From Parallel Play to Collaborative Problem-Solving: Scaffolding Teamwork with Pencil-and-Paper-Based Programming (Work in Progress)

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.

From Parallel Play to Collaborative Problem-Solving: Scaffolding Teamwork with Pencil-and-Paper-Based Programming (Work in Progress)

Introduction

Computational Thinking (CT) education for K-12 learners typically follows a progression through three distinct programming modalities. Early elementary students (ages 5–7) often begin with icon-based environments like ScratchJr [1]; middle childhood learners (ages 8–16) transition to block-based platforms like Scratch [2]; and high school students eventually move to text-based languages like Python [3]. While this pipeline can effectively introduce students to the syntax and logic of programming, it is often built upon a one-to-one device model that prioritizes individual mastery rather than collaborative creation.

Despite the collaborative nature of professional software engineering, which relies on modularity, version control, and the integration of many components, most educational programming tools are designed for a single user sitting in front of a single screen. Even when students work in pairs or small groups, the physical constraints of a computer, typically equipped with one keyboard and one mouse, can lead to a dynamic where one student drives development rather than true collaborative learning [4]. Furthermore, current sharing methods in popular platforms are largely asynchronous; a student coding in Scratch might ask another student to look at their project or "remix" another student's finished piece of code [2], but they rarely have the opportunity to contribute simultaneously to a shared, living system.

To address these challenges, researchers have long explored "unplugged" activities away from technology, including methods of paper-based coding, as a means of making abstract logic tangible and shareable [5], [6]. By moving the initial phase of algorithm design away from the screen and onto a shared physical surface, students can negotiate logic and iterate on ideas side-by-side without the bottleneck of a single input device. However, a recurring difficulty in these unplugged activities is the lack of immediate execution. Students can design a program on paper, but they cannot see the output of their program like they could for a program created on a device.

Our Solution: SnapBots for Collaboration

SnapBots is a programming environment designed to transform the act of coding from an individual task into a collaborative, systems-level design challenge. To use SnapBots, students create hand-drawn state diagrams using pencil and paper that are then scanned and converted into Python code using generative artificial intelligence (AI). A group of students, each with their own pencil, programming on a shared sheet of paper, can create a character that can be immediately tested in a digital environment. Further, this digital environment can be shared with other characters created by other students, encouraging interaction and collaboration.

State Diagram Programming as Logic Negotiation

In SnapBots, the "code" students write is a hand-drawn state diagram. Each state diagram is made up of three components: a short description of what the character should look like, one or more circles with words inside them representing the different states, or actions, a character can take, and arrows between two circles (states) specifying when the character should change from one state to another. All three of these components are written in natural language, which is then converted to formal Python code using generative AI. Figure 1 shows an example soccer player state diagram created by a group of 6th-grade students, where the player runs to the ball, then kicks the ball into the left goal, does a celebratory knee slide, and repeats the process.

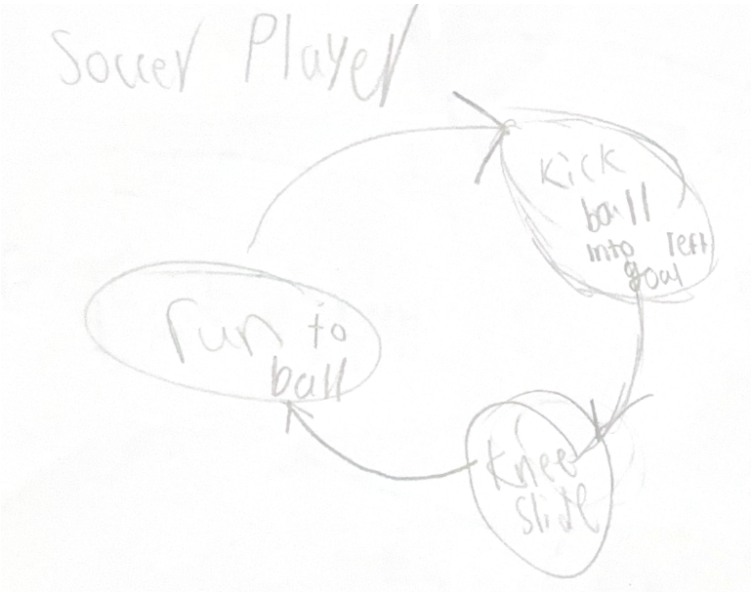

Figure 1. A soccer player state diagram created by a group of 6th-grade students

By having students describe their programs in natural language rather than formal code syntax, SnapBots allows students to focus their intragroup conversations on the high-level architecture of the program rather than low-level details. Students' discussions can center on what they want their character to achieve and why rather than the specific Scratch block they need to find or whether a command in Python is capitalized or not.

The Instructor-Created Soccer Environment

To move beyond the "look at my project" model of sharing, we developed a specialized soccer environment designed for collective participation. In this environment, the stage is not a blank canvas for a single student, but a shared digital field equipped with a soccer ball, two goals, and simple physics where each character can kick the ball if they are nearby. The shared digital soccer field is projected to the entire class, allowing students to see many students' characters interacting at once. Figure 2 shows the SnapBots interface with the soccer environment and the soccer player character described in Figure 1.

Figure 2. The SnapBots soccer environment interface with a ball, two goals, and the soccer player character from Figure 1

The environment acts as a centralized stage where multiple student-created characters are imported. This creates a shared project among all students in the class, where we imagine that students can gravitate to different roles within the soccer environment. In an ideal lesson, one group of students might create a "striker" character with the task of scoring goals and another group might create a "goalie" character to guard their goal. Students would practice abstraction as two groups negotiate the division of tasks into two different characters before splitting up to write a specific state diagram to achieve their group's role. We chose soccer as the first environment because many young students already understand the division of tasks within a soccer team through their play at recess or after school, but we imagine that many different environments could encourage this kind of collaboration.

Research Methodology

The research consisted of two interventions within a 6th-grade classroom at a public charter school in Boston, Massachusetts, during December 2025 and January 2026. While SnapBots is designed to support low-resource environments, this pilot utilized the school's 1:1 laptop ratio to allow for rapid iteration during the initial design phase. We believe that the 1:1 laptop-to-student ratio may have been a limitation of this study, as students were easily able to repeatedly upload their diagrams to test their output rather than having to work with fellow students to review and refine their diagrams before uploading to a central computer. A total of 12 students participated in the study, which was approved by an Institutional Review Board (IRB).

Session 1: Building Individual Familiarity

The first session, a 45-minute lesson, introduced students to the state machine syntax and SnapBots interface. Students created individual characters in a dance party environment. During this session, the focus was on the internal logic of a single character: defining states like "spin," "jump," or "glide" and determining the transitions among them. We initially intended for all

students to upload their character to a single computer so they could see other characters and learn from their classmates, but time constraints prevented this full-class dance party.

Session 2: Collaborative Soccer Environment

The second session was 75 minutes long and focused on group collaboration and shared goals. Students were organized into teams of 2–4 and introduced to the digital soccer environment shown in Figure 2. Students were told that the ball could be kicked by players and that partway through the session, there would be a showcase where each group would add a single character to the soccer environment and we would all watch the characters run together. There were three sections to the lesson: exploration within a small group, the full-class showcase where each group inserted their favorite diagram into a shared soccer environment projected to all students, and small-group refinement of characters after seeing the results of the showcase.

Data Collection

To evaluate the impact of this collaborative framework on student perceptions and engagement, data were collected from multiple sources:

- A post-survey administered after the second session asked for students' self-reported feelings toward collaboration. We did not collect any personal or demographic data.
- A member of the research team recorded observations on group interactions, quotes from students, and student reactions.
- We collected the state diagrams from both lessons to analyze changes in diagram complexity when students worked individually compared to working in teams.

Results and Discussion

During both sessions, there were significant technical issues with the SnapBots generative AI pipeline that caused generation for many students to take up to several minutes rather than the usual 20–30 seconds. From our observational notes and surveys asking students what they had trouble with, these technical issues clearly frustrated many students.

In the second session, where students worked in groups, we did not find strong evidence to support many of our inter- and intragroup collaboration ideals for programming with pencil and paper. For intragroup collaboration, we imagine a scenario where many students stand around a single piece of paper, discussing and negotiating with each other as they make a single character together. However, we mainly witnessed students individually creating diagrams before sequentially adding them to the groups' environment. Once the characters were added to the environment, we did see evidence of students discussing how their characters were acting in the environment. In the post-survey, many students reported that the hardest part of working in a group was finding agreement on what to create. For intergroup collaboration, we hoped to see students creating characters that took on divergent behaviors relevant to the soccer environment such as dividing into roles like a "striker" and "goalie," but found that most groups created a character with one core task: run to the ball and kick it.

Even without deep collaboration in diagram creation, students were excited to see their classmates' characters achieve the implicit task of the environment: scoring a goal. During the soccer showcase, with students sitting to watch all of the groups' characters play on a soccer field, the entire class erupted into cheers and applause when one character scored a goal. With a shared objective, students were invested in the success of their classmates' characters and were delighted even if their character wasn't the one who scored the goal.

Along with sharing in the success of fellow groups, we saw evidence of intergroup discussions about the technical aspects of creating a SnapBots character. Students from two different groups discussed the inclusion of timing in their diagrams so they could control how long different actions take. One group didn't realize that they could control the duration of a state, and a student from the other group showed how they had included time controls in their diagram. This points to a potential affordance of programming on paper, where students or groups of students can easily pass programs around to share knowledge with their peers.

Future Work

Future research and development will prioritize resolving the technical bottlenecks while expanding the collaborative and pedagogical attributes of SnapBots. To address latency issues, we will optimize the generative AI character creation pipeline, as faster generation times are necessary for maintaining continuous, on-task student discussion. Beyond technical improvements, we plan to scaffold intragroup collaboration by introducing a more structured diagram design process. This framework will guide a group of students through sequential phases of defining high-level goals, breaking down tasks, and describing states, requiring student agreement on overarching ideas before the low-level work of writing each state and transition can be divided among students. A key addition to this collaborative workflow will be the encouragement of peer debugging. One student or group could hand their paper-based state diagram to another student or group who would attempt to understand and execute the state diagram, pretending they are a robot that can move, make sounds, and say phrases. This process of diagram review encourages students to articulate their logic clearly and uses fellow students to help identify potential errors or poorly defined instructions before the diagram is ever scanned into the digital environment.

Once these technical and pedagogical improvements have been made to SnapBots, we hope to run future research studies with more participants and longer engagement than a two-hour exposure. We would like to understand the types of CT that students showcase once they have more time to become comfortable with the interface and modality.

We also aim to broaden student engagement by developing more diverse collaborative environments that necessitate specialized roles. Moving beyond soccer, future iterations might include a "beehive" environment where students must design distinct, interdependent characters that take on roles such as foragers, builders, and ventilators in a single hive. These types of environments would not only encourage complex systems-level thinking but also provide opportunities for connecting CT to subjects like biology and ecology.

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] J. Denner, L. Werner, S. Campe, and E. Ortiz, "Pair Programming: Under What Conditions Is It Advantageous for Middle School Students?" *Journal of Research on Technology in Education*, vol. 46, no. 3, pp. 277–296, 2014, Publisher: Routledge. [Online]. Available: https://doi.org/10.1080/15391523.2014.888272.

[5] A. Mehrotra, C. Giang, N. Duruz, J. Dedelley, A. Mussati, M. Skweres, and F. Mondada, "Introducing a Paper-Based Programming Language for Computing Education in Classrooms," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '20, event-place: Trondheim, Norway, New York, NY, USA: Association for Computing Machinery, 2020, pp. 180–186. [Online]. Available: https://doi.org/10.1145/3341525.3387402.

[6] T. Bell, J. Alexander, I. Freeman, and M. Grimley, "Computer science unplugged: School students doing real computing without computers," *New Zealand Journal of Applied Computing and Information Technology*, vol. 13, no. 1, pp. 20–29, 2009.