Harvey, B. (2022). Reflection: Twenty things to do with Snap! *Contemporary Issues in Technology and Teacher Education, 22*(1), 218-223.

Reflection: Twenty Things to Do With Snap!

Brian Harvey

Editor's Note: Brian Harvey and Jens Mönig jointly developed the educational programming language Snap*!*. A parallel computer science course, Beauty and Joy of Computing, which is tightly integrated with Snap*!,* is endorsed for Advanced Placement CS Principles credit by the College Board. With support from the National Science Foundation, this course has been taught by more than one thousand high school computer science teachers. For this and other accomplishments, they have received the National Technology Leadership Summit (NTLS) *Educational Technology Leadership Award,* which recognizes individuals "who made a significant impact on the field of educational technology over the course of a lifetime."

In 1971 Seymour Papert and Cynthia Solomon published "Twenty Things to Do With a Computer" (MIT Artificial Intelligence Memo No. 248 / Logo Memo No. 3). Some of the activities suggested included use of a turtle to create art, use of a computer to create music (using Logo music extensions developed by Jean Bamberger), creation of a computer-generated light show, creation of computer-generated poetry, and construction of computer-controlled puppets. The original publication is republished in the Seminal Articles section of this issue of the *CITE Journal*.

Design of an educational computing language such as Snap*!* does not occur in a vacuum. It is influenced by past developments and by the needs of future generations. This reflection is written from that perspective on the occasion of the 50th anniversary of "Twenty Things to Do with a Computer."

The real subtext of "Twenty Things" was "Twenty things to do *by programming* a computer." After all, in practice one of the most beloved things my high school students did with our computer was use its text editor to write their English papers, so when their teacher made them correct the errors in it, the task was not torture as it was before computers. That detail is not in "Twenty Things," however.

Some of the activities suggested in "Twenty Things" are still thriving 50 years later, even more so as technology improves. Back then, robot turtles had to be tethered to the computer with cables that got twisted and limited motion. Now, there are Bluetooth and Zigbee tetherless robots, and with Micro:Blocks you can even download a Snap*!* program to a robot and have it be fully autonomous.

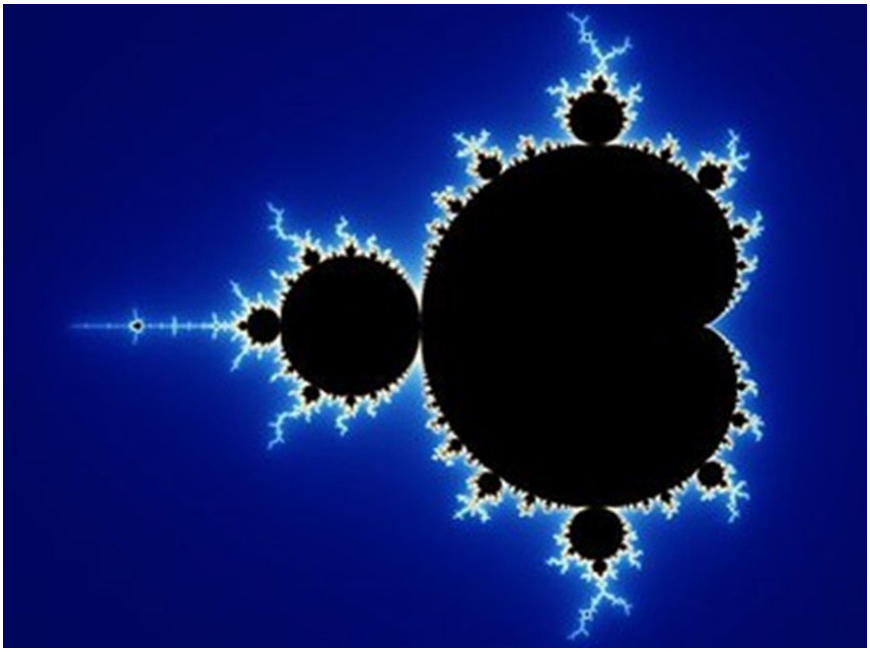
Many of the things listed in "Twenty Things" can be done better today *without programming* and, therefore, would not be in today's list. Making music in Garage Band is way cooler than programming Jeanne Bamberger's music box in Logo.

Yet, surprising exceptions exist. Anybody can draw a square in a modern drawing program with a simple click-and-drag. We thought that feature would kill turtle graphics, which was part of the impetus behind giving sprites costumes and using them to tell stories. It turns out kids are still excited the first time they draw a square in [insert programming language here]! And squirals (Figure 1) are just as magical as ever.

Figure 1. *Squiral.*

Fractals remain, too, although computers are now fast enough to draw Mandelbrot sets point by point, so you can program more beautiful ones using programming techniques that do not illuminate recursion the way trees and Koch snowflakes do (Figure 2).

Figure 2. *Fractal.*



Really redoing the list *in the spirit of the original* requires paying attention to "what do you learn by doing this?" For Seymour Papert, it was all about mathematics. Turtle graphics teaches finite differential geometry, concrete poetry teaches production grammars, and so on. Of course, the twenty things were *also* supposed to be fun!

So, in some respects, the list of things to do in Snap*!* would not be all that different. The currently exciting math has shifted to some extent, from geometry to statistics, because of machine learning. So artificial intelligence (AI) projects would be good to include, preferably projects that really teach a simplified but honest understanding of computational neural networks.

Robotics, too, would still be on the list but maybe in different forms. The FIRST robotics program (https://www.firstinspires.org/robotics/frc) has raised the bar tremendously on the hardware side. My (third-hand) understanding is that their teams do the software side with a lot of copy-pasting, and not so much understanding or originality as Logo teachers would want to see. Even in the old days, however, trying to drive a robot turtle up a ramp taught things about friction and, more generally, the uncooperativeness of physical reality with respect to theoretical understanding.

Kids write 3D ray tracing programs in Snap*!*. I find that level of attention to detail beyond my level of patience (not to mention competence), but the activity has plenty of good mathematics. It is well beyond the speed of the early personal computers.

Music, which I rejected earlier, also has some good mathematics in it: the Pythagorean investigation of musical intervals as rational numbers. In Jeanne Bamberger's early work with Logo, the notes were taken as primitive building blocks, not looking inside the sine waves. I would be happy with a music microworld that taught the beginnings of music theory.

Snap*!* continues to evolve as part of an iterative dialog between developers and the user community. A contemporary list of things to do with Snap*!* is found in many forms and locations:

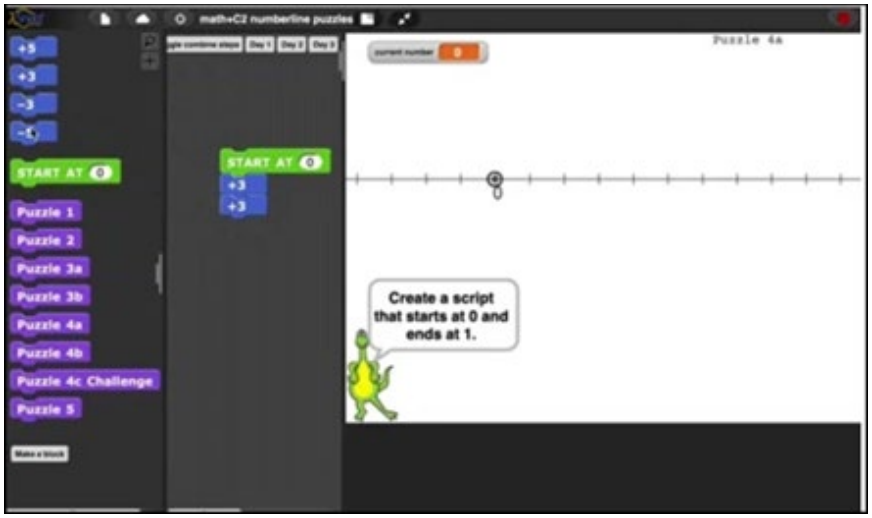
Joachim Wedekind's materials for *Art Across the Curriculum* (http://digitalart.joachim-wedekind.de/ueber-das-buch/) take the form of a book – the old-fashioned paper kind (Figure 3).

Figure 3. *Art Across the Curriculum Book Cover.*

Snap*!* lessons (https://open.sap.com/courses/snap1-1?locale=de) developed by Jens Mönig and Jagda Hügle consist of videos embedded in a Massive Open Online Course (MOOC; Figure 4). The media arts curriculum developed by the *Make to Learn* team led by Glen Bull lives in threads of the Snap*!* Forum. Paul Goldenberg's early-childhood math curriculum (https://elementarymath.edc.org/) centers on a series of Parsons Problems embedded in Snap*!*, but Snap*!* itself is hardly visible in them (Figure 5). Other Snap*!* activities are likely to live inside a Course Management System (CMS) such as Canvas.

Figure 4. *Screenshot from MOOC on Snap!*

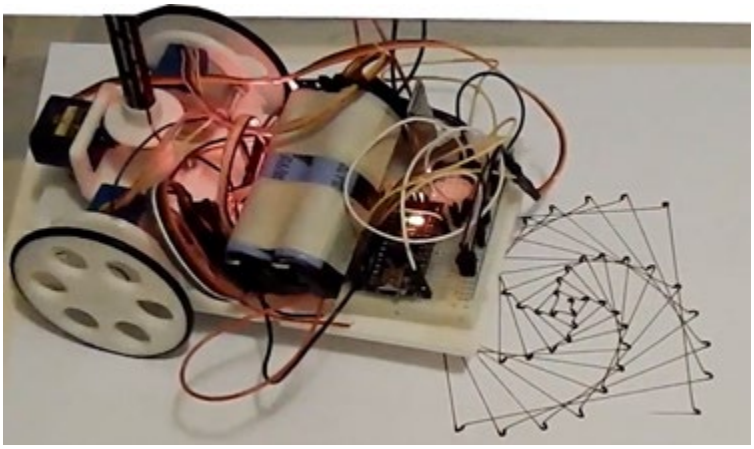
Figure 5. *Screenshot from Goldenberg's Early Childhood Math Curriculum*



The user communities that form around an idea are as important as the tool and associated activities and curricular materials. For example, TurtleStitch (https://www.turtlestitch.org/) is an extension of Snap*!* that enables users to create embroidered patterns using geometric designs (Figure 6). Snap4Arduino (http://snap4arduino.rocks/) is an extension of Snap*!* that enables users to interact with devices in the physical world via microcontrollers (Figure 7). NetsBlox (https://netsblox.org/) is an extension of Snap*!* that enables novice programmers to create networked programs such as multi-player games.

Figure 6. *TurtleStitch Embroidery*.

Figure 7. *Snap4Arduino at Work*.



The last line of "Twenty Things to Do With a Computer" is a recursion line: "Think up twenty more things to do." The user communities that have formed around these extensions have generated many more things to do with a computer, including, in some cases, generation of other extensions. In this respect, this recursive generation of ideas and things to do with a computer realizes the vision embodied in the original "Twenty Things" paper published 50 years ago.