Kart-ON: An Extensible Paper Programming Strategy for Affordable Early Programming Education

ALPAY SABUNCUOĞLU, Koç University - Is Bank AI Center, Turkey
METIN SEZGIN, Koç University - Is Bank AI Center, Turkey

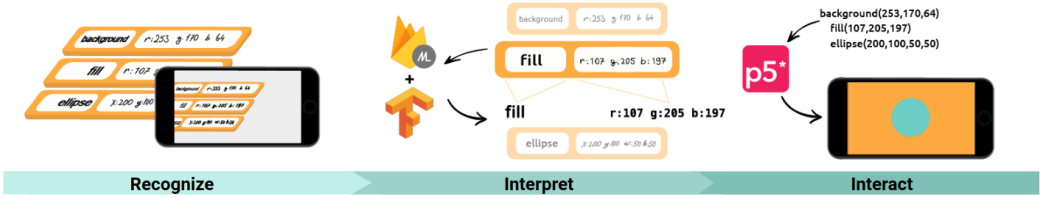


Fig. 1. **Recognize.** Students create programs using pre-defined programming cards and tangible objects. **Interpret.** Programs recognized by computer vision-powered mobile application and passed to an interpreter. **Interact.** The interpreter result is rendered to visual outputs on mobile WebView.

Programming has become a core subject in primary and middle school curricula. Yet, conventional solutions for in-class programming activities require each student to have expensive equipment, which creates an opportunity gap for low-income students. Paper programming can provide an affordable, engaging, and collaborative in-class programming experience by allowing groups of students to use inexpensive materials and share smartphones. However, current paper-programming examples are limited in terms of language expressivity and generalizability. Addressing these limitations, we developed a paper-programming flow and its variants in different abstraction levels and input/output styles. The programming environments consist of pre-defined tangible programming cards and a mobile application that runs computer vision models to recognize them. This paper describes our educational and technical development process, presents a qualitative analysis of the early user study results and shares our design considerations to help develop wide-reaching paper programming environments.
All the code and educational material is open-access at https://karton.ku.edu.tr

CCS Concepts: • **Human-centered computing** → *Collaborative interaction*; *Interface design prototyping*; • **Social and professional topics** → *K-12 education.*

Additional Key Words and Phrases: Tangible interface for programming, Shared mobile devices in programming, Collaborative classroom environment

ACM Reference Format:
Alpay Sabuncuoğlu and Metin Sezgin. 2022. Kart-ON: An Extensible Paper Programming Strategy for Affordable Early Programming Education. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 170 (June 2022), 18 pages. https://doi.org/10.1145/3534524

Authors' addresses: Alpay Sabuncuoğlu, Koç University - Is Bank AI Center, Turkey, asabuncuoglu13@ku.edu.tr; Metin Sezgin, Koç University - Is Bank AI Center, Turkey, mtsezgin@ku.edu.tr.

2573-0142/2022/6-ART170 $15.00
https://doi.org/10.1145/3534524

1 INTRODUCTION

Teaching programming in the early years helps students to build 21st-century skills such as computational and creative thinking [11]. Hence, programming education has become a core subject of the primary and middle school curriculum. Most curricula offer programming activities with web-based programming environments that support drag-and-drop interaction [16]. These environments became popular with their error-free syntax and low barrier to entry for novice users. However, as we elaborate below, they have two main shortcomings: 1) they are not accessible by the economically disadvantaged communities, 2) their traditional WIMP (windows, icons, menus, pointer) interface encourages solo problem solving and inhibits collaboration.

Existing programming environments are largely inaccessible for the low socioeconomic groups. Although they claim to make programming accessible to everyone with the premise that *everyone with a browser can run it*, this practically means the need to have a computer per child. Yet, schools in low-income communities lack sufficient resources for establishing and maintaining adequate computer laboratories [20]. Even the schools that have computers, their computing power might not be enough to run the applications [26]. Despite notable efforts of organizations like *One Laptop Per Child* [27], solutions that require distributing specialized hardware proved to be hard to scale and had limited reach [20]. Another limitation of the mainstream programming environments is their single-user workflow. These tools use traditional mouse and keyboard interfaces that allow only one user to be in control at a given time. Hence, opportunities for collaboration are usually limited to peer programming.

Recent research shows that using tangibles that foster hands-on exploration can increase collaboration and engagement in educational settings [23]. Despite these advantages, physical computing is still not popularized in early programming education, although most research states that teachers and students show great interest in research studies and workshops [32]. Some commercial examples [21, 24, 42] with computationally enhanced tangible tools that use modular blocks as programming materials have been integrated into curricula. Nevertheless, these tools still bear the cost disadvantage. Using *paper programming* has the potential to increase group activities in an affordable fashion. However, existing paper programming tools are not widespread due to their limited language expressivity, such as setting custom inputs, defining variables and functions, and extending the output capabilities.

This paper introduces a new paper programming environment that aim to deliver an effective programming education using inexpensive materials and smartphones. Our programming environment allows the development of creative coding application to teach programming fundamentals to K5-6 grades. We extended the programming environment to support simpler syntax and different modalities to meet the technical and pedagogical needs. Developing the tangible programming environment in distinct syntax, modalities and user groups allowed us to observe the potentials and the limits of paper programming, along with the usage requirements and necessary changes that led us to compile design considerations for practitioners.

Our programming environments consist of two main elements: (1) Physical programming cards that can be ordered to create new code parts, (2) a mobile application that recognizes programming cards via camera and allows defining functions using custom tangible representations. Utilizing a shared smartphone or a tablet rather than mandating each student to work on a personal computer allows large groups of students to work on programming tasks together and increases accessibility at a low cost. Distinctively, our programming environment can use custom tangible objects as programming elements. Students can use the mobile application to save functions and other algorithmic structures by associating them with their favorite physical objects or using cheap everyday materials such as paper, clay, markers, and other crafting materials.

We adopted an iterative approach while developing these applications. We regularly met with students from diverse socioeconomic backgrounds to test usability, collaboration, and engagement. In our development process, as summarized in Figure 7, we met with total 116 students from several schools and NGOs either individually or in classroom setting. In these studies, we used our paper programming environments as an introductory tool to teach programming, and observe students' behaviour while using the digital and tangible interfaces. Our observations demonstrate that our programming environments allow students to actively collaborate and contemplate the programming activities using tangible blocks. The hands-on interaction with cards also supports focusing on the given task and engages students in the activity. Our work contributes to supporting equal opportunities in programming education and supports researchers following similar paths.

In this paper, we share the development process of the paper programming cards, mobile applications, and user studies. Based on these experiences, we present design considerations for creating a paper-programming environment for K5-8 grades (9-12 years old) to guide other developers on determining the suitable design elements, activity style, input/output methods, and abstraction levels. Lastly, we elaborate on this new paper programming paradigm's unique advantages and challenges for teaching programming, as revealed through user studies.

2 BACKGROUND

We developed our programming environment to introduce the basics of programming in the early years of education and support the development of the computational thinking skills of students in a collaborative and engaging environment. We utilized the theories from HCI, tangible interaction, and education literature to design our mobile-powered tangible interface.

Teaching programming exposes students to develop computational thinking (CT) skills, which is a universal formative skill like numeracy and literacy in today's digital world that involves a set of problem-solving methods [45]. The first efforts to develop CT skills in early childhood are based on Papert's work on LOGO language, and the Turtle robot [28]. Since then, programming has been a subject of discussion for the school curriculum. In the last years, most countries have integrated programming education into their primary school curriculum. These national curricula mainly adopt web-based visual drag and drop programming environments to complete the offered programming tasks [16]. Scratch and similar visual programming environments have become popular by providing an easy-to-use and -setup programming environment to make programming education more accessible [22]. However, these web-based desktop-optimized tools require dedicating a computer per child for an effective learning experience, which is not affordable for most schools.

Using smartphones and tablets can increase the accessibility of programming education [43]. According to Qualcomm's report on modernizing education, with more than 6.3 billion mobile connections worldwide, there is an extraordinary opportunity to transform education for the 21st century by taking advantage of what has become the largest information and communication platform in history [19]. Despite the advances in mobile device capabilities, utilizing them in an educational task is challenging [12]. For example, Microsoft's Touch Develop was a mobile-focused programming environment that used all device capabilities, including touch interaction, sensory input, and multimodal output [41]. However, having a small screen size and the lack of a physical keyboard limits interaction abilities. Another challenge is shaping mobile education with learning theories, which is understudied compared to other digital platforms. Hirsh-Pasek et al. list four main pillars to entitle an application as educational: *Active, engaging, meaningful*, and *socially interactive* [12]. In this scheme, *active learning* implies being minds-on and physically active during the learning experience. *Engagement* is maintaining focus on a task. *Meaningful learning*

occurs when children understand the logic behind the subject and connect new material to existing knowledge. *Social interaction* promotes sharing and reflecting knowledge with social partners. The small screen size and the lack of external I/O devices are challenges to adopting these devices in education. We believe enabling embodiment through tangibles with mobile devices can support achieving these four pillars in building the educational application.

Tangible User Interfaces (TUI) allow hands-on interaction and control over digital features with physical artifacts [18]. These interfaces emphasize active physical interaction in the input and output spaces. Evidence from education and psychology suggests that using tangibles in learning settings supports building active, collaborative, and inclusive learning environments [23, 48]. These benefits led to the integration of tangible programming tools in the programming classroom [14]. Tangible programming tools can be categorized into three groups from a technical perspective: Electronic, unplugged, and digitally augmented paper-programming kits. Commercially-available electronic tangible programming tools [1, 2, 21, 24, 42] can be programmed on-device or via computer. However, these tools are still not affordable for most students and schools due to the cost of electronics and logistics. Unplugged activities use everyday materials to complete algorithmic tasks without requiring an electronic device [3]. In these activities, students generally mimic the behavior of an algorithmic agent and execute the algorithm step by step in a creative drama setting.

(a) Google Bloks (b) Strawbies (c) Hyper Cubes

Fig. 2. Some examples of tangible programming applications: (a) Google Bloks uses modular electronic blocks to create a programming sequence that can control actuators. (b) Strawbies uses pre-defined wooden blocks to control a game. (c) HyperCubes uses physical Ar-cubes to create 3D animations.

Between unplugged activities and computationally enhanced tangible programming tools, paper-programming tools [5, 8, 10, 13, 15, 17, 36, 37, 39] present a high-level tangible command-set that can be recognized using computer vision algorithms. These paper programming tools use mobile or stationary computer vision systems to recognize paper-printed command blocks. These tools tested their programming strategy in a domain-specific way, which mostly created game-like activities or robotic control. For example, "Hyper Cubes" orders the high-level programming cubes to control the algorithmic flow of stick-figures in an AR development environment. [8]. Tada et al.'s Sheets presents a stationary system with a lower-level command set for robotic applications, but it has limited mobility and abstraction capacity [39]. Overall, we addressed two main limitations of these paper-based tangible programming tools: (1) These domain-specific tools have limitations in terms of possible input and output variations. They do not support using custom tangible blocks to define new functions. (2) The semantic design of these *technologically appealing* tangible blocks does not demonstrate a similar structure with current popular scripting languages, which can cause frustration while transitioning in *real-life* coding [25, 29].

To overcome these challenges, we first developed a card-based programming environment where students can create drawings and animations using the pre-defined command set. Our main goal was to make this platform support expressiveness in terms of possible input/output modalities

and abstraction strategies. We believe supporting both saving functions via cards and self-made tangibles increased the expressive capacity of the application. Following our open-source tools and design considerations, developers and teachers with programming experience can easily add new commands to the programming environment or extend the language for a new subject. In the rest of this paper, we will describe the technical details of the programming environments, share the students' experiences and explain our design considerations to develop more extensible environments for paper programming experiences.

3 KART-ON'S PROGRAMMING ENVIRONMENT

In a classroom setting with Kart-ON, groups of 3 or 4 students can complete creative coding tasks such as drawing a snowman or animating an airplane using paper programming cards. To illustrate, we can define a scenario where the teacher asks students to animate a simple shape moving, which requires defining a variable, increasing the value in a loop, and using this variable as the coordinates of this shape. Each group would have a different pace. Some groups would face some challenges and request multiple tries. The teacher can either give feedback to their code without scanning or scan the code and show the error to students and give a hint. While one group thinks about the error, the teacher can move to another group and hand the smartphone to another group. This group can be fast and finish the task quickly. Now, the teacher can assign another task related to variables and give more cards to this group. As one can imagine, the group dynamics determine the quality of the session, and in an ideal environment, one smartphone can be enough to complete tasks in a classroom. Yet, providing a smartphone for each group and continuously testing the codes might be more suitable for another classroom.

Kart-ON uses *p5.js*, a JavaScript library for creative coding, to draw these shapes and animations on smartphones. Creative coding is a field of computer science that uses computer graphics and other output modalities to bring creative ideas into action [31]. The recently developed easy-to-use libraries (e.g., p5.js) and the growing attention of the maker community increased the popularity of this field in the education area. More recently, teachers and curriculum makers have started integrating creative coding frameworks into middle and high school computer science education [46]. Scratch also follows a similar path by putting drawings, animations, and storytelling as the application's core capabilities [31].

In the design of the mobile application and programming cards, we followed an iterative approach, where we tested multiple design decisions with children. Figure 3-a shows the older versions, and Figure 3-b shows the current version of an example program that uses the programming cards to create an ellipse drawing with a background color. Below, we present the final version of Kart-ON by walking through the design decisions that we obtained through the user studies.

3.1 Programming Cards

Kart-ON's programming commands are designed to allow students to quickly build their drawings and animations. The card commands can be used to draw primitive shapes, define variables and functions, control the program flow with *if/else* statements and *for loops*, add music, and use mathematical operations. The programming command subset is highly expressive, as it is a subset of *p5.js* commands where the programmer can create all control structures, including nested statements and callable states. Each control structure's scope is defined by the location of the *end* command card. Students can create programs by vertically stacking the command cards. Each card belongs to a category based on its functionality. Each category has a distinct color to increase the recognizability of the cards. These categories are primitive shapes (blue), color (green), size and position (yellow), control (red), and variables (pink). A card consists of *command* and *input* fields. It includes a maximum of two input fields, and each input field starts with a hash symbol

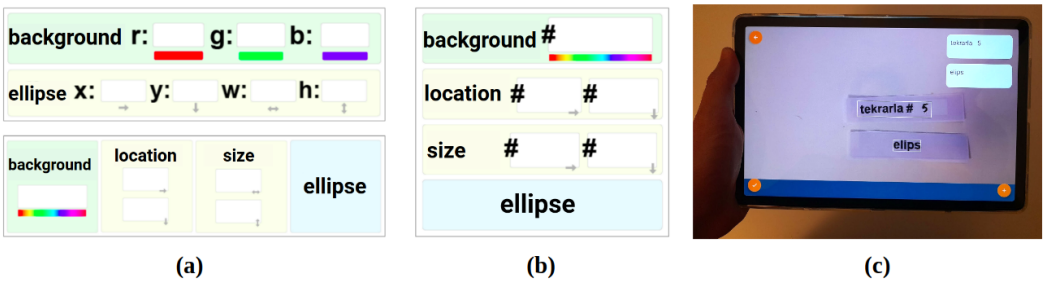


Fig. 3. The last three iterations of card design. (a) In the first iteration, input fields start with reminder initials with hint icons. We keep these icons for all iterations. In the second iteration, we tested horizontal ordering. (b) In the last and final iteration, we keep the vertically aligned commands. We limited the maximum number of input fields to two. (c) Students can show the programming commands to the mobile application one by one or multiple at once.

(#). The hash symbol is a visible reminder for students and also a distinguishable character for our computer vision model.

All the design decisions are the product of an iterative process, where we tested multiple card designs with children, as seen in Figure 3-a, b. In the first iteration, the pre-defined programming cards were a direct replica of Processing [34] commands. However, our user studies showed that if a card contains more than two input fields, introducing all the concepts related to these input fields takes an excessive amount of time and distracts the students. For example, the background command in Figure 3-a has three input fields to define a color with R, G, B definitions. To use this card, students need to learn the RGB color space. In our current design, students just use hue value to use this card. They simply change the hue value from the color picker and learn this HSL color space through experimentation. Students can create programs by vertically stacking the command cards. They can either scan each card and run the program to test if the output is correct or scan multiple cards at once and run them all. We also tested stacking the cards horizontally. Figure 3-a (bottom) shows the horizontally ordered cards. However, most children prefer to order vertically in our user studies, so we kept the vertical ordering.

3.2 Mobile Application

In the mobile application, students can find and modify examples, build new programs and save them. Any Android smartphone (> v18, which covers 70% of all in-use smartphones) with a camera can run this application.

3.2.1 Coding with programming cards. The coding process starts with recognizing the programming cards via camera. Kart-ON uses Google MLKit [9] to recognize the text on these programming cards. Students can show these programming cards one by one or multiple at once. The number of possible vertically stacked programming cards depends on printing size, as teachers can print custom cards in varying size, and the arm length of students. Each recognized physical block is listed on the coding screen as a separate graphical block. Students can also modify the commands and inputs via this graphical interface. When the program is complete, the recognized text commands are interpreted to p5.js code. The application uses p5.js to render the code and run the results on WebView.

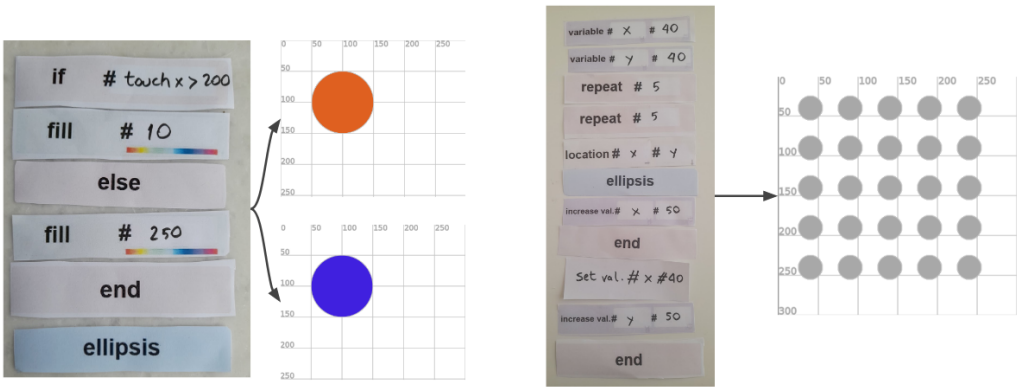


Fig. 4. ***a) Conditionals*** The program changes the ellipse color based on touch sensor position. If the horizontal touch position is more than 100 pixels, makes the ellipse red, else makes the ellipse blue. TouchX returns a numeric value between 0 and device-width and can also be used as an input in other commalnds. ***b) Loops*** The program creates a nested loop to draw a 5x5 grid of ellipses. This program uses "x" and "y" variables to control the location of the ellipsis.

After scanning a card or a card stack, the cards' content appears on the screen. To test the program's output, students should manually tap on the compile button. Then, they can update the output by either changing the cards physically or using the touchscreen.

3.2.2 Correcting the recognized text. Since the language has a limited set of commands, the application runs a fuzzy search to match the recognized text with the pre-defined Strings. For example, if the recognition of the command "fill # 120" results with "full # 120", the interpreter automatically corrects the command. We also utilized fuzzy search to correct the input fields when a variable is used. However, when it is a numeric input, we cannot use this method. So, we added easy-to-use sliders to change these inputs quickly.

3.2.3 Saving functions by using physical objects. students can save these programs by giving a name or assigning an object via camera. Assigning an object uses the transfer learning approach in a similar style to Senchanka et al. [38]. The transfer learning technique uses a model that is already trained with big data to retrain the last layers to solve a task that involves a few examples. It can recognize objects with high classification accuracy even if only a few examples are introduced to the system. In our application, the pre-trained base model uses MobileNet v2 architecture trained with the ILSVRC-2012-CLS dataset. When new examples are introduced, a fully connected layer with softmax activation is trained on top of this model. Students can assign physical objects to recall functions and programs by taking minimum of ten poses of these physical objects and training the model with these images.

3.2.4 Example Scenario: Figure 4 shows two examples that uses a conditional and a loop structure. To run these applications on Kart-ON, students open the application, select the "Code" category from the navigation bar, tap on the "Create a New Program" button on the opened page. When they tap, they see the camera's view. In this step, they can show the cards one by one or multiple at once. To add the physical cards to the program, they tap the "Add New Block" button. In the

case that the camera vision algorithm might give wrong results, students can tap on the block on the screen and change the command or inputs. When the students complete introducing all blocks to the application and are happy with their program, they tap the "See the Result" button.

4 EXTENDING THE CAPABILITIES OF PAPER PROGRAMMING

We developed two extensions of our card-based programming strategies. First, we developed Pen Mode as a beginner tool, which follows a simpler coding style similar to LOGO programming language. Second, we tested how different output modalities can use these card-based programming strategy and employ smartphones, and developed a set of commands to create 3D shapes and animations in an AR environment.

4.1 Extending as a Starter Tool: Pen Mode

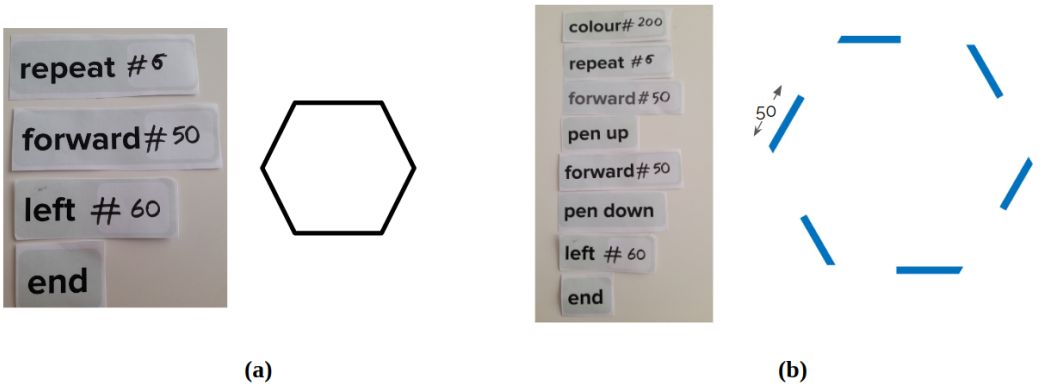


Fig. 5. (a) Drawing a hexagon with the Pen Mode Cards. (b) Drawing a dashed hexagon with the Pen Mode Cards.

In our user studies, we also observed the need for a simpler tool that teachers can use to introduce the basic concepts about coding style, coordinate systems, and simple geometry. Traditionally, educational programming environments provide a simplified subset of a programming language with graphical elements for novice users to understand the grammar, syntax, and control elements. For example, Python's Turtle Graphics [33] and JAVA's Stanford Karel Robot [30] allow users to solve puzzle-like maze questions to better understand the fundamentals of the language. In a similar fashion, our starter tool, Pen Mode, introduces a simplified version of Kart-ON. The tool borrowed its core idea from Papert's LOGO and its companion turtle robot, the first programming language to introduce computational concepts in primary and middle schools. Pen Mode also demonstrates an example case for our extensible paper programming strategy, where students can seamlessly switch between two programming environments. From a technical perspective, Pen Mode uses the same mobile camera scanner and code parser with Kart-ON, which presents an example scenario for future developers to extend the Kart-ON's capabilities.

Students can switch between Kart-ON and Pen mode inside the mobile application. Similar to LOGO, the pen agent moves on a canvas and draws geometric shapes with given commands. For example, we can draw a simple hexagon by using *forward* and *left* commands as seen in Figure5-a. *forward* command draws a straight line starting with a specified length. *left* command will rotate

the drawing agent's head towards the specified angle. Repeating *forward 50*, *left 60* commands one after the other six times will result in a hexagon. Figure5-b shows the programming cards to draw a dashed hexagon in Pen Mode. *pen up* and *pen down* blocks determine either the pen agent's color is visible or not. Moving forward with *pen up* state and then moving forward with *pen down* state will result in dashed lines.

4.2 Extending the Output Modality: Augmented Reality Outputs

Similar to Kart-ON's programming methodology, we developed an augmented reality (AR) version in an attribute-based language style. In this version, students scan the cards one by one to build 3D models and animations. The command set contains three groups of functions: Shape-creation, stylization, and animation. Shape cards include DRAW CUBE, DRAW SPHERE, DRAW CONE, and DRAW CYLINDER functions. Style cards determine the attributes (color, position and orientation) of objects or animations. These cards include PICK COLOR, ROTATE SHAPE ORIENTATION, SET POSITION, and SET SIZE. The last category, animation cards allow LOOPING, ROTATION, and CREATION OF COLOR GRADIENTS. Using these commands, students can build new 3D models that show some simple animations. A simple program can be seen in Figure 6.

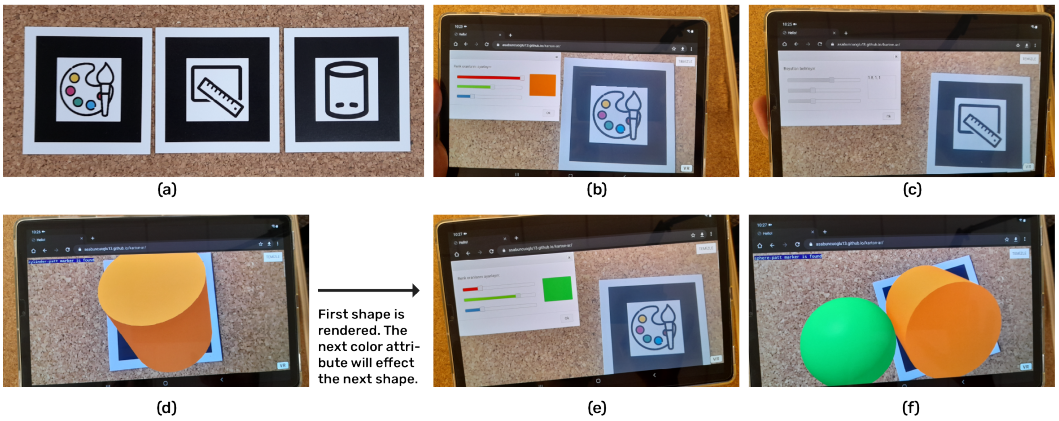


Fig. 6. (a) Three programming cards, from left to right, PICK COLOR, SET SIZE and DRAW CYLINDER. (b) A digital pop-up screen appears when the user shows the PICK COLOR marker to the AR camera. (c) Another digital pop-up screen appears when the user shows the "setting shape size" marker to the AR camera. (d) After determining the color and size via the digital interface, the user scans the sphere card. The output is rendered on the "draw a sphere" marker. (e) To add a shape with another color, the user needs to determine the color by scanning the color picker card. (e) This color is applied to the new shape on the sphere marker. As the figure shows, the new shape is rendered next to the previous shape automatically. The user can pick a location using the "setting shape position" marker to determine the position.

Scanning attribute cards such as PICK COLOR and SET SIZE results in a pop-up screen that globally defines an attribute. When the shape card is recognized, this particular 3D shape is rendered on the card with the determined attributes. For example, in Figure 6, the color card opens an RGB color picker pop-up interface. When students pick the color using this interface, this color attribute will be applied to shapes that come after this color card. In the end, the 3D model that applies the programmed attributes is rendered on the physical shape cards, similar to other marker-based interfaces.

5 USER STUDIES

We conducted our user studies in two steps. First, during the COVID-19 restrictions, we held several individual and group studies to improve the usability in an iterative approach. Then, we held three in-classroom studies using our three tools (i.e., Kart-ON, AR mode and Pen mode) to see the products in action. During the studies, we carried out semi-structured interviews.

5.1 Participants and research settings

In the first study, we conducted iterative user studies with a total of fifteen students ($\mu_{\text{age}} = 10.93, \sigma_{\text{age}} = 0.88$ | 6 females, 9 males). In this step, we used Kart-ON to gather information about the usability of our card-based methodology. After each session, we updated the application based on students' feedback. So, in each study, students used a slightly changing version of Kart-ON. Due to COVID-19 restrictions, these studies were held in various settings like schools and parks.

In the second study, we visited a public school and randomly assigned each tool (i.e., Kart-ON, AR mode and Pen mode) to three classrooms. Some of the students had experience with programming, but most of them had no previous experience. A total of ninety-one students ($\mu_{\text{age}} = 12.13, \sigma_{\text{age}} = 0.39$ | 43 females, 48 males) participated to our user studies. In each classroom, they had access to a limited number of tablets. Students who can bring their own or family's tablets used them with their group. A summary of the user study flow can be seen in Figure 7.

5.2 Procedure

In both studies, we followed a similar programming activity. First, we gave definitions of fundamental terms like programming and algorithm. Then, we introduced some real-life examples in an algorithmic structure. Later, we handed them basic programming cards. For example, in Kart-ON case, these cards contain drawing commands like *fill*, *location*, *size*, and *ellipse*, whereas in Pen-mode, the cards are *forward*, *left* and *right*). Before handing a card to students, we asked their guesses on the function of the given card. Prior to programming in a smartphone, we mostly show the step-by-step execution and possible output of the algorithm on the board, then used a smartphone to run the algorithm via paper cards. After this introduction to the setup, students worked on several programming activities. We also integrated a worksheet activity while introducing HSL color definition, where students used the mobile color picker interface to predict the *H*, *S*, and *L* values of given color on the worksheet. This activity aimed to teach students the basics of sequential thinking and helped them to understand using commands to draw shapes on a screen. In all the sessions, the main researcher helped when students asked but otherwise remained as the observer. The sessions lasted around one hour and were documented as field notes.

In the classroom studies, we asked students to bring an Android tablet if possible. In each classroom, we had enough tablets to form groups of five, which allowed us to teach the basics of programming and prototype their ideas freely. Although we followed similar procedures in these studies, we tested different variants which resulted in different outputs: *Kart-ON's output* was a rectangle and ellipse with different colors. We improved the output by adding touch input as location information. *Pen Mode's output* was drawing a rectangle first, then drawing a star by changing the inputs of commands. *AR Mode's output* was drawing three spheres with a color-changing animation.

5.3 Notes on COVID-19 Restrictions.

Our main programming flow was designed to boost collaboration and active discussion in the classroom. However, new health regulations promoted keeping social distances between students. The current health concerns still suggest these interaction rules, which affect the usage of all TUIs

Fig. 7. Our user study flow and corresponding changes in the product development. In the end of the classroom studies, we synthesized design considerations to accelerate the adaptation of paper programming.

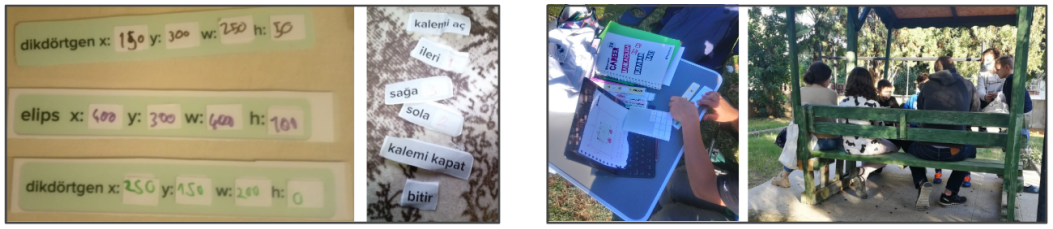
in the classroom. In our group studies, we warned children about keeping their distance from each other in several circumstances, which also shows their intention to collaborate.

5.4 Data Collection and Analysis

We employed three main data collection methods: (1) We took observational notes of the field study, (2) logged the mobile app activity, and (3) conducted semi-structured interviews with the students. We could not take video recordings of the study as our national education department does not permit it. So, we resorted to taking field notes that revolved around the participants' participation, interaction, involvement, enjoyment, and enthusiasm. We particularly noted students' facial expressions, posture, gestures during the study, as they are commonly used factors to understand the behavioral and cognitive engagement in mobile learning applications and [4, 47]. After the studies, we also conducted semi-structured interviews about students' regular tech usage, challenging spots using the app, and the parts they mainly enjoyed. In addition, all mobile app sessions are recorded using UXCam [44] to measure the screen time and see the screen interaction.

We performed open coding and inductive reasoning in the qualitative data analysis. We reflected on field notes and analyzed written answers of semi-structured interviews to understand the effects

of our application. We summarized our findings in two main branches: (1) Effect of using these environments on students' collaboration and engagement, (2) Benefits and challenges of providing a physical setup with paper cards and working with AI tools.



(a) **(b)**

Fig. 8. (a) Programming commands from the field and at-home studies. (b) User study setups from individual and group meetings.

6 FINDINGS AND DISCUSSION

Overall, the findings imply that our tangible programming environment Kart-ON has the potential to increase collaboration and engagement while reducing screen time, directly impacting affordability. Most importantly, students enjoyed using our paper programming environments even if they spent limited time using a tablet. Students individually asked for further details about downloading and running the application at the end of the studies, which showed their motivation to use our app further.

6.1 Collaboration

In our analysis, we defined *collaboration* as a mutual effort to learn, which involves a constructive conversation, asking topic-related questions, guiding the conversation, and reflecting on ideas [6]. In the group studies, students actively talked about the given tasks, reflected on each others' ideas, and challenged the given task, which fulfills our collaboration goal. For example, in Pen Mode classroom study, after learning to draw a rectangle, one group continuously challenged each other, which resulted in drawing several different shapes. Unlike other versions, AR mode increased the conversations *between* groups; they competed to create new 3D outputs. The novelty of the modality led students to display their outputs to each other and discuss how they achieved them.

UXcam logs revealed that in a one-hour study, active time spent on coding and output screens varied between 5-10 minutes. The time spent indicates that we can utilize one smartphone/tablet among 6-12 groups of students in an ideal classroom environment. As one can imagine, the group dynamics and previous experience determine the quality of the session, and the required amount of tablet devices. In an ideal environment, one smartphone can be enough to complete tasks in a classroom. Yet, providing a smartphone for each group and continuously testing the codes might be more suitable for another classroom.

Only two out of twenty-two groups had struggles in collaboration. In both cases, the tablet owners did not allow others to use their tablets, which resulted in negative experiences for other group members. To solve the issue, the lead researcher landed them his own tablet and helped them run the program. But, this situation showed us that, organizations that would like to use shared tablets in classroom should not depend on children's own tablet.

6.2 Engagement

Blending the movement and action in the physical space with digital programming is an ongoing challenge for TUIs [7]. The transition between these domains can interrupt the interaction and result in a less engaging environment. Yet, in the program building phase, we observed that this interruption occurred naturally as it feels like completing one task (algorithm building), and moving to another (compiling the program/debugging).

In addition, students' movements such as reaching the cards on the table, moving around the program blocks, and showing enthusiasm to participate in discussion demonstrate the direct effect of screen-free time. The observers note that students kept their excitement throughout the session as apparent in their demeanor (upright, gaze on the task) and continuous conversations [47]. Students discussed the programming concept while holding the programming card, switched it amongst themselves, and rearranged it on the table when stuck. Such physical interactions increased the involvement of even the most reluctant participants, who refrained from making comments but demonstrated their idea 'tangibly' in the conversations. Therefore, the engagement of the participants was kept alive with hands-on conversations.

Although three variants resulted in similar engaging experiences, using the AR version particularly excited students when they saw the program output 3D on a paper card. We heard most "*wow*" s when they created the output on 3D mode. Nevertheless, this can be a *novelty effect* and may not engage students in a long-term curriculum setup.

6.3 Providing a Physical Setup

We believe that the physical setup endorsed children's process-oriented nature, as their primary attention was not on the digital output. The user studies and UXcam results revealed that students spend most of their time in active discussions and negotiations on the final version of their code before trying these out in the mobile application. During our on-lecture conversations, students mentioned that sharing the smartphone came naturally, as they use smartphones to play games and watch videos together. Also, the students tended to divide the cards amongst themselves to find the right one and switch these like tokens. As it is a familiar material that children are already using to pass information and a natural medium for collaboration, they figured out easily how to use the card. The physicality of the setup also engaged reluctant students who refrain from making comments and prefer to demonstrate their ideas in a hands-on manner. They were always aware of their current tasks. The nature of card-based programming methodology requires keeping a neat workflow by scanning cards step-by-step. Our observations revealed that holding a single card and discussing the content step-by-step on the table helped students to focus.

The card-based programming led us to organically integrate worksheet activities in our workshop flow. Introducing the HSL definition of colors with small gamified activities increased their engagement. All students enjoyed the activity and wanted to continue it even after finding five different color definitions. Therefore, integrating interdisciplinary content with gamified activities is an opportunity in Kart-ON to help children interactively learn scientific concepts. Further, cards brought gamification ideas in the tangible domain. For example, we asked students to complete the missing cards to complete some tasks, which appeared as an exciting challenge. Teachers can use these kinds of activities and board-game-like activities, which can naturally be suitable for a card-based interface.

The main drawback of using paper cards appeared when the cards were deformed, and the mobile application could not recognize the card. We can solve this issue by printing them on a rigid material. This material choice can decrease the affordability, but it can significantly improve the user experience.

6.4 Working with AI tools

Powering our application with computer vision algorithms is both a challenge and an opportunity. For example, in some cases, when students encountered unrecognized illegible handwriting, they could not realize this, and the output differed from their expectations. We observed the need to improve handwritten text recognition accuracy on programming cards. Figure 8-a shows some examples of children's programming cards. Students used different pen types and varying colors to fill the input field. To increase the recognition accuracy, we are collecting a dataset that contains programming card images. On the other hand, introducing the inner workings of Kart-ON was an opportunity to build awareness of AI to children, which is becoming more and more important in today's world. In the studies, they could understand when and why the system did not detect the command or their input. We mention how character recognition data has regular lighting and style, so the model can only accurately predict this kind of input.

7 DESIGN CONSIDERATIONS

Our development and test experiences guided us to curate six design considerations for developing accessible paper programming environments. Researchers can combine these design considerations with other construction guidelines for children, such as Resnick and Silverman's [35], which provide ten principles to help children become more fluent, expressive, and comfortable using new technologies.

(1) **Choose card designs based on in-class activity flows.** One of the potentials of paper programming cards is their physical variance. They can be in multiple shapes, textures, and colors. Yet, current implementations are limited since most applications use QR codes and predefined shapes to recognize these cards. We suggest diversifying the cards by carefully considering the activity at hand. For example, Kart-ON activities require combining many programming cards vertically on one screen. To support this physical arrangement, we designed the card dimensions accordingly and used color as a distinguishing factor. The number of possible vertically stacked programming cards depends on printing size and the arm length of students. The maximum number of stacked cards in our studies was eight. We promoted stacking the cards based on the functionality of the given command set.
Alternatively, the AR cards were designed like a real-time prototyping tool that uses one programming card at a time, so the cards can be larger and more diverse. Even though we only used square cards, other shapes can be considered to make children recognize and distinguish cards more easily. Current object and text recognition models and fine-tuning these models with the transfer learning approach [40] can yield accurate results when custom design cards are employed.

(2) **Find technically relevant input methodology.** The main challenge of paper programming is determining the proper input methodology. Handwriting recognition, introducing new tangibles with transfer learning approaches, placing new NFC stickers to new tangible inputs can be all appropriate for different purposes. Kart-ON uses handwriting recognition for inputs as the inputs have limited character amount, which results in acceptable error counts. AR version utilizes a smartphone screen to enter new inputs as it actively involves mobile devices.

(3) **Define abstractions in the same modality.** In the physical computing domain, defining abstractions generally occurs via using mobile devices or computers. This situation increases the dependence on electronic devices and reduces affordability. Designing self-made tangibles for function and variable definitions decreased the total screen time, supporting our

goal of achieving a more efficient model for sharing smartphones. However, we also observed that children struggled to create new tangibles to represent abstract definitions. We still need to explore students' self-made tangible design process and develop a more detailed guideline to ease the process.

(4) **Choosing Black-boxes Carefully:** As Resnick and Silvermann describe, the abstraction-level of the kit defines the ideas that users can explore and remain hidden from view [35]. One of the main discussions during the development of our programming environment was choosing black boxes in high-level operations. For example, in the development of Kart-ON, our goal was to keep the drawing operations semantically similar to current creative coding frameworks from an educational perspective. In this way, when students encounter popular frameworks like Processing, they could become much more familiar. But, in the user studies, we observed that using four inputs in one command has drawbacks. So, we decided to add new functionalities in Processing-style command blocks.

(5) **Consider the pedagogically appropriate number of student inputs.** Determining the right number of inputs for children's comprehension requires iterative experiments. Our observations revealed that requiring more inputs in command has two drawbacks: (1) Students struggle to keep all the required inputs in mind, (2) Educators spend a substantial amount of time before using only one command. In our experiments, we first printed the Kart-ON cards in the same style with Processing commands. For example, similar to Processing, our *rectangle* command was taking x, y, w, and h inputs. However, students had difficulty remembering all these different inputs in one command. Then, we introduced two new commands to ease the shape commands: *size* and *location*. Each command takes two inputs, and students could easily understand the commands' roles in the code.

(6) **Support teachers' adaptation to the system.** Having paper cards puts an extra amount of work on teachers' shoulders. Before studying with children, we regularly met with teachers in the development process, and they were anxious about the work required to implement this programming approach in their classrooms. They need to print cards, actively observe the groups, walk around student groups while scanning the cards and ensure the computer vision model yields the correct output. Before the development stage, listening to them led us to design various elements in our programming flow like "grouping behavior," "structured worksheets," and "multi-screen interfaces."

(7) **Determine the physical workspace setup** The benefit of using tangibles in programming is keeping children physically active and enhancing collaboration. If the physical setup of the classroom cannot allow that, the tool cannot achieve these benefits. In our case, having a tablet for 4-5 students result in good collaboration, but asking students to bring their tablets is not the ideal scenario.

8 CONCLUSION

In this paper, we presented Kart-ON programming environments in different programming styles and modalities, which share a common goal to build an affordable and engaging programming experience. The programming environments consist of predefined paper programming cards and utilize a computer vision-powered smartphone application to recognize these cards. Additionally, students can use the application to save the functions and other algorithmic structures with their favorite physical objects. In a classroom scenario, students first complete card-based activities without any electronic or digital application, then use a shared smartphone to see the real-time output. Our meetings with fifteen students in diverse settings showed that using Kart-ON supports the collaboration in group activities, engages students in activities, and dramatically reduces screen time which directly impacts affordability.

Developing Kart-ON and its extensions in distinct programming areas using different output modalities led us to compile design considerations for practitioners to accelerate the adaptation of this affordable and engaging approach. Our considerations share best practices on card designs, input/output numbers and modalities, abstract definitions, and in-classroom flows. These programming environments share the same core programming strategy, which eases the development of other affordable applications using paper cards. All the tools and tutorials are open source and can be found at *https://karton.ku.edu.tr*, which provides code, educational material and example classroom activities with videos. Our work contributes to supporting equal opportunities in programming education and supports researchers following similar paths.

ACKNOWLEDGMENTS

This project is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) with project number 218K436. We would like to thank TUBITAK and KUIS AI Center for their support. We also gratefully acknowledge the support of the participant teachers, Şeval Gül from 21. yy Eğitim ve Kültür Vakfı-Nilüfer Gökay Ortaokulu, Burçin Yolacan from, İTO Yeniköy Mehmetçik Ortaokulu and Ece Saraç from Emirgan Osman Saçmacı Ortaokulu.

REFERENCES

[1] Arduino. 2021. Arduino - Home. https://www.arduino.cc/

[2] T. Ball, J. Protzenko, J. Bishop, M. Moskal, J. de Halleux, M. Braun, S. Hodges, and C. Riley. 2016. Microsoft Touch Develop and the BBC micro:bit. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 637–640.

[3] Tim Bell and Jan Vahrenhold. 2018. *CS Unplugged—How Is It Used, and Does It Work?* Springer International Publishing, Cham, 497–521. https://doi.org/10.1007/978-3-319-98355-4_29

[4] Melissa Bond, Katja Buntins, Svenja Bedenlier, Olaf Zawacki-Richter, and Michael Kerres. 2020. Mapping research in student engagement and educational technology in higher education: a systematic evidence map. *International Journal of Educational Technology in Higher Education* 17, 1 (2020), 2. https://doi.org/10.1186/s41239-019-0176-8

[5] Xiaozhou Deng, Danli Wang, Qiao Jin, and Fang Sun. 2019. ARCat: A Tangible Programming Tool for DFS Algorithm Teaching. In *Proceedings of the 18th ACM International Conference on Interaction Design and Children* (Boise, ID, USA) *(IDC '19)*. Association for Computing Machinery, New York, NY, USA, 533–537. https://doi.org/10.1145/3311927.3325308

[6] Pierre Dillenbourg. 1999. What do you mean by collaborative learning?

[7] Ylva Fernaeus and Jakob Tholander. 2006. Finding Design Qualities in a Tangible Programming Space. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 447–456. https://doi.org/10.1145/1124772.1124839

[8] Anna Fuste and Chris Schmandt. 2019. HyperCubes: A Playful Introduction to Computational Thinking in Augmented Reality. In *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts* (Barcelona, Spain) *(CHI PLAY '19 Extended Abstracts)*. Association for Computing Machinery, New York, NY, USA, 379–387. https://doi.org/10.1145/3341215.3356264

[9] Google MLKit. 2021. ML Kit | Google Developers. https://developers.google.com/ml-kit

[10] Sidhant Goyal, Rohan S. Vijay, Charu Monga, and Pratul Kalita. 2016. Code Bits: An Inexpensive Tangible Computational Thinking Toolkit For K-12 Curriculum. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction* (Eindhoven, Netherlands) *(TEI '16)*. Association for Computing Machinery, New York, NY, USA, 441–447. https://doi.org/10.1145/2839462.2856541

[11] Shuchi Grover and Roy Pea. 2013. Computational thinking in K–12: A review of the state of the field. *Educational researcher* 42, 1 (2013), 38–43.

[12] Kathy Hirsh-Pasek, Jennifer M. Zosh, Roberta Michnick Golinkoff, James H. Gray, Michael B. Robb, and Jordy Kaufman. 2015. Putting Education in "Educational" Apps. *Psychological Science in the Public Interest* 16, 1 (may 2015), 3–34. https://doi.org/10.1177/1529100615569721

[13] Michael S Horn and Robert JK Jacob. 2007. Tangible programming in the classroom with tern. In *CHI'07 extended abstracts on Human factors in computing systems*. 1965–1970.

[14] Michael S. Horn and Robert J. K. Jacob. 2007. Designing Tangible Programming Languages for Classroom Use. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (Baton Rouge, Louisiana) *(TEI '07)*. Association for Computing Machinery, New York, NY, USA, 159–162. https://doi.org/10.1145/1226969.1227003

[15] Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. 2015. Strawbies: Explorations in Tangible Programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (Boston, Massachusetts) *(IDC '15)*. Association for Computing Machinery, New York, NY, USA, 410–413. https://doi.org/10.1145/2771839.2771866

[16] Peter Hubwieser, Michail N Giannakos, Marc Berges, Torsten Brinda, Ira Diethelm, Johannes Magenheim, Yogendra Pal, Jana Jackova, and Egle Jasute. 2015. A global snapshot of computer science education in K-12 schools. In *Proceedings of the 2015 ITiCSE on working group reports*. 65–83.

[17] Hyejin Im and Chris Rogers. 2021. *Draw2Code: Low-Cost Tangible Programming for Creating AR Animations*. Association for Computing Machinery, New York, NY, USA, 427–432. https://doi.org/10.1145/3459990.3465189

[18] Hiroshi Ishii and Brygg Ullmer. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. 234–241.

[19] Irwin Mark Jacobs. 2013. Modernizing Education and Preparing Tomorrow's Workforce through Mobile Technology. *World Economic Forum Global Information Technology Report* (2013). http://iiij.org

[20] Kenneth L. Kraemer, Jason Dedrick, and Prakul Sharma. 2009. One Laptop per Child: Vision vs. Reality. *Commun. ACM* 52, 6 (June 2009), 66–73. https://doi.org/10.1145/1516046.1516063

[21] LEGO. 2021. LEGO® MINDSTORMS® About | Official LEGO® Shop GB. https://www.lego.com/en-gb/themes/mindstorms/about

[22] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[23] Paul Marshall. 2007. Do tangible interfaces enhance learning?. In *Proceedings of the 1st international conference on Tangible and embedded interaction*. 163–170.

[24] MBlock. 2021. Program Neuron with mBlock | mBlock. https://www.mblock.cc/doc/en/hardware-guide/neuron/neuron.html

[25] Luke Moors, Andrew Luxton-Reilly, and Paul Denny. 2018. Transitioning from Block-Based to Text-Based Programming Languages. In *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE, 57–64. https://doi.org/10.1109/LaTICE.2018.000-5

[26] OECD. 2020. *PISA 2018 Results (Volume V)*. 328 pages. https://doi.org/https://doi.org/10.1787/ca768d40-en

[27] OLPC. 2021. One Laptop Per Child. https://www.onelaptopperchild.org/

[28] Seymour Papert. 1999. What is Logo? Who needs it. *Logo philosophy and implementation* (1999), 4–16.

[29] Dale Parsons and Patricia Haden. 2007. Programming osmosis: Knowledge transfer from imperative to visual programming environments. (2007).

[30] Richard Pattis, J Roberts, and M Stehlik. 1981. Karel the robot. *A gentele introduction to the Art of Programming* (1981).

[31] K Peppler and Y Kafai. 2005. Creative coding: Programming for personal expression. *Retrieved August* 30, 2008 (2005), 314.

[32] Mareen Przybylla. 2014. Physical Computing in Computer Science Education. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14)*. Association for Computing Machinery, New York, NY, USA, 169–170. https://doi.org/10.1145/2632320.2632336

[33] Python Docs. 2021. turtle — Turtle graphics — Python 3.9.2 documentation. https://docs.python.org/3/library/turtle.html

[34] Casey Reas and Ben Fry. 2007. *Processing: a programming handbook for visual designers and artists*. Mit Press.

[35] Mitchel Resnick and Brian Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children*. 117–122.

[36] Alpay Sabuncuoğlu, Merve Erkaya, Oğuz Turan Buruk, and Tilbe Göksun. 2018. Code notes: designing a low-cost tangible coding tool for/with children. In *17th ACM Conference on Interaction Design and Children (IDC '18)*. 644–649.

[37] Alpay Sabuncuoğlu and Metin Sezgin. 2020. Kart-ON: Affordable Early Programming Education with Shared Smartphones and Easy-to-Find Materials. In *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. 116–117.

[38] Paul Senchanka. [n.d.]. Example on-device model personalization with TensorFlow Lite — The TensorFlow Blog. https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html

[39] Kazuki Tada and Jiro Tanaka. 2015. Tangible Programming Environment Using Paper Cards as Command Objects. *Procedia Manufacturing* 3 (2015), 5482–5489. https://doi.org/10.1016/j.promfg.2015.07.693 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015.

[40] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A Survey on Deep Transfer Learning. arXiv:1808.01974 [cs.LG]

[41] Nikolai Tillmann, Michal Moskal, Jonathan De Halleux, Manuel Fahndrich, and Sebastian Burckhardt. 2012. TouchDevelop: app development on mobile devices. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–2.

[42] Twin Science and Robotics. 2021. Twin Science | Science Kit - Robotics, Coding and Education Kits. https://www.twinscience.com/en/

[43] UNESCO. 2019. *Artificial intelligence in education, compendium of promising initiatives: Mobile Learning Week 2019.* Technical Report. https://unesdoc.unesco.org/ark:/48223/pf0000370307

[44] UXCam. 2021. Deliver the perfect app experience. https://uxcam.com/

[45] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[46] Zoe J. Wood, Paul Muhl, and Katelyn Hicks. 2016. Computational Art: Introducing High School Students to Computing via Art. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 261–266. https://doi.org/10.1145/2839509.2844614

[47] Woo-Han Yun, Dongjin Lee, Chankyu Park, Jaehong Kim, and Junmo Kim. 2020. Automatic Recognition of Children Engagement from Facial Video Using Convolutional Neural Networks. *IEEE Transactions on Affective Computing* 11, 4 (2020), 696–707. https://doi.org/10.1109/TAFFC.2018.2834350

[48] Oren Zuckerman, Saeed Arida, and Mitchel Resnick. 2005. Extending tangible interfaces for education: digital montessori-inspired manipulatives. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 859–868.

Received February 2022; accepted April 2022