Digital Experiences in Mathematics Education (2021) 7:48–65
https://doi.org/10.1007/s40751-020-00083-3

Programming as Language and Manipulative for Second-Grade Mathematics

E. Paul Goldenberg[1] · Cynthia J. Carter[2] · June Mark[1] · Kristen Reed[1] · Deborah Spencer[1] · Kate Coleman[1]

Accepted: 14 October 2020 / Published online: 27 February 2021

Abstract

This article reports on an exploration of how second-graders can learn mathematics through programming. We started from the theory that a suitably designed programming language can serve children as a language for expressing and experimenting with mathematical ideas and processes in order to do mathematics and thereby, with appropriate tasks and teaching, learn and enjoy the subject. This is very different from using the computer as a teaching app or a digital medium for exploration. Children tackled genuine puzzles – problems for which they did not already have a pre-learned solution. So far, we have built four microworlds for second-graders and tested them with a diverse population of well over three hundred children. The microworlds focus on the most critical second-grade mathematical content (as mandated in state standards), let children pick up all key programming ideas in contexts that make them 'obvious' (to maintain focus on the mathematics) and suppress all other distractions to minimize overhead for teachers or students using the microworlds. Because children see the results of the actions they articulate (in the computer language, Snap*!*), they can evaluate their methods and solutions themselves. The feedback is purely the outcome, not happy or sad sounds from the computer. Notably, nearly all children showed intense engagement, some choosing microworlds even outside of mathematics time. Teachers spontaneously reported this as well, with special mention of children whom they found hard to engage in regular lessons. We report our experiments and observations in the spirit of sharing the ideas and promoting more research.

Keywords Elementary school mathematics · Programming as expressive language · Microworlds

✉ E. Paul Goldenberg
pgoldenberg@edc.org

[1] Education Development Center, 43 Foundry Ave, Waltham, MA 02453, USA

[2] The Rashi School, 8000 Great Meadow Rd, Dedham, MA 02026, USA

Most digital manipulatives for mathematics function in one of three ways. Some are metaphorically physical, even if they have properties that are not easily instantiated physically: e.g. 'negative chips' that automatically annihilate 'positive chips', as a way to illustrate aspects of arithmetic with signed numbers. Others only exhibit properties of physical tools, but enhance or restrict manipulations, or help track results: e.g. virtual Pattern Blocks that are not scattered or dislodged as you work, or that draw explicit attention to transformations by allowing only certain ones. Thirdly, some manipulable environments, like *The Geometer's Sketchpad*,[1] have no physical analogue, but still involve direct physical manipulation of elements of the objects of interest.

In this article, we report on a project that explored programming as a virtual manipulative, involving: (1) indirect manipulation via programming, *influencing* objects of interest; (2) program blocks themselves as the manipulative. We are building microworlds where children give instructions to a virtual manipulative – in the tradition of Logo's instructions to a 'turtle' – but with mathematical objects reminiscent of the common physical manipulatives or images used in school mathematics. We believe that programming may help make more explicit the link between the visual/physical properties of the manipulatives and the mathematical objects they symbolize.

> Children must be able to see the manipulative as a symbol for a mathematical idea. This may be why using "bland," compared to realistic, manipulatives are more likely to serve as symbols, even for children as young as preschoolers (Sarama & Clements, 2016, p. 81)

Through iterative design guided by close observations of children and consultation with teachers, we built four microworlds in order to try to understand how children's programming *about and for the sake of* mathematics supports (or detracts from) their mathematics learning. In our experiment, elementary school children – so far, mostly second-graders in their regular mathematics classes – programmed as an integral part of their mathematical learning. The visible enthusiasm of the children and teachers pleased us, of course, but that could easily be solely attributed to change from the routine – something new and different (and not downright odious). What really captured our interest, however, was a growing collection of surprises – observations we had *not* anticipated – that suggest that this kind of programming can supportively alter children's mathematical thinking.

(1) *Abstraction*: more (or just quicker?) internalization of the manipulations they made with the programming blocks than we see with the manipulation of physical objects. Children may build mental abstractions more readily when they build programming abstractions than when they do comparable work with physical manipulatives or on paper.
(2) *Precision and clarity of thought and communication*: clearer (or just earlier?) verbal articulation of their mathematical actions, perhaps because their explanations in English were scaffolded by already having 'explained' the process to the machine through programming. When children assemble blocks of code to solve problems, they appear to have readier access to the steps they took (the algorithm)

[1] See also: https://blogs.ams.org/matheducation/2019/03/18/interactive-images-pictures-for-the-minds-eye/

and the train of thought behind their solution, potentially making it easier for them to articulate their thinking when asked to explain. This contrasts with what we see in children's use of physical manipulatives.

(3) *Proof and problem analysis*: analyzed working programs are, themselves, constructive proofs. Age-appropriate experiences with proof in elementary school are plausibly 'habit-forming', building an inclination to prove and an understanding of the role and (some) forms of proof students will encounter later. Plausible, yes, but will it hold?

(4) *Mental representation*: the ability to program the objects and algorithms that elementary students encounter as they learn to manipulate fractions and multiplicative operations might help them build better *mental* representations and make some of the typically difficult concepts more accessible. This seems to have been the case, but, so far, it remains conjecture.

Background

Our overall project was inspired by two earlier models for integrating programming into elementary school mathematics, one pioneered in Bulgaria and one (*ScratchMaths*) from the UK. The 1980s Bulgarian idea was that students could, over time and in age-appropriate contexts, learn to express and explore mathematical ideas smoothly in a computer language – their adaptation of Logo. That language would complement fluent expression in their natural language and description in mathematical notation. Starting in elementary school, students learned age-appropriate functional programming both in language and in mathematical contexts, and their school text, език и математика (*Language and mathematics*), creatively intertwined the two subjects.

Students composed elementary mathematical operations to make and explore simple functions and composed elementary language operations to explore grammatical constructions the same way. Starting early, infusing programming regularly into the children's other learning, and sticking to *one* computer language, gave students sufficient programming fluency that, by middle school, they could usethe language expressively, creating more complex programs to investigate geometry and algebra (Sendova & Sendov, 1994; Sendova, 2013).

The UK *ScratchMaths* project (from University College London's Institute of Education) involved children in a two-year experience in mathematics and programming, built on the premise that mathematics becomes more accessible when children have better language options – explicitly including an appropriately designed computer language – with which to express their mathematical ideas. A randomized-control trial in over 110 elementary schools across the UK evaluating the effect of learning to program on children's computational and mathematical thinking at grades 4 and 5 has shown significant impact (Benton et al., 2018a; Benton et al., 2018b).

Both projects revealed that, when suitably designed, programming had the potential to help students develop mathematical understanding *within the context of a school curriculum*. The way we articulate the theory to ourselves is that programming, unlike notation on paper, is a 'live' language; notation on a computer can be run, giving feedback on what it says. This is much like how we learn, and learn *through*, natural

language. We dialogue. We say things, others react, and we see what effects we have created. Conventional algebraic notation on paper just sits there, correct or incorrect, 'dead'. It gives no feedback unless we can rerun the code mentally.

Integrating Programming into Elementary Mathematics

Language develops through use and over time. And, of course, mathematics grows over time, ideally building on earlier experience. So, the real test of our idea would start early, with second-graders, and would assess growth both in mathematical and in computational thinking (CT), and in breadth and facility in their programming at the end of fifth or sixth grade. The design requirement, then, is a coherent progression of programming skill and a coherent view of mathematics, synergistically co-ordinated in a way that fits naturally into what schools and teachers already expect in their mathematics classrooms. Both CT and mathematical reasoning must grow in developmentally appropriate ways. At each point, they must be laser-focused on current goals – both to support children as they face externally-imposed hurdles like tests, and to have credibility with principals and teachers. The must also, ideally, foreshadow mathematical ideas to come.

Guiding principles include:

- focus on most critical mathematical content for the grade;
- center locus of control and authority in the child;
- use an accessible programming language that supports mathematical thinking;
- design for high cognitive demand, low cognitive distraction;
- design so that each environment can, with only small variations, serve multiple grades;
- create 'classroom-safe' environments, easy to enter with very brief, teacher-led introductions, while not requiring special teacher knowledge and intervention.

For second grade, we focused on addition and subtraction on the number line, base-ten arithmetic, navigating on a co-ordinate grid, and arrays as a foundation for multiplication. The children experienced programming strictly as sequences of commands to tell the computer what to do, but they had already learned to create new 'words' for the computer, compressing what they had originally created as sequences of commands into new single instructions. For third- and fourth-graders, the number line is extended to include fractions and decimals, and the programming begins to include functional programming – the creation of composable function machines that process input numbers and output results that other function machines take as inputs – to accord with their growing mathematics, with particular emphasis on multiples and factors and associative multiplication.

Choice of Language

We selected a blocks-based language so children could focus their attention on mathematics, not on semi-colons. Our choice of Snap*!*, in particular, was so children could create functions like 

and 

, and compose

them like 

. While algebra's function notation $f(x) = x + 10$ is inappropriate (not to mention unacceptable) in elementary school, the *idea* of function is not. Children regularly encounter processes that produce a single output for each given input in elementary school's ubiquitous input–output tables. Being able to *create* and *compose* objects with such behavior could let students explore the ideas and extend the arithmetic that they were intended to be taught.

Our own needs in designing the microworlds also influenced our switch to Snap*!* All blocks-based languages provide some blocks (e.g., **repeat**) that take *code* – blocks and scripts (block assemblies), not just numbers, words or lists – as input. For our microworlds, we needed to create other kinds of blocks (like **combine steps**, explained later) that take code as input (see Fig. 1), and we needed to create some behind-the-scenes tools that required recursive functions.

Finally, to adapt the programming environment for very young, first-time programmers, we needed a 'safe' environment. We would not 'dumb down' the programming – after all, we wanted children, even early on, to be able to create their own *functions* with outputs and, over time, to have the full expressivity of a powerful language. But, in order to acquire that language, the first contacts must not be overwhelming or distracting. Minimally, we needed to present all necessary blocks, regardless of color, in the same palette. In our first trials, children still stumbled – out of curiosity or by pure accident – into places that neither they nor their teachers knew how to extricate them from. We also had to hide unused palettes, the sprite corral, as well as all features, controls and menu options not yet required. 'Safety' also required creating new features. Bernat Romagosa, of the Snap*!* development team, built capabilities into it with which we could tailor each new microworld, offering increased power and options as children needed them.

The Four Second-Grade Microworlds

All four are engines for extensible microworlds with common features. For brevity, we describe only the variants we researched with second-graders, with brief notes on possible variants for other grades and uses. The NUMBER LINE is described in greatest depth as a way to illustrate general features of all the microworlds.

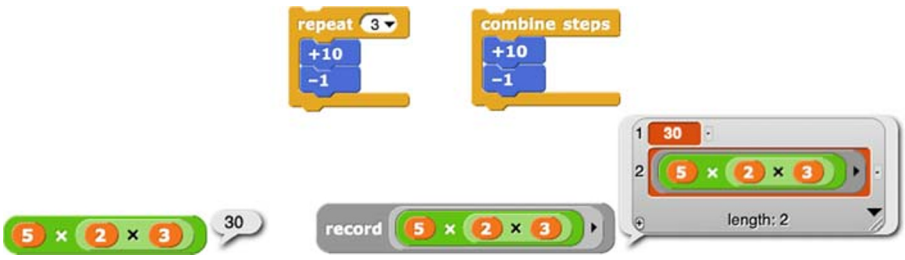


Fig. 1 Examples of blocks that take code as input: (top) two blocks that take scripts as inputs; (bottom) a child's script for making 30, and a tool we built to allow recording both code and result

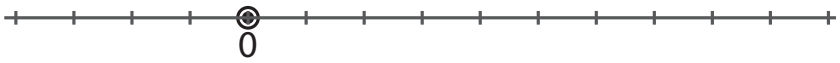


Fig. 2 A number line with ticks representing consecutive integers

Number Line Microworlds

Adding and subtracting on the number line is grade two content, as specified in many US state standards (and the Common Core). The idea – and this is critical – is not to use the number line as a 'crutch' for getting the answer, though students may initially use it that way, but as a tool for building an image of the *meaning* of addition and subtraction.

We display a number line with ticks that mark regular intervals, but interval size (and overall range) is adjustable (consecutive integers, eighths or other fractions, skip counting by any amount, starting anywhere). For seven-year-olds, the line is simple.[2] Ticks identify consecutive integers, but only 0 is labeled, intentionally chosen not to be left-most on the line (see Fig. 2).

The child starts with a very spare but playful environment (see Fig. 3): on the left is a palette containing five programming blocks[3] that can be dragged into the center 'scripting area' and used alone or snapped together to make a script. Clicking on a block or script runs it. That left-hand palette also contains six buttons that can be clicked, but not dragged out – they are not part of any program. Five of them let the child change puzzles, while the sixth (used much later) lets a child turn a script into a block. On the right is the stage with the number line and Dino suggesting a starting activity.

Because the environment is so spare, preparing a class of total novices for independent work at their own computers takes no more than a highly interactive, five- to eight-minute introduction on the rug. The teacher invited a child to read the puzzle; the

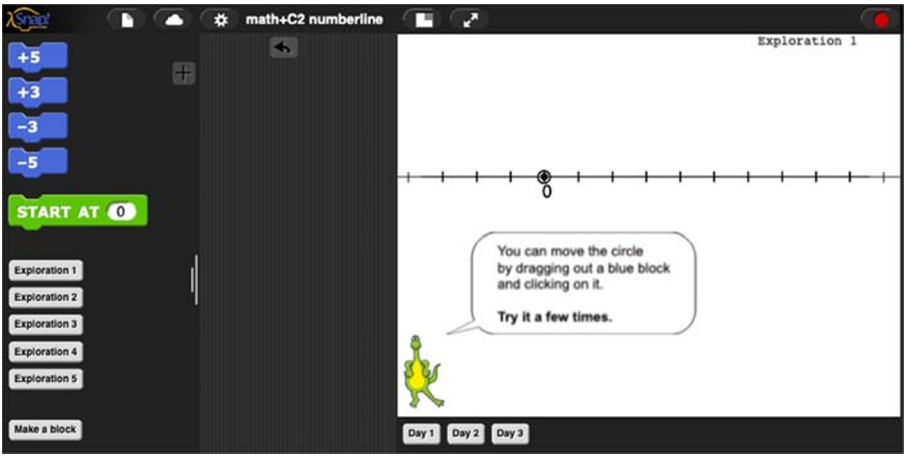


Fig. 3 A very spare environment for seven-year-olds' first experience programming in mathematics: programming tools, mathematics and images differ in each microworld, but all are this focused

[2] See: https://go.edc.org/MW-number-line
[3] This refinement is a further simplification of earlier versions, reported elsewhere, that had more blocks.

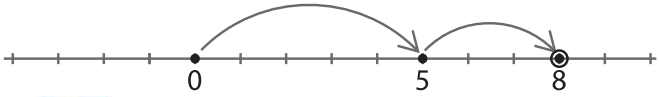


Fig. 4 A move of +3 from 5 to 8

teacher then demonstrated dragging a block and, separately and explicitly, clicking on it. Another child was invited to choose a block, drag it out, click on it and explain what happened. Fig. 4 shows the number line as it appeared after the teacher has dragged and clicked +5 and a child has dragged and clicked +3.

One or two more children illustrated and then, leaving the blocks in the scripting area, the teacher introduced two new programming things. "I can clean up and start back at 0 by clicking this green block START AT 0." This incomplete description deliberately left out, for now, changing the input, which children later (and regularly) discover on their own. Finally, the teacher showed how the first two blocks that had been used, the **+5** and **+3**, could be snapped together to make a *script* (a new word, used in context, but not explained further), and then clicked.

"To work on your own, you need to know one last thing: how to change puzzles. Any ideas?" Novices readily noticed the buttons for different explorations. When the puzzle was successfully changed, the teacher finished: "OK, you're ready to work on your own. Back to your computers!" If children failed to get a result they wanted, they learned to refine their work from the result they got.

The stage was always labeled with the current puzzle's name, and the children had a paper list of all the puzzle names with check boxes for "I did it!" and "I showed someone", so they could tick off each puzzle they have done, and choose the next one (or skip around if they preferred). The puzzles vary. Early ones are very open, like, "How many numbers from 1 to 10 can you label?" Even this elicited behavior we had not anticipated. While many children's early experiments followed no plan we recognized, several obviously invented their own challenges. Some were attracted æsthetically to patterns produced by the arrows, and kept restarting at 0, systematically reworking and trying to label *all* the numbers in some special way (Fig. 5).

Olivia, who had turned seven just a few months earlier, explained how she solved this puzzle. She said, "[To get to 1] I just went plus three, plus three, minus five. Then […] I just click it fifteen times." Nobody asked why she had said, "fifteen". Five clicks would do. What a great informal example of reasoning by mathematical induction! From a seven-year-old!

+3
+3
−5

Some of the two dozen puzzles (three of which are shown in Fig. 6) are more specific, requiring planning, mental arithmetic and experimentation. Some, still within this first experience in second grade, replace ±3 and ± 5 with ±300 and ± 500 on a zoomed-out number line. This felt 'new' to the children, but their expectations made them succeed and feel brilliant using such big numbers.

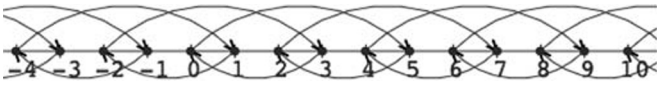


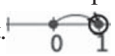
Fig. 5 All the numbers, labeled in one of the patterns we saw

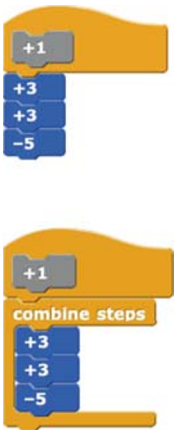
Fig. 6 Three mathematical puzzles (solutions are expressed through programming)

Teachers used class discussion to elicit and share students' thinking, and to let children analyze and explain why a script did what it did or predict a result they could not actually see. For example, given a script like Olivia's that moves from 0 to 1, one puzzle asks children "[to] predict where that script will land if you start at 19", a number that is not visible on the number line they have.

The point is to give students experiences that are a foundation for later formality: opportunities to explore and experiment, to learn from interaction (acting and seeing the result) and to use programming to express their ideas, which, we hypothesize, supports *verbal* expression of their ideas and furthers mathematical understanding. As children gained experience, and found a need for the **Make a block** button, we introduced it, individually at first, then to the whole class, once a few meaningful uses emerged. Several children, even early in their work with the microworlds, wanted to make a block.

We illustrated this with Olivia's script. Click the button, name the new block – in this case, they named it **+1** because that is what Olivia intended it to do – and drag in the script that makes it work.[4] The result was, 

a new block added to their palette for them to use. The children liked what this new abstraction did and understood it as a way to name a faithful reproduction of Olivia's script's behavior, but some of them clearly wished for a 'true' **+1** block that worked the same way that their other blocks did, drawing a single short arrow from one number to the next. This is a second level of abstraction, naming the *purpose* of Olivia's algorithm and not just its *steps*. So, we developed the **combine steps** block, which takes a script like Olivia's and produces a single arrow to reach the same end result. 



This environment has mathematical legs. It can grow with the child to serve learning in later grades; so its design permits a *set* of microworlds (plural). Just as children quickly understood the zoomed-out number line, the very same puzzle set could be used on a zoomed-in view to explore fractions. Now, replacing 

, 

, children could see blocks like 

, 

.

[4] As we began to notice children who would click **Make a Block**, but forget to drag in the script, apparently thinking that the name, alone, made it work, we changed the interface to offer a direct way to turn a script into a block. It is easier and avoids the misconception, but we are not yet convinced that it is the better choice. More research may tell us whether *facing* the misconception may be more educational.

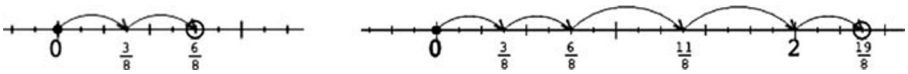


Fig. 7 **a** Adding eighths to eighths is like adding hundreds to hundreds, it just gives more of them; **b** results are always eighths, but there are some surprises mixed in, too

That the puzzles feel so familiar as to seem "trivial" is the point; fractions are just numbers too. On paper, the written notation $^3/_8 + {}^3/_8$ often pulls fourth-graders to add everything in sight, getting the canonically wrong $^6/_{16}$. But here, 

, 

and 

appear to tell the same story, leading children to *expect* $^6/_8$ (see Fig. 7a). Using $^6/_8$ rather than the reduced form $^3/_4$ helps to support their logic.

Except at integer locations, fractions in these beginning microworlds are expressed only as eighths (see Fig. 7b). Preserving the integers leaves children something to puzzle out, and something *familiar*, so children can quickly reason it out. As before, we allow accidental or deliberate excursions to the left of 0, but no puzzle requires them: for example, one puzzle says, "Start at $^3/_4$ and then move to $^3/_8$". Children may not anticipate the solution until they type $^3/_4$ into 

and see that it moves to the correct spot but labels it $^6/_8$. This design lets students build experience and knowledge from discoveries of their own, without the punch-line or surprise being explained first. Surprise adds salience and interest in mathematics, just as it *always* adds salience and interest.[5]

The initial choice of ±3 and ± 5 is strategic: at the most superficial level, these are numbers that are small enough for students to handle, yet offer challenge and opportunities for useful learning and serve as a basis for all integers. At a deeper level, this choice contrasts with one that will be used in later grades when the available blocks might be ±6 and ± 15, from which *not* all integers can be made. Having one format that can address fractions, decimals, factors, multiples, … shows coherence in mathematics and foreshadows, in grade-appropriate ways, ideas children will likely make explicit later. A central design principle is to craft our microworlds so that essentially the *same* manipulative can serve closely related mathematical ideas at *different levels* of the curriculum, consistent with best practice (see, for example, Sarama & Clements, 2016; Willingham, 2017).

In our second-grade number line microworld, *all* of the children visited the negatives, either deliberately or accidentally. Good? Bad? Indifferent? If it left children with the I-don't-get-it feeling or led to confusions that the teacher had to field, it would have been bad, because it would have put the teacher in an awkward spot, whatever good opportunities it might also have created. Few second-grade teachers, even those who are totally comfortable with the mathematics itself, will have thought about what level of detail to use with second-graders: formal treatment of negative numbers is not appropriate. But no

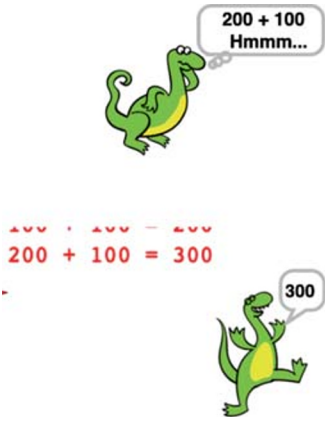
[5] We often misunderstand memory to depend on repetition. But think of a juicy bit of gossip or a frightening accident you see: one experience and you remember! Regular occurrences of something that matters can also make it salient, but it is the salience, not the repetition, that makes us remember it.

confusion ever arose to be unraveled. When the (common!) sequence **+3**, **−5** moved the little circle to **−2**, or when the children landed to the left of 0 by some other action, the children either ooohed at the negative numbers – *all* had heard of them from older siblings or some other way – or pronounced the symbol as the familiar 'minus two' of subtraction or asked only what the number was called and *then* ooohed when the response was "negative two". Nobody – none of the children – then asked any more questions. They all knew how to 'get back' to familiar territory and did not need anything else.[6] So, it was *not* 'bad' at all.

We would not make this happy accident an explicit task – it is not for second grade – but we *allow* it because it foreshadows ideas that are studied formally later and, perhaps, will make them easier. And the way children behaved in these cases even served second-grade goals. When sitting on one of the visible negative numbers, many children knew immediately where the +5 block would move them. Almost certainly they were not thinking of this as addition of two numbers, 5 and the negative number they were on; they were thinking how far past 0 that 5 would take them, composing that 5 out of two distances, both 'just plain numbers', one of which they already knew.

Physical manipulatives can also leave space on the other side of zero – floor number lines for children to hop on or small ones upon which to move a plastic frog – but that would draw deliberate, not accidental, attention to the negatives (because they would need labels), inappropriate at second grade. Programming gives access to experience without the formality.

±1, ±10, ±100 Microworlds



The tools are similar – blocks that add or subtract 1, 10 or 100 – but the range of numbers is too great for an easily viewable number line, so, in this second-grade version,[7] Dino thinks for a moment …

… and then just records the results and celebrates success. When asked to build the shortest script that would make Dino say 9, few children counted up by **+1s**; most built a **+10 –1** script. In a related, off-computer, mental mathematics task, children practiced playing Dino themselves with that two-step process applied to arbitrary two- or even three-digit numbers, e.g. the teacher said, "twenty-three"; the child added ten and then subtracted one; then responded, "thirty-two". This two-step way of thinking about adding 9 is *mathematically* important, so we provide **combine steps** to let children build a true **+9** block. We also provide a limited **repeat** block: given because children need it; limited because unlimited repetition was often an unproductive distraction. Minimizing unproductive distraction is a key design principle.

[6] For more on a kindergartener's ideas about negative, positive and 'just numbers', see Goldenberg (2018).
[7] See: https://go.edc.org/MW-1-10-100

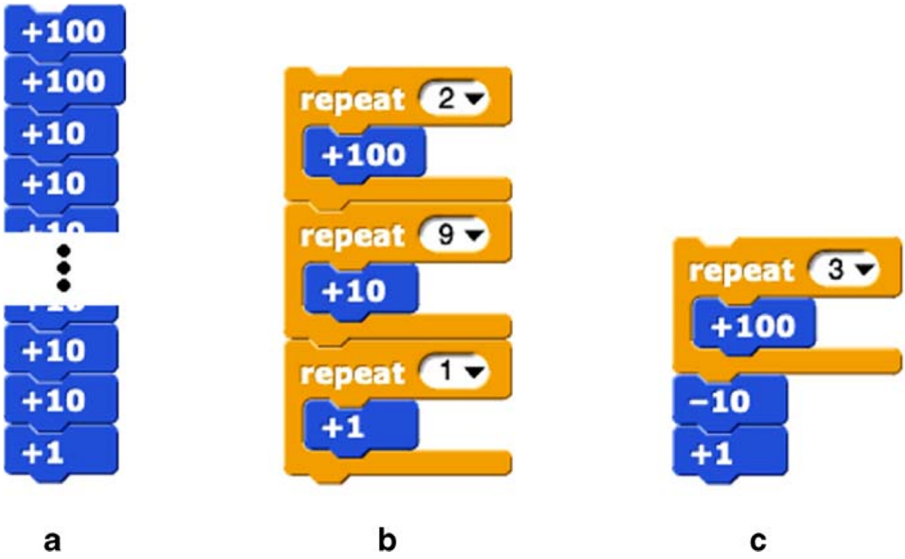


Fig. 8 Three (of many) ways to create the number 291 using only ±100, ±10, ±1: (**a**) what a child creates with base-10 blocks; (**b**) what schools teach about expanded notation; (**c**) how mental computation for adding 291 might best proceed, using rounding and adjustment

Children were intrigued (which, we believe, meant that their logic was engaged and tickled) that they could create 291 in such very different ways (as shown in Fig. 8). In fourth grade, similar puzzles might reappear with decimals ±0.1, ±1, ±10, or ± 0.01, ±0.1, ±1.

Map Grid Microworlds

Many introduction-to-coding materials[8] use movements in a grid-like space as the foundation for their first lessons. Our map grid microworlds – unlike the NUMBER LINE and ± 1, ±10, ±100 microworlds – share this feature, but with mathematics, not intro-to-coding, as the primary purpose. We rely on the power of children's curiosity and *readiness* to learn from experimentation.

The two prior microworlds needed hardly more than a five-minute introduction; at this point, the second-grade map grid world[9] (see Fig. 9) was almost self-explanatory, but, for seven-year-olds, some predicting and trying-out ideas as a group on the rug was a natural start. A 'tiny town' of seven paper-strip roads on the rug, three going one way and four crossing them, can give experience naming the roads (possibly with local street names) and describing intersections by naming the roads that intersect. Then, on the computer, the teacher points to the smiley face and says, "This is where you are now. You're outside, playing! There are places to visit." and with no further introduction invites a child to show how to "Visit Carla's home"[10].

[8] Choosing only from Hour of Code activities, we have, for example: https://lightbot.com/hour-of-code.html, https://www.kodable.com/hour-of-code#maze-maker and http://www.grinchhourofcode.com/game.html.
[9] See: https://go.edc.org/MW-map
[10] House locations are fixed, but the teacher can pre-tailor names, so that when the children first see the microworld, it names eight children in that class. Children can change names. Names used in the puzzle-tasks automatically match names assigned to the houses.

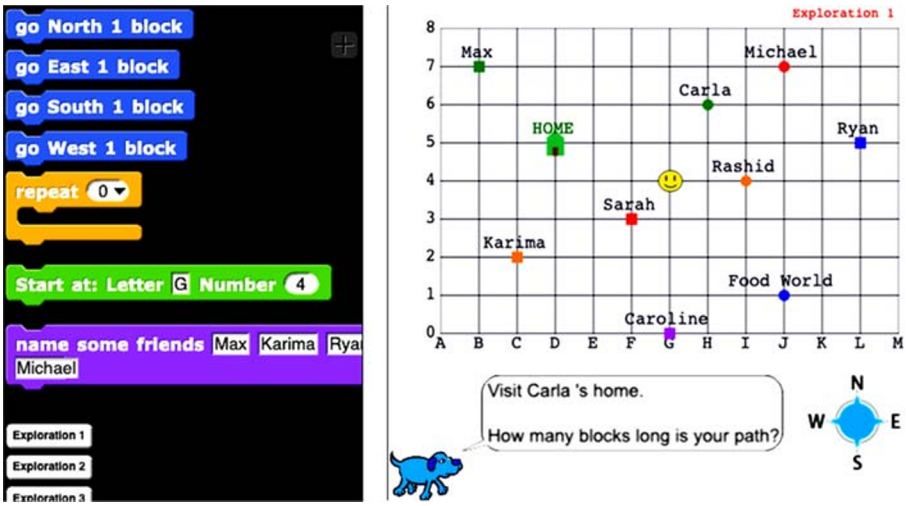


Fig. 9 The Map environment

But why *program* if the rug task is appealing and concrete and already seems enough? It may be enough to inform, but not enough to internalize. On the rug, the only source of information about the correct use of 'east' or 'west' is the external authority of the teacher. But when programming is the 'manipulative', children learn by trying things. If they choose the 'wrong' blue block, they can see what it does, use a **reset** button to cover unwanted tracks and try a different block, learning compass directions the way they learn nearly all of their natural vocabulary: by use in context.

We see the programming advantage with 

as well. The children had already twice seen blocks of this color and name; those were powerful hints at its purpose and even though its inputs were unfamiliar, they, too, contain hints. Some children spontaneously played with it[11]; some asked; some mostly ignored it, as it is not core content. But *playing* with it tells the child what it does. It does not need pre-explaining. Physical manipulatives are different: they let a child build a visible structure, but, unlike programming, the structure gives no feedback, placing the burden for any needed feedback on the teacher or perhaps classmates.

Children want to see *their* names on the map, so we provide a way. When we ask which block might let them put their own name on the map, they regularly guess 

with no further prompting. Beyond contributing to social relevance and personalization, this feature provides another chance to rehearse the distinction between *giving* input (typing it) and *using* it (clicking the block).

Some puzzles ask children to build *and keep* scripts to get from one place to another; some ask children to look at scripts they have and see if any of them can solve a new puzzle. For example, the script that got them from Karima to Sarah can be reused to get from Rashid to Ryan. One puzzle asks them to write a script to visit everybody. Getting

[11] The fact that it has not been explained may be an incentive for some children to play with it. See, for example, Schulz and Bonawitz (2007).

Fig. 10 Cookies on plates: commutativity is a miracle

from Max to Michael creates a very strong spontaneous incentive to use the limited **repeat** block.

First-graders have also excitedly used this microworld. Grade 5 students would use co-ordinates more.

Array Microworlds

The US state standards for second-graders include working with, "equal groups of objects" and "objects arranged in [small] rectangular arrays", in order to build "foundations for multiplication". These two foundations emphasize different properties. The image of cookies on plates (see Fig. 10) is easy to perceive, but the fact that three plates of four contain as many cookies as four plates of three – the essential commutative property of multiplication – appears as a miracle; we can count and check, but no logic is exposed. The array image – cookies in the baking pan (see Fig. 11) – makes it completely clear that $4 \times 3 = 3 \times 4$ Fig. 12.

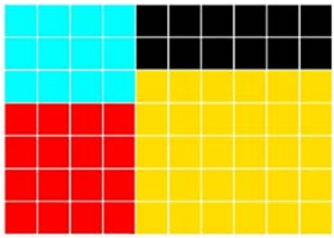
But if, as Clements points out (Goldenberg & Clements, 2014), young children do not always perceive the underlying row–column structure, this image may not be clear either! Perhaps experience creating the arrays of tiles – not by shoving them together, but by articulating the explicit lengths of rows or columns and the explicit number of them – would help. This, and our overall strategy of providing experience (and puzzling through) before formality, was the mathematical motivation for the array microworld.

Though the first three microworlds could be introduced in a different order, the ARRAY microworld significantly advances the programming ideas that children face. It thus relies on earlier learning.

Still, *with* that prior learning, getting started needed little introduction. "Exploration 0" says, "Try each of your blocks and see what you can do". Children experimentally determined what each block does. The logical challenges involved

Fig. 11 Cookies in arrays: no miracle at all that $4 \times 3 = 3 \times 4$, since baking pans can be turned any way round

Fig. 12 Blocks in the second–/third-grade array microworld



in this programming, though, took more time. In NUMBER LINE and 1–10–100, order hardly mattered. In MAP GRID, some puzzles called attention to order. But here, in the Array microworld, *sequence* became central to the programming. We wondered whether the programming challenges might eclipse the mathematics. Early observation with seven-year-olds showed that children were entranced by the colors and, through spontaneous play, by drawing their initials and other patterns on the stage (not ideas we proposed), many children grasped the navigation quickly, making programming challenges manageable. Observation also helped us shape the mathematical experience: puzzling through tasks that set constraints on the figure to be produced – observing dimensions or using total area (expressed as the number of internal squares), or needing to fill a region in some complex way. It also helped us to decide which blocks (like **move right/left**) should take inputs and which (like **move up/down**) should be absolutes, which was initially not an obvious choice.

As always, puzzles graduate from exploratory and low threshold to ones that involve challenge, but also invite creative flexibility. On the programming side, some are best solved with repeated *sequences* of blocks. And some asked children to **Make a block** for potential use in later puzzles that call for building $n \times m$ arrays. Though we created puzzles in sequence, and introduced a single new challenge at each step, children remained free to ignore our sequence and explore on their own.

This microworld expanded second-graders' programming, algorithm-building and abstraction skills, and appealed to children in several grades, but its utility *for mathematics* in later grades remains uncertain. While arrays model two-digit-by-two-digit multiplication well, it was not clear that *programming* such models added anything mathematical. This remains to be explored.

Discussion

Designing materials to address specific learning goals sets the agenda, automatically removing some agency from the student. In print media, the order and structure are set; the student's role is to respond. Tutoring systems add flexibility, but initiative still resides in the tutor; it asks, then the student responds and the system determines the path. Manipulatives – physical, virtual or programming – are different, almost inevitably calling for some 'messing about' (see Hawkins, 1965). They leave room for accidents – some messy and some happy – along with whatever deliberate purpose they were meant to serve.

Programming gives a programmer a lot of agency. Our microworlds offer students room to direct their own creative experimentation. The design challenge, then, is how to support a specific, externally imposed agenda of in-school mathematical learning, while leaving room for unscripted 'happy accidents' that enhance or foreshadow, like briefly visiting negative numbers (intentionally or by error) or discovering multiple ways of making 291.

Choosing *programming* as the medium for expressing and exploring mathematics adds two things, one for mathematics and one for programming. For programming, the added value is clear: children have a context, built into regular school, for developing programming facility that they can apply for their own purposes.

For mathematics, some of what programming added was part of the hypothesis that motivated our initial work. Programming is a live notation that let students articulate explicit mathematical statements – a kind of conscious intentionality that analogous physical manipulatives or visual aids do not require – and get feedback, which physical manipulatives do not provide. For example, we saw children treat drawing on a number line on paper as a distinct act to know how to perform, divorced from the mathematical idea it was meant to support. In one case, a boy (not in our current sample) who could easily add 3 to small numbers nevertheless consistently drew number line arcs containing three ticks rather than three intervals. Though the result on the line showed +2, not +3, he perceived no conflict because, "that's how you do it on a number line".

Also, programming blocks, as a manipulative, are more flexible than physical blocks. Physical base-10 blocks do not have negative correlates. One could invent them – for example, a set of red blocks identical to the blue ones – but, as with extending physical number lines below 0, it becomes more contrived and far more difficult to manipulate and manage. The *metaphor* of 1-bits, 10-rods, and 100-flats is their physicality. Ten bits make a rod; ten rods make a flat. 'Negative' bits or rods or flats make no more sense than negative apples. They are *inherently* not physical – that metaphor, like the red-and-black chips, is a different idea.

But there were also some genuine surprises. The first surprise we saw was the total comfort the children had with negative numbers on the line. A more profound surprise for us was a difference in what students seemed to pay attention to and internalize, particularly in the 1–10–100 microworld.

When children built numbers like 230 and 140 with physical base-10 blocks, some arranged the blocks carefully, but most just set out the required pieces and were done. When asked what the sum of those two numbers was, they often looked *down* at the blocks and counted. But when they made a cartoon dinosaur say those numbers by constructing scripts (as shown here) with programming blocks, at least some of them also appeared to have built an internal model. When asked to predict what Dino would say when one script and then the other is clicked, the abstraction already seemed to be in their heads. They could still count the blocks on the screen, of course, by looking at the scripts, but what it looked like they were doing was looking *up* to the adult – often with a confident grin – as they said 370.

We do not yet know why, nor what prior experiences are required for children to be able to do this.[12] Are the important mathematical ideas more readily brought to conscious awareness when dragging these blocks than when picking up and putting down physical objects? Or does producing a script better impose structure – the elements seen together rather than scattered about – and demand attention? Or is the act of assembling programming blocks, *with names*, somehow more linguistic than

[12] To be even more honest, we do not yet know how prevalent this is, but it was prevalent enough for us to notice – and to leave us daring enough to report it here!

moving physical ones and therefore slightly closer to the form in which the new challenge about the sum is posed and will be answered?

A critique of physical manipulatives – for example, base-10 blocks – is that children can learn the manipulations without connecting them to place value (Sarama & Clements, 2016). By asking children to create a script that tells the computer the steps to take, the feedback is immediate and the outcome is explicitly linked to the process that created it. Is *that* why our children were more readily building mental abstractions of the mathematics? Although the *linguistic* aspect of programming – formal articulation of ideas – was part of our original motivating theory, we had not thought about its effects on children's explanations in classroom discourse, which seemed to become more articulate, perhaps because the children had *already* articulated their thinking to the machine in the form of code.

Another surprise was programming's apparent effect on first-graders' mental computation. Their teacher had worked with them to help them add 9 by thinking "add ten, subtract one". (They had even acted it out with base-10 blocks, which frankly seemed cumbersome and unclear: adding 1 ten-rod, trading it in for 10 one-bits, removing 1 one, then resorting back to just counting.) When paper-and-pencil exercises asked them to add 9 to various two-digit numbers, they all counted. In our initial trials of the 1–10–100 microworld in second grade, we had not yet invented the **combine steps** block, but when first-graders used it to build 9 and 19, five of the stronger students quickly became adept not only at adding 9 mentally to any arbitrary 2-digit number, but at adding 19 and 99 as well. We did not have the chance to check with other first-graders, but we suspect that most would have been able to learn to do what these high-flyers did. Was it because of **combine steps**, or even the use of the microworld, or was it just the extra focused attention the children got?[13] Children also seemed readily able to invent a new algorithm, e.g. for adding 98, or 8, or 18 after such play.

Is the programming advantage due to its different metaphor? Physical base-10 blocks use size/shape to identify a 10 as ten 1s, and 100 as ten 10s. Programming does not show that, but makes the denomination more easily visible.[14] Second-graders could not be expected to know what to make of a physical "–1" block, but they easily treated the 'negative' *programming* blocks as subtraction (i.e. not as addition of a negative number), because that is what they looked like. Most of the children readily saw 

as a way of making Dino *say* 9. Those who could comfortably add 10 to arbitrary two-digit numbers without counting[15] then seemed to internalize +10, −1 as a way not just to *make* 9, but also mentally to *add* 9, perhaps because *they had invented it* and saw that it worked.

This report is worded with a lot of tentativeness – seem, may, conjecture, question marks – all obligatory, because the data are still coming in. But there are things we already know with certainty. *All* of the second-graders we have worked with – over 300

[13] All we know at this point about the **combine steps** block is that it was clearly very satisfying to the children – they *liked* the fact that it compressed multiple steps into one action. Perhaps that, in itself, had a meaning.
[14] In other work, we have watched children *count* the individual segments of the 10-rod, as if its physicality *did not* convey its meaning. Also see Sarama and Clements (2016) on bean sticks.
[15] Not all the second-graders built that skill (and Goldenberg et al., 2015, describe a bright ninth-grader who had not), but nearly all added 10 comfortably in the context of counting by 10 (adding 10 to a multiple of 10).

in their classrooms and many more in informal settings – and the first-graders, and even fifteen sixth-graders, are enthusiastic, puzzling hard, posing their own challenges to themselves, and thoroughly engaged. We know that the tasks they are engaged in are fully focused on mathematics, not overhead nor distraction.

Why first and sixth grade? We tested above and below grade level assuming we would need to calibrate our tasks. Instead of 'calibration error', we found surprises. When we tried the microworlds tentatively with end-of-the-year first-graders, they were as interested and able to do most of the puzzles as mid-year second-graders. Does that mean we aimed too low for second grade? Neither the second-graders nor their teachers thought so. Was the first grade not a 'matched' class? Perhaps, but no evidence we have suggests that. Or does programming, or exploration, help kids do things earlier than they otherwise might? We do not know. And sixth-graders definitely 'knew the math', but looked with greater depth. Learning through programming is just *different*.

Our work with students has already led to a better understanding of children's mathematical thinking, resulting in many refinements of the instructions and microworld environments. Given the exploratory nature of this project, we, of course, continue to work to understand some observations. One puzzle that nearly all students found very hard was "START AT 120. Build a script that makes Dino say 420." Even with Dino *already* saying 120 – and with the START AT block that let them *make* Dino say 120 – most children built scripts that *made* (which is to say *added*) 420, rather than ones that added 300.

Generally, children had difficulty with puzzles that asked them to create results after starting elsewhere than 0. Is this a failure to understand the puzzle's wording? The initial state of Dino? The true nature of the blue blocks? Does this mean that working with *physical* blocks might have been better for the children, at least initially? *Surely*, if 120 were already represented on the floor in physical blocks, children would respond only by placing the additional 100-blocks with it. If the screen had virtual analogues of the physical blocks, children would likely respond the same way. If, with physical blocks, the puzzle stated "*if* 120 were already on the floor, show what else you would need to make 420", would they still understand? Would that have been better than START AT? We do not yet know. Perhaps the extra focus needed to solve these puzzles in the programming environment *enhanced* children's learning, but without further study we could not say that it did not just take up extra time.

All else being equal, the enthusiasm and engagement alone are clearly a contribution; the flexibility allows more variation in mathematical tasks; and the focused time on task means, at the very least, that the learning time for mathematics is not diminished. What we are now trying to learn is how any of this might translate into positive effects that can be seen clearly in the standard measures of mathematical growth on which teachers, districts and the research literature rely.

Our own research continues. Our microworlds are freely available and we are eager for others to research similar questions.

Acknowledgments Funding for doing and reporting the work described in this paper was provided, in part, by the National Science Foundation, grants 1934161 and 1741792. Views expressed here are those of the authors and do not necessarily reflect the views of the Foundation.

Compliance with Ethical Standards

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

Benton, L., Kalas, I., Saunders, P., Hoyles, C., & Noss, R. (2018a). Beyond jam sandwiches and cups of tea: An exploration of primary pupils' algorithm-evaluation strategies. *Journal of Computer-Assisted Learning, 34*(5), 590–601.

Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018b). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child–Computer Interaction, 16*, 68–76.

Goldenberg, E, P. (2018). Ideas under construction: Children saying what they know. (https://blogs.ams.org/matheducation/2018/09/02/ideas-under-construction-children-saying-what-they-know/).

Goldenberg, E. P., & Clements, D. (2014). *Developing essential understanding of geometry and measurement for teaching mathematics in pre-K–grade 2*. Reston, VA: National Council of Teachers of Mathematics.

Goldenberg, E. P., Mark, J., Kang, J., Fries, M., Carter, C., & Cordner, T. (2015). *Making sense of algebra: Developing students' mathematical habits of mind*. Portsmouth, NH: Heinemann.

Hawkins, D. (1965). Messing about in science. *Science and Children, 2*(5), 5–9.

Sarama, J., & Clements, D. (2016). Physical and virtual manipulatives: What is "concrete"? In P. Moyer-Packenham (Ed.), *International perspectives on teaching and learning mathematics with virtual manipulatives* (pp. 71–93). Cham: Springer.

Schulz, L., & Bonawitz, E. (2007). Serious fun: Preschoolers engage in more exploratory play when evidence is confounded. *Developmental Psychology, 43*(4), 1045–1050.

Sendova, E. (2013). Assisting the art of discovery at school age: The Bulgarian experience. In P. Sánchez-Escobedo (Ed.), *Talent development around the world: A global perspective on gifted education* (pp. 39–98). Mérida: Lambert Academic Publishing.

Sendova, E., & Sendov, B. (1994). Using computers in school to provide linguistic approaches to mathematics: A Bulgarian example. *Machine-mediated Learning, 4*(1), 27–65.

Willingham, D. (2017). Ask the cognitive scientist: Do manipulatives help students learn? *American Educator, 41*(3), 25–30 40.