Algorithmen mit Computer-Aided Design erkunden – Ideen für den Mathematikunterricht

Frederik Dilling – Gregor Milicic – Amelie Vogler

In diesem Beitrag wird das Programmieren im Kontext von 3D-Druck thematisiert. Im Fokus steht dabei die Blockprogrammierungsfunktion der Computer-Aided Design Software BlocksCAD, welche die Formulierung von Algorithmen zur Entwicklung von maßstäblichen dreidimensionalen Objekten durch eine Anordnung von Codeblöcken als grafische Elemente ermöglicht. Dies lässt verschiedene Anwendungsszenarien im Bereich der Mathematik zu, von denen drei als konkrete Beispiele in diesem Artikel diskutiert werden.

1 Einleitung

Der Algorithmus gilt als fundamentale Idee der Mathematik und spielt demzufolge auch eine wichtige Rolle im Mathematikunterricht. Der Umgang mit Programmierungen bietet eine gute Möglichkeit, Algorithmen an der Schnittstelle zur Informatik im Mathematikunterricht zu thematisieren. Im Beitrag wird am Beispiel der CAD-Blockprogrammierung gezeigt, wie interdisziplinäre Problemlöseprozesse sowohl mathematische Inhalte und Kompetenzen als auch Grundlagen der Informatik

ansprechen können. Dazu werden drei konkrete Beispiele für mathematikhaltige Algorithmen vorgestellt und verschiedene Aufgabentypen zur Thematisierung von Algorithmen in differenzierenden Lernsettings beschrieben.

2 Problemlösen und Algorithmen im Mathematik- und Informatikunterricht

Das Identifizieren und Lösen von Problemen stellt eine typische Aktivität in der Mathematik und der Informatik dar und hat auch im Unterricht der beiden Fächer einen festen Platz. Die Definition eines Problems erfolgt dabei meist als die Transformation von einem Anfangszustand in einen Endzustand, wobei dem problemlösenden Individuum für den Übergang kein direktes Verfahren bekannt ist (Dörner, 1979; Newell & Simon, 1972). Stattdessen müssen eigenes Wissen und Strategien verknüpft werden, um eine Lösung zu finden. Schoenfeld (1985) fasst dies mit Blick auf die Mathematik so zusammen: „*The problem solver does not have easy access to a procedure for solving a problem - a state of affairs that would make the task an exercise rather than a problem - but does have an adequate background with which to make progress on it* […]." (Schoenfeld, 1985, 11)

Somit ist nicht die Komplexität einer Aufgabe entscheidend, sondern vielmehr ob direkte Lösungsverfahren bekannt sind oder nicht (Smith, 1991). Ob eine Aufgabe als Problem bezeichnet wird, hängt damit vom problemlösenden Individuum bzw. spezieller von dem verfügbaren Wissen ab.

Der Begriff des Problems ist damit zunächst nicht auf eine bestimmte Fachdisziplin beschränkt und bezieht bei der Problemlösung Wissen und Strategien aus verschiedenen Bereichen mit ein. In diesem Beitrag liegt der Fokus auf dem Mathematikunterricht, und es sollen mathematikhaltige Probleme betrachtet werden, das heißt solche, die bei der Problemlösung wesentlich von mathematischem Wissen und Strategien Gebrauch machen (Dilling, 2020).

Das mathematische Problemlösen von Schüler/inne/n lässt sich vereinfacht mithilfe der bekannten Stufen des Problemlöseprozesses nach Pólya (1949) beschreiben:

1. Verstehen der Aufgabe
2. Ausdenken eines Plans
3. Ausführen des Plans
4. Prüfen der Lösung

Die Schritte lassen sich auch als Kreislauf auffassen, in dem nach der Rückschau eine Verbesserung der gefundenen Lösungen durch erneutes Durchführen der Problemlöseschritte stattfinden kann. Die Stufen können auch explizit mit den Lernenden besprochen werden, um sie als allgemeine Hilfestellung und zur Strukturierung des eigenen Lösungsprozesses heranzuziehen. Sie sollten allerdings nicht normativ verwendet werden – Schüler/innen können ebenso auf andere Weise vorgehen und Lösungen für Probleme finden.

Auch in der Informatik stellt das Problemlösen eine charakteristische Tätigkeit dar, mit dem Ziel, einen Algorithmus zu entwickeln und in einem Programm umzusetzen: „*Das Ziel einer Problemlösung ist, ausgehend von einer Problemstellung ein System zu finden, das diese Lösung darstellt. Diese Vorgehensweise lässt sich in drei Schritten aufgliedern. Ausgangspunkt ist eine gegebene Problemstellung, für die eine Lösung durch den Rechner gesucht wird. Nachfolgend wird mittels Abstraktion aus dem Problem im Idealfall ein Algorithmus erstellt. Der Algorithmus ist im nächsten Schritt in ein Programm zu transferieren, welches dann auf einem Rechner ausgeführt wird. Das Programm stellt die Lösung des Problems dar.*" (Müller & Weichert, 2013, 16)
Der informatische Problemlöseprozess (Abb. 1) lässt sich ebenfalls als eine Abfolge bestimmter typischer Schritte beschreiben (Müller & Weichert, 2013):

1. Formulieren eines Problems
2. Analysieren des Problems
3. Entwerfen eines Algorithmus
4. Überprüfen der Korrektheit des Algorithmus
5. Analysieren des Implementationsaufwands
6. Programmieren

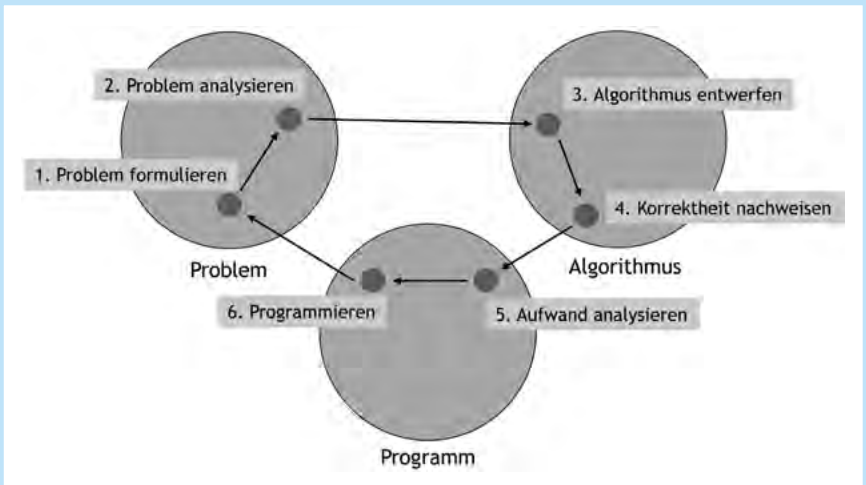

Abb. 1. Informatischer Problemlöseprozess in Anlehnung an Müller & Weichert (2013)

Auch der informatische Problemlöseprozess ist als Kreislauf zu betrachten, bei dem je nach Ergebnis die Verbesserung des Algorithmus und der Programmierung in weiteren Problemlösezyklen erfolgt. Der informatische Problemlöseprozess muss im Unterricht nicht immer vollständig durchlaufen werden. Beispielsweise kann lediglich ein Algorithmus entwickelt, nicht aber in einem Programm umgesetzt werden oder es wird ein Algorithmus vorgegeben, der mit einer bestimmten Programmierumgebung umgesetzt werden soll. Diese verschiedenen Aufgabentypen werden am Ende des Beitrags detaillierter betrachtet.

MNU-Journal – Ausgabe 01.2022 – ISSN 0025-5866 – © Verlag Klaus Seeberger, Neuss

In diesem Beitrag werden drei Problemstellungen vorgestellt, bei denen sich im Mathematikunterricht spezifische Algorithmen entwickeln und mit einer Blockprogrammiersprache umsetzen lassen. Das Programmieren bzw. die Bearbeitung mathematikhaltiger Probleme im Kontext von Programmierungen kann zu einer tiefen Auseinandersetzung mit den mathematischen Inhalten führen, die auch Beckmann (2003) betont: *„Wird Programmieren […] zur Lösung mathematischer Probleme eingesetzt, ist zu erwarten, dass damit auch ein Mathematiklernen einher geht. Denn Programmieren erfordert immer auch die intensive Auseinandersetzung mit dem Thema, hier also mit der Mathematik. Darüber hinaus kann Programmieren aber auch gezielt eingesetzt werden, um bestimmte mathematische Inhalte oder Methoden zu lernen und zu vertiefen.“* (Beckmann, 2003, 16)

Der Algorithmus bildet im informatischen Problemlöseprozess die Grundlage für den eigentlichen Programmierprozess (Abb. 1). Im Mathematikunterricht werden Algorithmen klassischerweise anhand mathematischer Algorithmen wie z. B. der schriftlichen Rechenverfahren in der Grundschule oder des euklidischen Algorithmus in der Sekundarstufe thematisiert und damit eher isoliert betrachtet. Dass der Algorithmus sowohl für die Mathematik als auch die Informatik als fundamentale Idee beschrieben wird (Winter, 2001; Schubert & Schwill, 2011), bestätigt jedoch die Relevanz der Auseinandersetzung mit Algorithmen im MINT-Unterricht. Gerade in Verbindung mit neuen Technologien eröffnen sich viele Anwendungsfelder mit Algorithmen im fächerübergreifenden Unterricht (Milicic, 2020). Auch unser Alltag ist mittlerweile von Algorithmen durchdrungen und stellenweise sogar bestimmt. Grundlegendes Wissen im Bereich der Algorithmen erscheint daher wichtiger und aktueller denn je und sollte dementsprechend auch von den Schüler/inne/n erlernt werden.

In diesem Beitrag werden verschiedene Einsatzszenarien der Blockprogrammierung vorgestellt, welche sich zur Untersuchung von Algorithmen und zur Förderung algorithmischen Denkens eignen. Nach Förster (2018) bewährt sich das Programmieren insbesondere, um das Überprüfen auf Korrektheit und das formale Präzisieren anzuregen. Dass das algorithmische Denken nicht nur auf das Verstehen und Anwenden von Algorithmen begrenzt sein sollte, betonen Kortenkamp & Lambert (2018, 2): *„Logisches, strukturierendes und formalisierendes Denken – und damit speziell auch algorithmisches Denken und Algorithmen – im Alltag und in der Mathematik zu entdecken, selbst zu entwickeln und zu verstehen, anzuwenden und zu reflektieren, ist ein allgemeinbildendes Ziel von Mathematikunterricht.“*

Nach Ladel (2020) wird das algorithmische Denken vor allem dann von Schüler/inne/n benötigt, wenn sie eine (Problem-) Situation formalisieren und das Problem verarbeiten. Falls das betrachtete Problem wiederholt gelöst werden muss, kann ein formalisierter Lösungsprozess die Bearbeitung erleichtern. Damit ermöglicht er die Fokussierung auf andere Aspekte des Problemlöseprozesses und erlaubt den Transfer der nunmehr monotonen Abarbeitung der Handlungsschritte auf eine andere Entität (Maschine oder Mensch). Die Ausführung des Plans kann anschließend automatisiert erfolgen. Bezugnehmend auf den beschriebenen informatischen Problemlöseprozess (Abb. 1) eignet sich dafür insbesondere ein Algorithmus als formalisierte Abfolge von Handlungsvorschriften. Die bewusste Thematisierung, Programmierung und Nutzung des Algorithmus ersetzt damit das kalkülhafte Abarbeiten der einzelnen Bearbeitungsschritte. Über die mögliche Generalisierung und Anwendbarkeit des Algorithmus auf andere Fragestellungen lässt sich zudem die vertiefte Analyse des Problems mit dem Ziel der Kategorisierung als einer der zentralen Prozesse innerhalb der Mathematik motivieren.

Das Prüfen und Reflektieren sind für das Optimieren eines Algorithmus von besonderer Relevanz. In den nachfolgend beschriebenen Programmierumgebungen entwickeln Schüler/innen 3D-Modelle auf Grundlage von Algorithmen, dabei verwenden sie die Blockprogrammierung, welche durch ihre besonders anschauliche Art einen geeigneten Einstieg in das Programmieren ohne einen erhöhten Formalisierungsgrad ermöglicht. Ein weiterer Vorteil dieser Art der Programmierung ist, dass sie Schüler/inne/n im Mathematikunterricht eine gemeinsame Sprache bieten kann, um über (eigens) entwickelte Programme und insbesondere die Struktur und die Hintergrundidee bzw. den zugrundeliegenden Algorithmus zu sprechen (Förster, 2011).

3 Computer-Aided Design durch Blockprogrammierung

Die 3D-Druck-Technologie erfreut sich im Bildungsbereich und auch speziell im Mathematikunterricht immer größerer Beliebtheit. In den letzten Jahren wurde eine Vielzahl verschiedener konkreter Anwendungsszenarien und Unterrichtsmaterialien auf Basis des 3D-Drucks entwickelt (Dilling, Marx, Pielsticker, Vogler & Witzke, 2021; Witzke & Heizer, 2019; Dilling & Witzke, 2020) sowie Forschungsergebnisse zu mathematischen Lehr-Lernprozessen mit dieser Technologie erzielt (Dilling, 2019, Pielsticker, 2020).

Insbesondere die individuelle Anwendung von Computer-Aided-Design-Software durch die Schüler/innen im Unterricht bietet viele Möglichkeiten für den Erwerb mathematischer Kompetenzen. In diesem Beitrag wird der Fokus, wie bereits im vorherigen Abschnitt beschrieben, auf dem Umgang mit mathematischen Algorithmen und dem Problemlösen liegen. Hierzu eigenen sich insbesondere skriptbasierte CAD-Modellierungsprogramme wie BlocksCAD (*https://www.blockscad3d.com/editor/*) oder die Programmierumgebung von Tinkercad (*https://www.tinkercad.com*).

Mit dem CAD-Modellierungsprogramm BlocksCAD können Schüler/innen im MINT-Unterricht auf kreative Weise erste Erfahrungen im Bereich Programmieren sammeln. Die aus der Informatik bekannte Programmiersprache ist hier in sogenannte „Blöcke“ verpackt, welche im Skriptbereich wie Puzzlesteine zu einem Code zusammengesetzt werden (Abb. 2). Daher steht bei dieser Art der Programmierung (auch Blockprogrammierung

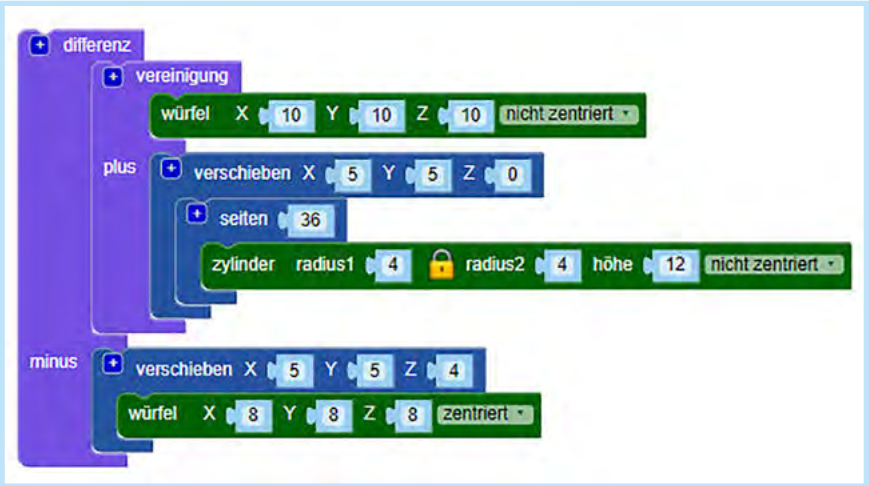

Abb. 2. Puzzleartige Anordnung eines Beispiel-Codes in BlocksCAD

genannt) nicht die Syntax im Vordergrund, wie z. B. bei Java oder OpenSCAD, sondern das inhaltliche Problem, für welches ein passendes Programm zur Problemlösung entwickelt werden soll. Zugleich ist es innerhalb von BlocksCAD möglich, zwischen der Block- und der textuellen Programmierung zu wechseln. Daher kann BlocksCAD insbesondere auch dazu genutzt werden, den Übergang von Block- hin zur textuellen Programmierung anschaulich und mittels bekannter Problemstellungen und Algorithmen zu gestalten.

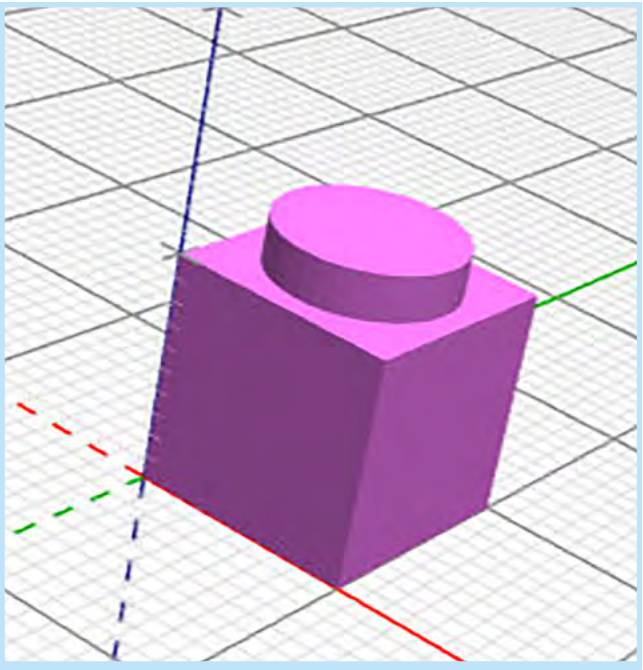

Abb. 3. Generiertes 3D-Modell zu Code aus Abbildung 2 in BlocksCAD

Mit BlocksCAD können 3D-Modelle programmiert werden. Durch die Funktion „Rendern“ wird der im Skriptbereich entwickelte Code in ein entsprechendes 3D-Modell übersetzt, welches dann rechts neben dem Skriptbereich auf der virtuellen Arbeitsebene angezeigt wird. In den Abbildungen 2 und 3 ist zu erkennen, wie ein einfacher Baustein u. a. mittels der Mengenoperationen „Vereinigung“ und „Differenz“ aus zwei Würfeln und einem Zylinder erstellt werden kann. Das durch das Programm entwickelte 3D-Objekt kann als STL-Datei heruntergeladen und dann über eine sogenannte Slicer-Software für den 3D-Drucker vorbereitet werden.

Diese Übersicht (Abb. 4) zeigt verschiedene Blöcke, welche durch ihre Farbe und Form besonders anschaulich in verschiedene Befehlskategorien eingeordnet sind. Wir beschränken uns im Folgenden auf die Vorstellung der bei den einzelnen Beispielen genutzten Blöcke.

Zunächst gibt es die 3D-Formen (dunkelgrün), wie Kugel, Würfel und Zylinder. Die ungenaue Bezeichnung der 3D-Formen bei BlocksCAD kann als Anlass zur Exaktifizierung der geometrischen Körper seitens der Lehrkraft genutzt werden. Durch entsprechende Variation der Variablen kann mittels des Würfelblocks ein Quader und mittels des Zylinderblocks ein Pyramiden- bzw. Kegelstumpf erzeugt werden. Transformationen in der Ebene und im Raum (dunkelblau), wie das Verschieben, Rotieren, Spiegeln und Skalieren, aber z. B. auch Extrudieren von ebenen Figuren, bilden dabei neben den Mengenoperatoren (helles lila) wichtige Befehle ab. Die Kategorie der Schleifen (dunkles lila) beinhaltet den für die Informatik typischen Schleifen-Befehl. Die Kategorie Mathematik (hellblau) verfügt über zahlreiche mathematische Größen und Operationen, wie z. B. die Grundrechenoperationen, das Wurzelziehen oder auch das Generieren einer Zufallszahl. Als letztes ist noch die Kategorie Variablen (braun) zu nennen, welche das Setzen und Ändern von Variablen beinhaltet. Die äußerliche, puzzleartige Form der Blöcke gibt vor, wie diese zusammengesetzt werden können, und verhindert daher Fehler auf Ebene der Syntax. Weitere logische Fehler können durch die Vorschau des programmierten Modells auf der Arbeitsebene des Programms vom Bediener eingesehen werden. Schüler/innen werden somit zu einem eher explorativen Vorgehen angeregt.

Im Folgenden werden drei verschiedene Problemstellungen (Wendeltreppe, Bausteingenerator und Schneckenhaus) und zugehörige in BlocksCAD entwickelte Programme als mögliche Problemlösungen vorgestellt. Zudem wird das Potenzial dieser Programmierumgebungen für den Einsatz im Mathematikunterricht beschrieben. Die Blockprogrammierung bietet zur Auseinandersetzung mit Algorithmen im MINT-Unterricht allerdings noch weitaus mehr Einsatzszenarien, von denen beispielhaft einige in Kasten 1 aufgeführt sind. Bereits in der Primarstufe können Würfelgebäude nach Plan generiert oder verschiedene Parkettierungsprinzipien erkundet werden. Darüber können in der Sekundarstufe regelmäßige Prismen und Sternkörper sowie Fraktale erzeugt und deren Eigenschaften untersucht werden. Zahnräder sowie Windrad-Modelle könnten im Rahmen von umfangreicheren fächerübergreifenden Lernumgebungen entwickelt, gedruckt und aus mathematischer sowie physikalischer Perspektive untersucht und anschließend auch mittels informatischer Kompetenzen optimiert werden.

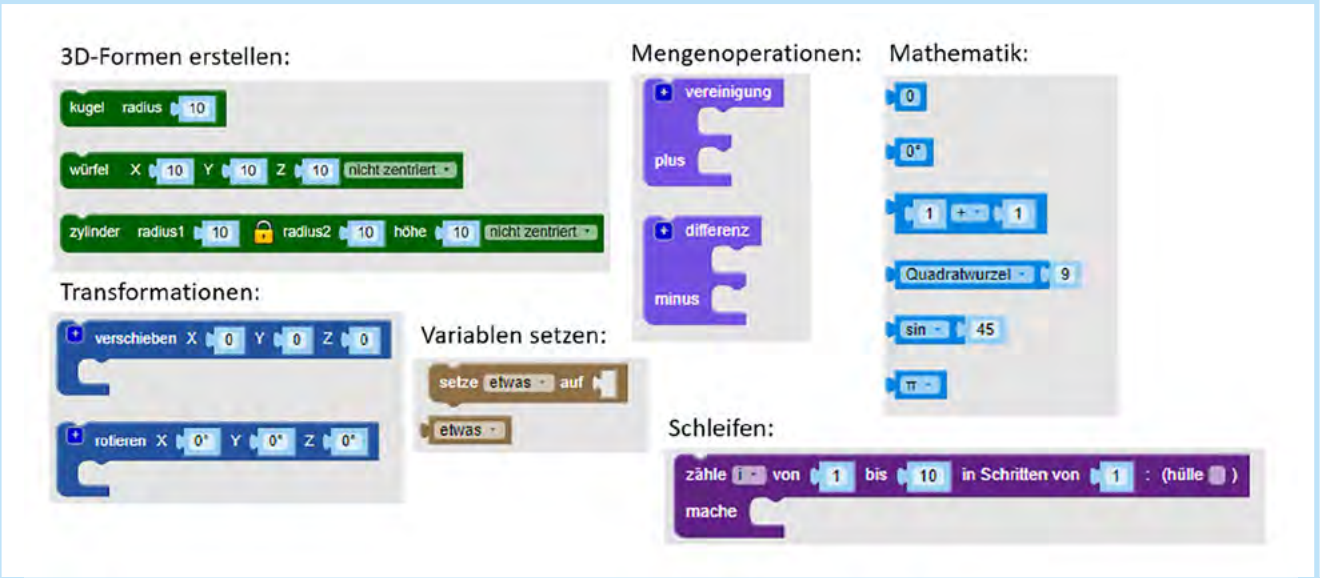

Abb. 4. Übersicht ausgewählter Blöcke aus BlocksCAD

Zur Bearbeitung der nachfolgenden drei Problemstellungen sollten die Schüler/innen bereits vorab mit einer blockbasierten Programmierung (z. B. Scratch) grundlegend vertraut sein und beispielsweise mit Anordnung der Blöcke zu einem Code umgehen können. Insbesondere differenzierte Kenntnisse bezüglich der Verwendung von Schleifen und Variablen ist für die Bearbeitung der Problemstellungen zuträglich, jedoch für die Beispiele 4.1 und 4.2 nicht zwingend notwendig.

Weitere Ideen für CAD-Modellierungen mit BlocksCAD

- Würfelgebäude
- Regelmäßige Prismen
- Sternkörper
- Parkettierungen
- Fraktale
- Zahnräder
- Windrad

Kasten 1. Weitere Ideen für CAD-Modellierungen mit BlocksCAD

Abb. 5. Code für eine Wendeltreppe in BlocksCAD

4 Beispielprobleme

4.1 Die Wendeltreppe

Ein erstes Anwendungsbeispiel stellt die Entwicklung eines Programms für eine Wendeltreppe dar. Die ausgedruckte Treppe könnte bei maßstabsgetreuen Modellen für Bauwerke oder Häuser eingesetzt werden und bietet sowohl aus mathematischer als auch aus informatischer Sicht verschiedene Möglichkeiten zur vertieften Auseinandersetzung (Milicic, 2020). Zur Bearbeitung der Problemstellung (Kasten 3) sind grundlegende mathematische und informatische Kenntnisse ausreichend, so dass sich die Aufgabe sehr gut zum Einstieg und Erlernen der BlocksCAD-Umgebung in der Sekundarstufe I eignet. Eine erste Wendeltreppe kann durch die wiederholte Erstellung und Verwendung von Quadern, sowie deren Rotation und Translation realisiert werden (Kasten 2). Eine entsprechende Umsetzung in BlocksCAD ist in den Abbildungen 5 und 6 dargestellt.

Die Verwendung einer Schleife ist ein wesentliches Element dieser Problemstellung. Die manuelle Verschiebung der einzelnen Stufen führt bei einer etwaigen Überarbeitung des Modells zu einem erhöhten Mehraufwand. Die Erzeugung der Quader als Stufenelemente mit anschließender Transformation sollte daher automatisiert innerhalb einer Schleife erfolgen. Das resultierende Modell ist in Abbildung 6 dargestellt.

Die Anzahl der Stufen sowie der Drehwinkel können variiert werden um eine möglichst gut begehbare Wendeltreppe zu erhalten. Der in Kasten 2 notierte Algorithmus kann als Teilschritt im Bearbeitungsprozess von den Schüler/inne/n erstellt oder bei einer gemeinsamen Reflexion von verschiedenen Lösungen thematisiert werden.

Notation des Algorithmus zur Wendeltreppe

(1) Erstelle eine Stufe mit Höhe *h*
(2) Vervielfache (1) *i*-mal und drehe um *ix* Grad
(3) Verschiebe (2) um *ih* in *z*-Richtung

Kasten 2. Notation des Algorithmus zur Wendeltreppe

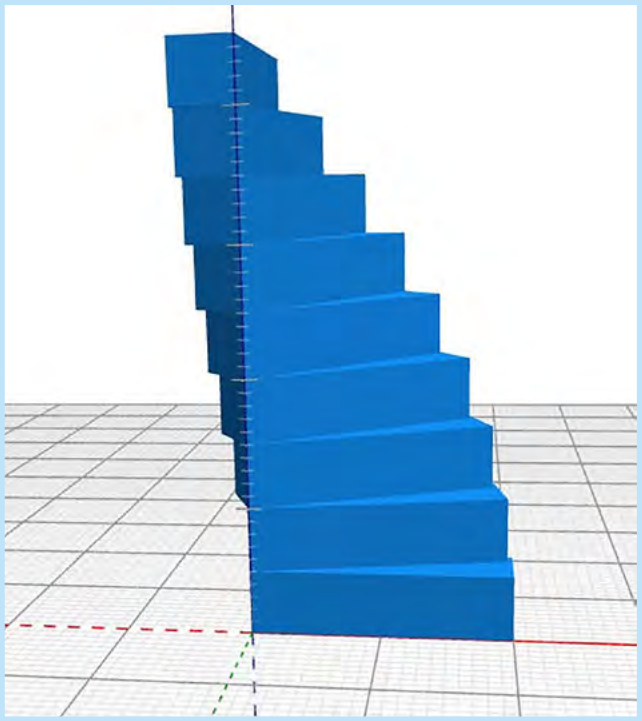

Abb. 6. Generierte Wendeltreppe in BlocksCAD

Einem mathematischen Modellierungskreislauf folgend kann der Entwurf iterativ verbessert werden. Abgerundete Stufen, wie sie bei generischen Wendeltreppen vorhanden sind, können durch die Nutzung von Kreissegmenten umgesetzt werden. Um diese Grundform zu erhalten, kann die Schnittmenge zwischen einem Quader und einem Zylinder gebildet werden. Zur Vergrößerung der Trittfläche können die Stufenbreite, d. h. die Tiefe der Quader, und der Abstand der Stufen zueinander, also der Drehwinkel, variiert werden, um ein leichter begehbares Modell zu erhalten.

Aus informatischer Sicht ist die Nutzung der Zählvariablen innerhalb der Schleife zur Platzierung der Quader für Schüler/innen mit geringen Programmierkenntnissen eine potentielle Herausforderung. Hier kann vor der Implementierung des Programms in BlocksCAD die Besprechung des in Kasten 2 gegebenen Algorithmus den Schüler/inne/n bei der Umsetzung helfen. In einer anschließenden Reflexionsphase kann insbesondere auf die Verwendung der Zählvariable innerhalb der Schleife bei diesem Algorithmus eingegangen werden. Des Weiteren kann zur Differenzierung der verschiedenen Leistungsniveaus die Problemstellung in Kasten 3 auch abgeändert werden. So können z. B. die einzelnen Codeblöcke in BlocksCAD zufällig verteilt im Skriptbereich vorgegeben werden. Weitere Variationsmöglichkeiten sind im Abschnitt 5 „Aufgabenstellungen mit Algorithmen variieren“ beschrieben. Der Algorithmus kann anschließend unter Nutzung von Variablen für die Anzahl der gewünschten Stufen und des jeweiligen Drehwinkels adaptiert und damit auch für weitere Anwendungen generalisiert werden.

Problemstellung – Wendeltreppe

Entwickle ein Programm, welches eine Wendeltreppe erzeugt. Gehe wie folgt vor:

a) Plane dein Vorgehen, indem du, bevor du mit der Programmierung beginnst, zunächst eine Skizze einer möglichen Wendeltreppe anfertigst. Wie sieht ein vereinfachtes Modell einer Wendeltreppe aus und welche Eigenschaften sind bei der Modellierung entscheidend?
b) Nutze zur Erstellung der Wendeltreppe eine Schleife. Welches Element tritt wiederholt auf? Wie muss das entsprechende Elemente wiederholt transformiert werden, um eine Wendeltreppe zu erhalten?
c) Entwickle nun in BlocksCAD ein Programm für eine Wendeltreppe.
d) Drucke die von dir programmierte Wendeltreppe mit dem 3D-Drucker aus.
e) Betrachte das ausgedruckte Exemplar. Notiere dir weitere Anpassungen, um eine möglichst gut begehbare Wendeltreppe zu erhalten. An welchen Stellen müsstest du dein Programm dafür anpassen?
f) Überlege nun, wie du dein Programm, zum Beispiel durch den Einsatz von Variablen, noch weiter optimieren kannst.

Kasten 3: Problemstellung Wendeltreppe

Problemstellung – Bausteingenerator

Entwickle ein Programm, welches durch die Eingabe verschiedener Größen einen entsprechenden Baustein erzeugt. Gehe wie folgt vor:

a) Plane dein Vorgehen, indem du, bevor du mit der Programmierung beginnst, beispielsweise zunächst eine Skizze eines möglichen Grundbausteines mit entsprechenden Größenangaben anfertigst.
b) Entwickle nun in BlocksCAD ein Programm, welches deinen Grundbaustein erzeugt.
c) Überlege nun, wie du Programm zu einem echten Bausteingenerator weiterentwickeln kannst, welcher verschiedene Bausteine erzeugen kann.
d) Überlege, ob dein Programm Bausteine erzeugt, welche sich zusammenstecken lassen.
e) Tipp: Verwende den Block „Differenz“, um Aussparungen für ein Stecksystem an deinem Baustein zu erzeugen.
f) Teste dein Programm und drucke zwei verschiedene Bausteine mit dem 3D-Drucker aus.

Kasten 4. Problemstellung Bausteingenerator

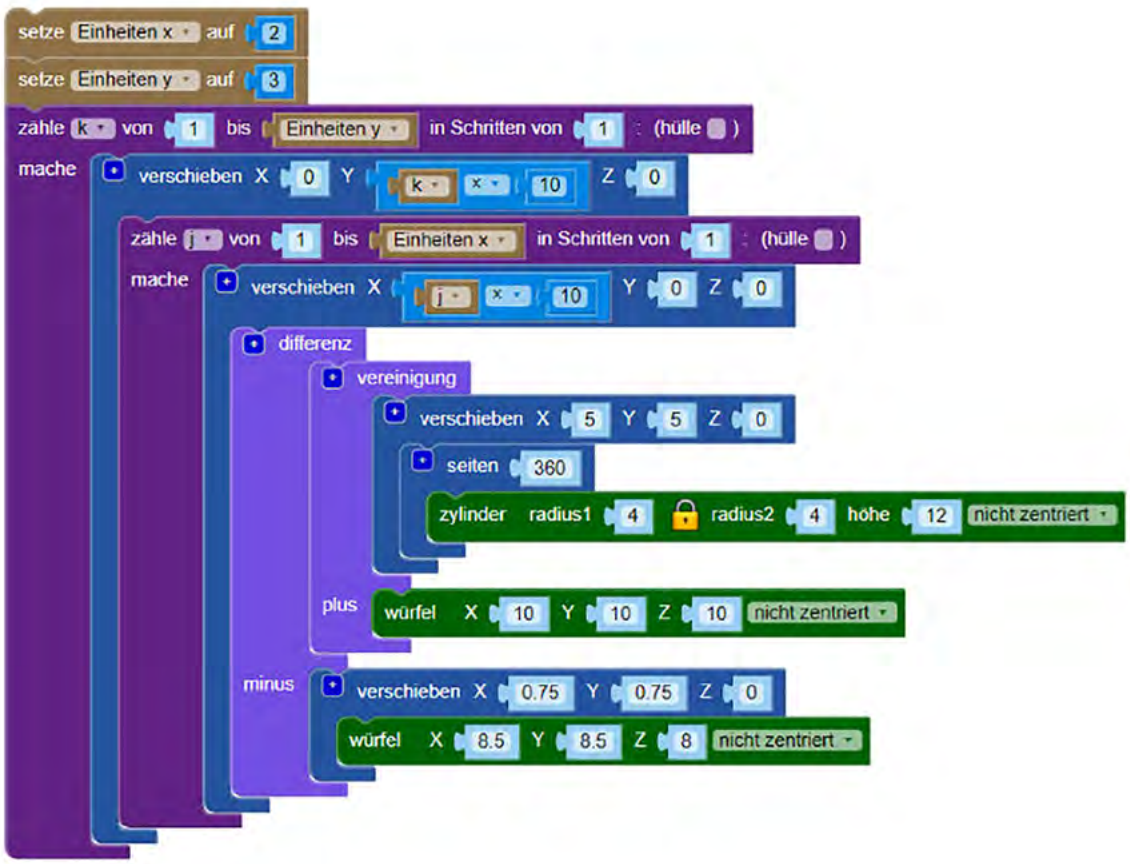

Abb. 7. Code für einen Bausteingenerator in BlocksCAD

4.2 Der Bausteingenerator

Die Entwicklung eines Programms zur Generierung eines Bausteins bietet in der Sekundarstufe 1 eine gute Möglichkeit, weitere Erfahrungen im Umgang mit Blockprogrammierung sowie CAD-Modellierung zu sammeln. Einbinden lässt sich die Problemstellung zur Entwicklung eines Bausteingenerators (Kasten 4) einerseits im Bereich der Geometrie bei der Betrachtung zusammengesetzter Körper, andererseits werden durch die Entwicklung auch Fähigkeiten zur Orientierung im Raum und das algebraische Denken, hier insbesondere der verständnisvolle Umgang mit Variablen, gefördert und gefordert.

Eine mögliche Lösung der Problemstellung im CAD-Modellierungsprogramm BlocksCAD ist in den Abbildungen 7 und 8 dargestellt. Der dem hier dargestellten Programm zugrundeliegende Algorithmus basiert auf einem Grundbaustein, ähnlich dem „Einer-Baustein“ in Abbildung 2, welcher durch die Eingabe verschiedener Werte für die Variablen „Einheiten *x*“ und „Einheiten *y*“ einen entsprechenden Baustein aus $x \cdot y$ aneinandergereihten Grundbausteinen generiert.

In Abbildung 7 ist zu erkennen, dass die Variable „Einheiten *x*“ auf den Wert 2 und die Variable „Einheiten *y*“ auf den Wert 3 gesetzt wurde. Durch die Vereinigung erzeugt das Programm zunächst einen Grundbaustein aus einem entsprechend auf der $x \cdot y$-Ebene verschobenen Würfel und Zylinder. Der Befehl der Differenz wird verwendet, um eine würfelförmige Aussparung im Baustein zu erhalten, welche durch Rotation der Arbeitsebene sichtbar wird. Diese an die zylinderförmigen Aufsätze der Bausteine angepasste Aussparung ermöglicht das Zusammenstecken der Bausteine.

Falls die Schüler/innen mit den Begriffen Vereinigung und Differenz noch nicht vertraut sind, bedarf es hier einer Erläuterung durch die Lehrkraft. Nun sind noch die beiden Schleifen des Programms zu beachten, welche das Programm einerseits im Sinne der Optimierung erheblich verkürzen und andererseits durch die Eingabe verschiedener Werte das Generieren unterschiedlicher Baustein ermöglicht. Somit entsteht ein „echter“ Bausteingenerators, wie in Teilaufgabe c) (Kasten 4) gefordert. Durch die innere Schleife wird der Grundbaustein *x*-mal in Richtung der *x*-Achse (rote Achse) verschoben. Die äußere Schleife verschiebt die so entstandene Reihe an Grundbausteinen *y*-mal in Richtung der *y*-Achse (grün). Dieser Bausteingenerator erzeugt somit beliebig große, quaderförmige Bausteine. Durch die Funktion des Renderns der Blockdaten wurde hier ein entsprechender „2 · 3“er-Baustein auf der CAD-Programmoberfläche angezeigt (Abb. 8).

Dieser Lösung liegt ein bereits sehr ausdifferenzierter Algorithmus zugrunde, die in Kasten 4 dargestellte Problemstellung ist jedoch durch die im Anforderungsniveau steigenden Teilaufgaben so offen und differenzierend formuliert, dass alle Schü-

ler/innen unabhängig von ihrem Leistungsniveau durch den Entwurf eines beliebig komplexen Grundbausteins mit der Bearbeitung beginnen können. Die Lehrkraft könnte zur Hinführung auf die Problemstellung verschiedene Bausteine, wie Legosteine mitbringen, und die Schüler/innen fragen, welche Art von Grundbaustein sich für das zu entwickelnde Programm eigenen würde. Für einen ersten Entwurf eines Grundbausteins ist der Einsatz von Variablen und Schleifen nicht notwendig, sodass keine erweiterten informatischen Kenntnisse zur Bearbeitung erforderlich sind. Ein planvolles Vorgehen, beispielweise durch ein Skizzieren des Grundbausteines, kann insbesondere Schüler/innen mit Schwierigkeiten im Bereich des räumlichen Vorstellungsvermögens helfen, einen zur Problemlösung geeigneten Algorithmus zu entwerfen.

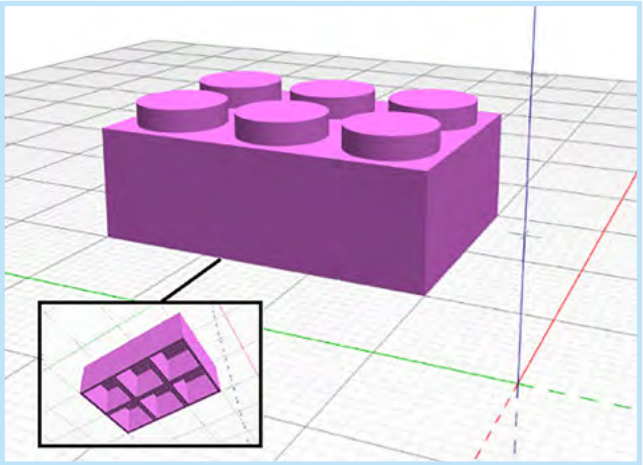

Abb. 8. Generierter Baustein in BlocksCAD

Um die mathematischen sowie informatischen Problemlösefähigkeiten der Schüler/innen zu schulen und mögliche auftretende Probleme im Lösungsprozess zu identifizieren, könnten sie aufgefordert werden ihr Vorgehen als Abfolge von Handlungsschritte in einer allgemeineren Art und Weise zu notieren, wie in Kasten 5 in Form einer nummerierten Liste dargestellt. Als weitere Notationsweise wäre auch ein Ablaufdiagramm möglich. Diese Dokumente in Verbindungen mit Screenshots der zugehörigen selbst entwickelten Programme können den Schüler/inne/n bei der Präsentation und Diskussion ihrer Lösungen als Gesprächsgrundlage dienen.

Mit Hilfe der 3D-Druck-Technologie ist es möglich, die verschiedenen Arbeitsergebnisse der Schüler/innen, in diesem Fall die Bausteine, auszudrucken. Neben dem Betrachten des durch Blockprogrammierung erzeugten 3D-Modells auf der virtuellen CAD-Arbeitsebene des Programms besteht somit die Möglichkeit, das 3D-gedruckte Modell auf physischer Ebene für die Schüler/innen als Produkt ihres Arbeits- und Problemlöseprozesses erfahrbar zu machen.

Notation des Algorithmus zum Bausteingenerator

(1) Erstelle einen quaderförmigen Grundbaustein mit der Grundfläche *xy*
(2) Vervielfache (1) *j*-mal und verschiebe um *jx*
(3) Vervielfache (2) *k*-mal und verschiebe um *ky*

Kasten 5. Algorithmus zum Bausteingenerator

Problemstellung – Schneckenhaus

Entwickle ein Programm, welches ein Schneckenhaus erzeugt. Gehe wie folgt vor:

a) Plane dein Vorgehen, indem du, bevor du mit der Programmierung beginnst, zunächst eine Skizze eines möglichen Schneckenhauses anfertigst. Wie sieht ein vereinfachtes Modell eines Schneckenhauses aus und welche Eigenschaften sind bei der Modellierung entscheidend?
b) Versuche nun die durch das Schneckenhaus gegebene Spirale zu parametrisieren. Wie muss die Gleichung der Kurve aussehen?
c) Welchen Körper musst du mithilfe einer Schleife entlang der Spirale verschieben, damit sich das Schneckenhaus ergibt? Notiere den entstandenen Algorithmus.
d) Setze deinen Algorithmus nun im Programm BlocksCAD um. Verändere gegebenenfalls durch gezieltes Ausprobieren die Parameter der Kurve und des verschobenen Körpers so, dass ein Schneckenhaus entsteht.
e) Teste dein Programm und drucke ein Schneckenhaus mit dem 3D-Drucker aus.
f) Überlege, wie ließe sich der Code noch verändern? Vergleiche deinen Code auch mit dem deiner Mitschüler.

Kasten 6. Problemstellung Schneckenhaus

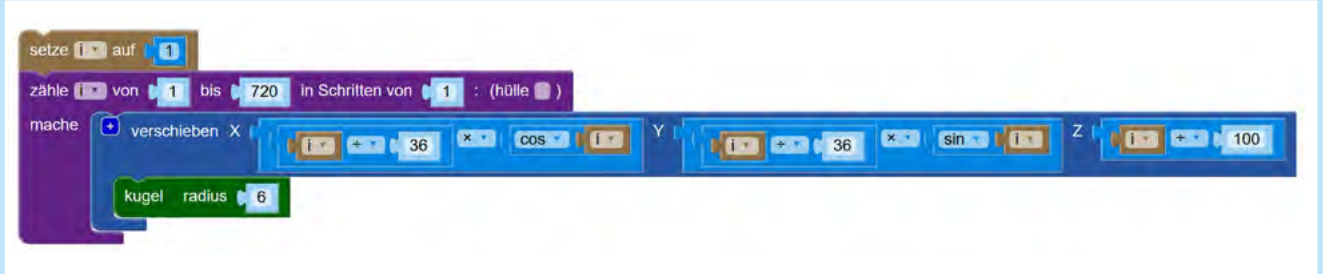

Abb. 9. Code für ein Schneckenhaus in BlocksCAD

Insbesondere das Stecksystem der Bausteine kann somit durch die Schüler/innen überprüft werden, anschließend können die entsprechenden Teilschritte im Algorithmus reflektiert und gegebenenfalls angepasst werden.

4.3 Das Schneckenhaus

Ein weiteres Beispiel für die Anwendung skriptbasierter CAD-Modellierung im Mathematikunterricht bieten Schneckenhäuser. Durch die Entwicklung eines Codes zur Generierung eines Schneckenhauses können Schüler/innen ihr Wissen über Trigonometrie und geometrische Kurven anwenden und vertiefen. Das Beispiel eignet sich insbesondere in der Oberstufe, da die Lernenden dort bereits vertieftes Wissen über Funktionen erworben haben. Eine mögliche Problemstellung, welche bereits Hinweise auf die Problemlösung enthält, befindet sich in Kasten 6. Eine Lösung dieser Aufgabenstellung ist in den Abbildungen 9 und 10 zu sehen. Das wesentliche Element dieses Codes ist eine parametrisierte Kurve, die eine konische Spirale erzeugt. Hierzu wird in einer Schleife eine Laufvariable i in diskreten Schritten von 1 zwischen 1 und 720 verändert. Bei jedem Durchgang wird eine Kugel mit dem Radius 6 mm erzeugt. Durch die trigonometrischen Funktionen Sinus und Kosinus wird die Kugel je nach Wert der Variable i kreisförmig um den Mittelpunkt verschoben (i in °). Die Spirale entsteht, indem der Abstand vom Ursprung innerhalb der x–y-Ebene linear zunimmt, umgesetzt durch die Multiplikation der Koordinaten mit dem Wert $\frac{i}{36}$ (i in mm). Eine lineare Zunahme des z-Wertes der Verschiebung führt schließlich zu einer konischen Spirale im Raum. Die Kurve, welche hierdurch modelliert wird, lässt sich in Parameterdarstellung durch

$$t \in \{1, \dots, 720\}, \qquad t \to \mathbb{R}^3$$

$$\vec{r}(t) = \begin{pmatrix} \frac{i}{36} \cdot \cos(t) \\ \frac{i}{36} \cdot \sin(t) \\ \frac{t}{100} \end{pmatrix}$$

beschreiben.

Durch die Vereinigung der insgesamt 720 auf der Spirale angeordneten Kugeln entsteht der in Abbildung 10 dargestellte schneckenhausförmige Körper.

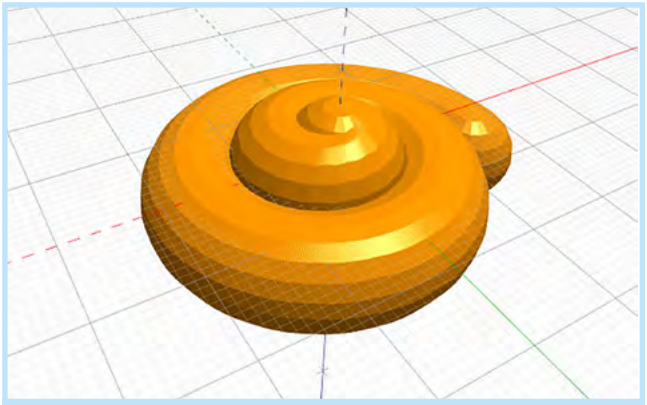

Abb. 10. Generiertes Schneckenhaus in BlocksCAD

Das Beispiel des Schneckenhauses eignet sich aus informatischer Sicht ebenfalls gut, um Laufzeiten von Algorithmen zu thematisieren. So benötigt der in Abbildung 9 dargestellte Algorithmus zum Rendern des 3D-Modells eine Zeit von etwa 5 Minuten. Dies ist gerade für die Entwicklung und Verbesserung des Programms unvorteilhaft. Daher ist es zu empfehlen, die Laufvariable während der Entwicklung in Schritten von 10 zu durchlaufen, was die Zeit zum Rendern wesentlich auf etwa 10 Sekunden reduziert. Auf diese Weise entsteht das in Abbildung 11 dargestellte Objekt. Es weist zwar noch nicht eine kontinuierliche Schneckenhausform auf, und die einzelnen Kugeln sind noch klar zu erkennen, für die Optimierung und Testung des Algorithmus reicht diese Version jedoch aus.

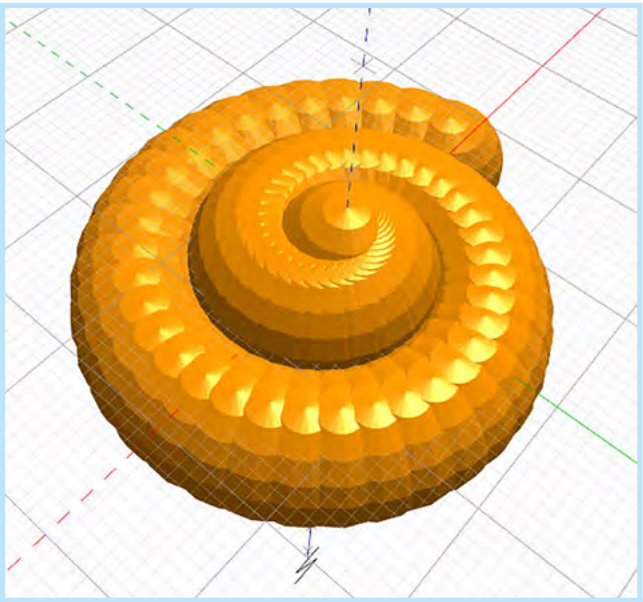

Abb. 11. Schneckenhaus generiert in Schritten von 10 in BlocksCAD

5 Aufgabenstellungen mit Algorithmen variieren

Die bisher vorgestellten Problemstellungen können dem Aufgabentyp „Implementierung“ (Milicic, Wetzel & Ludwig, 2020) zugeordnet werden. Bei diesem Aufgabentyp sind die Schüler/innen auf Basis einer gegebenen Beschreibung, möglicherweise auch in Verbindung mit einem Bild des gewünschten Modells, zur eigenständigen Umsetzung des Algorithmus angehalten.

Im Folgenden werden mit „Parsons-Puzzle“, „Analyse“ und „Finde den Fehler“ weitere mögliche Aufgabentypen für die Arbeit mit Algorithmen vorgestellt, und basierend auf den bereits thematisierten Algorithmen werden jeweils konkrete Aufgabenstellungen präsentiert. Der Schwierigkeitsgrad der Aufgaben ist hier ebenfalls von der konkreten Aufgabenstellung abhängig. Empirische Ergebnisse (Lopez, Whalley, Robbins & Lister, 2008) deuten jedoch darauf hin, dass der Aufgabentyp Parsons-Puzzle am leichtesten ist und die Analyse und Fehlersuche bei Programmen den Schüler/inne/n deutlich schwerer fallen. Als besonders herausfordernd wird die eigenständige Implementierung angesehen (Venables, Tan & Lister, 2009).

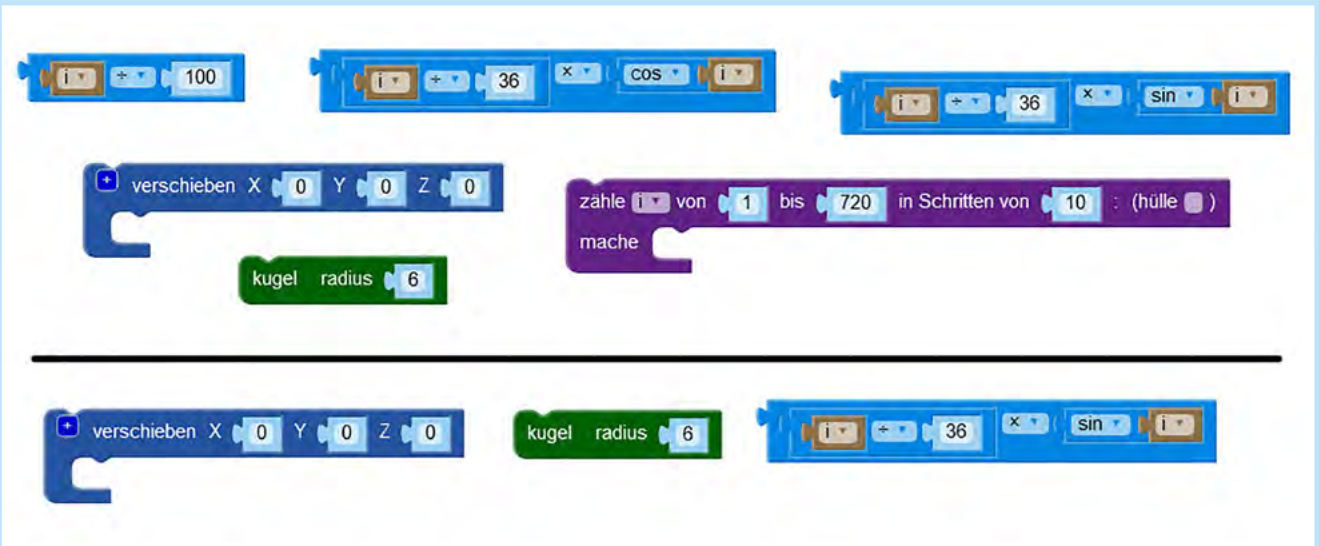

Abb. 12. Parsons-Puzzle zum Schneckenhaus

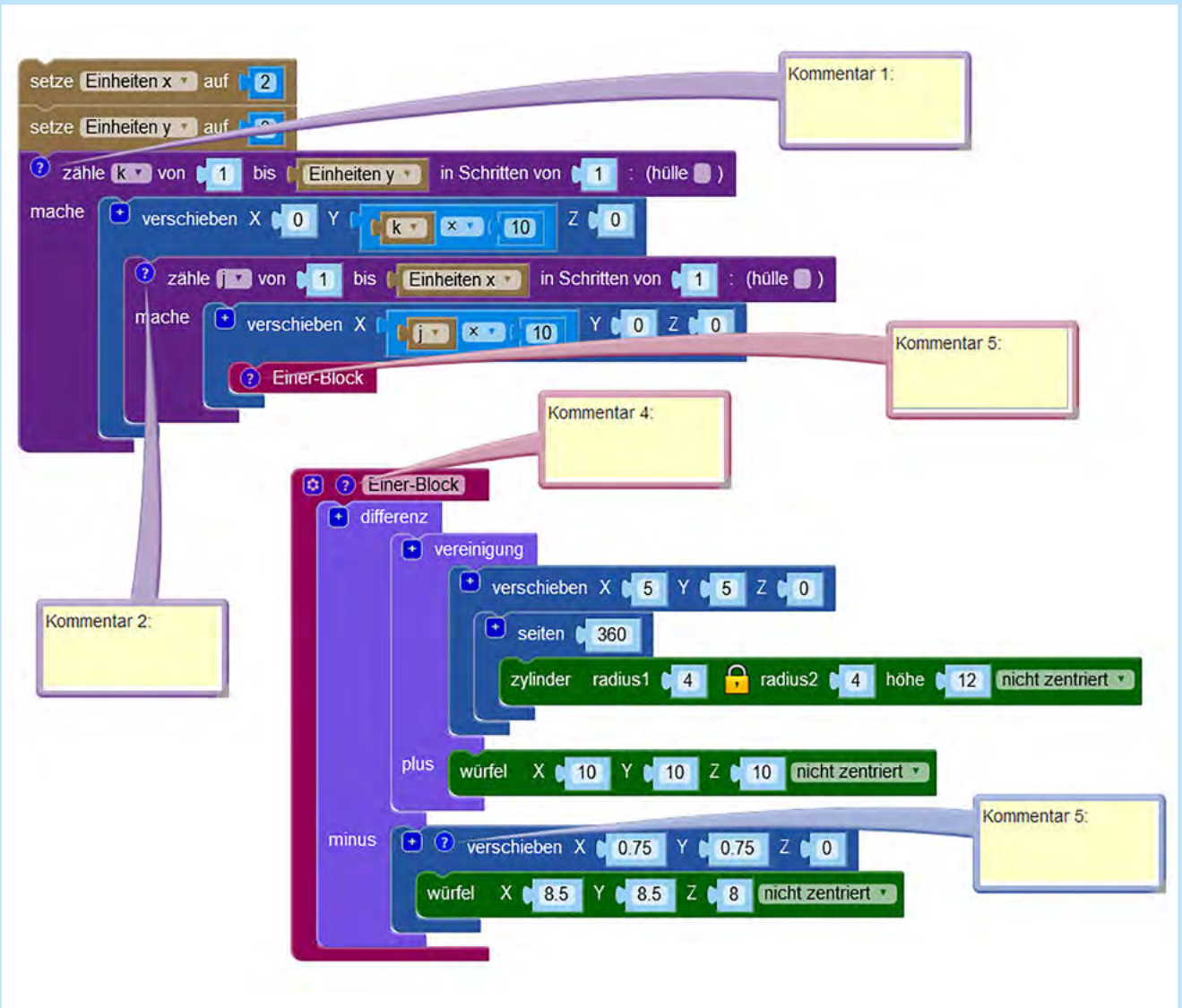

Abb. 13. Vorlage zur Analyse des Bausteingenerators

5.1 Parsons-Puzzle

Durch die zufällige Anordnung der Codezeilen bzw. Blöcke eines korrekten Programms wird der Aufgabentyp Parsons-Puzzle definiert. Die Aufgabe der Schüler/innen ist es, die Codeblöcke wieder korrekt zu positionieren. Zusätzliche, allerdings nicht benötigte Codeblöcke im Skriptbereich werden als Distraktoren bezeichnet und erhöhen die Komplexität der Aufgabe. Durch die Vorgabe der Codeblöcke wird bereits ein wesentlicher Teil des ursprünglichen Problemlöseprozesses vorweggenommen, nämlich die Übersetzung der Problemstellung in das mathematische Modell und die anschließende Auswahl der zur Verfügung stehenden Codeblöcke. Für die ursprünglich offen formulierte Aufgabe ist damit der Löseprozess entsprechend eingeschränkt. Gerade bei anspruchsvolleren Aufgaben wie dem Schneckenhaus kann dies allerdings durchaus erwünscht sein.

Die Parametrisierung der Kurve wird durch die Vorgabe der einzelnen Blöcke bereits implizit vorgegeben, sodass sich die Schüler/innen ausschließlich auf die informatische Problemlösung in Form der Erstellung des Programms konzentrieren können. In Abbildung 12 ist eine Vorlage basierend auf dem Schneckenhausalgorithmus gegeben. Falls auch die nicht benötigten Codeblöcke unterhalb der schwarzen Trennlinie vorgegeben werden, handelt es sich dabei um ein Parsons-Puzzle mit Distraktoren.

5.2 Analyse

Im Gegensatz zu den beschriebenen Typen Implementierung und Parsons-Puzzle ist beim Aufgabentyp Analyse nicht der Entwurf und die Umsetzung eines Algorithmus das Ziel der Aufgabe, sondern die Beschreibung eines bereits gegebenen Programms. Der Aufgabentyp Analyse kann z. B. zur gemeinsamen Wiederholung und Besprechung, zur Wissenssicherung oder auch zum Aufzeigen von neuen Konzepten genutzt werden. Die Lehrkraft kann durch das Anfügen von Kommentaren zusätzlich gewünschte Schwerpunkte setzen.
In Abbildung 13 ist eine entsprechende Vorlage zur Analyse der Aufgabe Bausteingenerator gegeben. Im Gegensatz zur ursprünglichen Aufgabe wurde die Erstellung eines Einer-Blocks des Bausteins in ein Modul ausgelagert, sodass die Definition und Nutzung von Modulen in BlocksCAD thematisiert werden kann. Die Schüler/innen können die Vorlage mit dem bereits bekannten Programm vergleichen. Die angefügten Kommentare dienen den Schüler/inne/n als Hilfestellung, um die ganzheitliche Beschreibung des Algorithmus durch eine gezielte Fokussierung auf die wesentlichen Elemente zu erleichtern. Falls eine alternative Notation der Handlungsschritte, wie in Kasten 5 gegeben, erfolgt ist, kann eine Verbindung zwischen der gewählten Notation und den einzelnen Kommentaren hergestellt werden.

5.3 Finde den Fehler

Bei der Aufgabe des Typs „Finde den Fehler“ werden in einem korrekten Programm zur Lösung einer Problemstellung bewusst einige Fehler eingefügt. Die Schüler/innen sind dann angehalten, die Fehler zu finden und zu korrigieren. Der Prozess der Fehlersuche und -korrektur wird im Kontext der Programmierung häufig auch als Debugging bezeichnet und spielt beim Programmieren eine große Rolle. Idealerweise basieren die bewusst integrierten Fehler auf bereits vorgestellten Aspekten oder sogar von der Lehrkraft bereits identifizierten und angesprochenen Fehlern der Schüler/innen, sodass die Aufgabe einen erneuten Reflexionsprozess initiieren kann. Dieser Aufgabentyp ist umfangreicher als die Analyse, da zunächst einmal auch alle Bestandteile des fehlerhaften Codes verstanden, analysiert und anschließend korrigiert werden müssen.

In Abbildung 14 ist ein fehlerhaftes Programm zur Bearbeitung der Wendeltreppenaufgabe auf Basis des Programms von Abbildung 6 gegeben. Der erste Fehler ist die Auswahl der Mengenoperation Differenz anstelle der Schnittmenge zwischen dem Quader und dem Zylinder. Der zweite Fehler ist im Translationsblock enthalten. Anstatt die Zählvariable zur Verschiebung zu nutzen, werden alle Grundfiguren konstant um 5 Einheiten in positiver Richtung der -Achse verschoben. Die Grundfiguren befinden sich also alle in einer Ebene.

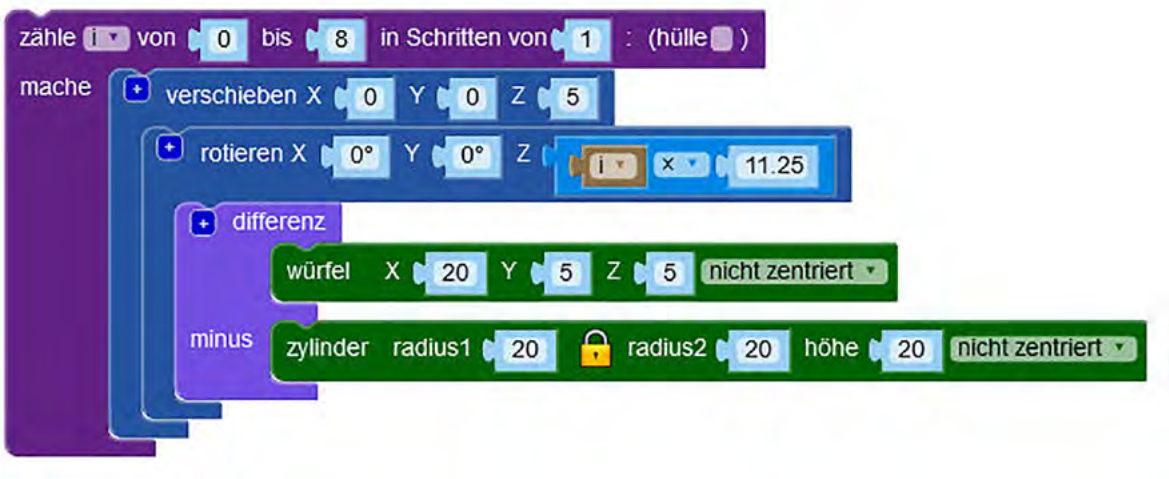

Abb. 14. Fehlerhaftes Programm zur Wendeltreppe

6 Ausblick

In diesem Beitrag wurde das Problemlösen im Kontext der 3D-Druck-Technologie an der Schnittstelle von Mathematik und Informatik betrachtet. Das Programmieren von Modellen für den 3D-Druck mithilfe der Blockprogrammierungssoftware BlocksCAD bietet im Mathematikunterricht die Möglichkeit, sich mit vielfältigen mathematischen Themen auseinanderzusetzen und gleichzeitig die Problemlösekompetenz und das algorithmische Denken zu fördern. Anhand der drei Beispielalgorithmen zur Generierung einer Wendeltreppe, eines Bausteins und eines Schneckenhauses wurde zudem expliziert, wie mathematische und informatische Kompetenzen bei der Blockprogrammierung einhergehen können. Schließlich wurden in einem weiteren Abschnitt Möglichkeiten aufgezeigt, wie sich die Algorithmen durch Variation der Aufgabenstellung auch in differenzierenden Lernsettings nutzen und an die Anforderungen des eigenen Unterrichts anpassen lassen. Unter den folgenden Links lassen sich die im Beitrag beschriebenen BlocksCAD-Projekte finden und direkt für die Weiterarbeit nutzen:

- Wendeltreppe:
  https://www.blockscad3d.com/community/projects/1170004
- Bausteingenerator:
- *https://www.blockscad3d.com/community/projects/1160786*
- Schneckenhaus:
- *https://www.blockscad3d.com/community/projects/1163745*
- Wendeltreppe – Finde den Fehler:
  https://www.blockscad3d.com/community/projects/1170076
- Bausteingenerator – Analyse:
  https://www.blockscad3d.com/community/projects/1169982
- Schneckenhaus – Parsons-Puzzle
  https://www.blockscad3d.com/community/projects/1169981

Literatur

Beckmann, A. (2003). *Fächerübergreifender Mathematikunterricht. Teil 4: Mathematikunterricht in Kooperation mit Informatik*. Hildesheim, Berlin: Franzbecker.

Dilling, F. (2019). *Der Einsatz der 3D-Druck-Technologie im Mathematikunterricht. Theoretische Grundlagen und exemplarische Anwendungen für die Analysis*. Wiesbaden: Springer.

Dilling, F. (2020). Authentische Problemlöseprozesse durch digitale Werkzeuge initiieren – eine Fallstudie zur 3D-Druck-Technologie. In F. Dilling & F. Pielsticker (Hg.), *Mathematische Lehr-Lernprozesse im Kontext digitaler Medien* (S. 161-180). Wiesbaden: Springer Spektrum.

Dilling, F., Marx, B., Pielsticker, F., Vogler, A. & Witzke, I. (2021). *Praxishandbuch 3D-Druck im Mathematikunterricht. Einführung und Unterrichtsentwürfe für die Sekundarstufen I und II*. Münster: Waxmann.

Dilling, F. & Witzke, I. (2020). Die 3D-Druck-Technologie als Lerngegenstand im Mathematikunterricht der Sekundarstufe II. *MNU-Journal*, *73*(4), 317–320.

Dörner, D. (1979). *Problemlösen als Informationsverarbeitung*. Stuttgart: Kohlhammer.

Förster, K-T. (2011). Neue Möglichkeiten durch die Programmiersprache Scratch: Algorithmen und Programmierung für alle Fächer. In R. Haug & L. Holzäpfel (Hg.). *Tagungsband GDM 45*. Tagung für Didaktik der Mathematik: WTM.

Kortenkamp, L. & Lambert, A. (2018). Wenn …, dann … bis … . *Mathematik lehren*, *188*, 2–9.

Ladel, S. (2020). Bildungstechnologien im Mathematikunterricht (Klassen 1-6). In H. Niegemann, A. Weinberger (Hrg.), *Lernen mit Bildungstechnologien* (S. 1–22). Heidelberg: Springer Reference.

Lopez, M., Whalley, J., Robbins, P. & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the fourth international workshop on computing education research*.

Milicic, G. (2020). Algorithmen und Neue Medien. *Mathematik im Unterricht*, *11*, 34–42.

Milicic, G., Wetzel, S. & Ludwig, M. (2020). Generic Tasks for Algorithms. *Future Internet 12*(9), 152.

Müller, H. & Weichert, F. (2013). *Vorkurs Informatik*. Wiesbaden: Springer.

Newell, A. & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Pielsticker, F. (2020). *Mathematische Wissensentwicklungsprozesse von Schülerinnen und Schülern. Fallstudien zu empirisch-orientiertem Mathematikunterricht am Beispiel der 3D-Druck-Technologie*. Wiesbaden: Springer Spektrum.

Pólya, G. (1949). *Schule des Denkens. Vom Lösen mathematischer Probleme*. Bern, München: Francke.

Rogers, H. (1987). *Theory of Recursive Functions and Effective Computability.* Cambridge: MIT Press.

Schoenfeld, A. (1985). *Mathematical Problem Solving*. New York: Academic Press.

Schubert, S. & Schwill, A. (2011). *Didaktik der Information*. Heidelberg: Spektrum.

SMITH, M. U. (1991). *Toward a unified theory of problem solving: Views from the content domains*. Hillsdale, NJ: Erlbaum.

VENABLES, A., TAN, G. & LISTER, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the fifth international workshop on computing education research*.

WITZKE, I. & HEITZER, J. (Hg.) (2019). 3D-Druck. Mathematik Lehren.

WINTER, H. (2001). *Fundamentale Ideen in der Grundschule*. *http://www.schulabakus.de/Wechselspiele/winter-ideen.html* (18.12.2020).

FREDERIK DILLING, dilling@mathematik.uni-siegen.de, ist Wissenschaftlicher Mitarbeiter an der Universität Siegen.

Dr. GREGOR MILICIC, milicic@math.uni-frankfurt.de, ist Wissenschaftlicher Mitarbeiter an der Goethe-Universität Frankfurt.

AMELIE VOGLER, vogler2@mathematik.uni-siegen.de, ist Wissenschaftliche Mitarbeiterin an der Universität Siegen.