ABSTRACT

Title of Document: PIXEL: A TOOL FOR CREATIVE DESIGN WITH PHYSICAL MATERIALS AND COMPUTATION

Michael Gubbels, Master of Science, 2015

Directed By: Dr. Jon Froehlich, Assistant Professor, Department of Computer Science, College of Information Studies

Creating information systems that sense and respond to the physical environment is a complex activity, requiring technical skills from disparate areas of practice, such as computer programming and electronic circuitry. Although recent tools have lowered barriers to creating such systems, they tend to be too technical and constraining for creating systems to be a feasible everyday activity. These tools often rely on traditional interaction techniques and draw makers' attention away from the system being built, thereby limiting makers' physical movement, removing systems from their use context, and preventing contextualized experimentation with system designs. This thesis explores techniques for designing tools with support for making systems a more feasible everyday activity. I present the novel design and evaluation of such a tool called Pixel designed to let makers use intuitive knowledge derived from experience with the physical world, rather than technical expertise, in creating custom information systems in the course of everyday life.

PIXEL: A TOOL FOR CREATIVE DESIGN WITH PHYSICAL MATERIALS AND COMPUTATION

By

Michael Gubbels

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2015

Advisory Committee:
Professor Jon Froehlich, Chair
Professor Allison Druin
Professor Hasan Elahi

Dedication

To the memory of Peter F. Leonovitz, Sr., a kindred spirit from a bygone generation who supported my early interest in computer programming.

Acknowledgments

I am grateful for working on this thesis with my advisor, Jon Froehlich. Working with him was delightful, but more often, it was challenging. While writing this thesis, Jon's advice proved to be incisive, usually critical, and ultimately enabled me become a clearer thinker and articulate ideas more effectively. I am sincerely grateful to Allison Druin and Hasan Elahi for their willingness to be on my committee, along with Jon. It's clear that their candidness and criticisms greatly enhanced the quality of this thesis.

I feel fortunate to have worked with Michael Smith-Welch. We worked on a range of projects, shared many conversations, and formulated many ideas together that were profoundly influential on this thesis and me as a person. I'm deeply grateful for Michael's mentorship and proud of our friendship.

I thank everyone that supported this work at the American Visionary Art Museum, FutureMakers, the Exploratorium's The Tinkering Studio, and KID Museum.

Finally, I'm deeply grateful for my friends and family who supported me while working this thesis, even when I was deeply withdrawn into the space oddity of academic writing. I give special thanks to Alanna for the joy she to my life with her music and our miscellaneous adventures, to Simon for sharing many wonderful and energizing conversations, to Josh and Greg for giving me a "just in time" home, and everyone at deadhau5 for not kicking me out of the house.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Physicality plays a central role in shaping everyday human experience. The way we engage with the physical world affects the way we perceive, transform, and derive meaning from it (Clark, 1998, 2010; Dourish, 1997, 2001; Klemmer, Hartmann, & Takayama, 2006). Increasingly, we use tools built with information systems, often as applications of computing and communication systems, to enable new kinds of engagement with the environment, influencing our experience. Despite the integral role and enabling potential of such systems in daily life, few can create them, and creating *custom* systems for everyday use is generally infeasible.

This thesis explores techniques for designing tools with support for making information systems in the course of everyday life. I present the novel design and evaluation of such a tool called Pixel with an aim to enable the use of intuitive knowledge derived from experience with the physical world, as an alternative to technical expertise, in creating custom systems in the course of everyday life.

Everyday Making

Everyday making is a composition of *"the processes we [carry] out in our daily life with different kinds of materials, tools and in interaction with other people. Everyday practices are those ordinary matters people do while busy with their everyday activities—while talking, writing, moving, doing manual work, interpreting and feeling"* (Tuomi-Gröhn, 2008, p. 8). As information systems influence more of our experience, designing tools that enable the everyday making of systems is not only potentially empowering but may become necessary (Holloway & Julien, 2010).

Making Information Systems

Presently, making information systems requires a familiarity with technical concepts, conventions, and an ability to express—or model—scenarios in an abstract formal language far removed from everyday experience. Even the most common tools for creating systems are technical. For example, Arduino, a platform popular among artists, hobbyists, and learners for prototyping interactive objects, relies on expertise in electronics and programming (Mellis, Igoe, Banzi, & Cuartielles, 2007).

Tools such as Arduino have significantly broadened participation in electronics and programming (*e.g.,* Kafai et al., 2010; Peppler, Glosson, Kafai, Fields, & Searle, 2011), but they tend to be limited in ways that make them infeasible for everyday making. They rely on the use of a keyboard, mouse, and display screen, drawing programmers' attention to editing a textual description of the system, and away from the system itself. This limits the mobility of programmers, separates them from their environmental context, and prevents direct experimentation with system designs in the environment.

Taken as a medium, information systems lack an appropriate representation and a corresponding set of creative tools appropriate for making systems in the course of everyday life. Researchers are exploring graphical, tangible, and material alternatives techniques that can engage more of our physical senses.

Graphical programing environments (GPEs) tend to emphasize the creation of interactive multimedia (Franklin et al., 2013) and many focus on simplifying programming to make it accessible to children (Kay, 2005; Resnick et al., 2009). Yet, GPEs have shown promise to support young people in making interactive systems

with sensors and actuators (Booth & Stumpf, 2013) and to enable product designers to quickly prototype interactive devices (Björn Hartmann, Klemmer, Bernstein, & Mehta, 2005). At the same time, GPEs can constrain making systems. Montemayor found that children using a GPE to define relationships between sensors and actuators might overlook the GPE entirely or become confused as they shift their attention between physical and graphical environments (2003). Until recently, GPEs relied on the use of a screen, keyboard, and mouse, fixing programmers' attention on the screen and restrict their movements to the space near it (*e.g.,* Millner & Baafi, 2011). Recent research mitigates these limitations by enabling touchscreen interactions on mobile platforms (*e.g.,* Chan, Pondicherry, & Blikstein, 2013) and shifting web environments (*e.g.,* Sadler, Durfee, Shluzas, & Blikstein, 2015). However, graphical and physical environments remain separate.

To an extent, tangible user interfaces (TUIs) can reconcile physical structure and information systems. This has made it possible to program systems by physically manipulating physical objects that represent programming language constructs (Suzuki & Kato, 1995). However, tangible formulations of logical constructs and programming language syntax provide limited physical manifestations of the counterparts in traditional programming paradigms. TUIs that explore more intuitive mappings between form and function offer more promising direction for designing everyday making tools. Researchers have developed kits for constructing robots and defining their behavior by directly manipulating parts of the built robots to demonstrate movements (Raffle, Parkes, & Ishii, 2004). Others have explored the use of gesture for creating custom gestural interfaces with everyday objects (Chung,

2010) and defining relationships between sensors and actuators embedded in objects to support immersive storytelling (Montemayor, 2003).

To make constructing systems easier, researchers are exploring uses of common materials for rapidly prototyping interactive devices (B. Hartmann et al., 2006) and making custom controllers for traditional computer software (Silver, Rosenbaum, & Shaw, 2012). Others are exploring the design of electronics that function more like familiar materials (Bdeir, 2009), microcontrollers as materials (Mellis, Jacoby, Buechley, Perner-Wilson, & Qi, 2013), and considering qualities such as texture as they relate to materials that are both physical and computational (Robles & Wiberg, 2010).

In my development of Pixel, I build on the research above to situate users' attention on the physicality of space and materials in their environment, by making computational processes accessible through physical gestures and touch interactions (with Pixel), and by eliminating the need to tether to a traditional computer to access a programming environment.

Pixel

Pixel is a tool for creating interactive information systems that sense and respond to the physical environment. It consists of one or more cube-shaped Pixels that together form a tangible user interface (TUI), to which electronic sensors and actuators can be connected with "snap connectors" on each Pixel, and a graphical programming environment (GPE) for mobile touchscreen devices (Figure 1).

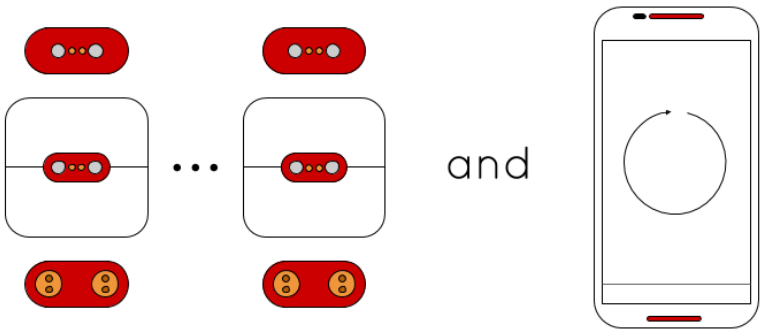


Figure 1: Conceptual drawing of Pixel, consisting of one or more Pixels (left) and, optionally, a mobile device to program Pixel behavior with the graphical programming environment.

Generally, Pixel was designed to be modular so it can be easily moved within and distributed throughout environments, combined with materials, and embedded in objects. Pixel's modular design is similar to that of Montemayor's *physical programming* tools in that it enables systems with sensors and actuators to be distributed in environments (2003). However, the systems differ in that Pixel enables components to be quickly connected to and disconnected from modules (*i.e.*, Pixels). On the other hand, Montemayor's system embeds sensors and actuators into its modules, called "physical icons," along with component-specific circuitry and code.

Systems made with Pixel can interact with the physical environment though *sensors*—devices for detecting physical phenomena—and *actuators*—devices for altering the physical environment. Pixel provides three interfaces for defining relationships between sensors and actuators.

First, users can perform *spatial gestures* by holding one or two Pixels and moving them according to certain patterns. The four gestures that each module can recognize are rest, swung, shaken, and tapped to another module (Figure 2).

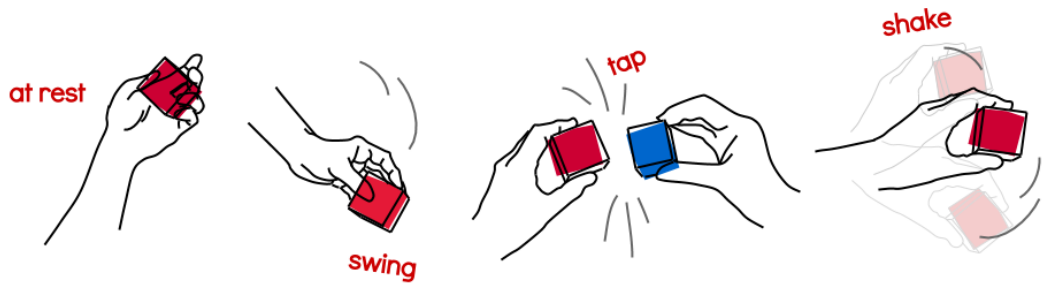


Figure 2: The set of gesture that are recognized by Pixel.

Spatial gestures are used solely for defining relationships between Pixels' connected sensors and actuators. Notably, Pixel doesn't require the use of an external tool to perform gestures, in contrast to both Montemayor's system, which requires a "magic wand" and "wizard hat" (2003), and Chung's OnObject, which requires wearing a ring device with an embedded RFID tag reader to perform gestures with objects (2010).

Pixel's second interface, the *snap connection interface* was inspired by the MaKey MaKey (Silver et al., 2012). The MaKey MaKey, designed for creating custom controllers for computer software (only), allows electronic components to be easily connected with alligator clips and emphasizes the use of construction materials (*e.g.*, cardboard, tape, glue, etc.). Pixel's snap interface offers comparable functionality, but extends it to support controlling actuators, too. Pixels have two snap connection ports and support one sensor and one actuator. Components can be connected to ports using the removable "snaps," to which they can be connected with alligator clips (Figure 3).

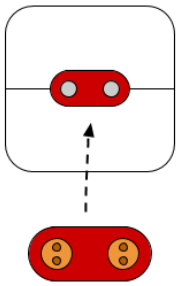


Figure 3: Drawing of Pixel's "snap connection interface." The component on the Pixel is the "port" to which the removable "snap" couples. The port's two gray circles represent magnets that hold the snap onto the Pixel. The snap has magnets on the opposite side of that shown. The two circular copper features of the snap are designed specifically for connection with alligator clips.

To assist users with connecting components, magnets are embedded into snaps and ports. The magnets are oriented so connectors can only be snapped onto Pixels one way.

Pixel diverges significantly from Montemayor's physical programming tools in its use of a graphical programming environment. Although GPEs can distract from physical engagement, they seem suitable for expressing information with which no single gesture intuitively corresponds. The GPE supports customizing behaviors of individual Pixels, connected sensors, and actuators. To maintain the freedom afforded by the modular design, the GPE was designed for mobile phones and tablets and can be used through touch-based gestures that "directly" manipulate graphical representations of a Pixel's actions.

To show how one can use Pixel to make everyday systems, three example scenarios are given in the following section.

Examples of Use

The three scenarios below show how to build three simple systems with Pixel. The scenarios also show how to build the same systems using Arduino. Showing both

of these highlights the advantages of using gesture and touch interactions (with Pixel) over traditional technical techniques.

Example 1: Making a Light Switch

This scenario shows how to build a simple light switch with both the Arduino and Pixel. This scenario is based on the common *Blink* example that serves as an introduction to Arduino. In contrast to *Blink*, this scenario incorporates a physical switch to control a light rather than automatically switching the light after a delay.

Part 1: Making a Light Switch with Arduino

Imagine that you're building a simple light switch device with Arduino. To do this with Arduino, you must identify the electronic input and output components required for the circuit, assemble them in a specific pattern, and program the Arduino to control the operation of components. In this situation, you use an input switch, output LED, four wires, a USB cable, and computer with the Arduino integrated development environment (IDE) installed, as depicted below.

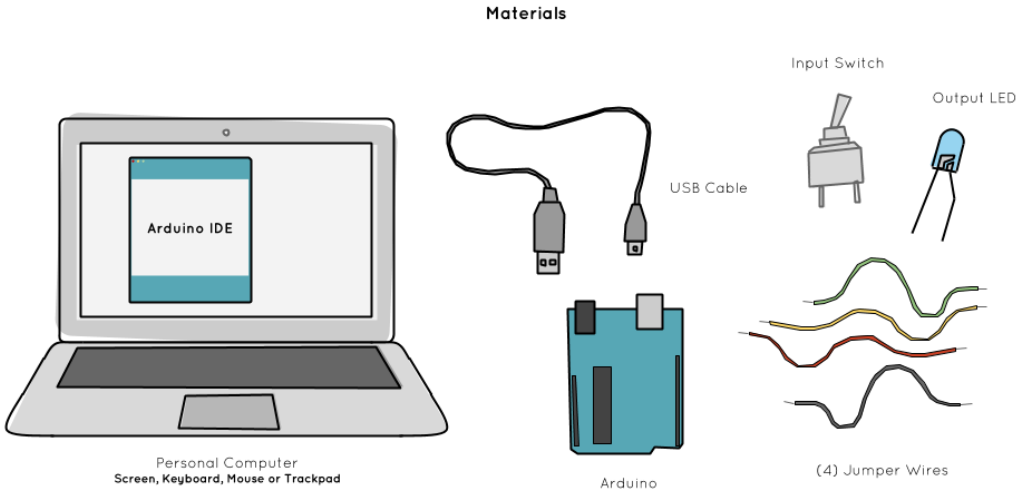


Figure 4: Illustration of the materials needed to make the light switch. Note that no current limiting resistor is used in this scenario, for simplicity.

You'd like to switch to turn the LED on and off. The electronic components need to be assembled in a pattern that will perform the desired light switching behavior. You connect the switch circuit as shown below.

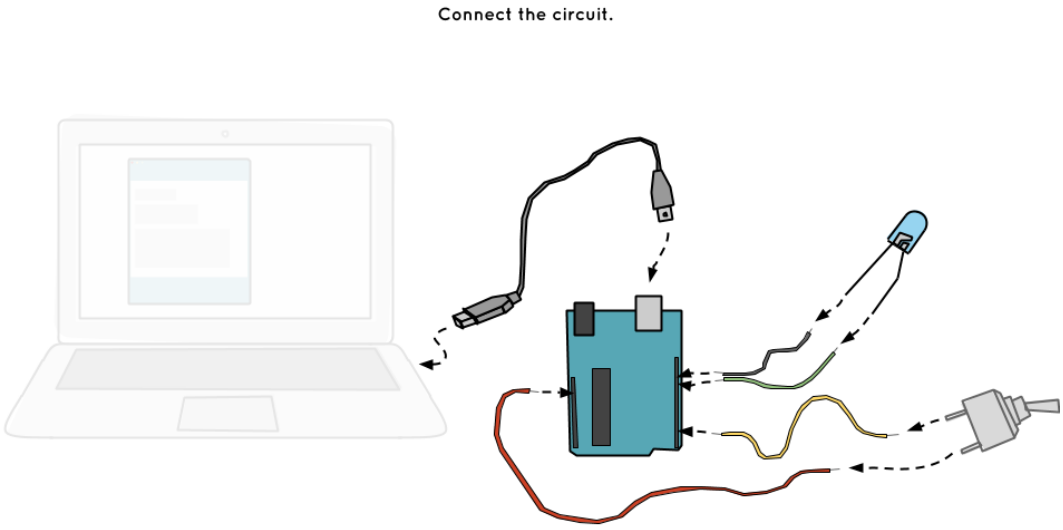


Figure 5: Drawing of connecting the components for the light switch.

Understanding even simple circuits like this one requires one to have some technical knowledge of electronics. Assembling the circuit with Arduino requires several steps, and each involves technical concepts, questions, and decisions. Components must be connected in way that matches their polarity. Some components, including LEDs, are unipolar, so they can be connected only one way. Some components need to be powered (or connected to power, as does the switch in this example). What is the power source? Arduino has no battery. It draws power over USB from the connected computer. Which components need power? How much (*i.e.,* 5V or 3.3V)? Which common voltage should be used (*i.e.,* GND)? How are the components actually connected to power and how does power relate to polarity? These are some basic concerns of working with Arduino.

Once assembled, the circuit needs to be programmed to control how the switch and LED interact. The program should deliver power the LED when the switch is in an “on” position and to cut the power when the switch is “off.” You type the program into the Arduino integrated development environment (IDE) with a programming language resembling the C programming language.

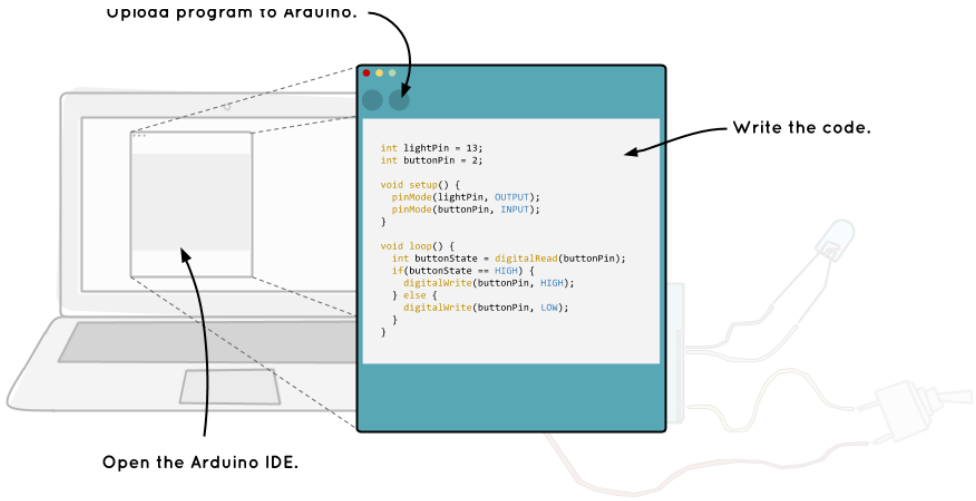


Figure 6: Drawing of programming the Arduino to control the light switch circuit. The complete program is shown in Figure 50 (Appendix A).

Now that the circuit is connected and programmed, the light switch device can be used. You toggle the switch from “off” to “on” as shown below.

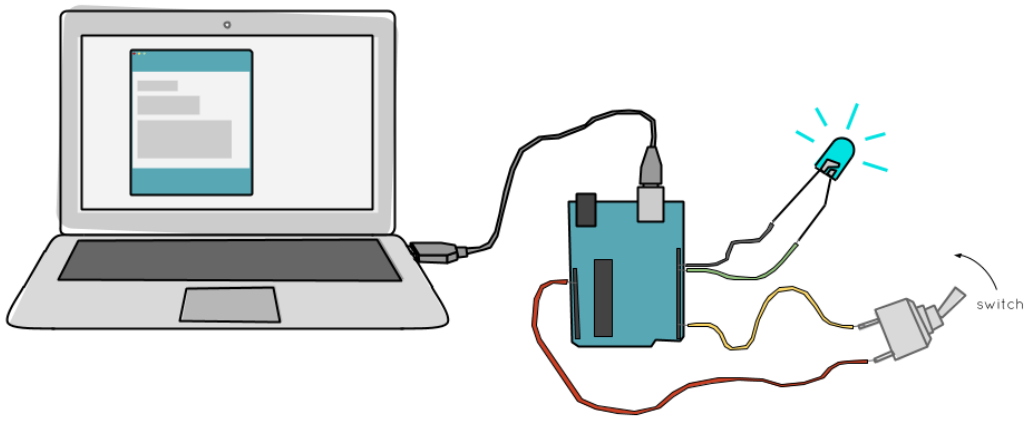


Figure 7: Drawing of the complete light switch.

Creating devices in this way with Arduino is technically challenging and physically constraining. To be appropriate for day-to-day applications, creating a simple switch device should be very easy and take no more than minute or two. The next section shows how Pixel simplifies creating switches, notably eliminating the need for a computer (and external power source) and a textual programming language.

Part 2: Making a Light Switch with Pixel

The materials required to build a light switch with Pixel are shown in Figure 9. No computer is needed to program the switch circuit.

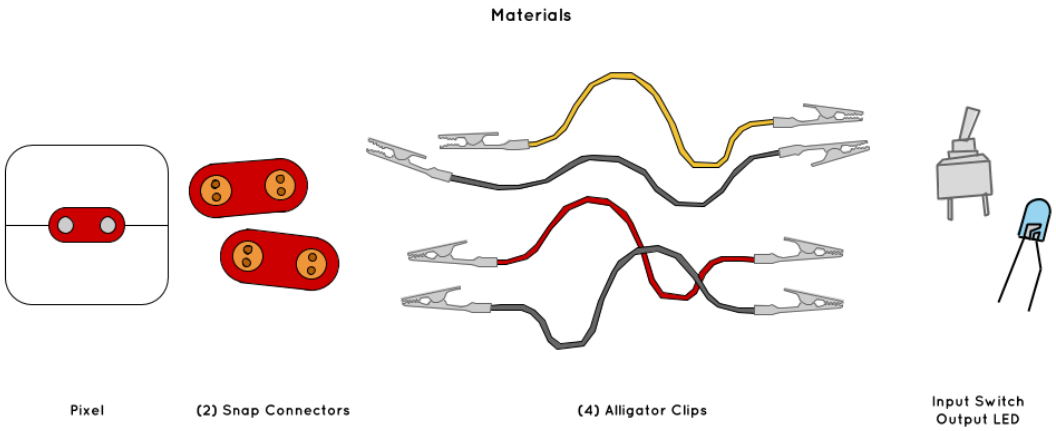


Figure 8: Drawing of the materials needed to make the light switch.

As with Arduino, the electronic components need to be assembled in a pattern that will result in expected behavior. A component can be connected Pixel by clipping it to a snap connector with two alligator clips, then snapping it to an input or output port. You connect the switch and LED in this way, as shown below.

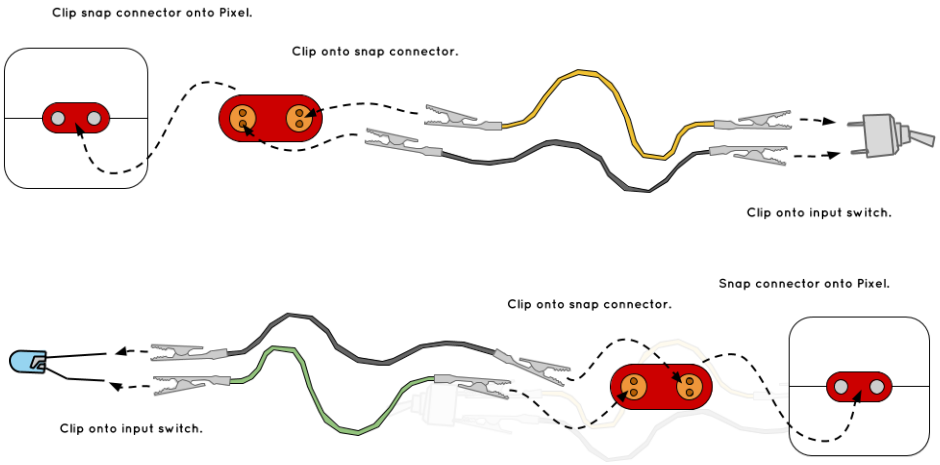


Figure 9: Drawing of connecting the input switch (top) and output LED (bottom) for the light switch. The complete system is shown in Figure 10.

By default, each pixel functions as a switch. That is, when its input port is activate—if a connected circuit is closed—its output port will actively power the connected component. As a result, creating the light switch can be done entirely through direct physical action. As soon as the switch and LED are snapped onto a pixel, you can use the light switch.

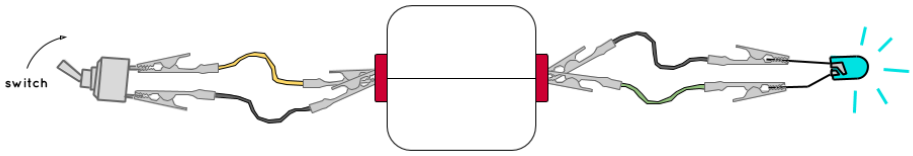


Figure 10: Drawing of the complete light switch.

In this case, no programming was required to make the light switch. Pixel was designed to provide "switch" as default a behavior because it is a design pattern that can to control a variety of electronic components. There is no need to program the I/O relationship because each pixel has only a one input and one output—their default relationship is set automatically. The "switch" behavior was chosen as the default for

each pixel because it provides immediate utility and is applicable in a range of situations. Example 2 shows the use of this primitive "switch" behavior (analogous to a control structure) in making a remote light switch.

Example 2: Making a Remote Light Switch

This scenario extends the previous one so the light can be turned on and off *remotely*. Again, I show necessary steps to build the system with Arduino before showing it is built with Pixel.

Part 1: Making a Remote Light Switch with Arduino

The additional materials required to make the switch are shown below.

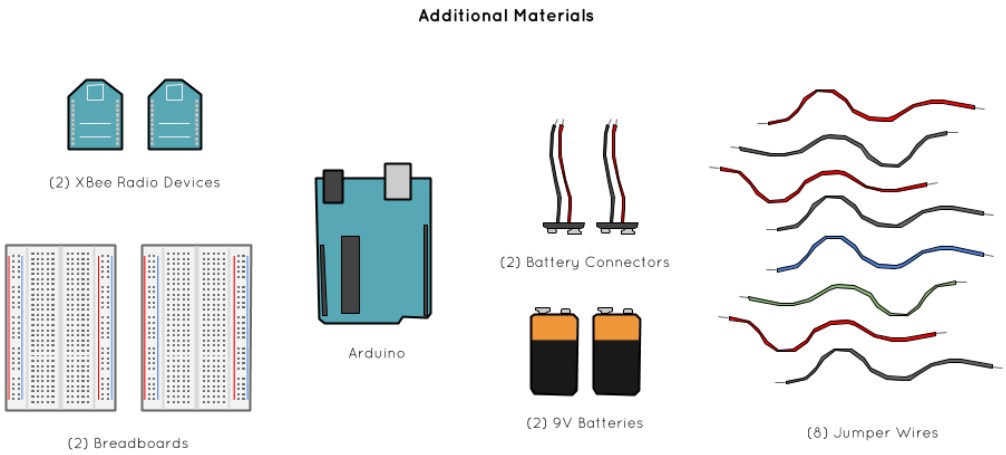


Figure 11: Drawing of the materials needed in addition to those used for the light switch in the previous scenario.

To make the Arduino light switch from the previous example controllable remotely, the LED can be integrated into a second circuit, and the original light switch circuit can be changed to function as a remote control for the new light circuit. You make these adaptations as shown below.

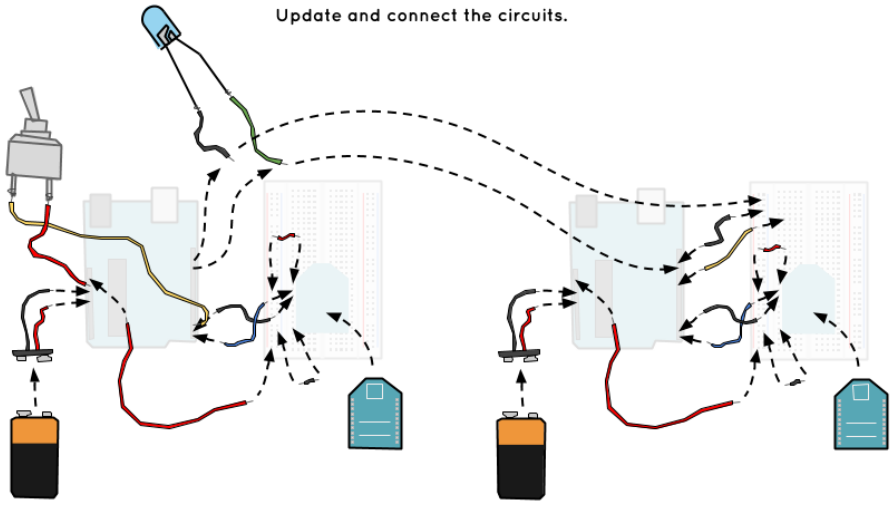


Figure 12: Drawing of the circuit connections and component placement on the breadboard for the remote light switch. Each dotted-line arrow represents a connection (and disconnection, for the LED) that must be made to update the circuits.

The programs controlling these circuits must be updated to handle communications between the devices. The switch circuit monitors the connected switch and sends a command to the light circuit accordingly. When the switch is "on," the circuit sends a command to the light circuit. You make the changes shown in the figure below.

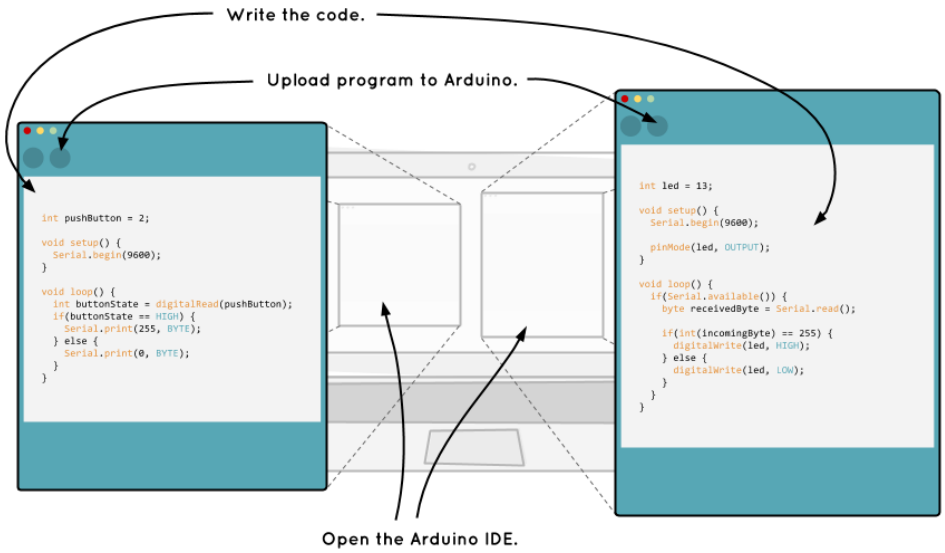


Figure 13: Drawing of the steps involved in programming the Arduino. The two separate Arduino sketches are shown to represent the two separate circuits in the remote light switch. The programming of both circuits needs to be updated. The complete sketches are shown in Figures 51 and 52 (Appendix A).

After you update the programs and upload them to the Arduinos, you can use the remote light switch.

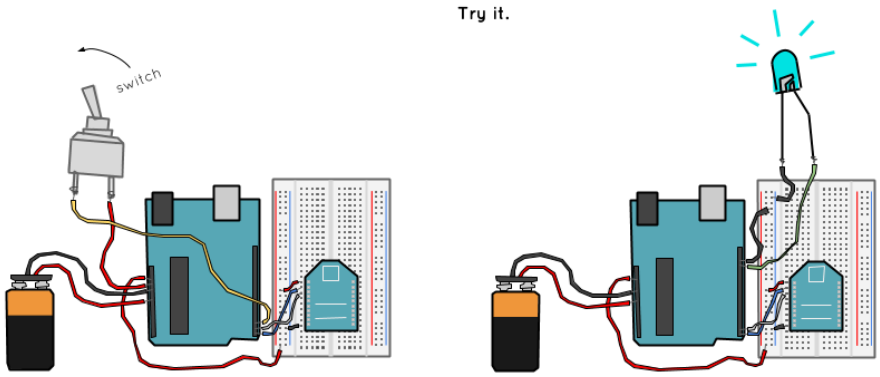


Figure 14: Drawing of the complete remote light switch.

This remote light switch is considerably more complex than the one from the previous example. The circuitry and programming in this example require much more technical knowledge than Example 1, even though the resulting behavior is quite similar. Even this very simple use of the wireless communication device requires a communication protocol to be designed and implemented for both the switch and light circuits. In this example, the switch circuit sends the value “255” to the light circuit to indicate that it should should turn “on” its LED. The next section shows that making a remote light switch with Pixel is significantly simpler.

Part 2: Making a Remote Light Switch with Pixel

Adapting the light switch made with Pixel to be a remote switch requires only one additional pixel. The adaptation can be done in three steps with only physical actions (Figure 16). First, you swing the additional pixel in a downward motion (top left). Next, you tap the pixel just swung to the other pixel in the light switch circuit

(top right). Finally, you unsnap the switch from the light switch pixel and snap it onto the other module that you swung (bottom).

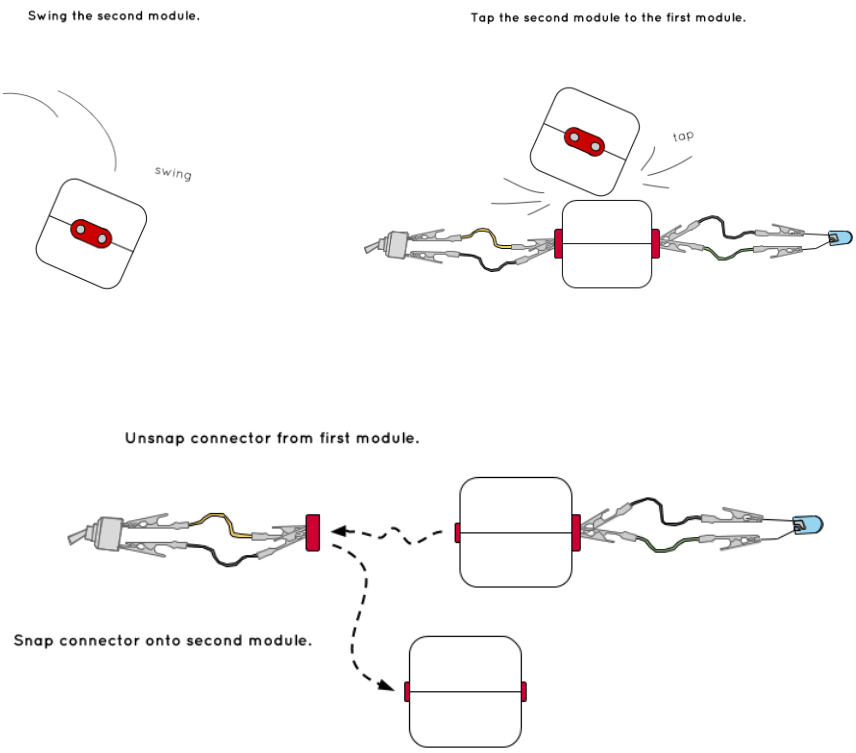


Figure 15: Drawing of the sequence of actions to adapt the light switch to be a remote light switch.

The complete remote switch is shown in the figure below.

Figure 16: Drawing of the complete remote light switch.

The process of adapting the light switch to be a remote switch with Pixel is relatively simple and involves little in addition to moving the switch to a second module. The additional processes are minimal—swinging a module and tapping it to

the other—and the actions to carry them out bear some intuitive relationship to their effect. Swinging a module engages it, indicating that it is the subject of attention, and tapping a module to another indicates that it is the subject of consideration in relation to another module.

Example 3: Making a "Scarecrow Tree"

This scenario is based on a suggestion of a participant in a Pixel evaluation (discussed in Chapter 5). This scenario was chosen to illustrate a realistic everyday situation in which an information system could be usefully and uniquely applied. The participant characterized the problem as follows.

"Here's a problem I have. I have a cherry tree. Just when the cherries get real ripe, the birds come and eat them. Now, if I can make some sounds or some kind of flashes or something—a scarecrow, right?—then the birds will not come. Now even a scarecrow that you have, it has to have moving parts on it, or, they say, you can buy a plastic owl and put it somewhere, but if the plastic owl is not moving at all then it won't work. The birds will learn and it's useless. So, if one had these kinds of things, and one of them has a motion detector, gets a motion from the birds around or something, then it can signal the other ones which would be on several branches in the tree. Something like that."

This is illustrated in **Figure 17**.

Figure 17: Drawing of the "scarecrow tree" problem as described by a participant in an evaluation of Pixel. I discuss my interaction with the participant in Chapter 5.

Below, potential solutions are illustrated for Arduino and Pixel. These solutions extend the remote light switches presented in the previous scenarios. Note that while participant P1 envisioned this scenario for Pixel, the solution for Arduino is shown for consistency with previous Examples.

Part 1: Making a Scarecrow Tree with Arduino

This scenario describes how to change the remote light switch circuits for use in making a scarecrow tree. The additional materials needed are a single piezo speaker, a resistor, and a jumper wire (**Figure 18**).

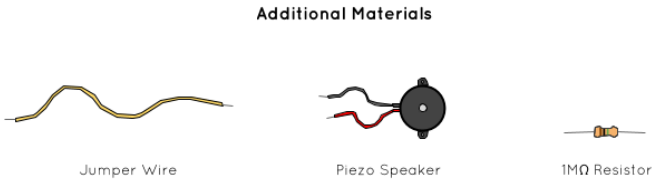


Figure 18: Drawing of the additional materials needed to adapt the remote light switch to play sound.

Extending the circuit is relatively simple. To do it, the three components must be connected to the “Output” circuit. Likewise, the program needs to be updated to control the speaker.

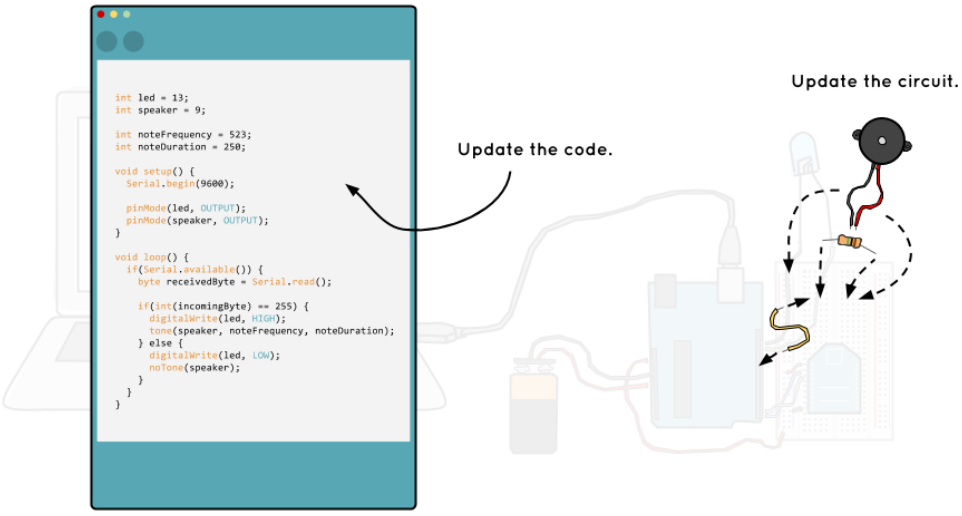


Figure 19: Drawing of connecting the speaker into the circuit and programming the circuit.

At this point, only one output device has been built for installation into the tree. To match the design adopted for Pixel, three more need to be made. Each of the additional devices requires the materials shown below.

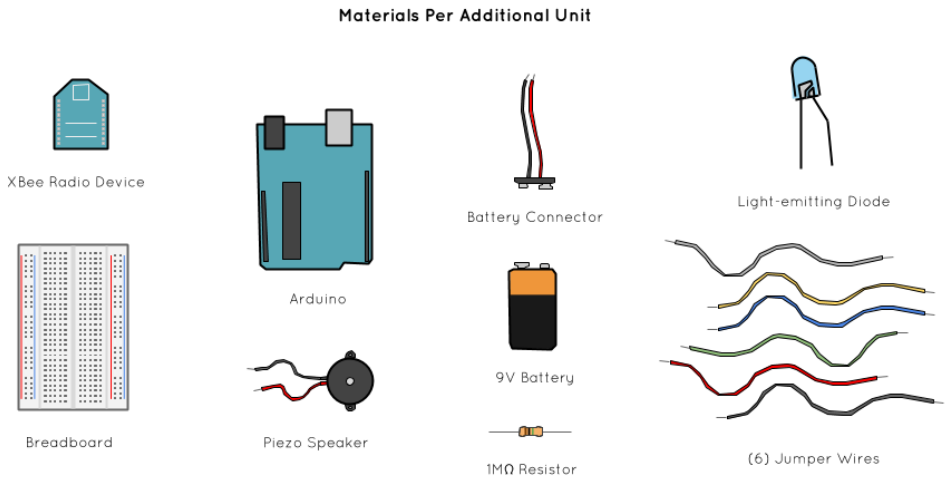


Figure 20: Illustration of the materials needed to create each additional “Output” device.

Using these materials, the circuit needs to be assembled and programmed. This is depicted in the figure below.

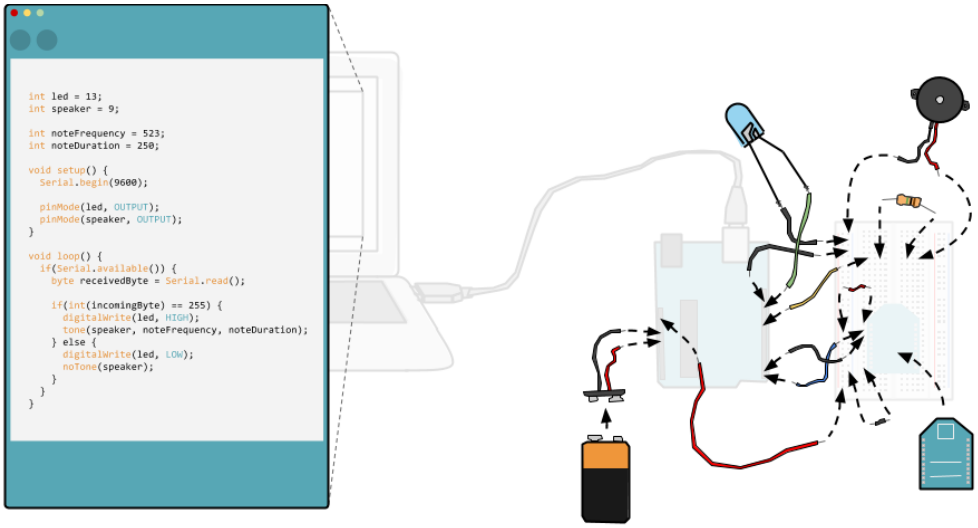


Figure 21: Drawing of connecting an additional "Output" device. This is done for each of the three additional devices. The complete program is shown in Figure 53 (Appendix A).

Once all four circuits are connected, they can function together to create the "scare" effect to embed in the tree. This is illustrated below.

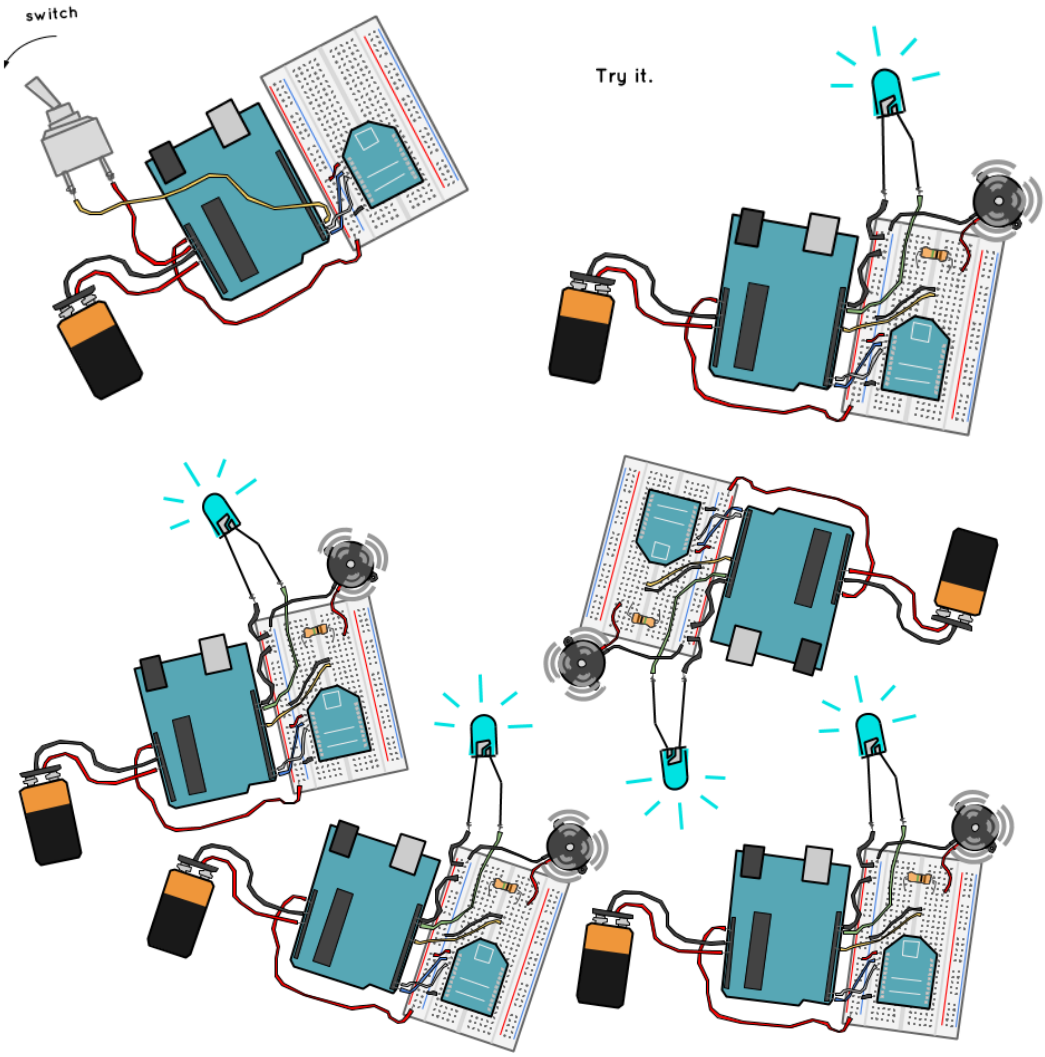


Figure 22: Drawing of connecting an additional "Output" device. This is done for each of the three additional devices.

At this point, the devices are functionally complete, but they still need to be packaged into an enclosure that can be installed in a tree. One possibility is using a plastic enclosure. In that case, the plastic would need to be translucent or modified to make the light visible and the sound audible.

Part 2: Making a Scarecrow Tree with Pixel

Making the scarecrow tree with Pixel can be done with five Pixels, one snap connector, an input switch, two alligator clips, and a mobile phone with Pixel's graphical programming environment (**Figure 23**).

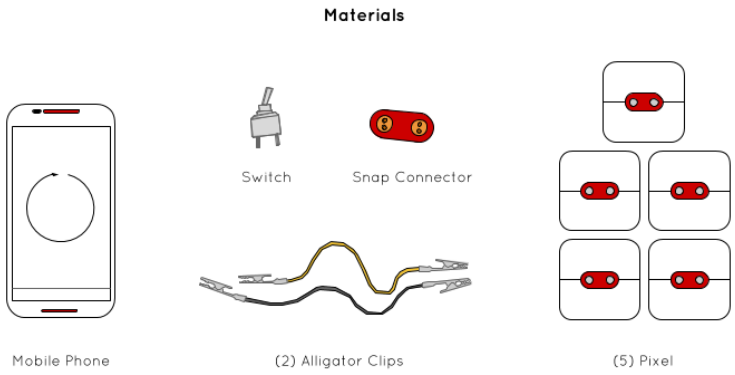


Figure 23: The materials used to make the scarecrow tree with Pixel.

One Pixel must be chosen to function as the "remote" to cause light and sound to be generated by the other four Pixels. Because each Pixel contains an LED and speaker, separate light and sound actuators are unnecessary. An input switch will be connected to this module. In turn it will cause the remote Pixels to flash light on and off and play a sequence of high pitch tones.

To start making the system, imagine that you swing one of the Pixels. This will function as the "remote" Pixel. Swinging the module *engages* it, or directs it to become available for further gestural interaction.

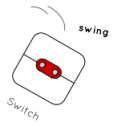


Figure 24: The *swing* gesture.

Recall that gesturing with Pixel is for defining stimulus-response interactions between Pixels. Next, you tap the "switch" Pixel to one of the other Pixels, represented as "Output A" in the figure below.

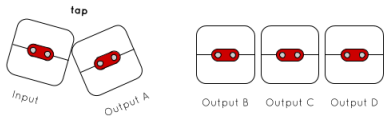


Figure 25: The *tap* gesture between two Pixels.

This defines stimulus-response relationship between the "Switch" and "Output A" Pixels in which the stimulating the "Switch" activates "Output A," which responds by emitting flashes of light and generating a succession of sonic tones. To make the "Switch" activate *all* other Pixels (not just "Output A"), repeat the swing-and-tap gesture sequence, swinging the "Switch" as before, but tapping it to "Output B," "Output C," and "Output D" in turn. You do this as shown below.

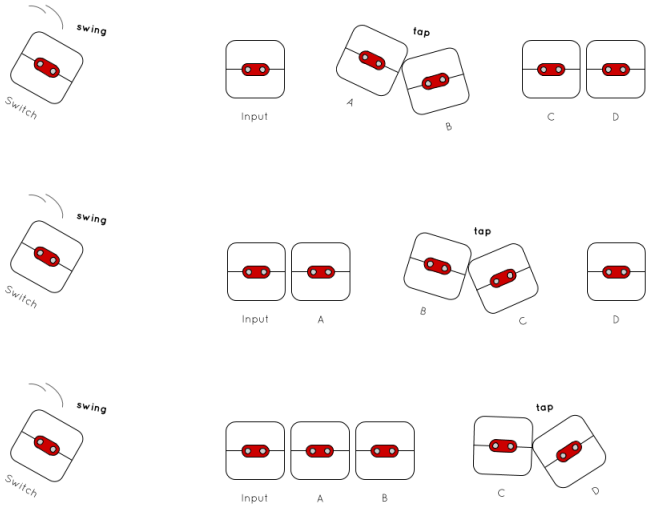


Figure 26: From top to bottom, this drawing shows the gestures needed to create define the relationships between the "Switch" and "Output" Pixels.

At this point, stimulating the “Switch” activates all of the “Output” Pixels synchronously. To stimulate the “Switch” easily, snap a toggle switch to “Switch” (with a snap connector), in the same way as was done in the prior scenarios. You do this as shown below.

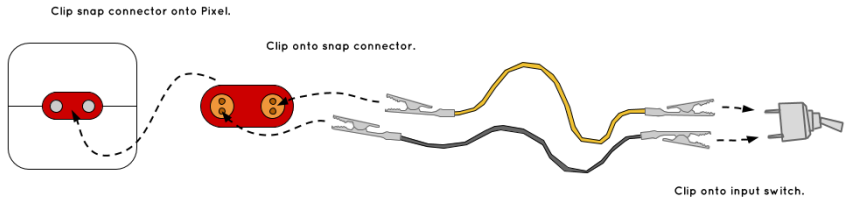


Figure 27: Drawing of connecting the input switch to the “Switch” Pixel.

The above sequence of swing-and-tap gestures and connecting the toggle switch to the “Switch” Pixel concludes the gestural interactions needed to make the scarecrow tree. However, as the Pixels are performing their default behavior, stimulating the “Switch” activates the “Output” actuators, to which no components are connected. The “Outputs” should, instead, display a sequence of flashing light and play a series of tones with their embedded LEDs and speakers. This can be done in the graphical programming environment.

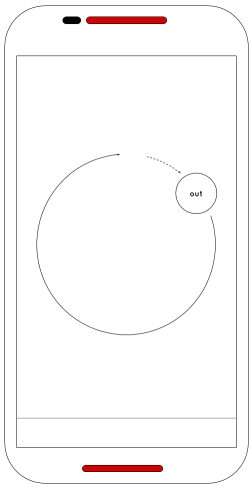


Figure 28: Drawing of the default state of the graphical programming environment.

When you open the GPE, it appears as shown in Figure 24. It shows a representation of the Pixel's behavior as a sequence of actions, represented as circles, ordered clockwise around a circular "loop." Pixels perform the action sequence repeatedly. The actions may be conditional, as represented by the loop segment before an action. In Figure 24, the dotted segment represents the "activation" condition, satisfied when the Pixel is activated remotely.

To make the "Output" Pixels produce light and sound to scare birds away from the tree, the corresponding actions must be added to their loops, and those actions must be set to fire only when the "Switch" activates the corresponding "Output." This can be done through a series of single-finger surface gestures with the graphical environment.

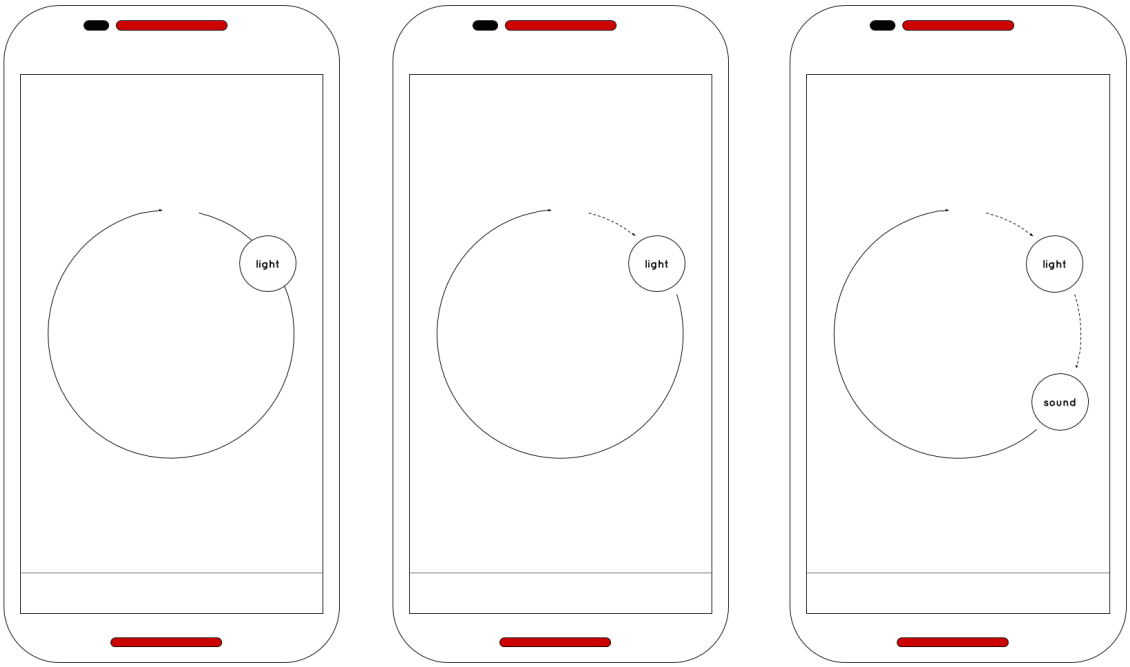


Figure 29: Drawing of the graphical state of the GPE after adding the light behavior, setting its activation condition, adding the sound behavior, and setting it's activation condition.

To start, you remove the existing action by touching and holding your finger to it, then dragging it away from the loop, and finally lifting your finger from the action. Only the actions on the loop will be performed. Actions that are not on a loop can be added to the same loop again or left off-loop to fade away (removing them from memory).

Now you create the new action sequence on each Pixel. First, you touch and hold anywhere off the loop (for about one second) until the possible actions are presented, then lift your finger. Second, you touch and drag the "light" action onto the loop (shown on the left). By default, actions are unconditional, so the Pixel will emit white light. White is the default light color. Third, to make the light action conditional, you tap (touch and then immediately lift) the loop segment prior to the action. This changes the condition that must be met to perform the action (shown second from left). Next, you repeat these three steps for the "sound" action (the result is shown third from the left), adding it to the loop.

These steps must be repeated on the remaining three "Outputs." Swiping left and right across the GPE changes between loops for different Pixels. To complete the scarecrow tree, you swipe left three times, each time repeating the steps done above. Once completed, the Pixels can be installed into the cherry tree. The complete scarecrow tree is depicted in **Figure 30**.

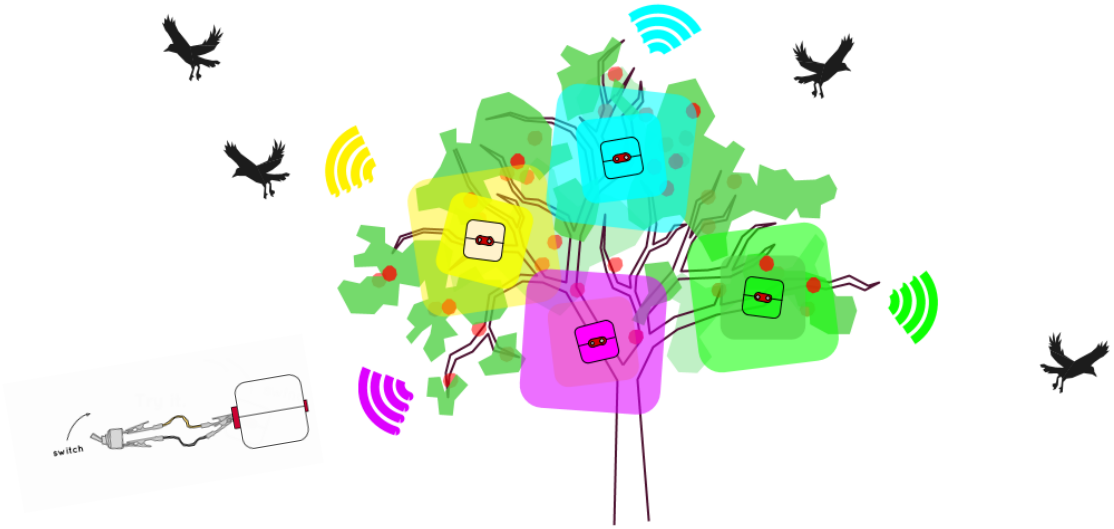


Figure 30: Drawing of the complete scarecrow tree built as built with Pixel.

Design Motivation

This thesis was motivated by the recognition that our experiences are increasingly mediated by and intertwined with technology, but, paradoxically, the ability to create, shape, and control these experiences is accessible only to those with a technical understanding of systems. This became apparent over the course of a number of personal experiences working with young people in museums and workshops, and from participating in hackathons. My exploration of alternatives to the technical tools for creating systems heavily guided by principles of constructionist learning theory, the approach of critical making (Ratto, 2011), physical computing (O'Sullivan & Igoe, 2004), as well as the "do it yourself" spirit of hacker and maker subcultures (*e.g.,* Hatch, 2013).

Design Approach

I designed Pixel "in the wild" over the course of eighteen months while participating actively in communities of hackers, makers, artists, and educators. This

gave me many opportunities to observe and interact with many people experimenting with system building tools, including electronics projects, software experiments, and artifacts made of physical materials such as cardboard. Computer programming, electronic circuit building, and physical materials were common elements in these projects. In the case of hackers, emerging technologies were commonly used in projects (*e.g.,* smart watches, virtual and augmented reality head mounted displays, and wearable sensing devices).

To a gain better understanding of the challenges encountered by the children, hackers, and other makers "in the wild," I experimented with a range of popular and emerging hardware platforms, breakout boards for components, and fabrication techniques while prototyping Pixel. While this allowed me to encounter more challenges first-hand, it was largely guided by the search for a combination that fit the design goals for Pixel.

Evaluation

To evaluate the design of Pixel and gain a sense of its feasibility as a tool for creating everyday systems, I conducted three workshops, each an hour long, at KID Museum in Bethesda, Maryland. These workshops were designed based on prior experience facilitating workshops, and similar to Montemayor's brainstorming sessions with children (2002) and the workshop-based deployments by Silver et al. (2012) and Buechley et al. (2006). In each workshop, I demonstrated Pixel and facilitated a hands-on, informal discussion about its design, how it could be used, and how its design could be improved. I encouraged participation from participants as design partners. I recorded video and audio of each session, which, along with my

direct observations and notes were used for my analysis and discussion. To gain greater perspective of how Pixel might be used by facilitators in making-oriented learning contexts, such as afterschool workshops, I conducted an informal focus group of "coaches" (facilitators) from FutureMakers, an organization that runs making-focused workshops in schools in the Washington, D.C. metropolitan area. During the group interview, I demonstrated Pixel and encouraged the coaches to play with it. This session was audio recorded.

In the workshop, participants created basic artifacts including a remote controlled light switch (as was illustrated previously) and a distributed, multi-person, light-based call-and-response game. Together, the participants in each of the museum workshops were able to create and change I/O channel mappings with swing and tap gestures, and connect components with alligator clips after a single demonstration. Multiple participants suggested changes to the Pixel design, including size, the addition of a wrist strap, and additional case options, signifying an interest in deeper physical interaction with the Pixels. In the focus group with facilitators, participants expressed enthusiasm for physical engagement as well, particularly to support learning computational thinking through physical movement, for example, "You could get kid who are kinesthetic learners to learn about something like programming from moving," and "It's coding at a human level.". They expressed far less interest in connecting electronic components to Pixels and programming them graphically than Pixel responsiveness to movement. Taken together, though brief, these evaluations help to highlight the benefits of Pixel and uncover important areas for future work.

Evaluations are discussed in Chapter 5.

Contributions

This thesis synthesizes concepts from prior research in tangible user interface design, physical programming, gestural interaction, circuit construction kits, and graphical programming environments, guided by personal experiences facilitating museum exhibits and workshop activities. It offers the following contributions:

It introduces Pixel, a tool designed to support the creation of interactive information systems for everyday application. This platform consists of a single module (*i.e.,* Pixel) design for building electronic circuits and programming computers. This enables constructing interactive systems with a programming environment that is distributed throughout the physical space encompassed as well as a mobile device that can connect to it.

It introduces a "loop" representation of module (*i.e.,* Pixel) behavior with which creators can interact in real-time using the graphical and tangible interfaces. This loop construct repeats automatically and continuously, providing a context for transforming system behavior in real time, between visual and gestural interfaces. This loop construct provides a common, shared communication context between for collaborative human and computer programming. This loop, inspired by the relationship between the sculptor and the potter's wheel, is a variation on the "event loop" as in Processing and Arduino.

It introduces a minimal gesture language for creating and transforming network structures for logically connecting sensors and actuators. This four-gesture language provides a way to connect electronic components that eliminates some of the technical aspects of connecting components, such as running wires,

writing computer code, or using breakout boards for communications devices. In contrast to other gesture languages for defining relationships between sensors and actuators and gesture interfaces, it does not require the use of an external peripheral for programming. This enables users to freely move through their environment and multiple users to use the interface simultaneously.

It presents a graphical programming environment design for touchscreen devices. The graphical environment is demonstrated for programming module (*i.e.,* Pixel) behaviors, including connected sensors and actuators, in real time. This language was designed to be minimal and compliment the gestural language. It is designed for mobile devices for use with one finger. The touch-based interaction design was inspired by the way a sculptor shapes clay with touch.

Overview of Thesis

This thesis is organized is follows: Chapter 2 provides background on physical computing and related challenges. Chapter 3 describes the academic literature and commercial development on tangible user interfaces and tangible programming languages, graphical programming languages, gestural interaction, and physical computing. Chapter 4 specifies design and implementation Pixel. Chapter 5 describes the three workshop evaluations conducted at KID Museum and the group discussion with FutureMakers facilitators. Chapter 6 presents a discussion of the evaluations presented in Chapter 5, conclusions drawn, and offers direction for future efforts.

Chapter 2: Motivation and Background

Though this research builds on past research, it stems from observations made while facilitating activities and exhibits at The Tinkering Studio in The Exploratorium in San Francisco, for KID Museum in Washington, D.C., at The American Visionary Art Museum in Baltimore, and in collaboration with FutureMakers. These experiences provided the formative context for this thesis, exposing physical constraints and technical challenges faced by children (and some adults) that took part in them. The observations that motivated the design aims for Pixel are enumerated below.

Confinement to the physical space near electronic and computing tools. Participants rarely built beyond the small space around the circuits they built or computing tools they were using. While participants appeared to be anchored to the computer, they also didn't seriously consider the possibility of moving away from the computer to build a large, expansive, or distributed, multi-part object. It seemed that designing the tools differently might have freed participants to consider such possibilities.

Fixation on the software environment. Tethered to a computer, participants would often give their attention to graphical software environments displayed on-screen, devoting much of their effort to drawing or adjusting the appearances of graphical objects, mostly overlooking their behavior.

Heavy reliance on the screen. Perhaps due to an increased interest or level of familiarity using computers, the screen absorbed children, perhaps because they found it was more engaging than programming or building with physical materials.

The challenges that arose were more often technical—concerning the tools being used—than conceptual or imaginative—concerning the thing being built. Generally, children seemed at ease with imagining what to do, but were puzzled by the software and hardware used.

I participated in several collegiate hackathons in search of additional examples of these challenges, hoping to gain a more comprehensive understanding of them. Chronologically, I participated in HackMIT, HackRU, MAKEwithMOTO Media Lab, hackTECH, BoilerMake, PennApps, and HackRU (again). I also co-organized and participated in Bitcamp. These events, unlike museum exhibits, took placed over the course of 24, 36, or 48 consecutive hours, and involved about 500–1,500 hackers per event, with the exception of MAKEwithMOTO Media Lab, which involved about 10 participants divided into 3 teams.

Participating in these events provided me with a rich context for observing and collaborating with fellow hackers and enabled me to gain additional perspective on the challenges (listed above) faced by children during the prior summer. Like the children facilitated at museums, hackers also fixated on the software environment. However, the reasons seemed different. Unlike the children's projects, the majority of hackers' projects seemed to be software-only. Some hackers built circuits, but they appeared to focus on "raw electronic functionality," giving little attention to their usability or applicability. Like the challenges encountered by children in museum exhibits and workshops, most that arose for hackers appeared to be technical, not conceptual or imaginative.

To supplement my investigation at hackathons, I helped facilitate senior undergraduate art students in a basic electronics workshop led by Professor Shannon Collis in the Department of Art at the University of Maryland, College Park (UMD). The students in the workshop had little or no experience working with electronic circuits, so this provided additional contrast to museums and hackathon experiences.

Finally, Conway's law (Conway, 1967) inspired me to exercise my own conception of the effect of environmental and social context on the kinds of challenges that arise while creating systems. I deliberately worked on Pixel at weekly "hack nights," organized by a group of student hackers (Terrapin Hackers), in hackerspaces, startup incubators, and co-working spaces, most notably, Collider and Startup Shell at UMD. I also set up makeshift workspaces on several-hour coastal train trips, international flights, at hostels, on apartment floors, at museums in San Francisco, at coffee shops in Lincoln, Nebraska and South Korea, and in a hostel in Aarhus, Denmark. These exercises gave me a visceral understanding of some physical constraints that result from the design of existing tools (*e.g.,* Arduino), and led me to develop a graphical programming environment to supplement gestural programming.

Observed Challenges in Physical Computing

Over the course of the above activities, I began to conceptualize the process of creating systems as physical computing, as shown in Figure 32. This model highlights the tension between working with physical materials, building circuits, and programming organizing it into three general activities.

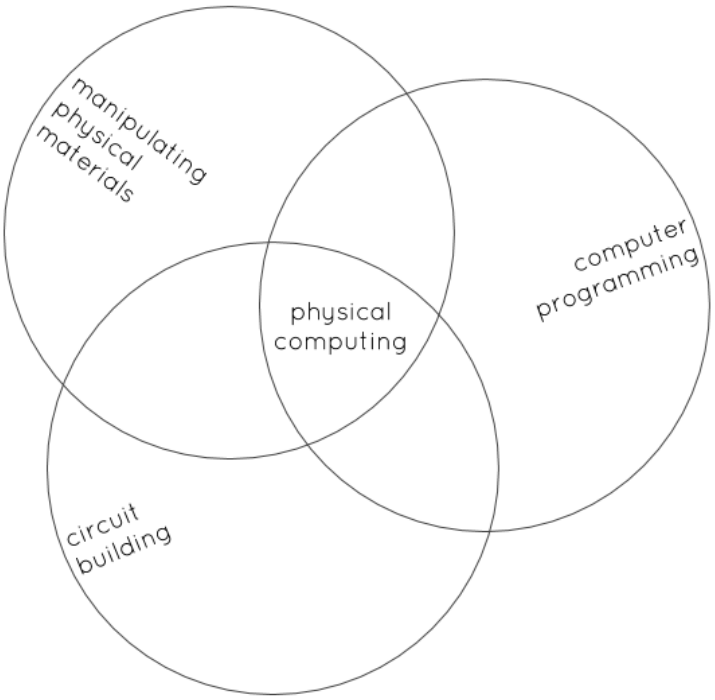


Figure 31: In general, physical computing is carried out through some combination of manipulating physical materials, computer programming, and circuit building.

Throughout my experiences working with children and participating in hackathons, I organized the observed challenges of physical computing into the following categories:

Physical computing is a multifaceted activity composed of disparate tasks, tools, and materials. It involves manipulating a variety of physical materials to form objects and environments, assembling electronic circuits and embedding them objects, and computer programming. Each of these involves a different set of skills, materials, and tools.

Not only are the areas of physical computing disparate, but also they tend to be technical and abstract. Individually, these are difficult to learn because they require separate skillsets and specialized conventional knowledge. In particular, circuitry and programming are highly technical and require both the corresponding

technical expertise to carry out the activity as well as some understanding of the underlying principles involves. Some of these principles are procedures, loops, conductivity, and current, to name a few. The need for such expertise makes them especially challenging to learn.

Individual activities are hard, but moving between them poses additional challenge, even for experienced users. Moving between these entails learning, managing, and integrating the individual processes and accounting for the context where they take place. This overhead imposes an additional cost on those doing it.

These observations are the basis on which I formulated the design aims for Pixel, discussed in the following section.

Design Aims and Methods

The aims of this thesis are the following:

Holism. Physical computing draws on an assortment of tools, materials, and methods from domains separated by technical skillsets, principles, and methods. Although achieving a singular approach to physical computing may be a dubious undertaking, there are clear opportunities for bringing it into better alignment through holistic design.

Computing tools that require physical connection to a computer can anchor users the space near it, thereby limiting their exploration of possible system designs. Pixel aims to support situated use by removing "anchors," including data transfer cables that tether users to computers too bulky to allow free movement in the environment. In particular, Pixel is designed to support distributed programmability and eliminate physical tethers.

Material engagement. This thesis strives for holistic design with the stipulation that creating interactive systems is an embodied phenomenon (Dourish, 2001). Specifically, the designs explore concepts of material engagement theory, which posits that *"[h]uman thinking is, first and above all, thinking though, with, and about things, bodies, and others"* (Malafouris, 2013, p. 77).

Inspired by experiences building interactive systems with children, I design Pixel to be nontechnical, trying to displace requirements for domain-specific technical knowledge in tools. Furthermore, this design encourages engagement with the physical environment and experimentation with materials through direct interaction.

Summary

In Chapter 4, I continue this discussion by outlining the approach taken in this thesis to address these goals. In the following Chapter, discuss work related to physical computing, discussing how it supports traditional physical computing and comparing it to Pixel from this more holistic perspective.

Chapter 3: Related Work

This thesis builds on decades of research exploring creativity support tools for making interactive physical systems. Pixel relates to rapid prototyping tools, construction kits for electronics and computing, tools for creating gestural interfaces, sensor-based programming by demonstration, particularly for gestural interaction, and embedded computing platforms. In this chapter, I discuss developments in these areas as they relate to Pixel.

Physical Computing

As highlighted in the previous chapter, physical computing, while supported with many tools, is complex and technical. This section summarizes development boards and their programming environments, rapid prototyping tools for building interactive systems, and plug-and-play hardware that simplifies use of electronic peripherals in computing.

Development Boards and Platforms

Much of the research into development platforms has focused on reducing technical challenges posed by system-building tools. The Arduino, a popular commercial development platform, provides a microcontroller development board and integrated development environment (IDE) that supports a programming language derived from C/C++ and updated for use in embedded computing projects (Mellis et al., 2007). This language simplifies common operations such as reading and writing analog and digital data from and to pins to single-line operations, making

interfacing with sensors and actuators easier. The Arduino IDE is based on Processing (Reas & Fry, 2006) and is popular among artists, hackers, students, hobbyists, and engineers with support from an enthusiastic user community.

Arduino resembles Phidgets, a system for "everyday programmers" that modified BASIC for use with custom electronic components (*i.e.,* "physical widgets") prepared for easier connection (Greenberg & Fitchett, 2001). Similarly, Logo adapted Lisp for use in controlling the robotic "floor turtle" (Papert, 1993). Recently, i*CATch, a plug-and-play construction kit explored this approach to simplify wearable and physical computing. Rather than representing programs to users as text, the i*CATch IDE denotes program elements graphically in a hybrid flow-based graphical programming environment (GPE) that generates textual code for the Arduino IDE (Ngai et al., 2010).

In contrast to the i*CATch IDE, Pixel provides a custom minimal GPE design for programming uniform tangible modules to which off-the-shelf electronic input and output (I/O) components can be connected with alligator clips.

Rapid Prototyping Platforms

Support for rapid experimentation with a variety of system designs is increasingly common feature of tools in physical computing.

The d.tools and BOXES systems are two such systems. d.tools is a statechart-based visual design tool for early-stage prototyping in design (B. Hartmann et al., 2006). BOXES focused on rapidly exploring interactive physical forms early in the prototyping process (B. Hartmann et al., 2006). Whereas d.tools fosters the development of higher fidelity prototypes through iteration, BOXES focuses on use

of everyday materials such as cardboard, tape, thumbtacks, and glue, to build early, functional lower-fidelity prototypes. While Pixel shares features with each of these systems, it relates most closely to BOXES. Like BOXES, Pixel is designed for building with everyday materials and supports digital input through capacitive touch sensing. However, Pixel, unlike BOXES, supports connection with output devices, and critically for its design, sans tethering to a desktop or laptop computer at any point in the building process.

Plug-and-Play Hardware

The d.tools design support custom plug-and-play hardware, similar to Phidgets (Greenberg & Fitchett, 2001). Both systems leverage plug-and-play hardware interoperating with desktop software, however, only d.tools provides a graphical programming environment. (Phidgets provides a textual language specialized for use with its plug-and-play components, as mentioned previously.) The GPE provided by d.tools enables designers to program prototype hardware by defining system states and defining control flow through transitions established by connecting input components on a (source) statechart to another (destination) statechart through drag and drop operations performed with a mouse. The input triggers the transition upon activation. Hardware prototypes are made by connecting devices to a central hub, connected over USB to the computer running the d.tools software. Pixel has no central control point, requires no tether to a desktop computer, and uses common I/O components. To balance between the convenience of plug-and-play devices and the availability of off-the-shelf components, Pixel provides

"magnetic snap connectors" for providing both ease of use in connecting components and convenience of readily available parts (discussed in Chapter 4).

Tool and Construction Kits

Inspired by the ease of using construction kits like LEGO, researchers have developed construction kits for building electronic circuits (*e.g.,* littleBits) and embedded computer programming (*e.g.,* Modkit Micro). These have broadened participation in physical computing to hobbyists, students, and designers. Pixel differs from many construction kits in that it provides only a single *unit* module design.

Electronic Circuit Kits

Circuits are systems of special-purpose components, each a small system of materials that define a specific electronic operation. Circuits are built by patterning components to control the operation of multiple components. Building circuits requires technical skill special with components ranging from understanding theoretical models (*e.g.,* Ohm's law) to experience applying conventional practice (*e.g.,* measuring voltage with a multimeter).

Modularity: In the 1960s, the Lectron electronic building set, or "Electronic Dominoes," was introduced to reduce the complexity of working with electronic circuits (Greger, 1965, 1969). The Lectron set is modular and abstracts electronic components, including resistors, capacitors, button switches, lamps, and even transistors[1], simplifying circuit bulding. Modularity is a key design feature in more recent kits such as Tower (Lyon, 2003), Cube-in (Gross, 2014), SAM ("SAM," 2014), as well as Pixel. Tower is designed with a number of "stackable" modules that can be layered onto one another to extend system functionality. Cube-in, a physical computing kit for children, adopts modules as cube-shaped "plugs" that are inserted into corresponding sockets a central base to form program expressions. SAM and Pixel incorporate modularity similar to each other, however SAM's modules are embedded with specific components (like littleBits), whereas common components can be connected to Pixel.

Moreover, unlike these recent kits, Pixel has one module design. While many systems incorporate modularity, relatively few adopt a unit design. Commonly, modules have a one-to-one correspondence with electronic components, as in

---

[1] It is interesting to note that the first commercial silicon transistor was produced in 1954 by Texas Instruments. The Lectron set made this new technology available as one of its bricks. Likewise, the first interlocking Lego began production in 1949. One can speculate that the modularity and ease of use of Lego bricks may have influenced the modular design of the Lectron set.

littleBits (Bdeir, 2009), LightUp (Chan et al., 2013), and SAM, although perhaps with support circuitry for interfacing with other modules. With Pixel, as with BOXES, common off-the-shelf components and capacitive touch can be used to connect input components. For Pixel (but not BOXES), common electronic output components can be used as well.

Kits for Children and Learners: Features of the Lectron, also found in littleBits and LightUp, simplify aspects of electronic circuit building for children (*e.g.,* Johnson & Thomas, 2010). These kits have been designed primarily for use with their respective proprietary components, not common, widely available electronic peripherals, limiting their extensibility. Others have simplified interfacing with prototyping platforms to ease connecting physical materials and electronics. For example, MaKey MaKey allows children to create electronic switches and improvise input devices to the computer using a wide range of conductive materials, including bananas, copper tape, wire, fruit, and skin (Silver et al., 2012). Pixel takes this concept further by additionally allowing children to connect output devices to modules (in addition to inputs) to create conditional I/O interactions and parallel sequences of conditional behaviors (Chapter 4).

Kit Module Design: Like the Lectron set, SAM consists of a number of modules, each a single electronic component, such as a button, light, servo, slider, tilt sensor, DC motor, light sensor, thermometer, and "cloud" modules. littleBits and LightUp also include modules associated with different electronic components. Pixel supports connection to one input and one output component using conductive materials such as alligator clips. Compared to the MaKey MaKey, Pixel supports an

output device and does not rely on an external computer to operate. The input device connected to the MaKey MaKey, itself connected to a computer over a physical USB connection, is recognized as a keyboard and mouse on the computer. Like MaKey MaKey, BOXES, and d.tools must be physically connected to a computer[2].

Communications: SAM's modules communicate with one another wirelessly (using Bluetooth Low Energy). Pixel's modules communicate with one another wirelessly over a mesh network and with a mobile device running the GPE over Wi-Fi. Pinoccio ("Pinoccio," 2014), an Arduino-like platform also supports mesh communication.

Programming: SAM, Pinoccio, and Pixel can be programmed wirelessly, and to varying degrees, in real time. SAM uses a flow-based, graphical programming language in which the modules are rendered graphically and can be moved about a 2D plane, can be connected by dragging from one component and dropping onto another (using a pointing device such as a mouse), generating JavaScript code as it does so. Programs can also be coded directly in JavaScript. Pinoccio is programmed using the Pinoccio HQ, a Web-based interface that provides limited real-time on-

---

[2] It is important to note that a separate peripheral device can add Bluetooth support to the MaKey MaKey. Using this peripheral involves updating the firmware of the MaKey MaKey to interface with the Bluetooth peripheral and communicate with a computer as a wireless keyboard.

board LED and pin I/O control, a shell interface called ScoutScript, and a RESTful API available for JavaScript. Pixel is also wirelessly programmable. In contrast to SAM and Pinoccio, Pixel allows basic I/O mappings to be programmed with gestures in real time, requiring no screen.

Tethering to External Computer: It appears that a computer running the SAM programming environment is needed to orchestrate the functioning of SAM modules. However, like both Pixel and Pinoccio, the modules are battery powered and can be recharged with a USB cable. Pixel does not rely on any external computer for its operation, aside from using the graphical interface to change system behavior. Like SAM, Pixel consists of modules that are battery powered, allowing them to be used away from a computer or other power source. BOXES, d.tools, and i*CATch require a tethered computer to program modules. Phidgets requires tethering to an external computer to perform processing.

Note that at the time of writing this thesis, SAM was not yet commercially available and little technical detail is available about it beyond news articles. As a result, I was unable to determine absolutely whether a computer running the SAM programming environment was needed, but it does seem to be the case based on several news articles. I was also unable to determine if each module included an embedded computer, and therefore, if so, whether the embedded computer could be programmed using SAM's graphical programming environment. Finally, I was not able to determine whether the SAM modules, once programmed, could run as a standalone system without a computer running the SAM programming environment. These relate to considerations fundamental to the design of Pixel and are important

points of comparison between the systems that should be considered by the critical reader, if feasible.

Building Bricks and Magnetic Coupling: The Lectron set also introduced some features that are still used in physical computing tools today, including Pixel. These include a "building brick" module design, magnetic attraction to produce "the necessary contact pressure between the contact plates of the two bricks" to facilitate connecting components, and abstract representations of technical components contained in modules (in its case, graphical symbols for notating electronic circuit diagrams). In contrast to Pixel, the Lectron set includes no brick containing a power source. Instead, it includes a brick to which external power source could be connected, limiting its use for physical computing.

Material Manipulatives & Digital Manipulatives

Interacting with artifacts and environments has long been studied in education as a possible way to enhance learning. In 1837, Friedrich Fröbel, the inventor of the first kindergarten in Germany, designed a diverse range of physical objects to help children learn and to recognize the common patterns and forms found in nature (Brosterman, 1997). These material manipulatives served as "thinking tools" by providing a "medium" for children to explore abstract concepts such as shape, size, and color.

Prompted by the promise of material manipulatives such as to help engage children in learning and explore their interests, researchers are investigating the role of manipulating materials with embedded computing (Raffle, 2004; Resnick et al., 1998; Zuckerman, Arida, & Resnick, 2005). In particular, researchers have looked at

how the computer can be used to create *digital manipulatives* that might help children discover and create patterns in the physical *and digital* worlds. Digital manipulatives combine elements of creating hybrid digital and physical systems in attempt to help children learn about dynamic systems through building and play (Resnick et al., 1998).

Children commonly use digital manipulatives in museum exhibits and after-school workshops. Researchers have invented digital manipulatives for finding novel approaches to help children learn about complex concepts in mathematics, programming, and robotics (respectively, Papert, 1993; Resnick et al., 2009).

MIT has pioneered much of the research into digital manipulatives, including a number of "programmable bricks," brick-shaped digital manipulatives for composing computer programs through physical construction. McNerney discusses this work in the larger context of educational computing research done at MIT (2004).

The Cricket platform, developed in the late 1990s, provided new direction for programmable bricks, being based on the notion that computational processes should be made visible and manipulable. The Cricket was the first in a family of general-purpose programmable computing platforms designed for use in learning about science and engineering. The Cricket provided two ports for connecting sensors and two output ports. It was programmable from a computer using a dialect of the Logo programming language specialized for the platform. Logo programs are communicated to the Cricket by way of an infrared transceiver tethered to the computer. The Cricket inspired a number of similar platforms based on its driving concept—including the HandyBoard, PicoCricket—and later extensions to the

concept—exemplified by MetaCricket, PicoCricket, Phidgets, and the GoGo Board. These systems are discussed by Blikstein (2013) and McNerney (2004).

Programming Interactive Systems

One of the earliest educational computing platforms focused on combining computing and building was the LEGO Mindstorms Robotics Invention System (McNerney, 2004). Mindstorms was designed around a new brick into which a battery-powered computer was integrated. The system introduced new sensor and actuator pieces connectable to the central brick using the same connection interface used by traditional bricks. Mindstorms is programmed in a graphical flowchart language designed specifically for use with the system. The graphical language presents programing constructs as jigsaw puzzle pieces that only allow syntactically acceptable connections to be made.

Gestural Interaction and Programming

Gesture been found to be tightly related to language development (Iverson & Goldin-Meadow, 2005), to augment mental representations of explained tasks by adding action information (Beilock & Goldin-Meadow, 2010), and to influence the actions of others (Cook & Tanenhaus, 2009). Gestures affect and can aid in learning mathematics (Novack, Congdon, Hemani-Lopez, & Goldin-Meadow, 2014) and they may lead to the construction (rather than reflect) of certain mental representations (Trofatter, Kontra, Beilock, & Goldin-Meadow, 2014).

Gesture is a natural and intuitive part of every human discourse. As such, supporting the use of gesture to interacting with computers has appealed to

researchers for decades and continues with advancements in sensing technologies. Topobo is a distributed, modular set of components designed to learn about the form and behavior of dynamic systems (Raffle, 2004). Topobo consists of a number of active and passive tangible components that can be snapped to one another to create robotic "animals," geometric configurations, and abstract forms. The active components contain kinetic memory enabling them to be programmed by demonstrating behaviors through manipulating them by hand. This technique records motions performed (demonstrated) and plays them back, animating the built form. Topobo's record and play model was based on that of the curlybot (Frei, Su, Mikhak, & Ishii, 2000).

Like the *physical programming* tools introduced by Montemayor (2003), Pixel supports the use of gesture for defining interactive relationships between sensors and actuators (those components that are connected to Pixels). However, unlike Montemayor's tools, Pixel does not require the use of external tools (*i.e.,* using a "magic wand" to do gestures while wearing a "wizard hat") for the performance of gestures. Instead, users can directly swing, shake, and tap Pixels together to create these relationships. Gestural interactions are intended to compliment the manipulations involved in connecting electronic components to the modules during system building. Pixel's gestural interaction design is closely related to OnObject (Chung, 2010) and The Toolstone (Rekimoto & Sciammarella, 2000). Below, I discuss these as they relate to Pixel. While discussing OnObject, I also relate it to MaKey MaKey to help point out the differences between the tools.

OnObject is a tool for creating gestural interfaces using everyday objects, including homemade objects. Although OnObject does is not for creating systems with sensors and actuators, it does use a set of gestures virtually identical to those supported by Pixel, but to record and trigger sound samples associated with gestures performed with specific objects.

While OnObject is used to create interactive gestural interfaces, MaKey MaKey is for creating tangible interfaces. Both of these are similar to Pixel in that they are tools for creating systems, however they differ in the methods by which users create interfaces with them, the kinds of interactions afforded by the built interfaces, as well as in their underlying interaction mechanisms. OnObject produces a sound in response a particular gesture, and like the MaKey MaKey, it can emulate a key press on the USB-connected computer, in turn, affecting applications monitoring keyboard input. As discussed previously, Pixel can be used to create interactive input interfaces like MaKey MaKey, but its output is a built system rather than software on a computer. Similar to the MaKey MaKey, Pixel allows users to incorporate their body into circuits. However, Pixel uses capacitive touch sensing rather than high resistance switching to implement digital switching.

Graphical Programming Environments

Pixel offers a graphical programming language, accessible from mobile devices, for finely tuning the behavior of individual modules, systems of modules, and connected peripherals.

Squeak eToys is an environment for graphically authoring multimedia objects such as custom digital drawings, images and videos, text, and sound (Kay, 2005).

Sqeak eToys provided conceptual inspiration for Pixel. Pixel is similar to Sqeak eToys in that it provides an environment for creating objects and exposes operations to transform their behavior *in situ*. In contrast to Pixel, the Sqeak eToys environment is displayed graphically on a computer screen and is designed for use with a keyboard and mouse. Simple objects from geometric shapes to representational objects (*e.g.,* cars) are provided for modification and use in projects. Custom objects are made with simple drawing tools and object-specific operations, represented graphically, for changing the properties and behavior of objects. Pixel further differs from Sqeak eToys in that it supports building physical objects rather than virtual objects and exposes operations for transforming object behavior through physical modules and on a mobile device.

Scratch is a graphical programming language designed to give children the ability to create and share interactive stories, animations, and games (Resnick et al., 2009). Scratch provides a language comprised of graphical block elements that fit together in a similar fashion as pieces in a jigsaw puzzle. Scratch provides a "stage" that shows the graphical output drawn by the scripts, an "sprite" editor that can be used to create, edit, and animate graphical sprites, and a script editor. There are many scripts available for use, including those for controlling the motion and appearance of sprites, creating turtle graphics, control structures, basic sensing, logical operators, events, and data such as variables. Scratch has been successfully introduced many children to programming concepts in an engaging manner (*e.g.,* Franklin et al., 2013).

While Scratch provides little native support for connecting to and control peripheral devices or programming physical computing platforms, it has been

extended to include some support for both of them. For example, Scratch can be extended with a plugin to interact with the LEGO WeDo kit.

Scratch was extended to support the LEGO WeDo construction kit, designed for use by children (aged about 7–11) in constructing simple robots. While designed for robots, it can be used for more general physical computing projects. The WeDo kit includes motor, light, distance sensor, tilt sensor, and a hub for connecting to a computer (running Scratch) over USB. These parts are physically designed to be compatible with LEGO kits. Each one consists of a connector—for connection to the hub part—and an electronic component separated by a relatively short length of wire. While these parts can be combined to form many objects, the projects achievable with the kit are quite limited in the context of general physical computing.

The functions of WeDo parts are accessible in Scratch as "Extension Blocks" (enabled by installing an extension) and can be combined with the existing Scratch blocks to create interactive experiences incorporate both the WeDo bricks and graphical output rendered in Scratch.

Scratch has been adapted specifically for use with commonly used physical computing platforms, including the Arduino. It has been highly influential in the design of a number of other GPEs for controlling peripheral devices that have simplified aspects of programming in physical computing. Modkit (and Scratch4Arduino), a GPE heavily influenced by Scratch, suggests that it simplifies some programming tasks on the Arduino (Booth & Stumpf, 2013).

The extensiveness of the design language popularized by Scratch is helpful in that it can help those already familiar with Scratch. Some of these also improve on

aspects of the Scratch interface. For example, Tickle implements the interface for tablet devices affording mobility and interaction through touch ("Tickle," 2014). Tickle is a tablet application that extends and adapts the Scratch design quite closely ("Tickle," 2014). Tickle is designed specifically for mobile devices and incorporates multi-touch screen-based gestural interaction. It also introduces high quality graphics (*e.g.,* new characters) and sound effects. Tickle can be used to create full-screen interactive graphical experiences and provides limited support for programming of physical systems (including RC drones).

Unlike the Scratch family of programming environments, Pixel explores a new approach to graphical programming that more easily affords real-time interactive programming (or live programming) of system behavior. In particular, Pixel offers a continuous looping context into which actions are placed. This allows system actions to be added, transformed, and removed in real-time in a simple manner using only touch-based interactions on the graphical language. This is similar to the dataflow model of graphical programming languages.

Summary

The range and variety of related work presented in this chapter illustrate the richness of tools for building interactive systems, and capture some of the breadth in approach to physical computing. I provided examples of work closely related to Pixel, specifically focusing on construction kits and programming methods offered by existing tools, orienting them in relation to Pixel, highlighting their weaknesses in supporting users to freely explore their physical environments and build systems that

extend through environments without the need to be tethered to a computer at any point during the creative process.

	Arduino (Mellis et al., 2007)	**Scratch** (Resnick et al., 2009)	**MaKey MaKey** (Silver et al., 2012)	**d.tools** (Hartmann, 2009)	**OnObject** (Chung, 2010)	**Topobo** (Raffle, 2004)	**StoryRooms** (Montemayor, 2003)	**Pixel**
Typical User	Hobbyists, artists, hackers	Children	Children	Designers without specialized engineering or programming knowledge	Casual users (e.g., parents and preschool children)	Children	Children	Everyday users
User Goal	Construct interactive and autonomous objects	Author interactive multimedia with basic interaction with the environment	Construct a custom controller for a computer with common materials	Rapidly prototype a functional interactive product	Turn objects into gestural audio interface	Construct and define the behavior of robotic creatures that resemble animals	Create interactive room-sized environments to facilitate storytelling experiences	Create interactive systems for use in an everyday situation
Technical Knowledge Requirements	Arduino programming language, circuit design, familiarity with desktop computing	Basic programming concepts	Basic computer use, web browsing	Familiarity with basic circuitry and statechart representations of system behavior	None, basic GUI to map additional responses	None	None	None
Physical Constraints Imposed on User and System Design	Tethered to computer, fixes attention on screen during programming	Fixes attention on screen during programming	Only supports making controllers for computer software	Tethered to computer, anchored to single location	Tethered to computer (prototype), must wear ring device, anchored to a location	*(System)* Can only interoperate with kit parts	Tethered to computer (prototype), must wear a wizard hat and use magic wand for programming	Relatively large Pixel size can make them difficult to embed in objects
Modalities Supported for Making Systems (Programming, Circuitry, Construction)	Connecting electronic components to form components, programming their relationships with a keyboard (and mouse)	Connect graphical components together like jigsaw puzzle pieces with mouse and keyboard	User creates input devices and connects them to a computer	Drag and drop mouse and keyboard actions with statechart editor, connecting plug-and-play electronic components	Attach RFID tags to an object, wear ring device, connect the device to a computer	(Physical) Connect parts from the kit to one another, (Behavior) move parts relative to one another to demonstrate behavior	Wear wizard hat, tap magic wand to sensors and actuator "physical icons"	Place pixels in objects and environments, create custom sensing and actuation circuits; perform gestures while holding pixels; drag and drop through direct touch on a mobile device
Representation and Interface of System Behavior	Symbolic Turing complete programming language (similar to C)	Graphical programming language (based on Sqeak)	N/A	Graphical statechart diagrams specifying device state transitions (and Java)	One to one mappings from gestures to audio recordings	Physical movement of a robot's parts	Many to one interactions between specific sensor and actuator "physical icons"	One to many interactions between an input port and output ports; conditional actions in a loop
Final System Format and Capability	Connected digital and analog output devices (*e.g.,* to controlling custom hardware)	Multimedia (*e.g.,* graphics and sound)	Keyboard and mouse events, Computer software (*e.g.,* a custom Scratch game)	Connected digital, pulse width modulation (PWM), and I^2C components	Objects that play audio in response to gesturing with them while wearing the ring device	Physical robotic creatures that resemble animals and perform demonstrated movements	Environments that respond to defined interactions between sensor and actuator "physical icons"	Integrated RGB LED, speaker; connected digital output components

Table 1: Summary of tools for creating systems that are similar to Pixel.

Chapter 4: Design and Implementation

Pixel is a tool for use in everyday situations for making interactive information systems that can sense and respond to the physical environment. It is composed of three interfaces, each with unique affordances that support making the different features of systems. This chapter discusses the design of these interfaces.

System Design

Pixel is a tangible user interface (TUI) composed of one or more cubic objects called Pixels. Individually, Pixels are small computers that can communicate with each other, other devices, and interact with their environment through sensors and actuators. Together, Pixels form a single system. Every Pixel in a system maintains its own representation of the system and each Pixel in it.

To support interaction with users, each Pixel recognizes a set of gestures, discover and communicate with other modules through a mesh network, and communicate with devices such as a handheld or tablet computers wirelessly. Pixels can be used together as a single system. To support computations that do not correspond to a simple gesture, Pixel offers a graphical programming environment (GPE), designed specifically for use on mobile devices that support touch-based interaction, such as cellular phones and tablets.

Module Design

During the development of Pixel, I considered each individual Pixel to be a *unit construct* of sensorimotor interactivity (or, alternatively, a "unit" of I/O). That is,

each one can interface with one sensor and one actuator. These were designed to enable the construction of interactive systems of varying complexity (*i.e.,* with different numbers of modules).

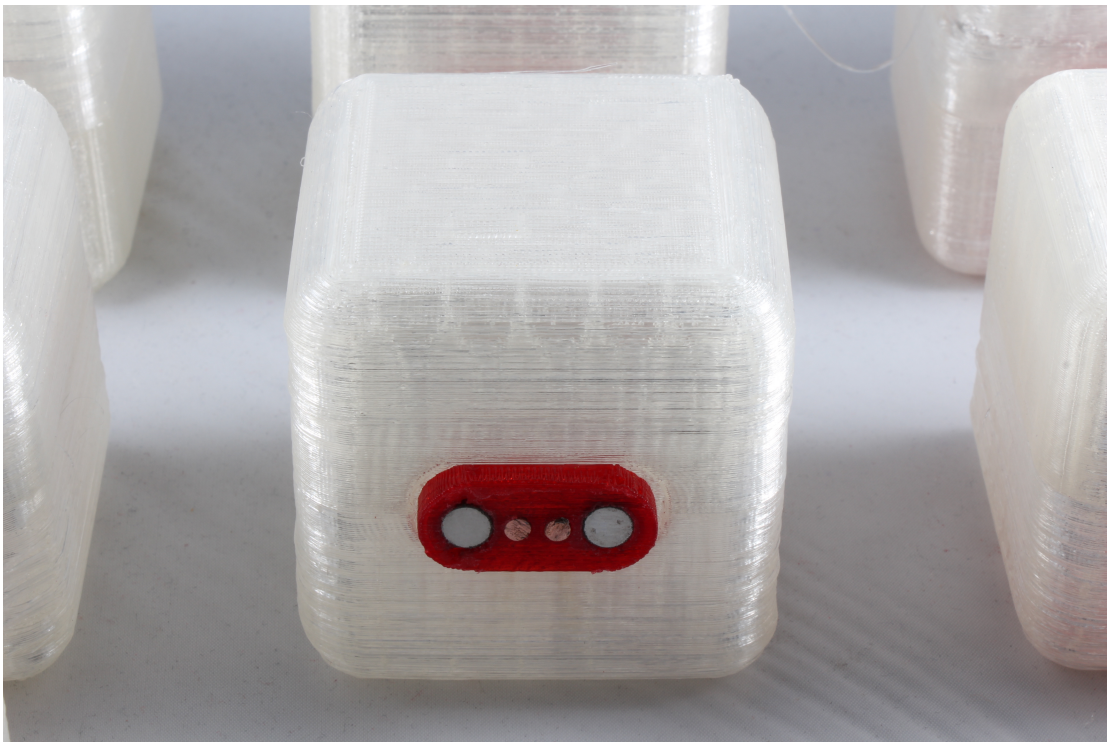


Figure 32: One Pixel.

Conceptually, an individual Pixel can be considered as roughly analogous to a single digital I/O pin on a development board such as Arduino. That is, each Pixel exposes a one port for connecting sensors (or input) and one for connecting actuators (or output). Similarly, a set of Pixels can together be seen as roughly analogous to a complete development platform with all of its pins. In opposition to development platforms, the number of units (*i.e.,* Pixels) available determines the number of I/O ports available for use with Pixel.

There are a number of external features apparent on each Pixel. First, each one extends about 2.5 inches in each dimension, forming a rounded cube. Several sizes

were considered during prototyping. The 2.5 inch cube was selected because it seemed was comfortable to grasp, manipulate, and hold in one's hand.

I fabricated several candidate Pixels. These were printed with a number of different 3D printers using both acrylonitrile butadiene styrene (ABS) and polylactic acid (PLA) filaments in several colors. The final Pixel design was made with natural (non-colored) translucent PLA because it allowed light emitted by the enclosed LEDs the pass through it visibly while also obscuring, and thereby visibly abstracting, its internal circuitry.

The thickness and internal support structure of the exterior wall were optimized to minimize wall thickness while increasing the distribution of light across the surface of the module (making the color more visible for users to see), and to increase robustness (so it can be dropped, for example). The chosen size also proved to be reliably printable and housed the hardware and batteries comfortably.

Each Pixel exposes connection ports for connecting one digital sensor and one digital actuator. The ports are identical and are positioned on opposing faces of the cubic form as shown in the figure below.

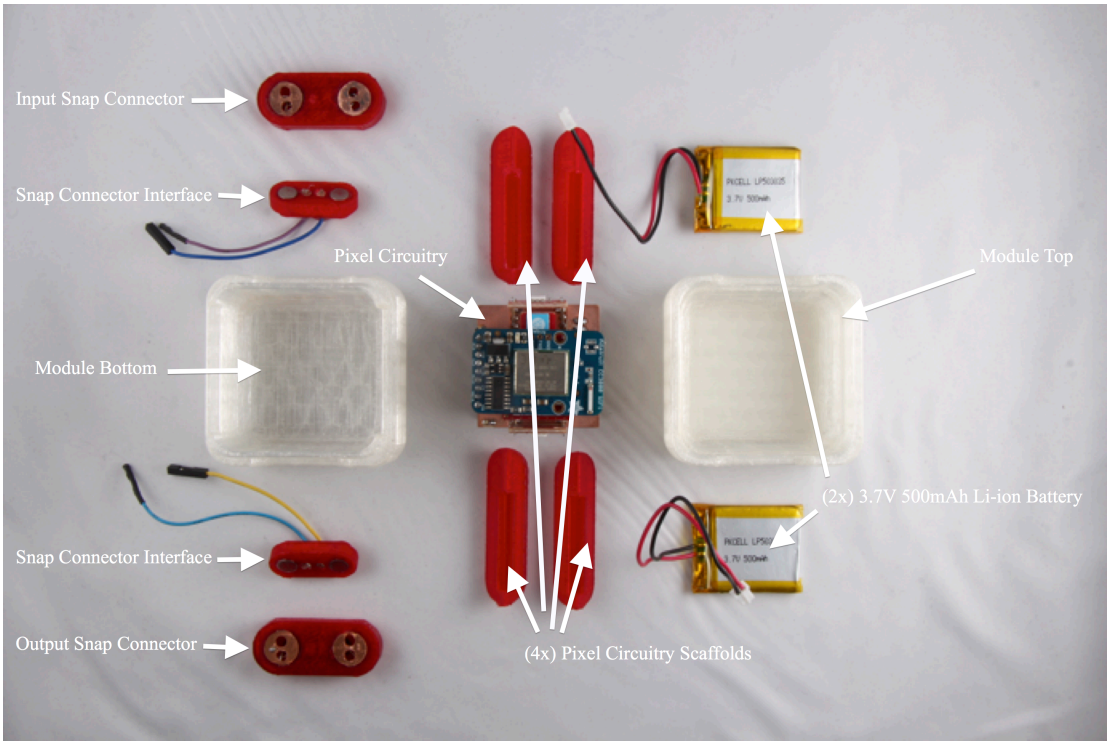


Figure 33: One complete Pixel module, opened and arranged to show the contained printed circuit board and parts.

Pixel is designed for use with magnetic snap connectors that magnetically couple with modules and expose ports to which users can connect electronic input and output components. The ports on modules do not support direct connection to electronic peripherals.

Magnetic Snap Connectors

Digital electronic sensors and actuators can be connected to Pixels using magnetic "snap" connectors. The connector can be "snapped" onto and "unsnapped" from the Pixels. This design facilitates quick experimentation with sensors and actuators and seems to add flexibility in connecting components when Pixels are embedded in objects or environments. The snap connector is shown below:

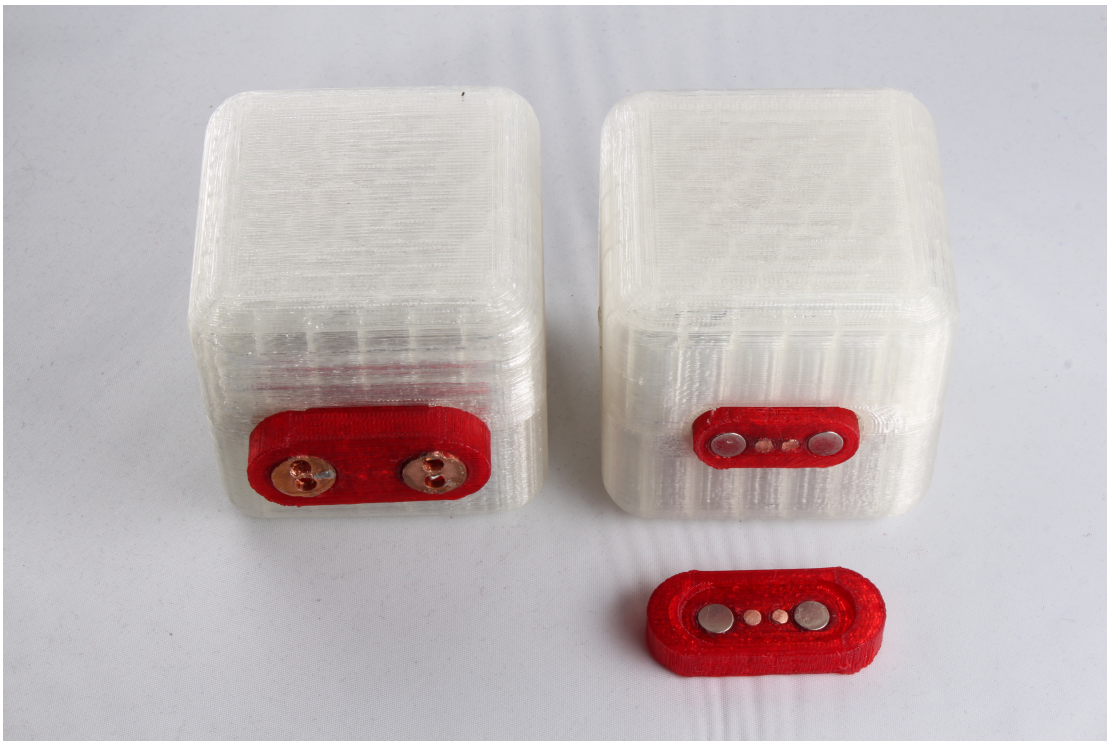


Figure 34: Photograph of the final prototype, showing the interchangeable port adapters, both attached (left) and detached (right) from modules.

On the front of a snap connector is connection interface consisting of two circular copper "outlets" designed specifically for connecting alligator clips. The outlets each expose two circular cutouts that guide the user in connecting the clips (see Figure 9, Top). Each copper outlet supports only one way to easily connect alligator clips, limiting the possible ways that the clips can be attached onto them.

On the back of each snap connector are the faces of two rare-earth magnets and two copper rods physically connected, through the plastic body of connector, with conductive wire and glue to the copper outlets on the front of the connector (Figure 9, Bottom). The magnets "snap" the connector to a module, physically holding the two copper rods on the connector against two opposing copper rods embedded in the ports. This provides a conductive path between a Pixel's internal

circuitry and the copper outlets to which components can be connected with alligator clips.

Figure 35: The front (top) and back (bottom) of the "snap" connection interface.

The decision to design a connection interface for alligator clips was inspired by experiences using the MaKey MaKey (Silver et al., 2012), which adopts a similar design. However, the MaKey MaKey only supports connection from input components. Pixel supports both input and output components. (Power is provided to output components from the battery enclosed in the module.)

Module Behavior

Pixel's behavior can be organized into two general stages. The first stage is "initialization," performed when a module is powered on, to make it operational as one system with other modules. The second stage is "ongoing" behavior. Below I discuss each of these, how they support interaction with users and how they support interfacing with sensors and actuators.

Initialization Behavior

Once Pixels receive power, they attempt to construct a "collective identity" that serves as a logical grouping construct that enables Pixels to interact with each other. Although Pixels are aware of each other, they can be grouped by identity to logically separate groups from one another. This could be done to separate different users' Pixels or organize the Pixels used in different systems.

The initialization behavior is done when a Pixel is turned on for the first time. When turned on, a Pixel turns on bright ("wakes up"), then checks if it has been given a collective identity. If not, it shuts off its light momentarily, then turns on dim ("in a fog"). At this point, a user could do one of two things. They could tap the Pixel to another Pixel, to give it that Pixel's identity (limiting it to use with other Pixels sharing the same identity), or they can shake it, to instruct it to remain without a collective identity (making it compatible with all other Pixels, regardless of their identity, and enabling it to serve as a bridge between sets of Pixels with different identities). Once given an identity, a Pixel will store it in its non-volatile memory (EEPROM), so it can recall it when it is next turned on. The identity can be removed through continuous shaking until the module plays a sequence of tones in rapid succession (*i.e.,* for 5 seconds continuously).

Ongoing Behavior

Pixel's primary behavior is supporting computing through physical action. Users can direct its behavior by performing spatial gestures with modules and surface gestures with the graphical programming environment (see **Figure 43**).

Default Behavior

By default, each Pixel detects the state of its electronic input switch (either "open" or "closed"), and, if "closed", powers a connected electronic output device, if any. This succession of behavior from input to output is called the module's activation routine. For example, a switch could be connected to the input port and a direct current light emitting diode could be connected to its output port with alligator clips as depicted in **Figure 36**. When the switch is flipped, the output port is activated, powering the diode as shown **Figure 36**.

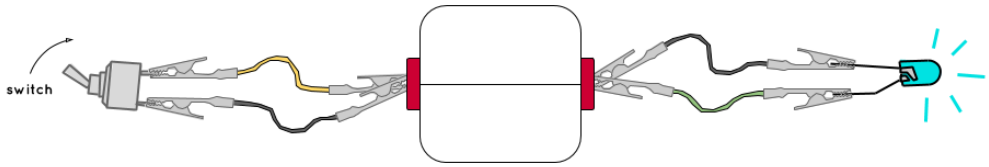


Figure 36: Drawing of a Pixel before (left) and after (right) it is switched on.

Custom Behaviors

When using Pixel, users customize the behavior of modules with spatial and surface gestures. Pixel's interfaces that enable customization are discussed in the next section.

Using Pixel: The Interface and Interaction Design

Users interact with Pixel through the gestural and graphical interfaces. The gestural interface is used to create and change I/O mappings between modules. The graphical interface, designed for use on handheld and tablet computers, gives users an environment in which they can interactively program module behavior.

Affording Embodiment: Movement, Gesture, Touch, and Manipulation

Pixel offers users interfaces affording free movement through the physical environment, gestural interaction with modules performed while holding modules, more specific single-finger touch gestures performed on a mobile device, and manipulation of and connection to electronic components using a custom connector and alligator clips (**Figure 37**).

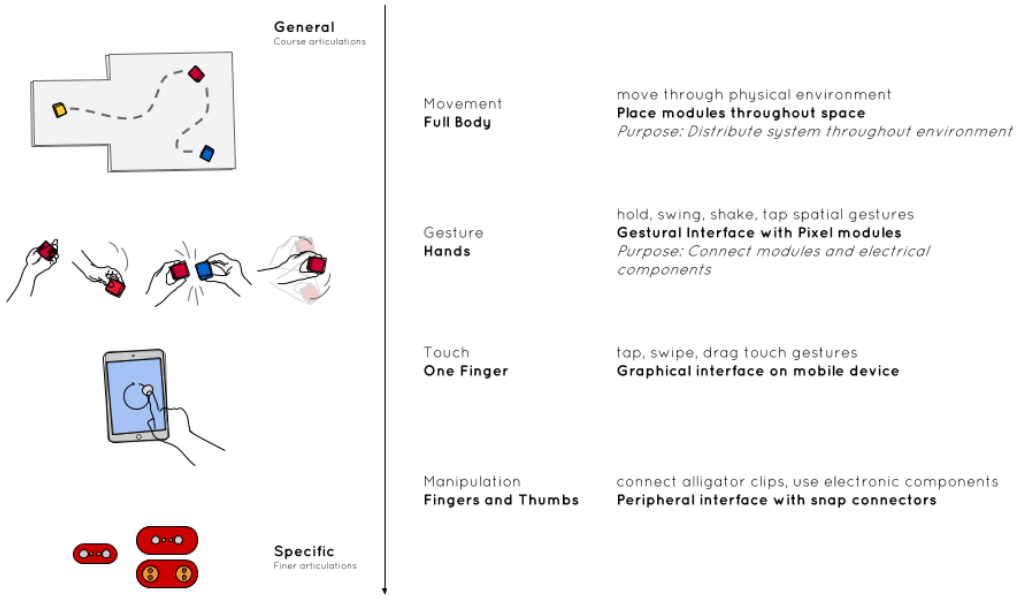


Figure 37: Drawing and description of the general relationships between the movement, gesture, touch, and manipulation engagement styles.

Generally speaking, these interfaces were designed to facilitate a system-building progression from "general" to "specific" styles of engagement with the physical environment, materials, and interactive behaviors that compose an interactive system. Because Pixel is modular and the graphical composer can be access from mobile devices, transitions between the movement, gesture, touch, and

manipulation interfaces are relatively seamless. Each of these interfaces are discussed in the following sections.

Material Engagement: Modularity and the Physical Interface

Pixel's modular design enables users to freely place modules throughout their environment as they build an interactive system. Modules automatically discover and connect to each other, providing a modular tangible user interface with which users can perform gestures to create I/O channels across modules. This provides users with a quick way to experiment with different input and output component combinations and interactions during system design. The graphical programming environment can be accessed from a mobile device such as a mobile phone and automatically discovers modules and provides a real-time programming interface. Users can freely transition between these activities to experiment with interactive behaviors and system designs as they assemble a system's physical topology (**Figure 38**).

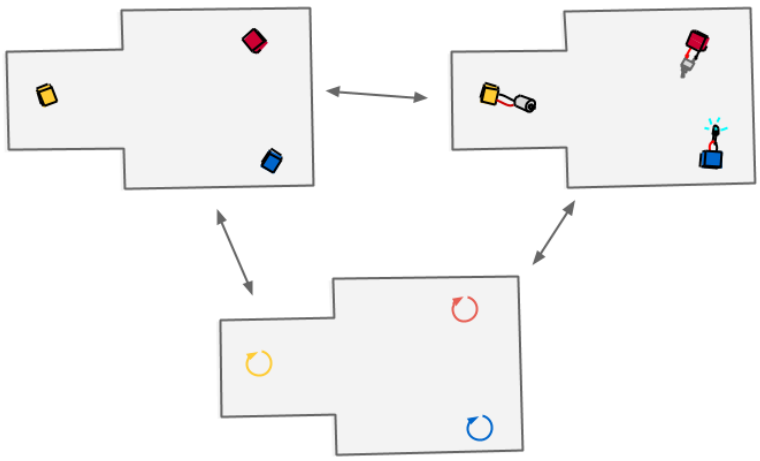


Figure 38: Movement is afforded by Pixel's modular design, enabling users experiment with system interaction designs by arranging modules in their environment, gesturing with modules, connecting components, and programming module behavior using the graphical programming environment on a mobile device.

Defining Multi-Pixel Interactions: Gesture and the Tangible Interface

Pixel can recognize four gestures, each characterized by the movement of a module over time. The "at rest" gesture is a relatively static, characterized by little movement, and results in no action when recognized. The "swing" gesture engages a module, enabling it to be used to create I/O mappings between modules. Tapping a module engaged by the swing gesture to another module results in a "tap" gesture, and maps the input of the engaged module to output of the module against which it was tapped. The "shake" gesture is only effective on an engaged module, and disengages it.

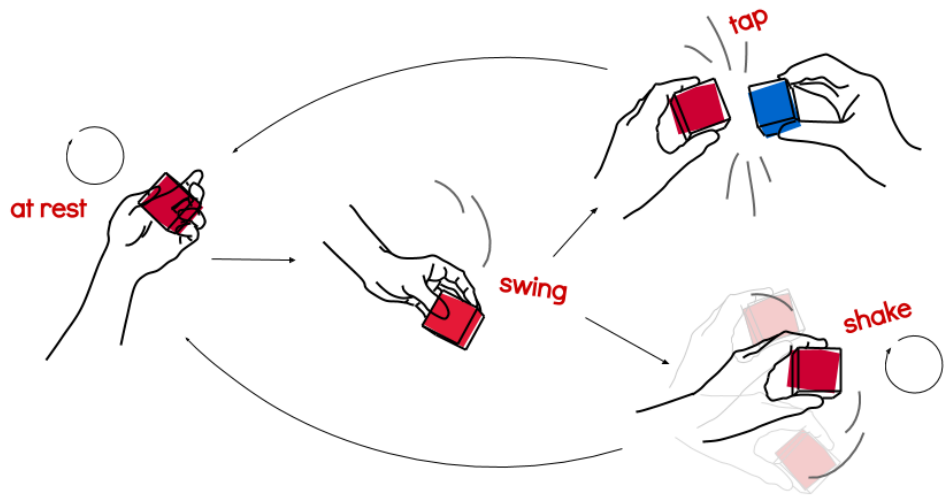


Figure 39: Drawing of the state transition diagram underlying the gestural language.

The gestural language was designed to be simple for nontechnical users to learn and do after one or two demonstrations. With this goal, gestures were chosen that were simple but result in an effect that is "complete" in that it affects only and all of the modules used to perform a gesture and resembles an analogous common experience with a similar effect. The swing gesture is kinematically analogous to a finger touching a screen or pressing a button. Tapping modules together resembles joining objects to each another. Shaking an Etch a Sketch erases the toy's screen.

To use gestures, modules must be in a second mode called "composition" mode. In this mode, gestures are used to initiate actions that affect the behavior of Pixel.

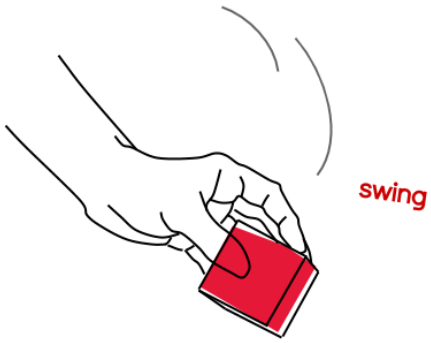


Figure 40: Drawing of the swing gesture. The swing gesture starts an interactive dialogue with Pixel in which it can assist in creating, remapping, and removing I/O channels through additional gesturing and feedback.

Swinging a module expands the scope of Pixel's interactive behavior, enabling the user to perform the tap and shake gestures to setting up I/O mappings between modules.

Pixel responds to a gesture by providing direct visual feedback and changing Pixel's behavior as defined by the gestural language. For example, "swinging" (raising a module, then quickly swinging it down again) causes a module in performance mode to enter composition mode and "shaking" reverses the effect of the gesture, returning to performance mode (*i.e.,* similar to common "undo" functions).

Communication Channels

Pixels send and receive messages through "communication channels" using a custom messaging protocol. System communication channels are created when devices are powered on. The relationships between pixels are defined by their communications with one another (as well as physical interactions, including those that occur through sensing and actuation). Their communications (and therefore, their relationship) are changeable with gesture. Taken together, channels define the communication network of a system (made with Pixel).

Channels between Pixels can be created and changed with the swing, shake, and tap gestures (**Figure 39**). Setting up I/O channels and placing modules throughout a physical environment provides a template for further refinement to Pixel's behavior (analogous to an approximate outline of an form when drawing).

One builds the high level structure of a system with Pixel by doing the following activities:

1. Connecting and experimenting with digital electronic input and output components to individual modules using the snap connectors.

2. Using gesture to create (swing, then tap to a module), change (swing, then tap to the mapped module, then tap to a module), and remove I/O mappings (swing then tap to a mapped module).
3. Using touch gestures in the graphical programming environment to fine-tune module and system behavior.

The physical infrastructure created through these activities provides a context to continuously change and refine. For example, imagine that you're in a room, and you'd like a button (an input) mounted on a wall in one part of the room to control an LED light set on a desk another part of the room (an output). To do it, one picks up two modules, one per hand, and swings one of the modules, engaging it, as indicated by blinking light, then one taps it to the other module, creating an I/O channel from the engaged module (input) to the other one (output). Now, flipping the switch plugged into the input module turns on the light plugged into the output module. Throwing the input switch plugged into the input module will cause the output component plugged into the output port of the corresponding output module to turn on.

Feedback and Response to Gestures

Modules provide feedback in response to certain interactions with light and sound. Before a module has been connected to any others, switching its input will activate its output. This is indicated by a single flash (off then on) of the light built into the module. If it has been connected to another module, it blinks twice upon

being switched.

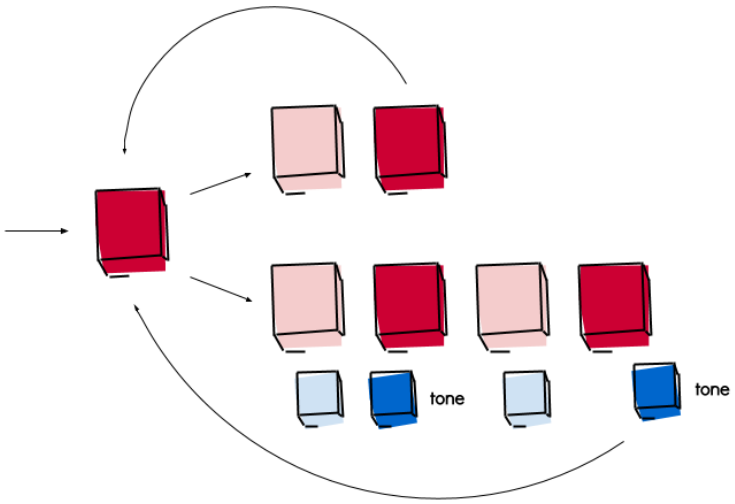


Figure 41: Illustration depicting the feedback behavior when the input switch is closed for the default behavior (top) and when a module's input maps to a different output module (bottom).

The connections and mappings between modules are represented visually by their color and light properties. Connected modules are shown in the same color. By default, modules are set to white. This means that modules are aware of one another but not yet modified. White was chosen because it is the color of a "blank" sheet of paper or an unpainted canvas prepared with gesso. When a module is waiting to be mapped to another module, it blinks repeatedly for several seconds. If it is not mapped to another module in this time, it stops blinking. The input to output mappings between associated modules are represented by a unique color. The light on the output side of the "input module" is set to the same color as the input side of the "output module."

Once an input switch is closed, the module will blink once if the module's input maps to its own output port. (No sound is made.) The module will blink twice if

its input maps to the output port of another module. The remote module will also generate two audible tones. The perception of sound on a module is intended to suggest "the output module is elsewhere" with respect to the module where the input was activated. This suggestion is reinforced by the correspondence in frequency of light blinks (on the held module) and tones played (on the remote module).

Defining Per-Pixel Behavior: Touch and the Graphical Interface

Users can directly specify the actions of individual Pixels with the graphical programming environment. Users interact with the GPE through touches, holding, and dragging (with a finger) along the surface of a mobile, touch-screen display. The GPE is shown in the figure below.

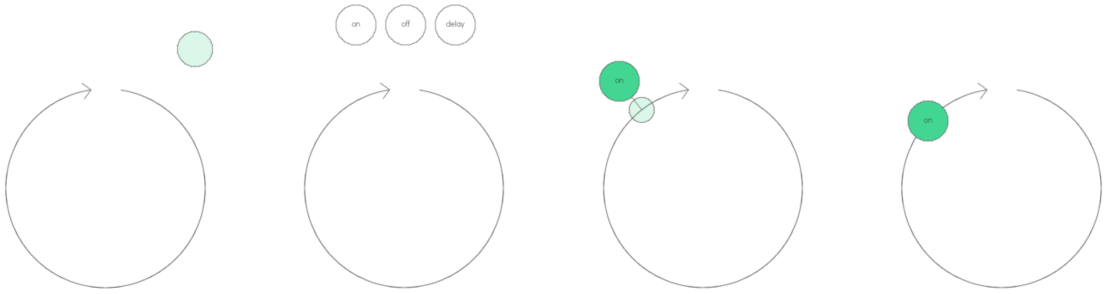


Figure 42: Four screenshots (not drawings) showing four separate states of Pixel's interface.

The behavior of each Pixel is represented in the GPE as a series of actions on a circular "loop" construct. This sequence is stored in each module's random access memory (RAM), and can be changed dynamically to affect Pixel behavior in real time. The loop is directed clockwise, as indicated by the arrow positioned near the

"end" point on the circular arc. This arrow indicates the order in actions will be performed by the corresponding Pixel.

To create actions, a user touches at any single point on the screen. At the moment of touch, one of two actions could be taken. The user could lift the finger from the screen just after touching it, resulting in a quick tap, to create a placeholder for an action, represented as a circle (Figure 11a). Alternatively, the user could continue touching the screen, summoning a palette of potential actions that the module could do (Figure 11b). Touching an action on the palette selects it, hiding all actions in the palette except the selected one. Similarly, to select an action for a placeholder, a user would tap the placeholder, summoning the action palette, and then touch a chosen action.

The loop drawn on the screen is a placeholder for a *sequence* of actions (see Figure 12). Actions can be dropped on and taken off the loop by dragging and dropping them, adding or removing it from the module's behavior. The actions on the loop are called *engaged actions*, and are performed by the module on its next performance (or iteration). The actions off the loop are called *disengaged actions*. Tapping a disengaged action causes Pixel to perform it spontaneously.

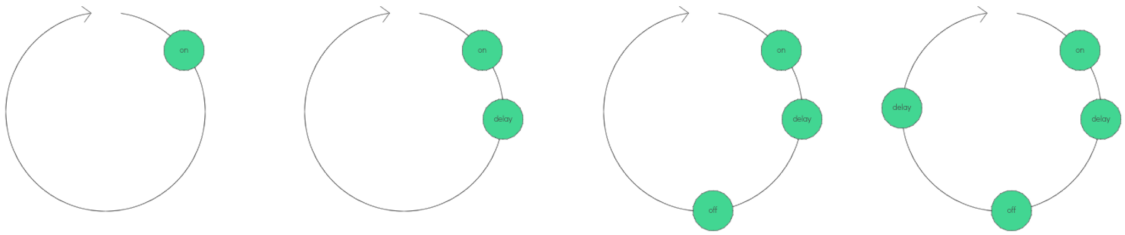


Figure 43: Four screenshots showing a four-action sequence for turning a connected light emitting diode on for one second then off for one second. This behavior, equivalent to Arduino's "Blink" example, was created leisurely in under ten seconds on a Motorola Moto X (2nd Generation) mobile phone using one hand.

By focusing on transformations of a module's behavior that can be done through simple gestures, this interaction design aims to give nontechnical users an alternative experience to that of traditional programming languages. This design is based on the "gentle cause and great effect" relationship between a sculptor's touch on wet clay and the resulting transformation of form while it spins on a potter's wheel.

To facilitate behavior transformation with the graphical interface, Pixel carries out the actions on the loop on behalf of the user. Pixel handles interactions that take place through the performance of simple gestures and the direct manipulation of visual constructs corresponding to behaviors.

Sensing and Actuation: Manipulation and the Peripheral Interface

Digital electronic components can be connected to Pixel's modules. Each module has a single input port and a single output port. Electronic sensors and

actuators, such as a direct current motor or light emitting diode, can be connected to the ports.

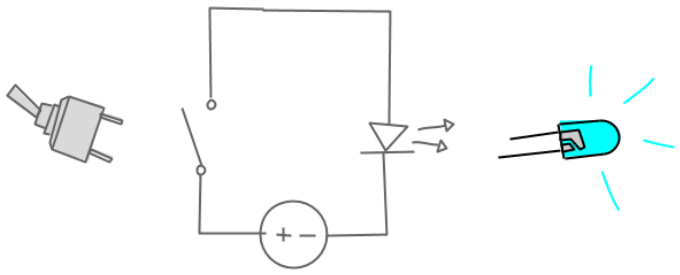


Figure 44: Conceptual drawing of the default "switch" behavior of a module represented as a switch circuit.

In addition, one can simply touch one's fingers on the input port to activate it. For example, one can pick up any modules, connect an output component using alligator clips, and touch the input port to "test" the component. This was done to make it easy to quickly test outputs on a module (or connected module).

Hardware Implementation

The electronic circuitry enclosed in each module is organized into five layers. These layers, dubbed "order," "life," "movement," "stimulus," and "connection" correspond to subsystems for computing, power, orientation and gesture sensing, light and sound actuation, and communications (**Figure 45**).

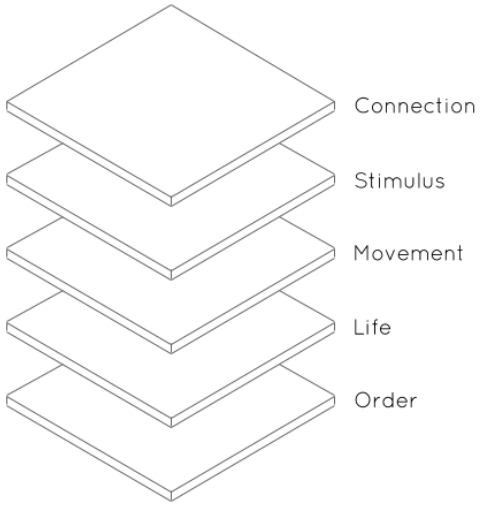


Figure 45: Drawing of the printed circuit board and subsystem organization.

The main electronic components are a 10-degree of freedom (10 DOF) inertial measurement unit (IMU), an IEEE802.15.4 RF mesh networking modules, an IEEE 802.11bg Wi-Fi, two RGB LEDs, a speaker, and two microprocessors. Two pins on one of the microprocessors are used in the input and output port interfaces.

Electronic Components Per Pixel

Order: The "order" layer contains processing components. Pixel performs computations using two Teensy 3.1 microprocessors. The major processing operations are associated with recognizing and responding to user interactions, measurements performed with the IMU sensors, estimating orientation, handling communications from other modules and devices, running a module simulator, and programming the simulator in response to user interactions.

Life: The "life" layer contains the power circuitry. Each Pixel module is powered by two (rechargeable) 3.7 V 500 mAh lithium-ion polymer (LiPo) batteries.

This layer includes two Japan Solderless Terminal PH series (JST-PH) connectors for connecting the LiPo batteries to the layered circuitry, a single pole, double throw (SPDT) slide switch for toggling power, and 3.3V and 5V voltage regulators.

Movement: The 10 degree of freedom (DOF) inertial measurement unit (IMU) on the "movement" layer measures acceleration, rotational, magnetic, and pressure and store them in memory. The data are transferred to the microprocessors on the "order" layer and used to estimate the orientation of each module. The data are incorporated into a direct cosine matrix (DCM) used to calculate the relative orientation of each module, relative to the global coordinate system (or inertial frame of reference) provided by the Earth (*i.e.,* the celestial sphere). This enables estimation of orientation including roll, pitch, yaw, and heading (*i.e.,* direction).

Stimulus: Visual and tonal audio components are included on the "stimulus" layer. Visual feedback is generated by two WS2811 multi-color light-emitting diodes (LEDs) that emit red, green, and blue (RGB) light. Tones are produced by a CEM-1203(42) piezoelectric buzzer rated at 2.048 kHz, capable of producing audible tones around that frequency, with a sound output of about 95 dB.

Connection: The "connection" layer includes radio frequency (RF) communications components. The Synapse RF266PC1 running the IEEE 802.15.4 protocol is embedded in each module for peer-to-peer communications. The device is also used to measure received signal strength indicator (RSSI) for coarsely estimating the relative distance separating modules, providing coarse system-level orientation. The Texas Instruments CC3000 wireless network processor enables each module to

connect to a wireless local area network (WLAN) in the IEEE 802.11bg Wi-Fi frequency band (2.4 to 5 GHz).

Software Implementation

Firmware

Generally speaking, Pixel's firmware performs computations for taking measurements with the IMU sensors, estimating orientation and movement, recognizing and responding to gestural, graphical, and textual interactions, communicating with other modules and devices, simulating itself, and programming the simulated device on the behalf of users or other modules.

The firmware was architected as a simple distributed operating system that creates and maintains a distributed simulation of each module, communicates with other modules to update the simulation to reflect the most recent state of modules, and handles interactions with users.

Gesture Builder

Gesture Builder was designed for use in defining custom gestures

Figure 46. Gesture Builder collects data from the inertial measurement unit, and with it, constructs a model used to later identify the gesture.

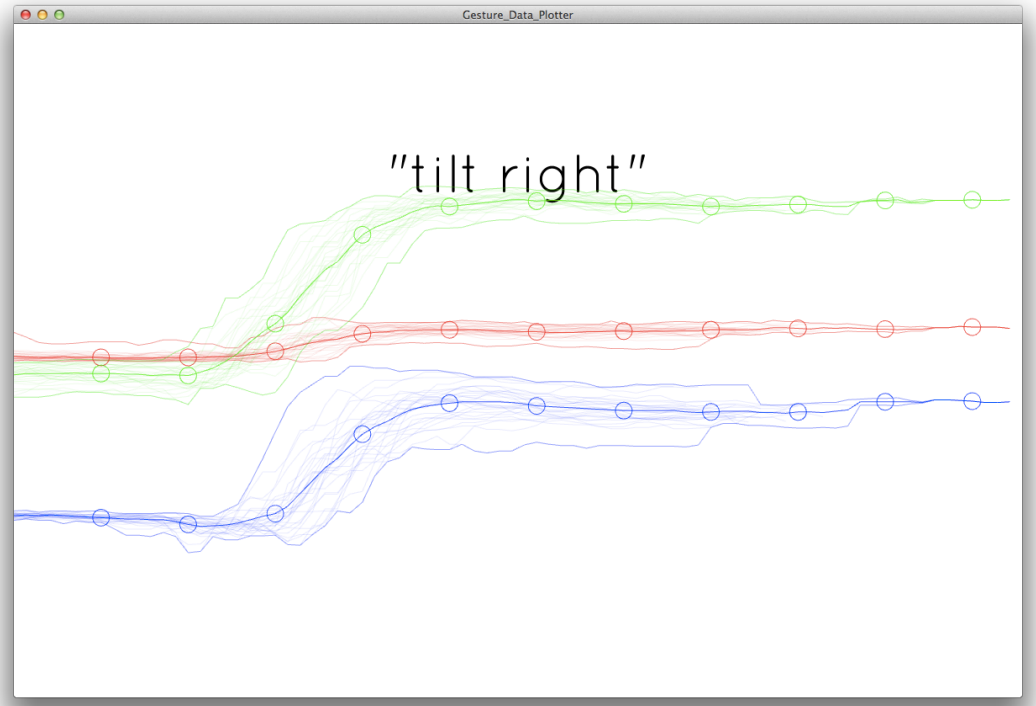


Figure 46: Screenshot of Gesture Builder. Gesture Builder is the tool used to create gesture models.

Presently, Gesture Builder can only be used on a traditional desktop or laptop computer. However, it could be integrated into Pixels with relative ease to enable the creation of custom gestures. This is left for future work.

Chapter 5: Prototype Evaluation and Critique

This chapter describes evaluations of Pixel and the results of each, illustrates examples of using Pixel offered by participants during evaluation sessions, and describes participants' interactions with the Pixel prototypes. These are followed by a summary of example usage scenarios envisioned by evaluation participants.

Evaluation Methods

Two types of evaluations were conducted. First, to better understand how children (and parents) approach and use Pixel to design interactive systems, three workshops were conducted at KID Museum, a children's museum in Bethesda, Maryland. To gain another perspective, a focus group with FutureMakers coaches, educators that deliver a variety of hands-on workshops to children to teach them crafts including sewing, woodworking, robotics, computer programming, and circuitry[7]. The two studies were intended to be complementary. Study 1 allowed me

[7] A third session, more in-depth classroom evaluation was organized together with FutureMakers. Unfortunately, the session needed be cancelled due to a concern about the study raised by a parent, received the day prior to the evaluation. In these classes, participants would have used Pixel to build custom musical instruments that respond to gestural and physical interaction and operate based on custom programmed behavior. This type of field deployment is important and would enable us to further assess how novices can approach and use Pixel and what they build with it, but it is left for future work.

to observe the use of Pixel in a facilitated context while Study 2 provided insights from experienced coaches familiar with teaching prototyping and craft skills.

Workshops at KID Museum

Overview

Workshop evaluations lasted approximately 60 minutes and consisted of four parts: *(i)* a brief introduction to myself and the session, *(ii)* personal introductions by participants and brief conversations about what kinds of things they like to build, *(iii)* a brief description and demo of Pixel, then *(iv)* open play with Pixel coupled with an free-form discussion about its design and capabilities as a system for creative expression. These evaluations were conducted on Saturday, November 8, 2014 at KID Museum in Bethesda, Maryland. The sessions were run as one-hour workshops at KID Museum. In consideration of the limited time, the workshops omitted formal interactions with participants, including written questionnaires. Instead, the full duration was dedicated to discussion about Pixel, anticipating it would provide an overall richer collection of qualitative data.

Figure 47: Workshop evaluation at KID Museum.

Participants

The three workshop sessions were advertised by KID Museum as "Invention Workshops" on its website as well as social media, including Facebook and Twitter. Workshops participants chose to sign up for one of the three sessions through an online system operated by KID Museum. Each workshop was limited to eight participants, of at least eleven years in age. Most participants in the workshops were middle school students. Some participants were accompanied by their parents or grandparents, who were invited to participate.

At the outset of each session, I asked participants to introduce themselves and answer the question *"What kinds of stuff do you like to make?"* Responses varied widely from building "aerodynamic things" and assembling decks for Magic the Gathering, to making YouTube videos and taking apart old toys that don't work

anymore then putting them back together. These diverse responses help demonstrate the varied skills of participants and provided useful context for running the workshops.

Evaluations

In each session, participants were encouraged to play with the Pixel prototypes. I provided alligator clips as well as electronic inputs and outputs mounted on wood boards borrowed from the museum's Circuit Boards activity, designed to be used with alligator clips. The available input components were a light switch and the available outputs were a red and a white LED.

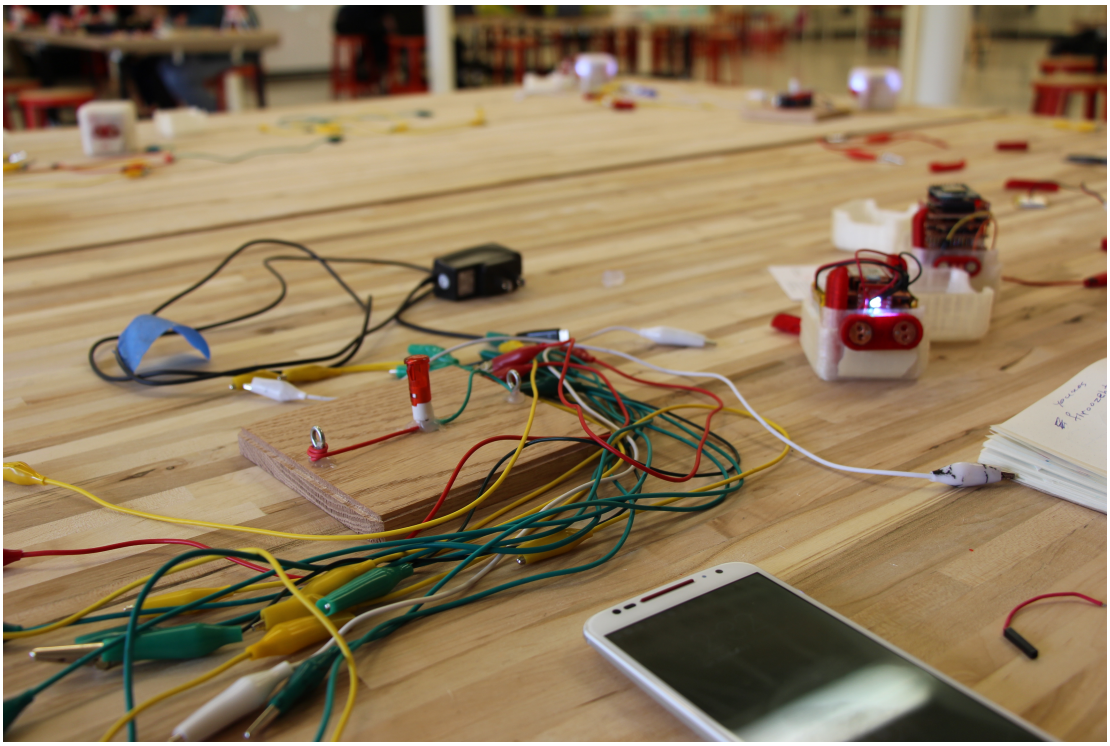


Figure 48: Photo showing the red LED "circuit board" after a workshop session.

Using these components, participants could connect components to the input and output ports on Pixel modules using alligator clips. Additional input and output

components were available, but were not used because the sessions were only an hour long, leaving little time actually build artifacts. During workshops, I observed the participants' interactions with prototypes, but most of the time was spent discussing of the design of prototypes and envisioned usage scenarios.

Feedback and Observations

In my evaluations, I observed participants' basic physical interactions with Pixel module attempting to identify any physical difficulties connecting peripherals using the magnetic adapter or gesturing to link modules, or conceptual challenges in understanding how one module's input could be made to activate another module's output through gesture. In addition to evaluating basic interactions, I was interested in how participants could envision using Pixel to assemble their own systems and specify their behavior for their own purposes, playful or practical.

In addition, participants provided feedback about *(1)* the current prototype design, *(2)* envisioned applications, and *(3)* design suggestions for future prototypes. Below, I summarize observations and feedback, discuss several specific scenarios in which they envisioned using Pixel, and detail three of the scenarios that were described very clearly by participants, suggesting an accurate understanding of Pixel, despite their inability to actually build the systems during the workshops.

Interactions with Pixel

Several aspects of Pixel's design were evaluated. These are given below along with feedback and observations.

Together with the participants, I evaluated *(1)* connecting input and output components to modules, *(2)* connecting modules using the gestural language, and *(3)*

interacting with modules using the graphical programming language. To evaluate these experiences by playing with the provided components and ten prototype modules. The graphical programming language was shown on an iPad Mini then made available to participants for play.

Connecting input and output peripherals to module with alligator clips: Participants did not appear to have any difficulty using alligator clips to connecting input or output peripherals to the magnetic adapters. They appeared to find attaching the magnetic adapter to Pixel modules easy, too.

Connecting modules to one another with gesture: Participants appeared to understand the gestural language after a single demonstration. They were able to reproduce the demonstrated gestures, namely swing, tap, and shake, having no apparent difficulty in doing so.

While no feedback was offered about the selection of gestures for connecting modules to one another, some participants indicated that the tap gesture felt unresponsive. Making modules responsive to gestures was a major focus in developing Pixel. I looked into the cause of the unresponsiveness following the evaluations. It appeared that the unresponsiveness was due to a disparity between the movement sensor and gesture model produced a technical limitation has since been resolved. The gesture model has now been fixed.

Participants also suggested ways to extend the gestural language. One participant wanted the ability to add custom gestures.

General Purpose Making

I described Pixel to participants as a tool designed for constructing interactive systems and setting up their behavior. Participants seemed to understand that Pixel was being designed as a general-purpose making tool. In particular, one participant described it as *"one product that can do multiple things."* Likewise, another offered that Pixel is *"multiple things you can program all in one thing."* One young boy consistently used the word *"modes,"* saying he wanted a mode to use Pixel as a flashlight, for personal storage, and to use as an input device for gaming, suggesting conceptualization of Pixel as somewhat flexible, if not general purpose.

One adult participant expressed interest in using Pixel to fashion systems in a variety of different situations. He said, *"I usually come up with all these kinds of ideas but I don't have the knowhow or anything."* He continued, *"A lot of people would want such a thing, but need to figure out which kinds of sensors can be used."* This seems to show willingness to use a device like Pixel to address needs creatively, and provides hints to a reluctance to do so because of the technical challenges involved.

Module Design

Uniform design: Multiple participants noticed that the prototype modules and their internal circuits were designed uniformly. They encouraged me to move forward with this approach, but suggested designing different kinds of cases while maintaining a uniform design for the internal circuit, so it can be fit into different cases.

Participants suggested several ways that they wanted to change Pixel in order to make using it more personal.

Opaque Color and Indicator LEDs: One participant asserted that Pixel should have an opaque case that has visible lights only at specific locations on the module.

Size: The groups seemed to like the size of the modules. One participant said he like *"[how] small it is."* Some suggested that it should be made *"a little smaller,"* so it could fit into one's pocket. The first group suggested that it should be able to fit in one's pocket and fit comfortably in hand.

Cases: In every session, participants suggested making multiple cases tailored to different kinds of use and for personalization. *"So there should be different cases, but you should be able to take out the internal hardware, and put it into a new case easily."* He continues, *"So, like, easily put it in new cases, like for different environments—waterproof, climbing, let's say it falls off or something like that—and since it's modular, you should also be able to swap out the hardware easily and install software easily."*

One participant thought that cases should be able to be changed more easily. *"If you take all this hardware inside and put it into a case then put that into this case, that way if they're switching cases, they don't have to worry about accidentally pulling out one of these wires, or accidentally bending one of these things, and not even knowing what's happening inside, so they're 'just taking it apart'."*

Robustness: Some participants felt the prototype case was too brittle. They said that it should withstand many drops. They also suggested insulated and waterproof designs, as mentioned previously.

Accessories: One participant wanted to connect a wrist strap to modules (similar to that used by the Wiimote) so it could recognize swinging gestures. Others suggested that this would also be helpful for preventing modules from being thrown accidentally.

Envisioned Usage Scenarios

Participants envisioned several specific applications of Pixel. Most of these concerned remotely controlling systems, monitoring environments or behaviors, responsive or intelligent environments, agriculture, health, personal fitness, or freeform making. Again, while participants were not able to create these systems during the one-hour workshop, the descriptions help reveal how Pixel was being conceptualized.

Specific Example 1: "Scarecrow Tree"

In the second session, one of the participants described an actual problem he had experienced for multiple years and explained how he would solve it with Pixel. Note this is the same "scarecrow tree" situation described in Chapter 1. It is provided again here verbatim for the sake of readability of this subsection.

"Here's a problem I have. I have a cherry tree. Just when the cherries get real ripe, the birds come and eat them. Now, if I can make some sounds or some kind of flashes or something—a scarecrow, right?—then the birds will not come. Now even a scarecrow that you have has to have moving parts on it or they say you can buy a

plastic owl and put it somewhere, but if the plastic owl is not moving at all then it won't work. The birds will learn and it's useless. So, if one had these kinds of things, and one of them has a motion detector, gets a motion from the birds around or something, then it can signal the other ones which would be on several branches in the tree. Something like that."

Approach: I discussed the following approach that could be used to implement this solution with the participant: *(1)* start with problem, *(2)* set up general structure with modules (i.e., connect an input with a sensor and put it on a branch and repeat this step, connect to outputs with motors on them and flashing lights, and *(3)* program specific behavior (sensor sensitivity) using the graphical programming enviornment.

Figure 49: Conceptual drawing of the scenario described by the participant with five modules, with two sensors (depicted as cameras), a light (depicted as a light bulb), a speaker (depicted as such), and two additional modules to produce additional sound and light.

Limitations: There are some limitations of the prototypes that would prevent it from being used in the way envisioned by the participant. Another participant noticed the most inhibiting limitation; the current prototypes only support digital

input and output, preventing the use of analog sensors and actuators. Support for analog sensors and controlling motors is needed to realize the interactive scarecrow as described by the participant.

Variations: The group suggested some variations on this example. For example, another participant commented that he could use an "ultrasonic device" (perhaps an ultrasonic waveform generator), to fend off squirrels as well as birds. Another suggestion was to record the sound made by a cat for playback when a squirrel was detected to scare the squirrels.

Specific Example 2: Autonomous Garden Robot

"You have a tiny robot that just moves in the path in between two lines of plants. It's going and it's looking around. It's going and it sees these worms, or an insect or something. Two ways. One, it has a laser connected to it and can zap it. It just goes through and finds them and zaps them. These robots would just walk through the different lines. It doesn't have to be so sophisticated."

"There is a lot of technology that is possible and available, it just needs people to think about them, and see how it can be used."

Overall, participants in these evaluations seemed to have a positive experience. Virtually all participants experimented with Pixel prototypes and were actively and voluntarily engaging in the discussions.

Discussion with FutureMakers Coaches

Overview

The semi-structured focus group about Pixel was conducted with four "coaches" that facilitate workshops for FutureMakers, an organization that teaches making-oriented workshops. These were conducted in Baltimore, Maryland at the FutureMakers office. This session consisted of *(i)* a brief demo followed by *(ii)* an informal conversation focused on how Pixel might fit in the workshops facilitated by FutureMakers. We also discussed the general design of Pixel and envisioned usage scenarios as they arose. This session was about one hour long and took place on November 10, 2014.

Our discussion focused on Pixel in the context of coaching, education, and learning. We touched on its current possibilities, future possibilities, and aspects of gesture in and beyond learning contexts. Most of our discussion was focused on gestural interaction. We covered gestural interaction, roles it can play in learning, and the ideas of movement and mobility as they relate to gesture.

Participants

Four coaches from FutureMakers participated in the discussion. Each had experience designing, leading, and otherwise facilitating workshops and a background in education.

Session and Analysis

I began the session by providing an overview of Pixel. I introduced myself, describing my experiences facilitating workshops and museum activities, such as programming, demonstrated Pixel, and then facilitated an informal, focus group discussion of Pixel. Our discussion is summarized below. Quotations *emphasize* fragments to call attention to recurring themes and highlight thoughts that seem especially important for Pixel. I discuss these emphasized points in the next chapter.

Results

During the discussion, one participant captures the essential topics of the discussion—physical movement, getting away from screens, and learning computational thinking—in saying, *"As coaches and educators, I would be really curious to see how we can start supporting the foundations of programming and computational thinking using objects, getting completely off of screens."* The participant continued, *"I would like to see how low on the age spectrum we can go in order to get really meaningful learning going on."* The participant continued to express interest not just "playing Simon" but making gestures and making Simon.

The participant expressed dissatisfaction with present physical computing systems, *"I would really like physical computing to stop being, sort of, just, making with certain things, and [start being] moving them in space, because that's what we do. We're gesturing the whole time we're here and we don't have to make physical contact with each other to be able to read those signs. We have sight, and we can hear, and we can sense, you know, we've got those things, and I'd love to see, especially with your product, that kids can move around."* Then, *"I mean, it is… until*

high school when they've beaten it all out of you… you still need to move. I would hope that there could be a much more, sort of, moving around, type option. I've been looking at a lot of products that are, 'take your device and make something do something' and I've seen very few things where it's like 'interact with this device—you do something—and something else does something' and you program something by doing things that you do."

Below, I synthesize the four main themes that emerged from our discussion: *(1)* computational thinking, *(2)* movement and gesture, *(3)* bodily feedback, and *(4)* kinesthetic performance and learning.

Computational Thinking through Physical Movement

Participants thought that even basic "if this then that" programming based on custom gestures and gestural interaction would be very valuable. *"Just something like, if I do this, this will happen type stuff. Very, very basic. The fact that it can be a physical action is huge, because we've dabbled a bit with these that are, in the physical sense, taken away from the actual screen, but still very code-like in that it's arrows or things like that. The fact that this would be a physical interaction would open it up more."*

The participant continued to envision one way to use Pixel, offering *"If there were two kids standing next to each other, and like, the output of [one module] was controlling [another module] so that every time [one kid] did a jumping jack, the other module made the sound of a jumping jack, then if the other kid knew that was a jumping jack, he could do a jumping jack, and then pass it on or something."*

Movement and Gesture

Movement was perhaps the topic discussed most. We covered gestural interaction, roles it can play in learning, and the ideas of movement and mobility as they relate to gesture.

Generally, custom gestures seemed to be easily understood. Participants grasped the idea of creating a custom gesture through demonstration. After explaining that custom gestures could be made since they're defined by *"motion data,"* participants indicated they understood. To verify her understanding, one participant asked, *"Oh, so, you could do a circle, and then you could look at the data that it recorded for the circle, and you could say, now, this circle, or within these parameters, this is done again."* Likewise, another participant indicated her understanding, *"So you can program a gesture."*

One imagined using it, saying, *"I'm going to make the letter A."* Another said, referring to using gesture programming to control sound, *"That could sound so amazing. You could perform music in such an amazing way."* He continued, *"I want kids to move things through space and generate sound. That blows my mind. That would be so amazing. We could create compositions, get a team of kids together to actually create a composition with gesture in space."* Another participant built on that, *"You could have one on a right foot, one of a left foot, and then you could have a dance or a pattern that they were all [doing]."*

They were really interested in ways Pixel could encourage movement or support kinesthetic learning. One participant concisely described the role of gesture in interacting with Pixel, *"It's coding at a human level."* The participant continued,

expressing general interest in the idea of using movement to express ideas, *"You know, I think it's good because it would get people moving."*

"You could get kid who are kinesthetic learners to learn about something like programming from moving." Another added, *"Coding with their body."* Similarly, one participant wanted to be able to see how the gesture was made. *"You can do a gesture and then the computer visualizes it. This is how the Pixel reads your gesture."*

They think it would be helpful to show users how a gesture was represented as data in graph form while making custom gestures and show them different properties of the graph data to help them understand the gesture and movement. For example, the acceleration could be shown to help them understand how acceleration factors into the performed gesture. Upon reflection, it seems that it may also be useful to add tools for fine-tuning gestures graphically.

They suggested that a visualization of gesture data would also provide a way for people to see the correspondence between the life data and a particular recorded gesture, to help them practice the gesture. That is, it could be used as a feedback tool for learning how to perform gestures (*e.g.,* for playing performance).

Bodily Feedback and Kinesthetic Performance

While considering Pixel as a tool for training and teaching, one participant enthusiastically pointed out that Pixel could also be used as a general *"feedback tool for your body,"* beyond the contexts of learning and making objects. Even in these broader applications, they discussed some of the same underlying ideas, including being able to move away from screens, and moving throughout physical space.

The participants were interested in applications in kinesthetic performance. *"I would love to create a gesture and see someone repeat my gesture. I think that playing that game and seeing if I was actually able to 'be them', you know… I think dance, it definitely has strong connections to dance."* Another participant made a similar suggestion, offering that devices like Pixel could be used to build a wearable kinesthetic sensor that would provide feedback about movement as well as haptic or visual feedback to help correct movements during training. They suggested that a soccer player could use it to learn how to train while learning to kick a ball.

Critical Feedback and Suggestions for Future Prototypes

Near the end of the discussion, I encouraged participants to provide critical feedback, suggestions for future prototypes, and any other examples envisioned but not discussed.

Participants envisioned using Pixel outside, as is conveyed in some of their comments above. They wanted the case to be robust enough to *"dunk in water"* and *"throw against the wall."*

Participants seemed generally satisfied with the size of the case. I asked two of the participants about the case size directly. One suggested, despite seeming satisfied, offered that it could possibly be a little bit smaller, *"If anything, I would just say it should be slightly smaller. I like that it fits in the palm of your hand."*

One participant suggested thought that it would be useful in some cases if modules could actuate themselves or actuate another module. Relatedly, some participants suggested adding haptic feedback to the modules, in addition to light and sound. One participant thought haptic feedback could provide way to provide

feedback to individuals with *"low vision and no vision."* They thought that additional feedback other than haptic feedback could also be incorporated, but didn't offer specific suggestions.

Chapter 6: Discussion, Summary, Future Work

The aim of this thesis was to better understand design techniques useful for designing creative tools that support the creation of information systems for application in everyday situations. Based on observations of challenges encountered by children, hackers, and artists in making these systems, I developed a tool called Pixel to explore a unique synthesis of design techniques explored separately in the literature, namely modular system design, gestural programming, and graphical programming. Based on evaluations, I suggest that together, these techniques offer affordances that are particularly suitable for designing tools that support the everyday making of systems.

Specifically, I suggest:

1. *Combining a modular design with direct gesturing afford greater freedom in using physical space, exploration, embedding.* I suggest this set of techniques provides affordances that can make building systems for everyday application more feasible.
2. *Simple gestures are sufficient for expressing a number of relationships topologies that connect elements of a modular system.* This builds on the gesturing approaches of Montemayor (Montemayor, 2003) and Chung (Chung, 2010).
3. *Graphical programming environments for mobile touchscreen devices provide a way to interact with the programmable elements of modular systems that are not directly accessible.*

4. Support for quickly connecting, disconnecting, and changing sensors and actuators connected to modules, afford exploration.

I believe this combination provides powerful affordances that can enhance the feasibility of creating custom systems in everyday life situations. Below, I discuss the advantages and limitations of each of these, their potential implications for makers and researchers, and finally, suggest some directions for future research.

Discussion

Overall, the level of enthusiasm expressed by evaluation participants was unexpectedly positive and helps underscore the promise of Pixel. Very soon after demonstrating Pixel, participants began to play with modules and describe scenarios in which they envisioned using it. Their suggestions were diverse, indicating they understood Pixel to be a general-purpose tool. Moreover, participants envisioned creating systems inspired by their own experiences that would be personally meaningful, such as the "scarecrow tree" scenario. This indicates that participants conceived of creating systems that are personally meaningful. However, participants expressed that they could not create the envisioned systems due to limitations in their "technical know-how." Despite their limited technical ability, it seemed that most participants could not only envision interactive systems but also describe their key features, suggesting that, if empowered with enabling tools, they would create systems to resolve everyday situations. Taken together, I believe these outcomes suggest that Pixel represents a viable approach to designing support tools for interactive systems in everyday situations.

Reflections and Insights

It seems that the positive response of participants resulted largely from a few design features of Pixel.

Both museum and focus group participants quickly learned the set of gestures recognized by Pixel. After a single demonstration, participants started to do the gestures themselves to connect and disconnect modules, suggesting the set of gestures chosen was simple to learn. Similar results were achieved by Montemayor with his physical programming system that could define relationships between sensor and actuators to support immersive storytelling environments for children (2003) and Chung with OnObject in defining gestural interfaces to everyday objects (2010). Both of their systems support gesture in a tool external to the systems built with them—a "magic wand" (along with a "wizard hat") and wearable ring device, respectively.

In contrast to these, Pixel supports direct gesturing with modules. This style of direct gesturing is analogous to direct manipulation of graphical objects displayed on a touchscreen device. Gesturing indirectly with a special tool is analogous to using a mouse to manipulate a cursor on a screen. I believe that directly gesturing with tools provides a more naturalistic everyday interaction because it seems to provide a more viscerally engaging making experience. Moreover, in the case of modular systems such as Pixel, direct gesturing provides natural support for multiple users. Existing systems can be modified directly, without the need to retrieve the tool for modifying it.

Some of the most stimulating outcomes of my research result from the combination of a modular design and the ability to gesture with modules directly.

This combination seems to deliver powerful affordances that fit naturally in everyday life. First, a single module can act as a source in multiple of such one-to-one relationships, one-to-many relationships can also be expressed between one source and more than one destination at (about) the same time. Moreover, these relationships can be defined between any two modules, providing considerable flexibility. In addition to granting expressive power, combining modularity with direct gesturing naturally enables multiple people to use Pixel simultaneously. Finally, supporting direct gesturing with modules seems to give makers more freedom to move around through space and leverage their intuitive sense of the physical environment.

Limitations

Though participants generally expressed enthusiasm, the evaluations of Pixel can only hint at the kinds of systems that might be built with Pixel. This is the case because participants were only given materials for building a remote light switch (in the workshop evaluations). They couldn't create systems of their own design other than various light switches and remote light switches. I felt this was the greatest limitation of the evaluations. This is a clear opportunity for future research. Fortunately, participants were able to demonstrate the ability to use the functions of Pixel by making the light switch variations. However, this provides little insight into important concerns of usefulness and usability of Pixel. While participants couldn't create systems of their own design, they discussed the numerous systems they envisioned creating, providing insight into their conceptualization of Pixel.

There are limitations in the design of Pixel that constrain its utility for making interactive systems. The current module design is quite large at about 2.5 inches in

each dimension. Though the module is quite comfortable to hold at this size, it is too large to embed within many everyday objects (*e.g.,* a remote control car) and transporting more than a few of them (*e.g.,* in a backpack) requires significant space. Reducing the modules module significantly is a feasible technical challenge and could resolve this limitation. I suspect that reducing the module size to one square inch would address these limitations. Moreover, I believe this smaller size would increase the overall utility of Pixel as a tool because the size is easy to manipulate with one's fingers (as opposed to one's whole hand). Likewise, from experimenting with different types of 3D printing materials, I believe a softer outer layer (*e.g.,* as offered by flexible filament or a rubberized material) may feel better in the hand.

Pixel modules can be programmed to receive sensor information, control output devices, and communicate with each other. The gestural language consists of a fixed set of gestures. These gestures seem to effectively support connecting modules into network topologies. The inability to extend the set of gestures, however, eliminates the possibility of leveraging the ability of modules to recognize gestures (and other movements) in custom systems. This limitation was quickly pointed out during the evaluations, but was a central concern of facilitators, who wanted to create gestures for multiple applications. Extending Pixel to support custom gesture creation would be quite easy, and could be accomplished through gesture, similar to OnObject (Chung, 2010), or through extensions to the graphical environment.

The current gesture language can only express a relationship between one source module and one destination module. There are some relationships that cannot be expressed with gesture. Notably, these include one-to-many (where the many

modules activate sequentially) and many-to-one relationships (where multiple source conditions must be simultaneously.

Finally, even though multiple people can simultaneously interact with Pixel, but minimal consideration was given specifically to interfaces to support it.

Implications for Makers

Pixel represents an approach to creating systems that enables systems to be made through interactions and interfaces that engage one's physical senses and leverages the intuitive knowledge of physicality derived from experience. This approach may minimize the role of technical expertise in making systems, and in as a result, broaden the participation in creating systems. Making the creative tools much more accessible to people with no technical background—such as through support for expressive physical actions appropriate for defining aspects of system behavior—could enable the creation of systems in everyday situations.

This approach may affect the creative process of making systems. First, it seems to foster exploration in constructing systems and defining their behavior. The snap connection interface seemed to facilitate quickly connecting, disconnecting, and changing sensors and actuators to modules using alligator clips. The simple set of gestures can foster quickly defining, erasing, and changing relationships between modules. Additionally, the graphical programming environment presents the available system behaviors for selection through touch interactions rather than requiring the manual recall and entry of programming language. Second, Pixel enables freer movement in the physical environment during the process of building systems. Traditionally, makers are confined to the space near a computer to a more physically

expressive. Third, Pixel shifts some common tasks involved in creating systems from being technically demanding. The support for gesture changes the activities involved in creating communication channels between modules from being quite technically demanding to be more playful activities involving experimentation in connecting electronic components and defining relationships between modules.

Implications for Researchers

This thesis explored the potential of tools that empower people with the ability to feasibly create information systems as an everyday activity. Further research into tools to support making everyday systems has implications on the access, use, and impact of information technology. I discuss each of these below.

Previous advancements in the design of tools for creating systems have made them more accessible, often by changing them to hide and manage technical complexity formerly managed by those using the system. Such advancements in accessibility have broadened the range of people to use them. Researching the design of tools specifically for use in everyday life could enable anyone to create systems regardless of technical ability, further broadening participation.

The technologies people use in everyday life are almost always commercial products designed and manufactured by industrial organizations. Tools that make it possible for people to create systems themselves may enhance the utility of systems by allowing them to be made for use in highly specific and personal situations that specialized commercial products cannot address in an economical manner.

Tools that make it possible for everyone to create systems could potentially lead to greater technological independence for individuals that presently rely on commercial technologies.

Lessons Learned from Designing Pixel

I learned a number of lessons while developing Pixel that may be useful for future design and development of tools for creating everyday systems. I discuss these below.

Modularity (Modular System Design): The modular design approach taken for Pixel seems to enable more real-time exploration of system design. Participants pointed out the importance of modularity and flexibility in Pixel's design. Users can more freely move through their physical environment to build and rebuild physical system topology while situated in the final deployment context.

Unit Design (Individual Module Design): Large module size and "poor quality." Both the internal circuitry and the enclosure are quite large. This is largely due to the use of widely available components in the prototypes. The mobile and modular design of the system lend well to a diverse range of use cases.

Module Feedback to Users: The light and tonal feedback provided by the current Pixel design are minimal. While enough for the limited I/O functionality, additional feedback, such as tactile feedback, may be useful. Tactile feedback, like haptic feedback, seems well suited for modular and gestural TUIs used with one's hands. Additionally, real-time interactive feedback could be made available in the graphical environment, providing users with multiple feedback channels.

Peripheral Interface: Pixel supports one input and one output per module. While this is provides simplicity, the limitations of the design quickly become apparent when building single objects that do not extend through space. Consider modifying an RC car. This can require several I/O ports for interfacing with the car's control circuitry. Even though modules connect automatically and I/O channels can be established in a quickly with gesture, using multiple modules is not feasible due to their collective size. To strike a balance between the minimal approach of Pixel and the general approach of the Arduino, both ports on Pixel could be available for use as either input or output. This could result in additional complexity in the graphical interface design. Alternatively, I/O connectors could be designed with additional I/O ports available for use while maintaining a simple interface between the connector and the module. This approach is common in general-purpose I/O expanders that communicate with a host controller (*e.g.,* Pixel) with the I^2C protocol.

Magnetic Snap Connectors: The snap connectors for I/O components were designed so components could be quickly interchanged to enable quick experimentation with different I/O components. This design was made in response to experiences with previous prototypes that embedded the I/O connectors directly into modules. While using alligator clips can make changing components easier, experiences with children show that alligator clips can be difficult open for children and can be frustrating to use when switching between a series of components during experimentation. While connecting the components to different snap connectors makes them available for rapid experimentation with Pixel, the connectors themselves are large for certain use, such as embedding within objects.

Gestural Interaction: Based on workshop evaluations, gesture seems to be a promising approach for setting up communications between separate parts of a single, distributed system. Participants were able to perform the swing, tap, and shake gestures to connect components after a single demonstration. While there are likely additional gestures that could be incorporated into the core set used to connect components, such as for basic configuration of analog components, additional flexibility to create and change gestures *in situ* may provide additional flexibility without loss of simplicity, as suggested by OnObject (*e.g.,* Chung, 2010).

Graphical Programming Environment: The graphical programing environment for changing module behavior is designed to compliment the gestural composition interface for module-module I/O channels. While this enables experimental programming *in situ*, the current graphical environment is quite limited in functionality. Moreover, the functionality of the mobile device displaying the graphical environment is not accounted for by Pixel, despite the possibility of voice and video interaction that might be provided.

Summary

Creating interactive systems that interact with the environment is a complex creative process that draws upon the tools, materials, and techniques from a range of technical domains. The purpose of this thesis was to investigate tool designs that can support more holistic engagement with the physical environment and materials in physical computing. To contextualize this investigation, a tool was prototyped iteratively over the course of about eighteen months while facilitating children in

museum exhibits and workshops focused on making, participating in a series of hackathons, and joining student hackers and artists in creative projects.

Preliminary design evaluations of Pixel were conducted in a series of three workshops with children at a children's museum and in a focus group discussion with educators working in a local organization that leads making-oriented workshops for young people.

Future Work

Pixel was an exploration in building a physical computing platform that offered more complete support for its constituent activities, specifically programming, circuit building, and working with physical materials. There are several opportunities for directly extending Pixel, and many directions for design research into more embodied approaches to physical computing.

One needed extension to this work is support for a broader range of input and output components, including analog sensors and analog output components. As part of this, one could experiment with gestures or gestural interaction designs to make connecting and exploring sensor and actuator configurations. Likewise, graphical representations could be investigated for extending the graphical programming environment to support the additional sensing and actuation capabilities.

The module case design could be modified to better support diverse use. As suggested by evaluation participants, the cases could be made more robust, specifically to support outdoor use. The enclosure could be made weatherproof and submersible. One approach would be to make modules with adjustable sizes. Another possibility is creating multiple interchangeable cases.

To afford use in a wider variety of situations, more materials could be explored to provide, for example, a soft exterior, a coating that allowed the module to bounce, or modules designs that allow it to be used with construction kits, similar to the Free Universal Construction Kit ("Free Universal Construction Kit," 2012). In some situations, but not all, a module of a different size may be desirable. In particular, the device could be made smaller. Finally, as suggested by a participant in a museum workshop, exploring module accessories such as wrist straps (so the module hang from one's wrist) could be explored.

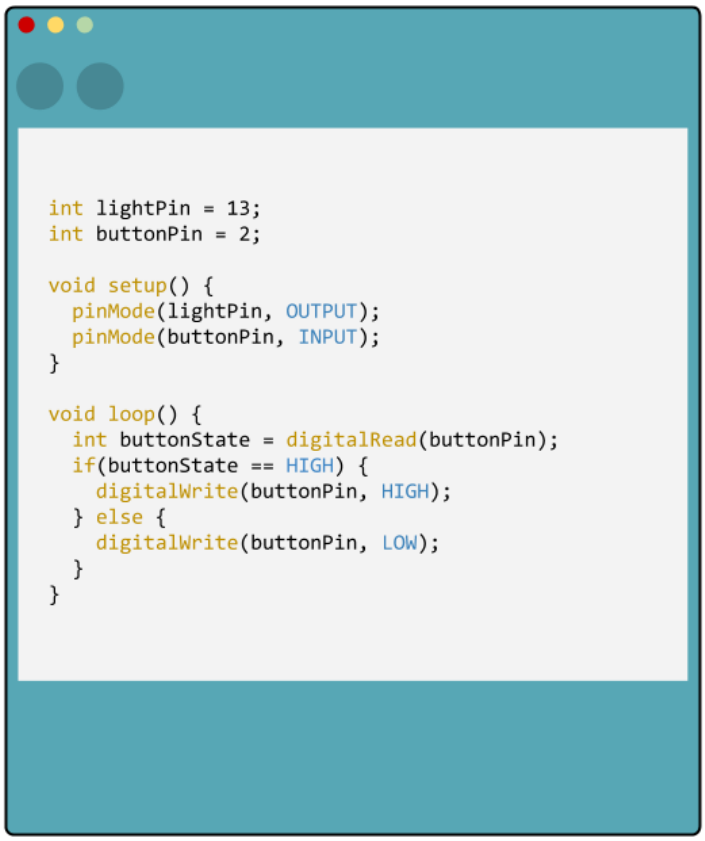
Pixel should support collaborative behavior design and sharing of designs. The experience should be easy and seamless. Pixel should handle all technical aspects of communicating with other modules and incorporate corresponding conversational interactions and facilitation into the implemented interaction model.

The gestural and graphical interfaces presented could be augmented with voice interaction to further explore embodied interaction designs for physical computing tools. Voice interaction may be particularly useful along with gestural interaction for supporting analog input and output components. Such components have signals that cover a range of continuous values with which it is natural to want to associate a point in the continuous range of output with a particular range of input. To support voice interaction, a better speaker should be added to each module in addition to at least one microphone (per module).

Of course, more comprehensive evaluations may provide deeper insight into, for example, gestural interaction with modular tangible interfaces for connecting input and output components. To develop understanding into the range of applications

for which the modular design is appropriate, evaluations could be conducted in a variety of contexts, including museums and workshops, with groups of different ages.

Appendix A: Supplementary Figures



```
int lightPin = 13;
int buttonPin = 2;

void setup() {
  pinMode(lightPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  int buttonState = digitalRead(buttonPin);
  if(buttonState == HIGH) {
    digitalWrite(buttonPin, HIGH);
  } else {
    digitalWrite(buttonPin, LOW);
  }
}
```

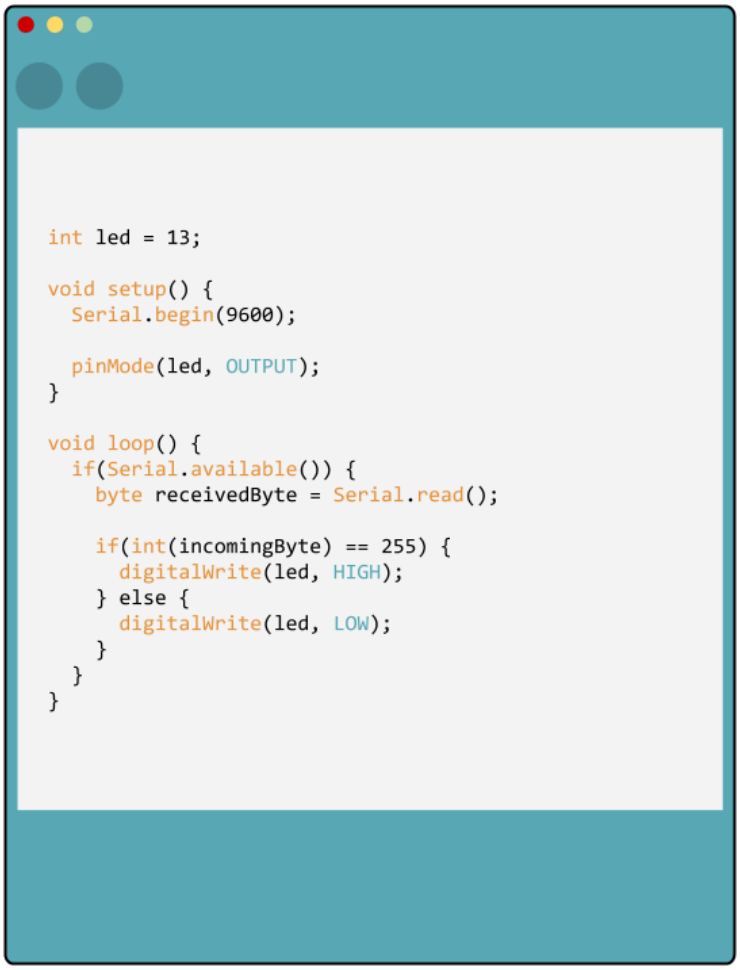
Figure 50: Drawing of the Arduino program for the light switch in Example 1, Part 1. This corresponds to Figure 6.

```
int pushButton = 2;

void setup() {
  Serial.begin(9600);
}

void loop() {
  int buttonState = digitalRead(pushButton);
  if(buttonState == HIGH) {
    Serial.print(255, BYTE);
  } else {
    Serial.print(0, BYTE);
  }
}
```

Figure 51: Drawing of the first Arduino program for the remote light switch in Example 2, Part 1. This corresponds to Figure 13.



```
int led = 13;

void setup() {
  Serial.begin(9600);

  pinMode(led, OUTPUT);
}

void loop() {
  if(Serial.available()) {
    byte receivedByte = Serial.read();

    if(int(incomingByte) == 255) {
      digitalWrite(led, HIGH);
    } else {
      digitalWrite(led, LOW);
    }
  }
}
```

Figure 52: Drawing of the second Arduino program for the remote light switch in Example 2, Part 1. This corresponds to Figure 13.

```
int led = 13;
int speaker = 9;

int noteFrequency = 523;
int noteDuration = 250;

void setup() {
  Serial.begin(9600);

  pinMode(led, OUTPUT);
  pinMode(speaker, OUTPUT);
}

void loop() {
  if(Serial.available()) {
    byte receivedByte = Serial.read();

    if(int(incomingByte) == 255) {
      digitalWrite(led, HIGH);
      tone(speaker, noteFrequency, noteDuration);
    } else {
      digitalWrite(led, LOW);
      noTone(speaker);
    }
  }
}
```

Figure 53: Drawing of the updated Arduino program for the "scarecrow tree" in Example 3, Part 2. This corresponds to Figure 21.

Bibliography

Bdeir, A. (2009). Electronics as Material: littleBits. *Proceedings of the 3rd International Conference on …*, 397–400. Retrieved from http://dl.acm.org/citation.cfm?id=1517743

Beilock, S. L., & Goldin-Meadow, S. (2010). Gesture changes thought by grounding it in action. *Psychological Science*, *21*(11), 1605–10. doi:10.1177/0956797610385353

Blikstein, P. (2013). Gears of Our Childhood: Constructionist Toolkits, Robotics, and Physical Computing, Past and Future. *IDC '13 Proceedings of the 12th International Conference on Interaction Design and Children*, 173–182. doi:10.1145/2485760.2485786

Booth, T., & Stumpf, S. (2013). End-user experiences of visual and textual programming environments for Arduino. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7897 LNCS, pp. 25–39). doi:10.1007/978-3-642-38706-7_4

Buechley, L., Elumeze, N., & Eisenberg, M. (2006). Electronic/computational textiles and children's crafts. *Interaction Design and Children*, 49. doi:10.1145/1139073.1139091

Chan, J., Pondicherry, T., & Blikstein, P. (2013). LightUp: An augmented, learning platform for electronics. In *Proceedings of the 12th International Conference on Interaction Design and Children - IDC '13* (pp. 491–494). doi:10.1145/2485760.2485812

Chung, K. (2010). OnObject: Programming of Physical Objects for Gestural Interaction, 146.

Clark, A. (1998). *Being There*. Cambridge, Massachusetts: The MIT Press.

Clark, A. (2010). *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. New York: Oxford University Press.

Conway, M. E. (1967). How Do Committees Invent? *Datamation*, *14*(5), 28–31.

Cook, S. W., & Tanenhaus, M. K. (2009). Embodied communication: speakers' gestures affect listeners' actions. *Cognition*, *113*(1), 98–104. doi:10.1016/j.cognition.2009.06.006

Dourish, P. (1997). What We Talk About When We Talk About Context, 1–18.

Dourish, P. (2001). *Where The Action Is: The Foundations of Embodied Interaction*. Cambridge, MA: MIT Press. Retrieved from http://dl.acm.org/citation.cfm?id=513034

Franklin, D., Kiefer, B., Laird, C., Lopez, F., Pham, C., Suarez, J., … Almeida-Tanaka, P. (2013). Assessment of computer science learning in a scratch-based outreach program. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE '13*, 371. doi:10.1145/2445196.2445304

Free Universal Construction Kit. (2012). *The Free Art and Technology (F.A.T.) Lab*. Retrieved from http://fffff.at/free-universal-construction-kit/

Frei, P., Su, V., Mikhak, B., & Ishii, H. (2000). Curlybot: designing a new class of computational toys. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 129–136. doi:http://doi.acm.org/10.1145/332040.332416

Greenberg, S., & Fitchett, C. (2001). Phidgets: easy development of physical interfaces through physical widgets. *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, *3*(2), 209–218. doi:10.1145/502348.502388

Greger, G. (1965). Electronikbaukasten. Germany: Deutsches Patent- und Markenamt.

Greger, G. (1969). Electronic Building Set. *US Patent 3,447,249*. United States: United States Pattent Office.

Gross, M. D. (2014). Cube-in: A Learning Kit For Physical Computing, 383–386.

Hartmann, B. (2009). *Gaining Design Insight Through Interaction Prototyping Tools*. Stanford University.

Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, a., & Gee, J. (2006). Reflective physical prototyping through integrated design, test, and analysis. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 299–308. doi:10.1145/1166253.1166300

Hartmann, B., Klemmer, S. R., Bernstein, M., & Mehta, N. (2005). d.tools: Visually Prototyping Physical UIs through Statecharts. In *Extended Abstracts of UIST 2005*.

Hatch, M. (2013). *The Maker Movement Manifesto: Rules for Innovation in the New World of Crafters, Hackers, and Tinkerers*. New York: McGraw-Hill.

Holloway, S., & Julien, C. (2010). The case for end-user programming of ubiquitous computing environments. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research - FoSER '10*, 167. doi:10.1145/1882362.1882398

Iverson, J. M., & Goldin-Meadow, S. (2005). Gesture paves the way for language development. *Psychological Science*, *16*(5), 367–71. doi:10.1111/j.0956-7976.2005.01542.x

Johnson, S., & Thomas, A. P. (2010). Squishy circuits. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems - CHI EA '10* (p. 4099). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?id=1753846.1754109

Kafai, Y. B., Peppler, K. a, Burke, Q., Moore, M., Glosson, D., & Wright, N. R. A. (2010). Fröbel's Forgotten Gift: Textile Construction Kits as Pathways into Play , Design and Computation. *Proceedings of the 9th International Conference on Interaction Design and Children*, 214–217. doi:10.1145/1810543.1810574

Kay, A. (2005). *Squeak Etoys, Children & Learning*.

Klemmer, S. R., Hartmann, B., & Takayama, L. (2006). How Bodies Matter: Five Themes for Interaction Design. In *Proceedings of the 6th ACM conference on Designing Interactive systems - DIS '06* (p. 140). New York, New York, USA: ACM Press. doi:10.1145/1142405.1142429

Lyon, C. T. (2003). *Encouraging Innovation by Engineering the Learning Curve*. Massachusetts Institute of Technology.

Malafouris, L. (2013). *How Things Shape the Mind*. Cambridge, MA: MIT Press.

McNerney, T. (2004). From turtles to Tangible Programming Bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, *8*(5), 326–337. doi:10.1007/s00779-004-0295-6

Mellis, D. A., Igoe, T., Banzi, M., & Cuartielles, D. (2007). Arduino: An open electronic prototyping platform. In *Proc. CHI* (Vol. 2007, pp. 1–11). Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Arduino:+An+Open+Electronics+Prototyping+Platform#0

Mellis, D. A., Jacoby, S., Buechley, L., Perner-Wilson, H., & Qi, J. (2013). Microcontrollers as Material: Crafting Circuits with Paper, Conductive Ink, Electronic Components, and an "Untoolkit." *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction - TEI '13*, 83. doi:10.1145/2460625.2460638

Millner, A., & Baafi, E. (2011). Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In *Proceedings of the 10th International Conference on Interaction Design and Children - IDC '11* (pp. 250–253). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?id=1999030.1999074

Montemayor, J. (2003). *Physical Programming: Tools for Kindergarten Children to Author Physical Interactive Environments*. University of Maryland, College Park. Retrieved from http://www.cs.umd.edu/~monte/papers/dissertation.pdf

Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., & Amour, A. D. (2002). Physical Programming: Designing Tools for Children to Create Physical Interactive Environments, (4), 299–306.

Ngai, G., Chan, S. C. F., Ng, V. T. Y., Cheung, J. C. Y., Choy, S. S. S., Lau, W. W. Y., & Tse, J. T. P. (2010). A Scalable, Plug-n-Play Wearable Computing Framework for Novices and Children. *Proceedings of the 28th International Conference on Human Factors in Computing Systems - CHI '10*, 443. doi:10.1145/1753326.1753393

Novack, M. a, Congdon, E. L., Hemani-Lopez, N., & Goldin-Meadow, S. (2014). From action to abstraction: using the hands to learn math. *Psychological Science*, *25*(4), 903–10. doi:10.1177/0956797613518351

O'Sullivan, D., & Igoe, T. (2004). *Physical Computing: Sensing and Controlling the Physical World with Computers*. Thomson Publishing Group.

Papert, S. (1993). *Mindstorms: Children, Computers, And Powerful Ideas* (Second Edi.). New York, New York: Basic Books.

Peppler, K., Glosson, D., Kafai, Y., Fields, D., & Searle, K. (2011). Articulating creativity in a new domain: expert insights from the field of e-textiles. In *Proceedings of the 8th ACM conference on Creativity and cognition - C&C '11* (p. 385). New York, New York, USA: ACM Press. doi:10.1145/2069618.2069708

Pinoccio. (2014).

Raffle, H. S. (2004). *Topobo: A Constructive Assembly System with Kinetic Memory*. Massachusetts Institute of Technology.

Raffle, H. S., Parkes, A. J., & Ishii, H. (2004). Topobo: a constructive assembly system with kinetic memory. *System*. doi:10.1145/985692.985774

Ratto, M. (2011). Critical Making: Conceptual and Material Studies in Technology and Social Life. *The Information Society*, *27*(4), 252–260. doi:10.1080/01972243.2011.583819

Reas, C., & Fry, B. (2006). Processing: programming for the media arts. *Ai & Society*, *20*(4), 526–538. doi:10.1007/s00146-006-0050-9

Rekimoto, J., & Sciammarella, E. (2000). ToolStone: effective use of the physical manipulation vocabularies of input devices, *2*, 109–117–109–117.

Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., & Silverman, B. (1998). Digital Manipulatives: New Toys to Think With, (April), 281–287.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for All. *Communications of the ACM*, *52*(11), 60. doi:10.1145/1592761.1592779

Robles, E., & Wiberg, M. (2010). Texturing the "material turn" in interaction design. *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction - TEI '10*, 137. doi:10.1145/1709886.1709911

Sadler, J., Durfee, K., Shluzas, L., & Blikstein, P. (2015). Bloctopus: A Novice Modular Sensor System for Playful Prototyping. *Proceedings of the Ninth ...*, 347–354. Retrieved from http://dl.acm.org/citation.cfm?id=2680581

SAM. (2014). Retrieved from http://samlabs.me/

Silver, J., Rosenbaum, E., & Shaw, D. (2012). Makey Makey: Improvising Tangible and Nature-Based User Interfaces. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction - TEI '12* (p. 367). New York, New York, USA: ACM Press. doi:10.1145/2148131.2148219

Suzuki, H., & Kato, H. (1995). Interaction-Level Support for Collaborative Learning. In *The first international conference on Computer support for collaborative learning (CSCL '95)* (pp. 349–355). Morristown, NJ, USA: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=222020.222828

Tickle. (2014). Retrieved from http://www.tickleapp.com/

Trofatter, C., Kontra, C., Beilock, S., & Goldin-Meadow, S. (2014). Gesturing has a larger impact on problem-solving than action, even when action is accompanied by words. *Language, Cognition and Neuroscience*, *0*(0), 1–10. doi:10.1080/23273798.2014.905692

Tuomi-Gröhn, T. (2008). *Reinventing Art of Everyday Making*. (T. Tuomi-Gröhn, Ed.) (1st ed.). Frankfurt am Main, Germany: Peter Lang GmbH, Internationaler Verlag der Wissenschaften.

Zuckerman, O., Arida, S., & Resnick, M. (2005). Extending tangible interfaces for education. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*. doi:10.1145/1054972.1055093