Toque: Designing a Cooking-Based Programming Language For and With Children

Sureyya Tarkan[1,2], Vibha Sazawal[1,2], Allison Druin[1,2], Evan Golub[1,2], Elizabeth M. Bonsignore[2], Greg Walsh[2], Zeina Atrash[3]

[1]Department of Computer Science
University of Maryland
College Park, MD 20742 USA
sureyya@cs.umd.edu

[2]Human-Computer Interaction Lab
University of Maryland
College Park, MD 20742 USA

[3]Center for Technology & Social Behavior
Northwestern University
Evanston, IL 60201 USA

ABSTRACT
An intergenerational design team of children (ages 7-11 years old) along with graduate students and faculty in computer science and information studies developed a programming language for children, *Toque*. Concrete real-world cooking scenarios were used as programming metaphors to support an accessible programming learning experience. The *Wiimote* and *Nunchuk* were used as physical programming input devices. The programs that were created were pictorial recipes which dynamically controlled animations of an on-screen chef preparing virtual dishes in a graphical kitchen environment. Through multiple design sessions, programming strategies were explored, cooking metaphors were developed and, prototypes of the Toque environment were iterated. Results of these design experiences have shown us the importance of pair-programming, programming by storytelling, parallel programming, function-argument relationships, and the role of tangibility in overcoming challenges with constraints imposed by the system design.

Author Keywords
Tangible UIs, education, children, design, programming languages.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation: User Interfaces – User-centered design.

General Terms
Design, Human Factors, Languages

INTRODUCTION
In today's increasingly technological society, people of all ages can benefit from understanding computational thinking and problem solving [1]. This knowledge helps people comprehend how and why technology works. While there are numerous ways to equip people with problem-solving skills, teaching computer programming is a well-established educational approach that not only presents computational thinking but also empowers users to design and modify software to suit their own needs [24]. Although programming skills can provide extensive benefits to learners throughout life, from strengthening vocabulary development to higher cognitive skills such as planning abilities and experience with problem-solving heuristics [3], the experience necessary to acquire these programming skills have become harder to find due to a lack of emphasis in recent years on programming in our mathematics curricula on all levels [20].

Figure 1. 8-year old boy using a *Wiimote* and *Nunchuk* to create an animation of on-screen chef preparing a virtual baked cheese and tomato sandwich and the equivalent recipe instructions in icon-based form.

To explore ways in which to provide children with more programming experiences and skills, we worked with an intergenerational design team composed of children (ages 7-11) and adults (computer and information scientists) to design a new visual and tangible programming environment for children called *Toque* (Figure 1). In this paper, we describe how we evolved the design of Toque, in which children can instruct an on-screen chef how to make a virtual dish. This iterative design work centered around programming a baked tomato and cheese sandwich recipe. This paper reports on design sessions that investigated the relationship between various tangible interactions and

CHI 2010, April 10-15, 2010, Atlanta, Georgia, USA.

computational thinking skills. We found that the cooking domain revealed several new, interesting findings about computational thinking with respect to programming by storytelling, pair-programming, parallel programming, the importance of order in command structure, and the role of tangibility in overcoming challenges with constraints imposed by the system design.

RELATED WORK

Several relevant areas of research inspired our work. These areas include visual programming languages, tangible programming systems, and games.

Visual Programming Languages

There is a rich history of research on visual programming languages as reported by Kelleher and Pausch [12]. An example, *Alice* [2], helps novice programmers learn programming by building 3D virtual worlds. *Storytelling Alice* [13] includes additional features to attract children that prefer a storytelling style of programming. Another example, *Scratch* [15] is a graphical programming environment that young people can use to create interactive stories, games, music, and art by snapping together building blocks. While these programming environments have been used by a large community, the relationship to tangible input devices have not been a focus.

ToonTalk [11], *Hands* [21], and *Kodu* [14] are programming environments that children can specifically use to make games. ToonTalk and Hands both present fanciful worlds in which the program is written in cartoon-like thought bubbles. Inside these worlds, elements such as birds, nests, and playing cards represent programming concepts or constructs. Kodu supplements this line of game-creation tools with *Xbox* features so programming can be done with a game controller. All three of these tools support rule-based programming languages that are natural for games (e.g., "when the dog eats the biscuit, give the dog an extra life"). While these programming languages have gathered a following, especially for solo individuals, tangible programming has been successful in fostering and reinforcing motivation to learn programming through social relationships [6].

Tangible Programming Systems

As McNerney [18] reports, there is considerable work in the area of physical systems for programming. Here, we report those that are specifically for young learners. *Electronic Blocks* [29] are tangible programming elements that young children can stack together to form robots that crash into each other, or lights that flash with a clap. Later, Zuckerman et al. [30] introduced the classification of *Montessori-inspired Manipulatives* (MiMs) that foster modeling of conceptual and abstract mathematic structures. Examples of this concept include computationally enhanced building blocks, *MiMs: SystemBlocks* that can simulate dynamic behavior, and *FlowBlocks* that can simulate abstract structures of dynamic processes.

Fernaeus et al. [6] developed a tangible programming space that enables children to collaboratively program play-worlds on a computer screen using physical objects. In construction mode, tangible objects can be programmed with behaviors by placing tokens onto a physical grid space. During the execution mode, the corresponding virtual picture acts according to the tokens. *Tern* [9] is a tangible programming language that enables students to create programs with passive blocks shaped like puzzle pieces that have no embedded electronics or power supplies. After a "writing" step, children "camera compile" their code to run physical or virtual robots. Both of these systems have *two modes* of interaction, which assumes the ability to design algorithmically in offline settings.

However, Montemayor et al. [19] found that switching between modes can be confusing to young children (5-9 years old). In their system, *StoryRooms*, a Physical Programming environment, the researchers developed computationally enhanced, tangible toys that can be used to program an entire room or any physical interactive space. In programming mode, children create stories, and in use mode, the stories are "executed" (played like a live-action movie). The lessons learned by Montemayor et al. need to be more widely considered by designers of tangible programming environments. Modes confuse children and better modeless metaphors are needed.

Games

Another area that inspired our research is children's games that have cooking themes. One such game is *Cooking Mama* [5], developed for the Nintendo Wii. Cooking Mama is a cooking simulation in which dishes are prepared by completing steps in the meal preparation process. Similarly, *Cooking Star* [7] is a mini-game application for the Apple iPhone. It lets the user tilt, touch, flick, and flip to cook meals using the iPhone's accelerometer capabilities. Yet another game is *Pizza Palace* [23], a Webkinz offering that can be played online. In Pizza Palace, the player attempts to serve customers of a pizza shop who custom order pizzas and the player can purchase equipment along the way to make the job easier. As more cooking-centric entertainment games emerge, this suggests a motivating framework with which to teach programming [26].

OUR WORK

Although there have been attempts to bring together child programmers via online communities [15], we wanted to understand how programming can be done in a social way with face-to-face, collocated context. Our focus therefore, has been on exploring new design directions for computational languages that can potentially support children who are not necessarily mathematically-able or explicitly interested in programming, but who do have access to digital technology. We look to support those children who may not naturally want to sit alone, silently and stationary in front of a computer screen and keyboard, for an extended period of time.

To do this, we worked with children (ages 7-11) as co-designers in our lab to explore prototype directions, refine design ideas, and better understand new possibilities [4]. This work is a part of our ongoing design partnerships with children at the University of Maryland. Since 1998, we have been working with children using design methods to bring together the ideas of children and adults.

In the remainder of this paper, we present how we worked with our intergenerational design team to design Toque. We will also discuss implications of the children's design suggestions.

Design Process

In developing the Toque prototype, children did not merely use, test, or inform the design; they actively helped create it. These children (ages 7-11) came from the local Maryland area and half attended private schools and half public. Only one had prior programming experience. There were three African-American children, three children of Asian decent, and four Caucasian children. Five were girls and four were boys. These children were part of an existing research team that meets twice a week after school and two weeks over the summer to develop new technologies that range from mobile technologies to digital libraries. Children on the team stay an average of two years, but some have stayed as long as 5 years.

The team uses a variety of low-tech prototyping methods [4] using art supplies to sketch and model ideas. We also used these methods to refine specific aspects of the system. The low-tech prototypes are not end products themselves, but rather a means to generate design discussion, new ideas, and directions. To discover approaches for a new programming language, adults and children worked together on developing the framework for, and intermediate prototypes of, the Toque system.

The Evolution of the Toque Prototype

We began the design sessions in summer 2008 with the idea that we wanted to create an accessible programming environment for children [26]. Our design discussions started by asking the team to "program" a robot, played by one of the adults. The "programming" moved from verbal commands to simple words and images. We also explored what we wanted our "robot" to do. Computer games, petting, and cooking (see Figure 2) were among the most popular ideas. In our second session that summer, we asked our child co-designers to make virtual hot chocolate using our wizard-of-oz tangible prototype. From our discussions, cooking emerged as a useful metaphor that appealed to both males and females in our group. Ultimately, it was decided that an on-screen robotic chef would be programmed to cook virtual food using tangible objects. By summer 2009, an initial cooking prototype was built and ready to be used in the lab. It enabled a child to use a Wiimote to program a recipe to cook, which is illustrated in detail in the next subsection.

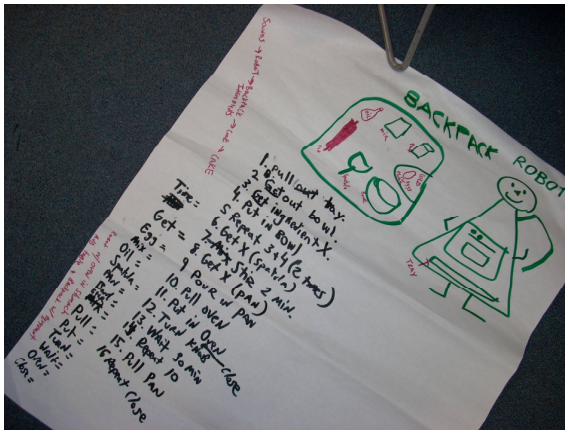

Figure 2. The robotic chef idea designed by children that guided Toque's implementation.

We designed physical gestures, which could be recognized via the Wiimote, for a set of common cooking actions such as *cut* and *put*. Although the Toque language is being designed with the long-term goal of having multiple cooks in the kitchen, the initial prototype supported only one Wiimote and thus one cook. A large-screen display was used to project the computer's output and the Wiimote was connected to a computer via Bluetooth [27] so the children would be able to perform gestures in free space without a tangle of wires. We also used the Wiimote to control the mouse pointer at times and to provide a way to issue an "undo" command.

The Wiimote gestures were used to control a cook, displayed on-screen via an animated virtual kitchen environment (see Figure 3a). To help provide feedback the system generated a visual representation of their cooking instructions. This visual representation made use of an icon-based language (see Figure 3b). To implement the virtual kitchen, the software was built upon a modified version of Alice 2.0 [2], along with some models built using *SmoothTeddy* [10].

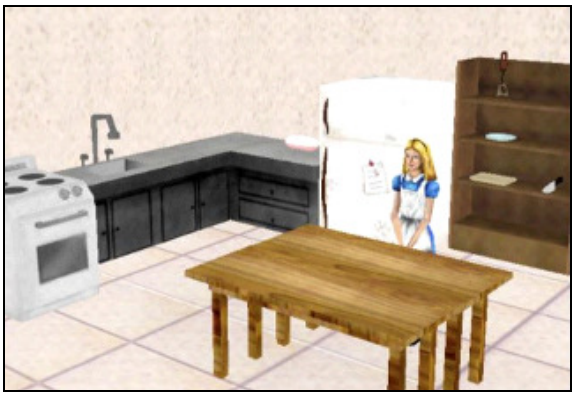

Figure 3a. The frame of the Toque application showing the graphical simulation of the kitchen space.

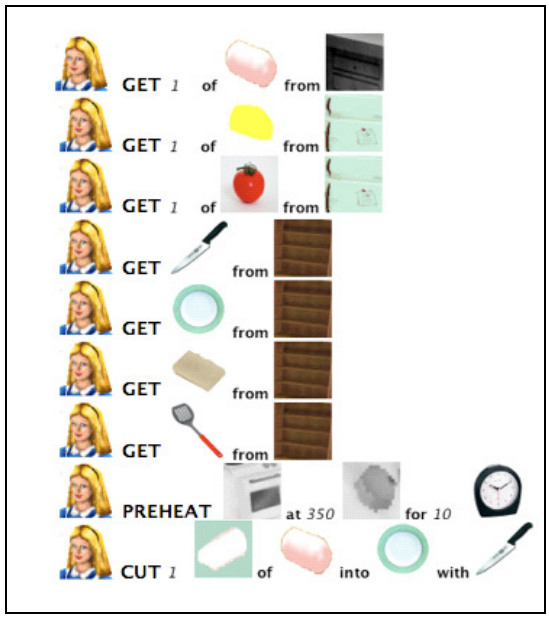

Figure 3b. The frame of the Toque application showing the actions that had been taken, using an icon-based language.

Get 1 loaf of bread from the counter
Get 1 roll of cheese from the fridge
Get 1 tomato from the fridge
Get 1 knife from the shelf
Get 1 plate from the shelf
Get 1 tray from the shelf
Get 1 spatula from the shelf

Preheat the oven at 350F for 10 mins
Cut 1 slice of bread onto the plate with the knife
Clean the knife in the faucet
Cut 1 slice of cheese onto the bread with the knife
Clean the knife in the faucet
Cut 1 slice of tomato onto cheese-bread with the knife
Clean the knife in the faucet
Cut 1 slice of bread onto tomato-cheese-bread with the knife
Put the sandwich onto the tray with the spatula
Bake the sandwich for 2 mins in the oven
Get the sandwich from the oven

Figure 4. Baked cheese and tomato sandwich recipe presented in textual form.

To facilitate the exploration of different tangible means, we worked on a single recipe for a baked cheese and tomato sandwich. Because we did not build models for all possible items and ingredients that could reside in a kitchen environment, the initial Toque prototype only supported the creation of one class of programs related to cooking this recipe, but a variety of programs are possible using these basic objects and functions. When the time comes for more formal evaluations, it is our intention that users will be able to create programs with or without written instructions but in our design sessions, each team member received a paper handout which contained a written recipe both in a pictorial (Figure 3b) and a textual (Figure 4) forms. The pictorial version used our icon-based language. By asking our child co-designers to replicate this concrete example using our interface, we were able to explore their design ideas for the interface and metaphor.

Programming a Recipe to Cook

Toque currently incorporates programming concepts such as events, objects, primitives, methods, arguments, and loops. The formal specification of Toque's language is reported elsewhere [25]. To illustrate how the system works, consider the following example of cutting one slice of bread on a plate with a knife. The first step to expressing this instruction is to perform a *cut* gesture with the Wiimote. As soon as the physical action is captured by the system, the animated cook responds with the question "What should I cut?" in her thought bubble. This requests the first argument to the function, i.e. *bread*. Once the user selects the *bread*, a dialog box presents a question about the unit of quantity that will be used: "Is it in *slices*?" which the user confirms by clicking "Yes" in the dialog box. Next, the actual quantity is obtained via another dialog box: "How many slices should I cut?" whose value should (in this case) be *1*. After these responses have been processed, the cook asks, "Where should I cut the bread?" the answer to which is the *plate* object. The final question is as follows "What should I use to cut with?", and the user provides the *knife* object.

At this time, one line of icon-based code gets added to the program and the cook performs the instruction. The animation begins with the cook moving the *plate*, *knife* and *bread* to be in front of her. She then performs a cutting gesture using these items. This causes a *slice* of bread to become visible on the *plate*. Lastly, she moves the *plate* (along with the *slice*), *knife*, and *bread* back to their initial location and stops the animation to wait for another instruction.

Figure 5 depicts how the child's physical actions tie into the virtual world. Every function has a number of arguments, each of which has an associated type. The arguments can either be objects (ingredients, kitchen items such as a knife or spatula) or primitives (integers, booleans). Depending on the argument type, the user either enters a value by pressing the Wiimote's **+/-** buttons or moves the Wiimote as a mouse cursor to point at objects. To confirm the selection, the user has to "click" via the **A** button of the Wiimote.

The system requires all of the arguments to be entered correctly; otherwise, the cook complains in the thought bubble used to allow the chef to "talk" to the child (as seen in Figure 5, items 1 and 5). For example, if the user has picked a tomato (of type ingredient object) when the cook was expecting a knife (of type kitchen item object) to cut with, the cook replies, "That can't cut!" If the user tries to bake a sandwich in the oven without preheating the oven beforehand, the cook says, "Not heated!" Successful completion leads the cook to perform the indicated actions, displayed via animation and a line of icon-based code

added to the recipe displayed on the right-hand side of the visual environment (as previously seen in Figure 4b). Both the animation and icon-based representation are dynamically interpreted rather than being compiled and executed, i.e. the children neither used a compiler nor produced an executable.

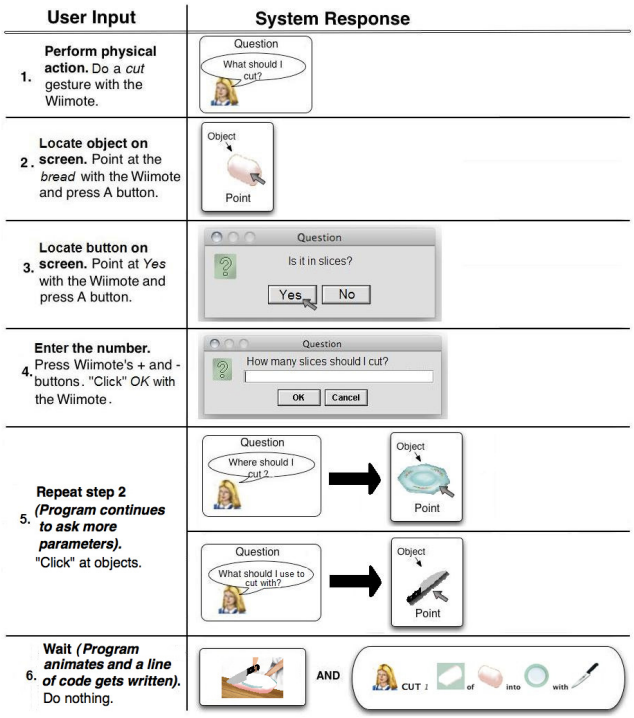

Figure 5. ***Usage scenario diagram for the Toque system.*** **By executing steps 1 through 6 above, user finishes a single line of input for Toque to simultaneously convert this given information into an animation and icon-based representation.**

Design Iterations

With this prototype, the team then explored various design directions for Toque. Half of the team played at a time with the interactive version of Toque. After they used the prototype, likes and dislikes were written on sticky notes (one idea per note). These notes were gathered on a white board and a frequency analysis was done to look for challenges and affordances of the system. One area that was of concern for the children on the team was the ability to point at objects using an absolute positioning system, built with Wiimote's IR tracking capabilities and Wii Sensor Bar. Children's hands would shake and the objects and buttons on the screen were not big enough for this type of interaction. Entering numbers by pressing Wiimote buttons was overwhelming.

While this design work was occurring the other half of the team, who had no experience with the prototype, developed new design ideas for a cooking system that involved a Wiimote as a controller. In a low-tech prototyping session, the group designed add-ons and gestures for cooking instructions and measurements (see Figure 6). In their designs, all tools were attached to the Wiimote and they mapped each newly designed input devise with a motion that could be performed after the selection of the specific item. It seemed that the physical input device really defined the interaction.

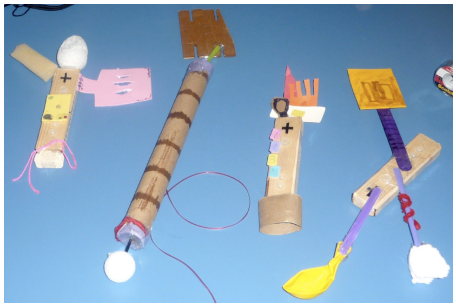

Figure 6. All Wiimote designs from the low-tech prototyping session.

Modifying the Interaction

After the design exploration was accomplished (see previous subsection) the following week was spent developing a new iteration of Toque based upon the design ideas that were generated for the physical input devices and the programming language.

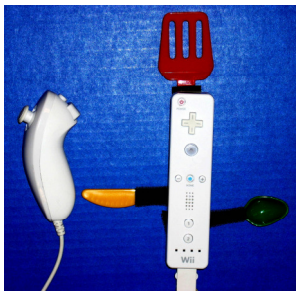

Figure 7. Additions to Wiimote and Nunchuk suggested by design team.

One change that was quickly implemented was enabling the programmer to use the Wii Nunchuck as a mouse rather than using the Wiimote for that purpose. This was done because the Nunchuk had an analog stick which could be used for relative pointing. We came to understand that younger team members found it far too challenging to point at objects precisely with the Wiimote. Based on the low-tech controller prototypes designed with the children (shown in Figure 6) we added utensil selection "accessory" extensions to the Wiimote. We attached a toy spatula, spoon, and knife to the Wiimote with hook-and-loop fasteners (see Figure 7). We expected these tools might be selected for use in combination with the gestures.

As suggested, we also incorporated an extra knob implemented with a Phidgets rotation sensor [8] into our design to help users input numeric values (Figure 8). We also updated the software based on team-generated ideas so that Toque will infer measurement values and units in certain scenarios, such as "1 slice" being automatically assigned as the measurement associated with cutting

ingredients. However, if the measurement is variable, the cook asks users to enter it (e.g. the number of degrees to preheat the oven).

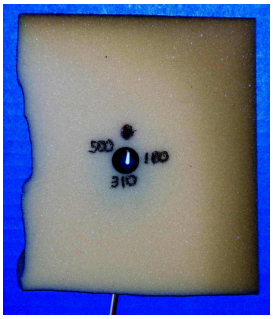

Figure 8. Control knob built based on design team suggestion for setting numeric values (0, 1, 2, …, 498, 499, 500).

SECOND DESIGN SESSION DIRECTIONS

The following week this revised prototype was brought back to the design team to again make cheese and tomato sandwiches. This time the issue concerning making multiple sandwiches (loops) was also explored.

In this section, we report on four of the key lessons learned. These lessons could also be applied to a variety of programming systems, not only Toque.

"Computer, understand our intent": The Ordering of Actions and Objects

During the second session, part of the design team identified potential revisions to Toque's icon-based language. Team members prepared a recipe to make French Toast using pictorial "programming cards" as part of a low-tech prototyping session. In order to facilitate the code writing process a large supply of cards were prepared, each had either a cooking-related icon or word. There were also blank cards provided, on which team members could write their own customized keywords or draw additional "key pictures" not included in the existing icon library.

As Figure 9 shows, the team members worked together and wrote a very detailed and structured program. Since we were interested in learning more about how the children would think about constructing a program, we encouraged team members think and create without being constrained by Toque's language definition on a table rather than in a fixed structure.

During this experience, team members became annoyed because utensils had to be washed every time there was a new instruction. Rather than waste time washing, they picked up another clean item each time. The model of cleaning utensils and reusing them was useful from the perspective of designing the virtual kitchen and keeping track of ingredients and items. However, it was not intuitive for the children and did not match how they would cook in their own kitchen. Children said they would probably not clean a knife that had been used during the recipe but would instead use another clean knife in real life. They explained cleaning is the last step of their meal preparation process and wanted to omit this final step altogether from their programs. Team members did not like that utensils changed state from clean to dirty, but they did understand the notion of ingredients changing states (e.g. eggs changing their state from "whole" to the "cracked" eggs in bowl). It was not simply a matter of not adapting to virtual objects changing states; it was that the constraints of the environment did not match their own real-world expectations.

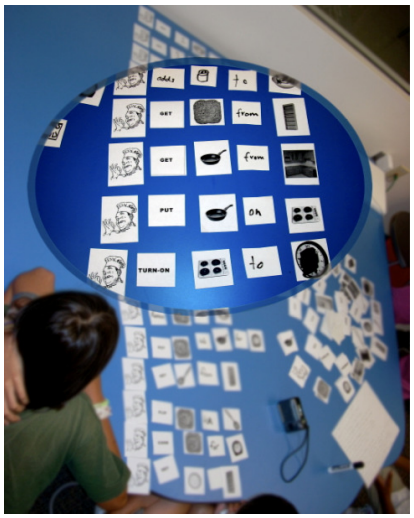

Figure 9. Children are programming French Toast recipe with cards to suggest revisions to the icon-based language.

In addition, we also found issues with the syntactic ordering of cooking actions. To write individual programming instructions, members arranged icon cards and text cards. Different members of the design team members preferred to arrange cards in different ways. Some used an action-object ordering for which others used an object-action ordering. This is in contrast to the traditional single-dispatch OO syntax, which requires the receiver first, then the method name, and then arguments in a specific order. Figure 10 presents an example of a full "line of code". Here, the object *whisk* came before the *mix* action; in typical OO programming, *mix* would be a function of *Cook* that took a *Whisk* as an argument. In this instance, *mix* was treated as an object that determined the type of action.

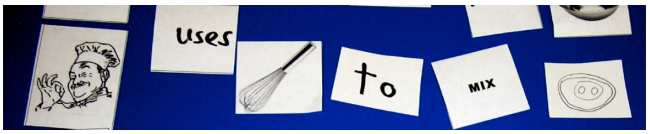

Figure 10. An example of an instruction where the object is prior to the action.

When using the interactive prototype, team members also wanted to enter instructions using various orders for actions and objects. Team members would sometimes feel that selecting the arguments first was more intuitive than choosing the action so often they wanted to write an instruction in which they started by selecting an object.

Since our prototype did not support interchange of arguments with actions, young team members were either confused or frustrated that their input was not accepted by Toque (the cook did not ask questions contrary to their expectations). They would try entering the same instruction a few more times until they were assisted by one of the adult partners and encouraged to think about actions first. In future work, we intend for all plausible orderings to be accepted by Toque.

"Confusing and boring": Loops

One aspect the team focused on in the second session was the use of a loop. We asked our young design team members to program a recipe that would lead to making *four* baked cheese and tomato sandwiches. Toque's initial implementation for iteration consists of a *do-while* construct. The Wiimote's **up** button is used to open the loop and the **down** button is used to close the loop. The loop count could be entered either with the knob device or the Wiimote's + and - buttons. To explore this concept from both the interface and the language perspectives, we divided into two subgroups. While the first sub-group of the team was using Toque, the second sub-group worked with the programming cards. The first sub-group discussed which instructions were related to sandwich-making and thus were necessary to repeat when they program in Toque. As can be seen in Figure 11, by drawing on the paper handouts and doing a little math, our young partners were able to decide which instructions needed to be repeated, and how many times.

During iterative programming, the children in the team were turning their heads away from the handouts, concentrating on things unrelated to the task, putting very little effort on the math calculations, giving incorrect answers to questions without thinking, and expecting to finish before all the instructions were considered. It was far from easy for them even with adult guidance. Since we were doing low-tech prototyping, we decided to not be constrained by the existing Toque language. Therefore we shifted to live role-playing with a human cook (see Figure 12) instead of using the live Toque system or even the icon-based language.

The children guided the human cook to prepare four make-believe sandwiches out of paper and plastic materials, using the Wiimote and the Nunchuk as props. The children told the cook to remember which instructions she had to learn (opening the loop body) so that she could repeat them once told to do so (closing the loop body). Within this setting, children appeared better able to grasp the idea of iteration. With the aid of the more relaxed constraints and no need for precision with the controls, the children were able to prepare their four sandwiches. When building a loop construct on the interactive tangible system, the children seemed to struggle with the start and end of a loop, and how many times to repeat the body. With text-based programs, it is easy to insert and modify previously written code to open and close a loop body, but in Toque, it is difficult to insert/modify parts of the code already executed unless numerous undo commands are issued or the user starts again from the beginning.

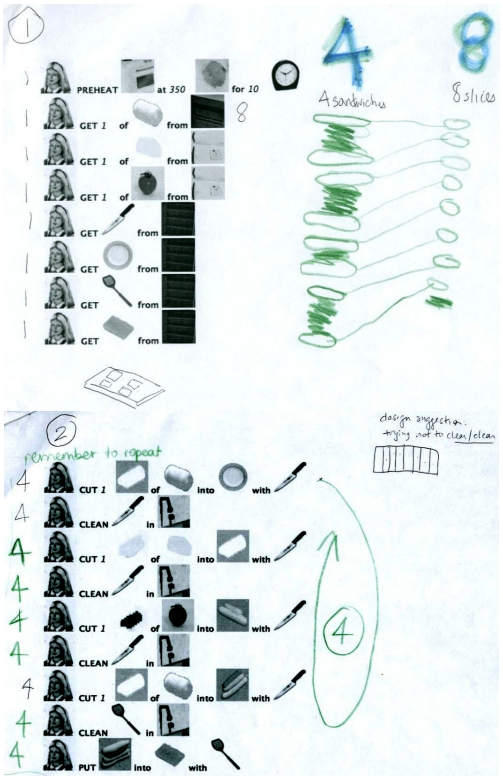

Figure 11. The annotations on the printed Baked Cheese and Tomato Sandwich recipe to implement loops.

Figure 12. Children "programming" a human cook to prepare four make-believe baked cheese and tomato sandwiches.

The second sub-group of the team approached looping from a slightly different direction. As can be seen in Figure 13, they decided to add an argument to a function that served the same purpose as creating a loop. In this particular case, cracking *two* eggs were combined into a single instruction rather than repeating a single instruction twice.

Figure 13. An example of an instruction where the loop is added as an argument to the function.

This raised the possibility that loops should be able to refer to a block of instructions. It is possible that children might prefer a function that takes an ordered list of instructions (block) that should be repeated a number of times. This alternative approach to iteration is similar to how the Loop construct is used in the Alice programming language [2]. On the other hand, Hands [21] avoided iteration altogether by using set notation.

"Let's program together": Sharing the Wiimote and Nunchuk

With the addition of the use of the Nunchuk as mouse controller in the second session, two children working as partners would spontaneously share the Nunchuk and Wiimote while programming the loop, as shown in Figure 13. The children explained that pairing with another person was more comfortable than using both controllers on their own. When one child used both, it was inconvenient to constantly switch hands or to use only the dominant hand to carry both. Having "enough hands" was even more of an issue while carrying the paper handouts containing the target recipe. Hands were not big enough to carry everything together.

As a result of this tie between pairs of children, our young designers were discussing with each other about what to do next and how to do it. They also decided to carry the recipe handout based on controller use. While the pairs of children sometimes switched who held which controller, whoever was holding the Nunchuk would take and hold the recipe sheet in the other hand. One possible explanation for why they switched the handout along with controllers is that performing the coding gestures with the Wiimote was a more involved activity than pointing at objects with the Nunchuk.

"I want to tell my story": Cooking Stories over Chef Controls

When a virtual chef needed more information, it would ask first-person questions, such as "What should I cut?" Some of the children on the design team quickly criticized this initial design choice. Instead of telling the cook what to do, they preferred to think of their program as a story about the cook. The cook had his/her own destiny, and they were just relating/retelling the cook's personal story.

When using the programming cards during brainstorming, this preference for story narratives became clearer. Some of the children added keywords for things unrelated to cooking such as *eat*ing the meal. One child even wanted to add rat poison to a French toast recipe (!) because he thought that it would make a more interesting story. Some of the children were concerned if their story was unfinished; one child even went back to the programming cards later, after the activity was over, to append a set of final instructions, such as to turn off the oven used by the cook and to eat the sandwich (as shown in Figure 14).

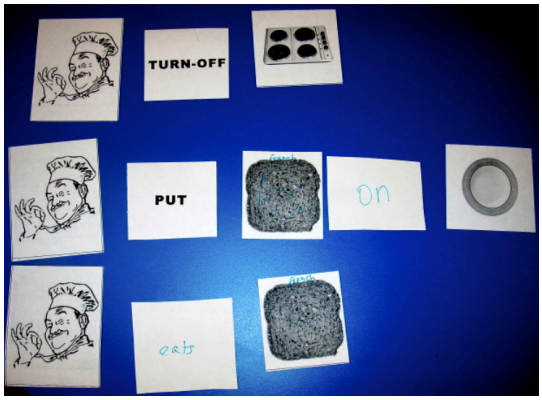

Figure 14. Code appended by a child member to the end of French Toast recipe for turning the oven off.

While the team was working with the programming cards, one of the children spontaneously began writing an English version of the "French Toast" program on a separate piece of paper (Figure 15). She indicated that she wanted to present the code in a more explicit storytelling style. Most of her verbs had an "-s" suffix (third person), which matches the previously indicated perspective of telling the cook's story. Storytelling Alice [13] has been similarly well received by children, who want to tell stories with computing, and in future Toque prototypes, we should adhere to their design criteria.

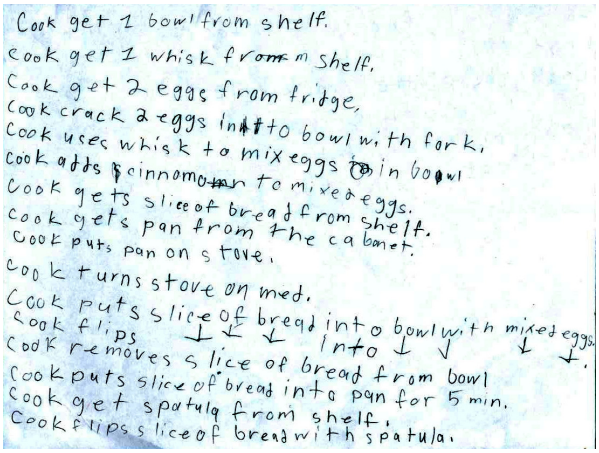
Cook get 1 bowl from shelf.
cook get 1 whisk from shelf.
Cook get 2 eggs from fridge.
Cook crack 2 eggs into bowl with fork.
Cook uses whisk to mix eggs in bowl
Cook adds cinnamon to mixed eggs.
Cook gets slice of bread from shelf.
Cook gets pan from the cabinet.
Cook puts pan on stove.
Cook turns stove on med.
Cook puts slice of bread into bowl with mixed eggs.
Cook flips
Cook removes slice of bread from bowl
Cook puts slice of bread into pan for 5 min.
Cook get spatula from shelf.
Cook flips slice of bread with spatula.

Figure 15. French Toast program in verbal English written by a young design team member.

We asked two of our child team members how they compared programming with Toque as opposed to sitting in their school classes. Both children replied that math was boring, but that cooking was fun! Later in the week we asked all of the children whether they had ever done any computer programming. With the exception of one child,

they said that they had not. We take this to mean that they did not consider their experience with Toque as programming. This could prove to be a useful perception, since from previous conversations they thought programming (whatever it was) was boring. It could be that because none of the children had prior programming experience, they did not generally recognize or identify their work in Toque as programming even though they wrote and/or implemented very detailed instructions throughout both design sessions.

LESSONS LEARNED FOR FUTURE DESIGN

In this section, we discuss some general implications of our results and relate them to existing work. This discussion suggests design guidelines for those who work on programming systems similar to Toque.

Pair Programming with Wiimote and Nunchuck

A pair of children sharing the Wiimote and Nunchuk provides a convenient way to support collaboration naturally. In this children's version of Pair Programming (PP), the Wiimote holder is the driver and the Nunchuck holder is the (recipe directions) navigator. In our design sessions when the children were exploring the programming tools, we found that the children naturally whispered among each other, they thought aloud, and decided together about the next instruction to execute, and corrected the partner's errors. Occasionally, children switched roles by changing the controllers (along with the paper handout). How the children naturally explored the programming environment confirmed the literature on the PP model and how effective it is for novice student programmers in introductory programming courses [17,28].

Complexity for Children: Iteration versus Concurrency

Although iteration is an essential construct in programming, our young designers had trouble with the design of loops. Pane et al. [22] found that children tend to describe iteration as operations on sets of objects. We found that this generalized to programming a virtual chef to cook.

Because the children in our team found loops "boring," we asked their opinions about what they would add to our system to make it more exciting. Their first suggestion was adding more cooks. They could quickly comprehend the differences and similarities between each cook preparing different dishes and all cooks collaborating on a single dish, although it was not implemented in the Toque prototype and was never mentioned throughout the sessions. These are means for concurrent programming, and so another possibility is that children prefer parallelism to iteration, which is the same finding that Pane found in his study.

Our young design team members preferred a third-person, storytelling perspective. However, researchers found fewer loops in Storytelling Alice programs than regular Alice programs [13]. Given our experience, we propose that the loop construct is not well matched to storytelling, but that iteration may still be possible by using a different approach such as blocks. Alternatively, in a storytelling setting it may make more sense to replace iteration with concurrency.

Flexible Ordering of Actions and Objects

While most programming languages impose a strict ordering for objects and actions, we found that in our design sessions the children used a variety of orders. Research by Pane et al. [22] confirms this; in their work, novices writing instructions to make a game would write instruction components in various orders. Some children assumed that an object knows how to behave (a typical OO design); an example is the instruction *the ghost will turn colors*. However, others used a more imperative ordering, e.g., *change the ghosts from original color to blue*.

With our design team, instead of expressing an instruction to whisk eggs, *cook mixes the eggs with whisk*, children felt that *cook uses the whisk to mix eggs* was more natural. Other ways of conveying the same idea include *cook makes the eggs by mixing with whisk*, *eggs get mixed by the cook using whisk*, or *whisk does mixing of the eggs for cook*. All these forms emphasize a different locus of behavioral control, but the meaning of all of them is the same to children. As discussed in our formal language specification [25], our attempts to formally define a precise but complete cooking domain highly constrained our language (including the action/object issues). Our child co-designers also did not seem to try to use it in a way outside the scope envisioned in our specification.

CONCLUSION

In this paper, we discussed our intergenerational design process, which supported the iterative development of a visual and physical programming experience for children. As a result, we came to understand the importance of pair programming with the Wiimote and Nunchuck, an order-independent view to function-argument relationships, and a storytelling approach to programming. In our sessions, while searching for tangible ways to create an engaging programming environment to teach computational thinking to young children, we found many similarities in the history of visual systems for the same purpose. Because the introduction of physical systems have been recent and have not been extensively explored, we invite future researchers working in the area of tangible systems to consider visual programming languages.

In future work, we will expand the capabilities of the Toque prototype to support multiple cooks (threads), recipes with branches, flexible orderings for objects and actions, and extended storytelling support. Empirically, we would like to compare "tangible solo programming" to "tangible pair programming" and the orderings of objects and actions that children actually use.

ACKNOWLEDGMENTS

We thank all of our child co-designers who partnered with us in our sessions. And we are grateful to Andy Ko and

Kori Inkpen for their untiring feedback and guidance in paper revisions.

REFERENCES

1. Bransford, J., Sherwood, R., Vye, N., and Rieser, J. Teaching Thinking and Problem Solving: Research Foundations. *American Psychologist 41*, 10 (1986), 1078-1089.
2. Cooper, S., Dann, W., and Pausch, R. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. *J. Comput. Small Coll. 15*, 5 (2000), 107-116.
3. Dalbey, J. and Linn, M.C. The Demands and Requirements of Computer Programming: A Literature Review. *Journal of Educational Commuting Research 1*, 3 (1985), 253-274.
4. Druin, A. The Role of Children in the Design of New Technology. *Behaviour and Information Technology 21*, 1 (2002), 1-25.
5. Nintendo. http://www.cookingmamacookoff.com/
6. Fernaeus, Y. and Tholander, J. Finding Design Qualities in a Tangible Programming Space. In *Proc. CHI 2006*, ACM Press (2006), 447-456.
7. Glu Cooking star. http://www.cookingstargame.com/.
8. Greenberg, S., Fitchett, C. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. In *Proc. UIST 2001*, ACM Press (2001), 209-218.
9. Horn, M.S. and Jacob, R.J.K. Tangible Programming in the Classroom with Tern. *Ext. Abstracts CHI 2007*, ACM Press (2007), 1965-1970.
10. Igarashi, T. and Hughes, J.F. Smooth Meshes for Sketch-Based Freeform Modeling. In *Proc. I3D 2003*, ACM Press (2003), 139-142.
11. Kahn, K. Drawings on Napkins, Video-Game Animation, and Other Ways to Program Computers. *Commun. ACM 39*, 8 (1996), 49-59.
12. Kelleher, C. and Pausch, R. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv. 37*, 2(2005), 83-137.
13. Kelleher, C., Pausch, R., and Kiesler, S. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. In *Proc. CHI 2007*, ACM Press (2007), 1455-1464.
14. MacLaurin, M. Kodu: End-User Programming and Design for Games. In *Proc. FDG 2009*, ACM Press (2009), xviii-xix.
15. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. Scratch: A Sneak Preview. In *Proc. C5 2004*, IEEE (2004), 104-109.
16. Martin, F., Mikhak, B., Resnick, M., Silverman, B., and Berg, R. To Mindstorms and Beyond: Evolution of a Construction Kit for Magical Machines. *Robots for kids: exploring new technologies for learning* (2000), 9-33.
17. McDowell, C., Werner, L., Bullock, H.E., and Fernald, J. The Impact of Pair Programming on Student Performance, Perception and Persistence. In *Proc. ICSE 2003*, IEEE (2003), 602–607.
18. McNerney, T.S. From Turtles to Tangible Programming Bricks: Explorations in Physical Language Design. *Personal Ubiquitous Comput. 8,* 5 (2004), 326-337.
19. Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., and D'Amour, A. Physical Programming: Designing Tools for Children to Create Physical Interactive Environments. In *Proc. CHI 2002*, ACM Press (2002), 299-306.
20. National Research Council. *Being Fluent with Information Technology*. National Academy Press, Washington, DC, 1999.
21. Pane, J.F. and Myers, B.A. The Impact of Human-Centered Features on the Usability of a Programming System for Children. *Ext. Abstracts CHI 2002*, ACM Press (2002), 684-685.
22. Pane, J.F., Myers, B.A., and Ratanamahatana, C.A. Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems. *Int. J. Hum.-Comput. Stud. 54*, 2 (2001), 237-264.
23. Pizza Palace, Webkinz. http://webkinztown.com/.
24. Swan, K. and Black, J.B. Results of Four studies on Logo Programming, Problem Solving, and Knowledge-Based Instructional Design. In *The International Conference on Technology and Education*, March 1990.
25. Tarkan, S. The Formal Specification of a Kitchen Environment. Master's Thesis, U. of Maryland, 2009.
26. Tarkan, S., Sazawal, V., Druin, A., Foss, E., Golub, E., Hatley, L., Khatri, T., Massey, S., Walsh, G., Torres, G. Designing a Novice Programming Environment with Children. Technical Report 2009-03, HCIL, Jan 2009.
27. WiiRemoteJ, http://www.wiili.com/wiiremotej-f68.html
28. Williams, L. and Upchurch, R.L. In Support of Student Pair-Programming. In *Proc. SIGCSE 2001*, ACM Press (2001), 327-331.
29. Wyeth, P. and Purchase, H.C. Tangible Programming Elements for Young Children. *Ext. Abstracts CHI 2002*, ACM Press (2002), 774-775.
30. Zuckerman, O., Arida, S., and Resnick, M. Extending Tangible Interfaces for Education: Digital Montessori-Inspired Manipulatives. In *Proc. CHI 2005*, ACM Press (2005), 859-868.