# Erste Programmiererfahrungen in der Primarstufe.

## Konzepte und Werkzeuge für Grundvorstellungen beim Programmieren und darauf aufbauend hin zu block- und textbasierten Programmiersprachen

Michael Rieseneder und Wolfgang Wagner

*Wer etwa vom Coden als grundlegender Kulturtechnik im Sinne des Schreibens 4.0 keine Ahnung hat, kann Computer und ihre virtuellen Welten weder für „gute" noch „schlechte" Zwecke einsetzen und ist umso mehr Sklave der heutigen Verhältnisse. (Barberi, Berger, & Himpsl-Gutermann, 2017, S. 2)*

*Die zunehmende Digitalisierung unserer Gesellschaft macht für eine kompetente und aktive Teilnahme eine Auseinandersetzung mit Computational Thinking notwendig. Über eigene spielerische Handlungserfahrungen soll ein Verständnis für Algorithmisierungen und Programmierungen aufgebaut werden. Das eigene Tun wird über den Aufbau von mentalen Modellen (Wartha, 2011, S. 11) und der Entwicklung von Grundvorstellungen (Vom Hofe, 1995) derart abstrahiert, dass den symbolhaften Programmen stets die zuvor erworbenen Handlungserfahrungen innewohnen. Dabei ist eine möglichst enge Passung (Walter 2017, S. 32ff.) bei Intermodalitätswechseln zwischen den Repräsentationsebenen der Handlung, des Bildes und des Symbols (Bruner, 1971) ein zentraler Punkt des vorgestellten Konzepts.*

*Aufbauend auf Handlungserfahrungen wird ein didaktisches Konzept vorgestellt, das von einem Bewegungsspiel ausgehend über den Einsatz von konkreten Bodenrobotern hin zu virtuellen Programmierumgebungen führt. Mit dieser Vorgangsweise soll gewährleistet sein, dass die abstrakten informatischen Ideen auf mentalem Prozessverständnis über eigenes Tun generiert werden. Das vorgestellte Konzept wird im Rahmen des Education Innovation Studios der Pädagogischen Hochschule Oberösterreich von den Autor*innen erforscht.*

### Einleitung

Digitalisierung verändert unsere Gesellschaft so stark, wie zuvor die Erfindung der Sprache oder die Erfindung des Buchdrucks und wirft für die Schule mehr Fragen auf, ob Computer genutzt werden sollen oder nicht (Döbeli Honegger, 2017, S. 30). Um einen kompetenten Umgang mit digitalen Medien pflegen zu können, bedarf es nach Döbeli Honegger (2017, S. 77) sogenannten digitalen Kompetenzen. Diese sind durch die drei Dimensionen der Anwendungskompetenz (kompetentes und effektives Nutzen digitaler Medien), der Medienbildung (Inhalte produzieren und reflektieren) und der Informatik (unter anderem Inhalte des Computational Thinking) definiert. Bereits in den 60er Jahren des vorigen Jahrhunderts forderte Alan Perlis eine Auseinandersetzung aller Studierenden in sämtlichen Disziplinen mit informatischen Inhalten ein (Guzdial, 2008, S. 25). Seymour Papert (1980) entwickelte in den 80er Jahren die Programmiersprache LOGO, die speziell für ein Coding nach den Grundsätzen des Konstruktivismus (Piaget & Szeminska, 1975) bzw. dem nach ihm entwickelten Konstruktionismus (Papert & Harel, 1991) für Kinder bereits ab dem Kindergartenalter geschaffen war. Janette Wing (2006) formulierte im jetzigen Jahrhundert die Schlüsselaufgaben zur Umsetzung eines Computational Thinkings einhergehend mit der Forderung diese im K-12 Bildungsbereich (Kindergarten bis zur 12. Schulstufe) in den STEM Bereichen (Science, Technology, Engineering and Mathematics) umzusetzen. Solche Umsetzungen werden im österreichischen Lehrplan der Volksschule derart gefordert, dass eine Medienbildung anzubieten ist, welche „die Jugend mit dem für das Leben und den künftigen Beruf erforderlichen Wissen und Können auszustatten hat und zum selbsttätigen Bildungserwerb zu erziehen [hat]." (BMUKK, 2012, S. 9). Weiters wird eine „Entwicklung und Vermittlung grundlegender Kenntnisse, Fertigkeiten, Fähigkeiten, Einsichten und Einstellungen" gefordert, ebenso wie das „Erlernen der elementaren Kulturtechniken (ein-

Rieseneder, M. & Wagner, W. (2020). Erste Programmiererfahrungen in der Primarstufe. Konzepte und Werkzeuge für Grundvorstellungen beim Programmieren und darauf aufbauend hin zu block- und textbasierten Programmiersprachen. F&E Edition, 26, 7-15.

schließlich eines kindgerechten Umganges mit modernen Kommunikations- und Informationstechnologien." (BMUKK, 2012, S. 9-10)
In den digitalen Kompetenzen digi.komp4 des BMBWF (2016), die bis zum Abschluss der vierten Schulstufe erreicht werden sollten, finden sich ebenso Kompetenzen, die das Computational Thinking beschreiben, z. B. die Automatisierung von Handlungsanweisungen und die Koordination von Steuerung von Abläufen. Computational Thinking ist im Lehrplan der Volkschule im Gegensatz zu dem Lehrplan ab der Sekundarstufe nicht explizit angeführt, es finden sich aber die Inhalte des Computational Thinking in verschiedenen Fächern (Antonitsch et al., 2014).

In diesem Artikel werden Möglichkeiten vorgestellt, Kindern im Primarstufenbereich (und teilweise davor) einen Zugang zum Programmieren zu bieten. Die konstruktivistische Vorgangsweise generiert ausgehend von Erfahrungen über das eigene Handeln des Kindes, darauf aufbauend zu Grundvorstellungen (Wagner, 2019, S. 71) mit zunehmenden Abstrahierungen schließlich zu textgebundenem bzw. blockbasiertem Coding. Dieses in seiner Abstrahierung fortgeschrittene Coding ist durch eigenes Tun aufgebaut und dieses Tun kann von den Kindern mit abstrakten Begriffen in Verbindung gebracht werden.

## Computational Thinking

„Computational Thinking ist nicht die Art, wie Computer denken. Sondern die Art, wie Menschen denken müssen, um Computer dazu zu bringen, erstaunliche Dinge zu leisten." (Curzon, 2018, S. 217). Die Informatik kämpft mit dem Klischee, dass dies genau umgekehrt aufgefasst wird (Gallenbacher, 2018, S. V).
Das Prinzip des Computational Thinking ist, wenngleich auch nicht so benannt, ein altes: Der Euklidische Algorithmus, eine Form der Computational Thinking, ist beinahe 2.500 Jahre alt. Im frühen 17. Jahrhundert gab es sogenannte Computer. Das waren keine Maschinen, sondern Menschen, die die Fähigkeit besaßen, mathematische Probleme so auszuführen, dass damit eine ganze Gruppe ähnlicher Aufgabestellungen gelöst werden kann. Diese menschlichen Computer bedienten sich schon der Kompetenz des Computational Thinking (Denning & Tedre, 2019, S. 17). In Folge waren Menschen bemüht, Maschinen zu konstruieren, die Berechnungen automatisiert ausführen können. Der Begriff des Computational Thinking kam erstmals in den 1940ern, als elektronische Computer auf dem Vormarsch waren, auf und entwickelte sich seit dieser Zeit weiter. Seymour Papert (1980) definierte Computational Thinking als mentale Fähigkeit, die Kinder beim Üben des Programmierens erlangen. Einige Wissenschaftler der Computerwissenschaften erklärten zu dieser Zeit Computational Thinking als Denkleistung beim Ausführen von Wissenschaften, die sich hauptsächlich mit Berechnungen befassen (Denning, 2017b, S. 35). Wing griff ihn 2006 erneut auf und forderte, dass Computational Thinking als weitere Kulturtechnik neben Lesen, Schreiben und Rechnen allen Kindern vermittelt werden sollte. Seit 2012 (Denning, 2017b, S. 35) wird oft auf die Definition von Al Aho (2012) zurückgegriffen, bei der Computational Thinking als Denkprozess, der beim Formulieren und Lösen von Problemen, die durch Algorithmen repräsentiert werden können und welche durch Computermodelle ausgeführt werden können, stattfindet, beschrieben wird.

Beim Computational Thinking sollen Code als Daten und Daten als Code interpretiert werden. Es geht darum, Handlungen genau beschreiben zu können, Algorithmen erstellen zu können. Hier bedarf es Strategien, um komplexe Probleme in lösbare Teilprobleme zerlegen zu können, Strategien der Abstraktion und Dekomposition. Um Lösungen zu kreieren bedarf es Muster erkennen zu können bzw. heuristische Verfahren anwenden zu können. (Wing 2006, S. 33–34) Denkprozesse, die unter anderem beim Computational Thinking benötigt werden, sind das Sequenzieren, Finden von Alternativen, Iterationen, Abstrahieren, Rekursionen, Dekomposition, Debugging usw. (Denning 2017a, S. 15).

Somit ergeben sich folgende vier Hauptaspekte des Computational Thinking unter dem Blickwinkel der Methoden (Denning & Tedre, 2019, S. 9):

- Abstraktion/Modellierung
- Dekomposition
- Muster erkennen
- Algorithmisieren

Ab der Sekundarstufe wird in der Verbindlichen Übung Digitale Grundbildung das Computational Thinking explizit angeführt. Die Schüler*innen sollen dabei „mit Algorithmen arbeiten" (BMBWF, 2018), Abläufe aus dem Alltag beschreiben, Codierungen verwenden, Algorithmen nachvollziehen und formulieren und Programmiersprachen kreativ nutzen können.
Im Lehrplan der Volksschule ist zu Computational Thinking, wenngleich nicht wörtlich angeführt, folgendes angeführt: Algorithmisieren beim Verstehen und Erstellen von Bildgeschichten oder Spielregeln, Anleitungen bzw. Kochrezepten; logische Operationen in Richtig-Falsch-Aufgaben; Repräsentieren von Daten; Erstellen und Lesen von Tabellen, Modellieren, Symbole finden und verstehen, funktionale Beziehung begreifen, schriftliche Rechenverfahren usw.
In vielen Aspekten überschneiden sich die Inhalte der Informatik mit denen der Mathematik (Savard & Highfield, 2015, S. 540-544). So sind mathematische Fähigkeiten ein Prädiktor für Erfolg beim Erlernen informatischer Fähigkeiten (Bower, 2008, S. 13). Wobei ein Unterschied darin liegt, dass die Mathematik sich mit der Beschreibung statischer Strukturen und die Informatik mit den dynamischen Strukturen beschäftigt (Schubert & Schwill, 2011, S. 15).
Computational Thinking ist kein explizites Wissen, sondern ein Bündel an Kompetenzen (Denning, 2017b, S. 36-37). Bower (2008, S. 13) beschreibt, dass Vorwissen zu den Konzepten der Informatik und das Lernen von Problemlösestrategien, gefördert durch einen kompetenzorientierten Laboransatz, bei späteren Programmierkursen zu besseren Ergebnissen führten. Daneben kritisiert er, dass viele Konzepte für Informatik- bzw. Programmierkurse den Fokus auf das Erlernen von Programmiersprachen, anstatt auf das Entwickeln von Programmierstrategien legen, obwohl die größte Schwierigkeit für Lernende im Planen von Algorithmen liegt und weniger auf dem Ausformulieren in spezifischen Programmiersprachen (Bower, 2008, S. 17-19). Dabei sollte der Blick nicht nur auf das Coding, das Ausformulieren von Algorithmen, gerichtet werden, weil dies nur einen kleinen Teil des Computational Thinking darstellt (Tedre & Denning, 2016, S. 126).
Mit den im Folgenden vorgestellten Konzepten und Werkzeugen können Teilbereiche des Computational Thinkings schon ab der Primarstufe gelehrt und gelernt werden. Dazu werden nicht zwingendermaßen Computer oder Tablets benötigt, in einigen Fällen nicht einmal elektrischer Strom.

## Zugang zu einem ersten Programmieren

Ein Verständnis für ein erstes informatisches Denken kann bei Kindern im Kindergarten- bzw. im Primarstufenalter generiert werden (Schwill, 2001, S. 13-30). Im Sinne einer Informatik ohne Strom (Bell, Alexander, Freeman & Grimley, 2009, S. 20-29) kann dies über eigene Handlungserfahrungen der Kinder erfolgen. Die Repräsentationsebene der Handlung sollen mit denen des Bildes und des Symbols – folgend dem EIS Prinzip nach Jerome Bruner (1971) – verbunden werden. Informatik beschäftigt sich hauptsächlich mit symbolhaft repräsentierter Information (Kalbitz et al., 2011, S. 137). Diese soll durch das enge Verknüpfen mit der ikonischen (bildhaften) und enaktiven (handelnden) Repräsentationsebene, welche von Kalbitz et al. (2011, S. 139) besonders betont wird, begreifbar gemacht werden. Abstrakte Begriffe (wie der Befehl forward) können somit mit einer Handlung und einem Bild einer Handlung verbunden werden, wodurch eine Grundvorstellung (Vom Hofe, 1995) des Begriffs generiert werden kann. Im Sinne eines konstruktivistischen Vorgehens kann dies auf spielerische Art über das Roboterspiel (Wagner, 2019, S. 71-72) erfolgen.

Bei diesem Bewegungsspiel steuert ein Kind (Ingenieur*in) ein anderes Kind (Roboterkind) zu einem vereinbarten Ziel hin. Die Steuerungs-

befehle sind (ein Schritt) vorwärts, rückwärts und die Drehbewegungen Linksdrehung und Rechtsdrehung, jeweils im rechten Winkel ausgeführt. Die Steuerungsbefehle können mit Zeichensymbolen noch vor dem Schreiberwerb notiert werden bzw. später in Form von Abkürzungen (forward FD, rückwärts BK; left turn LT und right turn RT). Die Algorithmisierungen der Bewegungen entsprechen der Programmiersprache LOGO (Papert, 1980) und finden sich in vielen Programmiersprachen (Scratch, Swift Playground, Java etc.) wieder.

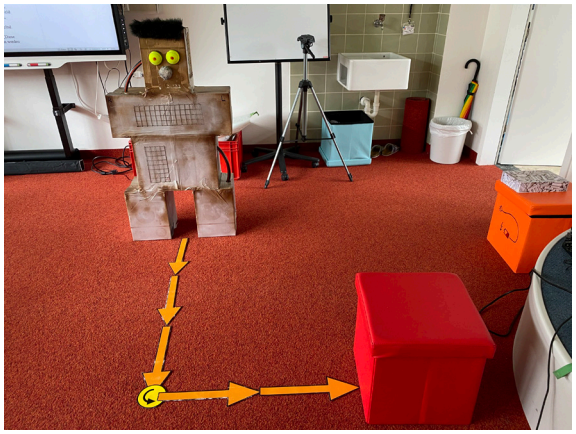

*Abb. 1: Visualisierung eines Bewegungsablaufs mit Bewegungskarten beim Bewegungsspiel (selbst erstellt)*

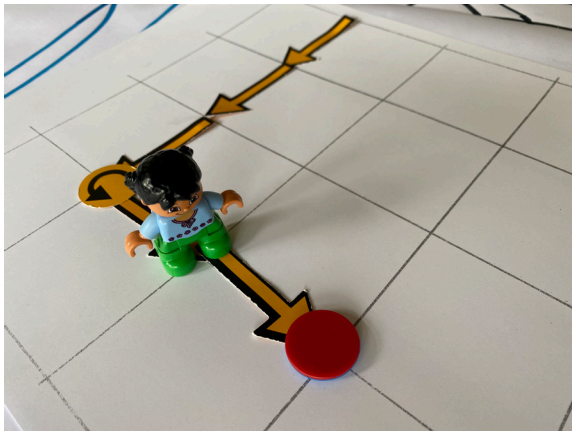

*Abb. 2: Visualisierung eines Bewegungsablaufs mit Bewegungskarten mit einer Spielfigur (selbst erstellt)*

Die Bewegungen der Kinder können mittels Bewegungskärtchen (siehe Abb. 1) am Boden visualisiert werden und von ihnen selbst abgeschritten werden. Die Bewegungskärtchen entsprechen in deren Form der Notierung durch Zeichensymbole der Befehle. Die Notierung eines Weges (z. B.: 3FD, LT, 2FD) entspricht bereits dem Vorgang des Programmierens, wobei die Kinder vom selben Ausgangspunkt, bei gleicher Schrittlänge immer wieder, also automatisch, zum selben Ziel gelangen. Das Roboterspiel kann im Anschluss mit einer Spielfigur, die auf einem Rasterfeld zu einem mit einem Spielstein gekennzeichneten, vereinbarten Ziel gesteuert werden soll, durchgeführt werden. Wieder ist es möglich mit Bewegungskärtchen den Weg zu visualisieren (siehe Abb. 2) und durch Ziehen der Spielfigur den Weg bzw. die Programmierung handelnd nachzuvollziehen. Beim Roboterspiel können Erkenntnisse für folgende Fähigkeiten entwickelt werden. Diese Erkenntnisse bilden die Basis für weitere Anwendungen, auf die stets zurückgegriffen werden kann und die stetig weiterentwickelt werden können:

- Dekomposition: Zerlegen eines Problems in Einzelteile.
- Algorithmisieren: Einzelteile werden zu Algorithmen, die Lösungen beschreiben, verbunden.
- Programmieren: Befehle werden so verbunden, dass ein Ziel damit erreicht wird.
- Mustererkennung: Beim Abschreiten eines Quadrates (z. B.: FD, RT – 4-mal).
- Embodiment: Ein sich Hineinversetzen in ein zu steuerndes Objekt.
- Debuggen: Fehler im Bewegungsablauf werden korrigiert.
- Problemlösen: Durch Ausschließen einer Bewegungsform, diese trotzdem verwirklichen (RT durch 3LT).
- Funktionen: Ein Befehl kann durch andere Befehle ausgeführt werden (LT wird mit 3RT beschrieben).
- Automatisieren: Erkenntnisgewinnung: Gleiche Programme führen zum gleichen Ziel.
- Grundvorstellung: Diese sind angebahnt, wenn Wechsel der Repräsentationsstufen gelingen.
- Iteration: Wiederholungen von Befehlen (FD, FD, FD, FD wird mit 4-mal FD beschrieben).
- dynamische Geometrie: Geometrische Inhalte werden durch Bewegungen dargestellt.

### Bee-Bot

Laut dem Lehrplan der Volksschule sollen Kinder zum „Denken und zur Abstraktion geführt werden." (BMMUK, 2012, S. 26–27). Grundvorstellungen, die durch das Roboterspiel erlangt wurden, können mit Bodenrobotern, z. B. dem Educational Robot (Catlin, Kandlhofer & Holmquist, 2018) Bee-Bot, auf eine weitere Abstraktionsstufe gehoben werden.
Im englischsprachigen Raum findet der Bee-Bot schon lange Verwendung. In Österreich kam sein Aufschwung mit dem Projekt Denken lernen – Probleme lösen für die Primarstufe des BMBWF, das seit dem Schuljahr 2017/18 läuft, bei dem österreichweit an Volksschulen Bee-Bots, iPads und Kästen mit LEGO WeDo2.0 verliehen wurden, um Computational Thinking damit zu schulen (Himpsl-Gutermann et al., 2018, S. 7-15). Alle Volksschulen in Niederösterreich wurden im Anschluss daran mit Bee-Bots ausgestattet (Niederösterreichische Landeskorrespondenz, 2018).
Bee-Bot ist ein Kofferwort aus den englischen Begriffen bee und robot, wobei das Aussehen des Bodenroboters beschrieben wird. Auf dem Rücken des gelb-schwarz gestreiften Roboters befinden sich Tasten (4 orange Pfeiltasten, grüne Go-Taste, Pause-Taste und Löschtaste), durch die er programmiert bzw. das Programm gestartet werden kann. Vier Tasten (FD, BK, LT, RT) zeigen ähnliche Zeichensymbole wie sie beim Roboterspiel verwendet wurden. Über diese Tasten können dem Bee-Bot bis zu 40 Befehle eingegeben werden, die er nach dem Drücken der der grünen Go-Taste ausführt. Sämtliche Befehle sind bereits aus dem Roboterspiel bekannt und ein Verständnis für diese sind mit eigenen Handlungserfahrungen verbunden. Dies bedeutet, dass den abstrakten Befehlen für das Programmieren des Bee-Bot eigene Handlungen bzw. mentale Bilder (Wartha, 2011) innewohnen. Dabei können die Kinder über Embodiment auf mentalem Weg unterstützt durch eigene Bewegungen mit dem Roboter interagieren (Catlin, Kandlhofer & Holmquist, 2018).
Ein Aufschreiben der Befehle begünstigt eine Reflexion über die Planungen der Programmierungen des Bee-Bot. Bee-Bots bieten keine Form einer symbolischen Darstellung von eingegebenen Befehlen an. Verschiedene Übungsformate bereichern den Umgang mit dem Bee-Bot. So können z. B. Kinder auf quadratischen Zettel mit der Seitenlänge von 15 cm (entspricht der Schrittlänge des Bee-bots) Teilbereiche einer Stadt (Häuser, Straßen, Dienstleistungen usw.) zeichnen und diese dann nebeneinander so anordnen, dass eine „Bee-Bot-City" entsteht. In dieser von den Kindern konstruierten Stadt können mittels Storytellings verschiedene Wege, die der Roboter beschreiten soll, entwickelt und programmiert werden. Aufgaben dieses Formats bieten eine natürliche Differenzierung (Krauthausen, Scherer & Scherer, 2017, S. 45-49) für heterogene Lerngruppen im inklusiven Sinne an.

### Blue-Bot

Blue-Bots sind Bee-Bots, die zusätzlich über Bluetooth und einen Tactile Reader (eine Steuerleiste, die mit Befehlen ausgelegt wird) steuerbar sind. Durch die Anordnung der Befehle auf der Steuerleiste wird die Notation des Programmes sichtbar. Diese Visualisierung in Verbindung mit der Bewegung des Blue-Bot ermöglicht eine Reflexion der abstrakten Programmierung simultan zur Ausführung des Roboters.
Beim Einsatz des Blue-Bots gemeinsam mit dem Tactile Reader brauchen Programme nicht mehr notiert werden. Debuggen ist durch Austauschen einzelner Befehle möglich und durch Aktivieren der Blue-Bot daraufhin überprüfbar. Um den Kindern weitere Elemente der prozeduralen Programmierung näher zu bringen, bietet sich z. B. der Bodenroboter Cubetto für die Einführung von Funktionen an.

### Cubetto

Der Cubetto ist ein würfelförmiger Bodenroboter aus Holz, der mittels einer quadratischen Holztafel, in die man die bereits von Roboterspiel, Bee- bzw. Blue-bot bekannten Bewegungsbefehle (FD, LT, RT) steckt.
Der Cubetto weist unterhalb seiner Eingabezeilen eine Funktionszeile auf, die mit einem einzelnen Funktionsbefehl abgerufen werden kann. Um erstmals Funktionen als solche zu benennen und in Form eines Befehls einzuge-

ben, kann auf die Erfahrungen des Roboterspiels zurückgegriffen werden. Hier wird den Kindern eine Aufgabe gestellt, bei der das Roboterkind rechts abbiegen soll, jedoch der Befehl für die Rechtsdrehung nicht erlaubt ist. Die Kinder gelangen zu der Erkenntnis, dass drei Linksdrehungen derselben Endposition wie die einer Rechtsdrehung entsprechen. Die beschriebene Vorgangsweise soll ein Verständnis für Funktionen aufgebaut auf eigene Handlungen gewährleisten.

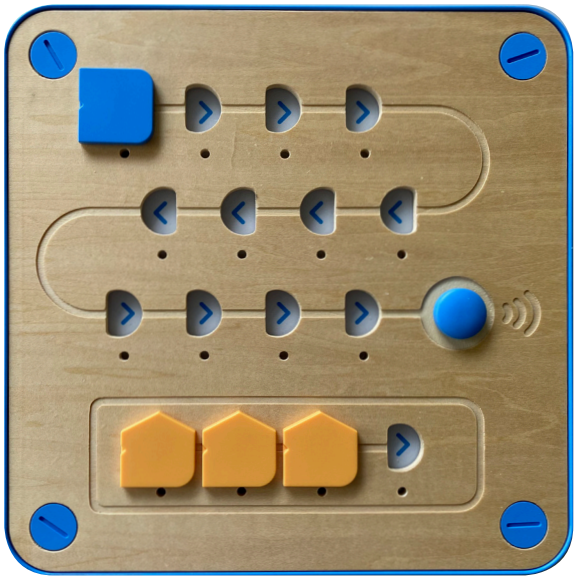

*Abb. 3: Cubetto mit blauem Funktionsbefehl der 3 Linksdrehungen ausführt (3 LT)*

### Matatalab

Ähnlich zu Cubetto wird der Bodenroboter Matatalab über das Einstecken von Bewegungsbefehlen (FD, BK, RT, LT) als Pfeilsymbole, welche aus dem Roboterspiel bekannt sind, und Funktionsbefehlen, welche hier durch die mathematische Notation einer Funktion dargestellt sind, auf ein quadratisches Feld programmiert. Den Bewegungsbefehlen kann erstmals die Häufigkeit ihrer Ausführung angehängt werden (z. B. 4FD). Zusätzlich können mit Matatalab Iterationen in Form von Schleifen programmiert werden. Aus Vorerfahrungen aus dem Roboterspiel, bei dem ein Kind den Umfang eines Quadrates mit einer bestimmten Seitenlänge z. B. zwei Schritte abschreiten soll, kann die Notation von Schleifen innerhalb von Programmen konzipiert werden. Nach ersten Programmentwürfen in der Form von 2FD, RT, 2FD, RT, 2FD, RT, 2FD, RT wird nach einer einfacheren Notation der Wiederholung (2FD, RT) gesucht und eine Notation ähnlich [2FD, RT]•4 verwendet. Matatalab bietet für die Begrenzung und die Häufigkeit der Ausführung der Schleife eigene Befehle an. Alle bis jetzt vorgestellten Roboter können als Derivate von Paperts Turtle LOGO angesehen werden. Um den Kindern den Schritt der Abstraktion zu blockbasierten (z. B. Scratch) bzw. textbasierten (z. B. Swift, Python) Programmiersprachen zu ermöglichen, werden den Kindern mit dem OSMO Coding Awbie und dem Ozobot andere Formen der Notation von Befehlen bzw. Programmen nähergebracht.

### OSMO – Coding Awbie

Osmo ist eine App, die auf Apple Umgebungen wie IOS, iPadOS läuft. Reale Blöcke werden von der eingebauten Kamera des Apple Gerätes erfasst und lassen auf Knopfdruck des Kindes die virtuelle Figur Coding Awbie in seiner virtuellen Welt bewegen. Ein Verständnis für die Anordnung von Blöcken beim blockbasierten Programmieren kann für Kinder vor dem Schreiberwerb generiert werden. Durch das Manipulieren der drehbaren Bewegungssymbole ist das Programmieren mit eigenen Handlungen verbunden. Das Verständnis für eine blockbasierte Programmierung kann in Folge etwa bei Ozoblockly, Scratch etc. genutzt werden.

### Ozobot

Ozobots sind kleine Bodenroboter, die auf zwei Arten programmiert werden können. Einerseits können sie optisch über Farbcodes und Linien gesteuert, andererseits mittels der Programmiersprache Ozoblockly programmiert werden. Die Entwicklungsumgebung für Ozoblockly bietet Programmierblöcke in fünf verschiedenen Komplexitätsstufen an.
Ein Verständnis für blockbasiertes Programmieren kann über eine Form des Roboterspiels enaktiv aufgebaut werden. Ungefähr halbmetergroße Platten (etwa aus Styrodur) repräsentieren in konkreter Form die virtuellen Blöcke von Ozoblockly. Mit Befehlskärtchen in Textform

können nun die konkreten Blöcke zu Programmblöcken vervollständigt werden. Die damit erzeugten Bewegungsanweisungen können daraufhin von den Kindern selbst ausgeführt werden (Wagner, 2018).

### Scratch

Scratch ist eine blockbasierte Programmiersprache ähnlich zu Ozoblockly, welche die Kinder durch die Steuerung physischer Roboter gelernt haben. Scratch wurde 2007 erstmals als visuelle Programmiersprache für Kinder und Jugendliche veröffentlicht. Im Jänner 2019 wurde die Version 3.0 der Programmiersprache veröffentlicht, welche nun auf HTML5 basiert und somit auch auf mobilen Geräten im Standard-Browser ausgeführt werden kann. Im Gegensatz zu den bisher vorgestellten Werkzeugen werden mit Scratch keine physischen Roboter gesteuert, sondern es werden virtuelle Figuren (Sprites) am Bildschirm bewegt. Zusätzlich zu den schon bekannten Befehlen und Kontrollstrukturen (z. B. Schleifen), werden erstmals Ereignisse (Interrupts) explizit benannt. Durch sie wird es möglich, die virtuelle Figur so zu programmieren, dass sie durch Maussteuerung oder Tastatureingaben gesteuert werden kann. Der Handlungsbezug bei Scratch ist enaktiv-virtuell. Über mit Scratch verbundenen Robotern (LegoWeDo etc.) und deren Bewegungsausführungen können Programme aus zusammengesetzten virtuellen Befehlsblöcken in konkreter Form überprüft werden.

### Swift Playgrounds

Swift ist eine textbasierte Programmiersprache von Apple für die Betriebssysteme von Apple und Linux, in der unter anderem professionelle Apps für das iPad programmiert wurden.
Die App Swift Playgrounds bietet einen spielerischen Erwerb der Programmiersprache durch verschiedene Aufgaben, die mit dem virtuellen Roboter Byte erledigt werden sollen, an. Dabei werden zunächst Aufgaben zur Steuerung des virtuellen Roboters durch einfache Labyrinthe gestellt. So gelangt er z. B. durch das Programm moveForward(); turnLeft(); moveForward(); zu seiner ersten Belohnung. Je mehr Aufgaben gelöst wurden, umso komplexer werden sie. Schnell werden zum Lösen der neuen Aufgaben neue Strukturen wie Funktionen, Schleifen, Bedingte Anweisungen usw. benötigt. Den Kindern kann in Form des Roboterspiels eine Vernetzung der symbolhaften mit enaktiven bzw. ikonischen Repräsentationen angeboten werden. Hierzu werden virtuelle Umgebungen aus Swift Playgrounds mit Schachteln nachgebaut. Die Kinder können ihre Lösungen mit den entsprechenden Befehlen in Form von ausgedruckten Befehlskarten auflegen und auf Handlungsebene durch Abschreiten der Befehle ihre Lösungen überprüfen. Die Swift-Umgebung kann in Folge in vermindertem Maßstab durch Bausteine dargestellt werden, in der Spielfiguren nach Auflegen der Befehlskarten ebenso auf Handlungsebene bewegt werden. Im Anschluss können die Programme, die über eigenes Tun überprüft bzw. korrigiert wurden, auf dem Tablet eingegeben werden.

## Conclusio

Erste Interview- und Videoauswertungen von Kindern weisen darauf hin, dass das vorgestellte Konzept ein erstes Programmieren verbunden mit Grundvorstellungen bei Kindern der Primarstufe ermöglichen kann. Mathematisches und informatisches Verständnis für Modellierungen, Problemlöseprozesse, Algorithmisierungen, Automatisierungen, statischer und dynamischer Geometrie, Dekompositionen, Iterationen, Funktionen und Musterkennungen kann im Primarstufenalter generiert werden. Eine möglichst enge Passung zwischen eigenen Handlungen und den mentalen Prozessen hin zu einem Verstehen der mathematischen und informatischen Ideen können eine erfolgreiche Umsetzung ermöglichen.

Eine Möglichkeit eines schrittweisen Aufbaus informatischer Inhalte ist in Abb. 4 dargestellt. Mögliche Repräsentationen und Inhalte, die vermittelt werden können, sind den vorgestellten Medien zugeordnet. Die Zuordnungen wurden nach dem vorgestellten didaktischen Konzept so gewählt, dass eine erfolgreiche Umsetzung gewährleistet werden kann.

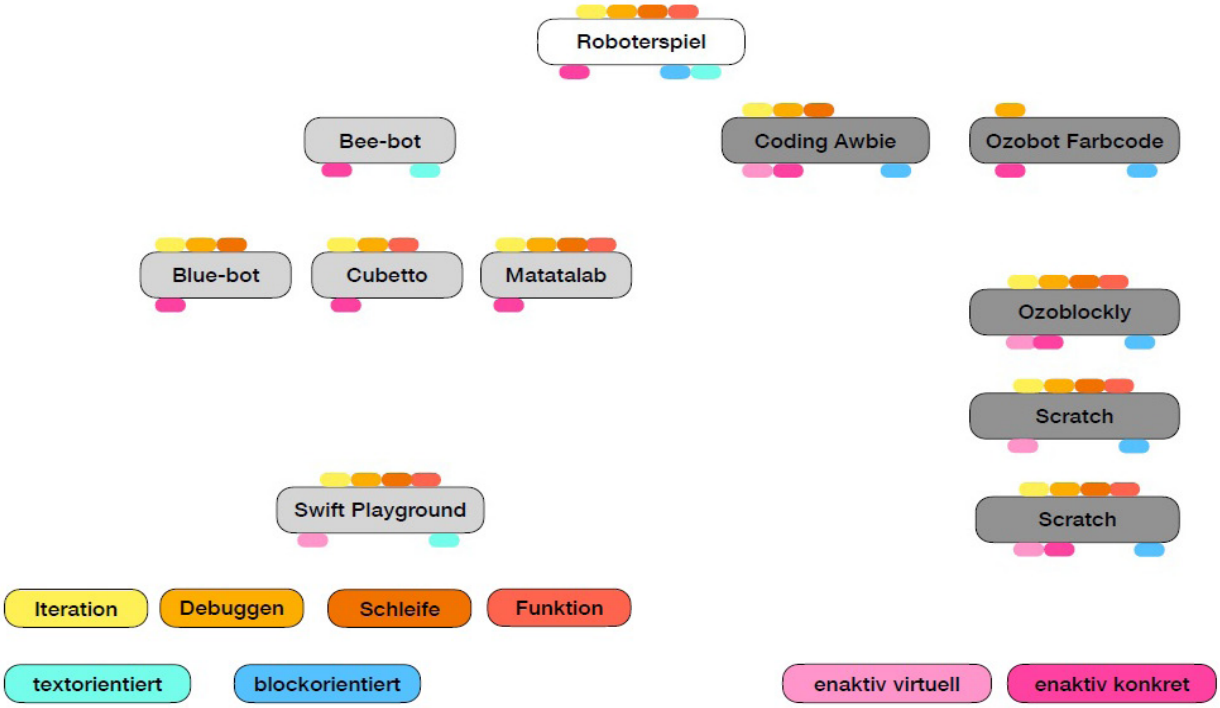


*Abb. 4: Übersichtstafel von Repräsentation und Inhalten zum Programmierverständnis*

## Literatur


Aho, A. V. (2012). Computation and computational thinking. The Computer Journal, 55(7), 832-835.

Antonitsch, P., Pasterk, S., & Sabitzer, B. (2014). Informatics Concepts For Primary Education: Preparing Children For Computational Thinking. In C. Schulte (Hrsg.), WiPSCE 2014: proceedings of the 9th Workshop in Primary and Secondary Computing Education, November 5-7, 2014, Berlin, Germany. Hg. v. Michael E. Caspersen, Carsten Schulte und Judith GalEzer. [Place of publication not identified]: ACM, S. 108–111.

Barberi, A., Berger, C. & Himpsl-Gutermann, K. (2017). Editorial 2/2017: Digitale Grundbildung. Medienimpulse, 55(2).

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers, 13(1), 20-29.

BMBWF (2018). Verordnung der Bundesministerin für Bildung, mit der die Verordnung über die Lehrpläne der Neuen Mittelschulen sowie die Verordnung über die Lehrpläne der allgemeinbildenden höheren Schulen geändert werden. Online verfügbar unter: https://www.ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2018_II_71/BGBLA_2018_II_71.html

BMUKK (2012). Lehrplan der Volksschule. Online verfügbar unter: https://www.bmbwf.gv.at/dam/jcr:b89e56f6-7e9d-466d-9747fa739d2d15e8/lp_vs_gesamt_14055.pdf

Bower, M. (2008). Teaching and Learning Computing. Online verfügbar unter https://www.researchgate.net/publication/228384558_Teaching_and_Learning_Computing.

Bruner, J. (1971). Studien zur Kognitiven Entwicklung. Klett.

Catlin, D., Kandlhofer, M., & Holmquist, S. (2018). EduRobot taxonomy: A provisional schema for classifying educational robots. Paper presented at the International Conference on Robotics in Education 2018.

Curzon, P. & P. W. McOwan (2018). Computational Thinking. Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln. Springer.

Denning, P. (2017a). Computational Thinking in Science. Am. Sci., 105(1), 13.

Denning, P. (2017b). Remaining trouble spots with computational thinking. Commun. ACM, 60(6), 33–39.

Denning, P., & Tedre, M. (2019). Computational thinking. The MIT Press.

digi.komp4 (2016). BMBWF. https://digikomp.at/index.php?id=542&L=0

Döbeli Honegger, B. (2017). Mehr als 0 und 1. Schule in einer digitalisierten Welt. 2., durchgesehene Auflage. hep der Bildungsverlag.

Gallenbacher, J. (2018). Geleitwort. In P. W. McOwan (Hrsg.), Computational Thinking: Die Welt des algorithmischen Denkens – in Spielen, Zaubertricks und Rätseln. Springer.

Guzdial, M. (2008). Education Paving the way for computational thinking. Communications of the ACM, 51(8), 25-27.

Himpsl-Gutermann, K., Brandhofer, G., Frick, K., Fikisz, W., Steiner, M., Bachinger, A., Gawin, A., Szepannek, P & Lechner, I. (2018). Abschlussbericht im Projekt "Denken lernen – Probleme lösen (DLPL) Primarstufe".

Kalbitz, M., Voss, H. & Schulte, C. (2011). Informatik begreifen – Zur Nutzung von Veranschaulichungen im Informatikunterricht. In M. Thomas (Hrsg.), Informatik in Bildung und Beruf. 14. GI-Fachtagung 'Informatik und Schule – INFOS 2011', Münster, 12.-15.09.2011 (S. 137–146). Gesellschaft für Informatik.

Krauthausen, G., Scherer, P. & Scherer, P. (2017). Natürliche Differenzierung im Mathematikunterricht: Konzepte und Praxisbeispiele aus der Grundschule. Kallmeyer/Klett.

Niederösterreichische Landeskorrespondenz, P. (2018). Bienen-Roboter für alle NÖ Landeskindergärten und NÖ Volksschulen.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

Papert, S., & Harel, I. (1991). Situating constructionism. Constructionism, 36(2), 1-11.

Piaget, J. & Szeminska, A. (1975). Die Entwicklung des Zahlbegriffs beim Kinde. KlettCotta.

Savard, A., & Highfield, K. (2015). Teachers' Talk about Robotics: Where Is the Mathematics? Mathematics Education Research Group of Australasia.

Schubert, S. & Schwill, A. (2011). Didaktik der Informatik. 2. Aufl. Spektrum Akademischer Verlag.

Schwill, A. (2001). Ab wann kann man mit Kindern Informatik machen. INFOS2001-9. GIFachtagung Informatik und SchuleGI-Edition, 13-30.

Tedre, M. & Denning, P. (2016): The long quest for computational thinking. In J. Sheard & C. Suero Montero (Hrsg.), Proceedings of the 16th Koli Calling International Conference on Computing Education Research – Koli Calling '16. the 16th Koli Calling International Conference. Koli, Finland, 24.11.2016 – 27.11.2016 (S. 120–129). ACM Press.

Vom Hofe, R. (1995). Grundvorstellungen mathematischer Inhalte. Spektrum Akad. Verlag.

Wagner, W. (2019). Entwickeln von Grundvorstellungen des Programmierens bei Kindern zum Aufbau einer digitalen Literarität. merz spektrum, 01, 70-77.

Wagner, W. (2018). Programmieren mit Programmbausteinen [Video]. YouTube: https://www.youtube.com/watch?v=xl0y0cr-aDI

Walter, D. (2017). Nutzungsweisen bei der Verwendung von Tablet-Apps: Eine Untersuchung bei zählend rechnenden Lernenden zu Beginn des zweiten Schuljahres (Vol. 31). Springer.

Wartha, S. (2011). Handeln und verstehen. Förderbaustein: Grundvorstellungen aufbauen. mathematik lehren, 66.

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.