The Tinker Environment: a New Paradigm for Novice Programming Using the Spatial Computing Framework

Pongsakorn Thiamwang[1], Arnan Sipitakiat[1], Paulo Blikstein[2]
[1]Department of Computer Engineering, Chiang Mai University
239 Huaykaew Rd, Muang, Chiang Mai, Thailand, 50200
[2]Graduate School of education, Stanford University
520 Galvez Mall, CERAS 232, Stanford, CA, USA, 94305
p.thiamwang@gmail.com, arnans@eng.cmu.ac.th, paulob@stanford.edu

ABSTRACT
This paper describes and provides examples of Spatial Computing, a method of using on-screen actions to perform programming and mathematical operations. The design goal is to create new ways for children to encounter and tinker with powerful ideas in programming and robotics. Two examples are depicted. First, a 'variable condition' object allows one and two conditionals to be visualized and manipulated. Secondly, a number-transform object converts data from one unit to another using on-screen graphical lines that can be directly manipulated by the learner. Initial case studies in robotics programming are discussed and the new learning opportunities are highlighted.

Categories and Subject Descriptors
H.5.2 [Information Interfaces and Presentation]: User Interfaces. K.3.1 [Computers and Education]: Computer Uses in Education.

General Terms
Design, Human Factors, Experimentation.

Keywords
Interaction Design, Spatial Computing Interface, Physical Computing.

1. INTRODUCTION
The increase in availability and popularity of visual programming environments such as Scratch [1], Google Blockly [2] and Snap! [3] has shifted expectations about what a programming environment for children should look like. While this change is positive, it does not mean we are exploring all he potential of the visual medium. For example, blocks environments, in essence, replace typing with drag-and-drop actions. Although coding becomes simpler, the resulting program structure remains very much the same as that written in text. Flow chart environments use different shapes and graphical lines to define programs, but these charts mainly imitate what is traditionally drawn on paper.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

IDC '14, Jun 17-20 2014, Aarhus, Denmark

ACM 978-1-4503-1918-8/13/06.

This work presents the Spatial Computing framework which attempts to use space in more productive ways as a programming tool. We aim to demonstrate interfaces that are conceptually different than text coding and also are not possible outside of the computational medium.

2. THE DIGITAL MEDIUM AS A NOVEL MEANS FOR HUMAN COMPREHENSION
Digital technology has led to new ways for people to observe phenomena, contemplate ideas, and understand. In a learning context, Papert and collaboprators demonstrated how children can relate powerful mathematical ideas to their body movement through "turtle geometry", an important concept within the Logo programming environment [4]. The Berkeley Boxer Project [5], as an evolution of Logo-like environments, was an early example of how the screen's spatial properties can increase the concreteness of abstract constructs such as variables, procedures, and object properties. Scratch, Google Blockly, and Snap! visualize the program's syntax and structure by replacing text with graphical blocks and, thus, allowing the program primitives to offer labels that make the overall program better resemble human language. Agent-based modeling environments such as NetLogo [6] and AgentCubes [7] allow children to contemplate ideas in decentralized systems by programming rules to a large number of concrete objects (agents) that then carry out those rules all at the same time. Toontalk [8] uses child play actions to construct rules allowing children to relate play and computer programming.

This work builds upon previous research in Spatial Computing Paradigm (SCP) [9] first introduced in the pyoLogo environment. SCP makes use of the screen space not only to visualize information but also to perform computation. Two examples are described: conditions and data transformation.

3. THE TINKER FRAMEWORK
Tinker is a new programming environment built on top of Google Blockly. Tinker integrates two Spatial Computing Modules into the blocks programming approach.

3.1 Variable Conditionals
3.1.1 Single Variable Conditionals

Figure 1. An example of a single variable object

upper and lower limits of the line defaults to the maximum and minimum sensor value pre-defined by the system (for example, 0 to 1023).

The value on the line object is divided into segments. The sensor variable has two default segments labeled TRUE and FALSE and are divided at the mid-point (in this case, 512). The number of segments, segment labels, and boundary values can be configured by the user through an interactive interface in the properties window. During execution, the sensor value is mapped onto the line object and the segment into which the value falls is considered activated and will be highlighted on the screen.

The line object generates a single output with a value equal to the activated segment label. This output can be later used as either a variable in a linear program (as described in section 3.3) or an input to a (nested) Spatial Data Interface.

3.1.2 Two Variable Conditionals

A multiple variable condition is a fundamental programming operation and the ability to formulate and synthesize such a condition significantly contributes to the level of programming fluency. However, learning about multi-variable conditionals in a traditional programming language can be difficult and often results in logical errors in the program that are difficult to comprehend and debug [10].

Tinker offers a 2D spatial representation of two variable conditions. The learner can associate the two axes of the 2D object to variables involved in the conditional. Similarly to the line object in a single variable situation, each axis is divided into

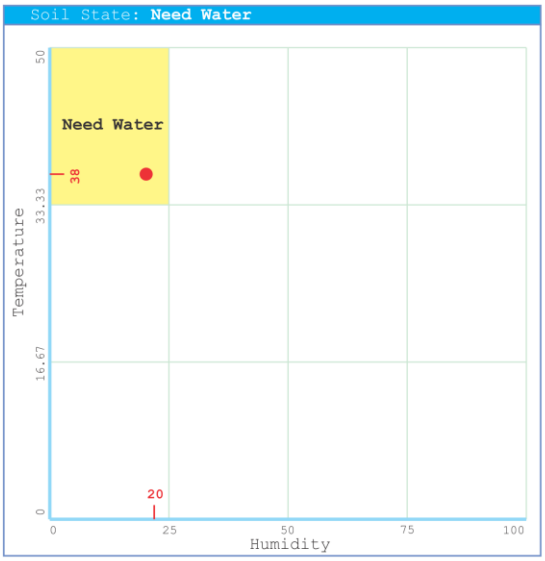

Figure 2. An example of a 2D object offering a spatial representation of a two variables condition

user-defined segments. This creates a grid, which classifies all the possible output combinations. The learner can name significant cells in this grid.

As an example, figure 2 shows a 2D object representing temperature on the vertical axis and humidity level on the horizontal axis. Assuming that the system needs to activate the watering system when the humidity is low and the temperature is high, the upper left cell where the temperature is above ~33 degrees Celsius and the humidity is less than 25% is labeled as "need water". During runtime, this 2D object will return the string "need water" when both sensor conditions are met, which can then be used to activate the watering system.

3.1.3 Date and Time Conditions

Date and time are special cases of variable conditions. Although date and time are straightforward to think about, creating date and time conditions is usually cumbersome and difficult to express in traditional text programming. Tinker offers a single and a two variable date and time objects.

A *time* variable object consists of a line object that is, by default, divided into 48 segments, each representing 30 minutes. The current time can be plotted onto the line object. The learner can label segments to reflect the time event that he or she is interested in detecting.

A date-time two-variable object adds the date to the vertical axis, allowing the learner to define different times for each date. There are a set of different date types to choose. A day-of-month variable will generate 31 segments one for each calendar day. A day-of-week variable will generate 7 segments representing each day in a week. This date-time object can, for example, create a classroom timetable used to define the time periods that classroom lights should be turned on or off.

3.2 Number Transformations

When working with sensors, data transformation is a common task. Many sensors gives readings that do not correspond to the standardized units such as degrees or lumens. Temperature, brightness, sound loudness, for example, are represented by a raw

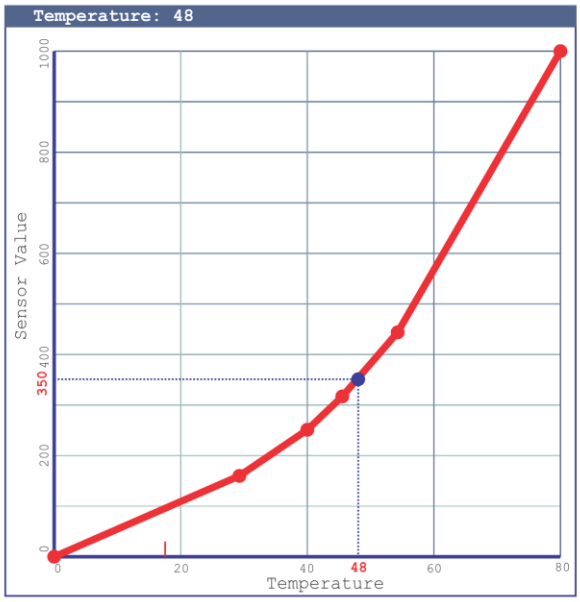

Figure 3. A number transform object converting raw sensor readings into temperature

number that is determined by the internal circuitry used to measure it. Converting these arbitrary scales to real units would either require the learner to derive a formula from the known data points or use data interpolation. Both methods are powerful mathematical concepts but are not easily learned and manipulated

through conventional programming languages. Tinker offers a spatial tool to deal with and learn about these conversions.

A *transform* object is shown in Figure 3. The vertical axis represents the input value while the horizontal axis is the output. A transform "line" visually defines how the input is transformed. A default diagonal transform line performs nothing (output=input). The learner can interactively manipulate this line to change the transform behavior. Figure 3 is an example of a transform object that converts raw sensor values into degrees Celsius. "Nodes" can be added to the transform line which, in this case, represents each known conversion point. By building this transform line the learner is essentially constructing an interpolation process in a visual way.

3.3 Integration with Blockly

Tinker employs Google Blockly for the learner to layout the overall program sequence. The spatial computing objects create new primitives or blocks for use in the program.

3.3.1 Blocks for Single and Two Variable Conditions

Figure 4 illustrates blocks for variable conditions generated by Tinker's spatial tool. In essence, all the functionalities of the spatial tool is translated into the traditional sequential representation and packaged as a callable function. This allows the learner to investigate the resulting code if he or she is interested.

3.3.2 Blocks for Number Transformations

Similar to the variable conditions, the graphical representation of a number transform is translated into command blocks and

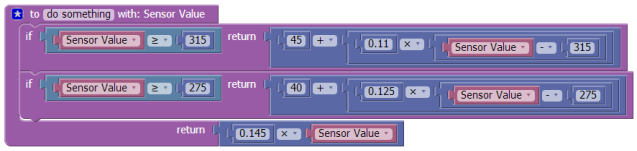

Figure 5. Command blocks generated from the data transform object

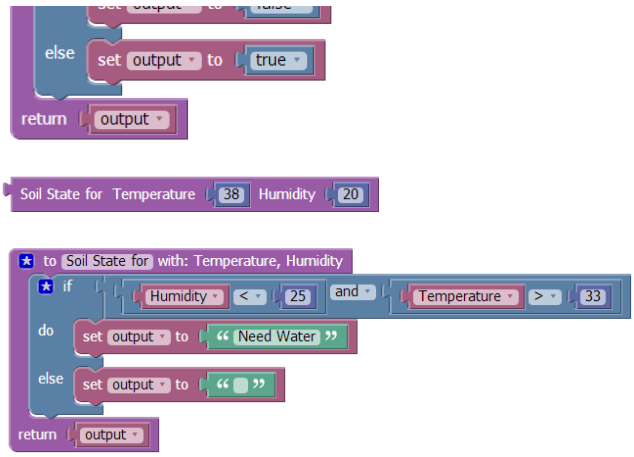

Figure 4. Command blocks generated by the variable condition object

wrapped inside a callable function as shown in Figure 5.

4. CASE STUDIES AND DISCUSSION

The following sections describe our initial experiments with scenarios that could benefit from the Spatial Computing framework.

4.1 Understanding and Avoiding Pitfalls in Multi-Variable Conditionals

When the number of variables involved in a conditional increases significantly the number of possible states, manipulating these states can become overwhelming. Describing each of them often requires a level of specificity that novices may not know how to describe.

Consider the following example. A student wants to use two on-off buttons to control a motor that follows these conditions. (A) Turn the motor clockwise when right button is pressed. (B) Turn the motor counter-clockwise when left button is pressed. (C) Stop the motor when neither button is pressed. (D) Stop the motor and trigger and alarm when both buttons are pressed at the same time. The textual explanation above makes sense when communicated verbally. But in conditions (A) and (B) we are leaving out details about the state of the unmentioned button. We are implicitly saying, for example, (A) the motor should turn clockwise if the right button is pressed "*and the left button is not pressed*". Since a computer does not have these assumptions, the learner often unintentionally creates a bug. During execution, the learner will notice that when both buttons are pressed, rules (A), (B) and (C) are all triggered. Tinker's spatial representation can be useful in this case.

Figure 6 shows a two-variable object that defines all the four states of this task. The spatial representation implicitly assists the learner in defining all the variables involved.

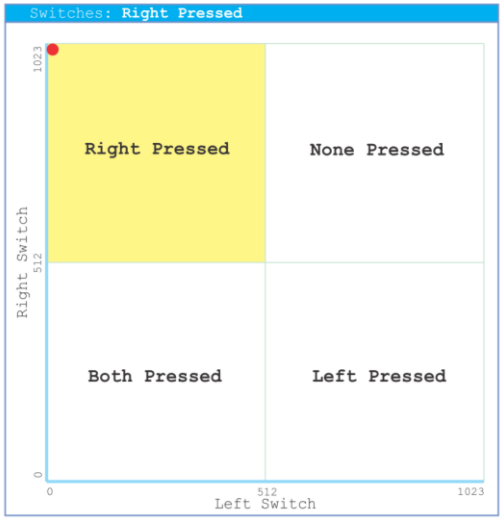

Figure 6. Defining states of two switches spatially is a simple way for learners to fully express the conditions.

4.2 Mathematical Operations Using Data Transform

Although the Data Transform object was first designed for interpolation purposes, we have later found that it provides a number of new mathematical affordances.

Let us say that a student wants to control a robotic pet by making it move at a speed proportional to the noise level: the louder one would cheer, the faster the pet will move. The robot uses a microphone which gives an output between 400 (silent) and 900 (very loud). However, the motor speed various only between 0 and 8. Normally one would need to perform the following calculation in order to map the sensor value to the desired output:

Motor Speed = (Sound-Sensor-value – 400) / ((900-400)/8)

Although the ability to articulate the above formula is a valuable mathematical skill, this abstract representation may not be the best stepping stone for a novice learner. In Tinker, this same operation can be done using the Data Transform object as shown in Figure 7. Any input value (the vertical axis) less than 400 outputs zero (the horizontal axis). This is the "Sound-Sensor-value – 400" term in the equation. Input values between 400 and 900 are scaled to output values between 0 and 8. The learner can observe the conversion in real-time and directly manipulate and experiment with the transform line.

We believe the benefit of this spatial approach is that learners can relate better to the meaning of the mathematical operation. The ability to play with the transform line and see the visual input-to-output relationship put our visual understanding of the world into play with the learning process. The benefits and limitations of this approach are topics we wish to continue to research.

5. CONCLUSIONS

This paper has described Tinker, an early prototype of a spatial computing environment that aims to offer new pathways for a learner to encounter and tackle programming and mathematical concepts using his or her existing understanding about space, shapes, and motion. Tinker also tries to create a continuum from its spatial representations to the more abstract forms commonly found in sequential programming paradigms. All the spatial operations are transformed into command blocks that can be later manipulated. We believe that, as we learn more from real-world interactions between students and our environment, there will be new types of spatial operations that can be added into Tinker. We hope to contribute to research in computationally-rich learning systems by showing how spatial representations--which computers do very well—are an important way to offer new affordances in interaction design for children.

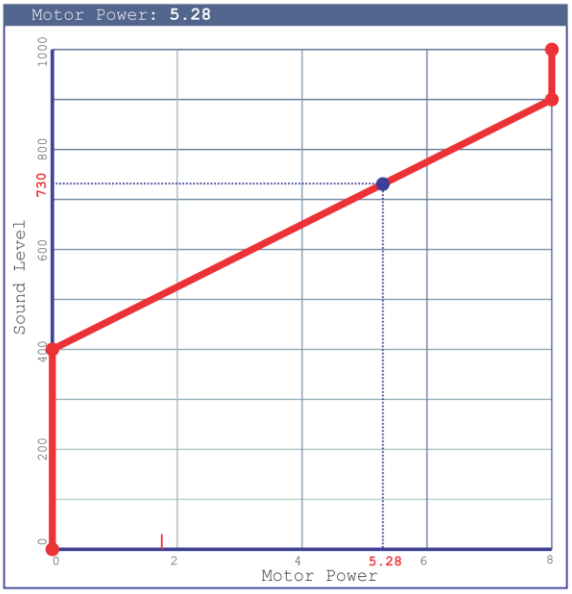

Figure 7. A transform line for converting sound sensor data into motor power levels.

6. ACKNOWLEDGMENTS

Our thanks to the research team at the Learning Inventions Laboratory, Chiang Mai University and the Transformative Learning Technologies Laboratory, Stanford University.

7. REFERENCES

[1] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silverman, B. and Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.

[2] Google Blockly. A Visual Programming Editor. https://code.google.com/p/blockly/

[3] Harvey, B. (2012). The Beauty and Joy of Computing: Computer Science for Everyone. *In Proceedings of Constructionism 2012*. Athens, Greece. 33-39.

[4] Papert, S. (1980/1993). *Mindstorms: Children, computers, and powerful ideas (1st and 2nd ed.)*. Cambridge, MA: Basic Books.

[5] DiSessa, A. A. (2001). *Changing minds: Computers, learning, and literacy*. Mit Press.

[6] Tisue, S., & Wilensky, U. (2004, May). Netlogo: A simple environment for modeling complexity. *In International Conference on Complex Systems* (pp. 16-21).

[7] Repenning, A.; Ioannidou, A., "AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D," *Visual Languages and Human-Centric Computing*, 2006. VL/HCC 2006. IEEE Symposium on , vol., no., pp.27,34. DOI= 10.1109/VLHCC.2006.7

[8] Kahn, K. (1996). ToonTalk—An Animated Programming Environment for Children. *Journal of Visual Languages & Computing*, 7(2), 197-217.

[9] Sipitakiat, A.,Cavallo, D. (2008). Giving the head a hand: constructing a microworld to build relationships with ideas in balance control. *In Proceedings of the 8th international conference on International conference for the learning sciences* - Volume 2 (ICLS'08), Vol. 2. International Society of the Learning Sciences 335-342.

Blikstein, P., & Sipitakiat, A. (2011, June). QWERTY and the art of designing microcontrollers for children. *In Proceedings of the 10th International Conference on Interaction Design and Children* (pp. 234-237). ACM.