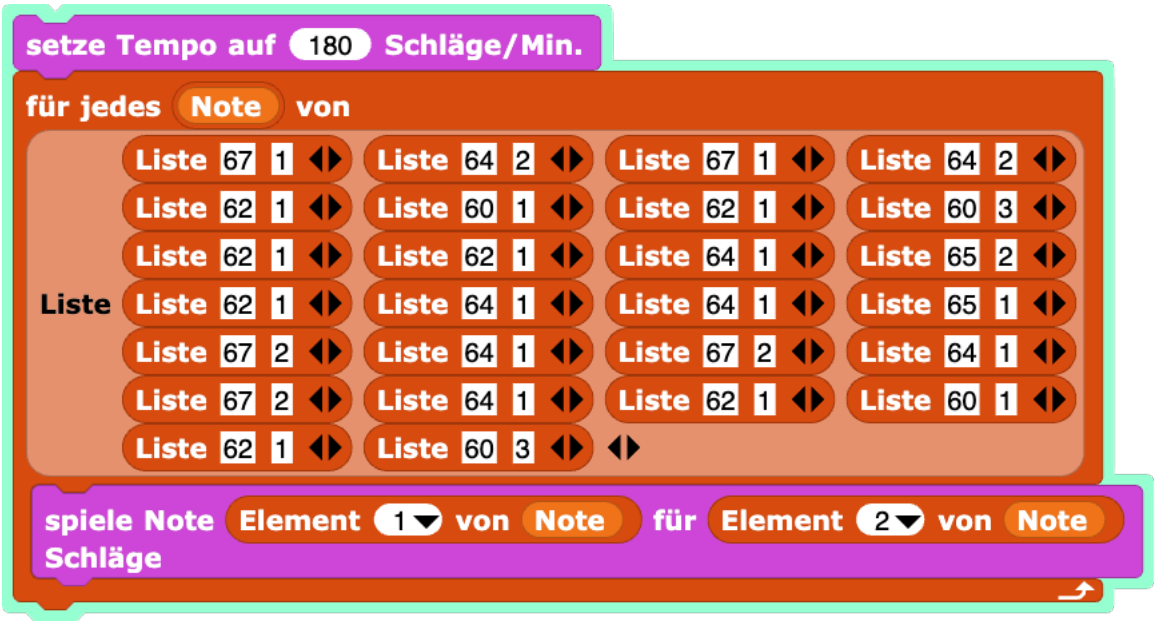


Abbildung 8: Abspielen des Kinderlieds „Kuckuck, Kuckuck, ruft's aus dem Wald" in Snap!

Die einzelnen Lieder habe ich wiederum alle aneinandergehängt, als csv-Datei gespeichert, und in mein SnapGPT Projekt importiert.

Um aus diesen Noten ein Musikmodell zu erstellen, habe ich *dasselbe* Skript wiederverwendet, mit dem ich zuvor das Sprachmodell aus den Märchen generiert habe (Abb. 3). Auch wenn die Daten jetzt etwas anderes bedeuten, und sogar in anderer Form (mehrdimensional) vorliegen, ist das Prinzip dasselbe. Der „n-Gramme" Funktionsblock erwartet lediglich einen Korpus in Form einer Liste. Wie viele Dimensionen diese Liste hat, ist ihm egal.

Damit können nun in Echtzeit Melodien improvisiert werden. Anstelle einer Benutzerführung habe ich dafür eine endlos-Schleife gewählt (siehe Abb. 9).

Abbildung 9: Mit den gleichen Bausteinen können in Snap! sowohl ein Textgenerator als auch eine Melodie-Improvisation programmiert werden

Horizonte Kritzeln

Und was ist mit Bildern? Gibt es da auch ein “Rate mal, was als nächstes kommt”? Anders als bei den Wörtern in einer Geschichte oder den Tönen in einer Melodie sind die Elemente eines Bildes nicht in einer Reihe angeordnet, sondern in einer Fläche. *Zeichnen* ist aber sehr wohl sequenziell: Der Weg, den ein Bleistift über das Papier einschlägt bestimmt sich aus einer Abfolge von Richtungsentscheidungen. Ermutigt von den Melodien, die beim Zusammenwürfeln von zerfledderten Kinderliedern entstanden sind, habe ich in einem kleinen Zeichenprogramm die Richtungsveränderungen in regelmäßigen Abständen eingefangen. Der Einfachheit halber habe ich mich dabei auf die Daten eines einzelnen Strichs beschränkt, also vom Absetzen des Stifts bis zum Anheben. Es entsteht eine Liste aus Zahlen, eine Himmelsrichtung für jeden Teilstrich. Aus diesen Teilstrichen habe ich dann mit der wiederum selben Anweisung wie bei den Märchen und den Kinderliedern ein Datenmodell aus n-Grammen erstellt (siehe Abb. 10).

Abbildung 10: Beispiel für ein Malprogramm in Snap!, das ein Strich-Bild in einer Liste aus Richtungsänderungen aufzeichnet, und daraus ein Datenmodell erstellt

Man kann dieses Programm anklicken und dann mit der Maus oder dem Trackpad auf der Snap!-Bühne etwas Kritzeln oder z.B. in Schreibschrift ein Wort schreiben (siehe Abb. 11).

Abbildung 11: Das Malprogramm kann nur Figuren aufzeichnen, die aus einem einzigen Strich bestehen, z.B. ein Wort in Schreibschrift

Anschließend fantasiert das Modell aus dieser Vorlage eigene Kritzeleien, wenn - abermals - derselbe Abfragebaustein eingesetzt wird, der das nächste plausible Element in einer beliebigen Sequenz errät (sie Abb. 12 und Abb. 13).

Abbildung 12: Mit den gleichen Bausteinen lassen sich in Snap! Geschichten, Melodien und Zeichnungen zusammenfantasieren

Abbildung 13: Beispiel für ein aus dem Schriftzug „hallo“ generiertes Gekritzel

Mit nur einem einzelnen Strich ist man ziemlich eingeschränkt, z.B. darauf, die eigene Handschrift für Schwungübungen imitieren zu lassen. Als wir im Snap!-Team gemeinsam mit meinem Kritzelprogramm gespielt haben, kam mein Freund Bernat Romagosa auf die Idee, es statt mit Kringeln einmal mit Wellen und eckigen Umrissen zu versuchen. Auf diese Weise entstanden

“ordentlichere” Striche, die uns an den Horizont am Meer und in den Bergen, bzw. an die Skyline von Städten und Burgen (siehe Abb. 14) erinnern.

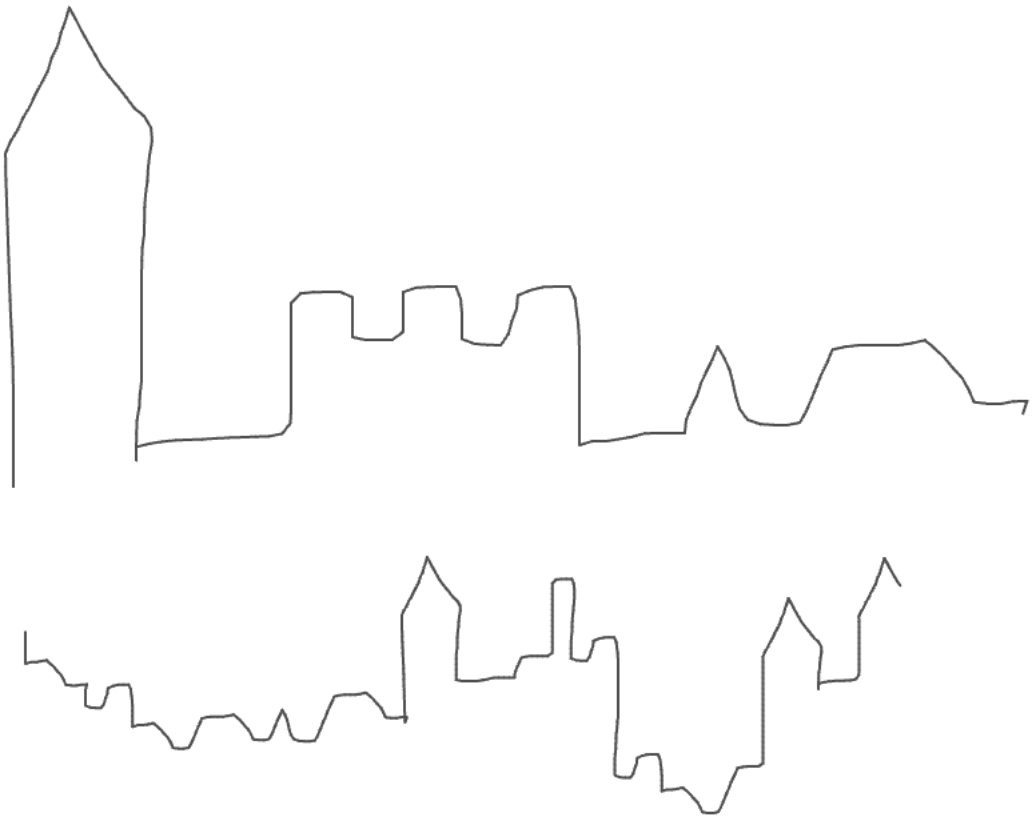


Abbildung 14: Beispiel für einen “Horizont”, oben die von Hand gezeichnete Vorlage, unten eine von Snap! daraus generierte Variation

Fazit

Das fertige Snap! Projekt findet sich unter https://tinyurl.com/SnapGPT-IBIS. Es enthält neben den Programmblöcken auch die Sammlung der 30 Märchen und 20 Kinderlieder. Diese Daten lassen sich einfach in ihrem Kontextmenü (per Rechtsklick auf ihre Anzeige) als txt- oder csv-Datei exportieren, um sie z.B. Lernenden zur Verfügung zu stellen.

Etwas wie ChatGPT im Informatikunterricht selbst zu programmieren, liegt keineswegs außerhalb des Möglichen. Das Projekt lässt sich auf verschiedene Weisen implementieren, und man erwirbt dabei Kompetenzen, die auch anderswo nützlich sind, z.B. das Umformen und Filtern von Daten. Mir persönlich gefällt, dass die Metapher der „Bausteine” in mehrfacher Hinsicht eine Rolle spielt: Die beiden Blöcke für den Textgenerator kann man unverändert auf das Improvisieren einer Melodie und sogar auf das Kritzeln von Silhouetten übertragen, und für die Idee hinter den n-Grammen spielt es keine Rolle, welche Daten damit abgebildet werden.

Herzlichen Dank an Michael Hielscher für die großartige Idee, in das schwierige Thema generative KI didaktisch mithilfe eines Markov-Kette-Textgenerators einzusteigen, für seine wunderschöne SoekiaGPT Software, und für seinen Rat und seine Impulse bei den hier vorliegenden Spielereien in Snap!.

Quellen

Alle Webseiten/Links wurden zuletzt geprüft am 02.12.2023.

Hielscher, Michael, SoekiaGPT – ein didaktisches Sprachmodell (2023), IBIS 01-01-04

Snap! – Build Your Own Blocks (2023), https://snap.berkeley.edu

The Beauty and Joy of Computing (2023), https://bjc.berkeley.edu

Rosetta Code: Markov chain text generator, draft (2023), https://rosettacode.org/wiki/Markov_chain_text_generator

Lizenz

Kontakt

Jens Mönig

SAP, Walldorf

jens.moenig@sap.com

Die senseBox Instruction Placemat zur Förderung einer lernendenzentrierten Vermittlung von Physical Computing

Özkaya, I. T.; Witte, V.; Bartoschek, T.; Pesch, M.
Universität Münster

DOI: 10.18420/ibis-02-01-05

Zusammenfassung

Die wachsenden Anforderungen der sich stetig verändernden Welt verstärken den Ruf nach der Förderung logischer und algorithmischer Denkweisen im Bildungssektor. Die senseBox – ein Toolkit zum Bau und zur Programmierung einer Messstation - stellt eine Möglichkeit dar, um dieser Herausforderung zu begegnen. Beim Einsatz der senseBox im MINT-Unterricht kann Physical Computing auf unterschiedliche Weise vermittelt werden. Eine neuartige und in diesem Paper fokussierte Methode ist die Instruction Placemat. In einer ersten Pilotierung, die mithilfe qualitativer Beobachtungen und einer quantitativen Erhebung mit n = 53 Schüler:innen evaluiert wurde, konnte der Vorteil der Placemat gegenüber einem vortragsgeleitetem Unterricht gezeigt werden.

Einleitung

Die Veränderungen des 21. Jahrhunderts lassen sich in zahlreichen Bereichen des Alltags sowie in einer sich verändernden Berufslandschaft wiederfinden und werden vor allem durch die Prozesse der Ökonomisierung, Globalisierung und Digitalisierung gestärkt (Hofmann/ Schaegner, 2020). Letztere nimmt durch die Möglichkeiten der Automatisierung und künstlichen Intelligenz insbesondere Einfluss auf einen abnehmenden Wert des Faktenwissens und einer zunehmenden Bedeutung der Wissensanwendung zugunsten komplexer Problemstellungen (Fadel et al. 2017). Die Informatik stellt dabei einen Bereich dar, dessen frühzeitige Förderung bedingt durch die genannten Entwicklungen an Bedeutung gewinnt. Im Schuljahr 2022/ 2023 besuchten allerdings nur 24 Prozent der Schüler:innen der Sekundarstufe 1 einen verpflichtenden Informatikunterricht (Hellmig el al. 2023). Daher kann ein transdisziplinärer Ansatz, der die informatische Grundbildung in andere Fächer integriert und mit ihren Themenschwerpunkten kombiniert, eine Möglichkeit darstellen, um die Lernenden dennoch mit den notwendigen Kompetenzen des 21. Jahrhunderts auszustatten. Diesen Grundgedanken verfolgt die senseBox:edu: Ein Citizen Science Do-It-Yourself-Toolkit (DIY-Toolkit), mit dem Schüler:innen selbständig im Bereich des Physical Computing forschen können (Bartoschek et al. 2018). Die

Implementierung in den Unterricht eröffnet die Möglichkeit, neue grundlegende Fähigkeiten und Fertigkeiten des 21. Jahrhunderts, wie z. B. das Programmieren, handlungsorientiert zu vermitteln und mit dem Themenbereich der Umweltbildung zu verknüpfen. Trotz des hohen Potenzials kann die Einarbeitung in das senseBox-Ökosystem (bestehend aus Hardware, Programmierumgebung und Datenplattform) für Lehrende sehr erklärungsintensiv sein. Schüler:innen müssen einige Aspekte beim ersten Zusammenbau und bei der Programmierung beachten, wodurch ein hoher Betreuungsaufwand für die Lehrkraft Zustandekommen kann. In diesem Paper wird dargestellt, inwiefern eine Instruction Placemat in Form und Funktion einer Tischunterlage dazu beitragen kann, den genannten Vermittlungsprozess zu vereinfachen. Bei der Placemat handelt es sich um ein didaktisches und vor allem in der Forschung selten berücksichtigtes Tool, dass die Lernenden bei einem möglichst individuellen und eigenständigen Lernprozess unterstützen soll. Das Hauptziel der Untersuchung stellt die Evaluation der senseBox Instruction Placemat hinsichtlich ihrer Vor- und Nachteile gegenüber vortragsgeleitetem Unterricht dar. Damit sollen Potenziale zur Optimierung von Lernszenarien mit der senseBox eruiert werden. Im Folgenden werden somit die theoretischen Erkenntnisse zu Physical Computing sowie zur Instruction Placemat aufgezeigt, sodass beide Themenbereiche bei der Entwicklung der senseBox Placemat sowie ihrer Evaluation berücksichtigt werden können.

Theoretische Grundlagen

Physical Computing

Mit Blick auf die Herausforderung der transdisziplinären Verankerung informatischer Grundkenntnisse im Kontext der Schule, stellt Physical Computing eine Möglichkeit dar, dieser mit theoretischen und praktischen Herangehensweisen zu begegnen. Unter Physical Computing wird ein Prozess verstanden, bei dem Lernende ihre eigene Projektidee prototypisch verwirklichen und bauen, mit Sensorik arbeiten sowie den Programmcode zusammenstellen. Dadurch schaffen sie konkrete, greifbare Produkte aus der realen Welt, die ihrer eigenen Vorstellung entspringen (Grillenberger 2023). Dennoch wird in den deutschen Bildungsstandards der Sekundarstufe 1 das Wort Physical Computing nicht erwähnt. Auch Wörter wie ‘Sensorik’ oder ‘Mikrocontroller’ finden keine Berücksichtigung (Gesellschaft für Informatik 2008). Dieser Erkenntnis lässt sich die Situation in der Schweiz gegenüberstellen, da der dortige Lehrplan bereits 2016 „explizit den Einsatz von Sensoren, Aktoren und Robotern im Unterricht empfiehlt, um Informatik greifbar und weniger abstrakt zu machen“ (übersetzt aus Grillenberger 2023, S. 226; D-EDK 2016). Genau diese Praxis ist notwendig, denn der beim Physical Computing am häufigsten beobachtete Prozess ist das „Tüfteln“. Dies trägt beim Experimentieren mit Materialien insbesondere zum Erkenntnisgewinn und zur Problemlösefähigkeit bei (Doorley 2012). Charakteristisch für das Physical Computing ist unter anderem auch die Einnahme einer neuen Rolle. Lernende werden beim Entwerfen und Realisieren ihrer Projekte, selbst wenn sie angeleitet werden, zu Erfinder:innen, was wiederum

nach Grillenberger (2023) zur Kreativität, zum Wissenserwerb und zum kreativen Lernen beitragen kann (ebd.). Dieser Lernprozess kann mit Materialien, wie Instruction Placemats, unterstützt werden.

Die Instruction Placemat

Die Arbeit mit Instruction Placemats findet in aktuellen wissenschaftlichen Untersuchungen wenig Berücksichtigung, jedoch ist diese Methode sprachlich paronym zur Placemat-Methode. Daher muss hier zunächst eine deutliche Abgrenzung erfolgen: Während die Placemat-Methode eher eine kooperative Arbeitsform darstellt, in der ein Austausch von Informationen zwischen den Schüler:innen stattfindet, ist die Instruction Placemat eine Informationsquelle sowie Scaffoldingmaßnahme in Form und Funktion einer beidseitig bedruckten (somit wendbaren) Tischunterlage, die das selbständige Lernen unterstützen soll (Beaugrand et al. 2017; Willner-Giwerc et al. 2021).

Willner-Giwerc et al. (2021) entwickelten bereits eine eigene Instrution-Placemat und analysierten in einer empirischen Studie den Einfluss der Placemat beim Arbeiten mit LEGO®-Robotik-Bausätzen. In der Studie wurde ein Vergleich des Arbeitsverhaltens der Schüler:innen zu verschiedenen Robotikaufgaben mit und ohne Instruction Placemat durchgeführt (ebd.). Die Studie basiert auf Videobeobachtungen und zeigt, dass Lernende mit Hilfe der Placemat eine höhere Lösungsvielfalt aufweisen. Die Gruppe, der keine Instruction Placemat zur Verfügung gestellt wurde, wies diese Lösungsvielfalt nur in einem geringeren Maße auf (ebd.). Laut der Cognitive Flexibility Theory hat eine zunehmende Variation in den Lösungen einen positiven Einfluss auf das Lernen (Cheng/ Koszalka 2016). Indem die Lernenden unterschiedliche Lösungswege kennenlernen und angehen, bauen sie sich ein methodisches Repertoire auf, mit dem sie in der Lage sind, Inhalte aus verschiedenen Perspektiven zu betrachten und zu erforschen. Das hierdurch geschaffene „flexible thinking“ trägt zur kognitiven Auseinandersetzung mit dem Unterrichtsgegenstand bei und ermöglicht ein tiefergehendes Lernen (ebd.).

Neben den Disparitäten hinsichtlich der Lösungen, konnte Willner-Giwerc et al. (2021) ebenfalls zeigen, dass die Instruction Placemat zur Differenzierung beiträgt. Leistungsschwächere Schüler:innen nutzten Hilfestellungen und Tipps, die auf der Instruction Placemat zur Verfügung gestellt wurden. Leistungsstärkeren Schüler:innen stand wiederum das Angebot zur Verfügung, gänzlich auf die Instruction Placemat zu verzichten oder nur wenige Hilfestellungen zu nutzen. Die Differenzierung wurde unter anderem dadurch ermöglicht, dass auf der Rückseite weitere Impulse und Tipps zu finden waren. Willner-Giwerc et al. (2021) gaben keine Arbeits- und Programmierschritte vor. Stattdessen bauten sie Impulsfragen in die Instruction Placemats ein, die die Schüler:innen in eine bestimmte Richtung lenken sollten. Des Weiteren enthielt die Instruction Placemat unscharfe Abbildungen, sodass die Lernenden die Bauanleitung lediglich als Orientierung ansahen. Dies hat den Vorteil, dass der Arbeitsauftrag geöffnet wird und die Lernenden zum entdeckenden Lernen animiert werden, wodurch sie wiederum „selbständig, explorativ und aktiv den Problemlöseprozess“ (Rinschede/ Siegmund 2020, S. 219) bewältigen. Die durch die Instruction Placemat ermöglichte Form des Lernens beruht auf der Lerntheorie

des Konstruktivismus und trägt somit auch zum autonomen und aktiven Lernen bei (Reinmann 2013).

Besonders häufig griffen die Schüler:innen bei der Programmierung der Robotik-Bausätze auf die Instruction Placemat zurück (Willner-Giwerc et al. 2021). Dennoch kam es vor, dass die Lehrkraft bei Fragen angesprochen wurde. In dieser Hinsicht wies die Instruction Placemat einen Vorteil auf, da die Lehrkraft direkt auf entsprechende Bereiche der Placemat zeigen konnte, die diese Frage hätten beantworten können (ebd.). Somit erfolgt einerseits eine Verringerung des Redeanteils und andererseits ermöglicht dies eine Entlastung der Lehrkraft (ebd). Trotz der Erfolge von Willner-Giwerc et al. (2021) stellt Grillenberger (2023) fest, dass das Physical Computing in der Schule auf kleine, stark angeleitete Aktivitäten begrenzt ist. Hierbei greifen einige der Lehrkräfte auf die darbietende Aktionsform zurück (Siegmund/ Rinschede 2020). Die Lehrkraft hält den Schüler:innen folglich einen Vortrag und macht den Arbeitsauftrag anschließend exemplarisch vor (ebd.). Die Lehrkraft fungiert somit als Informationsquelle des Unterrichts, während die Lernenden das Ziel verfolgen, den Erklärungen der Lehrkraft zu folgen. Durch diese Abhängigkeit erfolgt jedoch ein rezeptiver Lernvorgang, der selbständiges und erforschendes Lernen nur begrenzt oder gar nicht zulässt (ebd.). Ein weiteres potenzielles Risiko ist das variable Lerntempo und Niveau der Lernenden sowie zunehmende Konzentrationsschwierigkeiten, auf welche die Lehrkraft Rücksicht nehmen muss. Die genannten Faktoren sprechen in ihrer Gesamtheit für einen Einsatz von Instruction Placemats bei Aktivitäten im Rahmen des Physical Computing.

Entwicklung der senseBox Instruction Placemat

Grundlage: Das senseBox-Ökosystem

Wie vorab beschrieben, ermöglicht Physical Computing eine transdisziplinäre Vermittlung von zukunftsweisenden Kompetenzen, wie Kreativität und algorithmisches Denken. Ein Tool, das aufgrund der Kombination aus haptischer und digitaler Handhabung ebenfalls zur Teildisziplin des Physical Computing zählt und die genannten Kompetenzen anspricht, ist die senseBox. Das senseBox-Ökosystem gliedert sich in drei grundlegende Bereiche: Die Hardware bildet einen Bausatz bestehend aus einem Arduino kompatiblen Mikrocontroller, verschiedenen Sensoren (z.B. Temperatur, Luftfeuchtigkeit, CO2), Datenübertragungsmodulen (WiFi, LoRa, Bluetooth, Ethernet) und weiterem Zubehör (siehe Abb. 1).

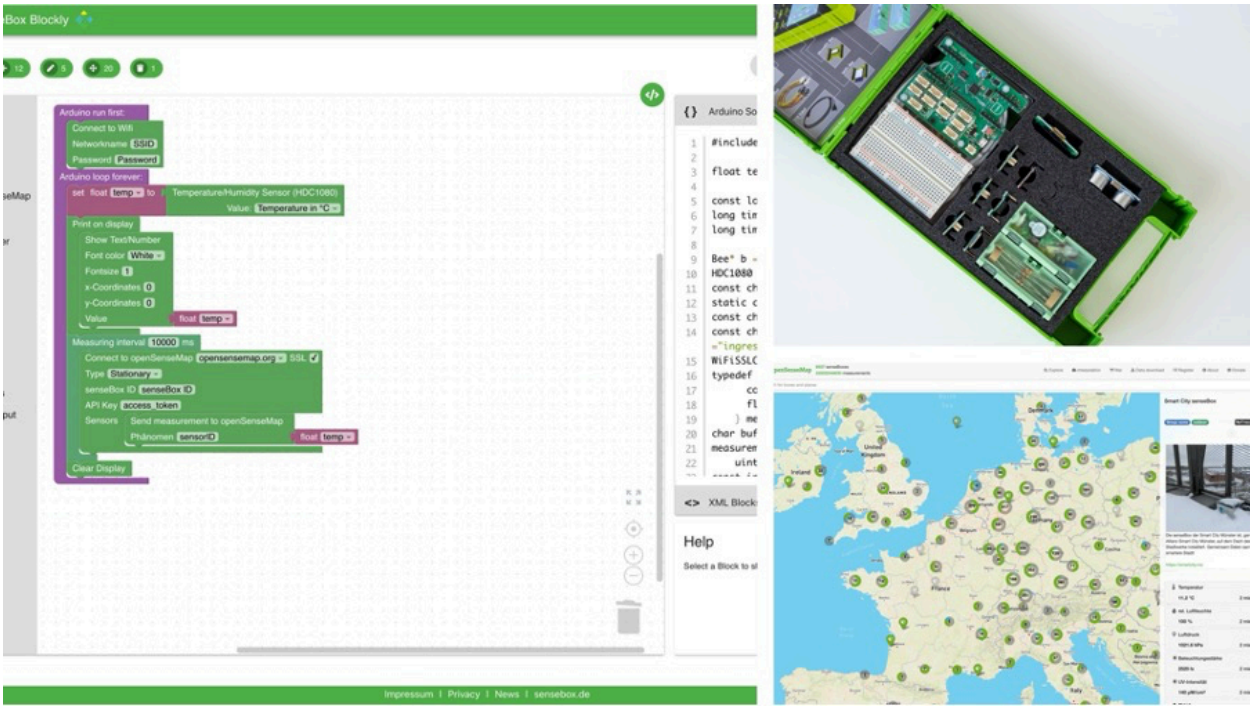


Abbildung 1: Das senseBox-Ökosystem: Blockly für senseBox, senseBox:edu, openSenseMap

Mithilfe eines einfachen Stecksystems können individuelle Messgeräte gebaut werden, beispielsweise eine kleine Wetterstation oder eine CO2-Ampel. Im nächsten Schritt werden die Messgeräte programmiert. Dafür steht die webbasierte, grafische Oberfläche ‚Blockly für senseBox' zur Verfügung (blockly.sensebox.de). Sie ermöglicht einen unkomplizierten Einstieg in die Welt der Programmierung, da die Blöcke wie Puzzleteile per Drag-and-drop zu einem Programmcode zusammengefügt werden können. Vorgefertigte Schablonen, ähnlich zum Prinzip von Linn (1985), reduzieren dabei die Komplexität logischer Verknüpfungen und Operationen und vermeiden Syntaxfehler. Ein integrierter Code Editor ermöglicht zudem textbasierte Programmierung und Anpassung des Codes der Blöcke für Fortgeschrittene. Der programmierte Bausatz kann nun mithilfe der angeschlossenen Sensorik (Umwelt-)daten erheben und diese mithilfe der verfügbaren SD-, WiFi- oder Bluetooth-Bees speichern oder versenden. Zur Sammlung der erhobenen Daten steht die IoT-Umweltdatenplattform openSenseMap zur Verfügung (openSenseMap.org). Im Sinne des Citizen Science Ansatzes können Lernende auf die offene Datenplattform zugreifen, selbst Daten erheben und hochladen, sowie mit vorhandenen Daten arbeiten oder sich bestehende Datensätze herunterladen. Insbesondere in Zeiten von Big Data und mit Blick auf die Förderung der Datenkompetenz von Lernenden, stellt diese Software eine bedeutende Ergänzung des senseBox-Ökosystems dar (Pfeil et al. 2015).

Neben den genannten Aspekten der Konzeption und Programmierung der Hardware, wird der Schwerpunkt beim Arbeiten mit der senseBox zudem auf die Projektplanung und -durchführung gelegt. Indem Lernende ihre Projekte eigenständig planen, bauen und programmieren, schaffen sie konkrete, greifbare Produkte aus der realen Welt, die ihrer eigenen Vorstellung

entspringen. Beim Physical Computing mit der senseBox gelingt es somit, neben der Vermittlung informatischer Fähigkeiten unter anderem auch die Struktur des Lernens und des Unterrichts zu verändern. Um darauf reagieren zu können, sind geeignete Lehr- und Lernmaterialien notwendig. Zur senseBox existieren bereits zahlreiche Anleitungen und Projektbeispiele als Open Educational Resources oder auch als integrierte Tutorials in die Programmieroberfläche[1], jedoch bedarf es bei ihrer Nutzung im Unterrichtsgeschehen teilweise einer intensiven Betreuung durch die Lehrperson. Neue Entwicklungen weisen jedoch auf eine Instruction Placemat als Alternative zugunsten eines autonomen Lernprozesses der Lernenden und somit einer Unterstützung der Lehrkraft bei der Arbeit mit digitalen Medien hin (Willner-Giwerc et al. 2021).

Die senseBox Instruction Placemat

Neben der Studie von Willner-Giwerc et al. (2021) finden sich in der englisch- und deutschsprachigen Literatur nahezu keine vergleichbaren Ansätze zum Thema 'Instruction Placemats'. Trotz dieser Forschungslücke bietet es sich aufgrund der vielversprechenden Ergebnisse und der Notwendigkeit zur zeitgemäßen Gestaltung des Unterrichts an, eine eigene Instruction Placemat mit Bezug zur senseBox zu entwickeln und zu evaluieren.

Der Grundgedanke der senseBox Instruction Placemat (SIP) basiert auf dem Konstruktivismus und dient den Schüler:innen als Unterstützungsangebot beim eigenständigen Zusammenbau und der Programmierung der senseBox. Die SIP besitzt eine Vorder- und Rückseite (siehe Abb. 2 & 3). Auf der Vorderseite der SIP ist die Aufgabenstellung, Bau- und Programmieranleitung sowie ein Bereich zum Testen zu finden. Jede SIP besitzt eine eigene Aufgabenstellung und bezieht sich dementsprechend auf ein Projekt. Die SIP wurde exemplarisch für den Zusammenbau und die Programmierung einer Umweltmessstation entwickelt, was insbesondere für den Einstieg ein niedrigschwelliges Projekt darstellt. Instruction Placemats zu anderen senseBox-Projekten verfolgen einen ähnlichen Aufbau und unterscheiden sich lediglich in den einzelnen Schritten und Hinweise zur Programmierung. In der Mitte der SIP befindet sich die Bauanleitung der Hardware mit entsprechenden Platzhaltern (grafische Repräsentationen der senseBox Bauteile), sodass die Lernenden ohne Vorerfahrung eigenständig die Hardware auf der SIP ablegen können und gemäß Abbildung die Sensorik an den Mikrocontroller anschließen können. Um ein Hin- und Herblättern zwischen der Vorder- und Rückseite zu vermeiden, befinden sich die Bauanleitung und Platzhalter sowohl auf der Vorder- als auch auf der Rückseite der Placemat (siehe Abb. 2).

Oben links auf der SIP ist die Programmieranleitung gegeben, die sich entlang des (im westlichem Kulturkreis gegebenen) Leseflusses von links nach rechts auf der Fläche fortsetzt. Sie besteht aus einzelnen Schritten, die neben schriftlichen Erklärungen und Hinweisen unteren anderem auch graphische Abbildungen, z. B. von Codeblöcken, zur Unterstützung enthält. Jeder Schritt stellt einen Programmiergedanken dar, wie z. B. das Initialisieren eines Displays, das Messen eines Sensorwertes oder das Einstellen einer Variable. Die Schritte der Anleitung

[1] OER: https://sensebox.de/de/material; Tutorials: https://blockly.sensebox.de/tutorial

orientieren sich somit am Prinzip der Elementarisierung: Eine Reduktion der Komplexität erfolgt durch Weglassen von zusätzlichem, überforderndem Material und das Einhalten einer einfachen, verständlichen Sprache (Kretschmner/ Stary 1998). Die Anleitung zeigt den Lernenden, welche Befehle sie für ihren Programmcode benötigen, jedoch wird ihnen nicht vorgegeben, wo sie diese Befehle in der Weboberfläche von Blockly für die senseBox finden können. Dadurch wird bezweckt, dass die Schüler:innen selbstständig die einzelnen Rubriken der Programmieroberfläche erforschen und kennenlernen. Um eine Differenzierung des Lernprozesses herzustellen, enthält jede SIP eine Rückseite. Zum einen sind hier Hilfestellungen, zum anderen aber auch zusätzliche Aufgaben zu finden. Diese Scaffoldingmaßnahmen sind nur dann zu sehen, wenn sich die Lernenden bewusst dazu entscheiden, die SIP umzudrehen und die Hilfestellungen in Anspruch zu nehmen. (Willner-Giwerc et al. 2021; Belland 2013). Zusätzlich wird hiermit die Lehrkraft entlastet, die bei Fragen auf die Tipps hinweisen und die Lernenden bei der Ausprägung ihrer Problemlösekompetenz unterstützen kann.

Neben den genannten Ähnlichkeiten zur Robotik Instruction Placemat, weist die SIP auch deutliche Disparitäten und Neuerungen auf. So enthält letztere auf der vorderen rechten Seite Informationen zu den Sensoren. Die Informationen beziehen sich unter anderem auf die Funktionsweise und den Messbereich der Sensoren. Hiermit erhalten die Schüler:innen einen Einblick in die Hardware der senseBox und ihre Funktionsweise. Eine weitere Ergänzung ist die Zusatzaufgabe, die sich an besonders gute und schnelle Schüler:innen richtet. Diese Aufgabe ist auf der Rückseite zu finden und bietet den Lernenden die Möglichkeit, ihr bereits programmiertes Projekt zu erweitern und zu verbessern. Als Inspiration dient ihnen ein vorgeschlagener Programmcode, der ebenfalls auf der Rückseite platziert ist. Auf Grundlage dessen können alle Schüler:innen in ihrem persönlichen Tempo mit der senseBox arbeiten (Rinschede/ Siegmund 2020).

Die SIP weist jedoch nicht nur unterstützende, sondern auch interaktive Merkmale auf. Dies wird beispielsweise durch den Bereich ‘Testen’, der auf der Vorderseite der Instruction Placemat zu finden ist, erzielt. Hier sollen Lernende ihr zusammengebautes und programmiertes Projekt direkt ausprobieren. Dazu wird ihnen bei der Umweltmessstation eine Tabelle zur Verfügung gestellt, sodass die SIP neben ihrer Funktion als Informationsquelle und Unterstützungsmöglichkeit ebenfalls eine Arbeitsoberfläche bietet. Durch die Kombination aus Aufgabenstellungen, Informationen sowie Aufbau- und Programmieranleitungen werden alle Informationen gebündelt dargestellt und es ist neben der SIP kein zusätzliches Medium notwendig.

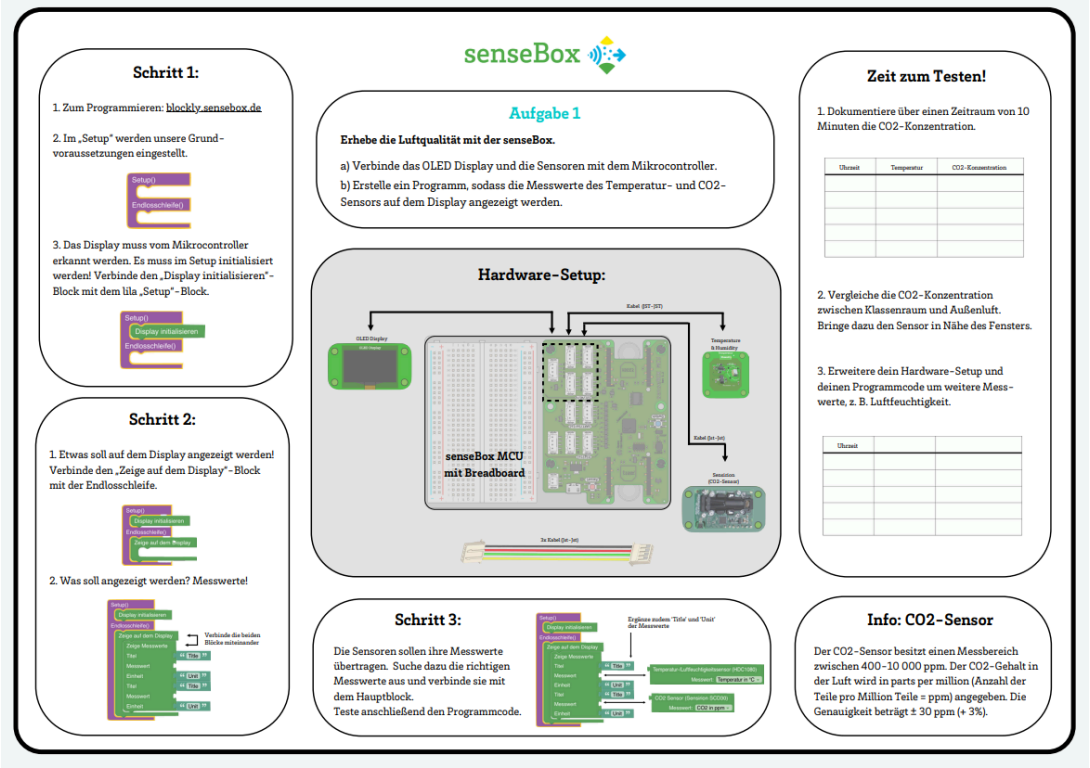


Abbildung 2a: Vorderseite der senseBox Instruction Placemat

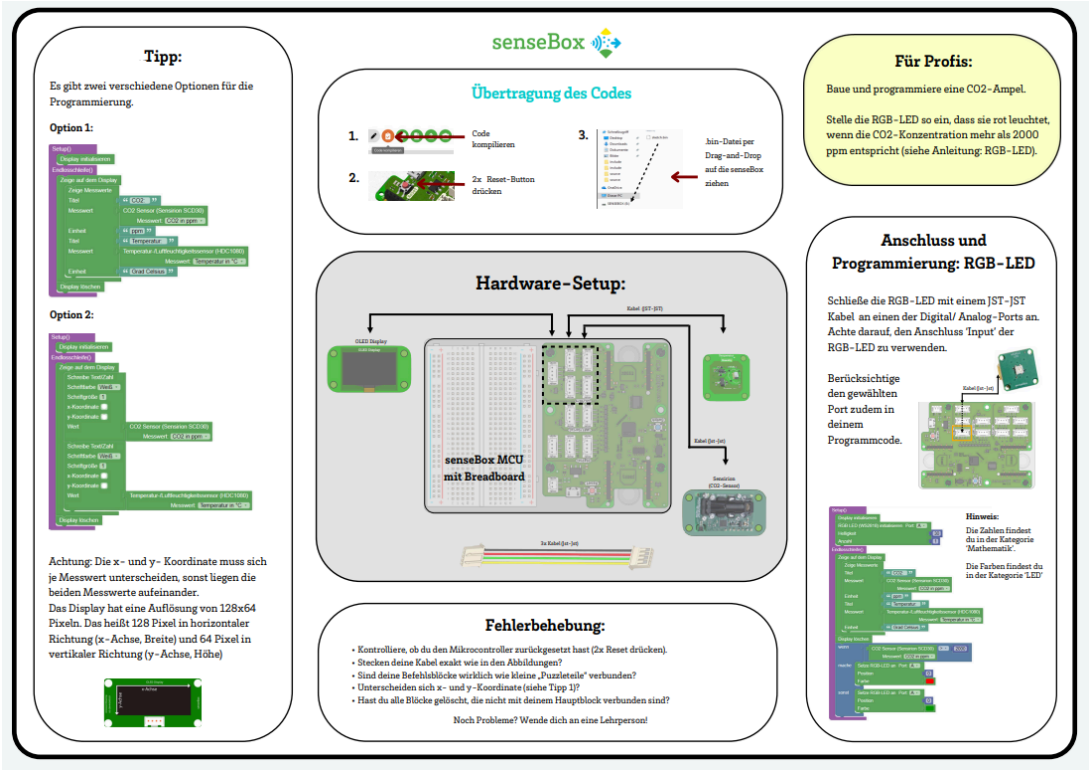


Abbildung 2b: Rückseite der senseBox Instruction Placemat

Evaluation der senseBox Instruction Placemat

Konzept & Vorgehen

Um den Einsatz der entwickelten SIP in der Schulpraxis zu evaluieren, wurde ein Vergleich zwischen Unterricht mit Instruction Placemat und Unterricht mit darbietender Aktionsform hergestellt. Die Stichprobe bilden n = 53 Schüler:innen der neunten und zehnten Jahrgangsstufe einer Real- und Sekundarschule. Die Schüler:innen durchlaufen ein Tutorial, welches die Grundkenntnisse (z.B. Übertragung des Programmcodes) der senseBox vermittelt. Anschließend folgen zwei Einheiten, in denen die Lernenden zum einen eine Umweltmessstation und zum anderen einen Verkehrszähler mit der senseBox zusammenbauen und programmieren. Dabei unterscheidet sich der didaktische Rahmen bei den beiden Projekten und zielt auf den Vergleich des Unterrichts zwischen mit und ohne Instruction Placemat ab. Die Unterschiede werden in einer teilstrukturierten Beobachtung konstatiert, die von zwei unabhängigen Personen durchgeführt werden. In einer anschließenden qualitativen Inhaltsanalyse nach Mayring und Fenzl (2014) werden schließlich die Unterschiede zwischen den beiden Unterrichtsformen herausgearbeitet. Neben einer Beobachtung erfolgt zudem auch ein quantitatives Vorgehen mittels eines Fragebogens. Den Fragebogen erhalten die Teilnehmenden am Ende der dreistündigen Evaluation. Durch die Befragung kann ergänzend zur Beobachtung ein Einblick in die Motivation, das Verständnis und die Selbstwirksamkeit der Lernenden zu den zwei unterschiedlichen Szenarien gewonnen werden.

Ergebnisse und Diskussion

Im Allgemeinen konnten deutliche Disparitäten zwischen dem Unterricht mit und ohne Placemat sowohl qualitativ als auch quantitativ konstatiert werden. Diese Disparitäten zeigten sich einerseits im Arbeitsverhalten, der Selbstwirksamkeit, Motivation sowie der verbalen Kommunikation der Lernenden und andererseits im Redeanteil und Arbeitspensum der Lehrkraft. In einigen Fällen wirkten sich diese Unterschiede im Positiven auf die Phasen mit SIP aus. Dies wird insbesondere dann deutlich, wenn im Folgenden ein genauerer Blick auf verschiedene Teilbereiche gelegt wird.

Entlastung der Lehrkraft

Die SIP führt zur Entlastung der Lehrkräfte, indem weniger persönliche Unterstützung erfolgen muss. Die Forschungsergebnisse haben hinsichtlich des Arbeitspensums und des Redeanteils der Lehrkraft eindeutige Erkenntnis erbracht. In den Beobachtungen wird an mehreren Stellen deutlich, dass die Phasen ohne SIP erklärungsintensiv sind. Die Ergebnisse der Beobachtung aus den Phasen mit SIP zeigen wiederum ein anderes Bild. Aus allen Beobachtungsbögen ist zu entnehmen, dass der Redeanteil der Lehrkraft gering ist, da die Instruction Placemat den Bereich der Vermittlung und individuellen Unterstützung übernimmt. Anhand mehrerer Beispiele konnte konstatiert werden, dass die Lernenden auf der Instruction Placemat nach Lösungen suchen. Des Weiteren war der Austausch der Schüler:innen untereinander und somit

auch eine gegenseitige Unterstützung in den Phasen mit SIP größer. Hiermit wird die Lehrkraft deutlich entlastet, da sie den Lernenden einerseits weniger erklären und andererseits weniger individuell unterstützen muss. Selbst in Situationen, in denen persönliche Unterstützung von der Lehrkraft verlangt wurde, konnte beobachtet werden, dass die Lehrkraft auf die Tipps der SIP hingewiesen hat. Dies stellt eine effiziente Art der Unterstützung dar und trägt andererseits zum selbständigen Problemlösen bei.

Erfolgt ein Einblick in die quantitativen Daten, so wird dieser Eindruck bestätigt. Es konnten signifikante Unterschiede in den Phasen mit und ohne Placemat festgestellt werden. Demnach wurden der Lehrkraft in den Phasen mit Placemat signifikant weniger Fragen gestellt. Verglichen mit der Arbeit von Willner-Giwerc et al. (2021), die ebenfalls eine Instruction Placemat entwickelten, unterstützt diese Erkenntnis somit auch ihre Beobachtungen, was den Redeanteil und die Entlastung der Lehrkraft betreffen (siehe Abb. 3).

werden. Demnach wurden der Lehrkraft in den Phasen mit Placemat signifikant weniger Fragen gestellt. Verglichen mit der Arbeit von Willner-Giwerc et al. (2021), die ebenfalls eine Instruction Placemat entwickelten, unterstützt diese Erkenntnis somit auch ihre Beobachtungen, was den Redeanteil und die Entlastung der Lehrkraft betreffen (siehe Abb. 3).

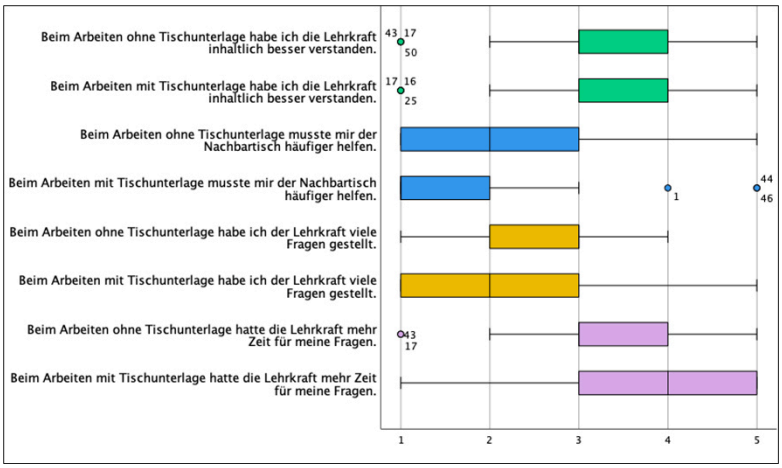


Abbildung 3: Boxplots zu den einzelnen Items zum Einfluss der Lehrkraft

Förderung diverser Lösungsvorschläge

In der Studie von Willner-Giwerc et al. (2021) trug die Instruction Placemat zu diverseren Lösungsvorschlägen bei. Sowohl in der Beobachtung als auch in der Befragung konnte dieses Merkmal nicht in einem aussagekräftigen Ausmaß identifiziert werden. Es wurde anhand der Beobachtung aber deutlich, dass die Schüler:innen ohne Instruction Placemat nicht außerhalb des Instruktionsraums gearbeitet haben. Die Lernenden folgten den Anweisungen der Lehrkraft, was wiederum dazu führte, dass wenig Raum für flexiblere Ergebnisse geschaffen wurde. Vermuten lässt sich, dass die Schüler:innen das vorgestellte Lehrbeispiel als einzigen Lösungsweg interpretiert haben, was wiederum der These von Blikstein (2015) entspricht. Hinzu kommt,

dass ihnen wenige Möglichkeiten geschaffen wurden, um sich untereinander auszutauschen: Hiermit wird eine Grundvoraussetzung blockiert, mit der die Lernenden eigene Ideen kreieren können. Nun könnte die Annahme getroffen werden, dass die Schüler:innen mit SIP deutlich freier arbeiten konnten und sich dementsprechend unterschiedliche Lösungen gebildet hätten. Doch auch dieser Gedanke trifft auf die Einheiten mit SIP nicht überwiegend zu. Zwar konnte mehrfach in den Beobachtungen festgestellt werden, dass die Lernenden selbständiger arbeiten, indem sie sich miteinander austauschen und eigenständig die Probleme lösen, allerdings enthält die Placemat einige Anweisungen und Orientierungshilfen, mit denen es den Schüler:innen ermöglicht wird, ihre Projekte zu realisieren. Dies bedeutet nicht, dass jeder einzelne Schritt auf der SIP vorhanden ist und dass es sich um eine Schritt-für-Schritt-Anleitung handelt. Dennoch werden die Schüler:innen in eine gewisse Richtung gelenkt, die möglicherweise auch Raum für alternative Lösungsvorschläge verhindert.

Stärkung der Selbstwirksamkeit

Die SIP weist positive Effekte auf die Selbstwirksamkeit und das Verständnis der Schüler:innen im Umgang mit der senseBox auf. Es lässt sich annehmen, dass die Schüler:innen mit SIP selbständiger gearbeitet haben. Selbständiges Arbeiten bedeutet jedoch nicht zwangsläufig, dass es selbstwirksam ist. Indikatoren für positive Selbstwirksamkeit können Ehrgeiz und Beharrlichkeit sein. Dies prägt sich insbesondere dann aus, wenn die Lernenden etwas selbständig und ohne Hilfestellung meistern möchten. Dieses Verhalten konnte ebenfalls in den Phasen mit SIP und weniger in den Phasen ohne SIP festgestellt werden. In den Beobachtungen konnte zudem häufiger ein moderates Verständnis in Einheiten mit SIP gemessen werden. Fraglich ist jedoch, ob sich das Verständnis objektiv beobachten lässt. Ein gutes Verständnis wurde in den Beobachtungen beispielsweise dann impliziert, wenn weniger Fragen gestellt wurden, der Fachwortschatz korrekt genutzt wurde und wenn aktiv mitgearbeitet wurde. Diese Tendenzen konnten im Fragebogen vertieft und ausgeweitet werden, wo die Schüler:innen die Möglichkeit hatten, ein persönliches Feedback über ihr erworbenes Wissen zu geben. Hierbei konnten die bereits festgestellten Tendenzen in deutlicher Hinsicht bestätigt werden. Zudem wurde gezeigt, dass die Schüler:innen eine Lösung der Aufgaben mit Instruction Placemat gegenüber ohne Instruction Placemat weniger kompliziert empfunden haben (siehe Abb. 4). Im Umkehrschluss lässt sich dadurch also ebenfalls ein besseres Verständnis mit Instruction Placemat ableiten (Kumar/ Jagacinski 2011).

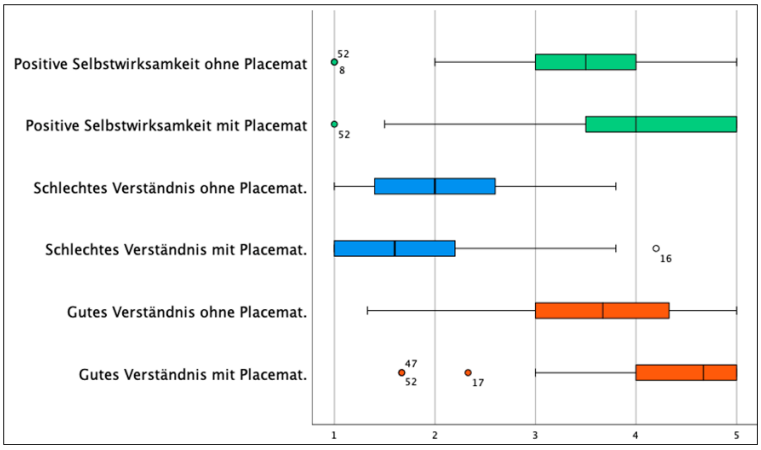


Abbildung 4: Boxplots zu den einzelnen Konstrukten mit und ohne Placemat

Förderung der Differenzierung

In den Unterrichtsstunden ohne SIP konnte in fast allen Beobachtungen festgestellt werden, dass der Unterricht bei offenen Fragen stagniert. Während einigen Schüler:innen geholfen wird, müssen die restlichen Lernenden zwangsläufig warten, bis die Lehrkraft wieder vor dem Plenum steht. Dies schafft Raum für Konzentrationsschwierigkeiten und Unterrichtsstörungen. Verglichen mit den Unterrichtsstunden mit Instruction Placemat, lassen sich deutliche Disparitäten erkennen. Bezugnehmend zur didaktischen Perspektive der Differenzierung liegt eine Differenzierung nach Zeit vor. Indem die Schüler:innen in ihrem eigenen personalisierten Tempo arbeiten können, wird gewährleistet, dass vor allem schwächere Schüler:innen nicht überfordert werden. Andererseits können leistungsstärkere Lernende bereits kognitiv herausfordernde Aufgaben erledigen, ohne auf die anderen Lernenden warten zu müssen. Während der Beobachtung zeigte sich jedoch insbesondere in den Phasen mit SIP, dass leistungsstärkere Schüler:innen nicht ausgelastet waren. Obwohl die SIP eine Sprinteraufgabe enthielt, die für leistungsstärkere und schnelle Schüler:innen konzipiert war, hat diese scheinbar nicht ausgereicht. Neben der Sprinteraufgabe enthielt die SIP auch einen Bereich mit Tipps zur Fehlerbehebung. Diesen sollten die Schüler:innen bei Bedarf nutzen. Den Beobachtungen zufolge wurde diese Form der Unterstützung von einigen der Lernenden genutzt, dennoch gab es auch hier Schüler:innen, die dies nicht beachteten und stattdessen die Lehrkraft um Hilfe baten. Daraus lässt sich schließen, dass zwar Differenzierung angeboten wurde, diese jedoch nicht von allen Lernenden angenommen bzw. wahrgenommen wurde.

Eine weitere Form der Differenzierung, die in Unterrichtsstunden mit Instrucion Placemat identifiziert werden konnte, ist die kollaborative Differenzierung. Indem die Schüler:innen selbständig arbeiten konnten und ihnen Freiraum gegeben wurde, wurde Raum für fachlichen Austausch und gegenseitige Unterstützung unter den Schüler:innen geschaffen. Dies konnte in den Unterrichtsphasen ohne SIP deutlich weniger beobachtet werden, da aufgrund des vortragsgeleitetem Unterrichts Gespräche unterbunden wurden. Anhand der genannten Beispiele wird deutlich, dass Differenzierung in den Phasen mit SIP häufiger und in größerer Variation

stattgefunden hat. Es konnten demnach zwischen der Arbeit mit und ohne SIP einige Disparitäten konstatiert werden.

Fazit

In Anbetracht der genannten Aspekte lässt sich hinsichtlich der Evaluation der senseBox Instruction Placemat festhalten, dass die Placemat die Lernenden in einem hohen Maß zum selbständigen Arbeiten animiert. Im Vergleich zum vortragsgeleitetem Unterricht ohne SIP, weist das Pendant mit SIP deutlich mehr Vorteile auf. Hierbei stellen die Differenzierung, das Verständnis und die Selbstwirksamkeit einige der Vorteile dar. Zwar weist die SIP noch einige Schwächen auf, wie das halboffene Format, jedoch ist jede Unterrichtsstunde von Individualität geprägt, weshalb die SIP von Klasse zu Klasse und von Unterrichtsthema zu Unterrichtsthema angepasst werden muss. Erst so wird garantiert, dass sie in all ihren Zügen zum Vorteil der Schüler:innen agiert. In den Beobachtungen wurde jedoch deutlich, dass die Instruction Placemat die Schüler:innen in eine gewisse Richtung gelenkt hat und so keine alternativen Lösungsvorschläge beobachtet werden konnten. Eine weitere Öffnung des Unterrichts sowie eine weniger strukturierte Instruction Placemat hätten dies eventuell ermöglicht, jedoch hätte dies zu einer Überforderung bei leistungsschwächeren Schüler:innen führen können. Die eruierten Schwächen der entwickelten SIP sollten in nachfolgenden Erprobungen näher betrachtet werden. Zudem bedarf es einer produktunabhängigen Evaluation von Instruction Placemats, die ihre Effektivität im Bereich des Physical Computing im Allgemeinen bewerten.

Literatur

Bartoschek, T., Wirwahn, J. & Pesch, M. (2018). Sensebox und opensenseMap - Umweltmonitoring für Jedermann. In: Fretag, Ul., Fuchs-Kittowski, F., Hosenfeld, F., Abecker, A. & Reineke, A. (Hrsg.), Umweltinformationssysteme 2018 - Umweltbeobachtung: Nah und Fern. Nürnberg: CEUR-WS.org.

Beaugrand, A., Latteck, Ä.-D., Mertin, M. & Rolf, A. (2017). Lehr- und Lernmethoden im dualen Studium Wissenstransfer zwischen Theorie und Praxis. 1. Auflage, W. Kohlhammer GmbH, Stuttgart.

Belland, B. R. (2013). Scaffolding: Definition, Current Debates, and Future Directions. In Handbook of Research on Educational Communications and Technology (S. 505–518). Springer New York. https://doi.org/10.1007/978-1-4614-3185-5_39.

Blikstein, P. (2015). Digital fabrication and ‚making' in education the democratization of invention. In: FabLabs: Of Machines, Makers and Inventors, No. September, (S.203-221).

Cheng, J. & Koszalka, T., A. (2016) Cognitive Flexibility Theory and its Application to Learning Resources. Syracuse University, RIDLR project (concept paper). Zuletzt online am 24. März 2023 unter: http://ridlr.syr.edu/publications/.

Deutschschweizer Erziehungsdirektoren-Konferenz [D-EDK]. (2016). Lehrplan für die Volksschule – Broschüre Medien und Informatik. Zuletzt online am 06. Juni 2023 https://v-ef.lehrplan.ch/container/V_EF_DE_Modul_MI.pdf.

Doorley, R. (2012). What is tinkering? Zuletzt online am 06. Juni 2023 unter: https://tinkerlab.com/what-is-tinkering/.

Fadel, C., Bialik, M. & Trilling, B. (2017): Die vier Dimensionen der Bildung. Was Schülerinnen und Schüler im 21. Jahrhundert lernen müssen. Hamburg.

Gesellschaft für Informatik (GI) e.V. (2008). Grundsätze und Standards für die Informatik in der Schule. Bildungsstandards Informatik für die Sekundarstufe I. Bonn, Berlin.

Grillenberger, M. (2023). Why and How to Teach Physical Computing: Research and Practice in Computer Science Education at Secondary Schools. In: Keane, Z & Fluck, A. E. [Hrsg.] (2023). Teaching Coding in K-12 Schools. Research and Application. Springer, Cham, Schweiz, (S.225-245).

Hellmig, L., Schieckoff, B., Schwarz, R., Süßenbach, F. (2023). Informatik-Monitor 2023/24. Zur Situation des Informatik-Unterrichts in Deutschland. Berlin.

Hofmann, J. & Schaegner, A. (2020): Digital und demokratisch – was zeitgemäße schulische Bildung leisten kann. In: Friedrichsen, M. & Wersig, W.(Hrsg.) (2020): Digitale Kompetenz Herausforderungen für Wissenschaft, Wirtschaft, Gesellschaft und Politik. Wiesbaden, (S. 161-171).

Kretschmer, H. & Stary, J. (1998), Schulpraktikum. Eine Orientierungshilfe zum Lernen und Lehren (Studium Kompakt), Cornelsen, Scriptor-Verlag, Berlin.

Kumar, S. & Jagacinski, C., M. (2011). Confronting Task Difficulty in Ego Involvement: Change in Performance Goals. In: Journal of Educational Psychology 2011, Vol. 103, No. 3, (S.664–682).

Linn, M., C. (1985). The Cognitive Consequences of Programming Instruction in Classrooms. In: Educational researcher, 1985, Vol.14 (5), (S.14-29).

Mayring, P. & Fenzl, T. (2014). Qualitative Inhaltsanalyse. In: Bauer, N. & Blasius, J. [Hrsg.] (2014). Handbuch Methoden der empirischen Sozialforschung. Springer Fachmedien Wiesbaden 2014. (S.543-559).

Pfeil, M., Bartoscheck, T. & Wirwahn, J. A. (2015) OPENSENSEMAP - A citizen Science Platform for Publishing and Exploring Sensor Data as Open Data.

Reinmann, G. (2013). Didaktisches Handeln. Die Beziehung zwischen Lerntheorien und Didaktischem Design. In: Ebner, M. [Hrsg.]; Schön, S. [Hrsg.]: L3T. Lehrbuch für Lernen und Lehren mit Technologien. 2. Auflage. 2013, [12] S. RN: urn:nbn:de:0111-opus-83381 - DOI: 10.25656/01:8338.

Rinschede, G. & Siegmund, A. (2020). Geographiedidaktik. Grundriss Allgemeine Geographie. 4. Auflauge. Ferdinand Schöningh Verlag, Paderborn.

Willner-Giwerc, S., Danahy, E. & Rogers, C. (2021). Placemat Instructions for Open-Ended Robotics Challanges. In: Lepuschitz, W., Merdan, M., Koppensteiner, G., Balogh, R. & Obdrzalek, D. (2021) Robotics in Education, (S.234-244).

Lizenz

Kontakt

Ilhan Talha Özkaya
Verena Witte
Dr. Thomas Bartoschek
Mario Pesch

Universität Münster

E-Mail: ilhanoezkaya@outlook.de; {verena.witte | bartoschek | mario.pesch}@uni-muenster.de

Den k-Means-Algorithmus verstehen: Mit Stift & Papier und BlueJ

Andres, D.; Joachim, S.; Hennecke, M.
Universität Würzburg

DOI: 10.18420/ibis-02-01-06

Zusammenfassung

Dieser Praxisbeitrag stellt unterrichtliche Aktivitäten vor, um den *k-Means-Algorithmus* einzuführen und ihn anschließend in *BlueJ* zu implementieren. Anhand eines Beispieldatensatzes mit Fundkoordinaten von Pilzen können sich die Schülerinnen und Schüler die Funktionsweise des *k-Means-Algorithmus* selbst erschließen. Der Datensatz ist zweidimensional und klein genug, um den Algorithmus mit Stift und Papier in angemessener Zeit zu erarbeiten. Anschließend kann der Algorithmus in der den Schülerinnen und Schülern sowie Lehrkräften bekannten Umgebung *BlueJ* programmiert und das Ergebnis durch Einlesen einer csv-Datei mit den vorher verwendeten Fundkoordinaten verifiziert werden.

Einleitung

Im Moment wird täglich in der Öffentlichkeit über Künstliche Intelligenz berichtet und heftig diskutiert, ob diese eher eine Chance oder eine Bedrohung für unsere Gesellschaft darstellt. Zur Versachlichung der Diskussion besteht die Möglichkeit zu vermitteln, dass auch KI-Systeme lediglich auf Algorithmen beruhen. Einige modernere Lehrpläne haben die Thematik bereits aufgegriffen und integrieren bekannte Algorithmen aus dem Bereich der Künstlichen Intelligenz in ihre Curricula. Ein Beispiel ist der bayerische LehrplanPLUS für die Jahrgangsstufen 11 bis 13.

Bei KI-Systemen unterscheidet man zwischen dem wissensbasierten und dem datenbasierten Ansatz (auch *maschinelles Lernen*). Teilbereiche des *maschinellen Lernens* sind u.a. das *Reinforcement Learning*, das *Supervised Learning* und das *Unsupervised Learning*. Beim *Unsupervised Learning* ist das Ziel, Gruppen oder Zusammenhänge in nicht *gelabelten* (kategorisierten) Daten zu finden. Dazu müssen Daten zunächst aufbereitet und normalisiert werden, bevor sog. Clusteralgorithmen auf den Datensatz angewendet werden können. Die Ergebnisse dieser Algorithmen müssen im Anschluss interpretiert und evaluiert werden. Ein typischer Clusteringalgorithmus ist der *k-Means-Algorithmus*. Er ist beispielsweise im bayerischen LehrplanPLUS der 13. Jahrgangsstufe sowohl im grundlegenden als auch im erhöhten Anforderungsniveau als Muster für einen Clusteringalgorithmus genannt: „Die Schülerinnen und Schüler […]

beschreiben die Funktionsweise des k-Means-Algorithmus als Beispiel unüberwachten Lernens und implementieren diesen für ein Beispiel.“ (ISB o.J.)

Der *k-Means-Algorithmus* durchläuft nacheinander drei Phasen: Zuerst werden *k* Clusterzentren auf den Koordinaten von *k* zufälligen oder manuell gewählten Datenpunkten initialisiert. Danach erfolgt die Zuweisung aller Datenpunkte zu dem Clusterzentrum, das ihnen am nächsten ist. Anschließend werden Clusterzentren aktualisiert. Dafür werden für jedes Clusterzentrum jeweils komponentenweise die Mittelwerte der Koordinaten aller Datenpunkte eines Clusters berechnet. Diese sind die Koordinaten des aktualisierten Clusterzentrums. Falls sich die Position eines Clusterzentrums geändert hat, werden die Schritte Aktualisierung der Clusterzentren und Zuweisung wiederholt, bis keine Änderung mehr eintritt (vgl. Ertel 2021). Der *k-Means-Algorithmus* kann sowohl mit numerischen als auch kategoriellen Daten beliebiger Dimension ausgeführt werden. Bei Kategorien ist eine andere Metrik zur Abstandsmessung erforderlich, während bei numerischen Daten mit der euklidischen Norm gearbeitet werden kann. Aus Gründen der didaktischen Reduktion beschränkt sich das folgende Unterrichtskonzept auf den zweidimensionalen Fall mit ausschließlich numerischen Daten. Die Merkmale werden deshalb nachfolgend als x und y bezeichnet.

Es existieren verschiedene Möglichkeiten diesen Algorithmus konkret zu implementieren. Unter dem *k-Means-Algorithmus* nach Lloyd (1982) wird meist die Umsetzung des oben genannten Schemas verstanden: Erst nach vollständiger Zuweisung aller Datenpunkte zum nächsten Clustermittelpunkt (oder auch Clusterzentrum) erfolgt eine Aktualisierung der Clusterzentren. Als Abstandsmaß wird der euklidische Abstand verwendet. Im Gegensatz dazu steht z. B. der *k-Means-Algorithmus* nach MacQueen (1967), bei dem die Aktualisierung der Clusterzentren nach jeder Punktzuweisung erfolgt. Damit benötigt der Algorithmus nach MacQueen zwar wesentlich mehr Laufzeit, terminiert im Schnitt allerdings nach weniger Iterationen als der Algorithmus nach Lloyd.

Allgemein kann nicht sichergestellt werden, dass der *k-Means-Algorithmus* überhaupt terminiert oder die tatsächlich beste Clusterung findet. So bildet der *k-Means-Algorithmus* aufgrund der Mittelwertberechnung z. B. bevorzugt konvexe, gleich große Cluster (vgl. Ertel 2021 und Abbildung 2).

Unterrichtskonzept

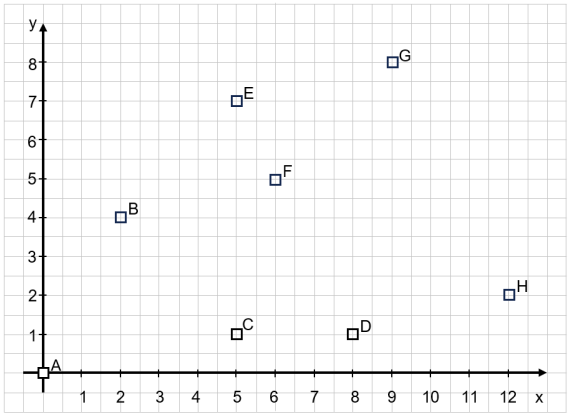


Abbildung 1: Koordinatensystem mit Pilzdatensatz

Wer in der Natur Pilze sammelt, hat vermutlich schon beobachtet, dass Pilze einer Art häufig gruppenweise zusammenstehen. Die Fundkoordinaten der acht Pilze A bis H sind im Koordinatensystem eingetragen. In diesem Beispiel wurden die Koordinaten bewusst so gewählt, dass sich rein optisch verschiedene Pilzgruppen nicht erkennen lassen (vgl. Abbildung 1).

Die unterrichtliche Fragestellung ist zunächst, wie man die Einteilung in eine vorgegebene Anzahl an Gruppen automatisieren könnte. Für dieses Problem wird der *k-Means-Algorithmus* als Lösungsvorschlag vorgestellt, den die Schülerinnen und Schüler nach einer kurzen Einführung selbst manuell auf den kleinen Pilzkoordinatendatensatz anwenden sollen. Dabei werden zwei Zentren bei der Initialisierung vorgegeben. Die Schülerinnen und Schüler erhalten so bei richtiger Durchführung des Algorithmus alle das gleiche Ergebnis und gewinnen Vertrauen in ihre Fähigkeiten.

Der bayerische LehrplanPLUS fordert zudem eine konkrete Implementierung des *k-Means-Algorithmus*. Eine Sprache für die Umsetzung ist nicht vorgegeben. Eine kompakte Implementierung des *k-Means-Algorithmus* über vorgegebene Methodenaufrufe, die beispielsweise das Berechnen des Abstands zwischen zwei Punkten oder das Finden des Minimums innerhalb einer Liste ermöglichen, ist z. B. in den Programmiersprachen R oder Python möglich. Diese beiden Sprachen haben zudem den Vorteil, dass eine einfache Visualisierung der Daten und Clusterergebnisse mit wenigen Befehlen innerhalb der Sprache für die Schülerinnen und Schüler zur Verfügung steht und somit eine visuelle Unterstützung beim Debuggen des Codes existiert. Allerdings ist die Einführung von R – zumindest in Bayern – im Curriculum der vorhergehenden Jahrgangsstufen nicht vorgesehen. Zudem arbeiten viele Lehrkräfte seit Beginn der textuellen Programmierung mit Java. Eine händische Umsetzung des *k-Means-Algorithmus* mit Java ist ebenfalls recht kompakt möglich, allerdings stellt Java keine ähnlich einfachen Befehle zur Visualisierung zur Verfügung, die Schülerinnen und Schüler schnell eigenständig erstellen und verwenden können.

Deshalb ist das zweite in diesem Beitrag vorgestellte Unterrichtsmaterial ein *BlueJ*-Projekt, in dem eine bereits implementierte Visualisierung existiert. Die Schülerinnen und Schüler können ihren Algorithmus in Lücken einer vorgegeben Klasse eintragen; dann lassen sich die Auswirkungen des von ihnen generierten Codes über wenige Klicks optisch visualisiert in einem zweidimensionalen Koordinatensystem betrachten (vgl. Abbildung 2). Neben der farblich angezeigten Clusterung werden auch die Koordinaten der Zentrumspunkte in der Visualisierung angegeben.

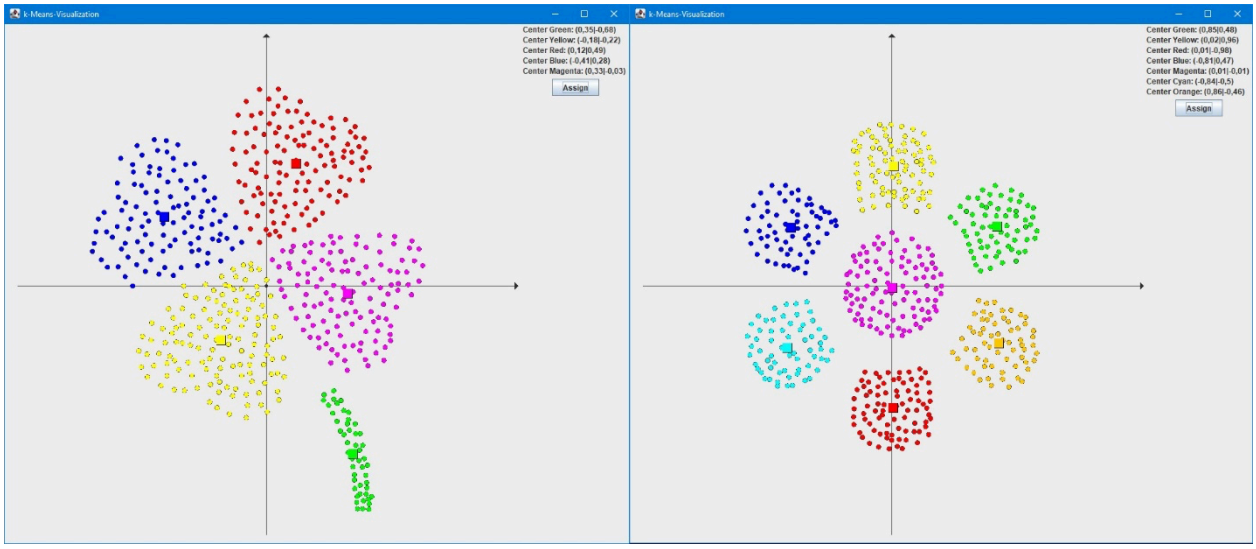


Abbildung 2: Oberfläche der Visualisierung des fertigen Clusterings der Datensätze „kleeblatt.csv“ und „blume.csv“

Material 1: *k-Means-Algorithmus* mit Stift und Papier

Nach einer kurzen Einführung in den Algorithmus führen diesen die Schülerinnen und Schüler mit Hilfe von Arbeitsblättern, verfügbar unter https://go.uniwue.de/ki, selbst durch. Beim ersten Arbeitsauftrag sind die Fundkoordinaten der Pilze bereits in ein Koordinatensystem eingetragen. Die Schülerinnen und Schüler müssen nun immer wieder Pilze dem nächstgelegenen Clusterzentrum zuweisen. Dazu sind verschiedene Möglichkeiten denkbar: So können die Abstände entweder mit dem Lineal gemessen, mit Hilfe des Satzes von Pythagoras berechnet oder dem Zirkel verglichen werden. Um einen möglichst guten Übergang zur Implementation zu erhalten, wurde im **Arbeitsauftrag 1** die **Zuweisung zu einem Clusterzentrum durch Abstandsmessung mit dem Lineal** gewählt. Dabei können die Ergebnisse der Schritte des Algorithmus in Tabellen festgehalten und ausgewertet werden. Mit der gewählten Methode gehen allerdings Messungenauigkeiten einher, welche später nach Überprüfung der Ergebnisse durch das *BluJ*-Projekt thematisiert werden müssen.

Ein Data Scientist analysiert große Mengen an Daten, um bspw. Muster zu erkennen. Eine Möglichkeit der Visualisierung von Mustern in Daten lernen die Schülerinnen und Schüler im **Arbeitsauftrag 2** kennen. Dort zeichnen sie die **Clusterhüllen für den Pilzdatensatz** ein.

Die Zentren bei der Initialisierung sind so gewählt, dass der Algorithmus die gleiche Aufteilung der Pilze in zwei Gruppen erhält, wie beim Supervised Learning mit den KI-Pilzen im Experimentiersatz „Künstliche Intelligenz“ der MEKRUPHY GMBH. Diese Aufteilung entspricht derjenigen, die durch die Label „giftig“ und „essbar“ bereits beim Supervised Learning vorgegeben war. Falls diese den Schülerinnen und Schülern bekannt ist, sollte im Unterricht die Frage thematisiert werden, ob diese Aufteilung auch bei anders gewählten Startzentren durch den *k-Means-Algorithmus* erreicht wird. Die Prognosen können später im *BlueJ*-Projekt überprüft

werden. Dies kann der Einstieg in die Problemstellung sein, dass der *k-Means-Algorithmus* bei unterschiedlichen Startzentren nicht immer zu den gleichen Clustern führt.

Material 2: *k-Means-Algorithmus* mit *BlueJ*

Um den *k-Means-Algorithmus* implementieren zu können, muss zuerst eine geeignete Umgebung bereitgestellt werden. Die Schülerinnen und Schüler benötigen zunächst Beispieldatenpunkte, auf die sie während des Algorithmus z. B. zur Mittelwertberechnung zugreifen können. Dafür müssen im Vorhinein Entscheidungen für die Speicherung der einzelnen Datenpunkte der Beispieldaten getroffen werden. Da dies im *BlueJ*-Projekt bereits vorgegeben ist, müssen die Schülerinnen und Schüler zunächst durch Einstiegsaufgaben an die Projektstruktur herangeführt werden. In **Arbeitsauftrag 3** diskutieren die Schülerinnen und Schüler deshalb verschiedene Modellierungsvorschläge für die Datentypen der Klasse POINT, die Speicherung der Clusterzuweisung sowie die Datenstruktur, die die Eingabedaten hält. Das *BlueJ*-Projekt speichert die Punkte der Eingabedaten als Objekte der Klasse POINT, welche wiederum in einem Array abgelegt werden.

Die Datenstruktur des Arrays wird somit als Vorwissen vorausgesetzt. In Bayern ist diese den Schülerinnen und Schülern je nach Zweigwahl seit der 10. oder 11. Jahrgangsstufe bekannt.

Nach dem Bearbeiten von Arbeitsauftrag 3 können die Schülerinnen und Schüler sich bereits im Projekt zurechtfinden und mit der Implementierung beginnen. Zur Unterstützung der Implementierung kann zusätzlich mit **Arbeitsauftrag 4** Pseudocode der Methoden mit Lücken zur Verfügung gestellt werden. Konkrete Hilfe bei der Implementierung etwa von Wurzelausdrücken oder der Umgang mit einem Random-Objekt, kann **Arbeitsauftrag 5** liefern. Das Projekt steht unter https://go.uniwue.de/ki in zwei Versionen zum Download zur Verfügung, einmal als fertiges, lauffähiges Projekt und einmal als Vorlage für Schülerinnen und Schüler mit Lücken in den Methoden der Klasse KMEANS.

Projektstruktur

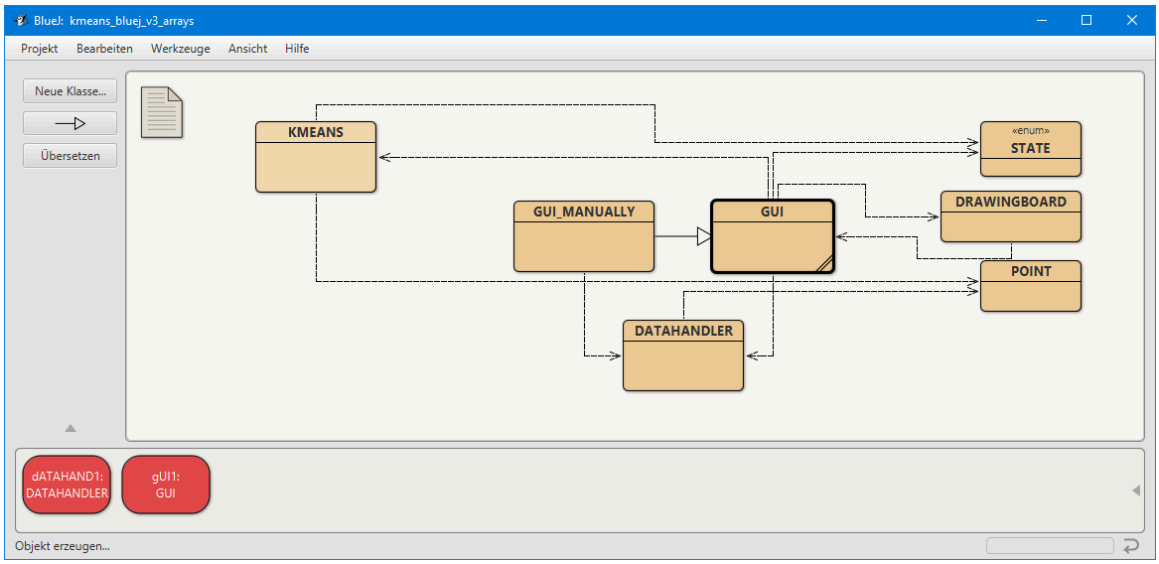


Abbildung 3 Projektstruktur auf BlueJ

Das Projekt besteht aus den sieben Klassen POINT, DATAHANDLER, STATE, GUI, GUI_MANUALLY, DRAWINGBOARD und KMEANS (vgl. Abbildung 3), von welchen lediglich KMEANS, GUI und evtl. DATAHANDLER und GUI_MANUALLY für den Arbeitsablauf der Schülerinnen und Schüler relevant sind.

Beschreibung des *BlueJ*-Projekts

Grundablauf des Programms:

Bei einer vollständig implementierten Klasse KMEANS, kann zum schnellen Testen der Methoden der Konstruktor von GUI aufgerufen werden. Dieser erzeugt einen Datensatz mit vier klar erkennbaren Clustern. Dieser wird in einem neuen Fenster angezeigt. Mit einer Schaltfläche (Button) kann der implementierte *k-Means-Algorithmus* schrittweise ausgeführt werden. Dabei werden beim Standardkonstruktor von GUI stets vier Clusterzentren initialisiert. Diese erscheinen im Koordinatensystem als farbige Quadrate. Nach der Initialisierung der Zentren benennt sich die Schaltfläche in *Assign* um und führt auf Klick die Methode **assign()**der Klasse KMEANS aus. Die Datenpunkte werden so dem ihnen nächsten Zentrum zugewiesen und im Koordinatensystem in der entsprechenden Zentrumsfarbe gefärbt. Hat sich die Zuweisung zu den Clusterzentren geändert, kann mit Klick auf die Schaltfläche *Update* die Methode **update()** ausgeführt und die neuen Koordinaten für die Clusterzentren berechnet werden. Die Zentren im Koordinatensystem verschieben sich auf diese neuen Koordinaten und die Schaltfläche wechselt zurück auf den Schritt *Assign*. So können die Methoden **assign()**und **update()**abwechselnd ausgeführt werden, bis keine Änderung der Clusterzentren mehr sichtbar ist. Ausschnitte eines Durchlaufes des *k-Means-Algorithmus* zeigt Abbildung 4.

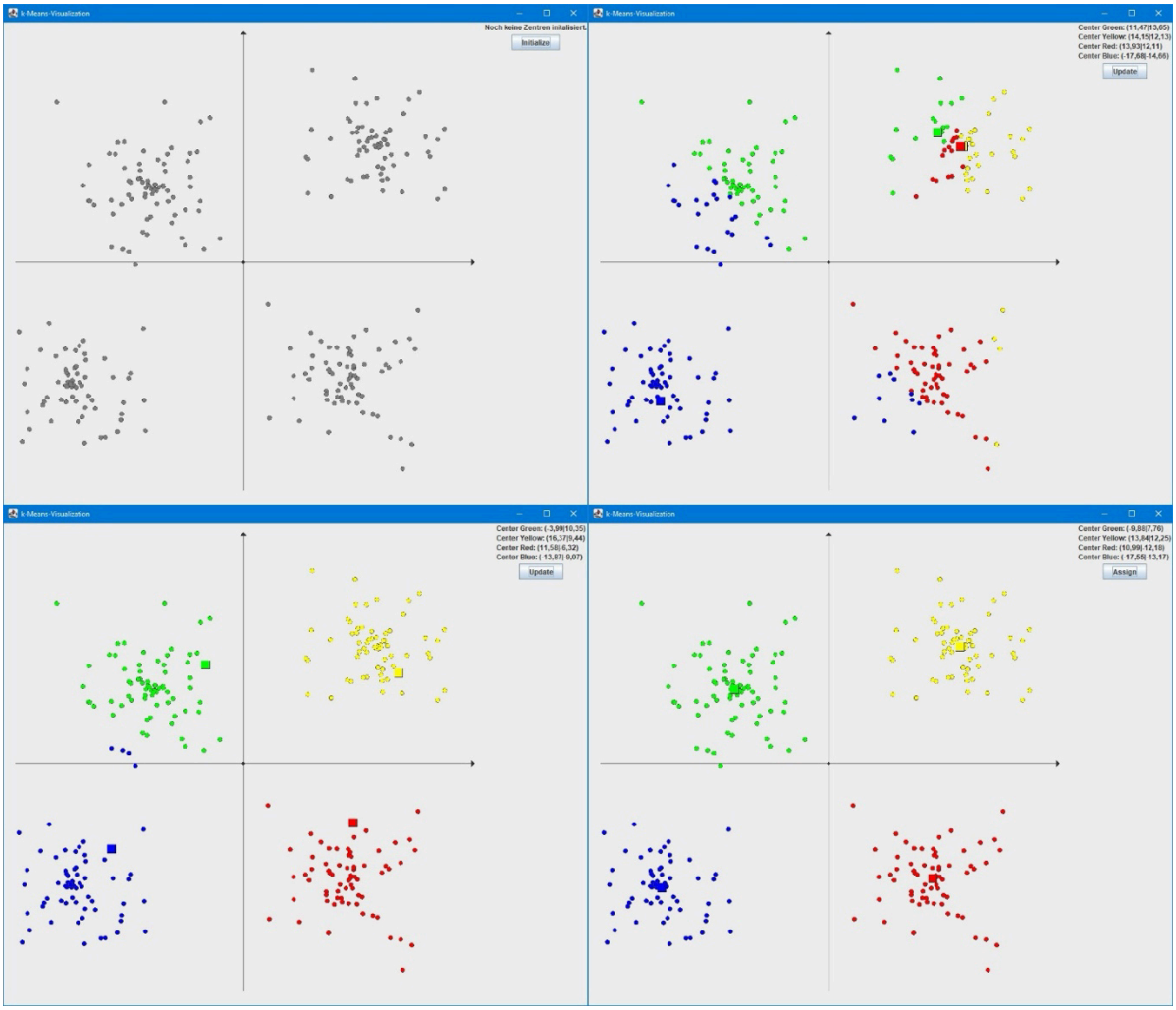


Abbildung 4 Einige Zwischenstände der Visualisierung des k-Means-Algorithmus

Möchte man den Datensatz oder die Anzahl der Cluster variieren, muss der zweite Konstruktor der Klasse GUI verwendet werden. Diesem muss ein zuvor erstelltes Objekt der Klasse DATAHANDLER sowie die gewünschte Clusterzahl *k* übergeben werden. DATAHANDLER verfügt über verschiedene Methoden, um Beispieldatensätze zu erstellen.

Schlussendlich gibt es die Möglichkeit zur Überprüfung der Implementierung die Zentren nicht über die Methode **initialize()**vonKMEANS, sondern händisch per Eingabe der Koordinaten zu initialisieren. Dafür initialisiert man ein Objekt der Klasse GUI_MANUALLY, welchem man immer ein DATAHANDLER-Objekt sowie die gewünschte Clusterzahl *k* übergeben muss. Um an Material 1 anzuschließen können die Koordinaten der Pilze zunächst mittels der Methode **loadData**mit dem Parameter „pilze.csv“ von DATAHANDLER in das Programm geladen werden und anschließend mit dem Konstruktor von GUI_MANUALLY die zwei Startkoordinaten per Hand eingegeben werden. Dies widerspricht zwar der eigentlichen Implementierung des *k-Means-Algorithmus* nach Lloyd, wird aber zur Überprüfung der im Vorhinein ausgeführten manuellen Berechnung mit vorgegebenen Zentren benötigt. So ergibt sich mit dem Pilzdatensatz

und den analog zu Arbeitsauftrag 1a gewählten Startzentren $Z_{grün}$ (5|1) und Z_{gelb} (6|5) eine finale Clusterung mit Zentren $Z_{grün}$ (3,75|1,5) und Z_{gelb} (8|5,5), welche mit der in Material 1 von Hand ermittelten Clusterung im Rahmen der Messgenauigkeit übereinstimmt.

Erwartetes Schülerhandeln

Aufgabe der Schülerinnen und Schülern ist es, den Code des *k-Means-Algorithmus* in der Klasse KMEANS zu ergänzen. Zur Übersichtlichkeit sind dafür bereits drei Methoden **initialize()**, **assign()** und **update()** (in der *BlueJ* Vorlage mit leeren Methodenkörpern) vorgegeben. Nach Implementierung der Methoden, können die Schülerinnen und Schüler die Auswirkungen ihres eigenen Codes grafisch durch den Aufruf des Konstruktors der Klasse GUI überprüfen.

Beschreibung der Klassen

Im Folgenden werden nur die Konstruktoren, Attribute und Methoden aufgeführt, die für die Implementierung des *k-Means-Algorithmus* von Bedeutung sind. Die Beschreibung der im Hintergrund für die Visualisierung zuständigen Klassen ist der Dokumentation des *BlueJ*-Projektes zu entnehmen.

Die Klasse **POINT** repräsentiert einen Datenpunkt oder ein Clusterzentrum und stellt somit das Grundgerüst der Datenstruktur dar.

- Attribute
  - Hält die x- und y-Koordinate des Punktes als Gleitkommazahl.
- Methoden
  - Setter & Getter
  - **getDistance(POINTp): double** berechnet den euklidischen Abstand zwischen diesem Punkt und einem anderen Punkt p.

DATAHANDLER erzeugt die Daten, die mit dem *k-Means-Algorithmus* geclustert werden sollen.

- Attribute
  - Die Liste **pointList**, welche alle Punkte des Datensatzes als POINT-Referenzen enthält.
- Methoden
  - **getRandomData(int countPoint, int max, int min)** bewirkt das Anlegen eines Datensatzes mit **countPoint**-vielen zufälligen Datenpunkten zwischen **max** und **min**.
  - **getRandomCluster(int countCluster, int countPoint, int max, int min)** bewirkt das Anlegen eines Datensatzes mit deutlich erkennbaren Clustern.
  - **loadData(String fileName)** lädt einen in dem Ordner „Beispieldaten“ angelegten Datensatz. Dafür muss der exakte Name des gewünschten Datensatzes z. B. „kleeblatt.csv“ in das Eingabefeld eingetragen werden.
  - **getPointlist(): POINT[]** wandelt die Liste pointList in ein Array um und gibt es zurück. So können die Schülerinnen und Schüler in KMEANS mit der ihnen vertrauten Datenstruktur des Arrays arbeiten.
- Konstruktoren

    - **DATAHANDLER()** erstelltstandardmäßig eine zufällige Clusterung mit vier Clustern und 300 Datenpunkten.

In **KMEANS** wird der *k-Means-Algorithmus* nach Lloyd umgesetzt. In der *BlueJ* Vorlage sind nur die dafür benötigten Methodenköpfe gegeben, die Schülerinnen und Schüler sollen hier ihre eigene Implementation umsetzen.

- Attribute
    - Das Array **POINT[]data** speichert die einzelnen Datenpunkte. Wird KMEANS von GUI übergeben.
    - Clusterzahl **intk**. Wird KMEANS von GUI übergeben.
    - Die Liste **POINT[]centers** hält die aktuellen Zentren des *k-Means-Algorithmus*.
    - Das Array **int[]assignment** speichert die Zuordnung der Datenpunkte zu den Clusterzentren.
- Methoden
    - **initialize()** initalisiert *k* Clusterzentren an zufälligen Datenpunkten.
    - **assign()** weist die Datenpunkte ihrem nächsten Zentrum zu.
    - **update()** aktualisiert die Zentren.
    - **contains(POINT[]array,POINTpoint):boolean** überprüft, ob der Punkt point bereits im Array array vorhanden ist. Wird zur Initialisierung von *k* verschiedenen Clusterzentren benötigt.

DRAWINGBOARD dient zur Umsetzung des Koordinatensystems. Die Klasse hat Methoden zum Zeichnen der Koordinatenachsen sowie der Datenpunkte und Clusterzentren.

STATE dient als Enumeration lediglich zur Verwaltung der Zustände der Visualisierung.

Die Klasse **GUI** verwaltet die Visualisierung.

- Konstruktoren
    - **GUI()** erstellt einen zufälligen, klar in vier Cluster einteilbaren Datensatz und verwendet den *k-Means-Algorithmus* mit Clusterzahl vier. Ermöglicht schnelle Überprüfung der Implementierung.
    - **GUI(DATAHANDLERdatahandler,intk)** ermöglicht das Erstellen einer Visualisierung mittels selbst modifiziertem Datensatz und individuell ausgewählter Clusterzahl *k*. Das Objekt der Klasse DATAHANDLER muss hierbei in *BlueJ* im Vorhinein erstellt und dem Konstruktor übergeben werden.

GUI_MANUALLY verwaltet die gleiche Visualisierung wie GUI, ermöglicht aber die händische Initialisierung der Zentren.

- Konstruktoren
    - **GUI_MANUALLY(DATAHANDLERdatahandler,intk)** ermöglicht das manuelle Eingeben der Startpunkte der Clusterzentren. Analog zu oben muss hier ebenfalls ein Objekt der Klasse DATAHANDLER vorher manuell erstellt werden.

Diskussion der Projektstruktur

Die Implementierung der Klassen POINT und KMEANS des *BlueJ*-Projektes sind angelehnt an die bayerischen illustrierenden Prüfungsaufgaben für *k-Means-Algorithmus*. Auch wenn bei

dem hier lediglich zweidimensionalen Einsatz eine Modellierung der Punkte als zweidimensionales Array denkbar wäre, wurde sich analog zu den illustrierenden Prüfungsaufgaben (gA) für das Einführen einer eigenen Klasse POINT entschieden. Dies hat zum Vorteil, dass der Algorithmus auch für anders strukturierte oder mehrdimensionale Daten mit lediglich kleinen Anpassungen des *BlueJ*-Projektes umgesetzt werden kann. Auch wird durch die Klasse POINT der Umgang mit mehrdimensionalen Strukturen vermieden.

Konkrete Implementierung des *k-Means-Algorithmus*

Einige Feinheiten der Implementierung müssen von der Lehrkraft entschieden werden. Zunächst muss im ersten Schritt der Initialisierung geklärt werden, ob *k* verschiedene Punkte als Zentren initialisieren werden sollen oder zwei Zentren auch am selben Punkt starten können. Im Verlauf des Algorithmus erfahren zwei Zentren, die am selben Punkt starten, immer dieselben Änderungen und agieren somit als wären sie ein einziges Zentrum. Somit hätte man statt zwei Zentren effektiv nur eines initialisiert. Verwendet man Eingabedaten mit mehreren hundert Datenpunkten tritt das Problem mit hoher Wahrscheinlichkeit nicht auf, bei kleineren Datensätzen wie dem Pilzdatensatz ist es aber wahrscheinlich, dass die Schülerinnen und Schüler auf dieses Problem stoßen. Möchte man bei der Implementierung sicherstellen, dass *k* verschiedene Datenpunkte als Startzentren gewählt werden, kann die Methode **contains(POINT[]array,POINTpoint):boolean** der Klasse KMENAS von den Schülerinnen und Schülern genutzt oder auch selbst implementiert werden.

Eine Möglichkeit zur Differenzierung ergibt sich mit der Umsetzung der Methode **update()**. Diese kann mit einer Laufzeit von *O(nk)* analog zum beschriebenen Pseudocode in **Arbeitsauftrag 4** umgesetzt werden. Allerdings gibt es auch effizientere Implementierungen, welche besonders bei großen Datenmengen von Vorteil sein können. Eine weitere Implementierung mit Laufzeit *O(n+k)* wird im *BlueJ*-Projekt als Variante 2 mitgeliefert. Eine solche effizientere Implementierung zu finden, kann für motiviertere Schülerinnen und Schüler eine gute Differenzierungsaufgabe sein.

Obwohl das *BlueJ*-Projekt auf die Umsetzung des *k-Means-Algorithmus* nach Lloyd ausgelegt ist, ist prinzipiell auch eine Implementierung nach MacQueen möglich. In diesem Fall sind allerdings die Arbeitsblätter daran anzupassen.

Fazit

Der durch die Pilze gegebene Realitätsbezug macht unmittelbar klar, wie wichtig eine korrekte Einteilung von Daten sein kann. Dennoch werden bei dem vorgestellten Unterrichtskonzept schnell zwei Probleme des *k-Means-Algorithmus* deutlich: Durch Experimentieren mit den im *BlueJ*-Projekt mitgelieferten Beispieldatensätzen oder durch ungünstige Startzentren (vgl. Abbildung 5) wird schnell klar, dass der *k-Means Algorithmus* auch bei klar erkennbaren Clustern nicht immer zu den erwarteten Clustern führt. Insbesondere muss die gewünschte Anzahl der

Cluster dem *k-Means-Algorithmus* bereits übergeben werden, das optimale *k* wird einfach durch Betrachten der Daten in einem Koordinatensystem ermittelt. Der Nutzen des vorgestellten Verfahrens kann von den Schülerinnen und Schülern also fundiert kritisiert und hinterfragt werden. Insbesondere bietet sich eine Möglichkeit zur Diskussion über die Qualität von Daten, den Einsatz von Clusteringalgorithmen und die Vertrauenswürdigkeit von KI-Systemen, wie etwa einer Smartphone-App zur Pilzbestimmung. Dies kann Motivation sein, andere Algorithmen zur Klassifikation der Pilze zu betrachten. Weiter kann dieser Hinweis motivieren reale Anwendungsbeispiele für den *k-Means-Algorithmus* zu recherchieren. Der *k-Means-Algorithmus* ist in realen Anwendungen von großer Bedeutung. Wird er beispielsweise auf Produktionsdaten angewendet, kann er die Erkennung von Mustern und Anomalien ermöglichen, die zur Ineffizienz in der Produktion führen. Besonders, weil der *k-Means-Algorithmus* mit ungelabelten Daten arbeitet, ist er ein wichtiges Werkzeug Muster in derartigen Daten korrekt zu interpretieren.

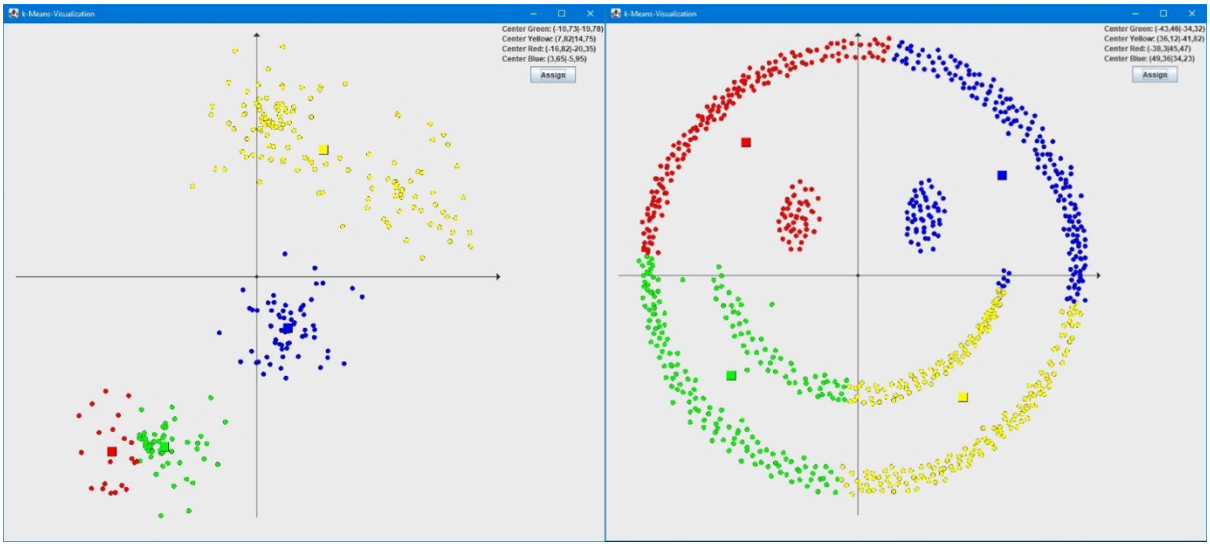


Abbildung 5 Ergebnis bei klar erkennbaren vier Clustern und ungünstiger Startzentrenverteilung sowie das Ergebnis des Clusterings des Beispieldatensatzes „smiley.csv“

Zusammengenommen können diese Probleme zu einer weiteren Unterrichtssequenz motivieren, in welcher evtl. durch einen Exkurs zu alternativen Clusteringverfahren wie dem *Hierarchischen Clustering* oder durch die Vorstellung von Validierungsverfahren für verschiedene Clusterungen die Ergebnisse des *k-Means-Algorithmus* in Abhängigkeit von *k* genauer betrachtet werden.

Quellen

Alle Webseiten/Links wurden zuletzt geprüft am 07.02.2024.

Ertel, Wolfgang (2021): Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung, Springer, Wiesbaden. [5. Auflage]

Heesen, Bernd (2023): Künstliche Intelligenz und Machine Learning mit R: Anwendungen im Bereich Business Analytics, Springer, Wiesbaden. [1. Auflage]

Illustrierende Prüfungsaufgaben für die schriftliche Abiturprüfung in Bayern ab 2026

- Informatik grundlegendes Anforderungsniveau (gA)

https://www.isb.bayern.de/fileadmin/user_upload/Gymnasium/IlluPA/Inf/IlluPA_Informatik_Beispielaufgaben_gA.pdf

Lloyd, Stuart P. (1982): Least squares quantization in PCM, IEEE Transactions on Information Theory, 28 (2): 129–137

MacQueen, J. (1967): Some Methods for classification and Analysis of Multivariate Observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. Band 1. University of California Press, S. 281–297

Staatsministerium für Schulqualität und Bildungsforschung München: LehrplanPLUS Gymnasium Bayern, Informatik 13 und spät beginnende Informatik 13 (grundlegendes Anforderungsniveau)

https://www.lehrplanplus.bayern.de/fachlehrplan/gymnasium/13/informatik/grundlegend

Staatsministerium für Schulqualität und Bildungsforschung München: LehrplanPLUS Gymnasium Bayern, Informatik 13 (erhöhtes Anforderungsniveau)

https://www.lehrplanplus.bayern.de/fachlehrplan/gymnasium/13/informatik/erhoeht

Materialien

Die Arbeitsblätter und *BlueJ*-Projekte zum *k-Means-Algorithmus* stehen zum Download auf der Homepage der Didaktik der Informatik der Julius-Maximilians-Universität Würzburg zur Verfügung. Sie können unter dem Direktlink https://go.uniwue.de/ki kostenlos heruntergeladen werden.

Lizenz

Kontakt

Daniela Andres
Dr. Silvia Joachim
Prof. Dr. Martin Hennecke

Didaktik der Informatik, Julius-Maximilians-Universität Würzburg, https://go.uniwue.de/ddi

E-Mail: {daniela.andres,silvia.joachim,martin.hennecke}@uni-wuerzburg.de

Reinforcement Learning Schritt für Schritt in Schulprojekten

Rau, T.
Graf-Rasso-Gymnasium Fürstenfeldbruck

DOI: 10.18420/ibis-02-01-07

Zusammenfassung

In diesem Artikel werden kleine Unterrichtsprojekte vorgestellt, anhand derer man sich dem Thema *Reinforcement Learning* – einem Teilbereich des *Machine Learning,* wie auch überwachtes und überwachtes Lernen – in mehreren Schritten in der Sekundarstufe II nähern kann. Jeder Schritt stößt am Ende auf Grenzen, so dass sich jeweils die nächste Erweiterung anbietet. Am Anfang steht das manuelle Anlegen einer einfachen handschriftlichen Tabelle über entscheidende Züge in einem Spiel. Weil das nur bei einfachen Spielen geht, ist der nächste Schritt das Anlegen und Anpassen einer Q-Tabelle mit kontinuierlicher Evaluation sämtlicher Züge. Im dritten Schritt wird die für manche Fälle nicht mehr ausreichende Q-Tabelle durch ein Neuronales Netz ersetzt.

Einleitung

Wenn man eine Strategie für das Lösen von einer Gruppe von Aufgaben entwickeln will, gibt es verschiedene Möglichkeiten: Man kann sich selber einen Algorithmus überlegen, aber das erfordert Analyse und geht nur bei überschaubaren Aufgaben; man kann einen bewährten Algorithmus wie den Minimax-Algorithmus verwenden, aber der erfordert ebenfalls Analyse und eine Bewertungsfunktion. Ein ganz anderer Ansatz ist das Lernen durch Bestätigung, *Reinforcement Learning:* Das ist eine Form des maschinellen Lernens, bei dem ein Agent zu Beginn noch nicht weiß, was eine geeignete Strategie ist, aber durch Rückmeldungen aus einem System nach und nach lernt, die eigenen Entscheidungen abhängig vom Zustand des Systems so zu treffen, dass schließlich das gewünschte Verhalten gezeigt wird – ohne eigene Analyse (Sutton & Barto 1999, S. 3).

1. Naives Reinforcement Learning: Notieren von spielentscheidenden Zügen

Man kann es bereits als eine einfache Form von Reinforcement Learning betrachten, wenn man sich lediglich bei einem Spiel merkt, welcher Zug in welchem Spielzustand zu einem unmittelbaren Sieg oder einer Niederlage geführt hat, und diesen Zug dann bei zukünftigen Wiederholungen im selben Zustand entsprechend bevorzugt oder vermeidet. (In den Beispielen geht es insgesamt meist um Spiele, weshalb von Spielzustand und Zug die Rede ist; allgemeiner würde

man wohl von Zuständen und Entscheidungen sprechen.) Das heißt, man probiert einfach mehr oder weniger durch *brute force* alles aus und merkt sich für jede Kombination von Spielzustand, wie oft er zu Sieg oder Niederlage geführt hat. Damit lassen sich bereits einfache Spiele erlernen.

Beispiel: Mau-Mau-Regeln erkennen

Mau-Mau ist ein einfaches Kartenspiel, bei dem als grundsätzliche Ablageregel gilt: Man darf auf eine Karte eine andere Karte legen, wenn sie den gleichen Wert oder die gleiche Farbe (im Sinn von Pik, Herz, Karo, Kreuz) hat. Theoretisch sind auch andere Regeln denkbar, im Spiel Eleusis von Robert Abbott geht es sogar darum, induktiv solche Regeln zu erschließen. Wenn man einfach ausprobiert und mitzählt, welche Karte anlegbar ist und welche nicht, errät man schnell die Regel.

Das lässt sich exemplarisch vorführen oder in Partnerarbeit durchspielen. Sinnvoll ist dabei das Anlegen einer Tabelle, um die Ergebnisse festzuhalten; sie wird später ein wichtiges Element beim Reinforcement Learning darstellen. Dort heißt sie Q-Tabelle oder *Q table*, weil die bewertete *Qualität* der Entscheidungen darin notiert ist.

Beispiel: Minischach

Dieses in der Informatikdidaktik viel verwendete Spiel ist eine Art Mini-Schach auf einem 3x3-Spielfeld, mit drei Bauern auf jeder Seite, oben die Figuren der menschlichen Spielerin, unten die Roboter (oder, in anderen Fassungen, Krokodile) des KI-Agenten.

Gezogen werden die Bauern wie im Schach (ohne Ausnahmezüge); gewonnen hat man, wenn man (1) die gegenüberliegende Seite mit einer Figur erreicht hat oder (2) alle gegnerischen Figuren geschlagen hat oder (3) der Gegner keinen Zug mehr machen kann,

Die menschliche Spielerin 1 beginnt immer mit einem Zug der Menschen. Der Agent ist immer Spieler 2 mit den Robotern oder Krokodilen. Bei optimaler Strategie gewinnt Spieler 2 immer. Das ist am Anfang aber noch nicht offensichtlich, etwa wenn sich Schülerinnen und Schüler enaktiv als KI-Agent versuchen, dem Schokolinsen-Algorithmus folgend. Dabei werden keine Strichlisten geführt, sondern farbige Schokolinsen stehen für die einzelnen Züge, ihre Anzahl entspricht der Qualität des Zugs – bei einem Sieg kommt eine hinzu, bei einer Niederlage eine weg.

Es gibt insgesamt 37 verschiedene Spielzustände, denen der Agent begegnen kann, wovon 18 spiegelsymmetrische Varianten anderer Zustände sind. Am Ende sollte der Agent also eine Tabelle mit 37 gefüllten Zeilen erstellt haben. Zugmöglichkeiten gibt es je nach Notation mehr, wobei nicht alle Züge in jedem Spielzustand legal möglich sind; je nach Zustand stehen maximal 4 zur Auswahl. Bald lernt der Agent, das Spiel fehlerfrei zu spielen, so dass er immer gegen einen Menschen gewinnt.

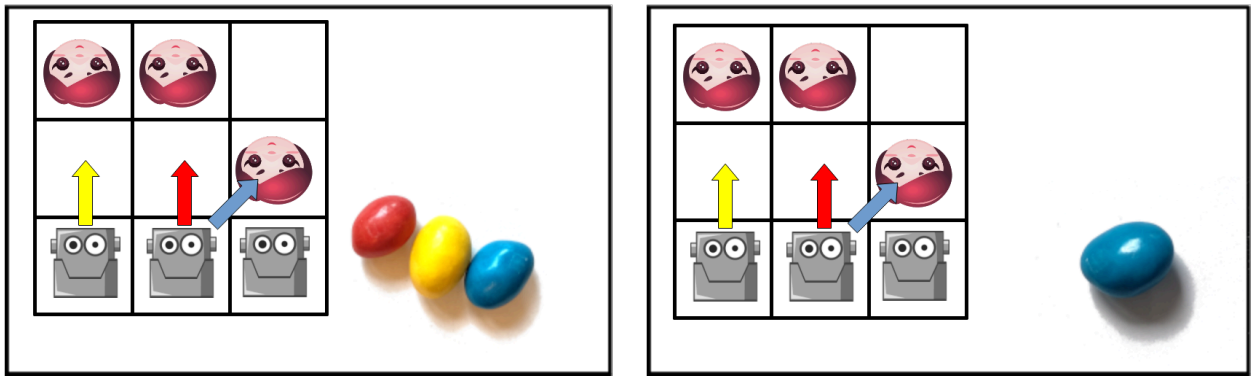


Abbildung 1: Ursprüngliche und spätere Bewertung der Züge: Alle drei Optionen sind am Anfang gleich bewertet, nach dem Lernen verbleibt nur eine.

Beispiel: Pong/Breakout

Selbst ein ganz simples Computer-Breakout oder Pong funktioniert. Ein Sieg ist dabei, den Ball mit dem Schläger zu erwischen, worauf der Ball abprallt; das wird mit 1 Punkt belohnt. Als Niederlage gilt, wenn der Ball die untere Linie berührt, was mit -1 Punkt bestraft wird.[1] Fürs Erste sei der Zustand des Systems die Differenz der x-Positionen von Ball und Schläger, eventuell durch 10 geteilt, um eine überschaubare Anzahl an Zuständen (je nach Spielfeldgröße um die 60) zu erhalten[2]. Die Zugmöglichkeiten sind: den Schläger 1. gar nicht, 2. nach links oder 3. nach rechts zu bewegen.

Es stimmt zwar, dass die KI im folgenden Zustand keinen Unterschied zwischen den Entscheidungen lernen kann, da alle Entscheidungen zu einer Niederlage, also dem Verpassen des Balls, führen:

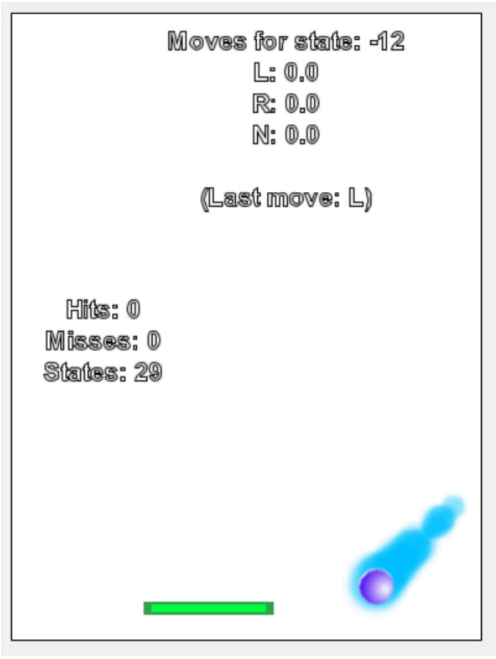


Abbildung 2: Spielsituation Breakout: Ball weit entfernt von Schläger

[1] Man kann stattdessen aber auch in allen anderen Situationen belohnen: je kleiner der Abstand zwischen Ball und Schläger, desto mehr Belohnung. So lernt der noch Agent schneller, aber er lernt vielleicht weniger, Breakout zu spielen, sondern er lernt eben, den Schläger parallel zum Ball zu führen – man hat das Problem für ihn vereinfacht.

[2] Bei den bisherigen Spielen war die Wahl des Zustands unkompliziert, nämlich meist die vollständige Information über das System. Bei Breakout gibt es mehrere Möglichkeiten: 1. Die Differenz der x-Positionen von Ball und Schläger. Allerdings nimmt diese Reduktion bereits einen Teil der Lösungsstrategie voraus: sie muss etwas mit den x-Koordinaten zu tun haben. Wenn man als Zustand stattdessen 2. ein Tupel wählt aus x- und y-Position des Balls, seiner Flugrichtung, und der x-Position des Schlägers, also die gesamten Systeminformationen, dann gibt man weniger vor. Die Anzahl der Zustände ist jetzt allerdings viel größer, das naive Belohnungssystem versagt dabei fast ganz.

Die Bewegungen Links, Rechts oder Nichts werden alle ausprobiert und alle bestraft. da alle gleichermaßen zu einer Niederlage führen. Dennoch kann das Spiel gelernt werden: Wenn der Ball in einer anderen Situation knapp am Schläger ist, so dass die Entscheidung einen Unterschied macht, wird die richtige belohnt. Zwar ist es so nur eine kleine Anzahl an Zuständen, bei denen sinnvolle Entscheidungen gelernt werden, aber das System bewegt sich am Ende in dieser simplen Version immer nur innerhalb dieser Menge an Zuständen.

Gegenbeispiel: Tic-Tac-Toe

Auch auf Spiele wie das Streichholz-Wegnehm-Spiel Nim oder Tic-Tac-Toe lässt sich diese einfache Form des Reinforcement Learning anwenden, allerdings nicht sehr gewinnbringend: Der so trainierte Agent spielt schlecht. Die Analyse, warum er schlecht spielt, führt zu den Grenzen des bisherigen Systems und weckt das Bedürfnis nach einer Erweiterung.

Die KI kann zwar lernen, nie einen Zug zu machen, der zu einer unmittelbaren Niederlage führt – aber nur, sofern sich das überhaupt vermeiden lässt. Doch anders als bei den bisher betrachteten Spielen gibt es bei Tic-Tac-Toe Situationen, in denen *jeder* mögliche Zug zu einer Niederlage führt. Eine KI, die den Namen verdient, sollte in der Lage sein, solche Situationen zu vermeiden, aber das ist mit dem bisherigen System nicht möglich.

Wenn die menschlichere Spielerin 1 links oben beginnt, gibt es acht mögliche Züge, die alle mit dem neutralen Initialwert bewertet sind. Keiner dieser Züge kann unmittelbar zu Sieg oder Niederlage führen, so dass keiner dieser Züge in der Bewertung eine Änderung erfahren kann. Tatsächlich führen aber sieben der acht Möglichkeiten gegen eine geübte Spielerin 1 immer zu einer Niederlage.

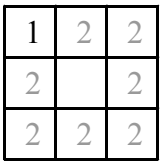


Abbildung 3: Alle eingezeichneten Züge für 2 führen in der Situation 100|000|000 mittelbar zu einer Niederlage.

(Auch bei dem Minischach gibt es einen einzigen solchen Zustand, in den man als Agent gar nicht erst kommen möchte, weil jeder mögliche Zug zu einer Niederlage führt. In der Schokolinsenschreibweise heißt das, dass keine einzige Schokolinse mehr zur Verfügung steht. Allerdings lernt die KI aus anderen Gründen, diese Situation gar nicht erst aufkommen zu lassen, da es bereits zuvor einen eindeutigen und damit präferierten Gewinn-Zug gibt.)

2. Eigentliches Reinforcement Learning: Kontinuierliche Bewertung

Die Lösung für das Problem: Man bewertet nicht nur den Zug, der Sieg oder Niederlage gebracht hat, sondern auch den vorhergehenden Zug, und damit also: *jeden* Zug. Der Agent kann dabei nicht in die Zukunft schauen, welcher Zug ihm später schaden oder nutzen wird, aber in die Vergangenheit. Deshalb wird in der Q-Tabelle im *aktuellen* Zustand bewertet, wie gut der letzte Zug im *vorhergehenden* Zustand war, abhängig von – wie bisher – Belohnung oder Bestrafung, aber eben zusätzlich abhängig von den Optionen, die jetzt im aktuellen Zustand zur Verfügung stehen. Gibt es da nur noch Züge, die schlecht bewertet sind, wird der Zug im vorhergehenden Zustand, der einen hierhergebracht hat, abgewertet. Gibt es da mindestens einen hoch bewerteten Zug, wird der Zug, der einen im vorhergehenden Zustand hierhergebracht hat, aufgewertet.

Ein ungünstiger Zustand und wie man dorthin kommt

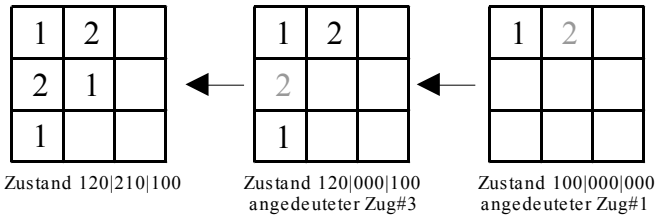


Abbildung 4: Drei mögliche Situationen für den Agenten/Spieler 2

Im Zustand 120|210|100 gibt es vier Zugmöglichkeiten für den Agenten 2, alle können zu einer unmittelbaren Niederlage führen, haben also einen niedrigen Wert in der Q-Tabelle. Dementsprechend müsste der Zug, der zu dieser Situation geführt hat, abgewertet werden, es handelt sich etwa um den Zug Feld #3 im Zustand 120|000|100. Allerdings sind die anderen fünf Züge in eben diesem Zustand ebenfalls schlecht bewertet, da sie unmittelbar zu einer Niederlage führen können. Demnach muss bereits der Zug, der zu diesem Zustand 120|000|100 führt, herabgesetzt werden, etwa Zug Feld #1 im Zustand 100|000|000.

So lernt die KI, diesen Anfangszug zu vermeiden, obwohl er erst später zu einer Niederlage führen wird.

Kontinuierliche Bewertung

Für diese kontinuierliche Bewertung des letzten Zugs im letzten Zustand unter Berücksichtigung von Belohnung, Bestrafung und den Optionen im aktuellen Zustand verwendet man die Bellman-Gleichung, hier in einer möglichen Version (nach Wang 2020):

$$Bewertung_{alt} = Bewertung_{alt} + Lernrate \cdot (\gamma \cdot Maximalwert_{neu} - Bewertung_{alt} + Belohnung)$$

Üblicherweise ist sie auch so formuliert:

$$Bewertung_{alt} = (1 - Lernrate) \cdot Bewertung_{alt} + Lernrate \cdot (Belohnung + \gamma \cdot Maximalwert_{neu})$$

Dabei ist $Belohnung_{alt}$ die Bewertung des bereits gemachten Zuges im vorherigen Zustand, $Maximalwert_{neu}$ die höchste Bewertung irgendeines Zuges im neuen, aktuellen Zustand. Diese Belohnung kann positiv oder negativ sein, in einem einfachen Fall beträgt sie zum Beispiel 1 bei einem Sieg und -1 bei einer Niederlage. Der Wert γ bezeichnet den Diskontierungsfaktor zwischen 0 und 1 (z.B. 0.5), der den zukünftigen Wert mehr oder weniger stark in das Ergebnis eingehen lässt, und die Lernrate zwischen 0 und 1 (z.B. 0.75) modifiziert die gesamte Werteänderung.

(Bei Lernrate 0 ändert sich nie etwas, ansonsten bestimmt die Lernrate, wie schnell die Bewertung im Falle keiner weiteren Änderungen konvergiert, und zwar zu $\gamma \cdot Maximalwert + Belohnung$. Bei $\gamma = 0$ konvergiert sie also allein zum Wert der aktuellen Belohnung, ohne dass die zukünftige Situation berücksichtigt wird.)

Wenn alle Züge in einem Zustand negative Werte haben, wird jeder Zug, der zu diesem Zustand führt, beim entsprechenden Durchgang im Wert herabgesetzt. Es spricht sich sozusagen herum, wenn ein Zustand eine Sackgasse darstellt.

Mit dieser Änderung lässt sich jetzt auch Tic-Tac-Toe erlernen.

Zustand	Zug	Bewertung
112 012 000	#3 #6 #7 #8	0.0 0.369140625 0.375 1.0
110 200 201	#2 #4 #5 #7	0.0 0.39825439453125 -0.75 -0.75
002 110 102	#0 #1 #5 #7	0.32967033384017147 -0.75 1.0 -0.75

Abbildung 5: Q-Tabelle für Tic-Tac-Toe, verkürzter Ausschnitt (~2100 Zustände, maximal 8 Optionen)

Exploratives Verhalten

Normalerweise wählt ein Agent während des Trainings als *greedy algorithm* immer den bestbewerteten Zug in einer gegebenen Situation. Allerdings ist es manchmal sinnvoll, auch andere Züge auszuprobieren. Deshalb ergänzt man oft eine Explorationsrate zwischen 0 und 1, etwa 0.05, die bestimmt, mit welcher Wahrscheinlichkeit ein zufälliger Zug statt des bestbewerteten ausgeführt wird.

3. Alltagsbeispiel selbstfahrende Autos: Weitere Probleme und Lösungen

Autonomes Fahren ist ein aktueller Kontext für Informatik und künstliche Intelligenz, auch in Computerspielen gibt es Fahrzeuge, die maschinell gesteuert werden. Deshalb bietet es sich an, zu schauen, wie weit man dabei mit Reinforcement Learning kommt. Die Aufgabe: Ein Auto soll einen Parcours entlang fahren können. Am Anfang soll das Fahren so weit wie möglich vereinfacht werden, später kann man immer noch weitere Elemente in das Modell einführen.

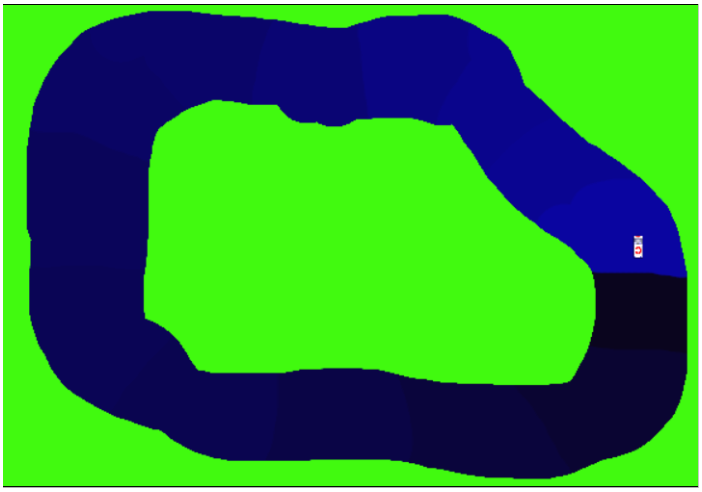


Abbildung 6: Eine einfache Fahrstrecke für ein Auto

- Es gibt nur ein Auto, keine weiteren, und keine Hindernisse. Das Auto wird als Punkt betrachtet.
- Das Auto kann jeweils nur links drehen und etwas fahren, rechts drehen und etwas fahren, oder geradeaus fahren. Es gibt keine Beschleunigung.
- Gefahren werden kann nur auf der Strecke, beim Betreten des grünen Bereichs außerhalb beginnt man wieder an der Startposition.

Umgesetzt wird das Projekt mit der didaktischen Entwicklungsumgebung Greenfoot.

Eine erste Lösung

Wieder muss man erstens entscheiden, was man als Zustand wählt. Eine naive, aber einfach umzusetzende Idee soll am Anfang stehen: x- und y-Koordinate des Autos und seine Drehrichtung. Alle drei bieten sich als typische Attribute eines Greenfoot-Actor-Objekts an. Es gibt dabei je nach Größe des Spielfelds als Obergrenze 640 * 400 * 8 Zustände, die aber nicht alle ausgeschöpft werden. Auf die Nachteile dieses Ansatzes wird später eingegangen.

Zweitens muss man entscheiden, welches Verhalten belohnt wird. Dabei reicht es, wenn das Verlassen der Bahn oder Fahren in die Gegenrichtung mit -1 bestraft und das Erreichen eines neuen Streckenabschnitts mit +1 belohnt wird. Die Streckenabschnitte sind farbig kodiert, erreicht das Auto einen dunkleren Abschnitt als den bisherigen, gilt das als Fortschritt.

Nach diesem System lernt der Agent relativ schnell, die Strecke abzufahren, im Bild sieht man die erste erfolgreiche Runde nach vielen Fehlversuchen (die jeweils zu einem Neustart am Anfang führen):

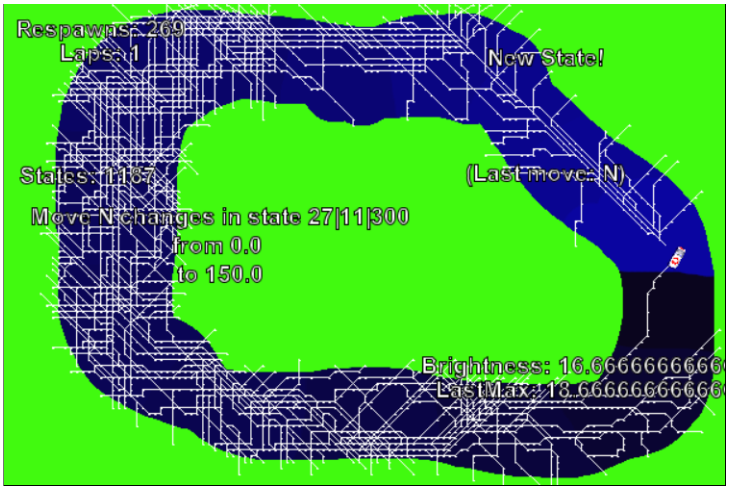


Abbildung 7: Erste vollständige Runde nach vielen erfolglosen Versuchen

Vielleicht sind Schülerinnen und Schüler bereits jetzt auf das Problem gestoßen, dass der Agent gelernt hat, genau eine Strecke abzufahren und bei anderen Strecken versagt. Als Test legt man einen zweiten Parcours an. Man kann zwar auch diesen trainieren, aber das Ergebnis lässt sich nicht übertragen. Es ist klar, dass das an der Wahl des Zustands liegt - die KI lernt einfach blind, an welcher absoluten Leinwandposition welches Verhalten gewünscht ist.

Eine zweite Lösung: Sensoren

Eine bessere Lösung ist, das Auto mit Sensoren auszurüsten, etwa drei oder fünf davon, die die Entfernung nach vorne und zu den Seiten messen. Abhängig von Länge und Anzahl der Sensoren gibt es als Obergrenze 25^5 mögliche Zustände, und auch mit diesem System lässt sich das Abfahren der Strecke lernen:

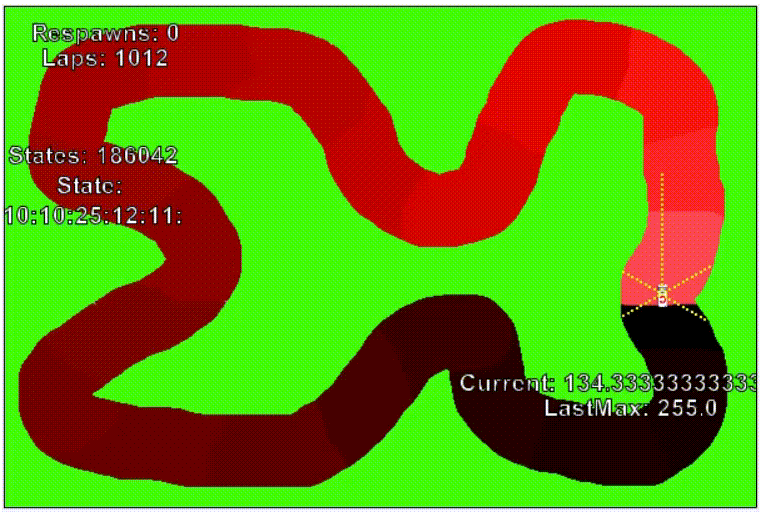


Abbildung 8: Auto mit fünf Sensoren auf schwierigerer Strecke

Allerdings ist die Übertragbarkeit auf andere Strecken immer noch gering. Bei einiger Überlegung wird die Ursache klar. Der Agent lernt zum Beispiel, sich in der Situation 05|25|20 (Abstände links, vorne, rechts) erfolgreich zu verhalten, und kann das auch übertragen auf genau diese Situation auf einer anderen Strecke. Aber der an sich ähnliche Zustand 05|25|19 stellt einen völlig neuen Eintrag in der Q-Tabelle dar, der neu gelernt werden muss und auf den sich die Ergebnisse ähnlicher, bereits bekannter Einträge nicht übertragen lassen.

Welche Lösungen gibt es für dieses Problem? Man könnte, wenn man das Lernen für abgeschlossen erklärt hat, bei unbekannten Zuständen nach dem jeweils ähnlichsten Zustand in der Q-Tabelle suchen. Oder man könnte so lange auf vielen verschiedenen Strecken trainieren, bis der Zustandsraum völlig erfasst und in die Q-Tabelle eingegangen ist. Oder man verwendet statt der Q-Tabelle ein Neuronales Netz.

Eine dritte Lösung: Neuronales Netz

Wenn die Kombinationsmöglichkeiten für eine Q-Tabelle zu umfangreich werden, hilft stattdessen ein Neuronales Netz, Ähnlichkeiten erkennen und zusammenfassen. Es besteht aus einzelnen vernetzten Neuronen, wobei die Eingabeknoten Werte annehmen und sie in der einfachsten Form modifiziert an eine versteckte Schicht von Knoten weitergeben, die ihrerseits aus den verschiedenen Inputs neue Outputs generieren und an eine Ausgangsschicht von Knoten weitergeben. Die Art der Modifikation wird dabei nach und nach lernend verbessert, bis ein Ziel erreicht ist. Statt der Q-Tabelle wird jetzt ein solches Netz verwendet.

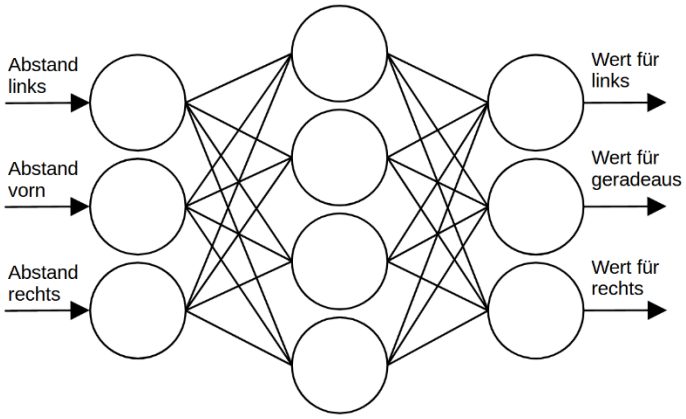


Abbildung 9: Neuronales Netz mit 4 versteckten Knoten, für ein Auto mit 3 Sensoren und 3 Fahrtmöglichkeiten

- Anzahl der Eingabeknoten: entspricht Anzahl der Sensoren beim Auto.
- Anzahl der Ausgabeknoten: entspricht Anzahl der möglichen Entscheidungen, im einfachen Fall drei: Geradeaus, Links, Rechts.
- Anzahl der versteckten Knoten, Lernrate, insbesondere Wahl der Initialbelegung und der Aktivierungsfunktion: hängt von der konkreten Situation ab.

Die Eingangswerte entsprechen den von den Sensoren gemessenen Werten. Die Ausgabewerte entsprechen dabei der Bewertung der verschiedenen Entscheidungen: Der Ausgabeknoten mit

dem höchsten Wert ist die Entscheidung, die am besten bewertet ist. Die Ausgabeknoten stehen für die Zugmöglichkeiten.

Das Trainieren des Netzes läuft ebenfalls über die oben genannte Bellman-Gleichung. Dabei ist $Belohnung_{alt}$ wieder die Bewertung des letzten Zuges im vorherigen Zustand. Sie ist zu ermitteln, indem man den vorherigen Zustand als Eingabe für das Neuronale Netz verwendet und die Bewertung aus dem Ausgabeknoten abliest, der dem Zug entspricht. Zum Beispiel könnten die Eingabewerte 05|25|20 sein, die Ausgabewerte 4|3|-2, der Zug selber war Zug 0 und dessen neue Bewertung (also die Ausgabe des ersten Knotens der Ausgabeschicht) 7.

Dann wird das Netz so trainiert, dass für die Eingabewerte des alten Zustands die Knoten der Ausgabeschicht nicht mehr das ursprüngliche Tupel 4|3|-2 ergeben, sondern jetzt zumindest näherungsweise 7|3|-2, also mit dem neu ermittelten Wert.

Das Ergebnis: Jetzt lässt sich das auf einem Kurs gelernte auch auf andere Kurse übertragen. Klar ist allerdings: Wenn ein Kurs nur Linkskurven enthält, wird das Auto auf einem Kurs, der auch Rechtskurven enthält, wenig erfolgreich sein –eine Hypothese, die man überprüfen kann.

Bereits mit drei Sensoren und vier Neuronen in der versteckten Schicht lässt sich ein Verhalten erlernen, das zum Abfahren zweier verschiedener Strecken ausreicht.

Warum überhaupt Reinforcement Learning?

Tatsächlich könnte man die Steuerung ganz ohne KI anlegen. Schließlich lässt sich für das bisherige System auch manuell ein Algorithmus finden und umsetzen: Wenn der Sensor vorne am größten ist, fahre geradeaus, sonst wenn er links am größten ist, mache die Linksbewegung, sonst die Rechtsbewegung. Die Vorteile, die Reinforcement Learning mit oder ohne Neuronalem Netz bieten, zeigen sich erst dann, wenn ein solcher Algorithmus für das optimale Verhalten nicht mehr so einfach manuell angelegt werden kann. Deshalb kann man Erweiterungsmöglichkeiten ausprobieren: Andere Autos, zusätzliche Hindernisse, Beschleunigung und Bremsen.

Quellen

Alle Webseiten/Links wurden zuletzt geprüft am 01.05.2023.

Michaeli, Tilman & Seeger, Stefan (2023): Schlag den Roboter – Maschinelles Lernen erfahren. https://computingeducation.de/proj-schlag-das-kroko/

Rau, Thomas (2023): KI: Reinforcement Learning 2. https://www.herr-rau.de/wordpress/2023/03/ki-reinforcement-learning-2.htm

Sutton, Richard & Barto, Andrew (1999): Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.

Wang, Mike (2020): Deep Q-Learning Tutorial: minDQN. A Practical Guide to Deep Q-Networks. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

Lizenz

Lizenz

Kontakt

Thomas Rau

Graf-Rasso-Gymnasium Fürstenfeldbruck

E-Mail: thomas.rau@ifi.lmu.de

Programmieren lernen mit dem genetischen Ansatz

Braune, G; Mühling, A.
Leibniz Institut für Pädagogik der Naturwissenschaften und Mathematik & Universität Kiel
DOI: 10.18420/ibis-02-01-08

Zusammenfassung

Durch die Ausweitung des Informatikunterrichts und insbesondere durch die Einführung von verpflichtendem Unterricht in immer mehr Bundesländern steigt der Bedarf an konkretem Material zur Umsetzung der Fachanforderungen und Curricula. Dieser Beitrag beschreibt ein Konzept für den Anfangsunterricht im Programmieren sowie das auf der Grundlage des Konzeptes erstellte Unterrichtsmaterial. Das Material zielt besonders auf das Pflichtfach Informatik ab, ist aber auch für Wahlkurse und Wahlpflichtkurse geeignet. Es betont besonders das *informatische Denken*, die *Themenorientierung* und die *prozessbezogenen Kompetenzen*. Als didaktische Leitlinie wird ein *genetischer Ansatz* verwendet. Das Unterrichtsmaterial besteht aus einer Folge von Arbeitsbögen und weiteren medialen Hilfsmitteln, die den gesamten Unterrichtsverlauf eines halben Jahres abdecken. Es steht online zur Verfügung.

Einleitung

Die Einführung eines Pflichtfachs Informatik an deutschen Schulen schreitet langsam, aber stetig voran (Meisner 2023). Dadurch steigt auch der Bedarf an Konzepten und Materialien, mit denen die Lehrkräfte in die Lage versetzt werden, den Fachanforderungen und Curricula entsprechend zu unterrichten.

In Schleswig-Holstein finden in den Schuljahren 2022-2024 landesweit Pilotprojekte statt, die die Einführung des Pflichtfaches Informatik vorbereiten sollen (IQSH - Institut für Qualitätsentwicklung an Schulen Schleswig-Holstein 2022). Der vorliegende Beitrag stellt ein Konzept mit zugehörigem Unterrichtsmaterial vor, das auf der Grundlage der Fachanforderungen Informatik (Land Schleswig-Holstein 2021) für eine der Pilotschulen entwickelt wurde. Alle Schülerinnen und Schüler des siebten Jahrgangs eines Gymnasiums erlernten über ein halbes Schuljahr hinweg nach Maßgabe des Konzeptes grundlegende Kompetenzen im Bereich des Programmierens. Dabei wurde besonderer Wert auf die in den Fachanforderungen besonders betonten Bereiche *Informatisches Denken*, *Themenorientierung* und *Prozessbezogene Kompetenzen* gelegt. Der Unterricht verbindet etablierte didaktische Theorien mit modernen Methoden und folgt dem *genetischen Ansatz*. Aufgrund ähnlicher curricularer Anforderungen im Bereich des Programmierens ist das Material ganz oder teilweise auch in verschiedenen Schularten anderer Bundesländer und in unterschiedlichen Anwendungskontexten, zum Beispiel in Wahl- und Wahlpflichtkursen, einsetzbar.

Leitgedanken des Unterrichtskonzeptes

Genetischer Ansatz

Die Grundidee des genetischen Ansatzes ist es, die Schülerinnen und Schüler an der Entwicklung fachlicher Konzepte teilhaben zu lassen, statt ihnen die Konzepte als feststehendes Wissen lediglich zu präsentieren. Für den Bereich der Informatik formuliert Schuster (2011) als „Arbeitsdefiniton“: „Genetischer Informatikunterricht fasst informatische Begriffe als ‚gewordene‘ auf und will ihr mögliches ‚Werden‘ im Lernprozess nachvollziehen lassen“. Tragendes Element des genetischen Unterrichts ist immer eine konkrete Fragestellung, die einen „sinnstiftenden Kontext“ schafft (Leuders et al. 2011). Dadurch wird echtes Verständnis gefördert und das isolierte Anhäufen von Wissen vermieden.

Das „genetische Prinzip“ als didaktischer Begriff ist facettenreich und daher hier nur kurz skizziert. Eine detaillierte Beschreibung und Analyse der verschiedenen Perspektiven findet sich z.B. bei Möller (2001).

Informatisches Denken (Computational Thinking)

Die bekannte Charakterisierung informatischen Denkens von Jeanette Wing (2012) lautet: “Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent”. Mithilfe informatischen Denkens werden also Probleme formuliert und gelöst - die Lösung ist aber ein Prozess. Dieser Prozess wird von einem informationsverarbeitenden „Agenten“ ausgeführt. Das kann ein Computer, aber auch ein nicht-kreativ handelnder Mensch sein, der Anweisungen mechanisch ausführt. Die Erkenntnis, dass Computer Probleme nicht lösen, sondern von Menschen erdachte Lösungen rezeptartig ausführen, ist von grundlegender Bedeutung für das Verständnis des Programmierens und auch in Zeiten der künstlichen Intelligenz weiterhin gültig. Sie wird in der vorliegenden Unterrichtseinheit an mehreren Stellen aufgegriffen.

Themenorientierung bzw. Kontextualisierung

Der genetische Ansatz führt fast automatisch dazu, dass man für den Unterricht eine von den Schülerinnen und Schülern als sinnvoll wahrgenommene thematische Fragestellung aufgreift und aus dieser Fragestellung heraus allgemeine (informatische) Konzepte entwickelt. In diesem Sinn ist der Unterricht themenorientiert, wie von den Fachanforderungen in Schleswig-Holstein gefordert – genauso aber auch aus der Perspektive der Phänomen- (Humbert & Puhlmann 2004) oder Kontextorientierung (Koubek et al. 2009) zu verstehen.

Prozessbezogene Kompetenzen

Franz E. Weinert (2001) definiert Kompetenz als „die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen [...]“.

Eine Kompetenz zu haben, bedeutet also nicht nur, etwas zu *wissen*, sondern auch, auf der Grundlage des Wissens *handeln* zu können. Es liegt daher nahe, bei Kompetenzbeschreibungen zwischen *inhaltsbezogenen* und *prozessbezogenen* Kompetenzen zu unterscheiden.

Die Fachanforderungen in Schleswig-Holstein (Land Schleswig-Holstein 2021) betonen dabei informatische Entwicklungsprozesse und folgen erkennbar den amerikanischen K12-Rahmenrichtlinien (K-12 Computer Science Framework Steering Committee 2016). Abbildung 1 stellt die für Schleswig-Holstein gültigen prozessbezogenen Kompetenzen dar. Der äußere Ring beschreibt das typische iterative Vorgehen der praktischen Informatik: Ein Artefakt erstellen, indem man ein Problem beschreibt, analysiert, löst und dann die Lösung implementiert, evaluiert und ggf. verbessert (Buttke et al. 2011, Best et al. 2019). Dies wird in der vorliegenden Unterrichtseinheit als eine direkte Leitlinie zur Gestaltung des Unterrichts gesehen. Während man das Problem schrittweise löst, ergibt sich die Notwendigkeit, die Inhalte und Vorgänge zu benennen, also durch Abstraktion Konzepte zu gewinnen. Dies wiederum spiegelt die Intention des genetischen Ansatzes wider. Grundsätzlich passt das gewählte Vorgehen auch zu den eher generisch formulierten Prozessbereichen der GI-Bildungsstandards (GI 2008).

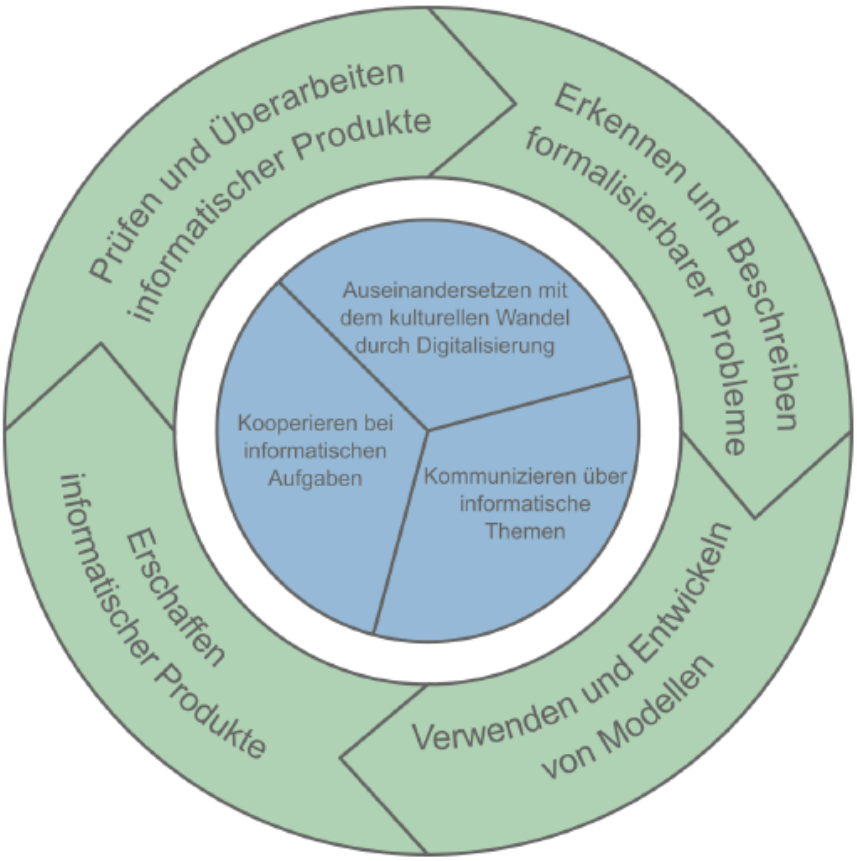


Abbildung 1: Prozessbezogene Kompetenzen in den schleswig-holsteinischen Fachanforderungen (Land Schleswig-Holstein 2021).

Methodische Aspekte

Programmierumgebung

In der vorliegenden Unterrichtseinheit wird als Programmierwerkzeug *Scratch* verwendet. Scratch ist (nicht nur) an schleswig-holsteinischen Schulen sehr weit verbreitet (Braune und Mühling 2020), was sicherlich mit der niedrigen Einstiegsschwelle und der Vermeidung von Syntaxproblemen zusammenhängt (Resnick et al. 2009, Grover 2020). Die schleswig-holsteinischen Fachanforderungen empfehlen auch explizit den Beginn mit einer blockbasierten Programmierumgebung.

In der hier präsentierten Unterrichtseinheit soll Scratch nicht zum Erstellen von interaktiven Geschichten oder Spielen verwendet werden, sondern als Werkzeug zum zielgerichteten, planvollen Lösen von Problemen. Die Unterrichtseinheit ist grundsätzlich auch auf andere (blockbasierte) Programmierumgebungen, etwa Snap!, portierbar, da sie nicht speziell auf Scratch-spezifische Merkmale angewiesen ist.

Unterrichtsmethoden

Zur Realisierung des genetischen Ansatzes werden speziell die folgenden typischen Methoden des Informatikunterrichts benutzt:

- *Computer Science Unplugged*[1], also die Idee, dass sich die eigentliche Problemlösung in den Köpfen der Menschen abspielt und dass es daher das Verständnis fördert, Vorgehensweisen und Lösungskonzepte zunächst abseits des Computers zu entwickeln (Curzon 2013, Bell und Vahrenhold 2018).
- *Cognitive Apprenticeship*, um die Unterstützung durch die Lehrkraft nach und nach zurückzufahren („faded scaffolding“) und dabei typische Praktiken des Fachs Informatik zu modellieren (Lave und Wenger 2011).
- *PRIMM*[2] *(Predict-Run-Investigate-Modify-Make)*: Die aus der Einsicht, dass Lernende nicht von Anfang an selbständig programmieren können, folgende Idee, zunächst gegebene funktionsfähige Programme analysieren und modifizieren zu lassen (Sentance et al. 2019).

Es wird davon ausgegangen, dass es nicht *die eine* Methode gibt, die auf alle Situationen erfolgversprechend anwendbar ist. Stattdessen kommen je nach Erfordernis unterschiedliche Vorgehensweisen zum Tragen.

Aufbau der Unterrichtseinheit

In Anwendung des Ansatzes *Cognitive Apprenticeship* besteht die Unterrichtseinheit aus einem eher lehrkraftzentrierten *Einführungsteil* und einem eher durch selbständige Arbeit der

[1] https://www.csunplugged.org/de/
[2] https://primmportal.com/

Schülerinnen und Schüler charakterisierten *Projektteil*. Im Pilotprojekt dauerte der Einführungsteil rund neun Wochen und der Projektteil rund 10 Wochen (bei einer Doppelstunde Unterricht pro Woche) – grundsätzlich lassen sich die jeweiligen Teile aber in ihrer Dauer auch etwas anpassen.

Unterrichtsmaterial

Überblick

Das Unterrichtsmaterial ist online zugänglich[3] . Es besteht aus drei Teilen:

- einer Einführung in das Unterrichtskonzept und die Nutzung des Materials,
- einer Folge von Arbeitsbögen, denen weitere Unterlagen (z.B. Programme, Präsentationen, Videos) zugeordnet sind und die den gesamten Unterrichtsverlauf des Pilotprojektes widerspiegelt, und
- einigen Ergänzungen und Erweiterungen, darunter Anregungen für weitere Themenorientierungen und noch nicht erprobtes Material für ältere Schülerinnen und Schüler (Ende der Mittelstufe und Oberstufe).

Jeder Haupt-Abschnitt wird durch Informationen für Lehrkräfte eingeleitet. Sie enthalten Empfehlungen für den Unterricht, die auf Erfahrungen aus dem Pilotprojekt basieren.

Die beiden folgenden Abschnitte vermitteln einen Eindruck von dem Material, indem sie beispielhaft zwei darin enthaltene Fragestellungen vorstellen.

Beispiel 1: „Wie speichert das Handy Bilder?“

Diese Fragestellung stammt aus dem eher lehrkraftzentrierten Einführungsteil. Der Unterricht vermittelt hier inhaltsbezogene Kompetenzen aus den Bereichen *Daten und Informationen* und *Algorithmen und Programmierung* (vgl. (Land Schleswig-Holstein 2021) und folgt den in den prozessbezogenen Kompetenzen beschriebenen Schritten (ebd.) der Problemlösung. Die Schülerinnen und Schüler haben im vorangehenden Unterricht erste Erfahrungen mit grundlegenden Kontrollstrukturen gemacht, in Bezug auf das Thema aber keine Vorkenntnisse. Dem genetischen Ansatz entsprechend beginnt der Unterricht mit einer konkreten, am Beispiel orientierten Arbeit an der Fragestellung und entwickelt aus dieser konkreten Arbeit heraus allgemeine Konzepte, die vor dem Hintergrund der Fragestellung „Sinn machen“ (Leuders et al. 2011), zum Beispiel „Rastergrafik“ und „Algorithmus“.

Um die Beschreibung und Lösung des Problems klar von der Implementation der Lösung zu trennen, wird die Fragestellung zunächst ohne Computer untersucht (*Computer Science Unplugged*, siehe oben). Schülergruppen, die in verschiedenen Räumen sitzen, sollen einfache Zeichnungen anfertigen und sich gegenseitig die Zeichnungen „per E-Mail“, also irgendwie

[3] https://oer.ipn.uni-kiel.de/edu-sharing/components/collections?id=3d505196-248c-4404-9e5a-1029f141aa33

zeichenweise, übermitteln, wobei die Lehrkraft den Transport der „Mails“ übernimmt. (Abbildung 2 zeigt einen Ausschnitt aus dem zugehörigen Arbeitsbogen). Anschließend vergleichen die Gruppen ihre Vorschläge und Ergebnisse. Die Erfahrung zeigt, dass neben vielen nicht brauchbaren Ansätzen (über die man dann gut diskutieren kann) oft Ideen geäußert werden, die im Kern mit Raster- oder Vektorgrafiken zu tun haben. Es erfolgt nun eine Beschränkung auf Rastergrafiken. In den folgenden Stunden geht es – den inhaltsbezogenen Kompetenzbereichen entsprechend – um zwei Fragen: Wie lassen sich Bilder als Rastergrafiken repräsentieren? Und: Wie kann man die Bilder in den Computer bzw. das Handy „bringen“ und wieder herausholen? Die erste Frage berührt die verwendete *Datenstruktur*, die zweite diejenigen *Algorithmen*, die auf der Datenstruktur arbeiten.

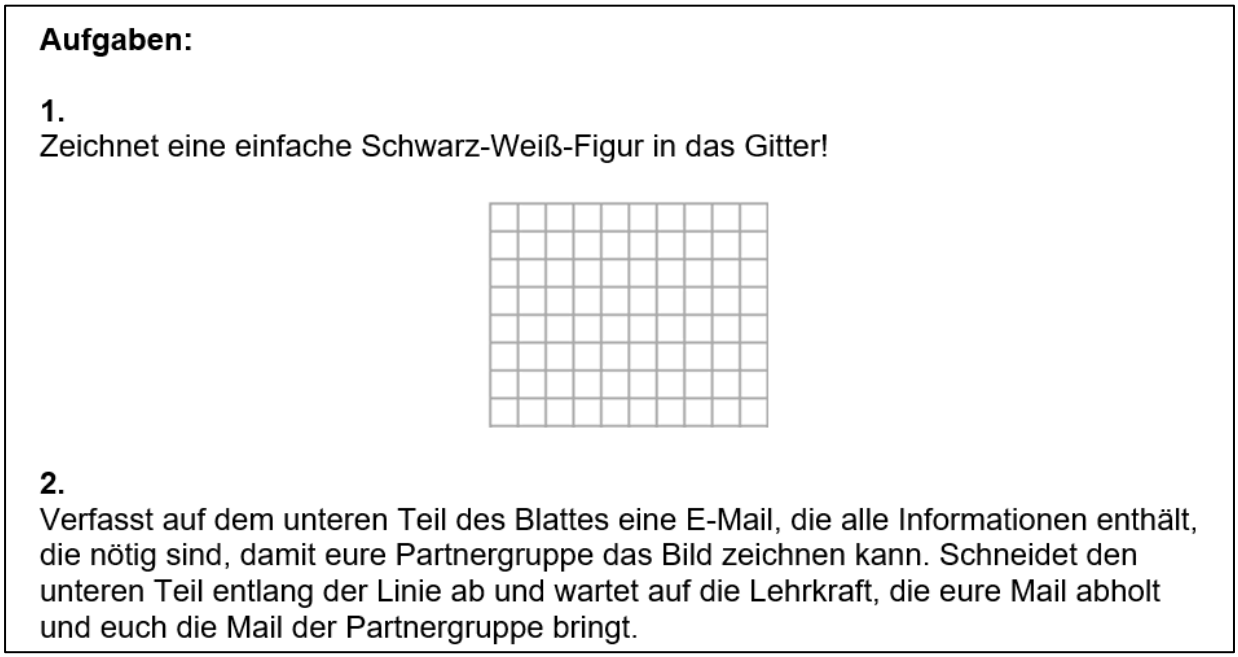


Abbildung 2: Bilder "unplugged" übermitteln

Dem genetischen Ansatz folgend werden diese fachlichen Konzepte jedoch erst eingeführt, wenn die Lösung des gegebenen Problems es erforderlich macht. In Bezug auf die zweite Frage bedeutet dies für den Unterricht Folgendes: Die Beschäftigung mit dem Scannen und Zeichnen von Bildern abseits des Computers („unplugged“) führt zu der Erkenntnis, dass zum Beispiel zum Einscannen eines Bildes in eine Rastergrafik eine bestimmte Folge von Anweisungen durchlaufen werden muss. Dabei wird diese Folge von Anweisungen „stur“ und rezeptartig abgearbeitet, das heißt, der oder die Ausführende muss die Anweisungen nicht inhaltlich verstehen, sondern sie lediglich abarbeiten. Um über eine solche Folge von Anweisungen, die schrittweise ausgeführt wird, gut sprechen zu können, wird ihr jetzt ein Name gegeben: *Algorithmus*. Dies ist natürlich nur eine erste Annäherung an das Konzept, eine genaue Definition kann erst im Laufe der Zeit entstehen, wenn wiederum bestimmte Fragestellungen dies nahelegen. (Zum Begriff des Algorithmus siehe z.B. Hubwieser et al. 2013.)

Die Schülerinnen und Schüler werden überwiegend nicht in der Lage sein, sich einen geeigneten Algorithmus selbst auszudenken und zu formulieren. Sie können aber gegebene Algorithmen gedanklich „nachspielen“ und dadurch *verstehen*, wie Algorithmen arbeiten. Abbildung 3 zeigt beispielhaft eine Aufgabe, die solche Aktivitäten verlangt.

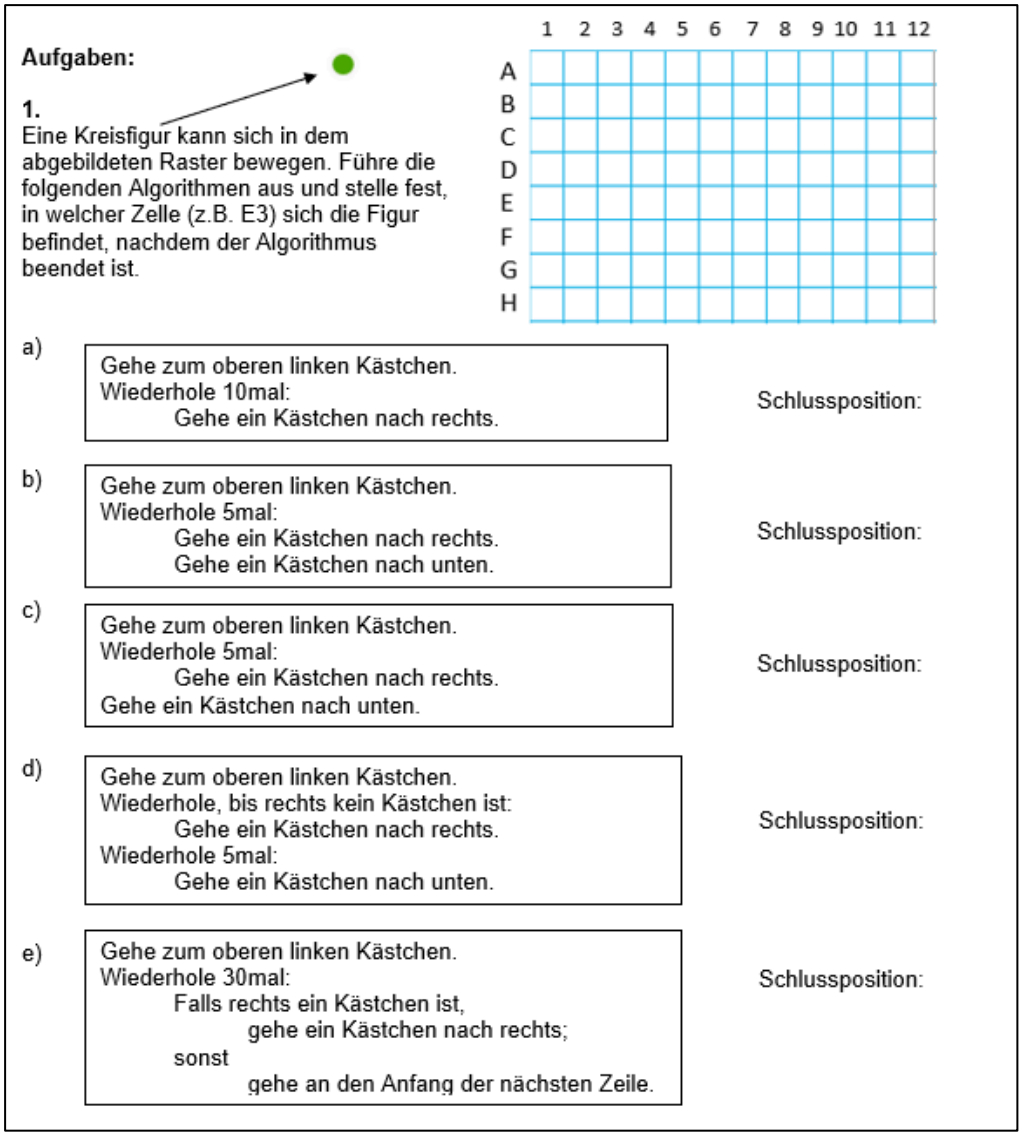
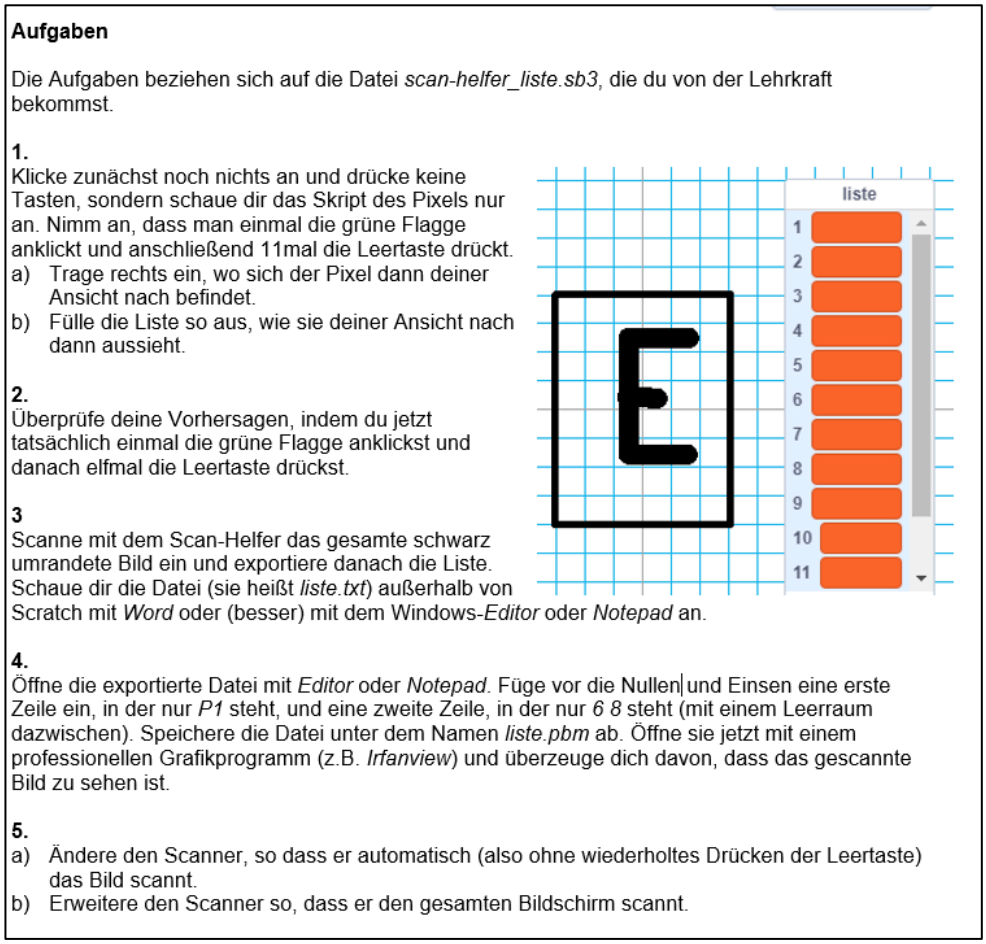


Abbildung 3: Algorithmen lesen und gedanklich abarbeiten

Nachdem das Problem *gedanklich* gelöst ist, geht es darum, die Rolle des oder der Ausführenden dem Computer zuzuweisen. Das bedeutet, dass der Algorithmus in eine Sprache übertragen werden muss, deren Anweisungen der Computer ausführen kann, mit anderen Worten: Er wird *implementiert*. Eine solche Implementierung können die Schülerinnen und Schüler zu diesem Zeitpunkt noch nicht selbständig leisten, sie können aber gegebene (Teil-)Lösungen analysieren, nachvollziehen, ändern und erweitern. Für diesen Zweck bietet der PRIMM-Ansatz (siehe oben) einen effektiv einsetzbaren methodischen Rahmen: Die Schülerinnen und Schüler bekommen funktionsfähige Programme, die Teile des gegebenen Problems lösen, aber noch

keineswegs „perfekt“ sind. Sie lesen die Programme und sagen die vermutete Wirkung vorher („Predict“). Dann lassen sie die Programme ausführen („Run“) und vergleichen die Vorhersagen mit der Realität. Anschließend untersuchen sie die Programme genauer auf bestimmte Aspekte hin („Investigate“) und modifizieren sie in Teilen, um sie zu verbessern („Modify“). Schließlich implementieren sie gewisse Programmteile selbständig („Make“). Die Abbildung 4 zeigt beispielhaft einen nach der PRIMM-Methode erstellten Arbeitsbogen.



Aufgaben

Die Aufgaben beziehen sich auf die Datei *scan-helfer_liste.sb3*, die du von der Lehrkraft bekommst.

1.
Klicke zunächst noch nichts an und drücke keine Tasten, sondern schaue dir das Skript des Pixels nur an. Nimm an, dass man einmal die grüne Flagge anklickt und anschließend 11mal die Leertaste drückt.
a) Trage rechts ein, wo sich der Pixel dann deiner Ansicht nach befindet.
b) Fülle die Liste so aus, wie sie deiner Ansicht nach dann aussieht.

2.
Überprüfe deine Vorhersagen, indem du jetzt tatsächlich einmal die grüne Flagge anklickst und danach elfmal die Leertaste drückst.

3
Scanne mit dem Scan-Helfer das gesamte schwarz umrandete Bild ein und exportiere danach die Liste. Schaue dir die Datei (sie heißt *liste.txt*) außerhalb von Scratch mit *Word* oder (besser) mit dem Windows-*Editor* oder *Notepad* an.

4.
Öffne die exportierte Datei mit *Editor* oder *Notepad*. Füge vor die Nullen und Einsen eine erste Zeile ein, in der nur *P1* steht, und eine zweite Zeile, in der nur *6 8* steht (mit einem Leerraum dazwischen). Speichere die Datei unter dem Namen *liste.pbm* ab. Öffne sie jetzt mit einem professionellen Grafikprogramm (z.B. *Irfanview*) und überzeuge dich davon, dass das gescannte Bild zu sehen ist.

5.
a) Ändere den Scanner, so dass er automatisch (also ohne wiederholtes Drücken der Leertaste) das Bild scannt.
b) Erweitere den Scanner so, dass er den gesamten Bildschirm scannt.

Abbildung 4: Mit der PRIMM-Methode erstellter Arbeitsbogen

Am Ende des Unterrichtsabschnitts verfügen die Schülerinnen und Schüler über funktionsfähige Scan- und Zeichenprogramme, die Rastergrafiken erzeugen und auswerten können. Die Rastergrafiken werden zudem in einem Format erstellt, das auch professionelle Grafikprogramme erkennen. Es handelt sich also um „richtige“ Grafiken, die Schülerinnen und Schüler verlassen damit die ansonsten künstlich abgegrenzte Welt von Scratch.

Beispiel 2: „Wie verläuft eine Epidemie?“

Dieses Beispiel stammt aus dem Projektteil der Unterrichtseinheit. Die mit dem genetischen Ansatz gewonnenen Konzepte werden jetzt in anderem Kontext *angewendet*. Die Schülerinnen und Schüler bearbeiten im Sinne des *Cognitive Apprenticeship*-Ansatzes (siehe oben) nun

weitgehend selbständig ein Projekt, erhalten jedoch bei Bedarf eine dem Leistungsvermögen angepasste Unterstützung. Es kann zwischen sechs Projektthemen gewählt werden, die alle eine außerschulische Fragestellung beinhalten. Abbildung 5 zeigt die Projektaufgabe zum Thema „Epidemien“.

Aufgaben zum Thema 1: Epidemien

Poster

Aufgabe:
Erstelle ein Poster zu *einem* der folgenden Themen:

- Verlauf der Covid-19-Epidemie (z.B.: Wo kam das Virus her? Wie hat es sich verbreitet? Welche Schutzmaßnahmen gab es? Wie ist die Situation heute?)
- Ausbreitung der Pest im Mittelalter (z.B.: Wie wird Pest verursacht und übertragen? Welche Pestwellen gab es in Europa? Wie wurde die Pest besiegt?)
- Viren (z.B.: Was sind Viren? Wie verbreiten sie sich? Wie kann man die Verbreitung verhindern?)
- Mein Leben während der Covid-Epidemie (z.B.: Persönliche Erfahrungen, Einschränkungen, Umgang mit Schutzmaßnahmen)
- Selbstgewähltes Thema aus dem Bereich „Epidemien“ (mit der Lehrkraft absprechen)

Programme

Den Verlauf einer Epidemie kann man mit dem Computer *simulieren*, das heißt, in vereinfachter Form nachbilden. Die Abbildung deutet eine solche Simulation an: Objekte bewegen sich in einem begrenzten Raum. Grüne Objekte stellen gesunde, rote Objekte kranke Menschen dar. Trifft ein grünes Objekt ein rotes, wird es selbst rot

Grundaufgabe:
Entwickle mit Scratch eine solche Simulation. Dabei soll gelten:

- „Kranke“ Objekte bleiben krank, es gibt keine Heilung.
- Die Geschwindigkeit der Objekte soll einstellbar sein.

Erweiterungsaufgaben:
Erweitere dein Programm, zum Beispiel so:

- Die Anzahl der Objekte in dem Raum soll einstellbar sein.
- Kranke werden nach einer bestimmten Zeit wieder gesund.
- Geheilte sind für eine bestimmte Zeit immun und können erst nach Ablauf dieser Zeit wieder angesteckt werden.
- Denk dir selbst eine Erweiterung aus und besprich sie mit der Lehrkraft.

Abbildung 5: Projektaufgabe zum Thema "Epidemien"

Für jedes Projekt fertigen die Schülerinnen und Schüler zunächst *Poster* an, mit denen der Bezug zur außerschulischen Wirklichkeit hergestellt wird. Die erstellten Poster können am Ende des Projektes zum Beispiel im Rahmen eines *Museumsrundgangs* (Klippert 2023) präsentiert werden.

Danach wird eine *Programmieraufgabe* gestellt, und zwar in Form einer Grundaufgabe mit Erweiterungsmöglichkeiten, die je nach Interessen und Fähigkeiten in Angriff genommen werden können. Im Falle der Epidemie-Themas geht es darum, den Verlauf einer Epidemie zu *simulieren*. Die Schülerinnen und Schüler können selbständig arbeiten oder aber in zwei Stufen *Hilfezettel* verwenden, die sie bei der Lösung des Problems unterstützen. Durch diese Differenzierung soll sowohl Über- als auch Unterforderung vermieden werden. Die erstellten Programme können zum Beispiel auch in die abschließende Ausstellung eingehen oder im Rahmen einer Präsentation gezeigt werden.

Ein Angebot zum Ausprobieren

Wie eingangs erwähnt, orientiert sich der Unterrichtsvorschlag zwar an Vorgaben aus Schleswig-Holstein, fügt sich jedoch in die Lehrpläne anderer Bundesländer zwanglos ein, da die blockbasierte Programmierung und das informatische Problemlösen ein typischer Bestandteil von modernem Informatikunterricht sind.

Wir hoffen, dass dieser Beitrag dazu anregen kann, das Material teilweise oder im Ganzen auszuprobieren, und würden uns über Rückmeldungen freuen.

Link zum Unterrichtsmaterial

Das Material findet man unter folgender URL:

https://oer.ipn.uni-kiel.de/edu-sharing/components/collections?_viewType=1&id=3d505196-248c-4404-9e5a-1029f141aa33

Quellen

Arbeitskreis »Bildungsstandards« der Gesellschaft für Informatik e. V. (2008): Grundsätze und Standards für die Informatik in der Schule. Bildungsstandards Informatik für die Sekundarstufe I. In: Log in 28 (150/151).

Bell, Tim; Vahrenhold, Jan (2018): CS Unplugged—How Is It Used, and Does It Work? In: Hans-Joachim and Komm Dennis and Unger Walter Böckenhauer (Hg.): Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday. Cham: Springer International Publishing, S. 497–521. Online verfügbar unter https://doi.org/10.1007/978-3-319-98355-4_29, zuletzt geprüft am 20.2.2024.

Best, Alexander; Borowski, Christian; Büttner, Katrin; Freudenberg, Rita; Fricke, Martin; Haselmeier, Kathrin et al. (2019): Kompetenzen für informatische Bildung im Primarbereich. Bonn: Gesellschaft für Informatik e.V.

Braune, Gert; Mühling, Andreas (2020): Learning to program. In: Proceedings of the 15th Workshop on Primary and Secondary Computing Education. New York, NY, USA: Association for Computing Machinery (WiPSCE '20).

Buttke, Robby; Engelmann, Lutz; Forman, Franz X. (Hg.) (2011): Informatik S I. Informatische Grundbildung; [Klassen 7 - 10]. 1. Aufl., 3. Dr. Berlin: Duden-Paetec-Schulbuchverl. (Duden).

Curzon, Paul (2013): cs4fn and computational thinking unplugged. In: Michael E. Caspersen, Maria Knobelsdorf und Ralf Romeike (Hg.): Proceedings of the 8th Workshop in Primary and Secondary Computing Education. the 8th Workshop in Primary and Secondary Computing Education. Aarhus, Denmark. New York, NY: ACM, S. 47–50.

Grover, Shuchi (Hg.) (2020): Computer science in K-12. An A to Z handbook on teaching programming. Palo Alto, CA: Edfinity.

Hubwieser, Peter; Mühling, Andreas; Aiglstorfer, Gerd (2013): Fundamente der Informatik. Funktionale, imperative und objektorientierte Sicht, Algorithmen und Datenstrukturen. 2nd ed. Berlin/Boston: De Gruyter.

Humbert, Ludger; Puhlmann, Hermann (2004): Essential Ingredients of Literacy in Informatics. In: Johannes Magenheim und Sigrid Schubert (Hg.): Informatics and Student Assessment. Concepts of Empirical Research and Standardisations of Measurement in the Area of Didactics of Informatics : GI-Dagstuhl-Seminar, September 19 - 24, 2004, Schloß Dagstuhl, Germany. Bonn: German Informatics Soc. (GI) (GI-Edition Seminars, 1), S. 65–76.

IQSH - Institut für Qualitätsentwicklung an Schulen Schleswig-Holstein (2022): Pilotphase Pflichtfach Informatik. Online verfügbar unter https://fachportal.lernnetz.de/sh/faecher/informatik/aktuelles/pilotphase-pflichtfach.html, zuletzt geprüft am 20.2.2024.

K-12 Computer Science Framework Steering Committee (2016): K-12 Computer Science Framework. Online verfügbar unter http://www.k12cs.org, zuletzt geprüft am 20.2.2024.

Klippert, Heinz (2023): Methoden-Kartei für die Sekundarstufe. 48 Lernkarten mit Kurzbeschreibung und Bild zu jeder Methode. 3. Auflage. Augsburg: Klippert Verlag in der AAP Lehrerfachverlage GmbH.

Koubek, Jochen; Schulte, Carsten; Schulze, Peter; Witten, Helmut (2009): Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht. In: Bernhard Koerber (Hg.): Zukunft braucht Herkunft. 25 Jahre „INFOS - Informatik und Schule" ; INFOS 2009, 13. GI-Fachtagung „Informatik und Schule", 21. bis 24. September 2009 an der Freien Universität Berlin. Bonn: Ges. für Informatik (GI-Edition Proceedings, 156), S. 268–279.

Land Schleswig-Holstein (2021): Fachanforderungen Informatik. Online verfügbar unter https://fachportal.lernnetz.de/files/Fachanforderungen%20und%20Leitf%C3%A4den/Sek.%20I_II/Fachanforderungen/21-14520%20Fachanforderungen%20Informatik%20SEK_WEB_PDF%20UA.pdf, zuletzt geprüft am 20.2.2024.

Lave, Jean; Wenger, Etienne (2011): Situated learning. Legitimate peripheral participation. 24. print. Cambridge: Cambridge Univ. Press (Learning in doing).

Leuders, Timo; Hußmann, Stephan; Barzel, Bärbel; Prediger, Susanne (2011): "Das macht Sinn!". Sinnstiftung mit Kontexten und Kernideen. In: Praxis der Mathematik 53 (37), S. 2–9.

Meisner, Julia (2023): In Deutschland erhalten zu wenige Schülerinnen und Schüler Informatikunterricht. Hg. v. GI. Online verfügbar unter https://idw-online.de/en/news822386, zuletzt geprüft am 20.2.2024.

Möller, Kornelia (2001): Genetisches Lehren und Lernen - Facetten eines Begrifffs. In: Diethard Cech, Bernd Feige, Joachim Kahlert, Helmut Schreier, Hans-Joachim Schwier und Ute Stoltenberg (Hg.): Die Aktualität der Pädagogik Martin Wagenscheins für den Sachunterricht. Walter Köhnlein zum 65. Geburtstag. Bad Heilbrunn: Klinkhardt, S. 15–30.

Resnick, Mitchel; Maloney, John; Monroy-Hernández, Andrés; Rusk, Natalie; Eastmond, Evelyn; Brennan, Karen et al. (2009): Scratch. Programming for All. In: Commun. ACM 52 (11), S. 60–67. DOI: 10.1145/1592761.1592779.

Schuster, Jan (2011): Ein genetischer Zugang zum Programmieren mit CGI-Skripten in Python. In: Informatik in Bildung und Beruf – INFOS 2011 – 14. GI-Fachtagung Informatik und Schule. Bonn: Gesellschaft für Informatik e.V, S. 227–236.

Sentance, Sue; Waite, Jane; Kallia, Maria (2019): Teaching computer programming with PRIMM. A sociocultural perspective. In: Computer Science Education 29 (2-3), S. 136–176. DOI: 10.1080/08993408.2019.1608781.

Weinert, Franz E. (2001): Concept of competence: A conceptual clarification. In: Dominique Simone Rychen und Laura Hersh Salganik (Hg.): Defining and selecting key competencies. Seattle: Hogrefe & Huber.

Wing, Jeannette M. (2012): Computational Thinking. Microsoft Asia Faculty Summit. Zianjin, China. Online verfügbar unter https://www.microsoft.com/en-us/research/wp-content/uploads/2012/08/Jeannette_Wing.pdf, zuletzt geprüft am 20.2.2024.

Lizenz

Kontakt

Gert Braune und Andreas Mühling

Arbeitsgruppe Didaktik der Informatik
Leibniz Institut für Pädagogik der Naturwissenschaften und Mathematik & Universität Kiel

gert.braune@email.uni-kiel.de / muehling@leibniz-ipn.de

Informatik vor, im und nach dem Schweizer Gymnasium

Komm, D.
ETH Zürich

DOI: 10.18420/ibis-02-01-09

Zusammenfassung

In diesem Beitrag wird die aktuelle Situation der Schulinformatik in der Deutschschweiz beschrieben. Im Zentrum stehen Unterrichtsbeispiele zum Aufbau von Kompetenzen, die über die gesamte Schulzeit gefördert werden können. Schließlich wagen wir einen Blick in eine Zukunft, in der die 2024 angestoßenen Prozesse zur Umsetzung eines Grundlagenfachs Informatik, welche die gesamte Schweiz betreffen, weitgehend abgeschlossen sind.

Einleitung

Eine (nicht die einzige!) der Hauptaufgaben des Gymnasiums ist es, Schülerinnen und Schüler auf ein Hochschulstudium vorzubereiten. Was aber sind die grundlegenden Kompetenzen, die benötigt werden, um sich erst einmal in der Hochschule der Zukunft und im Anschluss auf dem Arbeitsmarkt der Zukunft behaupten zu können? Oder vielleicht noch fundamentaler formuliert: Was ist das eigentlich für eine Welt, auf die Gymnasium und Hochschule vorbereiten sollen? In einer schnelllebigen Zeit, in der sich manch eine und einer von der Digitalisierung vor sich hergetrieben fühlt, ist die Beantwortung dieser Frage nicht ganz einfach. Einen Teil der zukünftigen Jobs heutiger Zehntklässlerinnen und -klässler gibt es noch gar nicht.

Unumstritten ist jedoch, dass Informatikgrundlagen in den Berufen der Zukunft einen immer größeren Stellenwert haben werden. Von der Ärztin bis zum Logistiker wird es immer wichtiger werden, zumindest das Grundvokabular von Informatikerinnen und Informatikern zu verstehen. Gerade in der Forschung werden die Teams immer interdisziplinärer und es ist schlichtweg unrealistisch, keinen Kontakt zur Informatik zu erwarten. Sollen Probleme aus der Praxis mit Hilfe der Informatik gelöst werden, müssen sie abstrakt dargestellt in die *Sprache der Informatik* übersetzt werden und können dann (innerhalb gewisser Grenzen) gelöst werden. Die Erkenntnisse aus dem abstrahierten Modell erlauben wiederum Rückschlüsse für das ursprüngliche Problem; siehe Abbildung 1.

So enthält beispielsweise der nun bereits etablierte Bachelor-Studiengang *Humanmedizin* der ETH Zürich gleich zwei Informatik-Vorlesungen, die einmal Grundlagen des Programmierens und einmal *Data Science* ins Zentrum stellen. Der Medizin-Studiengang ist dabei keine

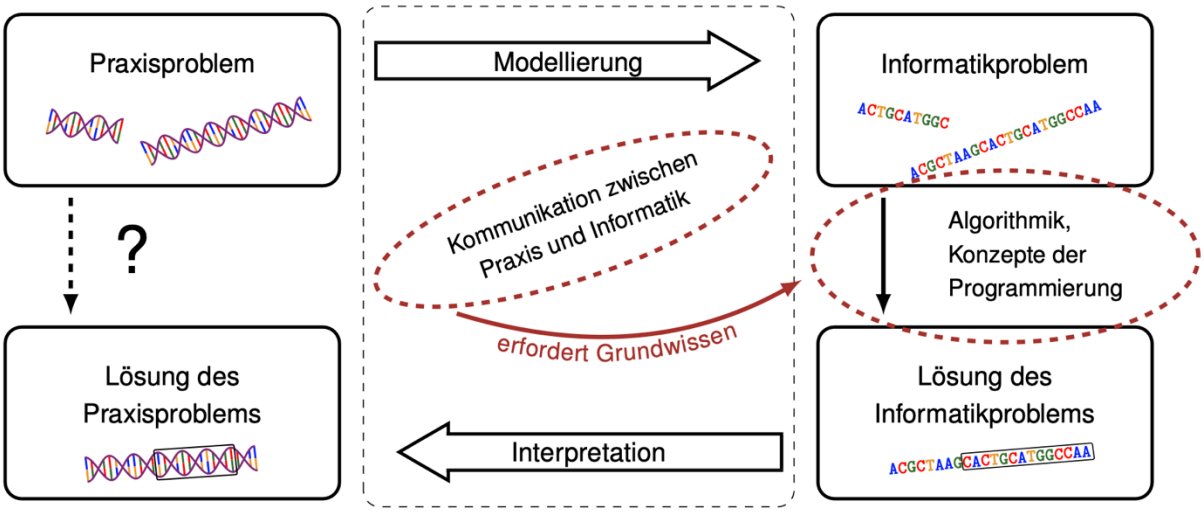


Abbildung 1: Modellierung als Informatikproblem und Interpretation der Lösung.

Ausnahme. Ganz im Gegenteil: Informatikkompetenzen werden in einigen Jahren zu den Grundlagen vieler weiterer Studiengänge gehören – wenn sie dies nicht bereits tun – und das aus gutem Grund; daneben seien natürlich auch per Definition interdisziplinäre Studiengänge wie Bioinformatik, Wirtschaftsinformatik etc. erwähnt, die sich großer Beliebtheit erfreuen.

Die Entwicklung der entsprechenden Lehrveranstaltungen stellt wiederum eine Herausforderung für die Dozierenden dar. Denn die Studierenden haben sich eben nicht für Informatik eingeschrieben und hatten in ihrer Schullaufbahn nicht unbedingt intensiven Kontakt mit der Disziplin; und aktuell bedeutet eine Grundlagenvorlesung somit tatsächlich, ganz am Anfang zu beginnen und nahezu kein Vorwissen vorauszusetzen.

Glücklicherweise nimmt die Informatik in der Schweiz aber auch in den Jahren vor dem Hochschulstudium – ja sogar in den Jahren vor der Sekundarstufe – einen immer wichtigeren Platz ein. Somit ist zu erwarten, dass die Studierenden der Zukunft bereits mit einem gefestigten Grundlagenwissen im Hörsaal sitzen werden und vieles, was aktuell noch an der Hochschule gelehrt wird, bereits aus der Schule bekannt sein dürfte.

In den folgenden Abschnitten wird ein teilweise ambitionierter – aber keineswegs unrealistischer – Blick in die Zukunft gewagt. Für die Schweizer Volksschule (diese umfasst den Kindergarten bis zu Klasse 6 (Ende der Primarstufe) bzw. Klasse 9 (Ende der Sekundarstufe I)) werden Informatikkompetenzen im *Lehrplan 21* formuliert,[1] der bereits vor einigen Jahren in den meisten deutsch- und mehrsprachigen Kantonen und Liechtenstein umgesetzt wurde; ähnliche Lehrpläne existieren für den französisch- und italienischsprachigen Teil des Landes.[2] Für das Gymnasium (je nachdem, wann der Übergang von der Volksschule stattfindet, handelt es sich entweder um ein *Lang-* oder *Kurzzeitgymnasium*) war die Informatik bislang ein sogenanntes

[1] https://www.lehrplan21.ch

[2] https://www.ciip.ch/Plans-detudes-romands/Plan-detudes-romand-scolarite-obligatoire-PER/Plan-detudes-romand-PER/

obligatorisches Fach, wird ab August 2024 (über einen Zeitraum von mehreren Jahren) allerdings zu einem *Grundlagenfach* aufgewertet, womit es genauso für die Schweizer Matura zählt wie beispielsweise Geschichte oder Biologie.

Informatik vor dem Gymnasium

Mit dem Schweizer Lehrplan 21 findet die Informatik offiziell Platz in der Schweizer Volksschule der 21 deutsch- und mehrsprachigen Kantone. Zwar ist nur ein kleines Stundengefäß vorgesehen, welches zu einem Teil zudem für den Aufbau von Anwendungs- und Medienkompetenzen verwendet werden soll, aber grundsätzlich formuliert der Lehrplan den begrüßenswerten Auftrag, Informatikkompetenzen bereits früh in der Schule aufzubauen.

Gerade in den ersten Jahren ihrer Schullaufbahn geht es darum, die Schülerinnen und Schüler zu begeistern und die Informatik (wahrheitsgemäß) vorzustellen als eine Disziplin, die sich zu einem Großteil mit dem kreativen Lösen von Problemen beschäftigt. Da im Lehrplan 21 vom Kindergarten bis zur vierten Klasse (je nach kantonaler Umsetzung) keine eigenen Lektionen für die Informatik vorgesehen sind, kann dies nur in anderen Fächern passieren.

Im Lehrplan 21 werden die aufzubauenden Informatikkompetenzen in *Datenstrukturen*, *Algorithmen* und *Informatiksysteme* unterteilt. Für diese werden dann wiederum Kompetenzstufen formuliert, die die Schritte des Aufbaus über die Schuljahre beschreiben. Im Folgenden werden drei Beispiele für Themen aufgezeigt, die in den unteren Schulstufen behandelt und mit denen unterschiedliche Kompetenzstufen abgedeckt werden können.

Darstellung von Daten in der Volksschule

Die Darstellung von Daten und alle Probleme, die sich daraus ergeben (*Geheimhaltung vor Dritten*, *Resistenz gegen Fehler*, *Effizienz der Darstellung*), machen einen wesentlichen Teil der Informatik aus. Im Lehrplan 21 findet sich unter anderem folgende Kompetenzstufe:

> **MI2.1.3g Die Schülerinnen und Schüler verstehen die Funktionsweise von fehlererkennenden und -korrigierenden Codes.**

Obwohl dies erst für die Sekundarstufe I vorgesehen ist, kann ein Grundverständnis bereits in den ersten Jahren der Volksschule aufgebaut werden. Wie auch bei vielen anderen Konzepten kann die Vermittlung in Form eines *Spiralcurriculums* umgesetzt werden. Dies bedeutet, dass fehlererkennende und -korrigierende Kodierungen in der Schullaufbahn immer wieder behandelt werden, wobei die Komplexität stetig gesteigert wird.

Begonnen werden kann spielerisch ganz vorne – und zwar mit Schülerinnen und Schülern, die womöglich noch nicht lesen, schreiben oder gar zählen können (Hauser et al., 2020):

1. Die Lehrerin einer Klasse ernennt eine Schülerin zu ihrer Assistentin und weiht sie in ein Geheimnis ein; vorgeführt wird ein Zaubertrick und die Assistentin wird fortan als *Hellseherin* bezeichnet. Unter Umständen findet eine Probe der beiden Protagonistinnen in einer vorangehenden Pause statt.
2. Danach steht die Lehrerin mit ihrer Klasse im Kreis um einen großen Tisch, auf dem sich ein Stapel Spielkarten befindet. Die Hellseherin verabschiedet sich für die nächsten Minuten und wartet vor der geschlossenen Türe des Schulzimmers.
3. Nun bittet die Lehrerin einen Schüler, einige Karten, die verdeckt auf dem Tisch liegen, aufzudecken. Hierzu gibt es keine genauen Vorgaben; zum Beispiel könnten insgesamt 13 Karten auf dem Tisch liegen, von denen nun acht aufgedeckt sind.
4. Als nächstes wird ein weiterer Schüler gebeten, entweder eine oder zwei beliebige Karten umzudrehen. Die Entscheidung liegt bei ihm.
5. Ist dies vollbracht, darf die Hellseherin den Raum wieder betreten; sie schaut kurz auf die Karten und kann im Anschluss angeben, ob eine oder zwei Karten vom zweiten Schüler umgedreht worden sind.

Dies kann beliebig oft wiederholt werden: Die Hellseherin verlässt den Raum, ein anderer Schüler deckt sechs Karten auf, eine weitere Schülerin dreht eine oder zwei um, die Hellseherin betritt den Raum und sieht sofort, was passiert ist.

Wie macht sie das? Nun, ein »kleines Detail« muss noch erwähnt werden. Bislang wurde davon ausgegangen, dass die erste Schülerin bzw. der erste Schüler jeweils eine gerade Anzahl Karten aufdeckt (im obigen Beispiel waren es acht bzw. sechs). Sollte es eine ungerade Anzahl sein, muss die Lehrerin eingreifen, damit der Zaubertrick funktioniert. In diesem Fall legt sie eine weitere aufgedeckte Karte dazu. Dies führt dazu, dass die Anzahl der aufgedeckten Karten immer gerade ist, bevor die zweite Schülerin bzw. der zweite Schüler eine oder zwei Karten umdreht. Und jetzt wird auch klar, was die Lehrerin und die Hellseherin als Strategie vereinbart haben: findet letztere eine ungerade Anzahl aufgedeckter Karten vor, wurde eine Karte umgedreht; findet sie eine gerade Anzahl vor, wurden zwei Karten umgedreht. Die Hellseherin »zählt« dazu im Kopf einfach alle aufgedeckten Karten durch: ungerade, gerade, ungerade, gerade, …

Das Konzept des *Prüfbits* kann noch expliziter gemacht werden, indem die Lehrerin auch im Falle einer geraden Anzahl aufgedeckter Karten eine weitere Karte dazulegt, aber diesmal eine verdeckte. Der Zaubertrick kann auch leicht so abgeändert werden, dass von der zweiten Schülerin bzw. vom zweiten Schüler entweder eine oder keine Karte umgedreht wird. Dies entspricht dann den beiden Situationen, dass ein Fehler passiert ist oder nicht. Bei der Übertragung einer Sequenz von Bits wird zum Beispiel versehentlich aus einer Null eine Eins.

Die Idee hinter den *fehlererkennenden Codes* kann in einem späteren Schuljahr zu einem *fehlerkorrigierenden Code* erweitert werden, mit welchem nicht nur erkannt werden kann, dass ein Fehler passiert ist, sondern auch, wo ein Fehler passiert ist. Mit diesem Wissen kann der Fehler dann wiederum automatisch korrigiert werden, was eine Grundlage moderner Datenübertragung darstellt. Die Idee einer Umsetzung mit Karten ist in Abbildung 2 dargestellt; den

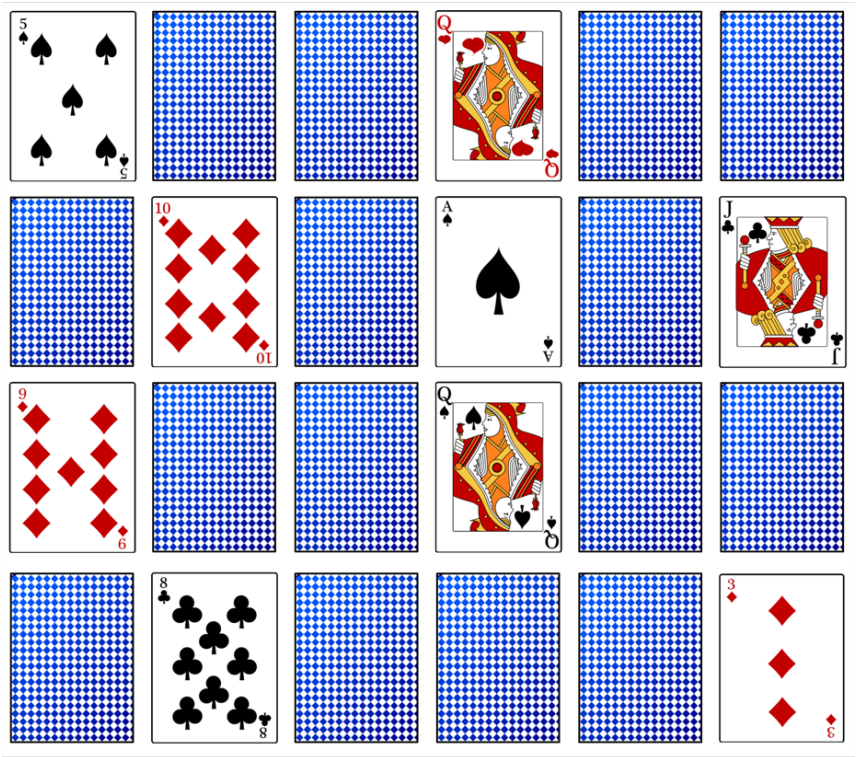


Abbildung 2: Ein fehlerkorrigierender Code. Zu Beginn war die Anzahl der aufgedeckten Karten in jeder Zeile und jeder Spalte gerade. Wird eine Karte (hier das Pik As) umgedreht, so wird die Anzahl der aufgedeckten Karten in genau einer Zeile (hier der zweiten) und einer Spalte (hier der vierten) ungerade.

meisten Leserinnen und Lesern ist dieses Beispiel sicher aus *Computer Science Unplugged* bekannt.[3]

In einer weiteren Kompetenzstufe steht die Verschlüsselung von Daten im Fokus:

MI2.1.2c Die Schülerinnen und Schüler können Daten mittels selbstentwickelter Geheimschriften verschlüsseln.

Zentral ist hier das Adjektiv »selbstentwickelt« – ein ausdrücklicher Auftrag, kreatives Arbeiten ins Zentrum zu stellen. Die intrinsische Motivation der Schülerinnen und Schüler ist kaum irgendwo so groß wie bei diesem Thema. Aufbauend auf Ideen, die die Menschheit seit vielen Jahrhunderten begleiten (beispielsweise der *Cäsar*-Verschlüsselung, welche auch in Sprach- und Mathematiklehrmitteln eine feste Stelle besitzt, *Skytale* oder der *Freimaurerverschlüsselung*) werden Geheimschriften kreiert, die dann von Mitschülerinnen und -schülern geknackt und verbessert werden können.

[3] https://www.csunplugged.org/en/at-a-distance/parity-magic/

Algorithmen und Algorithmisches Denken in der Volksschule

Unter dem Begriff *algorithmisches Denken* werden eine Reihe von Problemlösekompetenzen wie *Abstraktionsvermögen*, *Zerlegung eines Problems in Teilprobleme* oder auch *(clevere) Informationsdarstellung* zusammengefasst; letztere bildet einen zentralen Teil des Prozesses in Abbildung 1.

Viele dieser Problemlösekompetenzen sind nicht einzig und allein in der Informatik zu finden – gut so, wir möchten ja auch über fächerübergreifenden Unterricht sprechen. Das Ergebnis der Anwendung dieser Problemlösekompetenzen ist in der Regel ein Algorithmus, der beliebige Instanzen eines gegebenen Problems lösen kann. Ein solcher Algorithmus besteht aus einfachen, kleinen Schritten, deren Ausführung keinen Intellekt benötigt. Auf der anderen Seite ist das Formulieren eines Algorithmus unter Verwendung der oben erwähnten Problemlösefähigkeiten kreative und sinnstiftende Arbeit. Es geht entsprechend nicht darum, dass die Informatik in die Schule gekommen ist, um die Schülerinnen und Schüler »wie einen Computer denken zu lassen«, sondern ihnen näherzubringen, »wie eine Informatikerin bzw. ein Informatiker denkt«. Es geht um kreatives Problemlösen; es geht darum, die entstandene Lösung aber so gut verstanden zu haben und deswegen so sauber kommunizieren zu können, dass sie sogar einem Computer in einer Programmiersprache »erklärt« werden kann. Natürlich denkt man nun sofort an *Generative AI* und *Prompt Engineering* (welches ebenfalls Aspekte des algorithmischen Denkens benötigt). Dennoch vertritt der Autor die Position, dass Grundlagen des Programmierens erlernt werden sollten, bevor im Unterricht Sprachmodelle zur Codeerzeugung eingesetzt werden.

Heute, da Computer etwa durch Sprachassistenten, Chatbots und eben die jüngsten großen Sprünge in *Generative AI* immer mehr vermenschlicht werden, ist es umso wichtiger, sich den Unterschied zwischen Computern und Menschen bewusst zu machen. Dies findet sich im Lehrplan 21 als Kompetenzstufe deutlich gefordert:

> **MI2.2.2e** Die Schülerinnen und Schüler verstehen, dass ein Computer nur vordefinierte Anweisungen ausführen kann und dass ein Programm eine Abfolge von solchen Anweisungen ist.

Um sich dies zu vergegenwärtigen, können die Schülerinnen und Schüler durchaus einmal die Rolle des Computers einnehmen und bewusst jede Improvisation bzw. das »Lesen zwischen den Zeilen« ausdrücklich unterbinden; beispielsweise im Sprachunterricht:[4]

1. Eine Schülerin und ein Schüler, die sich zunächst in zwei unterschiedlichen Räumen befinden, bilden ein Zweierteam. Der Schülerin wird nun eine Figur aus Bausteinen präsentiert, etwa ein kleiner Turm aus fünf oder sechs Steinen. Die Aufgabe der

[4] https://algdenken.phgr.ch

Schülerin ist es, dieses Bauwerk so zu beschreiben, dass es aus denselben Steinen nachgebaut werden kann – Zeichnungen sind nicht erlaubt.
2. Nach zehn Minuten sollte der Text fertig sein, das Bauwerk wird fotografiert und anschließend zerlegt.
3. Nun betritt der Schüler den Raum und hat weitere fünf Minuten Zeit, um den Turm einzig anhand der schriftlichen Beschreibung nachzubauen.

Was zunächst einfach scheint, entpuppt sich dann oftmals als doch nicht ganz so unkompliziert. »Setze den kleinen blauen Stein auf den großen roten Stein« enthält doch noch einiges an Interpretationsspielraum und am Ende sieht der Turm ganz anders aus als auf dem Foto.

Das Fazit ist, dass es gar nicht so simpel ist, sich eindeutig auszudrücken, insbesondere wenn der Kommunikationspartner nichts anderes tut, als Eins zu Eins die aufgeschriebenen Schritte zu befolgen. Und auch wenn Computer, wie bereits erwähnt, stetig besser darin werden, uns vorzumachen, dass wir mit ihnen wie mit Menschen kommunizieren können, so braucht ihre Programmierung dennoch eine eindeutige Sprache.

Programmieren in der Volksschule

Apropos: Programmieren ist wahrscheinlich der prominenteste Aspekt der Informatik – und vielleicht auch (fälschlicherweise) einer der abschreckendsten. Dabei eignet sich gerade dieses Thema für ein Spiralcurriculum, das früh beginnt, spielerisch wichtige Konzepte vorstellt und nach und nach vertieft.

Wichtig ist in diesem Kontext der Begriff der *konzeptuellen Maschine* (»Notional Machine« (du Boulay et al., 1981)), welche sich aus den bekannten Befehlen und Konstrukten, die die verwendete Programmiersprache zur Verfügung stellt, ergibt. Diese abstrakte Maschine unterscheidet sich in der Regel stark von der *tatsächlichen* bzw. *ausführenden Maschine*, also dem Computer, den wir programmieren. Es ist offensichtlich keine gute Idee, den Schülerinnen und Schülern der Volksschule als ersten Kontakt mit der Programmierung den Befehlssatz aktueller Prozessoren vorzustellen. Besser beginnen wir mit einer einfachen konzeptuellen Maschine, welche die Programmiererin und der Programmierer bei ihrer Arbeit beobachten können. In der Programmiersprache *Logo* wird hierzu eine Schildkröte (die »Turtle«) verwendet, in *Scratch* z. B. eine Katze, in *Greenfoot* z. B. Wombats. Diese bieten den Programmiererinnen und Programmierern jeweils eine Identifikationsfigur: Die Kinder programmieren nicht mehr den Computer, sondern die »Turtle«, »Katze« oder den »Wombat«.

Die Logo-Turtle verfügt beispielsweise über einen sehr überschaubaren Satz von Befehlen, mit denen sie vorwärts und rückwärts laufen und sich nach links und rechts drehen kann; dabei zeichnet sie den zurückgelegten Weg. Wenige und einfache Befehle bedeuten, dass wir ein komplexeres Problem in kleinen, verständlichen Schritten lösen (müssen) und nicht durch Druck auf einen vorgefertigten Knopf, hinter dem alle wesentlichen Ideen verborgen bleiben. (Selbstverständlich werden auch hier aus einer simplen Vorwärtsbewegung unzählige Befehle in Maschinencode.)

Abbildung 3: Blockbasierte Zeichnung eines Vierecks in XLogoOnline Mini.

Jetzt ist wieder die bereits erwähnte präzise Kommunikation erforderlich; es wird genau das ausgeführt, was programmiert wird. Aber im Gegensatz zum Prozessor des Computers kann die Turtle nun dabei beobachtet werden; statt einer *Blackbox* ist sie eine *Glasbox*. Zu jedem Zeitpunkt sehen wir ihren Zustand: die Position, an der sie steht, in welche Richtung sie blickt, mit welcher Farbe sie zeichnet etc.

Für das Programmieren spricht der Lehrplan 21 über konkrete Konzepte, die die Schülerinnen und Schüler in Code umsetzen können sollen:

> **M2.2.2f Die Schülerinnen und Schüler können Programme mit Schleifen, bedingten Anweisungen und Parametern schreiben und testen.**

Über die zu verwendende Programmiersprache werden hingegen keine Vorgaben gemacht und natürlich gibt es auch keine »beste« Programmiersprache. Wenn bereits in den ersten Schulstufen einfache Programme entwickelt werden sollten, bieten sich blockbasierte Programmiersprachen an; Abbildung 3 zeigt beispielsweise ein Programm, mit welchem die oben erwähnte Logo-Turtle in *XLogoOnline*[5] *Mini* ein einfaches Viereck zeichnet. Wenn die Schülerinnen und Schüler die Tastatur beherrschen, ist es möglich, zu einer textbasierten Sprache zu wechseln. Zum idealen Zeitpunkt, zu dem dieser Schritt genau erfolgen sollte, gibt es unterschiedliche Meinungen; der vorgestellte Ansatz geht von einem Wechsel in der fünften oder sechsten Klasse aus.

Auf eine Besonderheit sei aufmerksam gemacht: Um in gängigen imperativen textbasierten Programmiersprachen das Konzept der Schleife zu verwenden, muss in der Regel das Konzept der Variablen eingeführt worden sein. Dieses ist allerdings alles andere als einfach und die Wurzel vieler Fehlvorstellungen (Kohn, 2017). Im Lehrplan 21 werden Schleifen vor Variablen erwähnt; und von »allgemeinen« Variablen wird erst gesprochen, nachdem Parameter als eine Art eingeschränkte Variablen thematisiert wurden. Es ist deswegen sinnvoll, es den Schülerinnen und Schülern zu erlauben, Schleifen ohne Variablen umzusetzen. In vielen blockbasierten Sprachen, aber auch in textbasiertem Logo (z. B. *XLogoOnline maxi*) ist dies mit der sogenannten `repeat`-Schleife möglich. Mit *TigerJython* bzw. *WebTigerPython*[6] existiert außerdem ein

[5] https://xlogo.inf.ethz.ch

[6] https://webtigerpython.ethz.ch

Python-Dialekt, der diesen Schleifentyp übernommen hat und die Umsetzung eines Spiralcurriculums erlaubt:

- Erste Programmiererfahrungen werden bereits in den ersten zwei Klassen mit einer blockbasierten Logo-Variante gemacht, die im Wesentlichen keine Zahlen verwendet.
- Später wird das Prinzip des Parameters eingeführt, wenn die Turtle nicht mehr einfach nur »nach vorne« geht, sondern sich beispielsweise »100 Schritte (Pixel) nach vorne« bewegt.
- Es folgt ein Wechsel zu textbasiertem Logo, in dem die blockbasierten Konzepte nun wiederholt umgesetzt werden. Zunächst werden Befehle einzeln und dann Befehlssequenzen ausgeführt.
- Als nächstes Konzept wird die *Modularisierung* in Form von Unterprogrammen eingeführt. Hier werden zum ersten Mal Parameter definiert und nicht bloß in vorgefertigten Befehlen verwendet.
- Später (etwa ab der Sekundarstufe I) tritt Python (z. B. *WebTigerPython*) an die Stelle von Logo.
- Zu Beginn der Sekundarstufe II hatten die Schülerinnen und Schüler somit bereits Kontakt mit grundlegenden Programmierkonzepten und einer »richtigen« Programmiersprache, die heute in immer mehr modernen Anwendungen eingesetzt wird.

Informatik im Gymnasium

In Zukunft werden Lehrpersonen an Schweizer Gymnasien im Idealfall also eine Klasse voll mit Schülerinnen und Schülern vor sich haben, die bereits Grundlagen über die Verarbeitung von digitalen Daten besitzen. Sie haben beispielsweise bereits mit dem Binärsystem gearbeitet und kennen dessen Bedeutung für die Informatik; sie haben selbst Ideen zur Geheimhaltung von Texten entwickelt; sie haben Daten mit Prüfbits gegen Übertragungsfehler geschützt; und sie haben Grundkonzepte der Programmierung untersucht und umgesetzt.

Im Gymnasium können diese Konzepte nun vertieft werden, wobei stetig an das Vorwissen der Schülerinnen und Schüler angeknüpft wird und weitere Brücken zu Themen innerhalb und außerhalb der Informatik gebaut werden. Parallel zum Aufbau der Kompetenzen in Mathematik können Grundkonzepte der Informatik weiter formalisiert werden. Es folgen zwei Beispiele, die einen Bogen zur Volksschule spannen.

Darstellung von Daten im Gymnasium

Auch für ältere Schülerinnen und Schüler haben Geheimschriften keineswegs ihren Reiz verloren. Hier kann eine Brücke zur Programmierung geschlagen werden, indem beispielsweise die bereits bekannte *Cäsar*-Verschlüsselung programmiert wird. Nach der Vorstellung des ASCII- bzw. Unicodes kann ein gegebener Text zum Beispiel mit wenigen Zeilen Python-Code einfach entschlüsselt werden; siehe Abbildung 4.

```
text = "TYQZCXLETVTDEVCPLETGPLCMPTE"

for k in range(0, 25):
    for i in range(0, len(text)):
        print(chr((ord(text[i]) - 65 - k) % 26 + 65), end="")
    print()
```

Abbildung 4: Entschlüsselung eines mit Cäsar verschlüsselten Texts durch das Ausprobieren aller 25 möglichen Schlüssel. Angenommen wird hier, dass der Text aus den 26 Großbuchstaben des lateinischen Alphabets besteht.

Das Entschlüsseln von Geheimschriften kann genauer unter die Lupe genommen werden und über das bloße Ausprobieren (wie in Abbildung 4) hinausgehen. Hierzu können einerseits statistische Eigenschaften typischer Text ausgenutzt werden und andererseits kann nach Möglichkeiten gesucht werden, dies zu verhindern.

In der Kryptographie kann nun außerdem der Schritt hin zu moderneren Systemen gemacht werden – etwa mit *Public-Key-Kryptographie* und deren teilweise überraschenden Ideen. Realistischerweise kann hier zwar nicht auf die letzten Details der zugrundeliegenden Zahlentheorie eingegangen werden, dafür kann aber ein zweifelsfreier Wirklichkeitsbezug abgeleitet und nachvollzogen werden. In den Lehrplänen für das *Grundlagenfach Informatik*, das in der Schweiz ab August 2024 in die Schulen kommt, findet der Oberbegriff *Cyber Security* einen festen Platz, und somit können viele Konzepte im Unterricht besprochen werden, die auf asymmetrischer Verschlüsselung basieren.

Ähnliches gilt für die anderen eingangs erwähnten »Probleme« bei der Darstellung (und Verarbeitung) von Daten, nämlich die *Resistenz gegen Fehler* und *Effizienz der Darstellung*. Dort wiederum werden die in der Volksschule betrachteten Konzepte nun zu *Hamming-Distanz* und *Huffman-Code*. Die Aufgaben, Daten sicher vor unautorisiertem Zugriff zu halten, sie vor Übertragungsfehlern zu schützen und sie speichersparend zu kodieren, sollten möglichst »hands-on« angegangen werden.

Welche Funktionalität stellt ein Betriebssystem zur Verfügung? Wie kommunizieren Rechner miteinander? Hierbei geht es um nicht weniger, als darum, viele uns umgebende Alltagsphänomene besser zu verstehen (und kritisch zu hinterfragen!), Neugierde zu wecken und möglichst vielen Schülerinnen und Schülern den Wunsch nach und die Fähigkeit zur Mitgestaltung mitzugeben.

Programmierung und Algorithmik im Gymnasium

An dieser Stelle kann damit begonnen werden, die Programmierung und die Algorithmik zusammenzuführen, also die erstellten Algorithmen formaler zu untersuchen. Hier findet zum ersten Mal der Begriff der *Zeitkomplexität* einen Platz. Dieser wird eingeführt, ohne zu genau auf das konkrete Maschinenmodell einzugehen – es wird weiterhin eine *konzeptuelle Maschine*

```python
def binsearch(data, searched):
    left = 0
    right = len(data) - 1
    while left <= right:
        current = (left + right) // 2
        if data[current] == searched:
            return current
        elif data[current] > searched:
            right = current - 1
        else:
            left = current + 1
    return -1

print(binsearch([2,3,5,7,11,13,17], 3))
```

Abbildung 5: Die binäre Suche in Python.

verwendet, bei der sich beispielsweise noch keine großen Gedanken darüber gemacht wird, wie viel länger eine Multiplikation als eine Addition dauert. Bereits für die Volksschule findet sich im Lehrplan 21 hierzu (wohlgemerkt gegen Ende und hinter dem sogenannten *Grundanspruch*) folgende Kompetenzstufe:

> **MI.2.2.3i Die Schülerinnen und Schüler können verschiedene Algorithmen zur Lösung desselben Problems vergleichen und beurteilen (z. B. lineare und binäre Suche, Sortierverfahren).**

Eine Möglichkeit, im Gymnasium Algorithmen in einer »echten« Programmiersprache umzusetzen, Komplexitätsbetrachtungen zu erlauben und gleichzeitig an das vorhandene Wissen anzuknüpfen, ist durch den Python-Dialekt, der bereits im vorangegangenen Abschnitt besprochen wurde, gegeben. Dieser beinhaltet ebenfalls »die Turtle«, die dann nach und nach durch abstrakte Daten und Objekte ersetzt werden kann. Sind Parameter, Variablen, bedingte Anweisungen und Schleifen jetzt bekannt, können nun einfache Datenstrukturen eingeführt werden, beispielsweise Listen.

Die oben erwähnte *binäre Suche* ist eines der bekanntesten Beispiele für eine clevere Lösungsstrategie, welche die Lernenden idealerweise bereits in den ersten Jahren ihrer Schullaufbahn kennengelernt haben – ganz ohne Computer. Eine beispielhafte Implementierung in Python ist in Abbildung 5 gezeigt.

Der Umsetzung ging ein Herleiten (Wiederholen) in Alltagssprache, die Stück für Stück weiter formalisiert wurde, voraus. Das Entwickeln des Codes wird außerdem von intensiven Tests begleitet; gerade die binäre Suche ist zwar recht schnell erklärt, aber viel weniger schnell korrekt umgesetzt (Bentley, 1999). Auf dieser Schulstufe wird die Korrektheit des Algorithmus genauer untersucht, indem die Schülerinnen und Schüler beispielsweise begründen, dass der Algorithmus sicher *terminiert*, also nicht für immer läuft. Auch die Zeitkomplexität kann betrachtet werden, wobei hier der wesentliche Punkt ist, dass der *Suchraum*, also der Bereich, in dem sich die gesuchte Zahl noch befinden könnte, mit jeder Ausführung der Schleife ungefähr halbiert wird – falls das Element nicht bereits gefunden wurde. Auch ganz ohne *Landau-Symbole* und *Rekursionsgleichungen* kann so eine gute Intuition für die logarithmische Zeitkomplexität des Algorithmus hergeleitet werden; im Kern steht die Frage: »Wie oft muss ich die gegebene Liste halbieren, bis nur noch ein Element übrig ist?«

Ähnliche Unterrichtssequenzen können einfache Sortieralgorithmen behandeln oder sich mit der Suche nach Mustern in Daten befassen. Auch hier ist es möglich (und wünschenswert), die entwickelten Algorithmen hinsichtlich Korrektheit und Laufzeit zu untersuchen.

Informatik nach dem Gymnasium

Was wird Dozierende an Schweizer Hochschulen also im Idealfall in ein paar Jahren erwarten? Erstsemesterstudierende, für die die binäre Suche ebenso zum Allgemeinwissen gehört wie Schillers Tell, die Photosynthese oder die erste Ableitung von $f(x) = 2x + 5$. Allgemein sind Techniken trainiert worden, die es erlauben, Problemlösestrategien anzuwenden und das Ergebnis formal und präzise zu kommunizieren. So kann in den Informatik-Grundlagenvorlesungen, die womöglich bald in vielen Studienrichtungen einen ähnlichen Stellenwert wie Mathematik-Grundlagenvorlesungen besitzen werden, bereits ein hohes Niveau erreicht werden. Der Fokus an den Hochschulen kann zudem auf den konkreteren Einsatz von Informatikkonzepten in den jeweiligen Studienrichtungen gerichtet werden – etwa, wenn in der eingangs erwähnten Vorlesung zur Humanmedizin untersucht wird, wie gewisse Voraussagen über bestimmte Gewebeproben gemacht werden können. Eines der Hauptziele muss es sein, dass Hochschulen Studierende in den Hörsälen sitzen haben, die ein korrektes Bild der Informatik haben, und die sowohl ihren Nutzen als auch die nicht zu bestreitende Eleganz ihrer Konzepte kennen.

Jedoch möchte ich mit einer anderen Gruppe von Personen schließen, die – zwar immer wieder erwähnt – in diesem Artikel bislang zu kurz kam; und das, obwohl sie einen der kritischsten Teile unserer Gesellschaft ausmacht: die Lehrpersonen, sowohl in der Volksschule und dem Gymnasium als auch an den Hochschulen. Wenn wir von »Informatik nach dem Gymnasium« sprechen und hier das Bild einer nicht zu fernen Zukunft zeichnen, so muss natürlich gesagt werden, dass die gesamte Vision mit den Lehrpersonen steht und fällt. Und nun haben die dienstälteren unter ihnen wiederum womöglich weder vor, noch während, noch nach dem Gymnasium besonders viel Kontakt mit der Informatik gehabt. Hier sind die Institutionen *nach dem Gymnasium* in der Pflicht.

Gerade Lehrpersonen der Volksschule sind (in der Schweiz) Generalistinnen und Generalisten. Sie hatten wahrscheinlich keine Informatik in der Schule, womöglich auch nicht im Studium und brauchen deswegen für den Unterricht eines Moduls bzw. Fachs, das es zu ihrer eigenen Schulzeit nicht gegeben hat, die Unterstützung *nach dem Gymnasium*. Hier sind gerade die pädagogischen Hochschulen gefragt; es muss beispielsweise sichergestellt werden, dass, nachdem die obligatorischen Lehrplan-21-Weiterbildungen abgeschlossen sind, freiwillige Weiterbildungsprogramme angeboten werden, die interessierten Lehrpersonen (und davon wird es immer genug geben) die Werkzeuge an die Hand geben, um einen sinnstiftenden und nachhaltigen Informatikunterricht durchzuführen. Hier gibt es mittlerweile einiges an Bewegung mit Dozierenden, die wiederum eine formale Informatikausbildung hinter sich haben.

Für Gymnasiallehrpersonen ist die Situation anders, aber dennoch an manchen Stellen vergleichbar. Die typische Schweizer Informatiklehrerin und der typische Schweizer Informatiklehrer haben einen Informatik-Masterstudiengang absolviert und anschließend ein Lehrdiplom erhalten. Aber nun werden schnell Lehrpersonen benötigt, weswegen das *GymInf*-Programm[7] (kurz für »Informatikausbildung für Gymnasiallehrkräfte«) ins Leben gerufen wurde. Auch hier haben wir es mit einem ambitionierten Unterfangen zu tun, wenn im Rahmen von 107 ECTS-Punkten zum Beispiel aus der Biologie- auch eine Informatiklehrperson gemacht werden soll. Nach einigen Semestern und vielen Interaktionen mit den äußerst motivierten Teilnehmenden kann aber auch hier ergänzt werden: ambitioniert – aber keineswegs unrealistisch.

Literaturverzeichnis

Bentley, J. (1999): *Programming Pearls*, zweite Auflage, Addison-Wesley Professional.

du Boulay, B., O'Shea, T. & Monk, J. (1981): The black box inside the glass box: presenting computing concepts to novices, *International Journal of Man-Machine Studies* 14(3), pp. 237–249.

Hauser, U., Hromkovič, J., Klingenstein, P., Lacher, R., Lütscher, P. & Staub, J. (2020): *Einfach Informatik – Rätsel und Spiele ohne Computer*, Klett und Balmer Verlag.

Kohn, T. (2017): Variable evaluation: an exploration of novice programmers' understanding and common misconceptions. In *Proc. of SIGCSE 2017*, pp. 345–350.

Lizenz

Kontakt

Dennis Komm

ETH Zürich

https://people.inf.ethz.ch/dkomm/, dennis.komm@ethz.ch

[7] https://www.unifr.ch/gyminf/de/