Designing a Novice Programming Environment with Children

Sureyya Tarkan, Vibha Sazawal, Allison Druin, Elizabeth Foss, Evan Golub, Leshell Hatley, Tejas Khatri, Sheri Massey, Greg Walsh, and Germana Torres
Human-Computer Interaction Laboratory (HCIL)
University of Maryland Institute for Advanced Computing Studies (UMIACS)
University of Maryland
College Park, Maryland 20742
sureyya@cs.umd.edu

ABSTRACT

When children learn how to program, they gain problem-solving skills useful to them all throughout life. How can we attract more children in K-8 to learn about programming and be excited about it? To answer this question, we worked with a group of children aged 7-12 as our design partners. By partnering with the children, we were able to discover approaches to the topic that might appeal to our target audience. Using the children's input from one design partnering session, we designed a prototype tangible programming experience based upon the theme of cooking. The children evaluated this prototype and gave us additional design ideas in a second session. We plan to use the children's design ideas to guide our future work.

1. INTRODUCTION

Programming allows children to explore creative topics and learn problem-solving skills. Popular novice programming tools include Scratch [6], Alice [1], and ToonTalk [5].

We want to reach children that may not be well-matched to popular novice programming tools. Of particular interest to us are children with kinaesthetic, auditory, and social learning styles. These children may not be well served by sitting alone silently in front of a computer for a long period of time.

One promising research area is *tangible programming* for novices. Tangible programming systems such as Tern [4], Electronic Blocks [8], and AlgoBlocks [7] reify programming constructs and objects as physical objects. Tangible programming systems allow children to easily work together and move around as they program. However, these systems make many compromises; for example, Tern is very full-featured but rather complex, and Electronic Blocks are simple but offer very limited programming capability.

To maximize the potential for tangible programming, we need a tangible programming system that is fun, complete, and age-appropriate in complexity. To accomplish this objective, we actively included children directly in the design process. Using a technique known as *design partnering*, adults and children worked together to uncover how tangible programming could be both appealing and educational.

2. DESIGN PARTNERING

Design partnering [2] refers to a design process in which children and adults are equal stakeholders. Children do not merely use, test, or inform the design; they actively help create it. As Druin notes, children "have special experiences and viewpoints that can support the technology design process ... With this role of design partner, the impact that technology has on children may not be as significant as the impact children can have on the technology design process."

Allison Druin created the Kidsteam project in 1998 at the University of Maryland as a vehicle for design partnering. In each design session, 4 or more children (aged 7-12) and 6 or more adults work together on a variety of design problems. In August 2008, the Kidsteam spent two 2-hour sessions on tangible programming.

Two adult members of Kidsteam jumpstarted the design process with an initial prototype of a tangible programming system. This prototype was not intended to be a solution, but rather an example of what is possible.

Because we seek to attract children with kinaesthetic, auditory, and social learning styles, we decided to build a basic collaborative programming environment that encouraged moving around and making sounds. Using styrofoam and embedded analog sensors [3], we created a set of tangible objects. Most of these objects (see Figure 1) were shaped like musical instruments and musical notes, and children could press them to programmatically create a song. A round object enabled iteration, and an eraser-shaped object allowed children to delete previously "written" code. An audio-based tutorial assisted users as they interacted with the tangible objects. The output of the tangible objects were both sounds and on-the-fly generated Java-like code for programmatically creating the corresponding sequence of sounds. Figure 2 shows an example of our generated code.

2.1 Kidsteam Session I

The adult kidsteam members first introduced the concept of computer programming by programming a human "robot." One of the adult members of Kidsteam acted as a robot and responded to a set of instructions – `turn`, `walk`,

Figure 1: Kidsteam Session II

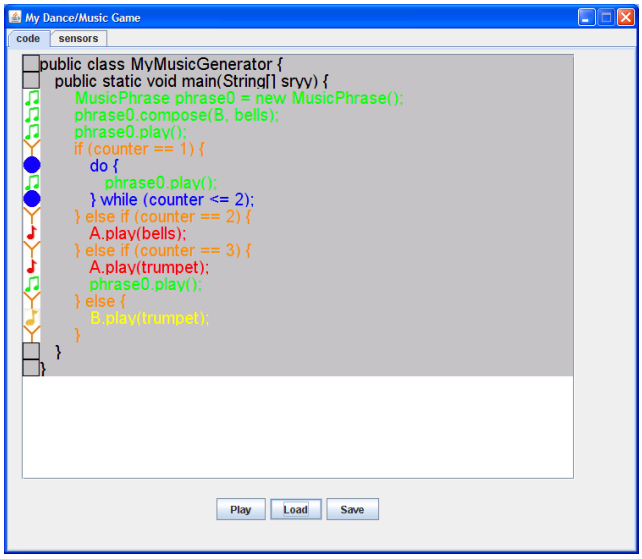

Figure 2: The GUI for `Music Game`.

`lift`, `push`, and `put` – written on a sheet of paper. The "robot" also understood the following keywords: `right`, `left`, `forward`, `a bit`, `number of steps`, `down`, `up`, `high`, and `low`. The Kidsteam then "programmed" the "robot" to move from one side of a room to the other side without running into obstacles. If an instruction was erroneous, the robot was stopped and reset to the start of the obstacle course.

After successfully completing this step, the Kidsteam mapped each keyword (`turn`, `right`, etc) to sounds such as *clap*, *beep*, *boop*, *snap*, and *buzz*. The children reimplemented the program using the sound-based instructions.

We then presented the prototype system to the children to help them understand the possibilities in scope for a tangible programming system. We asked the children to identify an idea of what they would like to program using tangible objects such as these.

The Kidsteam generated three concrete ideas: (i) a computer game, (ii) a robotic dog, and (iii) a robotic chef. To further elaborate these ideas, the entire group was separated into smaller groups of three to further flesh out their ideas.

Tangibly programming a computer game. This group considered the functionality available in current games, such as materials, players, options, rules, levels, characters, inputs, and outputs. They came up with a chocolate/strawberry finding game, a racing game, and an adventure game. However, because the computer game was rather an abstract idea, this group did not generate concrete ideas of how they could program a game.

Tangibly programming a robotic dog. The second group came up with commands to control a robot dog. Example tasks were `get (the paper)`, `chase`, `guard (the house)`, `dust`, and `talk (I love you)`. Furthermore, they specified the robotic dog's implementation to include holders to carry things around, audio speakers, a GPS device, a motherboard, face recognition of owners' pictures, and a dog house.

Tangibly programming a robotic chef. The third group wrote a program that a robotic chef could execute in order to prepare a dish. More specifically, the cook could `pull`, `get`, `put`, `pour`, `pull`, `turn`, `mix`, `close`, `stir`, and `wait for` (things). They also listed the kitchen utensils to be supported: (`spatula`, `pan`, `plate`, `bowl`, `oven`) and ingredients (`oil`, `milk`, `egg`). The robotic chef also understood `Repeat` and time. The designed robot had a backpack to hold things, a tray, a table, and an oven in its belly.

2.1.1 Results of Session I

After all of the sub-groups presented their ideas to the entire Kidsteam, the adults held a debriefing session to decide on the next course of action. We decided to implement the programmatic cooking idea, because it was the most concrete design and it appealed to both males and females in the group. Consequently, we added a simulation of a robotic chef in Alice [1] to our prototype system. The tangible objects (still musically themed) now corresponded to cooking instructions. A team member would "execute" the cooking instructions on the chef simulation in a wizard-of-oz style by manually updating the chef simulation according to which tangible objects were pressed.

Figure 3 shows our robotic chef simulation. In addition to "executing" the tangibly programmed instructions, the simulation also provided error messages for syntax and intent errors.

Figure 3: The simulation of `Cooking Game`.

2.2 Kidsteam Session II

During our second session approximately one week later, we asked our young design partners to create (simulated) hot chocolate (similar to the recipe in Table 1) using the tangible objects.

```
Heat(syrup)
Heat(milk)

for (int i = 0; i < 4; i++) {
    Pour(syrup)
    Pour(milk)
    Mix
}
```

Table 1: Hot chocolate program. The keywords are in bold; directions and ingredients are represented starting with upper and lowercase letters, respectively.

In our new system, the musical instruments were mapped to `milk` and `chocolate syrup`. Musical note objects were mapped to `mix`ing, `pour`ing, and `heat`ing. Moreover, we attached written definitions to each tangible object for guidance. Two children recorded sounds for mixing ("clink" of a spoon in a cup), pouring (sound of water flowing), and heating (sound of sizzling). To represent the ingredients they simply said the names of those ingredients, i.e. "chocolate syrup" and "milk". These sounds were mapped to the tangible notes so that pressing them generated the corresponding sound.

The children wrote a program by interacting with the tangible objects without any help from the adults. After a few mistaken tries, the children successfully guided the simulated chef to make four cups of hot chocolate. However, they didn't recognize that their program could be simplified with a loop. The adult members of Kidsteam asked the children to consider what the program should look like if there were more mugs to fill, and then children and adults together designed a new program on paper. The children then "wrote" the new loop-oriented program using the tangible objects. The children shared the tangible objects amongst each other.

2.2.1 Results of Session II

After this experience, the children then provided feedback on the existing design and generated additional design ideas. Children put their design ideas, what they liked, and what they disliked on separate sticky notes. These sticky notes were then organized into groups with the help of adult Kidsteam members.

The children responded positively to the tangible objects used to program and the simulation that executed their commands. They also enjoyed working together to make hot chocolate! However, there were many features of the prototype that they wanted to improve. The system had too many wires, some buttons were difficult to press, and the simulated graphics could be improved.

The children's design ideas included many concrete ways to improve the robotic chef simulation. They also emphasized flexibility: they wanted the freedom to assign tangible objects to utensils at will, assign sounds of their own choosing, and also choose their own recipes to implement. The children also proposed that iteration be implemented with two tangible objects: one for START (the set of instructions to be iterated) and one for STOP.[1]

After this session, the adult members of Kidsteam met to elaborate on session outcomes. During the second session, children worked together to build a working computer program. Cooking was an interesting activity even to boys in the group. Sound generation, however, was not particularly interesting to the children. We, therefore, decided to pursue a cooking-based novice programming system using tangible objects that are more appropriate for cooking.

3. CONCLUSIONS

Our goal is a tangible programming space that allows children to collaboratively learn computer programming. We included children in the design process for a tangible programming system. The children in our design team steered us towards cooking as a fun activity amenable to algorithmic thinking. In future work, we plan to implement a fuller-featured tangible programming environment based on the design ideas obtained and assessed with our child design partners.

4. ACKNOWLEDGMENTS

We thank all the children who were our design partners.

5. REFERENCES

[1] M. J. Conway. *Alice: Easy-to-Learn 3D Scripting for Novices*. PhD thesis, University of Virginia, 1997.

[2] A. Druin. The role of children in the design of new technology. *Behaviour and Information Technology*, 21(1):1–25, 2002.

[3] S. Greenberg and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218, New York, NY, USA, 2001. ACM.

[4] M. S. Horn and R. J. K. Jacob. Tangible programming in the classroom with tern. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 1965–1970, New York, NY, USA, 2007. ACM.

[5] K. Kahn. A Computer Game to Teach Programming. In *Proceedings of the National Educational Computing Conference*, pages 127–135, 1999.

[6] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick. Scratch: A Sneak Preview. In *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, 2004.*, pages 104–109, 2004.

[7] H. Suzuki and H. Kato. Interaction-level support for collaborative learning: Algoblock—an open programming language. In *CSCL '95: The first international conference on Computer support for collaborative learning*, pages 349–355, Hillsdale, NJ, USA, 1995. L. Erlbaum Associates Inc.

[8] P. Wyeth and G. Wyeth. Electronic Blocks: Tangible Programming Elements for Preschoolers. In *Proceedings of the Eighth IFIP TC13 Conference on Human-Computer Interaction*, pages 496–503, 2001.

[1] Not unlike loops in Ada!