Robot Karol: Eine Grabrede

Michaeli, T.
TU München

DOI: 10.18420/ibis-02-01-03

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

Liebe Trauergemeinde,

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

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

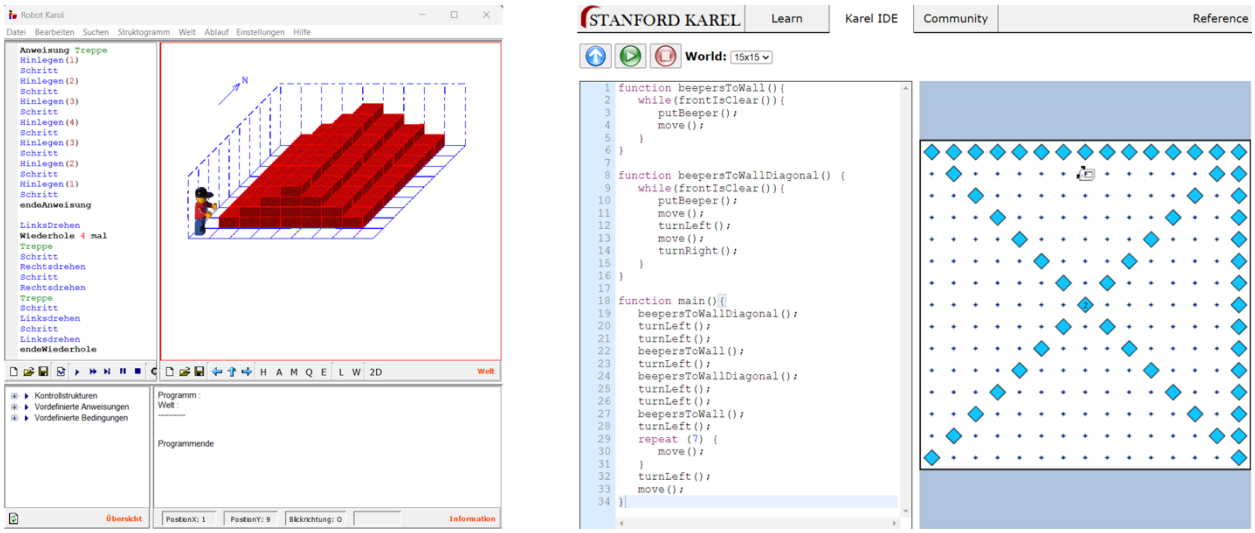

Abbildung 1: Karol und Karel (Screenshots)

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

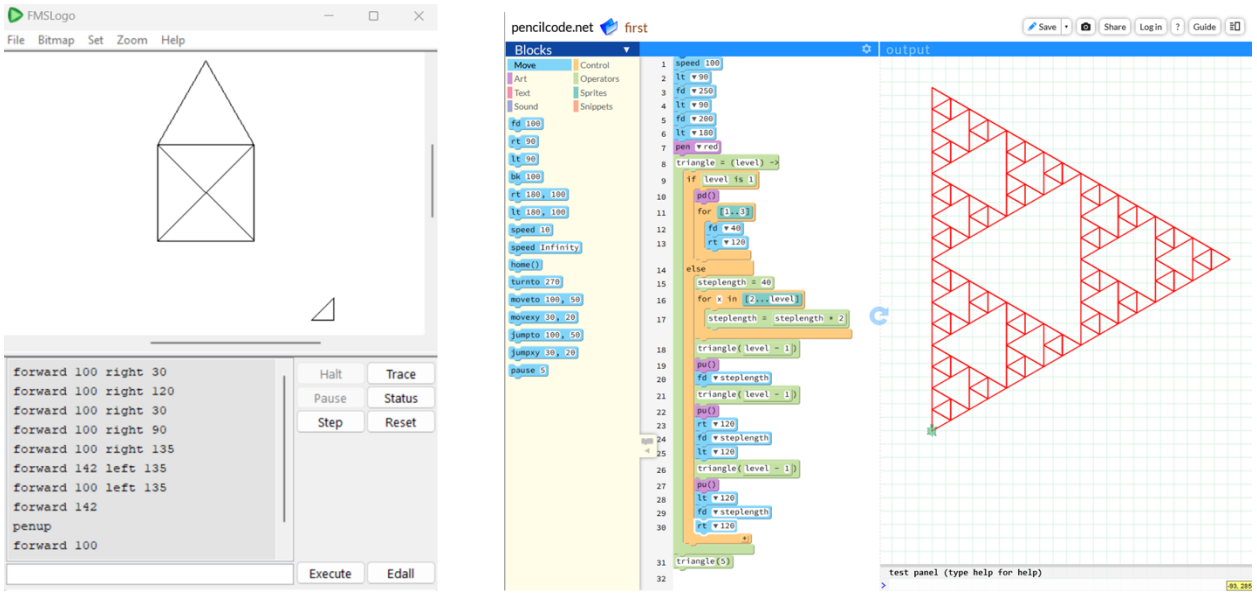

Abbildung 2: Turtlegrafik in Logo und PencilCode (Screenshots)

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

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

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

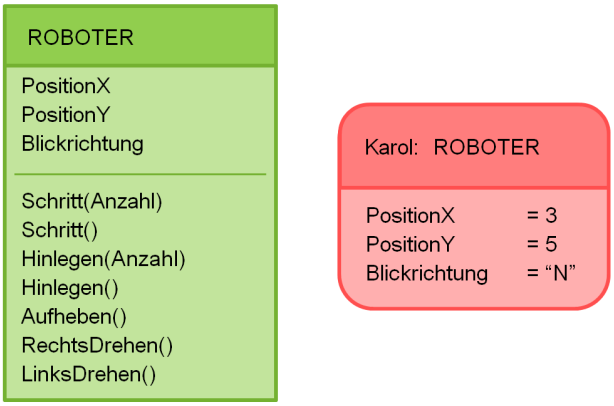

Abbildung 3: Exemplarische Darstellung eines OO-Karol ohne Zweck der Modellierung wie etwa in Schulbüchern üblich

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

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

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

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

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

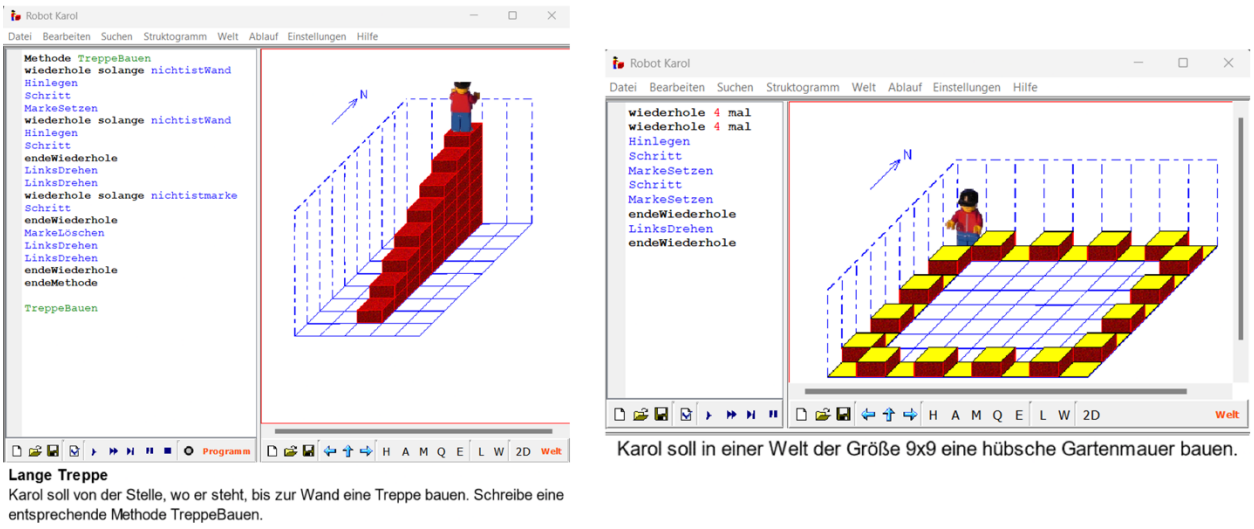

Abbildung 4: Typische Aufgaben (keine Probleme) in Robot Karol (Screenshots)

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

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

[3] Eigene Übersetzung aus dem Englischen.

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

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

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

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

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

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

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

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

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

Quellen

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lizenz

Kontakt

Prof. Dr. Tilman Michaeli

Didaktik der Informatik, TU München

tilman.michaeli@tum.de