# WIP: SnapBots: Fostering Collaborative and Accessible Computational Thinking Education for Elementary and Middle School Students by Programming with Pencil and Paper

**Duncan Johnson, Tufts Center for Engineering Education and Outreach**

Duncan Johnson is an undergraduate student at Tufts University majoring in Computer Science. He is the co-founder and Executive Director of BX Coding, a STEM education nonprofit dedicated to building open-source software and curriculum for STEM educators. Through his work with BX Coding and the Tufts Center for Engineering Education and Outreach (CEEO), his research has focused on educational methodologies and technologies that introduce elementary and middle school students to Computer Science. Currently, he's exploring how technologies can leverage generative AI to better support educators.

**Dr. Ethan E Danahy, Tufts University**

Dr. Ethan Danahy is a Research Associate Professor at the Center for Engineering Education and Outreach (CEEO) with secondary appointment in the Department of Computer Science within the School of Engineering at Tufts University. Having received his graduate degrees in Computer Science and Electrical Engineering from Tufts University, he continues research in the design, implementation, and evaluation of different educational technologies. With particular attention to engaging students in the STEAM content areas, he focuses his investigations on enhancing creativity and innovation, supporting better documentation, and encouraging collaborative learning.

# WIP: SnapBots: Fostering Collaborative and Accessible Computational Thinking Education for Elementary and Middle School Students by Programming with Pencil and Paper

## Introduction

Computational Thinking (CT) education for K-12 learners typically progresses through three distinct programming modalities, each tailored to a specific developmental stage. Icon-based environments, such as ScratchJr, allow early elementary students (ages 5–7) to construct programs without needing to read [1]. Block-based environments, most notably Scratch, utilize colorful blocks that snap together to lower barriers for learners (ages 8–16) [2]. Finally, text-based languages like Python and Java serve as the standard for high school courses, with the goal of preparing students for employment or university coursework [3].

Despite this established pipeline, a significant pedagogical hurdle remains. The four core pillars of CT – problem decomposition, pattern recognition, abstraction, and algorithm design – are often overshadowed by the immediate need to learn specific tool-based vocabularies. In many introductory contexts, students must navigate a vast library of blocks or strict textual syntax before they can build systems complex enough to require high-level decomposition or abstraction [4]. Consequently, early CT coursework, especially for elementary and middle school students, often heavily emphasizes basic algorithm design at the expense of vital systems-level skills.

Beyond these cognitive barriers, economic and structural inequities further limit access to CT education. Many modern programming environments assume a 1:1 student-to-device ratio, but this model is often financially unsustainable for under-resourced schools and informal learning environments such as after-school or summer programs [5]. The high upfront cost of hardware, combined with the ongoing burden of maintenance and IT support, prevents many students from engaging with CT education in a meaningful way.

Even when devices are available, many computing projects fall into one of two paradigms: all students following the same highly structured curriculum or each student individually creating an open-ended project [6], [7]. However, neither of these structures mirrors the majority of professional software development, which relies heavily on modularity, negotiation, and the integration of diverse components created by different teams.

## Our Solution: SnapBots

We present SnapBots, a programming environment where elementary and middle school students create characters using hand-drawn state diagrams which are then scanned by a camera and converted to Python code using a Large Language Model (LLM). SnapBots was developed as a fork of Patch [8], a web-based Python programming environment which is itself a fork of Scratch, the block-based programming environment developed by the Scratch Foundation. This means that the characters students create in SnapBots can look like any sprite from Scratch, play most sounds from the Scratch audio library, and take any action a Scratch character can take such as moving,

saying quotes, and sensing other characters.

*Hand-Drawn State Diagrams*

The hand-drawn state diagrams, like the one in Figure 1, are made up of three components, all of which are explained via natural language. First, each diagram has a word or two at the top explaining what the character should look like, such as “Cookie,” “Princess,” or “Frog.” Second, each diagram is made up of one or more states, which are circles with words written inside of them. States are actions that the character can take, which can be described using as few words as “move forward” or as many words as “move so that the character is slightly to the right of the soccer ball.” Third, each diagram has transition arrows that denote when the character should change from one state to another. For example, an arrow pointing from state “move forward” to state “turn left” might say “when facing and close to the wall” so that the character turns before bumping into the wall.

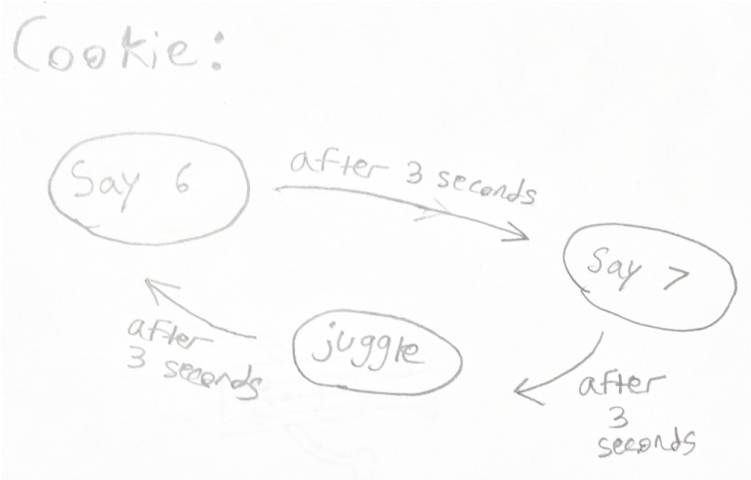


*Figure 1.* A 6th grade student’s state diagram for a Cookie character inspired by the 2025 internet meme “6-7”

By abstracting the programming process into a series of natural language states and logical transitions, SnapBots allows students to engage with problem decomposition and abstraction from the very beginning of their experience. In traditional block-based environments, a new student must spend time hunting for a specific “change x position by 10” or “if touching color blue” block, effectively stuck in the weeds of low-level algorithm design. In contrast, the state diagram approach requires students to first conceptualize the high-level “states” of their character, such as “chasing the ball” or “celebrating,” and then define the conditions that trigger a change in behavior. Because there is no fixed vocabulary to memorize and the syntax is limited to hand-drawn circles and arrows, students are free to focus on the logical structure of their system rather than the mechanical constraints of the interface.

Because each student’s coding happens with pencil and paper, SnapBots fundamentally changes the hardware requirements of the CT classroom. In a resource-constrained environment, an entire class of students can work simultaneously on their desk surfaces, iterating on their logic and

drawing their characters without needing to wait for a turn at a computer. This "unplugged" creation phase encourages students to treat their programs as tangible, shareable artifacts. If one student creates a diagram that another student wants to take inspiration from, the first student can simply hand their sheet of paper to the other student and continue programming on a different sheet of paper, allowing for knowledge sharing between students without complex user accounts or sharing permissions. Then once a student wishes to see their character come to life, they can bring their diagram to a single shared computer or document camera to capture an image and generate a character, making CT education viable in spaces where a 1:1 device ratio is physically or financially impossible.

*Collaborative Design and Common Goals*

The physical nature of the state diagrams also facilitates more natural intragroup collaboration. Unlike a screen, which often gives priority to the student holding the mouse, a piece of paper on a shared table provides a canvas for multiple students to point, draw, and negotiate logic simultaneously. When task-oriented environments are introduced, such as a soccer environment with a ball and goals, groups must use abstraction to determine how their character will interact with others. This encourages collaboration as students attempt to achieve goals together: one group might create a "striker" character who tries to score, while another might create a "goalie" that defends their goal. Each student doesn't need to create an entire soccer team; they only need to create one player and rely on their classmates to build other roles.

*The Role of Generative AI in Character Creation*

To bridge the gap between hand-drawn diagrams and functional software, SnapBots utilizes a multi-step generative AI pipeline powered by an LLM to translate hand-drawn state diagrams with natural language into executable Python code.

SnapBots first processes the uploaded image to identify the structure of the graph. It extracts the character's intended appearance and maps every circle (state) and arrow (transition) into a specific function contract. The goal of this step is to understand the structure and high-level goals of the student's diagram before any code is written.

Once there are function contracts for each state and transition, SnapBots then makes parallel LLM calls to generate the code for each state and transition. To reduce the potential for programming errors, the model is provided with a set of human-written primitive functions such as "move forward" or "distance to wall" that it can rely on when creating new functionality. These primitives act as a bridge to the underlying Python environment, and reduce the total amount of new code that the LLM must generate. During this process, the LLM is provided with environmental context, such as the names of available sounds, the list of other sprites in the scene, and any global variables that it can reference. This allows the AI to write code that enables complex interactions, such as "move towards Sally" or "cheer if the ball is near the goal."

Once the character's code has been generated, SnapBots' interface, as seen in Figure 2, shows the character's current state on a digital version of the student's diagram in the bottom left, allowing

them to visually see the execution of their logic as their character takes actions in the environment on the right side of the screen.

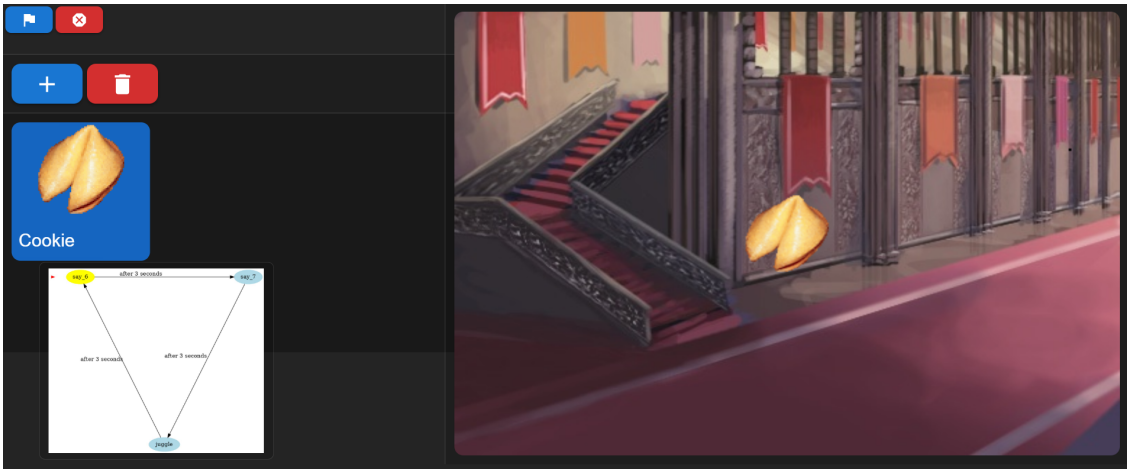


*Figure 2.* The SnapBots interface with the Cookie character from Figure 1

**Research Methodology**

The research included two interventions within a 6th-grade classroom at a public charter school in Boston, Massachusetts. While a primary design goal of the SnapBots platform is to facilitate programming in learning environments without significant device access, this pilot leveraged the school's existing 1:1 student-to-laptop ratio, allowing each student to engage individually with the SnapBots interface for more frequent iteration with their state diagrams. Twelve students participated in the research study. The research was approved by an Institutional Review Board (IRB).

The study was divided into two sessions, with the first in December 2025 and the second in January 2026. During the first session, which lasted 45 minutes, students were introduced to the SnapBots platform and the state diagram coding methodology. After completing a pre-survey, students were challenged to design a "dancer" character. This session focused on individual exploration, where students drew state diagrams on paper, captured a photo of them via the SnapBots interface, and observed their characters' behaviors on their individual screens. The second visit, which was 75 minutes, was focused on student collaboration in a digital soccer environment that included a soccer ball and two goals. Characters uploaded to the soccer environment could kick the soccer ball, and if the soccer ball went into the goal, a clapping sound effect was played. Working in small groups of 2–4, students created characters before uploading their group's final state diagram to a centralized teacher computer. This allowed for a shared project where multiple student-created characters could interact simultaneously in an environment projected onto a screen.

To evaluate the efficacy of SnapBots and its impact on student perceptions of coding, data were collected from several primary sources:

- A Pre-Survey assessed students' coding backgrounds, interest levels, and perceptions of the

field.
- A Post-Survey at the end of the second session assessed students' attitudes towards collaboration and perceptions of their abilities during the SnapBots activity.
- A Reflection at the end of the first session asked students about their initial experiences working with SnapBots.
- Student-created state machine diagrams were collected at the end of each session to analyze the complexity and structure of their state diagrams.
- Observational field notes taken by a member of the research team focused on student engagement, reactions, and conversations.

Two core limitations to this methodology should be noted. First, the sample size was limited to a single 6th-grade classroom, with 12 students consenting to the research and even fewer completing every instrument. Second, the duration of the intervention was brief, consisting of only two hours of total interaction, preventing us from capturing long-term shifts in computational thinking. More research is required to make stronger claims about how students interact with the SnapBots platform.

## Results and Discussion

The pilot study examined how 6th-grade students engage with a paper-based programming environment powered by an LLM. While the system successfully enabled students to translate hand-drawn logic into digital characters, the results highlighted several areas for refinement. We categorize our findings into four primary themes: technical infrastructure, collaboration and engagement, student imagination versus system capabilities, and the pedagogical impact on CT.

### *Technical Infrastructure*

A primary challenge was the latency of the generative AI pipeline. With a full classroom using the tool, the system became significantly slower; character generation often took minutes rather than the expected 20–30 seconds. In the Day One Reflection, four out of ten students cited these loading times as the most frustrating aspect of the activity. Additionally, several aspects of the user interface were confusing for students. For example, one student asked "how do I take a picture?" when trying to scan their diagram, and another exclaimed "Oh, I didn't press start!" after realizing that they were required to press a flag icon to run their character's code.

### *Collaboration and Engagement*

In the second lesson's soccer challenge, we observed high levels of class-wide engagement, particularly during the "showcase" phase. When one student-created character successfully scored a goal, the entire classroom erupted in cheers and applause. Student surveys corroborated these observations, with eight out of nine students responding "Agree" or "Strongly Agree" to the statement "I enjoyed working with my group on the soccer challenge." However, we did not find strong evidence of intragroup collaboration with students discussing their character around a single sheet of paper. In many groups, each student created their own diagram and then scanned the diagrams into the soccer environment sequentially.

*Student Imagination versus System Capabilities*

SnapBots allowed for a wide range of creative choices, with students designing characters ranging from a cookie to a princess. However, we observed a gap between student imagination and the system's current constraints. Because SnapBots currently maps student descriptions to a pre-defined library of roughly 350 sprites from the Scratch library, specific requests made by students such as "LeBron James" or a "kid with a quarter-zip" could not be fulfilled precisely. Furthermore, there was notable confusion regarding the character's capabilities. Students asked questions such as "Can it say 'stop'?" or "How do I make it wave?", suggesting that while natural language could lower the barrier to entry compared to a block-based or text-based language, students can be unsure of the boundary between possible and impossible actions.

*Computational Thinking and Pedagogical Impact*

In terms of core CT pillars, the results were mixed. Students quickly grasped the fundamental concept of state diagrams, and while some drawings were not syntactically perfect, many students demonstrated algorithmic thinking by creating sequences of actions. However, we did not find strong evidence of more advanced CT concepts, such as complex decomposition or abstraction. We believe we did not find such evidence due to several factors, including the latency issues that made it difficult for students to quickly iterate on their diagrams and the lack of a structured curriculum that could provide motivation for creating more complex characters.

All 10 surveyed students reported previous coding experience in the pre-survey. Between the pre- and post-surveys, several students reported a decrease in self-reported coding interest and confidence, with four out of ten students decreasing their level of agreement with the statement "I think I am good at coding" between the two surveys. This decrease may be attributed to the minimal scaffolding provided as students were introduced to a novel and unfamiliar programming modality. However, with the small sample size and short intervention duration, there were no statistically significant changes to any answers between the two surveys (all $p > 0.05$).

## Future Work

Future work is planned across all four core categories of results:

- To address the platform issues, we will optimize the LLM character creation pipeline and tune the user experience for the process of scanning and creating a new character.
- Rather than simply asking groups to collaborate on a single character, we will provide more scaffolding around group discussion. For example, groups could be asked to walk through a more rigid character-creation process of defining a character's high-level goal, look, and low-level states, with students taking on different roles and responsibilities at each step.
- To mitigate the observed confusion around possible behavior, we will develop a structured curriculum that introduces students to state machines and the SnapBots platform.
- We plan to expand the SnapBots syntax of circles and arrows to support more complex CT concepts. This could include hierarchical state machines where a complex state in one diagram could be defined by another diagram, event handling, and granular timing control.

## References


[1] L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick, "Designing ScratchJr: Support for early childhood learning through computer programming," in *Proceedings of the 12th International Conference on Interaction Design and Children*, ser. IDC '13, event-place: New York, New York, USA, New York, NY, USA: Association for Computing Machinery, 2013, pp. 1–10. [Online]. Available: https://doi.org/10.1145/2485760.2485785.

[2] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009, Place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/1592761.1592779.

[3] L. Grandell, M. Peltomaki, R.-J. Back, and T. Salakoski, "Why complicate things?: Introducing programming in high school using Python," in *ACM International Conference Proceeding Series*, vol. 165, 2006, pp. 71–80.

[4] Y. Qian and J. Lehman, "Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, Oct. 2017, Place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3077618.

[5] D. D. Williams, "DIGITAL EQUITY: DIFFICULTIES OF IMPLEMENTING THE 1:1 COMPUTING INITIATIVE IN LOW-INCOME AREAS," *Dissertations*, no. 1978, May 2022. [Online]. Available: https://aquila.usm.edu/dissertations/1978.

[6] G. Falloon, "Advancing young students' computational thinking: An investigation of structured curriculum in early years primary schooling," *Computers & Education*, vol. 216, p. 105 045, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131524000599.

[7] S. Grover, S. Basu, and P. Schank, "What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18, event-place: Baltimore, Maryland, USA, New York, NY, USA: Association for Computing Machinery, 2018, pp. 999–1004. [Online]. Available: https://doi.org/10.1145/3159450.3159522.

[8] E. B. Roe, D. Johnson, and E. E. Danahy, "BOARD # 201: Development of a Programming Environment to Bridge Students from Block-Based to Text-Based Programming (Work in Progress)," in *2025 ASEE Annual Conference & Exposition*, Montreal, Quebec, Canada: ASEE Conferences, Jun. 2025. [Online]. Available: https://peer.asee.org/55557.