Aufgabe ist nicht gleich Aufgabe – Vielfältige Aufgabentypen bewusst in Scratch einsetzen

Katharina Geldreich[1], Mike Talbot[1] und Peter Hubwieser[1]

Abstract: Aufgaben sind ein fester Bestandteil jeden Unterrichts und können je nach Einsatz verschiedene didaktische Funktionen übernehmen. Auch beim Programmierenlernen spielen Aufgaben eine zentrale Rolle und können bei entsprechender Auswahl und Anwendung den Lernprozess der Schülerinnen und Schüler entscheidend beeinflussen. In der internationalen Fachliteratur lassen sich verschiedene Frameworks und didaktische Ansätze finden, die sich damit auseinandersetzen, welche Entwicklung die Lernenden beim Programmierenlernen durchlaufen. Diese Ansätze und einhergehende mögliche Aufgabenstellungen wurden im Rahmen einer Fortbildungsmaßnahme für Grundschullehrkräfte zum Thema Algorithmik und Programmierung thematisiert. Damit die Teilnehmerinnen und Teilnehmern gleichzeitig eine große Vielfalt an Aufgabenstellungen kennenlernen, wurden bei dem Design der Aufgaben bereits bestehende Aufgabenklassifikationen für Programmieraufgaben herangezogen. Der Praxisbeitrag beschreibt die erwähnten theoretischen Ansätze und zeigt, wie sie bei der Aufgabengestaltung umgesetzt werden können. Unser Fokus liegt darauf, wann welcher Aufgabentyp im Lernprozess sinnvoll eingesetzt werden kann. Sämtliche Aufgaben beziehen sich dabei auf die blockbasierte Programmiersprache Scratch.

Keywords: Programmieren; Aufgabentypen; Scratch

1 Einleitung

Aufgaben können auf verschiedene Ziele ausgerichtet sein und dementsprechend unterschiedliche didaktische Funktionen übernehmen. Dabei unterscheidet man grundlegend zwischen Aufgaben, die Lernende beim Aufbau und der Festigung von Wissen und Fähigkeiten unterstützen, sowie prüfenden Aufgaben zur Ermittlung des Leistungsstands [Ri05]. Erstere stellen für Lehrkräfte ein zentrales Gestaltungselement für den Unterricht dar und können ihnen helfen, Lernprozesse gezielt zu steuern. Auch beim Erlernen einer Programmiersprache sind geeignete Aufgaben wichtig, um den Lernprozess der Schülerinnen und Schüler bei entsprechender Auswahl und Anwendung entscheidend zu beeinflussen. Wie der Prozess des Programmierenlernens gestaltet werden sollte, ist nicht abschließend geklärt und stellt Lehrende wie Lernende vor Herausforderungen [MR02]. Verschiedene Autoren beschreiben didaktische Ansätze, in denen die Lernenden zunächst mit bereits bestehenden Programmen arbeiten und diese immer weiter modifizieren, bis sie schließlich selbstständig programmieren. Diese Ansätze und damit zusammenhängende Aufgaben waren Gegenstand einer Fortbildung für Grundschullehrkräfte zum Thema *Algorithmen und*

[1] Technische Universität München, School of Education, Professur für Didaktik der Informatik, Arcisstr. 21, 80333 München, {katharina.geldreich,mike.talbot,peter.hubwieser}@tum.de

Programmierung. Um die Lehrkräfte gleichzeitig mit unterschiedlichen Aufgabentypen in Scratch vertraut zu machen, wurde bei der Aufgabengestaltung eine bestehende Taxonomie für Programmieraufgaben herangezogen. Dieser Beitrag gibt zunächst einen Einblick in die zuvor erwähnten didaktischen Ansätze und Aufgabentaxonomien und beschreibt, wie im Kontext der Fortbildung damit gearbeitet wurde. Um einen möglichst praxisnahen Einblick zu geben, werden verschiedene konkrete Beispielaufgaben beschrieben, die sich auf die visuelle Programmiersprache Scratch beziehen. Diese können sowohl im Fortbildungs- als auch im Unterrichtskontext eingesetzt werden.

2 Theoretischer Hintergrund

2.1 Scaffolding im Unterricht

Der Begriff *Scaffolding* (zu dt. *Gerüst*) bezieht sich im pädagogisch-psychologischen Kontext auf die Unterstützung eines Lernprozesses durch verschiedene Maßnahmen wie Anleitungen, Denkanstöße oder andere Hilfestellungen [Sc06]. Diese Maßnahmen sollen die Schülerinnen und Schüler auf dem Weg zum selbstständigen Handeln begleiten und werden mit zunehmendem Können zurückgenommen. Scaffolding kann auf verschiedenen Ebenen stattfinden und sich auf die Planung von Lernaufgaben und Unterrichtssequenzen (*Makro-Ebene*), die Strukturierung von Lernaufgaben in einzelne Arbeitsschritte (*Meso-Ebene*) oder die Interaktion zwischen Lehrkraft und Lernenden (*Mikro-Ebene*) beziehen [Va96]. Im Kontext der Informatikdidaktik untersuchen sowohl Lee et al. [Le11] als auch Sentance und Waite [SW17], welche Entwicklung die Schülerinnen und Schüler auf dem Weg zum selbstständigen Programmieren durchlaufen sollten und wie man sie im Sinne des Scaffolding dabei unterstützen kann. Lee et al. entwickelten das *Use-Modify-Create*-Lehrkonzept, in dem die Lernenden zunächst mit bereits bestehenden Programmen arbeiten und diese schrittweise modifizieren bis sie schließlich eigene Programme erstellen. Daraufhin folgt ein iterativer Prozess des Testens, Analysierens und der Weiterentwicklung. Sentance und Waite stellen den Lehransatz PRIMM vor, der unter anderem auf der Arbeit von Lee et al. basiert. PRIMM ist die Abkürzung für *predict*, *run*, *investigate*, *modify* und *make* (zu dt. *vorhersagen*, *ausführen*, *untersuchen*, *anpassen*, *machen*). In beiden Ansätzen arbeiten die Lernenden zunächst mit bestehenden Programmen, nehmen kleinere und größere Veränderungen daran vor und erstellen schließlich komplette Programme eigenständig (Abb. 1).

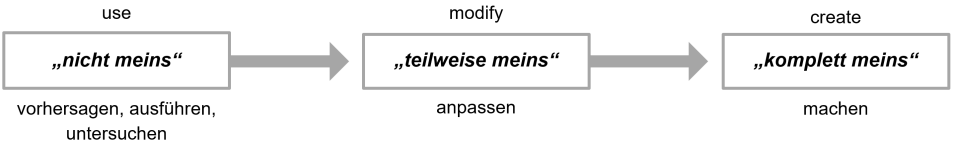

Abb. 1: Scaffolding beim Programmierenlernen

2.2 Aufgabentypen

Aufgaben sind ein zentraler Bestandteil des Informatikunterrichts und spielen sowohl für das Erarbeiten neuer Inhalte als auch für die Ermittlung des Leistungsstands eine wichtige Rolle. Besonders beim Programmierenlernen wird betont, dass sich Expertise nur durch praktische Aufgaben und intensives Üben einstellen kann [HM06]. Für die Entwicklung von Aufgaben lassen sich verschiedene Qualitätskriterien finden, an denen sich Lehrende orientieren können [BHP14]. Neben der Berücksichtigung von Faktoren wie Kompetenz- und Kontextorientierung wird empfohlen, die Aufgabentypen zu variieren. Welche Aufgabentypen im Kontext des Programmierens verbreitet sind, wurde mehrfach untersucht. Dabei entstanden verschiedene Aufgabentaxonomien, die z.B. deduktiv durch die Analyse verschiedener Curricula [Bo08] sowie induktiv aus der Analyse von Programmieraufgaben abgeleitet wurden [RBH13]. Ruf et al. [RBH15] unterscheiden zwölf verschiedene Aufgabentypen, die sich durch die zur Lösung erforderlichen Schülerhandlungen (z.B. Code testen, Code schreiben) und die Art der Darstellung der gegebenen Informationen (z.B. Text, Code, Bilder) unterscheiden.

3 Kontext

Um zu beurteilen, ob sich Aufgaben für die jeweilige Zielgruppe und deren Kenntnisstand eignen, ist es wichtig, dass Lehrkräfte entsprechende Kriterien kennenlernen [HB14]. Neben allgemeindidaktischen Kriterien wie z.B. Offenheit der Aufgabe, Lebensweltbezug [Ma10], sollten Aufgaben immer auch im Kontext der jeweiligen Fachdidaktik gesehen werden. Der Bereich *Aufgaben* ist daher auch in Modellen des fachdidaktischen Wissens (*pedagogical content knowledge*) für die Informatik zu finden [Hu13].
Zur Ausbildung von unterrichtsmethodischem Know-How wurden Grundschullehrkräfte im Rahmen einer mehrtägigen Fortbildungsmaßnahme mit verschiedensten Programmieraufgaben konfrontiert. Um die Programmierkenntnisse der Teilnehmenden weiter zu festigen, wurden sämtliche Programmieraufgaben zunächst eigenständig bearbeitet. Im Anschluss wurde der *Use-Modify-Create*-Ansatz, PRIMM sowie die Aufgabentaxonomie von Ruf et al. eingeführt (vgl. Punkt 2). Um die Theorie mit der Praxis zu verknüpfen, ordneten die Lehrkräfte die Aufgaben im Anschluss in die Scaffolding-Kategorien *use*, *modify* und *create* ein. Hier arbeiteten die Lehrkräfte zu zweit mit einem Satz Kärtchen, auf denen die Programmieraufgaben in Miniatur abgebildet waren und die entsprechend gruppiert werden mussten. Dieses Vorgehen förderte die Auseinandersetzung mit den verschiedenen Aufgabentypen in Bezug auf die Einordnung in die Scaffolding-Kategorien.
Die Fortbildung ist Teil des Projekts *AlgoKids – Algorithmen für Kinder*, das vom Bayerischen Staatsministerium für Unterricht und Kultus gefördert und über einen Zeitraum von zwei Jahren durchgeführt wird [GTH18]. Im Rahmen des Projekts werden insgesamt 40 Grundschullehrkräfte in den Bereichen *Algorithmik* und *Programmierung* fachwissenschaftlich und fachdidaktisch fortgebildet und bei eigenen Unterrichtsversuchen begleitet.

4 Aufgaben

Die nachfolgende Auswahl zeigt exemplarisch, welche Aufgabentypen im Laufe der Fortbildung bearbeitet wurden. Die teilnehmenden Lehrkräfte haben in einer ersten Fortbildung bereits intensiv mit Scratch gearbeitet, der Fokus lag hierbei jedoch vermehrt auf fachwissenschaftlichen Inhalten, insbesondere den algorithmischen Grundstrukturen.

4.1 Programme werden gelesen, ausgeführt, untersucht (*use*)

Programme nachbauen (Code→Code)
Ein komplett vorgegebenes Programm wird in Scratch nachprogrammiert und ausgeführt. Im Anschluss daran sollen Stellen markiert werden, bei denen man besonders sorgfältig sein musste (z.B. die Auswahl der richtigen Kostüme, Einfügen der *fühlen*-Blöcke in eine bedingte Anweisung). Der Fokus dieser Aufgabe liegt vorrangig auf der Bedienung von Scratch und der Handhabung der Programmierblöcke.

Blöcke erkunden (Code→Text)
Die Lernenden öffnen ein bereits bestehendes Scratch-Projekt und führen dieses aus. Auf einem Aufgabenblatt wird notiert, welche Funktionen die einzelnen Blöcke haben (Abb. 2). Aufgaben dieser Form eignen sich besonders für die Einführung von noch unbekannten Blöcken oder komplett neuen Scratch-Erweiterungen (z.B. die Übersetzungsblöcke in Scratch 3).

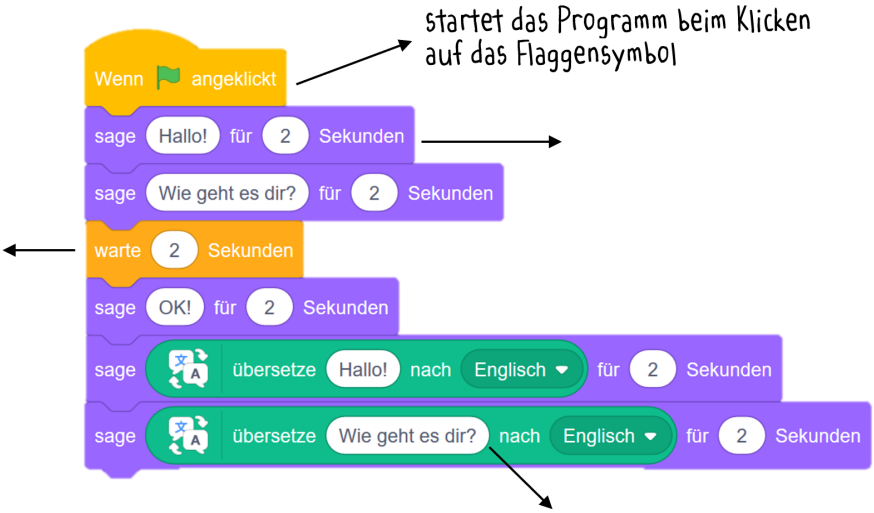

Abb. 2: Blöcke erkunden – Welche Funktion haben die einzelnen Blöcke?

Programme lesen (Code→Text)
Ein vorgegebenes Programm wird zunächst auf einem Aufgabenblatt gelesen (Abb. 3).

Danach soll beschrieben werden, was passiert, wenn der Mauszeiger unterschiedliche Positionen auf der Bühne einnimmt (z.B. Mauszeiger steht auf Position 2 → Schmetterling bewegt sich in Richtung des Igels). Die Lösungen werden im Anschluss durch Ausführen des Programms überprüft.

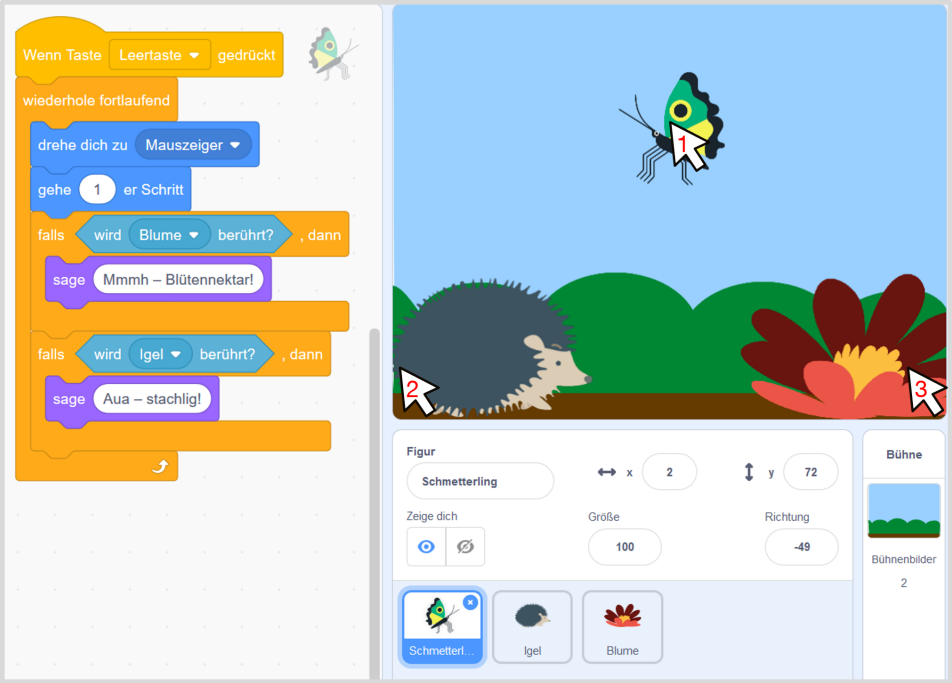

Abb. 3: Programme lesen – Was passiert nach dem Drücken der Leertaste, wenn der Mauszeiger auf den verschiedenen Positionen steht?

Programme lesen und Fehler finden (Code→Code)

Ein vorgegebenes Programm soll auf Fehler überprüft werden. Der Weg eines Marienkäfers durch ein Raster wird zunächst verfolgt (Abb. 4). Im Anschluss sollen Fehler im Programm gekennzeichnet werden, die dazu führen, dass der Käfer nicht bei seinem Ziel, der Blume, ankommt.

Programme testen (Code→Text)

Es soll getestet werden, ob ein bestehendes Programm das tut, was es soll. Hierzu wurde ein Programm erstellt, in dem eine Zahlenfee zwei durch den Nutzer eingegebene Variablen addiert und das Ergebnis ausgibt. Die Lernenden füllen eine Tabelle aus, in der sowohl die beiden Eingaben als auch deren Summe erfasst werden. Zusätzlich soll getestet werden was passiert, wenn Dezimalzahlen, besonders lange Ziffern oder Wörter eingegeben werden.

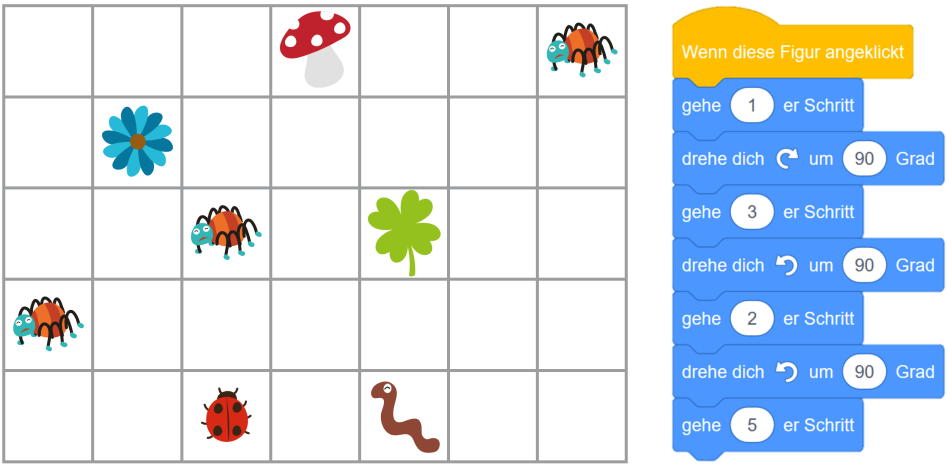

Abb. 4: Programme lesen und Fehler finden – Wie muss man das Programm verändern, dass der Marienkäfer bei der Blume ankommt?

4.2 Programme werden verändert, angepasst, erweitert (*modify*)

Programme debuggen (Code→Text+Code)
Ein Programm erfüllt nicht die gewünschte Funktion (Abb. 5). Es wird analysiert, wo sich der Fehler befindet und das Programm wird angepasst, um ihn zu beheben. Der Fokus liegt hier auf dem genauen Nachvollziehen des Programmablaufs.

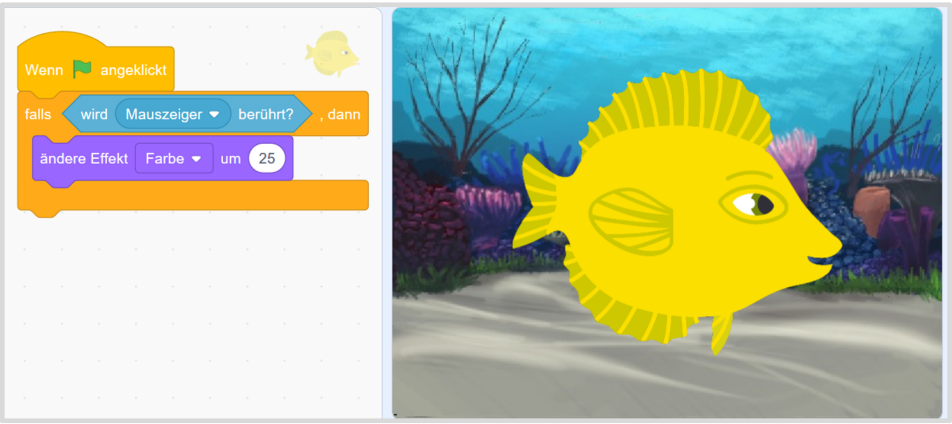

Abb. 5: Programme debuggen – Warum wechselt der Fisch beim Anklicken der grünen Flagge nicht die Farbe?

Programme erweitern (Code+Text→Code)
Ein Programm ist in Teilen vorgegeben und soll um weitere Funktionen erweitert werden. Die Bewegung einer Figur lässt sich bereits mit den Pfeiltasten steuern. Das Programm soll so erweitert werden, dass die Figur einen Dialog startet, wenn sie eine andere berührt.

Programme optimieren (Code→Code)
Ein fertiges Programm, welches die gewünschte Funktion scheinbar erfüllt, wird mittels verschiedener Fallunterscheidungen untersucht. Der Programmcode soll so optimiert werden, dass die gewünschte Funktion in jeder Situation erfüllt wird. In einem vorgegebenem Programm läuft eine Maus nacheinander zu verschiedenen Futterquellen. Da die Maus jedoch nur einzelne Koordinaten abläuft, kann sie das Futter nicht mehr finden, sobald man es auf der Bühne verschiebt. Die Lernenden sollen das Programm so modifizieren, dass die Maus immer zu ihrem Futter findet (Abb. 6).

Abb. 6: Programme optimieren – Mit dem rechten Programm läuft die Maus auch zu ihrem Futter, wenn man die Figuren auf der Bühne verschiebt

Szenario in Scratch anpassen (Code+Text→Code)
Für ein vorgegebenes Programm soll das Bühnenbild so angepasst werden, dass die gewünschte Funktionalität erfüllt wird. Ein Geist möchte zu seinem Schlüssel (Abb. 7). Er kann bereits mit den Pfeiltasten gesteuert werden, ist aber nicht in der Lage, durch die Wand zu seinem Schlüssel zu fliegen. Das Bühnenbild soll so angepasst werden, dass der Geist zu seinem Schlüssel fliegen kann (z.B. durch Verändern der Randfarbe, Vergrößern des Rechtecks).

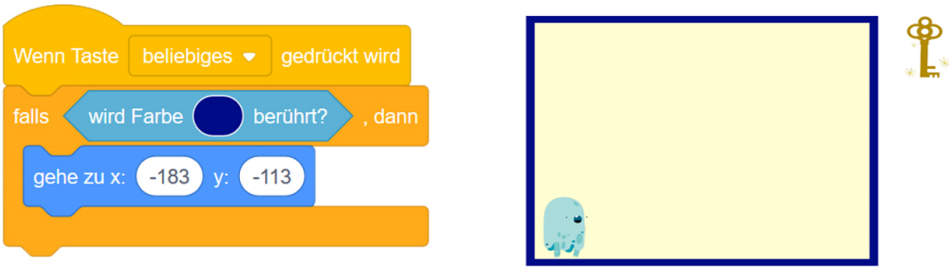

Abb. 7: Szenario in Scratch anpassen – Wie kann man das Bühnenbild verändern, dass der Geist zu seinem Schlüssel kommt?

4.3 Programme werden eigenständig erstellt (*create*)

Programmieren nach bildlichen Vorgaben (Bild→Code)
Eine vorgegebene Bildergeschichte soll in Scratch nachprogrammiert werden (Abb. 8). Dabei kann variiert werden, ob die Lernenden zusätzlich zu der Programmierung der Figur z.B. auch die Bühnenbilder selbst gestalten sollen.
In der Klassifikation von Ruf et al. [RBH15] gibt es den Aufgabentyp *Diagramm→Code*, in dem ein Programm zu einem Diagramm (z. B. Klassen-, Zustands-, Sequenzdiagramm) geschrieben wird. Da Diagramme dieser Form in der Grundschule nicht eingeführt werden, wurde stattdessen *bildliche Vorgaben* gewählt.

Abb. 8: Programmieren nach bildlichen Vorgaben – Die Bildergeschichte soll in Scratch implementiert werden

Programmieren nach textlichen Vorgaben (Text→Code)
Das Schreiben von Code nach textlichen Vorgaben gehört zu den häufigsten Aufgabentypen für Programmieranfänger [RBH15]. Es kann dabei variiert werden, wie detailliert man vorgibt, was programmiert werden soll.
Es soll eine Abwandlung des Spiels *Breakout* programmiert werden. Dazu bekommen die Lernenden eine kurze Beschreibung des Spiels:

Ein Erdball prallt vom Rand der Bühne und von einem Paddel ab. Er darf den Boden unter dem Paddel nicht berühren – sonst stoppt das Spiel. Mit dem Paddel, das man als Spieler von rechts nach links steuern kann, versucht man den Erdball vom Boden fernzuhalten.

Programmieren eigener Ideen nach textlichen Vorgaben (Text→Code)
Auch wenn man vorgibt, was programmiert werden soll, kann man die Aufgaben so gestalten, dass die Lernenden dennoch ihre eigenen Ideen umsetzen können. An einem Weihnachtsbaum hängen verschiedene Kugeln. Diese sollen so programmiert werden, dass jede Kugel beim Anklicken etwas anderes macht (z.B. Ton abspielen, Farbe ändern, sich bewegen).

5 Fazit

Im Beitrag wurde gezeigt, wie man durch variierende Aufgaben den Lernprozess des Programmierens gezielt steuern kann. Dabei handelt es sich keinesfalls um eine strenge Handlungsempfehlung, die Aufgaben sollen vielmehr dazu anregen, den eigenen Blick für das Potenzial unterschiedlicher Aufgabentypen zu schärfen.
Das Feedback der Grundschullehrkräfte, die an der beschriebenen Fortbildungsmaßnahme teilgenommen haben, war durchweg sehr positiv. Es wurde berichtet, dass in vielen Büchern, die sich mit dem Thema *Programmieren für Kinder* auseinandersetzen, nur Aufgaben bzw. Anleitungen zu finden sind, in denen Programme schrittweise nachgebaut werden müssen. Sich damit auseinanderzusetzen, wann dieses Kopieren von Code sinnvoll sein kann und wie man die Schülerinnen und Schüler darüber hinaus auf dem Weg zum selbstständigen Programmieren unterstützen kann, wurde von den Lehrkräften als sehr hilfreich eingeschätzt. Das Programmieren im Kontext der Grundschule ist für die Lehrkräfte eine sehr neue Thematik und sie können noch nicht auf einen vergleichbaren Aufgabenpool wie in anderen Fächern zurückgreifen. Wir halten es deshalb für umso wichtiger, sie von Beginn an mit den unterschiedlichsten Aufgabenstellungen vertraut zu machen und sie zu befähigen, Aufgaben gezielt und bewusst einzusetzen.

Literatur

[BHP14] Brichzin, P.; Humbert, L.; Puhlmann, H.: Aufgabenkultur im Schulfach Informatik. LOG IN 34/1, S. 27–32, 2014.

[Bo08] Bower, M.: A Taxonomy of Task Types in Computing. In: Proceedings of the 45th annual Design Automation Conference. ACM Press, New York, S. 281–285, 2008.

[GTH18] Geldreich, K.; Talbot, M.; Hubwieser, P.: Off to new shores: Preparing Primary School Teachers for Teaching Algorithmics and Programming. In (Mühling, A.; Cutts, Q., Hrsg.): Proceedings of the 13th Workshop in Primary and Secondary Computing Education. ACM Press, New York, USA, S. 1–6, 2018.

[HB14] Humbert, L.; Brichzin, P.: Unterrichtskultur und Aufgaben im Informatikunterricht. LOG IN 34/1, S. 45–53, 2014.

[HM06] Hassinen, M.; Mäyrä, H.: Learning Programming by Programming: a Case Study. In (Berglund, A.; Wigbberg, M., Hrsg.): Proceedings KolliCalling 2006. S. 117–119, 2006.

[Hu13] Hubwieser, P.; Berges, M.; Magenheim, J.; Schaper, N.; Bröker, K.; Margaritis, M.; Schubert, S.; Ohrndorf, L.: Pedagogical content knowledge for computer science in German teacher education curricula. In (Caspersen, M. E.; Knobelsdorf, M.; Romeike, R., Hrsg.): Proceedings of the 8th Workshop on Primary and Secondary Computing Education. ACM Press, New York, USA, S. 95–103, 2013.

[Le11] Lee, I.; Martin, F.; Denner, J.; Coulter, B.; Allan, W.; Erickson, J.; Malyn-Smith, J.; Werner, L.: Computational thinking for youth in practice. ACM Inroads 2/1, S. 32, 2011.

[Ma10] Maier, U.; Kleinknecht, M.; Metz, K.; Bohl, T.: Ein allgemeindidaktisches Kategoriensystem zur Analyse des kognitiven Potenzials von Aufgaben. Beiträge zur Lehrerinnen- und Lehrerbildung 28/1, S. 84–96, 2010.

[MR02] Milne, I.; Rowe, G.: Difficulties in Learning and Teaching Programming — Views of Students and Tutors. Education and Information Technologies 7/1, S. 55–66, 2002.

[RBH13] Ruf, A.; Berges, M.; Hubwieser, P.: Types of assignments for novice programmers. In (Caspersen, M. E.; Knobelsdorf, M.; Romeike, R., Hrsg.): Proceedings of the 8th Workshop on Primary and Secondary Computing Education. ACM Press, New York, USA, S. 43–44, 2013.

[RBH15] Ruf, A.; Berges, M.; Hubwieser, P.: Classification of Programming Tasks According to Required Skills and Knowledge Representation. In (Brodnik, A.; Vahrenhold, J., Hrsg.): Informatics in Schools. Springer, Heidelberg, S. 57–68, 2015.

[Ri05] Rieck, K.: Gute Aufgaben. IPN, Kiel, 2005.

[Sc06] Schnotz, W.: Pädagogische Psychologie: Workbook. Beltz, PVU, Weinheim, Basel, 2006.

[SW17] Sentance, S.; Waite, J.: PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. In (Barendsen, E.; Hubwieser, P., Hrsg.): Proceedings of the 12th Workshop on Primary and Secondary Computing Education. ACM Press, New York, USA, 2017.

[Va96] Van Lier, L.: Interaction in the Language Curriculum - Awareness, Autonomy and Authenticity. Longman, Harlow, 1996.