Minksky, M. (2022). Reflection: College students respond to "Twenty Things" in 2020. *Contemporary Issues in Technology and Teacher Education, 22*(2), 286-292.

Reflection: College Students Respond to "Twenty Things" in 2020

Margaret Minsky

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), republished in this issue of the *CITE Journal* (Papert & Solomon, 2022). Note: a related article, "Twenty Things to Do with Snap*!*," is published in this issue of the *CITE Journal* (Harvey, 2022).

We asked six college students in a class on constructionist learning to read "Twenty Things to Do With a Computer" and respond to a series of prompts.

The students had several weeks of introductory experience using the educational computer language Snap*!* and were concurrently starting projects in Turtlestitch, a language variant for programming computer-controlled embroidery machines. They also read portions of the book, *Mindstorms: Children, Computers, and Powerful Ideas* (Papert, 1980) and some counterpoint essays with alternative perspectives about computational thinking. In addition, they listened to a recorded presentation by Cynthia Solomon (2019). This paper presents students' interpretations through summary highlights followed by longer quotations from their written responses.

Here are three of the prompts:

- The paper "Twenty Things to Do With a Computer" was written in 1971. Are any of the topics in the paper outdated? Why?
- Choose two or three of the topics in "Twenty Things to Do With a Computer" that are available today and comment on them. (Choose areas that you could write programs for yourself, or that you could teach to beginners … using Snap*!* or any other computer programming environment that you know about).
- Write at least two questions you have for Cynthia Solomon based on her presentation or the Twenty Things paper. Why are these questions important to you?

Let's begin with "What's outdated"? "Twenty Things to Do With a Computer" was written 30 years before the oldest students in the class were born.

Their answers encompass a range of responses, from enthusiastic appeal to a critical look at areas in the paper as retro vintage. Some students took a historical perspective on the text itself. Others expressed a timeless empathy with the project of creating a beginner learning environment, while still others looked backward from their immersive literacy in the creative media and audio-visual systems that are available today. Some were surprised by the paper's contemporaneous provocations about Artificial Intelligence and understanding human thinking.

Are Any of the Topics Outdated

Message is not outdated, but never heard of Spacewar.

I don't think the message behind the Twenty Things is outdated, particularly regarding the way people in education treat computers, but I do think that Number 5 (Play Spacewar) of the twenty things is the most outdated of the things listed … and I only say that because I've never heard of Spacewar before. I've heard of games like Galaga and that old ping-pong game where you had to prevent the 'bal'" from hitting the left and right edges of the screen, but Spacewar is not one that I've heard of, so I had no point of reference to fall back on for that part of the text.

Still relevant

I don't think that any of the ideas in our reading this week were necessarily out of date.They are all things that one can program a computer to do – most of them are less alien or novel than they may have been when written about in 1971. However, the ideas and types of programs talked about are still relevant, intriguing and explored on a regular basis today …

Responses to the Tools Described

Processing's shape manipulation surpasses the simplistic shapes in the text

Connections between numeric coding and generated visuals definitely has already been eclipsed by modern day languages. Processing's shape manipulation surpasses the simplistic shapes in Twenty Things to Do with a Computer. [Processing is a visual language developed in 2001.]

A modern music box is way beyond the text

The areas where a music box is referenced is quite outdated, as computerized music has emerged over the last decade and has taken over the music industry. The capacity of the modern 'music box' is way beyond what is described in the text.

Turtle Programming is more accessible now**; Blocks are more engaging than a text language**

When it comes to the outdated part of the paper, I think what should be emphasized is that the turtle programming is quite accessible nowadays. Therefore, the topics such as Make A Turtle, Draw A Heart, etc. seem to be quite easy today. The computing language used in the paper, LOGO, seems complicated and not that interesting. Instead, the commands and blocks in Snap! are more beginner-engaging, which add visual aids to the basic black-and-white screen.

Responses Indicating a Radically Different View of what AI Means:

AI is different now, in fact, it has surpassed humans

Another thing that is quite outdated is the signifying of these codings as 'AI'. I think back in the day the article was written, in 1971, have definitely surpassed. It would be difficult to still relate to these functions as Artificial Intelligence today. As artificial intelligence of 2020 has far surpassed these boundaries, into territories that match the human being's own empathetic processes. The AI of today has surpassed even humans, and is able to develop itself.

A different view of AI from that proposed in the text

About the explain yourself part of this article, I think that the machine is very different from a person. It is very hard to use a machine to simulate human. Machine have their own sight of world which is very different from human. For example, Alpha Go made some impossible steps in the game Go, to make people reconsider the steps of playing Go.

The second prompt, asking for topics in the text that students would still like to explore today, elicited a wide range of choices: Differential/Turtle Geometry (1 student), Spirals (1 student), Debug a Heart (1 student), Growing Flowers (2 students), Make a music box (2 students), Text Generator (2 students), and Think of your own things…make a game! (1 student). The original intent of the paper shines through in the student responses. Students who are beginning programmers, as well as those with experience in contemporary Creative Coding, discuss their inspirations from the paper.

Text Generator; text "Without the Sarcasm"

One area I really like, based on personal interest as something that I could modify, in the Twenty Things to Do with a Computer is number 16 [Text Generator]. This one generates specific true statements and then taunts the person interacting with the program. One thing that I've been thinking about how to code would use similar mechanics, without being as sarcastic. I've been considering ways to create a code that would generate an I Ching hexagram for divination purposes. It would involve randomization, math, and coming to specific conclusions, i.e. drawing

specific lines, based on the results it got, and I think this program could potentially be a good place to start with that if I decided to actually code this project.

This student also elaborated:

> In the Twenty Things paper, number 16 that I mentioned earlier is similar to a very basic prototype of an AI, in that it generates responses based on user input. If we're considering this basic AI as the basis for a rich world, which I think it would in that it's similar to music which is something with specific building blocks that could potentially have a lot of outcomes, I think the divination project I mentioned would be a good example. Because while you could keep it very simple (random generator gives numbers that are added together to select a specific outcome until you reach one or two of 64 possible outcomes), you could also add in speaker prompts where it actually tells you what the hexagram means and gives you the answer within the program itself. From there, you could code the answer to be given with any attitude you want, depending on what type of personality you want your programmed oracle to have, be it sincere, blunt, sarcastic, or something else.
>
> Another area I like that is useful for teaching is number 9, which is for growing flowers. I think this one would be really easy to bring into Snap!, at least after adding a block for curved lines. It might not be the best for having beginners start from scratch with zero guidance, but if a teacher wanted to create the project with a dedicated curve block in advance or have their students brainstorm how to make a curved line using the repeat function, then I think it would give them more things to experiment with.

The student who contributed the following response enjoys using simple yet highly generative patterns, similar to the spirals that are inspiring current Turtlestitch programmers. They focussed on the imagery as "psychedelic", which is an interesting callback to the 1970's (whether it is historically rooted or has become newly interesting to this generation). In writing about their programming in the context of the Twenty Things paper, the student demonstrates that they understand that describing their program is important; they enjoyed creating narrative descriptions of programming concepts.

Psychedelic patterns and spirals

I am quite interested in using differential geometry in order to create psychedelic and geometric images for use in the turtle stitch platform. Following the entries in the Twenty Things paper, in order to create a simple geometric figure, it requires three steps.

1. Draw forward by a certain number
2. Turn at a certain degree
3. Repeat

In this very simple function, one can create a symmetrical shape by drawing forward a certain degree and then rotating the turtle. Doing this over the desired many of times suitable for the shape will create a geometric function, anything as simple as a square to a giant polyfigure. In order to teach this to kids, I think it is beneficial to provide the 1,2,3 coding structure as a base program, and then allow them to adjust the different values of the STEP, ANGLE, and POLY values in order to create different figures and learn the values of each equation by creating different shapes within the same function.

Draw Spirals requires using the same function as the differential geometry program, however with each repeat of the system, add a value to the drawing forward number in order to make the figure sequentially expand outward. This function will look something along these lines.

1. Draw forward by a certain number
2. Turn at a certain degree
3. Add a number to the value in step 1, then repeat

By altering the final step, by making the STEP value something that will change over the course of the turtle stitch, we can allow our original geometric shape to become a sequential spiral that increases the STEP value over the running of the code. In order to apply this to an educational setting, it is important to first teach the students the importance of differential geometry, and then challenge them to figure out how to make their geometric shapes into expanding spirals, they can first be given the challenge to figure it out on their own, and then more and more hints given. Allowing them to figure out how to evolve from a geometric shape from area #6 to the spirals of area #7 will give them the opportunity to make these spirals and learn on their own how they can manipulate their own turtle stitch coding.

In response to the third prompt, questions for Cynthia Solomon, here are three student responses.

> I really enjoy her talking about early history of programming language. I am super surprised that they invented a language for kids! In my understanding, at that time, computer is still a high-technology symbol. I will not let children be in charge with “high technology”. However, they chose to simplify everything, and let kids learn them. It is never an easy thing to simplify things, if you can explain something to people who know nothing about it clearly, you must understand that thing deeply. It is even harder to make a system for beginner! If I remember correctly, at that time punch card was still one of the mainstreams. I really enjoy her talking about some out of expected things during teaching kids. By outputs, not only children learn about programming, but they learned more as well. I really love her idea about letting kids to be the co-designers and collaborators.

> *So here are my questions. When I was young, I was so willing to learn about programming. However, because of the limitation of the language, I could not access to low threshold programming language. I really don't think that language should be a thing stopping people to learn. For those young children, how they learn about this language, if they are confused by the language? Also, I noticed that Logo is very much based on the English language. Also, because I am able to speak in two languages, I know that here are some logic differences while you are speaking in different languages. Since the grammar of each language is so different, do you think the native language will influence some programming learning?*

And, I really want to know why you chose to do the kid education with programming. Before I know more about turtle and Snap!, I was thinking that it is impossible for children to learn programming. How can I convince their parents, and most of their parents are also beginners of programming, is it possible to let them learn together? How about a family learning?

You mentioned in your talk that making things for machines has some limitations, and I'm curious about what you think some of these limitations are as someone who was fundamental in creating these programs? I can come up with a few limitations that are mostly inspired by my lack of experience in coding and programming, but I'm curious as to what you think these limitations would be. In that same sentence you mentioned "interesting obligations" as well, and I wanted to ask what that meant?

In the video, the host mentioned Cynthia Solomon's project to combine computing with social justice in order to encourage kids to rediscover ideas and look into the definition of the world. I am curious about how to combine those two while keeping social issues easy for the kids to understand and analyze?

Also, in reality, many patterns we program cannot fit in the Total Turtle Trip Theorem, what is the importance of the theorem and are there any applications of it in the real world? They are important to me because I'm quite impressed by the project and the theorem when hearing them in the talk.But I'd like to be more specific on their details for further exploration. I think they are closely related to the design of programming environment for kids and computational thinking skills discussed in the course.

The students' comments stem from two perspectives. One is the perspective of introductory creative media arts students using their historical, or imagined historical, contextual assumptions about the

1970's. The other perspective comes from their contemporary literacy in creative media software, and Artificial Intelligence, generating both enthusiasm for the original paper and critiques of some aspects. Their enthusiasm manifests in suggestions for continued exploration of the projects in this paper and for new project areas. For these students, at least, many of the topics addressed in original "Twenty Things to Do with a Computer" paper are still relevant today.

References

Harvey, B. (2022). Reflection: Twenty things to do with Snap! *Contemporary Issues in Technology and Teacher Education, 22*(1). https://citejournal.org/volume-22/issue-1-22/seminal-articles/twenty-things-to-do-with-snap

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.

Papert, S. & Solomon, C. (2022). Reprint: Twenty things to do with a computer. *Contemporary Issues in Technology and Teacher Education, 22*(1). https://citejournal.org/volume-22/issue-1-22/seminal-articles/twenty-things-to-do-with-a-computer

Solomon, C. (2019). *When computers were new: Keynote presentation in acceptance of the FabLearn Lifetime Achievement Award*. FabLearn 8th Annual Conference on Maker Education, New York, New York.