Christine BESCHERER, Ludwigsburg & Andreas FEST, Ludwigsburg

Mathematische Entdeckungen und Computational Thinking

Entdeckendes Mathematiklernen ist nach Winter (2016) „die Idee [], dass Wissenserwerb, Erkenntnisfortschritt und die Ertüchtigung in Problemlösefähigkeiten nicht schon durch Information von außen geschieht, sondern durch eigenes aktives Handeln unter Rekurs auf die schon vorhandene kognitive Struktur, allerdings in der Regel angeregt und somit erst ermöglicht durch äußere Impulse.“ (Winter, 2016, S. 3)

Die Anregungen und die Impulse, die für das entdeckende Mathematiklernen notwendig sind, müssen auf die individuellen Charakteristika und Lern- bzw. Entdeckungsprozesse der Schülerinnen und Schüler abgestimmt sein. Abgesehen von der grundsätzlichen Forderung nach einem individualisierten Unterricht (u.a. Lipowski & Lotz, 2015) ist auch klar, dass ein Entdecken mathematischer Zusammenhänge nicht gemeinsam im Klassenverband gesteuert durch die Lehrkraft erfolgen kann. Eine der Grundlagen für entdeckendes Lernen sind geeignete Aufgaben. Alfieri, Brooks, Aldrich & Tenenbaum (2011) haben in einer Metastudie verschiedene Arten des entdeckenden Lernens verglichen und schließen, dass optimale Aufgaben für entdeckendes Lernen mindestens eine der folgenden Eigenschaften erfüllen sollten:

- Aufgaben, die ein Gerüst zur Unterstützung der Lernenden bieten
- Aufgaben, die verlangen, dass Lernenden ihre eigenen Ideen erklären sowie ein zeitnahes Feedback zur Sicherstellung, dass diese Ideen richtig sind
- Aufgaben, die ausgearbeitete Beispiele zur erfolgreichen Bearbeitung beinhalten (Alfieri et al. 2011, S. 13)

Damit Lernende mehr oder weniger selbst die mathematischen Entdeckungen machen können, müssen die Aufgaben an sich noch durch unterstützende Materialien, Hilfestellungen oder (digitale) Werkzeuge zu didaktisch gut geplanten Lernumgebungen erweitert werden. Ein Beispiel für digital unterstützte Lernumgebungen sind Mikrowelten, die Papert (1980) schon vor ca. 40 Jahren beschrieben hat und die u.a. von Kynigos (2007) zu “halbgaren Mikrowelten“ (‘half-baked microworlds‘) weiterentwickelt wurden. Unter ‘half-baked microworlds‘ werden didaktisch sorgfältig gestaltete, unvollständige Computerprogramme verstanden, die die Lernenden unmittelbar zum Nachfragen, Ausprobieren und Weiterdenken anregen. Dabei fokussiert die didaktische Gestaltung das Weiterdenken in die von der Lehrperson

Siller, H.-S., Weigel, W. & Wörler, J. F. (Hrsg.). *Beiträge zum Mathematikunterricht 2020.* Münster: WTM-Verlag, 2020.
doi: 10.37626/GA9783959871402.0

geplante Richtung, möglichst ohne die Lernenden in ihren Denkmöglichkeiten einzuschränken.

Im Folgenden wird ein Beispiel für eine Lernumgebung in Form einer „halbgaren Mikrowelt“ zum Thema *Teilbarkeit ganzer Zahlen* auszugsweise vorgestellt. Hier soll es den Lernenden durch das eigene Erstellen von Programmcodes ermöglicht werden, entsprechende mathematische Vorstellungen zu den Teilbarkeitsregeln zu entwickeln und direktes individuelles Feedback zu bekommen. (Es ist selbstverständlich möglich, unterschiedliche Programmiersprachen zu verwenden. Im folgenden Beispiel wird die Sprache FMSLogo (http://fmslogo.sourceforge.net/) verwandt.

Beispiel für eine Lernumgebung „Teiler und Teilbarkeit“ (Ausschnitt):

Einleitung: Mit dem Programm „teilt?“ kann man prüfen, ob die erste eingegebene Zahl ein Teiler der zweiten Zahl ist (s. Abb. 1).

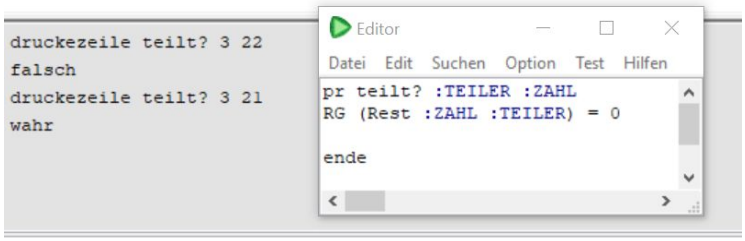

Abb. 1: Screenshot des Programms „teilt?“ (Editor) sowie zweier Beispiele der Ausgaben der Antworten auf dem Bildschirm („3 teilt 22 ist falsch“ bzw. „3 teilt 21 ist wahr“). Mathematisch wird überprüft, ob der Rest bei der Division 0 ist oder nicht.

Auftrag:

- *Welche Zahlen geben bei der Division durch 4 keinen Rest?*
- *Welche Zahlen geben bei der Division durch 9 keinen Rest?*
- *Welche Zahlen geben bei der Division durch 6 keinen Rest?*

Vorgehen:

- *Probiere zuerst die in der Befehlsliste angegebenen Befehle aus und notiere, was du nicht verstehst.*
- *Beantworte die Fragen oben. Dazu helfen einige – aber nicht alle – der Befehle aus der Liste.*
- *Untersuche, ob und wie sich die verschiedenen Regeln aus den Tippkarten auf die Fragen übertragen lassen.*
- *Schreibe eigene Algorithmen, die diese Regeln für beliebige Zahlen überprüfen können.*

... (Die Struktur des Beispiels ist angelehnt an Bescherer, 2005.)

Ein mögliches Programm, das die Teilbarkeit durch 4 prüft, wäre ein Vorgehen, in dem die zu prüfende Zahl als eine Liste von Ziffern aufgefasst wird. Die „Prüfzahl“ wird dann aus den letzten beiden Einträgen gebildet und auf die Teilbarkeit durch 4 geprüft. Diese Lösung wäre eine mehr oder weniger direkte Umsetzung der Regel für die Teilbarkeit durch 4, wie sie häufig in Schulbüchern steht.

Eine andere Variante des Algorithmus für die Teilbarkeit durch 4 könnte wie das in der folgenden Abbildung 2 dargestellte Programm aussehen:

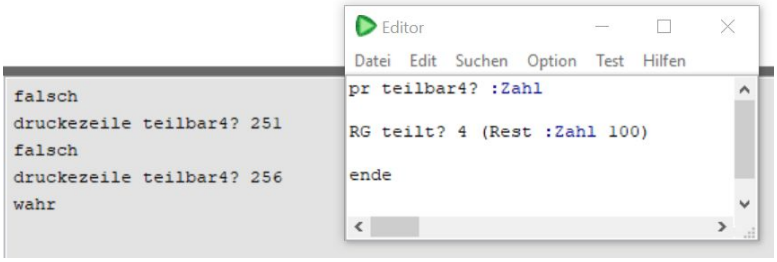

Abb. 2: Mögliche Lösung für die Prüfung der Teilbarkeit durch 4 unter Verwendung des Programms „teilt?“ Mathematisch wird überprüft, ob der Rest einer Zahl, der bei Division durch 100 bleibt, dann durch 4 teilbar ist oder nicht.

Bei dem Beispiel aus Abb. 2 kann mit etwas Nachfragen die Beweisidee für diese Teilbarkeitsregel abgeleitet werden, da 100 immer durch 4 teilbar ist und so also nur noch der Rest bei Division durch 100 betrachtet werden muss.

Wenn die Materialien entsprechend „hilfreich“ gestaltet werden, d.h. die Tippkarten nicht zu viel verraten, die Befehle in der Liste so zusammengestellt und erläutert wurden, dass die Auswahl für unterschiedliche Strategien ausgelegt ist, usw., können die Schülerinnen und Schüler durchaus selbstständig die mathematischen Entdeckungen machen und das direkte Feedback, ob ihre Überlegungen richtig sind, erfolgt immer durch die Reaktion des Programms.

Neben dem Entdecken mathematischer Zusammenhänge werden durch das Coding noch weitere – eher informatische – Fähigkeiten und Kompetenzen entwickelt.

Computational Thinking

Beim Erstellen der Programme wenden die Schülerinnen und Schüler automatisch typische Denk- und Arbeitsweisen aus der Informatik wie das Modellieren, das Zerlegen in Teilschritte, das Algorithmisieren und sicherlich das Testen und Eliminieren von Programmfehlern (Debugging) an. Genau solche Prozesse und Arbeitsweisen werden neben weiteren dem Computational Thinking zugeordnet.

Unter Computational Thinking versteht man, „die individuelle Fähigkeit einer Person, eine Problemstellung zu identifizieren und abstrakt zu modellieren, sie dabei in Teilprobleme oder -schritte zu zerlegen, Lösungsstrategien zu entwerfen und auszuarbeiten und diese formalisiert so darzustellen, dass sie von einem Menschen oder auch einem Computer verstanden und ausgeführt werden können." (Eickelmann, 2017, S. 53) Dabei bedeutet Computational Thinking nicht automatisch, dass auch programmiert werden muss, aber Programmieren ist eine wichtige passende Tätigkeit, Computational Thinking zu erwerben und anzuwenden (Grover & Pea, 2013).

Dabei hängen mathematisches Denken und Computational Thinking zwar zusammen und das eine kann durch das andere unterstützt werden, aber dies geschieht nicht automatisch. Futschek (2016) unterscheidet zwischen mathematischem Denken als dem Beweisen von Zusammenhängen und dem Computational Thinking als dem effizienten Erzielen von Ergebnissen. Beides muss jedoch im Unterricht entsprechend thematisiert werden und in sorgfältig geplanten Lernumgebungen von den Schülerinnen und Schülern selbst entwickelt werden.

Literatur

Alfieri, L., Brooks, P. J., Aldrich, N. J. & Tenenbaum, H. R. (2011). Does discovery-based instruction enhance learning? *Journal of Educational Psychology,* 103(1), 1.

Bescherer, C. (2005). LoDiC – Learning on Demand in Computing. *Proceedings of 8th IFIP World Conference on Computers in Education 2005,* Capetown.

Bescherer, C. & Fest, A. (in Druck). Mathematical Discoveries using Computational Thinking. In B. Barzel & F. Schacht (Hrsg.), *Proceedings of the 14th International Conference on Technology in Mathematics Teaching*. Essen, Germany: ICTMT 14. Dordrecht. Springer.

Eickelmann, B. (2017). Computational Thinking als internationales Zusatzmodul zu I-CILS 2018-Konzeptionierung und Perspektiven für die empirische Bildungsforschung. *Tertium Comparationis,* 23(1), 47.

Futschek, G. (2016). Computational Thinking im Unterricht. *Schule Aktiv,* 10, 4–5.

Grover, S. & Pea, R. (2013) Computational Thinking in K–12 A Review of the State of the Field. *In Educational Researcher,* 42(1), 38–43.

Kynigos, C. (2007) Half-baked logo microworlds as boundary objects in integrated design. In *Informatics in Education – An International Journal,* Vol 6,2, 335–359.

Lipowsky, F. & Lotz, M. (2015*). Ist Individualisierung der Königsweg zum erfolgreichen Lernen? Begabungen entwickeln & Kreativität fördern*. München: kopaed.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* Brighton: Harvester Press.

Winter, H. (2016). *Entdeckendes Lernen im Mathematikunterricht.* 3. Auflage. Wiesbaden: Springer Spektrum.