Exploring Children's Use of Self-Made Tangibles in Programming

Leave Authors Anonymous
for initial submission
City, Country
e-mail address

Leave Authors Anonymous
for initial submission
City, Country
e-mail address

ABSTRACT

Defining abstract algorithmic structures like functions and variables using self-made tangibles can enhance the usability and affordability of the tangible programming experience by remaining within the context throughout the activity and reducing the dependence on electronic devices. However, current tangible programming environments use digital interfaces to save these abstract definitions, as designing new tangibles is challenging for children due to their limited experience using abstract definitions. We conducted a series of design workshops with children to understand their self-made tangible creation abilities and develop design considerations for tangible computing such as paper programming.

This paper reports:

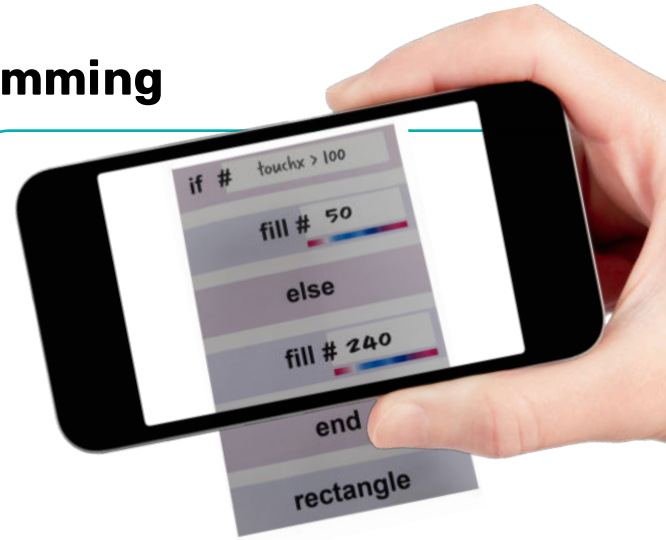
1. Our insights on how students conceptualize and design tangible programming blocks
2. Design considerations of self-made tangibles to yield higher understandability and memorability
3. Tests of our design considerations in creating self-made tangibles in real-life coding activities.

Authors Keywords

Co-design with children, Tangible programming, Self-Made Tangibles for Paper Programming

CSS Concepts

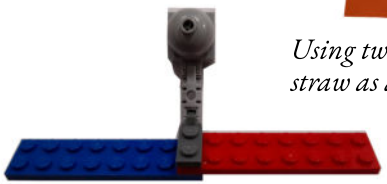
• Social and professional topics → K-12 education; • Applied computing → Education;



Our research begins with this question:
How would a child save this function via a smartphone camera by building tangibles using craft materials or everyday objects?

Using play dough with red and blue colors?

Using two cardboard pieces with a straw as a sperator?



Using LEGO with a separator that symbolizes the conditional statement?

Although agreeing upon a precise answer of this question is subjective , at the end of our research, we curated design considerations to help children create well-designed code elements.

INTRODUCTION

There is a push to integrate programming into the primary education curriculum, given that computational literacy has become a crucial 21st-century ability [4]. This integration aims pupils to acquire vital skills such as creative and computational thinking as well as hands-on skills like building electronics, animations, and integrating technology into daily life [13]. Currently, visual programming environments like Scratch and Alice have become standard tools to teach programming [6, 5]. Interest in tangible programming environments where students code with physical blocks is also growing since they support inclusivity and collaboration in the classroom environment [9, 11, 12]. However, current tangible environments either require an external coding device like a computer or tablet or costly internal electronics, rendering them out of reach for low socioeconomic levels.

Paper programming environments such as Kart-ON [10], and Budgie [18], bring tangibles to the classroom while making them affordable. In this scenario, students use pre-defined, easily accessible programming blocks to create their programs and run computer-vision-powered devices to scan these blocks. On top of this recently formulated paper programming research, we wanted to explore a programming setup where children create their self-made tangible programming blocks to represent abstract definitions. Research posits that learning occurs when children build and design personally relatable elements [15]. In our scenario, students use everyday objects to associate functions, entities, and properties such as variable assignment and procedure calls. This setup can allow groups of students to create and combine self-made tangible programming blocks for coding, then "run" and observe the behavior of their program using a shared smartphone acting as an interpreter.

We assert that students may actively explore improved ways of framing their algorithms and consider the relationship between commands, inputs, and outputs via self-made tangibles while collaborating with their friends in this screen-free environment. However, integrating self-made tangibles to define abstractions in these paper programming environments brings two main questions:

(1) Is using self-made tangibles to represent abstract programming definitions educationally and pedagogically appropriate?

(2) Can self-made tangible become well-designed code elements, which can be memorized and understood in later uses?

We designed a four-step workshop to address these questions where students learned how to define algorithmic tasks, create self-made tangibles, and use them to create programs. In the first step, we teamed up with 7-11-year-old children and decomposed a problem/task together. Next, they designed a set of tangible objects for programming the selected task. In the third step, we tested the memorability and understandability of their self-made tangibles. Based on the test results, we developed design considerations. Finally, we utilized these considerations in a real-life paper programming environment.

This paper presents our quest to integrate student-made tangibles into programming environments. We report our observations, tangible outcomes, findings, and design considerations. We also share our preliminary test results of using design considerations in a real-life paper programming task and discuss the potentials and drawbacks of self-made tangibles for programming.

SELECTION AND PARTICIPATION OF CHILDREN

We obtained ethical approval for this study from our university's Committee of Human Research. Due to COVID-19 restrictions, we conducted the studies outside with the children from the same neighborhood. We contacted the parents via a WhatsApp group and informed the study details. We submitted the informed consent forms via Google Forms for those who responded positively. At the beginning of each workshop, we introduced ourselves, explained the research in a simplified way, and clarified that they were free to go back to their home if they desired. None of the children quit the session or showed stressed behavior.

RELATED WORK

Between unplugged activities and computationally enhanced tangible programming tools, paper programming offers an affordable way to complete programming activities. Paper programming environments use card-based tangible programming blocks to create a code. Mostly, a separate scanner or a mobile device is utilized to interpret the recognized programming blocks [7, 10]. These environments show the advantages of physical computing while using inexpensive materials as programming blocks. While using inexpensive materials as programming blocks, the environment shows the benefits of using tangibles in learning. Further, paper programming can be a beneficial step in programming education as it bridges physical computing to scripting languages. This learning approach is coined in "concreteness fading", which argues an intermediate step between concrete and abstract definitions to help learners to distill the generic knowledge and develop conceptual understanding [2].

Current paper programming environments present a high-level tangible command-set that can be recognized using computer vision algorithms. Although research on using these programming environments in primary and middle school programming education report several benefits such as increasing engagement and collaboration, teachers do not adopt these applications into their curriculum. We addressed two main limitations of these paper-based tangible programming interfaces: (1) These domain-specific tools limit possible input and output variations by not supporting using custom tangible blocks to define abstractions. (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 [20]. Using self-made tangibles to define abstract definitions in programming languages can enhance the user experience by keeping the same modality and extend the language expressivity in the tangible domain. However, we first need to explore students' needs and behaviors with self-made tangibles to effectively integrate this approach in programming environment design.

Tangible User Interfaces (TUI) allow hands-on interaction and control over digital features with physical artifacts [22]. Tangible programming tools can be categorized into three groups from a technical perspective: Electronic, unplugged, and digitally augmented paper-programming kits.

Marshall defines three main advantages of using tangibles in programming [14, 21]:

1. It keeps children mentally and physically active,
2. Enhances collaboration by supporting natural group interactions and increases the visibility of outputs,
3. Naturally invites students to action and improves inclusivity by lowering the threshold for children's participation.

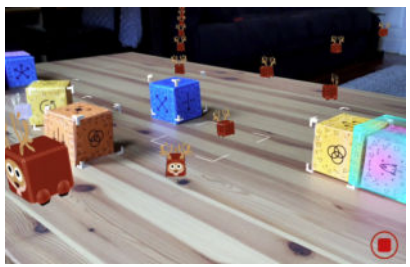
Druin defines four roles that children can partake in a technology design process: user, tester, informant, and design partner [1]. In our workshops, students assumed varying roles, and acted as informants, testers and design partners.

In the informant role, children play a part in the design process at various stages, based on when researchers believe children can inform the design process. As a tester, children are again observed with the technology and asked for their direct comments concerning their experiences. With the role of design partner, children are considered equal stakeholders in the design of new technologies throughout the entire experience.

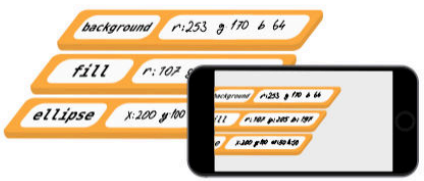
We started with discovering the interest of children and understanding how they decompose the given problem. Then, we created and tested tangible objects to represent the commands obtained from decomposed problems. Throughout the process, we approach them as design partners.

SOME EXAMPLES OF TANGIBLE INTERFACES

Topobo uses electronics with motor memory. Raffle et al. 2004. [8]

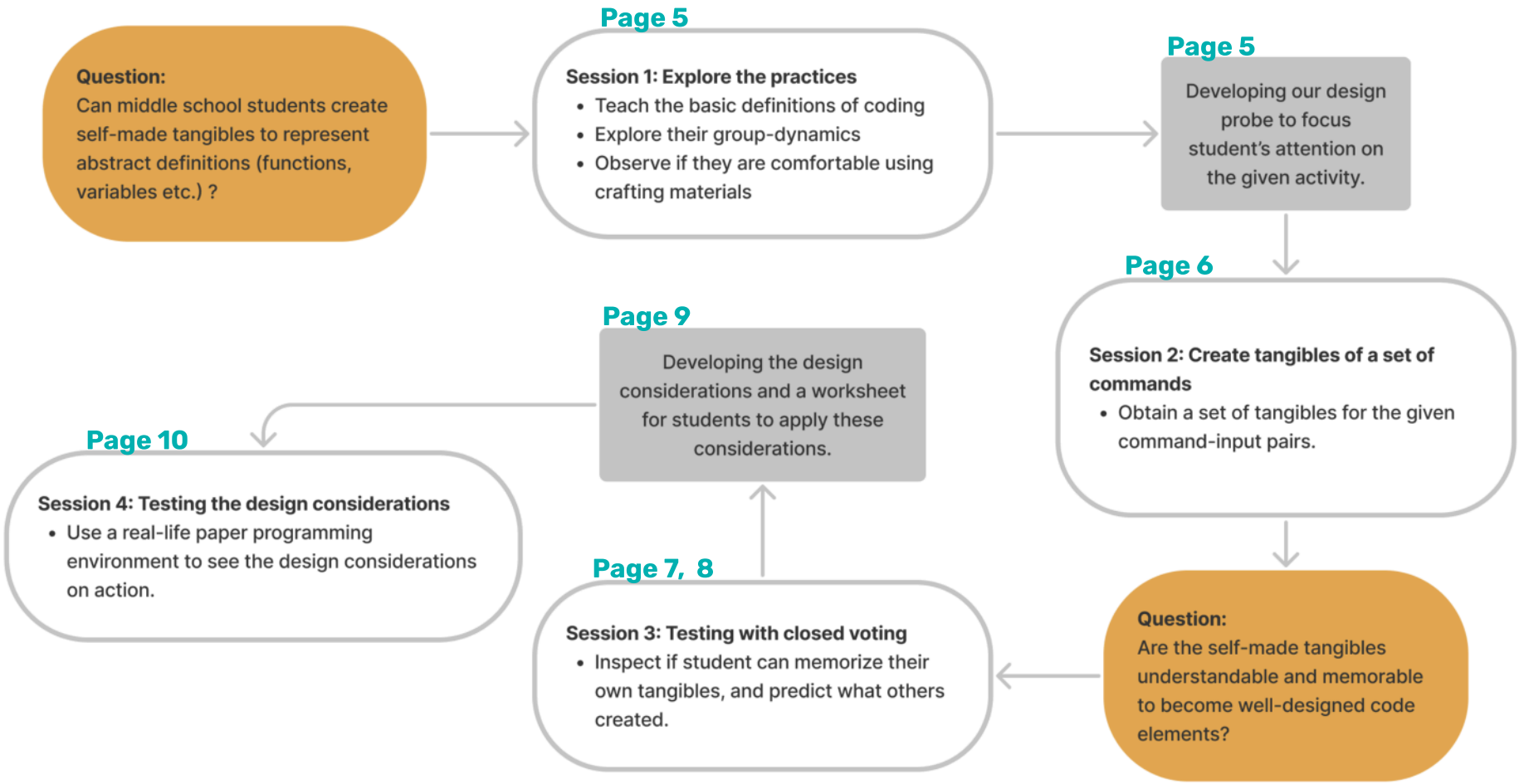


HyperCubes uses AR to program some motion for 3D elements.. Lleixà. 2018. [3]



KartON uses text recognition on smartphone camera. Sabuncuoglu et al. 2019. [10]

FLOW OF THE DESIGN STUDIES



Limitations of the Studies. In total, we had six children of ages 7-11 years old. Considering participant number and diversity, we cannot easily generalize the findings. Yet, we open-sourced all the activity guides and materials to encourage teachers to replicate these studies in their classrooms and share their observations. All of the workshops were conducted outdoors due to COVID-19 regulations. Since children have limited experience in programming, we included an introduction before the design process to give the necessary foundation.

SESSION 1: INTRODUCING THE CONCEPTS AND LEARNING THE NEEDS

In the workshop, the provided material box consisted of various items, including paper, clay, wires, LEGO pieces, etc. Using these materials, each child created one tangible object per sub-task. Their collaborative output can be seen in this figure.

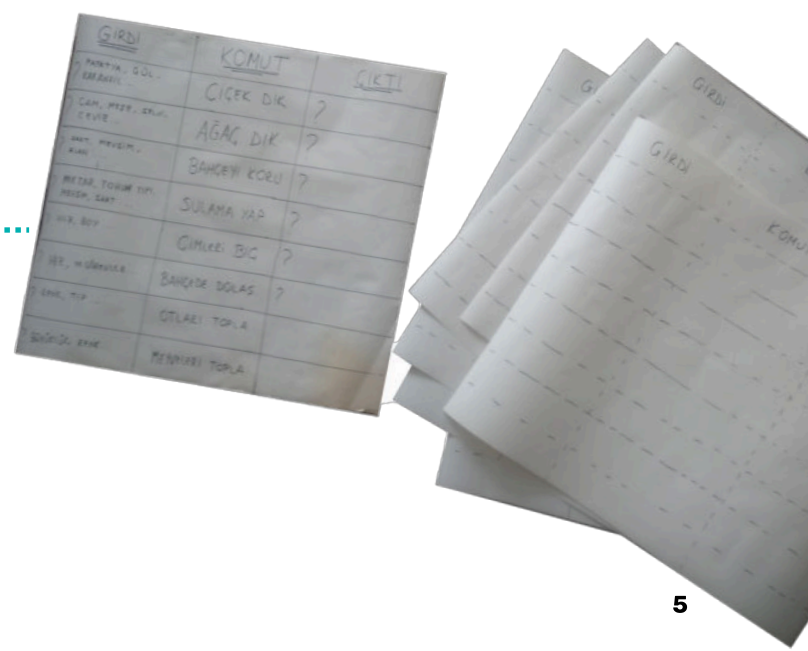
We held the first workshop with six children from our local area (1 seven-year-old, three ten-year-old, two eleven-year-old children). First, we introduced programming and discussed its role in many breakthroughs in society. Later, we asked them to define a problem or a task to solve with programming. The children came up with the task of "gardening" and started decomposing the possible sub-tasks that could be solved with programming. After listing all the tasks, we agreed upon eight sub-tasks: potting flowers, planting trees, protecting the garden, watering the garden, mowing grass, removing weeds, walking around, and picking fruits.

In this workshop, we wanted to see if children were comfortable with the given materials and their ability to understand the task at hand. Our observations revealed that all students were comfortable with the provided materials and were eager to explore programming concepts regardless of their age. As seen in the above figure, almost all students created tangible objects with different materials. The sizes of the tangibles were similar in each child's creation. Overall, this showed us that our materials were diverse enough to accommodate children's preferences.

PREPARING FOR THE NEXT SESSION

Throughout the workshop, we observed some tendency to disengage from the task due to the informality of the workshop. To explain, some children knew each other and also the vicinity. While the familiarity led them to collaborate easily on the given task, it could quickly turn into a free chat and distract them from the task. It is known that group dynamics directly impact children's creativity [19]. Based on this experience, we decided to hand out a design probe for the next workshop to help focus children's activities.

We developed this design probe to help children focus on their tangible creation task. It is an ~A3-sized paper that contains a 9-row table with "input," "command," and "output" headers.

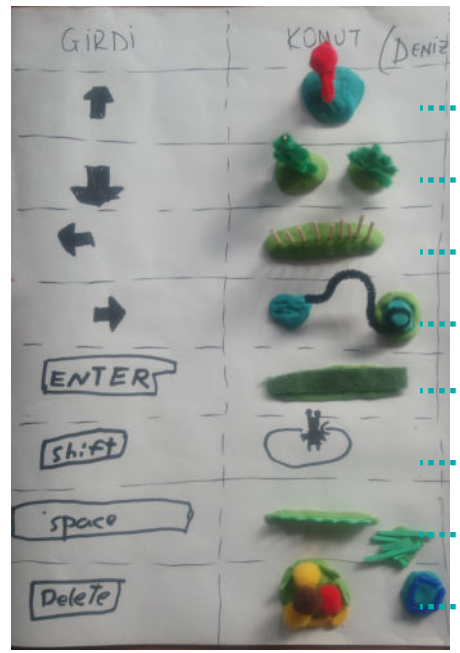


SESSION 2: DESIGNING TANGIBLES OF A SET OF REAL-LIFE COMMANDS

We held the second workshop with five children from the previous study (1 seven-year-old, 2 ten-year-old, 2 eleven-year-old); one child could not attend the workshop. In this session, we distributed the design probes to help them maintain their focus. We hung a larger copy with commands and possible input names to a wall that all students can see. We handed out empty copies of this probe and asked them to place their final self-made tangibles on pre-defined positions. As intended, the probe helped them to use their time and communicate their ideas more efficiently.

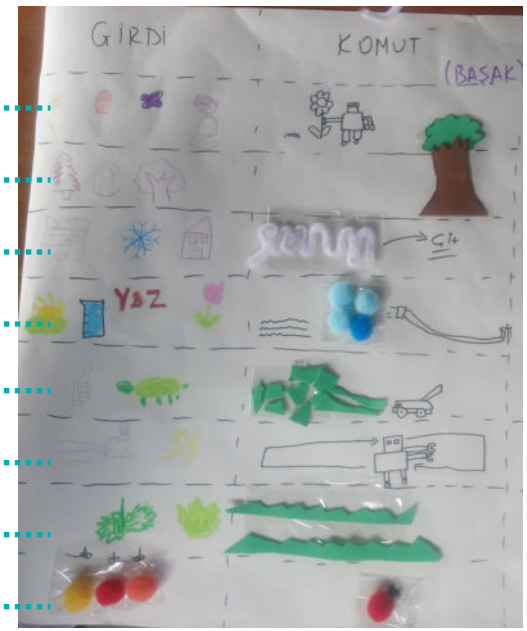
In this session, students created a programming interface for gardening with a total of eight commands, as decided upon earlier. Overall, we obtained forty different programming objects from five children. Here, we shared two distinct samples of these probes that shows different modalities of inputs.

After collecting all the objects and seeing how children created these tangibles, *we asked if these objects were qualified to become coding elements.* A **well-designed code element** should be easily understood by others and easily memorable by the author of the program. To this end, we tested the self- and inter- **memorability** and **understandability** of the programming objects. In this context, we defined understandability as the ability to correctly interpret the meaning of the given tangible objects by other students. And, memorability is the ability to remember the represented role of their own tangible object when they see the tangible object later.



Deniz offered using keyboard presses to change the given input. He shows the command first, then selects the possible inputs using keyboard shortcuts. Deniz's friend also shared the same design decisions, and resulted in very similar command-input pairs as expected.

Potting Flower
Planting Tree
Protecting Garden
Watering
Mowing Grass
Walking Around
Removing Weeds
Picking Fruits



Başak used a combination of drawing and given materials to create programming objects. Providing input to these commands can take different forms. For example, changing the flower's color in the robot's hand is the input of the "potting flower" command, but watering the garden requires combining two separate drawings.

TESTING THE MEMORABILITY AND UNDERSTANDABILITY

1

2

We conducted the test with four children from the previous studies (2 ten-year-old and 2 eleven-year-old). Before children arrived at our workshop area, we prepared a grid of programming objects on a table. We handed color-coded function names to children to test the understandability and memorability of the workshop's outcome.

1. We selected sixteen blocks (four tangible blocks from each participant) and placed them on a table.
2. We distributed color-coded small paper pieces and handed them to the children. Each child is assigned to one color to keep track of recognizing their tangible objects, and they place the enfolded paper pieces onto the grids. They placed this paper onto the grids in a closed form.
3. We discussed whether they remembered these commands or predicted their friends' tangible outputs. We tried to understand the effect of using different materials and different representation methods.

3

TEST RESULTS

On the next page, we summarized the test results using a table, which shows the understandability accuracy (top-right corner of each square), the correct label of the tangible object (bottom-left corner of each square), and each child's answer with color-code information (bottom-right corner of each square).

These results indicate that three main patterns affect the understandability and memorability of a programming object in general:

- Using materials that drive children to create abstract shapes (construction bricks) reduces future understandability.
- The object's recognizability also increases when children depict the action stated in the command rather than directly imitating the object's appearance. Trying to tell the action increases the level of details.
- Similarly, using more than one material increases the level of details.

Watering: Representing action as a tangible object with highlighting the correct details generally increases understandability. The blue bricks in the object representation was planned as water cannons, but they were interpreted as protecting gadgets or flowers.

Picking Fruits: This tangible representation states nothing (verified by the student; it was random). Surprisingly, this random object is correctly understood by most students.

Mowing Grass: Both tangible objects's motivation was using the toothed lego plate as the mower's knives. The top tangible's wings reduced the understandability score.

Protecting garden: Both tangibles represent an object related to protecting. The top one is a surveillance camera, and the bottom one is a fence.

Potting Flower: The accuracy of the object at the top was 75% since this student was out of paper pieces, so it was potting flower for this student; however, he couldn't place it.

Planting Trees: Contrast to the previous case, a tree is the only object that is used by commands. So, representing a tree is understandable at first sight.

Walking Around: Representing this command with a car seems reasonable, but an abstract car-shaped object can be used for other purposes like mowing and placing seeds.

Removing Weeds: Both tangibles represent the object of the action. In this example, mowing and removing weeds have a very similar object. In this case, it is better to represent the action.

DESIGN CONSIDERATIONS:

Based on our observation notes, material analysis of student's tangible creations, and memorability/understandability test results, we listed four main considerations to inform the design of student-made tangibles. We also considered the fact that these models will be used in the computer vision systems in real-life programming applications.

1

Give hints about the action:

The most understandable and memorable designs involve representing the object's visual resemblance and stated action together. For example, "watering a garden" can just be represented using a "hose." But, in our test, we observed that students can interpret this as a protection object and say the command is "protecting the garden." So, adding contextual layers and giving hints about the stated action supports the understandability of the self-made tangible.

2

Choose the right materials for the environment:

Considering the materials for different locations is an important aspect of the quality of programming time. If the students plan to use the objects more than one-time, using play-dough is not encouraged since it is cracking after drying. Also, considering the fact that these materials will be recognized by a computer vision system in the future studies, light conditions can affect the vision model's accuracy significantly. Similarly, using only wires to represent programming objects result in a high dependency on the foreground- background clarity. Therefore the environment should be considered while choosing the materials.

3

Add details with 'simple' shapes:

Adding details helped students to memorize the function of designed tangible. Yet, we emphasized adding details with 'simple shapes' to use students' time more efficiently. In the workshops, they tended to add details with 'fancy' materials and zigzagged scissor moves which distract them from the tasks.

4

Combine materials:

Combination of materials enhances the interpretability of programming objects from the student perspective. Using combinations of different textures and colors can also increases the accuracy of the computer vision model's recognition.

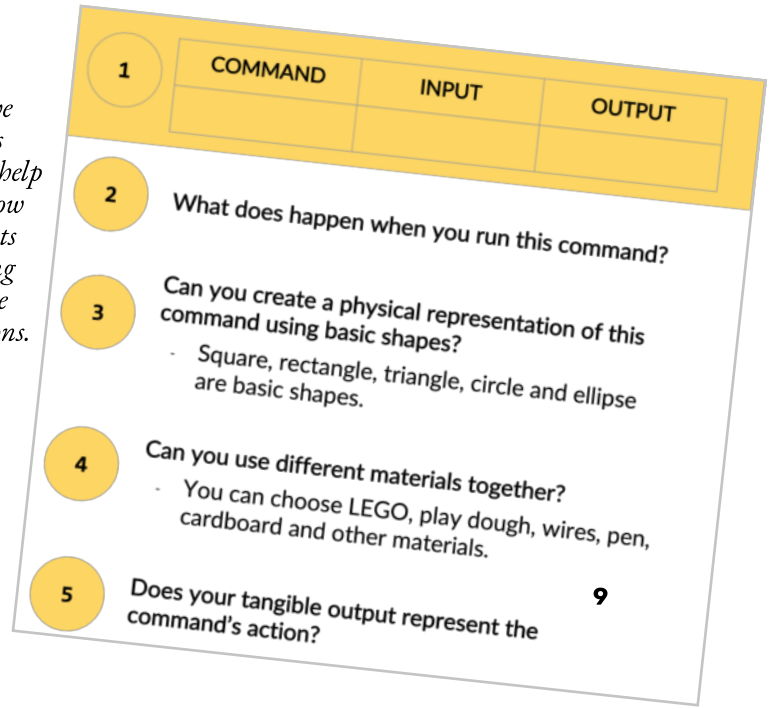
MATERIAL ANALYSIS:

Using Play Dough, Wires and Other Shape-Changing Materials: They are great for adding details, but not longing last, which decreases the repetitive use of self-made tangibles as an algorithmic structure.

Using LEGO as Building Blocks: Five students created total 40 tangible function representations, and 15 of these tangibles are completely created with LEGO bricks. Although students enjoyed using these bricks, the understandability and memorability results are considerably low, compared to other materials.

This photo shows all the LEGO bricks used in the studies. Most of the creations consist of ~4-7 bricks. They combine different colors.

In the end, we prepared this worksheet to help children follow some checklists while creating their tangible representations.

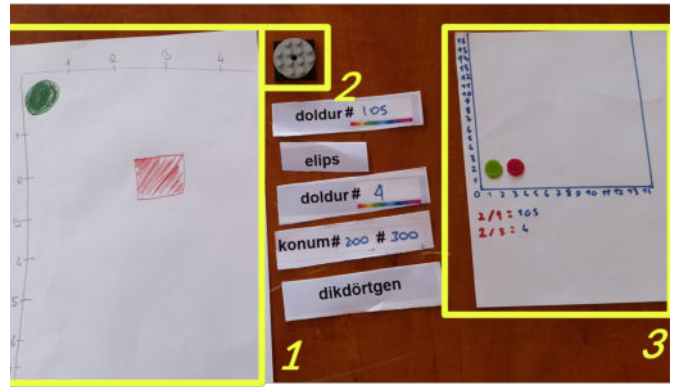


APPLYING THE DESIGN CONSIDERATIONS ON PAPER PROGRAMMING CODES

To test our design consideration's efficacy in student-made tangibles, in the final study, we created tangible representations for four different creative coding examples. Two children (Student A and Student B) who did not participate in previous studies used Kart-ON paper programming blocks and created self-made tangibles as function definitions for the paper programs. Here, each frame shows the code in the middle, Student A's output on the left, and Student B's outcome on the right. We numbered each output to ease the referencing. During the session, we talked with children about their intended desing and helped them to keep track of the worksheet.

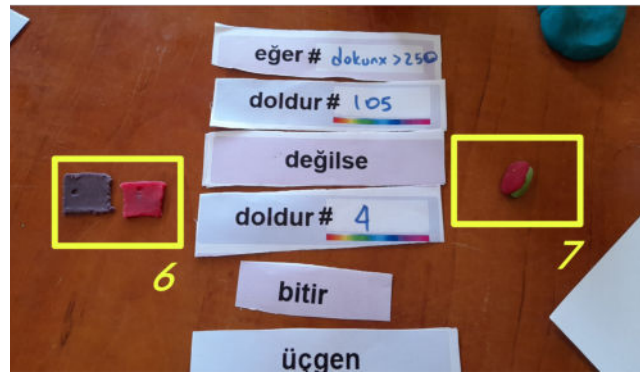
In the previous studies, children created tangible representations of functions that they are already familiar with from real life, such as "watering the garden". Building tangible representations of these functions can be seen as an easier task when compared to more abstract programming blocks like conditional statements or loop structures. In the study, we first gave a programming task and asked to complete it using Kart-ON programming commands. After creating each code, we ran the code to see the output. Then, we asked students to create a tangible representation to save this code as a function.

Using Simple Shapes



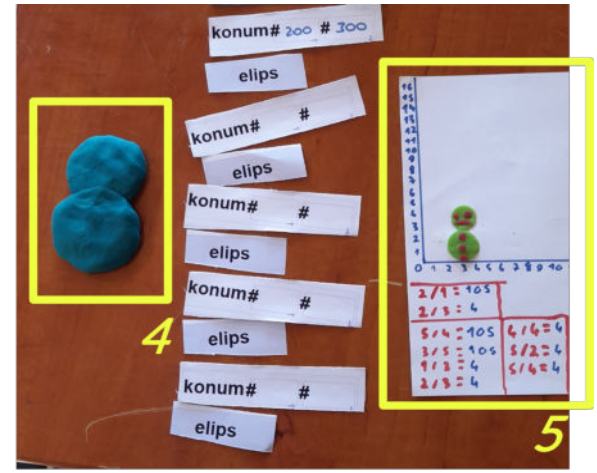
Drawing a green ellipse and red square on different locations. ***Box#1*** *shows a drawing of the end result and* ***Box#2*** *is also part of this tangible design.* ***Box#3*** *uses a similar approach where ellipse and rectangle are made of play dough.*

Changing Colors Of A Rectangle Using Conditionals



Drawing a triangle either green or red based on touch input. ***Box#6*** *uses two play-dough rectangles to show the color changing operation.* ***Box#7*** *is a one piece play dough that contains two adjacent triangles.*

Drawing Snowman By Combining Multiple Shapes



Drawing a turquois snowman using the "fill", "location", and "ellipse" commands. ***Box#4*** *consists of two blue play dough ellipses.* ***Box#5*** *follows a similar approach with Box3 and uses two green smaller play dough ellipses on a coordinate system drawing.*

Drawing A Series Of Rectangles With A For Loop

Drawing 50 consecutive rectangles in a vertical line. ***Box#8*** *have 50 lines that indicates the loop count.* ***Box#9*** *is a LEGO gear that completes many cycles and stops when the loop count is completed.*

DISCUSSION

We observed that while creating and using self-made tangibles for algorithmic tasks, the screen-free time accelerated the active and collaborative learning experience. Additionally, using self-made tangibles helped students focus on a code scope's overall working mechanism. We developed design considerations based on workshop outcomes to enrich semantic layers in the building phase of tangible objects, which increases the object's understandability and memorability as a programming block. Nevertheless, our preliminary tests of the design considerations yielded it to be inefficient at this point. Defining programming elements in a tangible form seem to be a challenging task still. Distributing predetermined teacher-made materials can be a better approach in programming flow.

At the end of the final study, our observations demonstrate that asking students to create tangible representations helps them understand the code, which resonates with the research in active learning [2]. For example, using student-made tangibles in math learning of young children can help them engage in activity, familiarize the concepts in real-world situations and help them understand relational information better. We observed similar benefits in our experiments, and this kind of art and programming activities have already been integrated into the classroom by teachers. Although these activities can be useful in engaging students and promote active learning, we conclude that using this kind of interaction in tangible interfaces needs lots of guidance for students and teachers to become usable in abstract definitions of programming elements such as defining new variables and functions.

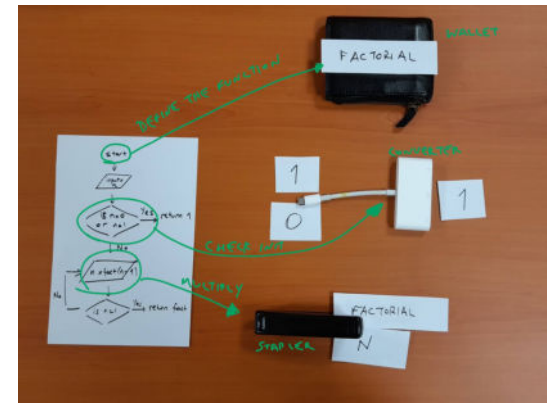
CONCLUSION

In this paper, we explored the potential use of self-made tangibles in a programming environment with four workshops that gradually investigates and evaluates tangible programming tasks from decomposing problems to utilizing tangibles with computer vision models. Based on our qualitative observations and analysis, we can summarize the answers of the main research questions as following,

Can students in K3-6 grades build a tangible representation of abstract programming definitions?	Yes, we observed that students liked building tangible representations of abstract programming definitions, and they could link the function with the representation.
Can students understand and memorize self-made tangible representations in later uses?	It is not an easy task, even with the guideline, and it requires further research. Our list of design considerations helped students manage their time and use the materials in a structured way, yet they still had challenges in following them.

FUTURE WORK:
Extending the Considerations for More Abstract Structures

Throughout the user studies, we used self-made tangibles in physical actions such as coding a high-level robot or an arcade game. For example, “potting a plant” command of gardening task can be effortlessly translated into an imaginary picture. In this sense, limited high-level commands are transferable to the tangible domain. Although it is not the scope of this project, one might question the possibility of using personally meaningful objects in this kind of setup. For example, the factorial function, which is denoted as n! and is equal to $n! = 1*2*3*...*(n-2)*(n-1)*n$, can be recursively defined. As a first step, we can draw the flowchart of this pseudocode to make it more tangible. Then, following the design considerations we created, we can design new or use existing tangible objects to represent the commands. Finally, we should define a way to link inputs to the commands. For example, only using the office objects, we created the structure in figure below.



We would like to emphasize that the primary goal of these studies was to explore self-made tangibles to design a language that can help students grasp computational thinking skills rather than designing a low-level language. Yet, exploring these open research questions can be helpful to extend these workshops' outcomes in an embodied medium such as *Dynamicland*.

REFERENCES

[1] Allison Druin. 2002. The role of children in the design of new technology. Behaviour & Information Technology 21, 1 (jan 2002), 1–25. https://doi.org/10.1080/01449290110108659.

[2] Emily R Fyfe, Nicole M McNeil, Ji Y Son, and Robert L Goldstone. 2014. Concreteness Fading in Mathematics and Science Instruction: a Systematic Review. Educational Psychology Review 26, 1 (2014), 9–25. https://doi.org/10.1007/s10648-014-9249-3.

[3] 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. 379–387. https://doi.org/10.1145/3341215.3356264

[4] 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.

[5] Carnegie Mellon University. 2019. Alice. Tell Stories. Build Games. Learn to Program. https://ww.alice.org/

[6] 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.

[7] 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. 410–413. https://doi.org/10.1145/2771839.2771866

[8] Hayes Raffle. 2010. Topobo: programming by example to create complex behaviors. In Proceedings of the 9th International Conference of the Learning Sciences - Volume 2 (ICLS '10). International Society of the Learning Sciences, 126–127.

[9] 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.

[10] 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

[11] 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

[12] Cecily Morrison, Nicolas Villar, Anja Thieme, Zahra Ashktorab, Eloise Taysom, Oscar Salandin, Daniel Cletheroe, Greg Saul, Alan F. Blackwell, Darren Edge, Martin Grayson, and Haiyan Zhang. 2018. Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities. Human-Computer Interaction 00, 00 (2018), 1–49. https://doi.org/10.1080/07370024.2018.1512413

[13] 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.

[14] Paul Marshall. 2007. Do Tangible Interfaces Enhance Learning?. 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, 163–170. https: // doi.org/10.1145/1226969.1227004

[15] Yasmin Kafai. 1994. Minds In Play: Computer Game Design as a Context for Children's Learning

[16] Jennifer A Kaminski and Vladimir M Sloutsky. 2020. The use and effectiveness of colorful, contextualized, student-made material for elementary mathematics instruction. International Journal of STEM Education 7, 1 (2020), 6. https://doi.org/10.1186/s40594-019-0199-7

[17] 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.

[18] Alpay Sabuncuoglu. 2020. Tangible Music Programming Blocks for Visually Impaired Children. In Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction (Sydney NSW, Australia). 423–429. https://doi.org/10.1145/3374920.3374939

[19] Maarten Van Mechelen, Mathieu Gielen, Vero vanden Abeele, Ann Laenen, and Bieke Zaman. 2014. Exploring Challenging Group Dynamics in Participatory Design with Children. In Proceedings of the 2014 Conference on Interaction Design and Children (Aarhus, Denmark). Association for Computing Machinery, New York, NY, USA, 269–272. https://doi.org/10.1145/2593968.2610469

[20] 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

[21] Oren Zuckerman, Saeed Arida, and Mitchel Resnick. 2005. Extending Tangible Interfaces for Education: Digital Montessori-inspired Manipulatives. CHI'05 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (2005), 859–868. https://doi.org/10.1145/1054972.1055093 , Vol. 1, No. 1, Article . Publication date: January 2022.

[22] 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 (CHI '97). Association for Computing Machinery, New York, NY, USA, 234–241. DOI:https://doi.org/10.1145/258549.258715