Code Against the Machine: Beginner-Friendly Physical Computing Technologies for Performance

by

Annie Kelly

Kelly, Annie (M.S., Creative Technologies and Design)

Code Against the Machine: Beginner-Friendly Physical Computing Technologies for Performance

In typical collaborations between artists and technologists, artists do not have much control over authoring the technical aspects of their projects. DIY and beginner-friendly programming technologies seek to mitigate some of the challenges novices face when learning to program. These technologies are often designed for making interactive systems, which have applications for use in artistic installations and live performances. While beginner-friendly technologies can mitigate some of the challenges of programming and electronics, the current tools are still challenging for novices who want to implement artistic interactive systems that are authentic to their creative goals. This thesis encompasses my research around investigating the challenges that novices face implementing their own creative computational systems for live artistic applications and performance. I discuss findings from these studies and how they have informed my design decisions for several toolkits I have built, and discuss design possibilities for future technologies that might empower artists and performers to have more creative control of the technical aspects of their work.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

Patti Smith said, "Every human being has a creative impulse, and we all have the right to exercise this creative impulse" [62].

The inaccessibility of interfaces for creative expression stands in the way of many human beings' creative impulses. The inaccessibility of these interfaces is further compounded by systematic gatekeeping in STEM fields that impedes access to computing education on the basis of social and economic factors. This prevents those who might want to learn computation to support their creative pursuits from doing so, and further, adds to a culture where underrepresented communities might not express interest in the field or believe they are not capable of learning.

In this thesis, I focus on the evaluation of physical computing technologies for expression, and discuss ways to adapt these technologies to be more accessible to novice programmers who want to use computation in live, interactive art and performance.

There are many possible applications of interactive technologies in creative work and performances. Perhaps an artist wants to add some form of audience-interaction to a show, or use physical gestures to generate visual or sonic effects for a composition, or they may simply want a button they can press on stage to change the color of the lights. In this context *interactive technology* means a computational system that takes one or more kinds of input, and maps it to some expressive output(s). The design and development of interactive systems usually requires prior knowledge of programming and electronics. Typically, when an artist has a vision that requires complex technical components such as these, they contract the work to a technologist such

as a programmer or an engineer. This results in artists and technologists having unequal levels of involvement in different aspects of the project, with the technologist usually the sole lead in implementing the technology [15]. This is understandably a common model, as currently available physical computing tools for designing interactive systems can be challenging for non-technologists to work with and many artists just do not have the resources, time, or desire to have to learn how to implement these systems for themselves. These problems are exacerbated for independent, DIY artists and musicians who have even less access to technological resources and support than their more renowned artistic counterparts. Usually, DIY artists are not sought after for these types of collaborations, nor can they afford them. Meanwhile, creators that can afford to hire technologists still experience inequity in these collaborations because they have less control regarding the designs and implementations of these computationally expressive systems [15].

Power comes from having control over production and having access to education [27]. A response to calls to democratize this power and increase inclusivity in computing fields is the push for beginner-friendly technologies. These technologies seek to provide non-technologists a starting point to use computing despite lacking access to computing education. While these beginner-friendly technologies support expressive computation in some domains, they are still lacking in accessibility for non-technologists to use them in ways that align with their creative goals.

When it comes to studying the impacts of DIY and beginner-friendly technology for the designs of interactive systems, much of the focus is around children and K-12 learning environments. Programming is becoming a more ubiquitous skill taught throughout K-12 schools and is now an activity those without technical backgrounds increasingly have more access to. However, generations of adults who have long since graduated high school missed out on these opportunities to learn about computer science — leading to the lack of capacity that artists have to work with these tools themselves. In addition, the types of projects that these beginner-friendly tools and curriculum are designed for are not well-suited for enabling artists to create meaningful projects. Artists should not be denied the right to exercise their creative impulses due to inaccessible tools. We, as designers, can and should do better to support artists who want to make use of computing

as a medium for expression.

1.1 Research Questions

My agenda for addressing this current state of inequity, and for designing more creatively empowering tools involves three phases: understand, evaluate, and empower.

- *Understand*: What computational artistic projects do non-technologists want to create?
- *Evaluate*: In what ways do currently available beginner-friendly physical computing toolkits support their goals, and in what ways do they impede them?
- *Empower*: How can we design more expressive and accessible tools for artists?

Through *understanding* what computational projects artists want to create and *evaluating* what is challenging about implementing these projects with currently available tools, we can redesign physical computing technologies that might *empower* artists who want to integrate technology into their work.

1.2 Key Terms and Definitions

To clarify my research, I will provide context for the terms *performance*, *physical computing*, and *beginner-friendly*. I discuss performance and physical computing in this section, and in section 2.6 I will provide an in-depth account of how various researchers and practitioners interpret beginner-friendliness.

1.2.1 Performance

This work encompasses a wide spectrum of possibilities for what defines performance. By *performance* I mean artistic performance, not performance in the way that researchers might use it as a term to describe efficiency or speed of a system.

It is almost futile to attempt to define performance, just as it is difficult to define what any type of art is. For the purposes of this thesis, performance broadly describes a person, or multiple

people, enacting an embodied activity live for an audience. I leave this definition intentionally vague to provide room for one's own interpretation of what an embodied activity is, what an audience is, and what a performer should be. However, in the studies I discuss later on, an embodied action will typically refer to a person's use of their body and/or mind, and can describe a person's interactions with an environment, other people, physical and digital objects, and more. Placement, space, momentum, movement are all important elements of an embodied performance and can describe how a performer interacts with their own body, and the world around them. Some traditional examples of embodied performance include: dance, live music, drag, stand up comedy, theater, etc.

Performances can occur in any number of settings. Many traditional performances occur in venues, such as a concert hall or bar, and typically have a clear division between the performers on stage and the audience who is either seated or standing. However, performances are not bound to occur in these specific or traditional environments; they can occur out in the streets of a crowded city, unbeknownst to the surrounding public, or even be live streamed to the world from a living room.

Regarding content, performances can be emotional, funny, political, educational, self-expressive, thought-provoking, and sometimes performances can be done for seemingly no point at all.

1.2.2 Physical Computing

Physical computing is the integration of physical phenomena, electronics, and computation to create interactive systems. These interactive systems can respond to real-word phenomena, as well as manipulate its environment through different electronics-based output and actuation. Typically, people build physical computing projects with microcontrollers because they enable engineers and designers to create mappings between inputs and outputs. Throughout this thesis I refer to certain physical computing technologies as *programmable*, meaning that people can customize the technology by providing computational instructions for how it should behave.

Physical computing systems have applications in the arts [29], museums [57], industrial environments [107], educational settings [11], and more. My research focuses on physical computing

systems and their applications to art and performance.

1.3 Chapter Summaries

Chapter 2: "Background" discusses the motivations and relevant work for this research. I summarize research regarding interfaces for creative expression, artist-technologist collaborations, the current state of "Do It Yourself" (DIY) expressive technologies, feminist perspectives on computing education and technologies, and researchers' and practitioners' perspectives on beginner-friendliness.

Chapter 3: "Investigating Equity Within Artist-Technologist Collaborations" describes a study I co-facilitated around investigating non-technical artists' learning about the applications of computing for creative expression during a creative design event. I discuss the interactive creative projects they built in their groups, and their roles in these collaborations.

Chapter 4: "Woe is LED" discusses two studies that describe the challenges of working with computational technology to creative interactive LED lighting systems. The first study is an autoethnography about my experience engineering software for a dancer's performance, and the second is a case study of a pair of students' challenges implementing an LED data visualization system for an art installation.

Chapter 5: "Beginner-friendly Tools, Meet Advanced Composition" describes results from working with a group of students enrolled in a dance course to help them integrate interactivity into their performances using novice-oriented physical computing tools.

Chapter 6: "Redesigning Physical Computing Tools" describes a series of technologies I have designed as a response to my experiences working with non-technical artists, as well as proposals for future designs that could make these tools more accessible.

Chapter 7: "Conclusion" provides a recapitulation of the research I have completed, as well as future work that needs to be done.

Chapter 2

Background

2.1 Personal Motivation

"Let me change this and see what happens," has been my approach to working with technology since I was a preteen learning HTML to customize my Neopets and Myspace accounts, and learning to create custom content for my Sims games. It has always been important to me to use computers to make things that are creative and playful, not simply productive and pragmatic. I see computers as powerful mediums for creative expression.

I am passionate about music and technology, and I have always felt empowered being able to express myself through these mediums. It was not until my first college computer science course that I realized other people did not view me through such an empowering lens. I experienced both outward and implicit sexism from my peers. All of a sudden I became an "other" in a community that I thought I belonged in and had to fight to make sure I did not begin to internalize their demeaning perceptions. Prior to this, I was naive to the rampant racism and sexism that plagues STEM fields, and experiencing this for the first time was surprising and frustrating. It opened my eyes to the constructs that we impose on ourselves and others that impact our identity and our self-efficacy, dictating the life paths we choose based on what we are told we will not be good at, or what we will not be interested in. Growing up, I had the freedom to explore these activities without any expectations, with no one telling me not to try something because of some preconceived notion that it was not something I would be good at. Many young girls and adult women do not experience support like this, and many of them begin to believe that they would never be able to

learn computing. Many people do not see the creative potential of using technology, and many who do lack the opportunities and resources to learn how to harness it. I want performers and artists without engineering backgrounds to be able to harness computation, and to be able to meaningfully express themselves with it.

2.2 Interfaces for Creative Expression

There are communities of artists, engineers, and researchers that work in the realm of interfaces for live creative expression. The application of computing for musical expression used to be a niche practice, but is now more common in performance. Primarily, computer musicians or experimental artists would use these technologies, but now these technologies are used in mainstream live performances, museums, schools, and interactive art installations. Physical computing technologies enable artists to add unique components to their work by exploring various modes of interaction for performers and audiences.

The *New Interfaces for Musical Expression* (NIME) community promotes research on the designs of musical interfaces and controllers. Instruments and controllers from the NIME community frequently make use of physical computing tools that use sensing of physical inputs to generate musical output. These performance technologies incorporate the use of microcontrollers, electronics, sensors, as well as audio, visual, and mechanical outputs. These projects explore a variety of interaction paradigms and control mechanisms. This technology can be automated, or it can be controlled by performers, environments, and/or audiences. Physical computing technology allows performers to try new gestures that go beyond what is limited by acoustic instruments [18]. Marije Baalman's Chrysalis apparatus uses e-textiles and stretch sensors embedded into a cocoon-shaped instrument where she, the performer, can affect the lighting and sound by making small minimal movements from inside [3]. In another project, researchers created an audience interaction performance framework called iPoi that lets participants affect sounds and visuals by swinging around traditional Poi balls with embedded accelerometers [95].

Cook's work on musical controller design illuminates a point of contention regarding the

role of technology and its place in performance. Cook argues technology sometimes has the issue of lacking "transparency" in a performance, meaning that the technology is doing something to mediate input and output that is difficult for the audience to interpret. For example, an audience can easily interpret the act of a performer pressing on piano keys and the resulting sound of a beautiful composition—this is an example of transparency. An example with less transparency would be a performer that has non-visible accelerometers in their shoes and a projection of abstract visuals that are based on the reading of said accelerometers, but do not seem to match what gestures the performer is enacting. The general opinion of the NIME community is that a lack of transparency hides effort, and can lead to a confusing performance [18]. These tensions are important to consider, because while we can appreciate the affordances of performance technology, it is important to be critical of its issues.

2.3 Artist-Technologist Collaborations

While there are many talented performers with multidisciplinary artistic and technological skills, i.e. can develop their own technical systems and perform with them, most performers are not engineers. Typically, when an artist has creative goals that require the implementation of computational systems, they must work with technologists to realize their visions.

Many researchers prefer to study artist-technologist collaborations that involve renowned artists and expert technologists. These studies often provide technical resources and support for the artists involved, and result in interesting work that sheds light on the collaborative and co-creative structures that occur between experts from both fields. One of the most famous examples of artists and technologists collaborating together is *Experiments in Art and Technology* (E.A.T.) and its resulting series of performances called "9 Evenings: Theatre and Engineering" [67]. E.A.T. paired artists with engineers and provided them access to new cutting-edge technologies in order to see what was possible when applying these technologies in artistic domains.

Some renowned artists already have access to cutting-edge technologies and enjoy integrating them into their performances. For her 2007 *Volta* tour, Björk collaborated with the Reactable

team to customize and integrate the technology into her performance [50]. Imogen Heap, inspired after visiting with researchers at the MIT Media Lab, collaborated with a team of technologists to create *The Gloves*, a pair of gloves she could wear to control live music with her gestures [44].

Johnston lists several criteria for how to conduct practice-based research in the domain of live performance and suggests working "with experienced and high-calibre artists." [49]. Collaborations with prestigious artists help ensure that the value of the collaboration remains focused on the artistic practice and less-so on the technological innovations spurring from it. This is a way of addressing the "value reduction problem" which says that focusing on the technological innovations of the work mitigates the fact that creativity and the arts are of value themselves, not just because of their economic impacts [61]. Performers like Björk and Imogen Heap certainly fit the mold of the "high-calibre" artist, and therefore attract experienced engineers and funding. However, these opportunities are not available, nor are they accessible methods for independent performers that do not have the name recognition or prestige. Unrenowned and unfunded, independent artists who want to exercise their creative impulses must rely on DIY practices, resources, and technologies.

Even esteemed artists that have access to technological collaboration and support, can experience issues of inequity within artist-technologist collaborations. Inequity in this case means that a particular population lacks access and resources in becoming technologically literate [42]. Candy and Emonds studied collaborations between artists and technologists, and found the technologist is usually the sole lead for the technical implementation in these collaborations, and the artist is the lead for the conception and evaluation component of the project. They describe many of these collaborations as a technological "support" model where the artist has a vision the technologist implements it [15]. For many, this model is tried and true. The artist has the ideas and the technologist has the means. In order for the artist to truly be in charge of the conception and evaluation, there must be a mutual respect between parties, and it is important for them to "develop a common language" when discussing design criteria and technical issues in order to agree on a plan [15]. However, even technologists who listen and communicate effectively with their artistic collaborators exercise more control over the technical implementations of their projects [15].

Another one of the inequitable aspects about this model of collaboration is that it is often incredibly gendered. Steve Albini, renowned producer and engineer, has criticised the male-dominated environments of the performance industry saying that "there's sort of a fraternal network" when it comes to who is in control of the more technical roles, highlighting the unseen power and control that men in the performance industry have over artists and creators (Steve Albini as quoted in [43]). This "fraternal network" is not only limited to the music and performance industries, but is generally an imbalance that is prevalent across all computing and engineering fields, with only 26% of the computing workforce being women in 2017, and only 3% of those women being black and 1% hispanic [74].

These criticisms are not to say that well-funded initiatives between experienced artists and engineers are not valuable, these partnerships often result in wonderful performances and informative research. However, if we continue to only focus our resources on providing computational creative support for esteemed artists, we are choosing to ignore less-privileged makers, artists, and performers who have the potential to create incredible art as well, further perpetuating divisions of creative, technological production between those deemed worthy and those not.

2.4 DIY, Expressive Computation

For the aforementioned artists who do not have the means to have someone implement technical work for them, there are independent creators whose mantra is "do it yourself" (DIY). DIY, as the term implies, is about people building, repairing, and creating projects on their own. With the emergence of DIY technologies we are seeing more creators embarking on projects that they normally would not have been able to do without intervention from a professional.

One example of the benefits of making DIY tools more accessible to independent creators is how the ease of sharing content online has democratized the music industry for independent creators. While lack of access to producers, sound engineers, and studio time used to be very limiting for musicians, now independent musicians can create content at home and share their work at minimal to no cost [79, 78, 99]. While the quality is still disparate, this is an example of how

the designs of accessible and low-cost technologies can shift the power dynamic for creators and and give them more autonomy over their own work.

In the realm of DIY for expressive computational systems, we are seeing a push for creators to design their own computational systems for expressive interactive projects. DIY communities and resources have been around for quite some time, especially relating to crafting. DIY has extended itself to the "maker" and "hacker" communities which takes the idea of doing it yourself and extends it to the activities of programming and working with electronics. Technologies like Arduino and Raspberry Pi are popular computational technologies within the DIY community. In addition to the genesis of more DIY toolkits, there are communities and online resources dedicated to helping people with DIY technical projects, like Instructables and Hackster.io.

Despite DIY being a movement that celebrates the ability for non-experts to engineer their own projects, technical DIY is still challenging and reserved for the tech savvy [68, 69, 70]. Tools like the Arduino are great for "doing it yourself" when you already have enough technical know-how [20]. Novice-oriented tools exist, but this thesis will demonstrate how they are limited in their capabilities and accessibility.

2.5 Feminist Perspectives on Computing Education and Work

Beliefs within DIY culture are aligned with feminist and marxist perspectives on demoncratizing work and education [27, 32]. DIY enables creators to reap the rewards of their own hard work, instead of it being lost to capitalist systems that they produce labor for [90]. DIY has strong roots in feminist circles, particularly in the creation and sharing of self-made zines which freely circulate information to other community members [17]. Online marketplaces for DIY makers and crafters, like Etsy, have allowed self-employed women creators to earn income by having an easier way to build an online profitable presence [77].

Feminist critiques of technology culture consider the male bias of technology and the division it creates between how society values private (home and reproductive labor) work less than public (industrial) work. These critiques also reject the notion that innate biological sex differences lead

women to be inferior to men in technological fields, but rather that these are constructs that are constantly reaffirmed through existing power structures and inequitable divisions of labor [103].

Despite the increase in DIY platforms for makers and crafters, DIY tools for computing and electronics are still lacking in their accessibility for novice programmers. Women, despite earning 57% of undergraduate degrees, earn only 18% of undergraduate computer and information science degrees [74]. Maker communities are often comprised of a similar makeup to that of computer science fields — white, male, wealthy, and college educated. In a 2013 survey from MAKE and Intel analyzing the demographics of the "maker universe" (i.e. Maker Faire exhibitors, MAKE magazine subscribers, and MAKE newsletter subscribers), they showed that these groups are comprised 81% of men (a similar number to the 82% of CS degrees awarded to men) with a median age of 44, and a median income of over $100,000. In addition, 97% attended college, with 80% also having post-graduate education [65]. This group has co-opted the phrase "maker universe," despite having an incredibly homogenous makeup of a particular type of maker. Meanwhile, women's participation in computer science has been continuously decreasing since the 1980s, further erasing them as members of this technocentric "universe" [2].

The number of undergraduates who choose to major in computer science is steadily increasing, but gender and racial diversity among CS majors is not [14]. Enrollment booms have posed challenges for CS departments regarding how to keep up with the influx of students. In an investigation into how Computer Science departments are responding to these enrollment booms, Patitsas et. al found that many departments use gatekeeping of college major (rigorous requirements to be a CS major with the use of "weeder" or "weed-out" courses) and gatekeeping of particular courses (non-CS majors cannot enroll in courses) as common solutions for limiting class size when faced with staff and space constraints. These methods create more competitive cultures within departments, force beginners out of the field, and perpetuate the stereotype of the hardcore, individual, male "hacker," all resulting in decreased female participation [66, 82].

Some attribute low representation and high dropout rates to thoroughly debunked myths perpetuating the idea that women leave the field because they do not like programming. They

back these claims with suggestions that innate biological differences cause women to be inferior in performing these skills [72]. Others, suggest that women *are* capable of being competent programmers, but that they prefer to pursue more nurturing professions that are people-oriented. While research does suggest that women are more highly motivated to pursue work that does social good [37, 55], many women who enter computing fields cite that they enjoy the challenging problem-solving it requires, collaborating with colleagues, and applying computing to different contexts [100]. The suggestion that women, or anyone for that matter, do not pursue computing because it lacks collaboration or lacks a direct impact on society only further perpetuates the idea that programming and technology cannot be used for social good, or that it cannot be a highly collaborative and creative activity. Programming activities and jobs extend beyond the stereotype of the lone, brilliant male hacker who sits at a computer screen all day.

Instead of simply relying on societal perceptions and justifications that perpetuate myths of why women leave computing, researchers at Carnegie Mellon directly interviewed women who left CS majors before completing their studies to assess why many left the field. The authors outline reasons women CS students exit the field before completion of their studies. Some of these reasons include women not relating to the aforementioned "hacker" culture that suggests one must be "obsessed with computing" to belong. Other reasons involve self-confidence issues where they judge their perceived success against male students in their programs who came in with prior programming experience, despite achieving equally high marks in their courses. In addition, the authors discuss how weed-out courses increase drop-out rates for those who do not have prior experience with computing, typically women, non-binary, and minority students [31].

2.6 What Does it Mean to be "Beginner-Friendly?"

DIY computing technologies are not currently accessible enough to empower most novice technologists to create their own computational projects. Designers are responding to demands for more accessible and inclusive DIY computational technologies and STEM education by developing "beginner-friendly" programmable technologies. Beginner-friendly usually means that they are

designed for people with little-to-no programming experience. With the push to have more of these technologies in schools and homes and in the hands of non-expert hobbyists we see even more initiatives, both educational and consumer-based, to make them more accessible.

The pursuit for beginner-friendliness begs the question: What is currently unfriendly about current technologies and educational initiatives? I separate the discussion into tool-, practice- or context-, and critical-based perspectives (see Figure 2.1). These three levels of framing include perspectives on: cognitive aspects of tools, designs of tools that are situated in particular pedagogical contexts, and critical sociocultural perspectives that include addressing widespread social and cultural change. This inside-out view is similar to Bronfenbrenner's ecological systems theory that illustrates micro-to-macro perspectives ranging from how individuals interact with their immediate environments, to how they coexist within larger social systems [12]. Similarly, Kafai et al.'s overview of how researchers study computational thinking ranges from cognitive, to situated, to critical framings [51].

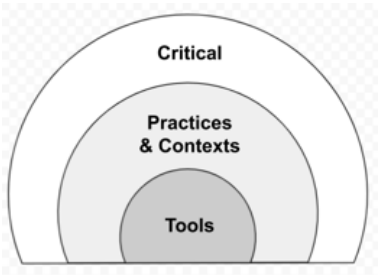

Figure 2.1: Perspectives on beginner-friendliness ranging from critical, to practices and contexts, to tools.

2.6.1 Tools

Highlighting the unfortunate inaccessibility of programming languages, Ko said, "Programming languages are the least usable, but most powerful human-computer interfaces ever invented" [56].

Research has shown us that learning to program is not a beginner-friendly experience, but rather a beginner-hostile one [5, 10, 26, 89, 97]. Blackwell's definition of programming is activities

that move away from working with concrete controls, to working within a given notational abstraction [5]. In other words, programming usually involve the loss of direct manipulation. Hutchins et al. describe some of the advantages of direct manipulation that include supporting novices' abilities to learn functionality quickly, and its legible feedback of users' actions and how they affect the program state. The "gulfs of execution and evaluation" describe the challenges a user must overcome in order to achieve a goal using a given computational system. The *gulf of execution* refers to a user taking action to accomplish a goal, and the *gulf of evaluation* describes a user's understanding of a system's current state. Direct manipulation interfaces can help tighten these gulfs, but do so at the cost of flexibility [46].

In addition, Spohrer and Soloway's "plan composition" problems describe problems learners encounter while designing programs [98]. These problems involve different aspects of program design: failing to consider unexpected cases, drawing inappropriate analogies between the natural language and the programming language, and incorrectly summarizing problems [89]. Designing both small- and large-scale software projects requires deciding what a program should do (requirements), planning when the system will execute differently and accounting for unexpected cases (specification), determining how this behavior will be supported (design), implementing the program using abstractions (coding), and determining when something goes wrong and finding a solution (debugging) [5]. Similarly, Du Boulay has identified four categories that describe the challenges of programming, which include orientation, pragmatics, notional machine and notation, and structures [10]. The *How to Design Programs Design Recipe* is a response to these challenges and lays out a defined process for designing programs [26]. Even this recipe, which breaks down the process of programming into a series of steps, still assumes that the designer has a working concept of foundational CS concepts required to correctly implement non-trivial programs.

As a way of addressing the cognitive challenges of working with programming tools, researchers have created heuristics that generally support more positive user experiences. These heuristics consider how technologies can facilitate interaction and use by limiting cognitive load. Nielsen identified a set of ten usability for heuristics for user interfaces that relate the Hutchin

et al.'s gulfs of execution and evaluation, which include, making system status visible to users, using language that aligns with users' expectations, helping users recover from errors or preventing them entirely, and more [75]. Similarly, Gerhardt-Powells created a set of Cognitive Engineering Principles that include limiting data-driven tasks, grouping data, automating unwanted workload, and other principles that align with Nielsen's [35].

2.6.2 Practices and Contexts

While many researchers take low-level and cognitive-oriented approaches for assessing usability, some consider how to design programming languages situated within given contexts. Some argue programming languages and interfaces are not well-aligned with practices of end-users [73], and the usability of these tools can be improved by being designed with their pedagogical contexts in mind (e.g. a programming language for computer music should use keywords and terminology that are related to this domain).

Due to booms in computer science education initiatives, there has been an increase in the number of designers building programmable technologies for educational contexts. For many of these designers, the image of a "beginner" user is a young child or an adolescent, and they aim to develop programming languages that are student- and learner-centered and align with the pedagogical goals and practices of a K-12 classroom.

The earliest designers considering computer science applications in education identified programming languages as too cumbersome to use, and that there should be a child-friendly language to enable them to design programs. One of the first examples of this is the move away from the BASIC programming language to a more child focused language called Logo [80]. Perlman's TORTIS (Toddler's Own Recursive Turtle Interpreter System) was one of the first physical computing devices for children which allowed them to use physical blocks to program the Turtle, which was a programmable disc that moved around and drew [85]. Later, researchers created LEGO/Logo which was a combination of the Logo programming language and LEGO construction tools [87]. However, according to Pea, Logo does not perform well in authentic classroom conditions [83].

Eventually, researchers moved away from Logo to more direct manipulation based interfaces, like Cricket where computation was embedded in tangible bricks that children could physically configure [6]. Some of these languages are designed to support pedagogical goals beyond learning computer science, fostering skills within math or physics education [96].

Scratch is a popular browser- and blocks-based programming interface for youth that has a large online community of creators that share and remix projects [86]. Scratch's remixing culture promotes the *use-modify-create* learning progression that supports learners' creation of projects by first providing them with remixable and customizable programs before asking them to create their own entirely [64].

Guzdial's *Five Principles of Programming Languages for Learners* lists several criteria for programming languages in educational settings. The first principle is "connect to what learners know"; instead of treating programming as an entirely new concept that students have never encountered before, we should build on concepts they are familiar with. The second principle is to "keep cognitive load low"; which recommends designing languages that reduce human working memory. The third is "be honest," suggesting that designers should create languages that are suitable for learners but still intellectually authentic regarding computer science concepts. The fourth principle is "be generative and productive"; programs should enable learners to express ideas and to work productively in different domains. Finally, the fifth principle is "test, don't trust"; designers should put these tools in the hands of learners and see what happens. We should not assume that we have accomplished the four aforementioned principles without actually testing in situated contexts [39].

In addition, researchers have created programmable technologies for artistic and self-expressive contexts. Many of these include rapid prototyping kits for hobbyists and technologies like Raspberry Pi and Arduino. Some designers have addressed the lack of accessibility these computational toolkits support and have made more domain specific kits to support creative projects. For example, the LilyPad is a hardware platform designed by Leah Buechlay that supports the use of soft materials in physical computing projects to create e-textiles and wearable electronics. LilyPad-based curricula

provide materials that supports novices in sewing electronics into clothes with modular electronics that are easy to connect and attach with conductive thread [13]. While these expressive toolkits make some activities easier like working with electronics, artists using these tools for fashion related projects still encounter technical challenges when implementing their ideas [47].

A more general purpose, novice-oriented toolkit for working with physical media is the micro:bit [4]. Educators and engineers developed the micro:bit as a low-cost tool to use in classrooms to engage more youth with STEM [91]. In 2015 the BBC provided over a million free micro:bits to year seven students in the UK. Evaluations of students' experiences with the micro:bit demonstrate its "easiness" to program [92]. Microsoft MakeCode is a browser-based programming editor that lets users program the micro:bit with blocks or JavaScript. MakeCode offers event-driven blocks which users can nest code inside of that run on given state changes (e.g. `on button pressed` do something). These blocks are helpful for beginners in designing their first interactive systems, because simple interactions can rely on the event-driven blocks as opposed to using variables and conditionals [48]. Several of the studies I discuss in this thesis utilize the micro:bit, and I will discuss my findings of the affordances and disadvantages of its use as a tool in the hands of artists.

Participatory design methods can help generate more beginner-friendly tools by illuminating existing challenging of tools and help align design choices with the natural practices of particular populations [8]. In 2017, my research group and I co-designed a curriculum with a middle school music teacher around teaching tangible and distributed computer music systems using our BlockyTalky toolkit [93]. Part of our co-design involved changing the notational representations of the music composition blocks to better align with the teacher's musical pedagogical practices. The teacher, who I will refer to as Mr. Lindon, taught music composition through his pedagogy of scale degrees and melody rules. Instead of students programming sequences of literal notes like, "play 'C,' 'E♭,' then 'G,' students choose what key they are composing in with a 'play in the key of C minor" block and then use scale degrees or "finger numbers" to program the notes, "in the key of C minor, play 1, 3, and 5" (see Figure 2.2 for the block code for both of these programs) [94]. Mr. Lindon's theory behind this method is that students can learn compositional rules more abstractly

and generalize them between keys.

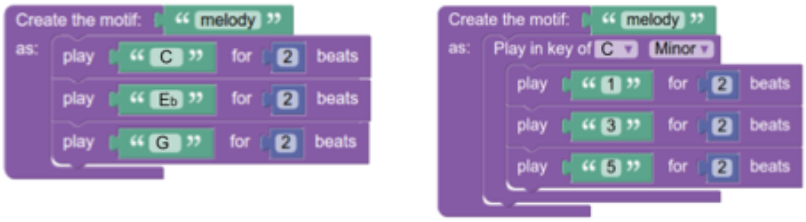

Figure 2.2: Two BlockyTalky programs that produce identical musical output. The left is the standard way of programming compositions in BlockyTalky, the right is using the finger number abstractions that Mr. Lindon used for teaching music theory.

The BlockyTalky example also demonstrates the idea of designing a programming toolkit that is domain-specific rather than general purpose. A *domain-specific* language (DSL) is designed for working on a problem within a particular domain. DSLs specialize in making the features of that domain easier to work with. HTML is an example of a DSL as it is specifically designed for creating web pages. A *general-purpose* language (GPL) is more generalized and can be used in applications across many different domains [60]. Python, C++, JavaScript are all examples of GPLs. Research has shown that one of the benefits of DSLs is that they are easier to learn, with both beginners and experts having an easier time working with them [59]. Fischer and Nakakoji compare general-purpose environments to domain-oriented design environments by discussing the effort required that is "related to the computer" versus "related to the authentic task." They argue that in general purpose environments people spend more effort related to the computer than the authentic task, whereas domain-oriented environments have the benefit of focusing more effort on the authentic task itself [30]. Guzdial, using the term "task-oriented" rather than "domain-oriented," argues that programming languages should be task-oriented and designed to teach ideas within particular domains, instead of designing languages that we claim to be well-suited over a whole domain of knowledge (like computer science) [40]. However, a result of this domain specificity is that it leads to narrower walls by design. These languages intentionally limit the scope of what they can be used to accomplish, making them suited to help students "learn something" [40], however it is questionable how helpful they are for practical, general building purposes.

2.6.3 Critical

Some researchers focus more on the sociocultural structures that impact learning and engagement with computer science. Researchers suggest aligning with cultural practices can encourage learning computer science and increase engagement [22, 23, 25, 41, 52]. In addition, researchers highlight sociocultural factors that decrease engagement. While poorly designed programming technologies can deter productivity and inhibit learning, there are sociocultural structures that impede programming engagement and inclusivity as well, some of which I discussed in section 2.5.

One method for engaging novice programmers is supporting the creation of personally meaningful projects. Constructionist learning is engaging people to develop mental models and understand the world through projects that connect to their interests and passions [41]. In interest-driven learning frameworks, learners are said to be more motivated to *master* certain skills rather than *demonstrate* them as a way to pass assessments [23]. For example, in a comparison between two novice programming environments, researchers found that the environment designed for storytelling was more engaging for young girls and increased the amount of time they wanted to spend programming [52]. Unfortunately, designers sometimes create programming-based "edutainment" applications that seemingly connect to young learners' pop-culture interests, but provide shallow learning experiences—highlighting the importance of striking a balance between designing for interest and designing for learning [22]. Culturally responsive education seeks to explore scientific and technical phenomena directly related to the population receiving education [25] and can be mediated through culturally situated design tools [24]. According to Freire, oppressive education systems remove students from their cultural contexts and treat them as passive objects; an empowering system on the other hand would acknowledge students' connections to each other and the world, as well as promote a more equitable relationship between teacher and student [34]. Value-driven learning, which supports the motivation of community-scale learning through targeting culturally valuable and meaningful projects as opposed to individual interests [21].

For technologies that promote the creation of personally meaningful projects, "low floor, high

ceiling, wide walls" has become somewhat of a mantra for the beginner-friendly designs of these tools. The key ideas are to design systems that are easy to get started with, allow for creating increasingly more complex projects, and support the creation of a variety of different projects so as to engage many different types of people. While I agree with these ideas in theory, I will demonstrate later on why very little designers have come close to accomplishing this. In addition, these principles are tool-centric and it is hard to assess these boundaries. Walls are significantly wider and ceilings significantly higher in the hands of experts, even if the floor is low enough for novices to get started.

2.6.4 A Synthesis of Perspectives

In Table 2.1 I have identified several common themes among the varying perspectives of "beginner-friendliness."

Perspective	**Description**	**References**
Motivation and alignment with identity	Designers and facilitators should increase motivations to learn computation by aligning tools and curricula with individuals' and/or communities' identities and goals.	[22, 21, 23, 24, 25, 34, 52, 84]
Narrow gap between user goals and computational system	Tool should support users in accomplishing their computational goals by making it easier to perform actions and subsequently understand the results of those actions.	[35, 46, 75]
Low floor	Tools and curricula should support novices and first-time users by being "easy" to start using.	[38, 81, 88, 86]
High ceiling	Tools and curricula should provide the ability to add increasing complexity to projects as users learn more skills.	[38, 81, 88, 86]
Wide walls	Tools and curricula should support the creation of different types of projects.	[84, 86]
Connect to what people know	Tools and curricula should use language and paradigms familiar to users and "match" with real-world socio-cultural conventions.	[39, 75, 104]
Provide learner support and promote inclusivity	Learning environments, both informal and formal, should avoid gatekeeping practices and perpetuating stereotypes that drive underrepresented groups away from computing.	[16, 31, 33, 82, 100]
Scaffolding & modularity	Tools and curricula should facilitate learners in progressively adding complexity to their projects.	[45, 20]
Apply tested heuristics for improving usability	Designers should evaluate tools against thoroughly tested heuristics proven to improve usability.	[35, 75, 104]

Table 2.1: Synthesis of some perspectives on beginner-friendliness.

Chapter 3

Investigating Equity Within Artist-Technologist Collaborations

In 2017, I co-designed a workshop in order to study collaborations between artists and technologists building expressive computing systems with the use of a beginner-friendly technology called *BlockyTalky*.

BlockyTalky is a programmable toolkit for building physically interactive, networked music controllers from our research group [93]. Up until this study, members of the BlockyTalky research team had primarily studied its application in informal and formal K-12 environments. In these settings, the research goals were to facilitate students' learning of computation and networked systems in the context of creating musical compositions and controllers [94]. I participated in many of these research studies hands on, and am a BlockyTalky developer. I saw the value of providing people with an accessible toolkit for making expressive interactive systems and considered its applications in the hands of adult artists who wanted to build these types of systems as well. I wanted to explore BlockyTalky's affordances for adult creatives who want to create interactive physical and digital systems. In this study, I wanted to investigate the following research questions:

(1) What physical creative systems do adults create with BlockyTalky?

(2) In what ways does BlockyTalky mediate collaborations between artists and technologists?

3.1 BlockyTalky

3.1.1 Networking

BlockyTalky was designed to mediate students' learning of network programming and distributed computing. Multiple BlockyTalkys on the same local network can communicate with one another using blocks for sending and receiving messages. BlockyTalkys can send messages of strings to other BlockyTalkys by name. On the receiving end, BlockyTalkys have event-handlers that fire when they receive messages. Figure 3.1 shows BlockyTalky programs for sending and receiving messages. The left program sends the message "play" to a BlockyTalky named "walle" when a button is pressed. The right program shows "Walle's" BlockyTalky code that loops a motif called "melody" when it receives the message "play." In order to make complex compositions and

Figure 3.1: BlockyTalky code that sends a message (left), and code that receives a message and triggers an event (right).

controllers, students have to use multiple BlockyTalkys for their project, requiring them to design network protocols across distributed devices.

3.1.2 Sensing

BlockyTalky software runs on the Raspberry Pi. With the addition of a GrovePi shield, BlockyTalkys support integration with a variety of analog and digital Grove sensors. Users can program event-handlers that fire on different sensor events like button presses, temperature changes, and more. In addition, users can write BlockyTalky programs that turn on LEDs and actuate motors. The GrovePi shield is equipped with connectors that let users rapidly plug-and-play with sensors; users do not need to build circuits to work with the I/O of the Raspberry Pi. BlockyTalky provides built-in block abstracts for interfacing with these sensors (see the sensor event-handler in

Figure 3.1), enabling students to rapidly construct physically interactive and networked systems [53].

3.1.3 Music

BlockyTalky supports the synthesis of real-time audio output, and provides block abstractions that interface with SonicPi, a Raspberry Pi music synth application. Users can plug in headphones or speakers into the Pi's audio jack. Users can create named sequences of notes called *motifs* and *cue* them (play them once) or *loop* them (play them repeatedly). For example, students have programmed BlockyTalkys to play full-length John Legend and Coldplay songs where they could trigger different parts of the songs to play based on different sensor events that they designed.

3.1.4 Enhancing Networking with Open Sound Control

In its earlier iterations, BlockyTalkys could only communicate with other BlockyTalky devices. They had no ability to communicate with external software or hardware. In order to enable BlockyTalky interfaces to be used to control professional artistic applications, I added Open Sound Control (OSC) functionality to BlockyTalky to allow connections to external software and hardware. OSC is a popular standard for networked communication in computer music performance that enables labeling and sending of data that can be used to affect parameters of the system [106].

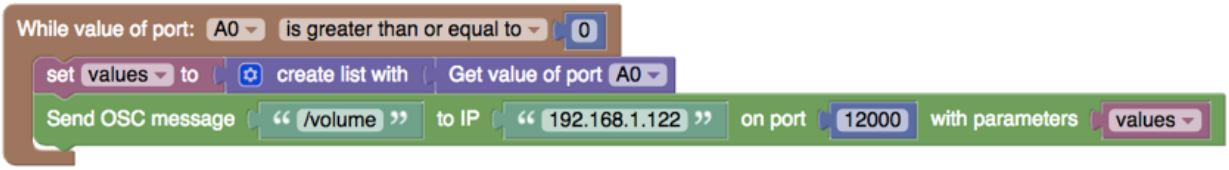

Figure 3.2: A BlockyTalky OSC program.

The addition of OSC to BlockyTalky meant it could now integrate with applications like Max/MSP, Processing, Unity, PureData, or anything that had the capability to send and receive OSC messages. With BlockyTalky, users can rapidly construct physical interfaces of sensors to control output in these external applications. Figure 3.2 is a BlockyTalky program that monitors

the state of a potentiometer connected to the board on Analog Port 0 (A0). When the potentiometer (knob) is turned and greater than 0, the BlockyTalky sends an OSC message, containing the value of the potentiometer, to a Max server at 192.168.1.122 on port 12000. The Max application receiving OSC messages (not shown in the figure) maps the value of the potentiometer to the volume. Figure 3.3 shows two photos of a live performance with BlockyTalkys. This performance uses a Max patch for audio playback and rendering for live visuals. The left photo shows a tangible BlockyTalky interface to control the tempo, distortion, pitch bend, and formant control of real-time audio playback with three potentiometers and a water sensor respectively. The right photo shows a BlockyTalky controller that controls the zoom, randomness, and color of the visuals using an ultrasonic sensor, a soil moisture sensor, and a temperature sensor respectively.

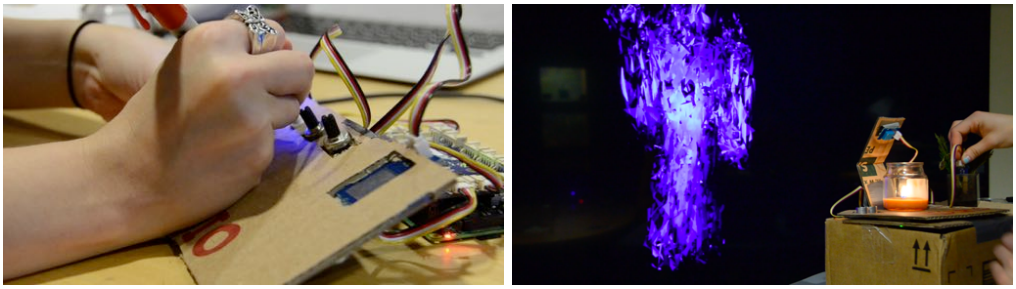

Figure 3.3: Using two custom-built BlockyTalky interfaces for an audio visual performance.

3.2 Creative++

In order to investigate how non-technologists could use BlockyTalky for the creation of expressive, live performances. Monica Bolles and I designed *Creative++: A Creative Arts and Technology Jam.*

3.2.1 Methods

3.2.1.1 Recruitment

We drew inspiration for the structure of this event from other design jams, which are collaborative and creative hackathon-style events where participants work in groups to create a compu-

tational project [36]. We wanted to study artist-technologist collaborations, therefore we wanted to attract non-technical artists to attend and not just computer scientists. To accomplish this, in our recruitment materials we emphasized that no prior experience was required and that we would welcome and support beginners with introductory workshops and beginner-friendly materials. We hoped that advertising this event as a beginner-friendly, creative jam would encourage attendance of participants from disciplines other than computing. Particularly, we wanted the event to be inclusive to underrepresented groups in computing that might normally feel inadequate to participate in a hackathon [58], or that would normally not be interested in stereotypical computer science projects. The sign-up form served as a pre-screening interest form and allowed us to choose who to admit, striking a balance between novice and experienced technologists in attendance. We chose this over a first-come-first-serve model because we wanted to avoid the typical demographics of a hackathon, which is usually experienced, confident technologists [19].

We admitted 26 participants to the event. Our demographics included a self-identified gender breakdown of 2 non-binary participants, 11 women, and 13 men. We had an even split between participants who were in non-engineering majors and jobs, and those who were not.

3.2.1.2 Facilitation

In order to actually support beginners in participating in a design event like this, we hosted an optional three hour workshop the night before to cover the basics of the technologies we would be using. During the workshop we provided an introduction to using BlockyTalky. We began with a hands-on tutorial of how to create networked musical devices between BlockyTalkys, and then moved on to how to use the OSC blocks for communicating with other applications. We provided a set of sample projects in Processing, Max, Unity, Python, and a variety of other applications. We set up the sample projects as skeletons containing the bare minimum amount of code necessary to receive an OSC message that modified some parameter in the project (e.g. we designed the Unity example to receive an OSC message that would change the speed of a cube's rotation). We hoped providing remixable OSC project skeletons would give participants the ability to quickly

connect BlockyTalky to an example project and then modify it later on. We modeled this after the example bundles that are offered on the Wekinator website that simplify the process of incorporating different input and output applications for custom interactive machine learning applications [28], and the use-modify-create scaffolded learning structure [64]. All of the participants that attended the main event came to the workshop the night before.

On the morning of the event, we asked participants to find people with similar project interests to them, but complementary skill sets. Once participants formed their groups, we asked each group to spend the next 12 hours creating a project using BlockyTalky that used physical and tangible interaction to control some type of live, interactive output. We told groups that they could either solely use the BlockyTalky toolkit for their projects, or connect BlockyTalky with outside software and hardware. We hoped that requiring the use of BlockyTalky would provide the non-technologists in the group the ability to work on the interaction design between the physical sensors and their corresponding artistic outputs, leading to an increased amount of equity in the actual implementation of their computational system. We assumed that groups would likely split responsibilities such that the non-technologists would work within the BlockyTalky environment, and that the technologists would be responsible for working within more traditional text-based environments.

3.2.1.3 Data Collection

We collected participants' pre-screening information to provide us context for the demographics of participants including their age, gender identity, college major or career and responses for their answers regarding why they want to participate and what they hope to learn. In addition, we conducted semi-structured interviews of participants, collected questionnaire responses, collected project documentation such as photos, code, and notes, and captured recordings of groups' final presentations and performances. We gave participants the questionnaire at the end of the event, therefore a few people who left early did not have the opportunity to fill one out. The questionnaire asked identity- and project- specific questions like, "have you ever programmed something before

this event?" "do you identify as a programmer?" "have you made art before?" "do you identify as an artist?" "what did your group make?" "what was your role on this project?" "what were your teammates' roles?" and other questions asking about their opinions on the usability and experience of using BlockyTalky.

3.2.2 Results

All six groups successfully completed either a proof of concept of their project or a fully functioning project (see 3.4 for photos of each project). The final project titles and descriptions are as follows:

- ***Blind As a Bat***, an echolocation inspired game where players use two rotary angle sensors and try to navigate through a maze and collect objects without looking at the screen. Players can only use LED and audio feedback. The audience can watch the player move through the maze (made in Processing) in real-time.

- ***Orphan***, an audio visualizer that used one BlockyTalky for physical input, three BlockyTalkys for audio synthesis, and Max to generate visuals. A person controls the music and visuals in real-time by interacting with the BlockyTalky sensor unit.

- ***The Pyramid***, "an interactive physical computing puzzle game." This group constructed a cardboard pyramid with embedded sensors (ultrasonic, button, rotary angle, slide potentiometer, light, and water). Players had to figure out how and in what order they needed to interact with the sensors to solve the puzzle. As players interact with the pyramid a Python script triggers text-to-speech to give them audible feedback.

- ***GwendyTine***, a cute cardboard car equipped with hydraulics, lights, and music. Using feedback from the BlockyTalky sensors, the car detects input and can trigger engine noises and alarm sounds indicated through BlockyTalky synthesizers and servo actuation from a Python program on a Raspberry Pi.

- ***WiredASL***, a gesture and sign language controlled instrument that required two performers to train their gestures with a Kinect and Leap Motion using Wekinator. The performer using the Kinect controlled the octave of the BlockyTalky synth based on how high or low they were in the frame, and the performer using Leap Motion controlled the pitch of the BlockyTalky synth using ASL gestures that they mapped to specific notes.

- ***Shark Attack***, an installation that uses an ultrasonic sensor and facial recognition to detect when someone gets too close to a hungry shark's plate of cookies. When the shark is angry, a Max patch plays Jaws music which gets louder and faster when a person is close to the cookies. An animation of a shark made in Unity attacks the person to protect the cookies.

Figure 3.4: Six photos of different projects. Top row, from left to right: Blind as a Bat, Pyramid, Gwendytine. Bottom row, from left to right: WiredASL, Orphan, Shark Attack.

Participants used external hardware (Raspberry Pi, Kinect, LeapMotion, and an LCD display) and software (Wekinator, Max/MSP, Unity, Processing, Matlab, text-to-speech) in their projects. Participants used OSC as the primary protocol for communicating between their applications, including between BlockyTalky to BlockyTalky, BlockyTalky to non-BlockyTalky, and non-BlockyTalky to non-BlockyTalky. All but one group (WiredASL) used BlockyTalky for the implementation of the sensor interactivity; the WiredASL group instead used a Kinect and LeapMotion as the input methods, and used BlockyTalky for its music synthesis abilities. All other groups used BlockyTalky for sensor input and sent the real-time data from the sensors to the external applications of their choice, although the GwendyTine group used BlockyTalky for both sensor input and musical output.

In regards to participants' self reported artist and computer scientist identities, we found that

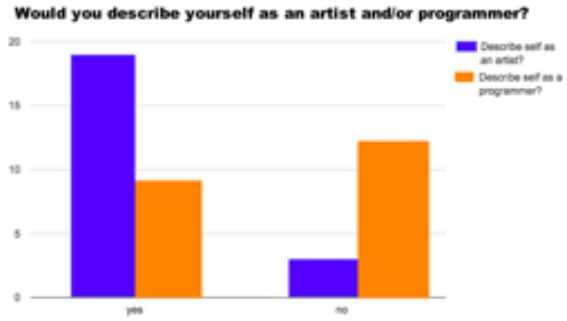

Figure 3.5: Participants' art and computer science identities.

people who have made art before were more comfortable identifying as an artist, but many of those who have programmed before did not identify themselves as a programmer. 20 of 21 participants said they have made something artistic before, and of those who said they have made something artistic 90% said they identify as an artist. 17 of 21 participants said they have programmed something before, and of those 17 responses 47% said they identify as a programmer. Table 3.1 shows the questionnaire responses of each participant sorted by group regarding their self-described roles in the project, as well as their descriptions of their teammates' roles in the project. Table 3.1 also includes their responses to the programming experience and identity question. In the last column of the table, "team described role," the numbers in parentheses after the quotes indicates which teammates' surveys these responses came from.

Blind as a Bat

#	P^I	P^E	Self-Described Role	Team-Described Role
10	N	N	"Communication between OSC and BT, construction of remote"	"Networking/blocks" (18) "Designer/Artist" (32) "UX Design" (20) "Code/bat controller" (17)
17	N	Y	"Moral support"	"Blocky Audio" (20) "Programmer" (32) "Blocky sound" (18) "Music, BlockyTalky setup" (10)
18	Y	Y	"Max/MSP + OSC"	"Programmer" (32) "Max program design" (17) "Max/MSP routing" (20) "OSC programming" (10)
20	N	Y	"Processing game dev"	"Programmer" (32) "Processing game design" (17) "Processing" (18) "Programming with Processing?" (10)

32				"BlockyTalky programming" (10) "Block, LED" (18) "Blocky components" (20) "Coding" (17)

Orphan

1	Y	Y	"code"	"main programmer" (9) "programmer" (27)
9	N	Y	"designer/visuals"	"visual" (1) "designer" (27)
27	Y	Y	"designer"	"Code, visual" (1) "Main programmer" (9)

The Pyramid

2	Y	Y	"Programming/design"	"Coding the entire setup" (7)
7	N	Y	"Helping with all the tasks"	"Design" (2)
				"Programming/planning/design" (2) "Set up pyramid" (7)
				"Physical computing aspects" (2) "Testing sensors" (7)
				"Physical computing aspects" (2) "Helped make the pyramid/coding" (7)

Gwendytine

12	Y	Y	"Car design + LED grill"	"Python integration / car display" (14) "LED display, hydraulics master" (28)
14	N	Y	"Sound design"	"Sound system & hydraulics" (12) "Sound guy" (28)
28	Y	Y	"I was Gwen caretaker"	"Babysat Gwen & programmed her" (12) "Code comp & syncing" (14)

WiredASL

13	Y	Y		"Kinect" (22) "Setup Xbox for octaves" (26) "Kinect" (29) "Setup Xbox Kinect, coding" (34)
22	N	Y	"Social media, records"	"Visuals, train hand sensor, docs" (26) "Documentation" (29) "Documentation, social media, charts" (34)
26	N	N	"Powerpoint, trained hand sensor, documentation"	"Presentation, tester & modeler" (13) "Hand and Kinect model" (22) "Presentation & signing" (29) "Made powerpoint, recorded hand motions" (34)
29	Y	Y	"Code and background knowledge"	"Coder" (13) "Coding" (22) "Programmed — notes, accidentals" (26) "Code, communication b/w computers/pis" (34)
34	N	N	"Powerpoint, recorded hand motions"	"Presentation, tester & modeler" (13) "ASL expertise" (22) "Powerpoint, train hand sensor, docs" (26) "Presentation & signing" (29)
				"Coding the Kinect" (13) "Kinect" (22) "Setup Xbox for octaves" (26) "Kinect" (29) "Setup Xbox Kinect, coding" (34)

Shark Attack

5	N	Y	"Unity"	"3D artist & nautical env. designer" (6) "Unity software for animation" (21) "Unity. Dev. Shark movement. Leader" (33)

6	N	Y		"Max MSP" (5) "Max software to provide sound" (21) "Max. Dev. Shark/water sounds. Leader" (33)
21	Y	Y	"Worked on integrated face recognition in MATLAB with BTU & triggering specific events based on that"	"BT, facial recognition, code" (5) "Facial recognition specialist" (6) "Interface & BT. Dev face recognition to interact with shark program" (33)
33	N	N	"Programmed sensors to interface w/ Unity (shark) - rate of spinning and Max (Jaws sounds) volume"	"Sensor extraordinaire" (5) "E-textile design & sensor" (6) "Integrating depth sensor w/ BTU" (21)

Table 3.1: Groups' final projects, and their group- and self-assigned identities.

3.2.3 Discussion

The addition of OSC to BlockyTalky enabled groups to create interfaces to control professional artistic software like Max, Processing, and Unity instead of having to rely on the musical capabilities of the BlockyTalkys themselves. This ability to interoperate with more sophisticated software enabled the rapid construction and customization of live controllers for expression, demonstrating its potential beyond K-12 settings for use by adult creators.

As expected, the projects that participants created still required at least one technical expert on the team. However, all participants were able to contribute to some level of the technical implementations of their projects. The non-technologists tended to assume the design-oriented roles, and their roles in the technical implementation primarily involved programming sensor-based or musical events within the BlockyTalky environment. This allowed novices to have some control over the interaction design of the system, despite not being able to contribute to work that required more traditional text-based programming.

Regarding identity, participants showed a tendency to more freely identify as artists and not as programmers. We did not gather data to support *why* participants might not identify as a programmer despite having programmed before, but speculate that this occurrence may be because many of those who have programmed see it as a way to create art and therefore it supports their artistic identities. In addition, it is possible that participants were not as confident about their programming abilities to identify as a programmer, suggesting there might be more issues

of self-efficacy in programmer identities as opposed to artistic ones. This seems to be validated through participants' tendency to make a distinction between BlockyTalky and programming as separate entities as noted in quotes like "BlockyTalky and coding" instead of just "coding". Other participants referred to their roles as "BlockyTalky programming" which shows that they associated BlockyTalky as a programming activity. Future studies could investigate the sociocultural phenomena behind these responses to assess what leads a person to identify or not as artists or programmers, and what people deem valid programming activities.

To summarize, groups were able to collaborate on creative computational projects and perform with or demonstrate them by the end of the event. BlockyTalky enabled those with less programming experience to contribute to the designs of their physical interactive systems by programming the sensory inputs and mappings, as well as the client portion of the OSC protocols. Even though all participants were able to contribute to the technical implementations of their projects, I still wanted to improve on these findings and further remove the need for intervention from technologists. I am motivated to create resources that would allow non-technical artists to collaborate with others out of creative desire, not out of necessity because of the inaccessibility of computational tools.

Chapter 4

Woe is LED

In this chapter, I discuss the challenges of programming interactive lighting systems using a novice-oriented physical computing toolkit. First, I describe an autoethnographic account [1] of my collaboration with a dancer to engineer software for a performance. Then, I present a case study of a pair of novice programmers building an interactive LED data visualization. Both projects incorporated the use of similar materials: the micro:bit, MakeCode programming platform, Neopixels, and basic electronics components.

4.1 An Autoethnography of Engineering Software for a Dancer

In 2018, I assisted a student named Erin (pseudonym) with the technical elements of a performance. Erin is a dancer and costume designer who was producing a dance performance for her undergraduate senior thesis. Her performance was around exploring relationships between different forms of dance, and she wanted to use light-up, interactive costumes to convey these relationships to the audience. I hoped that through supporting a performer's creative technical goals I would get firsthand insight into the affordances and challenges of participating in an artist-technologist collaboration, as well as elucidate the limitations of novice-oriented physical computing tools.

Erin is a talented costume designer and has prior experience working with wearable electronics. She enjoys sewing LEDs into her costumes to make them light up. She also has programming experience from a few introductory undergraduate courses, and from her prior experience using the

micro:bit to change the colors of the lights in her costumes. Still, even with her experience creating light-up costumes, she was overwhelmed by how to implement the complex interactive system that she envisioned for her performance, and conveyed little interest in learning how to implement the software herself. This was understandable as she was responsible for all of the other aspects of the show — building a trapeze, choreographing the performance, recruiting dancers, sewing her costumes, picking the accompanying music for the show, etc. In one meeting Erin said, "Someone asked me today about my project and I said if I didn't have to build anything I'd be happy. I care about the final product, not so much the process." However, she initially had to try implementing everything herself because, as an undergraduate student, she did not have the budget or the resources to hire an expert technologist to assist her. I offered to collaborate for free and took on the role of implementing the software for Erin's costumes.

4.1.1 Methods

Because of my background in computer science, I was able to employ autoethnographic methods rather than study an artist-technologist collaboration from the outside. In doing so, I hoped to better understand what it is like to be the technical support for an artist's performance, and to provide insight on the challenges of using beginner-oriented tools for a live dance performance.

Following similar models of artist-technologist collaborations as described by Candy and Edmonds, I assumed a technologist "support" role for Erin. In this collaboration model, Erin was in charge of the concept, I led the construction and implementation aspects of the project, and we were both involved in the evaluation of the work, with Erin having more control given it was her performance. Our collaboration lasted about three and a half months

4.1.2 Results

4.1.2.1 Creative Goals

Erin wanted to represent the tensions and synergies between dance genres through choreography, music, and interactive LEDs. She wanted the distance between dancers to affect the colors of

the LEDs on their costumes, and use individual dancers' gestures to affect the animations of LEDs on the costumes. By the time I joined the project, Erin had already sewn micro:bits and LEDs into some of her costumes. She had also written short programs that made all of the LEDs light up the same color when they were powered on to show that everything was wired properly. Figure 4.1 shows an early-stage illustration by Erin for what she envisioned for the costume designs.

Figure 4.1: Digital renderings created by Erin of the appearance of her costumes, left to right: hip hop (tanktop), house, hip hop (pants), swing, blues, contemporary, aerial. Salsa not pictured.

Where I stepped in was where she was having trouble actualizing the next steps of her project. Her performance goals required mapping proximity and accelerometer data to the animation and color transitions of LEDs. Erin wanted each costume to have a default color. She chose colors that she felt represented the genre of each costume. In addition, she wanted the costumes' colors to change and blend based on distance as dancers moved in and out of proximity to one another. For example, if Dancer 1's color was red and Dancer 2's color was blue, their costumes would begin to appear more purple (the combination of red and blue) as the dancers moved closer to one another. I will refer to this interaction as *proximity-based color blending*. In addition, Erin wanted several of the costumes to have LED animations that would correspond to an individual dancer's movement. At this point, Erin was not sure what animations she wanted or what gestures she wanted to use as inputs, but Table 4.1 shows the eventual interaction rules for each costume.

Genre	Costume Components	Interaction Rules
Aerial	Top (1 micro:bit, 85 LEDs) Skirt (1 micro:bit, 78 LEDs)	Maps acceleration intensity of wearer to sparkle animation rate of the LEDs
Contemporary	Shirt (1 micro:bit, 34 LEDs)	Maps proximities of the House and Blues costumes to the blend of the LEDs' colors
Blues	Suspenders (1 micro:bit, 68 LEDs)	Maps proximities of the Swing and Contemporary costumes to the blend of the LEDs' colors
Hip Hop	Tank Top (1 micro:bit, 65 LEDs) Pants (2 micro:bits, 84 LEDs/leg)	Maps proximities of the House and Salsa costumes to the blend of the LEDs' colors
House	T-Shirt (1 micro:bit, 75 LEDs) Pants (1 micro:bit, 63 LEDs/leg)	Maps proximities of the Hip Hop, Contemporary, Swing, and Salsa costumes to the blend of the LEDs' colors
Salsa	2 shoulder pads (1 micro:bit, 63 LEDs/pad)	Maps proximities of the House and Hip Hop costumes to the blend of the LEDs' colors
Swing	Skirt (1 micro:bit, 97 LEDs)	Maps proximities of the House and Blues costumes to the blend of the LEDs' colors

Table 4.1: Descriptions of costume components and the interaction rules for their behavior.

4.1.2.2 Implementation Challenges

Even as someone with a background in computer science, my experience engineering the software for these costumes was time consuming and challenging.

One of the more complex challenges of this project was implementing reliable proximity detection between dancers. The micro:bit has a built-in radio that lets micro:bits send and receive packets (i.e. messages that contain data) to and from other micro:bits. Users can send messages that include numbers, strings, or key-value pairs, and can program event-handlers on the receiving micro:bit that run code when it receives any of these data types. Micro:bits send additional information with the packets such as the time the packet was sent, the serial number of the sending micro:bit, and the signal strength of the packet. The signal strength depends on the transmission strength of the micro:bit, the directionality of the signal, and the distance between micro:bits. With some calculations, micro:bits can generally detect proximity between one another by sending packets in intervals and reading the signal strength. The function call for accessing a remote micro:bit's signal strength returns a value between -128 and -42 decibel-milliwatts (dBm), ranging from weak to strong respectively. The micro:bit does not provide an abstraction for detecting the general distance between micro:bits (e.g. a block that might say "get distance to micro:bit with serial number

X" and return a value between 0–100 or null depending on the relative distance between the two). Instead, users who want to utilize proximity-detection must implement their own mechanism to do so, which is where the capabilities of the tools go beyond the abilities of the beginners that this tool is designed for (see Figure 4.2). In order to achieve proximity detection, I had to accomplish several tasks: (1) write a program where two (or more) micro:bits are constantly sending packets to each other, (2) implement an event-handler that stores the signal strength when a packet is received, (3) keep a rolling average of a few of the last received signal strengths to account for dropped packets or outliers, and (4) utilize information from multiple micro:bits pinging one another to account for the directionality of the radio (i.e. two micro:bits right next to each other will report a weak signal strength if the radios are facing opposite directions).

Figure 4.2: Code that detects the proximity between two micro:bits.

With the given proximity information, I could now begin mapping the distance between dancers to the LEDs' color blending effect. The premise was simple, each costume has a default color, and each costume affects the colors of one another in real-time by mixing the colors based on proximity. However, neither the micro:bit nor the Adafruit Neopixel library contain built-in linear interpolation (lerp) functions for color, which meant I needed to implement this feature from scratch. Essentially, I needed a function that would accept two colors and return a resulting color somewhere along the line between the two (e.g. the half-way point between red and blue would result in a purple LED). At the time, you could not create MakeCode block functions that accept parameters (now you can), so I could not make one arbitrary function to do this. Instead, on each costume I implemented functions like "shift Red" and "shift Blue" that updated the LED colors depending on what costume it was blending with.

Because Erin wanted each dancer to be able to affect the appearances of every other dancer, each costume needed to monitor proximity between multiple other costumes. The software manipulated the LED colors in the costumes differently according to which dancer they are closest to (i.e. the LEDs displayed different colors when the House dancer was near the Swing dancer versus when it was near the Hip Hop dancer). Therefore, I had to design network protocols for each costume that would check the serial number of the micro:bit sending the packet and use that to determine which costume the proximity data was coming from. Also, during the performance it is not like dancers were powering down their outfits, meaning regardless of what part of the performance it was every outfit was actively sending out packets to each other. This meant that outfits' LEDs would abruptly change colors because costumes did not know which costume it should be blending to. To resolve this, I programmed a micro:bit to serve as a remote control that let me send out information to each costume about what part of the performance we were in. For instance, at the start of the show I could specify that the Hip Hop costume should only blend with the House costume and therefore only listen to the relevant packets, and then in the middle of the show I could specify that the Hip Hop costume should be blending based on its proximity to the Blues costume.

In addition to the complexity of dealing with proximity detection, identifying where packets are coming from, and blending colors, animating a nonlinear layout of LEDs is a challenging programming task. Furthermore, animating a layout of LEDs to correspond to the input of real-time sensor data is even more challenging. When it came to the animations on the costumes, Erin had distinct visions on how she wanted them to look. Here is part of a text message from Erin to me describing how she wanted the LEDs on each dancer's costume (House, Salsa, and Hip Hop) to animate based on different types of accelerometer input:

ERIN: House - have every like 10th led be black and have that rotate faster as there is more accelerometer action

Salsa- sparkle with increased accel[erometer] activity

> Hip hop- same as house, but all 3 side stripes move at once. Hips can always stay lighted but then the back strip in the tank top moves like the pants

Erin's request to have the LEDs sparkle at different speeds depending on the acceleration strength of the dancer is a straightforward request. The micro:bit has a built-in accelerometer and corresponding blocks to make it easy to access acceleration data in real-time. In addition, the micro:bit has support for programming individually addressable LED strips. However, while this animation pattern was easy for her to visualize, it was challenging to implement. Programming dynamic animations ends up being a complex task that requires methods used in game design. It involves accounting for the built-in update rate of the computational device and implementing timing mechanisms to change the LED patterns at particular times without using pauses because that would freeze the entire running program. Figure 4.3 is a program I wrote for one of Erin's costumes that maps the acceleration strength of the micro:bit in the costume to the rate of the sparkling LEDs. In this example, the timing mechanism is achieved by the variable `sparkle_rate` which is used to make sure the LEDs only change at a given rate instead of on every single update. The program counts how many updates have passed, and if it is greater than or equal to `sparkle_rate`, then we update the LEDs. The sparkle appearance is achieved by lighting up and turning off random LEDs each time we update them, and requires us to loop over the strip of LEDs each time we do this.

Programming animations over nonlinear patterns over a linear strip is a complex task. Large arrangements of LEDs can sometimes involve multiple strips and varied placements around a body or object. While the programming blocks are great for working within a linear model, they are extremely cumbersome and limiting when users want to do something more interesting than making a stick light up. For example, there is a built-in animation function for LED strips called `rotate` which updates each LED's color to that of the one before it in the strip. Refer to Figure 4.4 for the LED wiring layout of one of the salsa costume's shoulder pads.

The squares indicate LEDs, and the red line indicates the wiring path. We can see that although the LEDs form a shoulder-pad shaped layout, the connections form one continuous path.

```
basic.forever(function () {
    previous_acceleration = current_acceleration
    current_acceleration = input.acceleration(Dimension.Strength)
    acceleration_change = Math.abs(current_acceleration - previous_acceleration)
    sparkle_rate = pins.map(
        acceleration_change,
        0,
        1023,
        1,
        10
    )
    sparkle_rate = 11 - Math.min(10, Math.max(1, sparkle_rate))
    updateLights()
})

function updateLights () {
    if (update_count - sparkle_rate <= 0) {
        update_count = 0
        stripA.clear()
        stripB.clear()
        for (let index = 0; index <= num_lights - 1; index++) {
            if (Math.randomBoolean()) {
                stripA.setPixelColor(index, neopixel.colors(NeoPixelColors.Black))
                stripB.setPixelColor(index, neopixel.colors(NeoPixelColors.Black))
            } else {
                stripA.setPixelColor(index, color)
                stripB.setPixelColor(index, color)
            }
        }
        stripA.show()
        stripB.show()
    }
    if (update_count > 10 || update_count < 1) {
        update_count = 0
    }
    update_count = update_count + 1
}
```

Figure 4.3: JavaScript program for the aerial costume. This program maps the acceleration strength detected from the micro:bit in the aerial costume to the speed that the LEDs sparkle at.

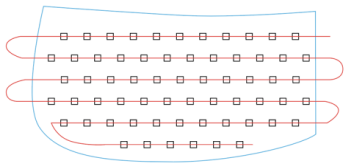

Figure 4.4: The layout of LEDs on one of the salsa costume's shoulder pads. The squares indicate LEDs, the red line indicates the path of the wiring.

If Erin wanted her costume to animate along this line it would be fairly straightforward to program

because we could simply increment over each LED turning it on or off. However, if Erin wanted

Figure 4.5: A photo of two dancers during the final performance. The colors of the swing and house costumes are slightly blended to the other due to the proximity of the dancers.

her LEDs to animate a burst pattern (light up in the center and then light up larger out of rings one after the other) this would be complex to program as this would need to be done by addressing each individual LED in a way that would be harder to increment algorithmically because it would likely required hard-coding of LED indices. This complexity of animating large nonlinear layouts of LEDs proved to be too challenging and prevented us from completing this effect for every costume before the show.

4.2 Case Study of Non-Technologists Programming Interactive LEDs

In the fall of 2018 Lila Finch, Clement Zheng, Celeste Moreno, and I hosted a 12-hour design event called *Talking Trees Jam*, a weekend workshop for participants to collaborate on interactive art installations that embody Colorado forest and climate data. This section presents a case study that further illustrate the complexities of programming interactive LEDs. Here I will discuss the experiences of one group of two non-technologists.

4.2.1 Methods

4.2.1.1 Materials

We supplied participants with micro:bits, input sensors (UI elements, resistive paper, and the embedded sensors on the board), and output devices (LEDs, audio, motors). We hoped these

materials would facilitate the construction of interactive installations due to the simplified process of building physical interfaces that the micro:bit supports compared to the Arduino. In addition, my collaborators and I were comfortable facilitating workshops with these materials.

In addition to physical computing materials, we provided participants with climate and forest data from scientists at our university. The data was in the form of spreadsheets and came with brief contextual explanations from the scientists.

4.2.1.2 Facilitation

We gave tutorials at the beginning of the workshop to teach participants how to program the micro:bit, the basics of working with physical inputs and outputs, and how to design programs that use data. We designed the tutorials around these materials such that participants would gain the knowledge to build an exhibit where users would interact with some type of tangible input which in turn would display data in some form of aesthetic output like lights, sound, and/or motion.

We began our tutorials with a mix-and-match activity. We distributed print-outs of MakeCode blocks-based programs to participants, with some participants receiving the same code. There were six different programs that corresponded to six different interactive micro:bit demos that we placed around the room. We tasked participants with interacting with each of the demos to figure out which one matched the code print-out we gave them. Eventually, participants ended up at a table with other participants who had the same program as them. We then showed participants how to access the MakeCode editor and gave them links to the programs that they had copies of. We tasked participants to work in groups to tweak the code in some way, and once they did, to flash it onto the micro:bit from the demo.

After this, we progressed through different tutorials that involved working with inputs and outputs (I/O) on the micro:bit, how to use resistive paper and how to write a program to use it as a UI, and how to integrate the I/O examples with the resistive paper ones. Because the event was around visualizing climate and forest data, we also went through tutorials using the data generated by our university's climate scientists. We guided participants through how to interpret the data

sets, how to import the data on the micro:bit, and how to use it in their programs.

After the tutorials, we dedicated ample time for group formation, an improvement we made over Creative++. To facilitate this we led a project brainstorming speed dating activity where each participant got to discuss their interests and ideas with every other participant and then afterwards we brainstormed project ideas in a full-group discussion. We synthesized the project ideas in the GoogleDoc and asked each participant to sign up for a project.

At this point, we let participants work on their projects for the remainder of the event (about 6 hours) and supported them throughout the process.

4.2.2 Results

I will focus on the group project made by Sierra and Raquel (pseudonyms), particularly I will focus on Sierra's experience implementing the LED visualizations. Sierra was a Technology, Arts, and Media undergraduate student who was also taking her first introductory programming class, Raquel was a recent Media Production graduate with no prior programming experience. Although there were ten other participants at the event, I found their project offered insight on the challenges of programming LEDs with novice-oriented tools that seemed within the scope of a beginner project. The challenges Sierra faced with the LEDs were similar to the challenges expressed in the prior study with Erin.

4.2.2.1 Project Description

Sierra and Raquel designed an installation of a light up tree that would animate and change as a user moved a wheel that could choose between viewing forest data from four different years (Figure 4.6). Their self-described project description was, "A tree that lights up a certain pattern depending on the year chosen from the wheel. The wheel is divided into four sections, each labeled with a specific year. The data being used will be the Tree Ring Index for that specific year."

The project was made up of two components: a tree and a wheel. The tree contained LEDs and served as the aesthetic, visual output component of the installation. The wheel was for a

person to interact with in order to choose what year to view forest data from, which would in turn change the visual appearance of the LEDs. As a person turns the wheel, a piece of copper tape makes contact with different pins, indicating the position that the wheel is at. Sierra and Raquel wanted the tree's LEDs to change colors and animate differently in a way that corresponded to the data they were visualizing.

Sierra and Raquel did not complete their installation by the end of the event, but Sierra wanted to continue working on the installation in her own spare time to eventually install it in the university library. Sierra came to my office hours a few times after the event for assistance, but was never able to finish the installation due to other course responsibilities she had. The biggest challenge for Sierra that kept her from finishing the project was figuring out how to program the LEDs and make them display differently depending on which year a museum visitor has selected.

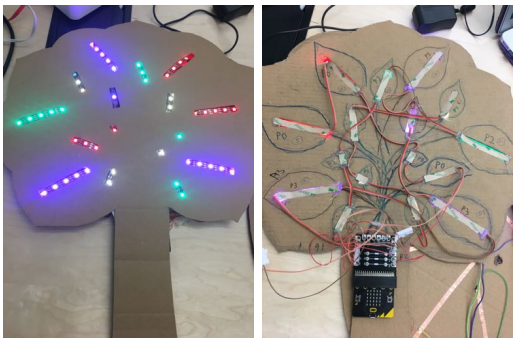

Figure 4.6: Front and back view of the interactive LED tree project. Left photo is the back of the tree installation showing the LED wiring configuration, the right photo is the front view with the LEDs illuminated.

4.2.2.2 LED Programming

If we look at figure 4.6 we can see Sierra's LED wiring configuration on the back of the tree. We can see that she drew outlines of leaves in marker over particular sections of the tree. She labeled them according to which of the four pins that particular LED strip is attached to, hoping to make it easier for her to access the lights leaf by leaf in her program later on. However, this setup is quite complicated because there are four separate LED strips connected to pins 0–3 on the

micro:bit, with each individual strip further broken up so Sierra could place her LEDs in specific parts of the tree. This configuration makes it cumbersome to program the lights leaf by leaf because it requires hardcoding which LED indices make up which leaf. For example, instead of being able to say "I want to update the color of leaf5 to purple," Sierra would have to figure out which pins each of the corresponding LED strips are connected to, and know the indices of the LEDs that are within leaf5. So something as simple as wanting to change leaf5 to purple ends up looking like this (note the different strips and indices):

```
1 lightstrip1.setPixelColor(7, 'purple');
2 lightstrip1.setPixelColor(8, 'purple');
3 lightstrip2.setPixelColor(4, 'purple');
4 lightstrip3.setPixelColor(5, 'purple');
5 lightstrip3.setPixelColor(6, 'purple');
```

As Sierra wrote code to update the 16 different leaves of the tree it became unreadable and hard to keep track of which pixels on which light strip corresponded to what. In addition, she wanted to have animated transitions between different wheel selections which, as shown in the previous example with Erin, only added complexity. Sierra and the I had the following conversation about the difficulties of programming the LEDs how she would like:

SIERRA: I'm thinking about just like doing a very basic visualization of growth of the tree...so just have all the lights lighting up on this board to the index based on the size of the tree or growth of tree index... [and] having that change to be reflected by basically shutting off the number of LEDs to make them go dark.

ANNIE: So more things are lit up on years where there's more growth essentially?

SIERRA: Right, right. But, I'm thinking about still going back to what I said last week I feel like that way is gonna make this thing look a little bit complicated cus the way the four pins are soldered.

ANNIE: Yeah another reason it's hard is cus we are looking at this like "these are the bottom LEDs," "these are the middle" but the way it's wired is what actually

determines —

SIERRA: — Yeah...I feel like [a better way to program] it could be achieved if we had a large grid of LEDs so this is (0,0) at this point or like (1,0) ...ideally if there's like a giant sheet of LED lights then you could basically just cut out the board to cover part of it to make the tree shape and then even outline like "this part of the grid falls into this [tree] branch" and that part falls over there and then programming it that way...but in real life we only have LED strips.

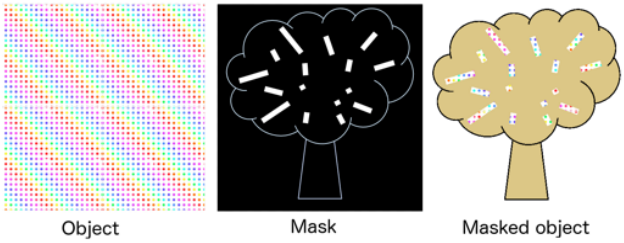

Figure 4.7: Example of the LED masking system Sierra describes.

Sierra wanted to program her project using what is essentially a masking technique (Figure 4.7). Basically, she wanted to start with a large grid of LEDs and then overlay and outline on the grid where she could label different sections as they related to different parts of her tree. This would effectively give her higher-level abstractions to work with and program her lights based on parts of the tree rather than individual LED indices. One of the issues with this technique is that the mask would confine LEDs to snap to a grid instead of having the flexibility to create less structured, more organic layouts. However, her desire to designate specific LEDs as particular programmable entities such as "this part of the grid falls into this [tree] branch" would be a much easier way to program the problem. This way Sierra could think about programming her tree in terms of changing the appearance of sections of the tree, rather than having to enumerate through the right indices of the pixel strips.

4.3 Discussion

LEDs are a desirable technology for performers and artists who want to build interactive visual systems. In Erin's performance, LEDs were an important visual story-telling component. In the Talking Trees workshop, nearly every group wanted to incorporate LEDs into their project; Sierra and Raquel wanted to use LEDs as the primary means of visualizing their data.

Individually addressable RGB strips and the MakeCode Neopixel blocks for the micro:bit make it easier to attach LEDs to a board and light them up compared to accomplishing the same task on an Arduino or Raspberry Pi. While the initial difficulties of attaching lights to a microcontroller and the low-level code of sending signals to the lights are simplified, the creative projects that people want to use LEDs for are still challenging.

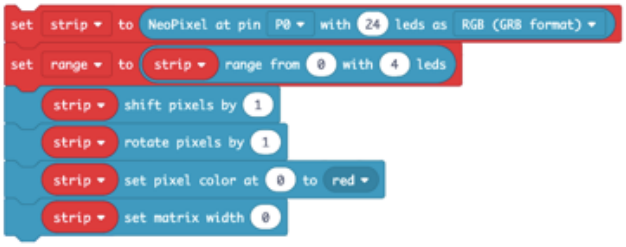

Figure 4.8: Some MakeCode Neopixel blocks.

In both studies, artists worked with the micro:bit and Neopixel libraries to create interactive layouts of LEDs. Unfortunately, even with the affordances of block-based abstractions, this library does not support the creation of LED-based projects beyond linear strips. Figure 4.8 shows a few of the Neopixel MakeCode blocks for initializing a strip of LEDs, and a few blocks for updating their color and appearance. Even the default name and abstraction for a group of LEDs is "strip," which is understandable because in order for LEDs to work properly they have to follow one continuous directional path. Therefore, the abstractions provide manipulations that work best for working over linear strips of LEDs (with support for LED matrix strips as well). However, there is no editing paradigm offered for turning LEDs on and off on less linear grid-like layouts.

Strip-based LED layouts were neither suitable for the dance project nor the tree installation.

In Chapter 6, I discuss ideas for visual programming editors that would work in tandem with these libraries and tools that would be helpful for creating interactive projects for nonlinear layouts of LEDs. While the electronics are confined to being attached as continuous strips, higher-level programming abstractions do not need to be limited in the same way.

Chapter 5

Beginner-friendly Tools, Meet Advanced Composition

In Fall of 2018, I co-facilitated a project curriculum with a Dance professor teaching the course Advanced Composition. My goals were to investigate the expressive and educational potential of integrating physical computing technologies into a performing arts curriculum, and the professor, who I will refer to as M, wanted to provide her students with an opportunity to learn computing.

I wanted to understand what a group of student performing artists' expressive goals are, what physical computing applications are necessary to accomplish these goals, and what aspects of novice-oriented physical computing technologies support and impede their creative goals. Understanding dancers' and performers' perspectives on the applications of computing for performance and seeing how they integrate the technology into their practice would build on my findings from chapters 3 and 4.

5.1 Methods

5.1.1 Materials

I structured the technical aspects of the curriculum around the use of micro:bits and the MakeCode blocks editor. As I have expressed in prior chapters, I believe these materials are useful for creating physical, interactive elements. In addition to the built-in sensors on the micro:bit, I provided them with materials to create custom switch and UI mechanisms like copper tape and aluminum foil. For physical outputs, I provided students with LEDs and servo motors. I also provided one participant with an EEG sensor. The micro:bit has networking capabilities and can

communicate via serial, bluetooth, and radio. I welcomed students to design projects that required connecting their micro:bits to outside software (with my support) such as Scratch, Processing, Max/MSP, MindWave Mobile, etc). For those who did not have personal laptops to program their projects, I lent them Chromebooks. I hoped students would find these materials adequate for integrating technology into the performances they wanted to create.

5.1.2 Facilitation

Before touching any technology or writing any code, M and I explained what we would be working on and co-facilitated a group discussion around what physical computing and interactive technology is, as well as its applications and implications for performance. During the group discussion I passed around micro:bit demos loaded with code that mapped different inputs (acceleration, light, and buttons) to outputs (Neopixel LEDs, on-screen LEDs, and servos). We discussed what the demos were doing and how they worked.

After discussion and demos, I passed out a micro:bit to each student with either a Neopixel LED strip or a servo pre-attached, with the corresponding program that I loaded ahead of time. I showed students how to get to the MakeCode editor from their browsers and how to open the programs running on their micro:bits. I initially asked them to play around and try and "remix" the code, which I have done in past workshops, but they were not comfortable with that and asked for more direct instruction. With the exception of two students, no one had prior experience with programming. So, I had them delete the programs so we could start from scratch, and I improvised the rest of the tutorial. I showed them the different categories in the MakeCode editor, particularly focusing on the input, basic, pins, and Neopixel categories. I asked them to follow along as we made a Hello World style program that showed an image on the LED screen when the micro:bit turns on, using the `on start` block. After that, I showed them how to use a `forever` block to animate an image on the LED screen. I also showed them how to use the event-handlers like `on button A pressed` and `on shake` in the input category to update the attached output (LED screen, Neopixel LEDs, or a servo).

M and I asked the students to brainstorm what they wanted their performances to be and what technology would or would not be necessary to accomplish this. M told students they could continue working on an existing performance, or create a new one. We also gave students homework to take the micro:bit materials home and build three different projects that did whatever they wanted. The three projects did not need to be related to the performance they were making.

The next time we reconvened as a class, each student described what performance they wanted to create, what they thought they would need to build, and what programming it would require, as well as any questions they had about their own project or others'. I gave students advice and suggestions during class, and M and I requested that each of them to schedule an appointment with me before the next class. We decided that we could dedicate class time to some project work, but that it would be more productive for me to work one-on-one with students to give them more personalized assistance. At this point students had two weeks to work on their projects before their final performances.

5.1.3 Data Collection

I audio recorded classroom discussions as well as some one-on-one meetings with students' consent. In addition, I collected sketches, project code, and photographs of prototypes at different phases of completion. I also took my own field notes during classes and one-on-one meetings. A student in the class filmed each of the final performances and gave me access to the videos.

5.2 Results

I separate my results into two sections. I first talk about our initial group discussion highlighting interesting questions and comments that arose regarding technology and its applications to performance. I then talk about students' creative goals and the projects they made.

5.2.1 Group Discussion

M and I loosely structured the group discussion. I began by introducing myself and talked about my research goals of building better novice-oriented technologies for live performance. My goal in the group discussion was to talk about what computing is and what computing's applications in performance are. I introduced the micro:bit technology and the concept of physical computing. Initially, I had trouble figuring out the best way to talk about it in a way that would speak to their interests. M was able to direct the conversation and ask me to list each of the sensors on the micro:bit, that way we could talk about how they work, what their affordances are, and how they might be used in a performance — providing relevant and meaningful context to the discussion. We first discussed the temperature sensor, which was fairly straight forward, but then we moved on to discussing the accelerometer which led to a rich discussion due to gesture's relevance to performance. I talked about what acceleration is, what three axes it can measure input from, and what it can actually detect in terms of gesture. M gave a hypothetical example of a performance that involved choreographed slapping motions.

M: If I went like this (slaps her hand to her leg), would you use the accelerometer to register this gesture? You could say first slap, do this response, second slap, you know what I mean? So you could have a whole pool of different... yeah so the first time you get the accelerometer information play this and then play this.

KYLE: What would you be sensing in that situation? Would it be the ceasing of movement and that triggers the thing?

M: Speed.

ANNIE: You could even use the light sensor, like 'oh now it's dark'. Or use the accelerometer... and I could know this because I know exactly how I choreograph my performance that I will make this slapping movement.

This exchange prompted M to mention the value of using the micro:bit hands on and of experimenting with the sensors to determine their affordances. We then began discussing how some inputs

might be better suited and easier to use than others for specific contexts.

M: And the light might be nice because then you could just go like this... and that's the thing is you'll put them on and you can figure out which sensors work the best or which sensors are the most meaningful. You might think it's this but oh my gosh but the light sensor, when you say this...

KYLE: And it's so funny how easy it is to just over engineer things just like oh the accelerometer when it stops it's going to do the thing but just press it against your body and you're done.

The idea of overengineering led us to discuss the implications of technology as it applies to performance. Mainly, that they should avoid using technology for things they do not need, and that the goals of the performance should necessitate the use of technology and not the other way around.

ANNIE: A lot of times I do think it's easier to think about what it is you want to make and then worry about the tech after, instead of going the other direction of like 'oh I have an accelerometer how can I use the accelerometer?' Because maybe what you want to do doesn't require it at all.

M: Right, and I will say this point that Annie is making right now is really important which is what do you want to say? And then what technology do I need to do it? Not, I have this technology what can I say. That always has a certain...hmmm...gimmick. A trick.

KYLE: Tech fetishism.

M: Yes. And so that question is spot on, what do you want to do? And it could be a piece you've already made.

After much discussion about the different sensors on the micro:bit and their affordances, a student asked a question about whether the micro:bit can be used to manipulate or affect the body, as opposed to using input from the body.

MIRANDA: And in the opposite direction, could the output be a physical reaction? Can you make it vibrate on a body?

Overall, the group discussion offered insight about the questions and thoughts that student performers have about physical computing. Students had limited knowledge of what computing is, particularly physical computing, and it was useful to facilitate the conversation by imagining as a group how these technologies are applicable in performance. The more we discussed how different sensors can be used to measure various input, the more students generated more questions along the lines of "how would I measure X" or "could I use X to do Y?" The group discussion provided students with useful context for the technology before they began programming and working with electronics.

5.2.2 Projects

With the exception of one group of two students, each student completed their computational system and performance and presented it in class (see Figure 5.1 for two photos of participants performing with their system). Table 5.1 provides a summary of the completed student projects.

The Wake by Miranda. Each audience member chooses to either place their rose on the foot switch, or on the floor. The performer is wearing a dress with a micro:bit sewn into the back that has a servo with a needle attached to it. When a rose is placed on the foot switch, the micro:bit on the floor sends a message to the micro:bit in her dress which rotates the servo, therefore scratching her back with the needle.

- **Input(s)**: Foot switch
- **Output(s)**: Servo motion

Motherhood the Game by Sadie. The performer must balance on a balance-board with a micro:bit taped underneath it for as long as she can. The micro:bit reads the state of the x-axis acceleration and when it detects that she has lost her balance it sends a message to a Processing sketch that plays an audio clip saying "game over, you are a terrible mother."

- **Input(s)**: X acceleration
- **Output(s)**: Audio clip

CU Loves Me by Rena. The performer touches the palms of her hands with the opposite hand's finger to trigger different lighting effects on a string of LEDs. • **Input(s)**: Copper tape switches • **Output(s)**: LED animations and colors
Dieting the Videogame by Dana. An audience member interacts with the performer's custom built Scratch game and makes decisions on what to eat and drink at a virtual party. The game prompts them to place translucent chips into a cup with a light sensor at the bottom. As more chips fill the cup, the light sensor value decrements and as it hits certain thresholds the Scratch game plays an audio clip saying "shame!" which prompts the performer to throw bean bags at the participating audience member. • **Input(s)**: Light, UI interactions • **Output(s)**: Audio clip, game state
Untitled by Maggie. The performer wears a brainwave sensing headset and interacts with each of the audience members. Data is sent via OSC through an intermediate application, and then via serial to the micro:bit with a servo attached. The "meditation state" from the brainwaves is a value from 0-100 and continuously maps to the angle of the servo between 0-180° in real-time. • **Input(s)**: EEG • **Output(s)**: Servo motion
Slut/Prude by Riley. Each audience member votes using buttons whether the female dancer is acting like a "slut" or a "prude" based on her interactions with a male dancer. The votes are sent to a centralized micro:bit and are tallied in real-time, which sends serial information to a text-to-speech application that says "slut" or "prude" based on the current votes. • **Input(s)**: 2 built-in micro:bit buttons • **Output(s)**: Text-to-speech audio
Spam by Kyle. Using a text-file containing content from spam emails, the performer populated 3 arrays on the micro:bit with random words. The performer then tilts the micro:bit on 3 different axes in order to sweep through the indices of the 3 different arrays, which in turn creates 3-word phrases that are then projected onto a screen. • **Input(s)**: X, Y, Z Acceleration • **Output(s)**: 3 arrays of words & Projection of randomly generated text strings

Table 5.1: A summary of the projects Advanced Composition students made.

5.2.2.1 Micro:bit-Only Projects

Only two students built projects that did not require integrating the micro:bit with external technology — Miranda and Rena. Miranda's project, *The Wake*, used two micro:bits, two battery packs, a custom-built cardboard and foil foot switch, one servo, alligator clips and wires, and a

needle. Rena's project, *CU Loves Me*, used one micro:bit, a battery pack, a Neopixel strip, copper tape, alligator clips, and a pair of gloves (both shown in Figure 5.1).

Figure 5.1: Two different project performances. Left is *CU Loves Me* and right is *The Wake*.

For Rena's *CU Loves Me* performance, she wanted a pair of gloves for controlling the lights to accompany music. Her music was a satirical song that she composed and recorded. She had no prior programming or electronics experience and was not sure where to begin in embedding electronics into her gloves. I suggested that we use copper tape and walked Rena through the process of where the tape could go and how we could attach them to the board depending on what hand gestures she wanted to use. Rena wanted to use her pointer fingers to touch the centers of her palms. On each glove we placed two sections of copper tape: her palm and her pointer finger. The copper tape on the palms of the gloves was attached to two input pins, and the copper tape on the pointer fingers was attached to power (3V). Therefore, the micro:bit could detect a signal when Rena touched either of her palms with her pointer fingers. Rena used a Neopixel strip as output and the lights would change colors and animations depending on what configuration she was touching or not touching her palms.

In a one-on-one meeting that took place before we started the gloves circuitry, Rena was explaining the progress she made on her project. She explained that she figured out how to light up a strip of LEDs in a rainbow pattern because she found a `show rainbow` block.

RENA: I know there's a rainbow thing on there... the "show rainbow." And so I did the

> rainbow lights but I didn't know how to make them move positions, so that's about as far as I got. I just made the rainbow and then it came on when I pushed the button and then a face came up. Like I push this and it does a thing, and then I push this and it does a thing! But no moving parts or anything. . . oh except I had one forever command and it was the little heart to the big heart so it looked like it was beating, you know to show that it is alive.

We spent the next few minutes pair programming to figure out how to "move" the lights. Rena was in control of the computer and I helped by pointing to the relevant categories and blocks. I showed her the `strip rotate pixels by 1` block. I explained that this block would shift the lights on the strip, meaning that if light 1 was red and light 2 was orange, that each color would move down the strip one so light 2 would become red, light 3 would become orange, and so on.

In Miranda's performance, *The Wake* she wanted to express the pain of grieving at a wake. She wanted to physicalize the pain of having to stand for hours next to a casket while people come and offer condolences one after the other. To physicalize this emotional pain, she wanted to actuate a motor that would scratch her back with a needle whenever an audience member placed a rose by her feet. She sewed one micro:bit with a battery pack and a servo with a needle attached to the back of her dress, and had another micro:bit that would detect the presence of someone placing a flower by her feet. Miranda and I discussed different options for how to detect the flower and landed between using the light sensor and using a custom pressure pad. Miranda liked the idea of the pad because she thought there would be less error since people could place the flower anywhere on the cardboard surface instead of having to carefully place it over a light sensor. Miranda followed a tutorial I wrote for creating a cardboard and aluminum foil foot switch and she said she got it working correctly on her first attempt. I helped Miranda set up the radio sending and receiving between micro:bits, and also helped her with the timing of her servo.

Both Rena and Miranda's performances did not necessitate external technology, which enabled them to adjust their projects at home more often than other students did. They were also the

only two students with projects that necessitated building circuits to create custom switch mechanisms (the foot switch and the gloves switches). For them, it seemed that the means of audience interaction were more important to them than using sophisticated output applications.

5.2.2.2 Micro:bit Projects Integrated with External Tools

All other students (Maggie, Riley, Dana, Kyle, and Sadie) had to rely on integrating the provided materials with some external software in order to achieve their performance goals. I encouraged these decisions and supported them by helping them integrate their micro:bit with other platforms where necessary. It was important to me that students could create performances that were authentic to their expressive goals. In addition, helping them implement the more complex aspects of their projects provided me with insight about what future, modular tools could more easily support.

Because the micro:bit has Bluetooth, radio, and serial capabilities, it is possible to integrate it with just about anything, but not without developing software in less friendly environments that require programming experience. The external software I built to help students with their projects used languages such as Python, Bash, Processing. Within these languages I used both OSC and serial features. Most of the external applications we integrated were for adding audio and/or visual components to the performance that were beyond the micro:bit's capabilities. I will discuss each project and how it integrates with external software and/or hardware, for more context please refer to Table 5.1 from earlier.

In Maggie's *Untitled* performance, her main piece of external hardware was the Mindwave Mobile EEG headset. In most projects, the external application was for the output, but in her project the external hardware served as the input. I built an external application in Processing to route the EEG data from the MindWave Mobile to the micro:bit so Maggie could map the data to the angle of her servo. The Processing application I built provided a checklist of different brainwave data that Maggie could select from, indicating which EEG data she wanted to send to the micro:bit (e.g. alpha waves, beta waves, etc.). The application received OSC messages from the headset,

parsed the data depending on what Maggie selected, and sent the corresponding data over serial to the micro:bit. The creation of this application allowed Maggie to utilize the EEG data more easily, while still having control over the program design on the micro:bit. On Maggie's micro:bit she wrote code to read and parse serial data to contain the number that indicates a person's "relaxed" or "meditation" state (a number from 0–100) and mapped it to the angle of the servo (0–180°).

Figure 5.2: Processing sketch that listens for serial messages to play and stop an audio file (top), and the subsequent MakeCode blocks for sending the commands (bottom).

In Sadie's project, *Motherhood the Game*, she wanted to play an audio file on an event. Sadie wanted to play audio file that said, "Game over, you are a terrible mother," whenever she lost her balance on a balance board. On the underside of the balancing board she had a micro:bit that we set up to measure the x acceleration. If the micro:bit detected that the board was outside of the range she specified, it would send a radio message to another micro:bit plugged into the computer running the Processing sketch, and then send the serial message "play," which would trigger playback of the audio file.

Dana's *Dieting the Videogame* required integration with two pieces of software. She wanted to play an audio clip at given intervals to indicate when she should throw bean bags at the player, and she wanted to use Scratch to create her game and virtual party environment. Dana figured out how to integrate the micro:bit with Scratch on her own, but could not figure out how to play an audio file, so I explained that I built a Processing sketch for Sadie to play an audio file and she could use it for hers as well. Dana's project involved a micro:bit placed at the bottom of a cup with a small lamp over it. In her performance, small translucent chips represented "calories" and the

light sensor would detect the presence of calories placed in the cup as the light decremented. Once I showed Dana how to use conditionals to trigger different events as the light values decremented, she was able to add more on her own and customize the values that she thought made the most sense for the performance's pacing. Figure 5.3 shows a still from the performance of another student (acting as an audience member) interacting with the Scratch game and placing chips in the cup.

Figure 5.3: A student interacting with Dana's *Dieting the Video Game*.

Kyle was the most comfortable with technology and was able to do some development on his own after I gave him advice on how to design the system and got him started with code. Kyle's project, *Spam*, used a large text file full of content from spam emails and the three axes of a micro:bit accelerometer to generate random strings of text projected onto a screen. I first showed Kyle how to write a Python script that could parse a file and split text into an array using spaces as delimiters. Once we finished this, we split the array into three smaller arrays, copy and pasted the arrays into the MakeCode JavaScript editor, and set them to three different variables. This allowed Kyle to write code to map the tilt of each accelerometer axes to an index in each array and then combine words to make a random phrase. Kyle wanted to project these phrases, so I showed him how to use the terminal command `screen /dev/cu.usbmodem1422 115200` to output serial data. For his final performance he simply sent the random strings of text as serial data from the micro:bit to the computer and used the `screen` command to display it in real-time.

For Riley's *Slut/Prude* performance, the requirement that necessitated the need for outside software was her wish to have the words "slut" and "prude" play depending on the current number

of tallied votes from the numerous micro:bits in the audience. I was the one who suggested the possibility of using text-to-speech for this, and showed her what it sounded like by running it through my computer terminal. She liked that the voice was robotic and masculine and thought that it fit well with the judgemental and gendered theme of her performance. In order to achieve text-to-speech I programmed a short Python script that listened for serial data from the micro:bit and took whatever the incoming data was and sent it through text-to-speech. Riley's tallying micro:bit sent serial messages sending what word to synthesize when the tally changed.

In section 3.2.1.2, I described example project skeletons I built for the Creative++ workshop. In this study, I built higher-level applications for students that offered customizability without having to touch code. While this limited the amount they could customize the applications, it allowed students to focus on customizing their projects within the MakeCode for sending and receiving serial data to interface with the external hardware and software, supporting students' abilities to use-modify-create [64].

5.3 Discussion

Through observation of in-class activities and one-on-one interactions with students building their projects, I have highlighted different aspects of the materials and how they "closed" and "widened" the gap between students' expressive goals and the computational tools they were using [46]. In addition, I relate some of these aspects to the synthesized perspectives of beginner-friendliness from chapter 2.6 and provide examples from the data. The following aspects of the materials **closed** the gap:

- **Blocks-based editing** supported users in the interpretation of code by providing a record of legible code, and supported the generation of code by providing a catalog that users could get ideas from to discover what was possible [105].
    - "I know there's a rainbow thing on there... the 'show rainbow."'
- **Exposed circular pins on edge of board** offered the standard affordances of breaking

out pins to allow direct access to I/O [7] and enabled prototyping with alligator clips as opposed to breadboarding for simple circuits which supports flexibility and efficiency of use [75].

  * Both *The Wake* and *CU Loves Me* involved custom-built interfaces that connected to the board via alligator clips instead of soldered connections.

- **Event-handling block abstractions** enabled users to choose event blocks such as "on button A pressed do" and then put code statements inside of it that would execute automatically when the micro:bit detected that the state of button A is pressed (see Figure 6.16 for examples), aligning with the way that non-programmers expect interfaces to behave [102].

  * "I push this and it does a thing, and then I push this and it does a thing!"

- **Availability of built-in sensors** provided users access to information about acceleration, temperature, magnetic field, light, and button presses through legible blocks and removed the need for them to connect these sensors themselves [4].

  * Several students incorporated the use of the built-in sensors for their project and did not need to build custom circuits or inputs.

The following aspects of the materials **widened** the gap:

- **The lack of debugging support for physical components** led to similar debugging difficulties as other microcontroller kits that make it difficult to determine issues in software versus hardware [9]. The LED screen is intended to serve as a possible means of output debugging, but there is no ability to asynchronously print real-time values from sensors to LED screen therefore calls to print to the LED screen block the rest of the thread until the text is finished scrolling. This is an issue of lacking visibility of system status [75].

  * One student could not figure out why there were long pauses between calls to update

an LED strip and it was because they had a "show string 'Damnnnn"' after every call to set brightness and did not realize it blocked the thread.

- **Non-obvious and non-visible value ranges for different digital and analog sensors** made it difficult for students to understand what values different sensors could measure, and why they were so different. In addition, this made it difficult for students to write code that mapped inputs to outputs. This is an issue with consistency and standards, as well as help and documentation [75].
    - "I don't know what ranges to use for acceleration to know when I've lost my balance"
- **Confusing error messages** that are vague and/or use unfamiliar language did not help users recover from their errors without intervention from me (recognizing and recovering from errors) [75].
    - Students using the Adafruit Neopixel library [71] received cryptic errors when they failed to correctly initialize their strips using the programming blocks (see Figure 5.4 for an example).
- **Important blocks hidden in advanced categories** made it difficult for students to find standard blocks that are necessary for most physical computing projects.
    - This lessens the affordances of blocks in a user's discovery of what is possible when many of the blocks required for simple physical computing systems are not directly visible.
- **The lack of accessibility to interoperate with other applications** was unfortunate given that many students wanted to do this, and the ability to do so is there but is inaccessible. Micro:bits have the ability to share data over serial, radio, and bluetooth, but the development of networked distributed systems was difficult for students.

* I had to build Processing sketches for students that listened for serial messages from the micro:bit to play a simple audio file.

Figure 5.4: The resulting error message when a user writes code to update a strip that has not been initialized yet. A novice programmer has no context for what the terms "sim," "cast," and "null" mean.

My work with these performing arts students elucidated the types of projects that performers want to create, as well as offered insight on the affordances and limitations of a novice-oriented physical computing toolkit in its use for building interactive performance technology. In the next chapter I will discuss how certain designs may be able to leverage the current affordances of these tools, as well as address the existing limitations of them.

The micro:bit is incredibly useful in facilitating the creation of physical interactive systems despite some areas for improvement. However, most students wanted to be able to control output beyond the capabilities of the micro:bit. While the capabilities to connect with outside software and hardware exist and are fairly trivial for an experienced programmer to implement, they are not accessible to people without extensive programming experience. In the next chapter I will also discuss the importance of supporting modularity and interoperability between novice tools.

Chapter 6

Redesigning Physical Computing Tools for Performance

In this chapter, I discuss several programmable physical computing technologies I have designed as a response to limitations elucidated in chapters 3–5. I start with discussing, *ARcadia*, a rapid prototyping toolkit for designing electronic-less tangible interfaces. Next, I describe designs for a toolkit for building custom, physical computing based lighting controllers. Then, I describe my early-stage prototypes for direct manipulation-based programming editors for LED and DMX fixture layouts. Finally, I propose ideas for more accessible physical computing programming abstractions.

6.1 ARcadia: Rapid Prototyping of Tangible Interfaces

In 2018, I designed a programmable, browser-based toolkit for creating low-cost tangible interfaces called *ARcadia* [54]. I built ARcadia in collaboration with the MakeCode team at Microsoft Research. Our goal for ARcadia was to enable novices to design custom tangible interfaces without the need for electronics. We took several approaches to achieve this goal:

- build a low-cost toolkit that only requires basic craft materials, a webcam, and a browser,
- provide event-driven abstractions to align with how people think about designing interactive systems,
- support user understanding of system state by providing real-time feedback with graphical overlays in the browser,

- provide built-in abstractions for making common UI elements like buttons, sliders, and knobs,
- situate the toolkit in a creative context by providing audio and visual effects that users can control with their custom interfaces.

6.1.1 System Design

ARcadia is a browser-based programming editor for turning physical objects into interactive tangible interfaces. ARcadia does this through the use of tracklable fiducial markers, which I will simply refer to as *markers*. A person can easily print or draw their ARcadia markers, and when the markers are in view of a webcam the application performs image tracking. ARcadia uses ARToolkit, AR.js, and three.js libraries to support real-time image tracking and placement of graphical overlays on the markers in an augmented video feed. ARcadia's marker tracking enables the use of event-driven programming that uses changes in markers' states such as, x, y, and z position, as well as rotation. I built ARcadia using Microsoft's MakeCode framework for building block and JavaScript programming editors. Figure 6.1 shows a standard ARcadia setup that includes a cardboard interface with ARcadia marker print-outs, a webcam pointing down towards the interface, and a browser that shows the programming editor with a real-time video feed of the interface with graphical overlays.

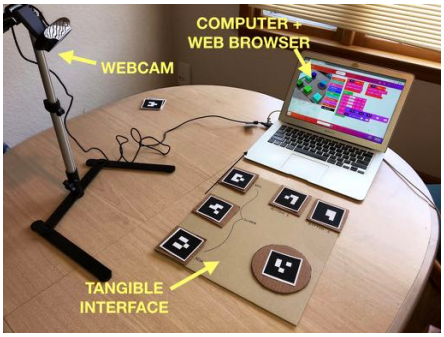

Figure 6.1: A possible ARcadia setup configuration. The video feed in the web browser shows the tangible interface captured from the webcam in real-time with graphical overlays.

ARcadia follows an event-driven programming model that lets users define event-handlers

that run when ARcadia detects that the interface has changed in a particular way. ARcadia monitors the state change of individual markers (e.g. "do something when marker is moved left"), as well as changes of markers in relation to one another (e.g. "do something when marker is touching marker "). Figure 6.2 shows the code abstractions for ARcadia event-handlers.

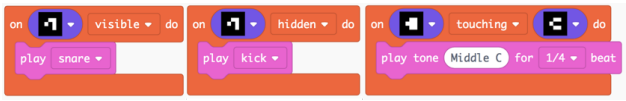

Figure 6.2: ARcadia marker events.

The tracking of multiple markers in relation to each other enables the creation of interactive tangibles like sliders, buttons, knobs (see Figure 6.3). This enables ARcadia to be used to create complex interfaces that mimic designs of familiar and standard UI controls. Sliders work by using three ARcadia markers; the outer two markers indicate the minimum and maximum points of the slider, and the inner marker is the piece that the person moves to adjust the value of the slider. A button works with just one marker and the `on visible` and `on hidden` events; a person interacts with the button by moving their hand over and away from the button. A knob also works with one marker, as ARcadia can track the rotation of the marker relative to the camera.

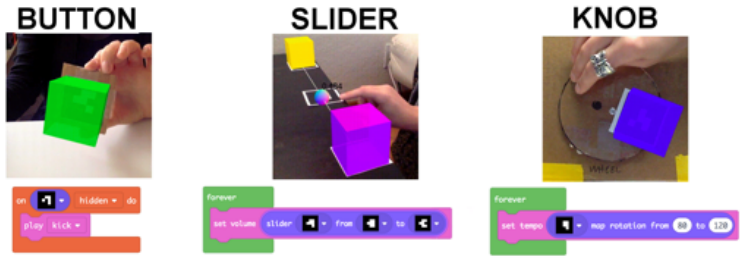

Figure 6.3: Built-in UI abstractions in ARcadia. The second column shows the tangible control with graphical overlays, the third column shows the corresponding code that uses the state of the UI to control a musical parameter.

6.1.2 Pilot Evaluation of Interaction Design Capabilities

As an initial investigation into the usability of ARcadia, I led a workshop with 120 high school girls affiliated with a group aimed at promoting increased representation for young women

in STEM. We provided each participant with a computer that had both front- and back-facing webcams, construction and craft materials like cardboard, paper, pens, scissors, and glue, and several print-outs of ARcadia markers. I gave a few demos demonstrating ARcadia being used as a musical controller and walked participants through how to create a basic example that used the `on visible` and `on hidden` events to trigger music, as well as an example that triggered a sequence of notes when two markers touched. Given that this was a STEM group, most of the girls had at least some experience with block-based programming, so after this brief tutorial we let them work for the next 45 minutes on any project of their choosing.

One group made a keyboard of ARcadia markers that contained the required notes to play Happy Birthday transposed to the key of C (see Figure 6.4). The girls designed the keyboard so that each note was triggered by an `on hidden` event, allowing them to play notes by hovering their hand over the marker. Participants also experimented with drawing markers by hand, coloring them in, and incorporating them into larger aesthetic designs.

Figure 6.4: Participant made AR keyboard (left), and participant customized markers (right).

We asked participants to write down on sticky notes things they liked about ARcadia, things they wished ARcadia could do, and ways ARcadia could be improved. Their responses about things they liked included excitement about being able to create interactive projects with paper, and the flexibility of being able to customize their interfaces:

- "I liked that we could use paper and cardboard to make musical instruments"
- "You can draw the blocks and it still works!"

- "I liked that I could easily experiment and play around with it"

Participants offered constructive criticism around the usability of ARcadia, but what we found most exciting was their interest in using ARcadia to control other outputs besides in-browser graphics:

- "I wish we could see other things we could do"
- "I wish I could connect a MIDI controller"
- "I want to put blocks on my face!"
- "I wish I could make my own Snapchat filter with it"
- "The possibilities of augmented reality in office settings"
- "To make my own game with it in the future"

6.1.3 WebRTC and OSC

In response to participants' desires to use ARcadia for "other things," as well as my experiences working with artists who want to connect interactive interfaces to external software, I added the capabilities for ARcadia to communicate with outside applications via OSC and WebRTC. As mentioned earlier, Open Sound Control (OSC) is a standard of networked communication that is widely used in computer music performance and allows for sending and receiving of data between applications [106]. WebRTC is similar in that it also enables data sharing between applications, but allows for streaming data peer to peer which is fast and does not require separate applications to be on the same local network [76]. Because WebRTC allows for direct browser to browser communication, JavaScript developers can easily modify their applications to allow for streaming data via WebRTC so that creative online applications can be parameterized and controlled using other applications. This means that creators can use ARcadia to build tangible low-cost interfaces and use it to control outside applications that support OSC and WebRTC, which is a substantial amount of creatively expressive applications. Each project generates a unique identifier called a peer ID, and messages can be sent between projects through commands like `send "hello" to "x837dkualdsc6660"`.

Figure 6.5: An ARcadia program that sends a WebRTC message to a peer. This program sends the key "loud" to a peer when a marker is hidden.

I built another MakeCode based application called *Tubular* which is an in-browser programmable application for controlling YouTube videos in real-time. Tubular contains the same block abstractions and support for WebRTC data streaming, so ARcadia and Tubular programs can share data between one another. Developers can easily import the WebRTC package I made into any browser-based MakeCode application by adding one file.

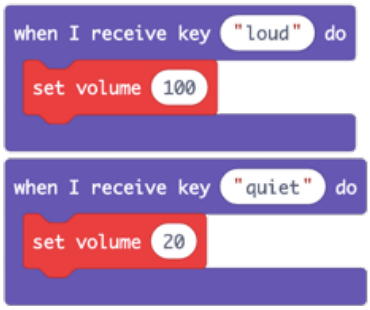

Figure 6.6: A Tubular program that receives WebRTC messages from the peer in Figure 6.5. This program sets the video playback volume to 100 when it receives the key "loud," and sets it to 20 when it receives the key "quiet."

6.1.3.1 Networked Prototypes

I have built several proof of concept projects demonstrating ARcadia's ability to control a variety of output over networks. Figure 6.7 shows a three slider RGB lighting controller I made that controls the color of a strip of LEDs. Using a box cutter I cut three vertical slits into a piece of cardboard. At the top and bottom of each slit I placed an ARcadia marker. For the "slideable" markers I glued binder clips to their backs and popped them through the slits so that I could move it up and down. The three sliders map to the red, green, and blue values of the LEDs respectively.

In the left photo of Figure 6.7 the red and blue sliders are almost maxed out, and the green slider is at zero, therefore the LEDs are set to magenta. In the photo on the right, the red and green sliders are at zero and the blue slider is almost maxed out, therefore the LEDs are blue.

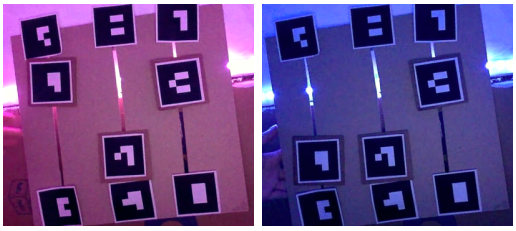

Figure 6.7: An ARcadia RGB controller with three sliders. The sliders control the red, green, and blue levels of the LED lighting shown in the photos. The lighting in the left photo is magenta, and the lighting in the right photo is blue.

In addition to the lighting controller, I have experimented with using ARcadia interfaces as VJ and DJ controllers, as well as controllers for musical effects in Processing and Max. Figure 6.8 shows two ARcadia and Tubular projects. In the left photo I am using a slider to control the playback rate of a music video, and the right photo shows a small usability study I did of a group that collaborated on a VJ interface to scrub between a music video and a scene from the movie Top Gun.

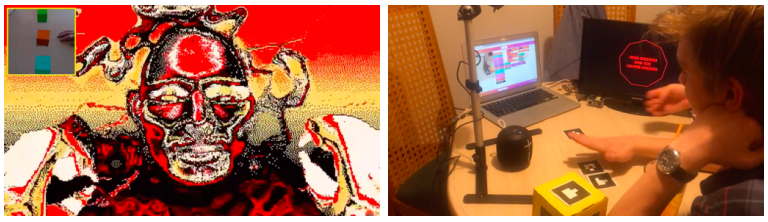

Figure 6.8: ARcadia and Tubular. Using an ARcadia slider to control playback rate of a music video (left), a group user testing WebRTC blocks for ARcadia and Tubular (right).

Figure 6.9 shows me using an ARcadia marker that adds and removes distortion on my keyboard. My keyboard was connected to my computer, and in this example I was not using WebRTC, but instead I was running a version of ARcadia on my computer that could send and receive OSC messages. As I moved my foot on and off of the marker, ARcadia sent OSC messages to a Max patch that added distortion effects to my keyboard.

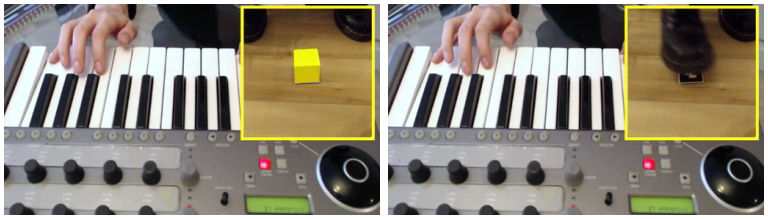

Figure 6.9: Using an ARcadia button as a distortion "pedal." When my foot covers the marker and the yellow cube disappears, distortion is enabled

6.2 LED and Stage Light Programming

Throughout these studies investigating what types of expressive projects performers want to create, there was a strong desire for people to create interactive LEDs and other lighting systems. As we have seen, LED programming is quite complex when people want to go beyond strips and nonlinear animations. In addition, interactive stage lighting is difficult and typically reserved for professionals and tech experts that are comfortable with DMX-512, electronics, and low-level programming.

6.2.1 First Iteration: Raspberry Pi and micro:bit

Typically, programming sensor-based, interactive DMX-512 fixtures requires attaching a DMX controller to a capable microcontroller, attaching sensors to the microcontroller, writing code to read sensor data from the GPIO pins, and sending DMX commands using a language like C or Python. Choosing to use the micro:bit allowed me to build a package extension for the MakeCode editor that gives users DMX-based blocks to work with, providing a front-end editor with higher-level abstractions. Because the micro:bit does not support attaching a DMX controller, I had to integrate the micro:bit with the Raspberry Pi. When the DMX blocks are compiled on the micro:bit, the blocks send serial data to the Pi. The data is formatted to include the type of DMX command as well as its arguments like channel number and value. For example, the block "create fixture 'light1' with 8 channels" would send the string "addFixture:light1,8" over serial. In order to actually execute the DMX commands, I used a ultraDMX Micro DMX controller (after

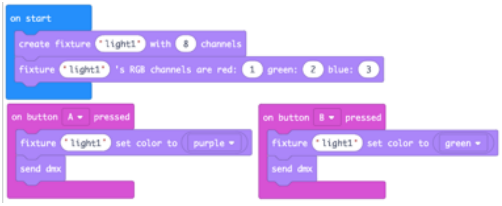

Figure 6.10: Example micro:bit program for controlling a DMX fixture. The program initializes a DMX fixture, and changes the light color between purple and green with buttons A and B.

experimenting with a few other products), Open Lighting Architecture, and Python. The Open Lighting Architecture allowed me to write a Python program that could communicate with the ultraDMX Micro DMX controller. I wrote a Python program that parses incoming serial data from the micro:bit and translates it into proper DMX commands that the controller can execute.

To test the toolkit's capabilities for building lighting controllers for live performance settings, I built several prototypes. One prototype I built allowed me to use three cardboard and foil foot switches while I was playing bass to trigger different lighting effects. Another prototype I built was a three potentiometer interface I used to control the RGB levels of the stagelighting during a band's live show (see Figure 6.11).

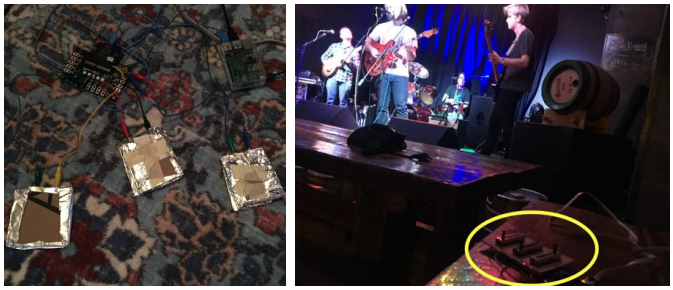

Figure 6.11: Two low-cost stage lighting controllers I built. The left photo shows three pressure activated foot pedals that affect the lighting, the right photo shows an RGB controller for controlling stage lighting in a bar during a show.

In addition, I hosted a small workshop with three participants that had little to no programming experience. I walked everyone through how to use the micro:bit's embedded sensors to control stage lighting, and then helped them each build custom circuits using copper tape. One participant put copper tape on each of her drum sticks so when they were touching the lights would animate

through a sequence of colors, and when they were not touching the lights would remain static.

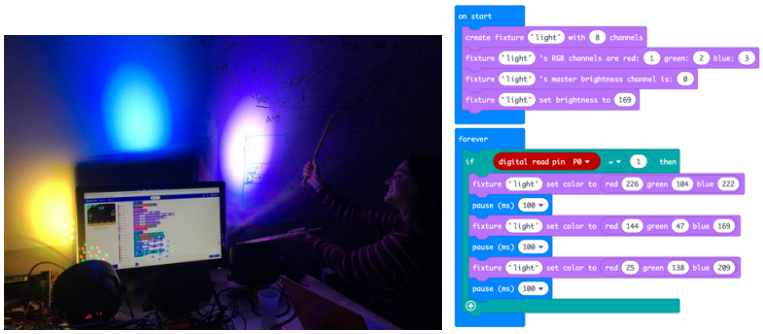

Figure 6.12: A participant-made micro:bit project. The system detects when two drumsticks are touching and changes the stage lighting colors.

One of the limitations of this first design iteration was that it relies on the micro:bit for the front-end interface, and the Raspberry Pi for the back-end DMX controls. This raises issues of cost and also increases the possibility of system failure when programs on either platform are not run in the correct order. In addition, the micro:bit does not have built-in WebRTC or OSC capabilities so it cannot communicate with the other existing applications I have built such as ARcadia, BlockyTalky, or Tubular.

6.2.2 Second Iteration: BlockyTalky Raspberry Pi

The second iteration of my programmable lighting toolkit uses BlockyTalky 3, removing the need for the micro:bit. BlockyTalky 3 uses the MakeCode editing platform as a means to program the Raspberry Pi, instead of the proprietary software we originally used in the version of BlockyTalky from Chapter 3 [101]. This meant that although I was not using the micro:bit anymore, I was still working within the MakeCode framework and could use the same custom blocks I created. Therefore, users can program the lights using the same blocks as before, but instead of the messages sent via serial between the micro:bit and Pi, the compiled program makes direct calls to a NodeJS based library for DMX communication. I transitioned to using the Node library over the Open Lighting Architecture because the MakeCode framework is written in TypeScript, therefore it was easier to integrate within the blocks' back-end functionality. In addition, BlockyTalky 3

supports WebRTC and can now integrate with other WebRTC applications. BlockyTalky 3 also supports integration with GrovePi sensors.

6.2.3 Direct Manipulation Lighting Programming

The aforementioned designs allowed for easier creation of professional stage lighting systems that can be controlled in real-time based on physical input. In the small lighting workshop that I mentioned, participants with little to no programming experience were able to create physically interactive lighting systems in under three hours. When I tried to build a Raspberry Pi interactive DMX system for the first time it took me a week to learn how to implement it, and I have a background in computer science. Despite improving the low-level aspects of DMX programming, the programming challenges of designing lighting animations are still the same as the ones I identified in chapter 4.

To address this, I am working on designs for a domain-specific visual editor that allows for creating layouts of lighting fixtures (eventually any arbitrary DMX fixtures) that generate blocks that users can add to their project code. This editor can be accessed from the DMX blocks category on BlockyTalky 3; there is a button that opens into a larger custom editor. Figure 6.13 shows a screenshot of the current state of the lighting editor. In the editor, people can use the standard methods for creating light shows by creating scenes and patterns. A *scene* is a saved preset for the appearance or actions of a set of fixtures. A *pattern* or *sequence* is a group of scenes with functions that define transitions between scenes [63]. In this screenshot, I have created a lighting layout of four RGB lights. On the right side there is a pattern creator. I have created a pattern called "Pattern 0" that contains 3 scenes with 500ms transitions between them.

Once a user is done defining their layout, scenes, and patterns, they can press "Go back" to return to the MakeCode blocks editor. The editor automatically generates programming blocks based on the user's lighting layout they defined. Figure 6.13 also shows the automatically generated blocks that correspond to the edits made in the visual editor.

A person can now use these abstractions to cue and loop different lighting animations, instead

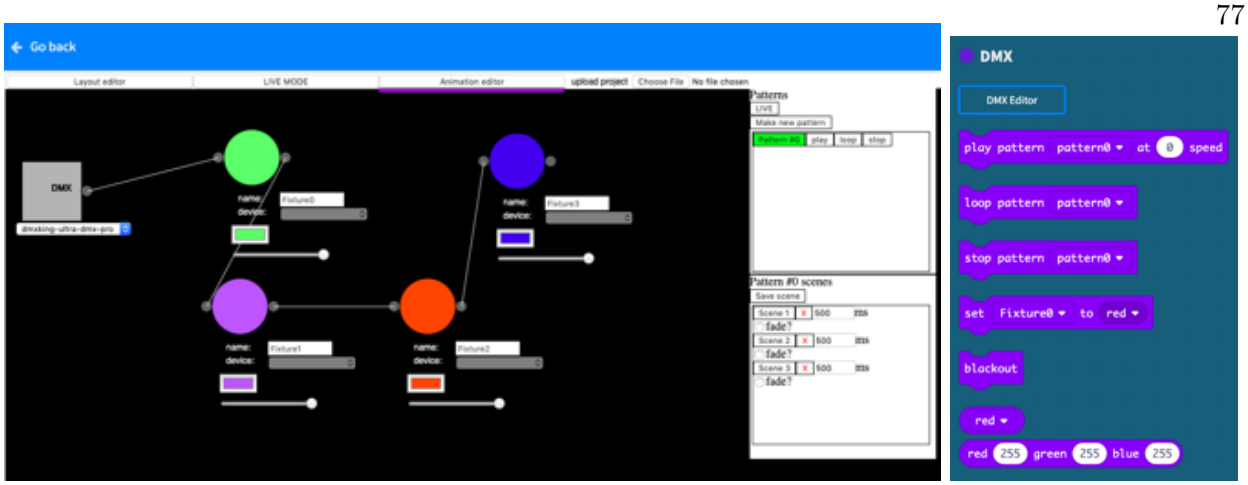

Figure 6.13: Visual editor for making interactive DMX-512 fixtures (left), and the autogenerated MakeCode blocks (right).

of programming them line-by-line using blocks. For example, now a person can write a program like the one in Figure 6.14, instead of the one shown in Figure 6.12.

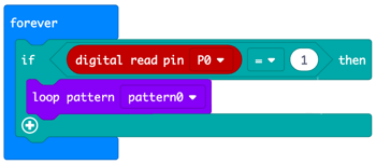

Figure 6.14: A shorter version of the program in Figure 6.12 using autogenerated blocks from the lighting editor.

Although I have not begun implementing a working prototype for the LED version of this visual editor, Figure 6.15 shows an imagining of what it might look like. The left photo shows what a user could create by adding LEDs to the layout, moving them around, and connecting the wires between other LEDs on a given pin based on how they connect it to their micro:bit. A person could then create higher-level abstractions of groups of LEDs based on their physical locations, and not by their positions on the strips. The right photo shows a possible grouping a user might want to make for their LED layout, notice how the circles encompass LEDs on different strips in a nonlinear fashion. Similar to the DMX editor, this would automatically generate block abstractions that a person could use in their code.

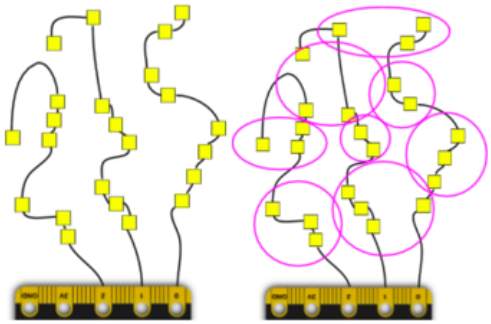

Figure 6.15: Prototype of a visual editor for designing complex interactive layouts of LEDs.

6.3 Ideas for Improved Physical Computing Abstractions

The micro:bit has several built-in abstractions for handling I/O events. Figure 6.16 shows the four available event-handlers that do something when: a button is pressed, the accelerator meets certain criteria (e.g. `on shake`), a sensor attached to P0 reads `HIGH` (`on pressed`), a sensor attached to P0 reads `LOW` (`on released`).

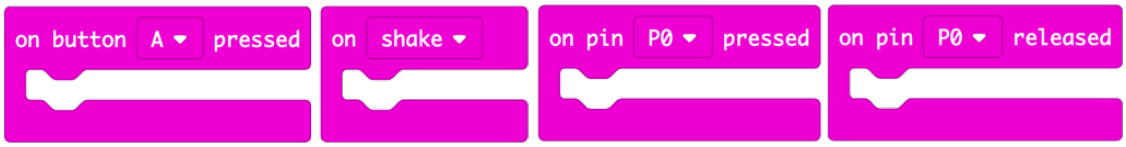

Figure 6.16: Examples of the micro:bit's built-in event-handler abstractions.

However, these event-handlers do not support doing something repeatedly *while* a condition is met. The `on button pressed` block will run whatever code blocks are nested inside of it once the board detects a change in the button's state from `LOW` to `HIGH`. If a user wants to run a chunk of code repeatedly while an event occurs (a common need for interactive projects demonstrated in this thesis) the program becomes increasingly more complex (see Figure 6.17).

In this example, the user now has to understand how to use the `forever` block mechanism which is similar to Arduino's `loop()` function. In order to do something repeatedly, they must create a conditional inside of the `forever` block that executes one series of instructions if a condition is met, and a different series of instructions if that condition is not met. This is a simple if-

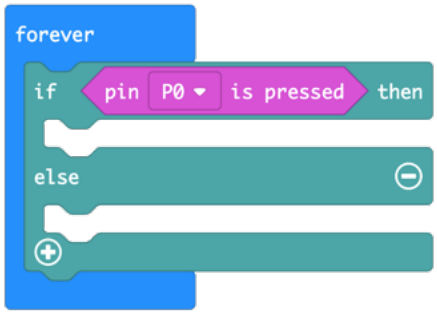

Figure 6.17: Example block code for a custom "while" event-handler. This program contains a conditional within a forever loop that would accomplish executing code while P0 is pressed, and do something else when it is released.

else mechanism which is trivial to experienced programmers who have memorized structures and solutions to problems they have encountered repeatedly, however, a novice has no context for this [10]. Novice programmers frequently misunderstand how loops and conditionals work and have trouble applying them even in block-based physical computing projects [48]. In addition, novices using these tools for the first time tend to rely on event-handlers when they can, either by choice or by instruction from facilitators or online tutorials, and then have no context for building events using other mechanisms when they find that they cannot achieve their goal using the built-in abstractions. With the addition of new programming abstractions for more customizable event-handling, the micro:bit could easily provide more support for designing more various interactions. Users' needs and wants to have interactions that do things repeatedly is extremely common, and the tools should not make it extraordinarily difficult to accomplish when the mechanisms are already there for similar event abstractions.

In addition, the participants I have worked with typically wanted to build projects that map the state of an input to a given output. Depending on the project goals, participants sometimes wanted these mappings to be continuous, and others wanted more discrete mappings. The difference is that *continuous mapping* is a function that maps input to output over a complete range of values, whereas *discrete mapping* uses conditionals to map input to output at specific intervals. Figure 6.18 shows the differences between the two methods. In these examples, the discrete program on

the right might be beneficial for a project where it is important for the servo to only turn to specific angles instead of being able to sweep through every possible angle. Through working with people I have found that oftentimes they want to use a continuous mapping, but end up relying on discrete, conditional mapping techniques because they are unable to figure out how, which is interesting because the discrete example is much longer and contains more conditionals. Let us walk through the process of what it would be like to design a continuous mapping mechanism for the first time. Assuming a person does not want to implement their own range mapping function, the best way

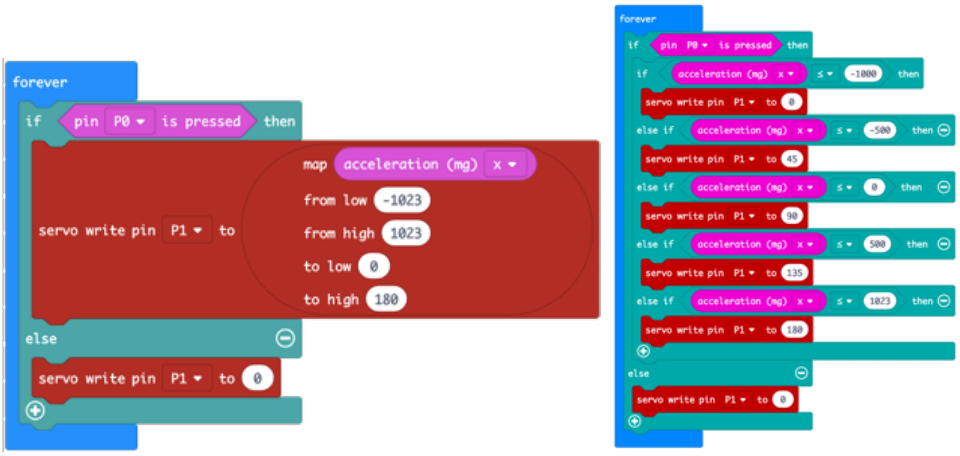

Figure 6.18: Micro:bit programs that use I/O event-handling and value mapping. The left program continuously maps x-acceleration to servo angle while someone presses a button, the right program performs a similar task but uses discrete thresholding with conditionals instead of continuous mapping.

to do this in MakeCode is to use the `map` block. The `map` function takes five arguments: the input value, the lowest and highest possible values of the input value, and the lowest and highest possible values of output. Figure 6.18 maps x acceleration to servo angle, whose possible value ranges are -1023–1023 and 0–180° respectively. In order to find the `map` function, one must already need to know to look in the Pins category which is hidden and nested within the Advanced category (many standard physical computing blocks in MakeCode are automatically hidden and tucked inside of the Advanced category). I would argue that the `map` function might be better suited in the Math category rather than the Pins category. Nonetheless, upon finding this category, a person must look at the `map` block and identify it as an abstraction that is useful for their project. If you hover

over the `map` block to see the tooltip to learn what it does it says, "Map a number from one range to another. That is, a value of "from low" would get mapped to "to low," a value of "from high" to "to high", values in-between to values in-between, etc." This tooltip is vague and uses mathematical language like "map" that has a different context to novices. It also does not provide information for why someone might use it or how, nor does it mention that this is a non-constraining function and can therefore return values outside of the specified ranges (users can add a `constrain` block to fix this). In addition, it assumes that the user knows the value ranges of the inputs and outputs they want to use. The built-in sensors on the board contain a variety of different value ranges in different units: acceleration of each axis ranges from -1023–1023, the light sensor ranges from 0–255, buttons return 0 or 1, rotation returns -180–180, the compass returns 0–359 (-1003 when it is "not ready"), and packet signal strength returns -128– -42. Some sensors do not even have documentation of the possible ranges, like the temperature sensor and magnetometer, so a person must establish their own reasonable ranges for them depending on the context. All in all, the `map` function is incredibly useful but confusing for novices.

Figure 6.19 shows a rough sketch I made for a MakeCode programming abstraction. Although this design has room for improvement, it offers users the ability to define more tailored event-handlers beyond the ones shown in Figure 6.16. This design would allow users to customize several parameters:

- The timing of the event-handler. A more customizable event-handler could let users define code segments that could run repeatedly (unlike the options in Figure 6.16), and by adding pauses users could write events that, for example, repeat every 2 seconds while an event occurs.

- The input that the event-handler is monitoring. MakeCode supports events that trigger on button presses, pin changes from `LOW` to `HIGH` or vise versa, and hardcoded accelerometer events (like `on shake`, `on tilt left`, etc.). A new design could support the arbitrary input of different built-in or custom inputs and sensors.

- The value that the event-handler is waiting for the input's state to change to. Based on the sensory input a user chooses, they can then choose a value or a range of values that the event should trigger on. For example, a user could define an event like "when the temperature is in between 15° to 27° Celsius, repeatedly run the following code."

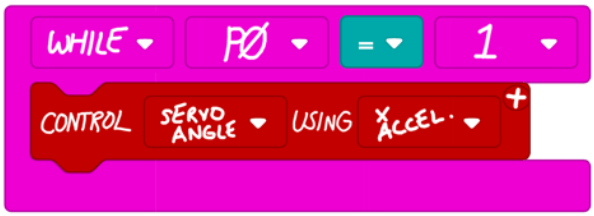

Figure 6.19: A rough design idea for more customizable event-handling and I/O mapping that would replace the program in Figure 6.18.

In addition, the sketch includes ideas for changing the language and structure for mapping inputs to outputs. This block could encapsulate the values of known inputs and sensors. The "+" symbol shown in the Figure could open up to another editor where users could select more specific ranges that they want their inputs and outputs to map over (it could default to all possible ranges) and whether they want the mapping results to be constrained.

6.4 Discussion

Based on what I have learned around what closes and widens the gap between artists' goals and the tools they use, in this chapter I presented toolkits, prototypes, and proposed designs for eliciting more powerful and accessible programming tools for performance.

The design of ARcadia supports novices in the rapid prototyping and creation of interactive interfaces by removing the need for electronics. Adding WebRTC and OSC functionality to ARcadia extends its abilities to be used in creative contexts, and I demonstrated some of the possibilities through student-made and personal projects. My work supporting physical computing based stagelighting systems demonstrates steps towards providing artists means to build interactive systems using complicated, professional hardware. Furthermore, my designs for visual LED and

DMX fixtures address programming challenges that arose in Chapter 4 and provide possible designs for mitigating these challenges. Finally, I proposed new abstractions for the micro:bit MakeCode platform for simplifying designs related to physical computing. While the MakeCode editor makes it easier for novices to integrate electronics, the designs I have discussed offer improvements in the realm of programming semantics and abstractions.

Chapter 7

Conclusion

7.1 Recapitulation

I have presented my work investigating beginner-friendly physical computing technologies' applications to performance, as well as how future designs of these technologies can improve usability and empower creators in accomplishing their creative goals.

Chapter 1–2: Discussed the motivations and relevant literature for this research.

Chapters 3–5: Described four studies that investigated artist-technologist collaborations as well as novices' experiences working with physical computing technology.

Chapter 6: "Redesigning Physical Computing Tools" described a series of technologies I have designed as a response to my experiences working with non-technical artists, as well as proposed design ideas that would hopefully make these tools more accessible.

7.2 Future Work

Many of the tools and designs I have described require more thorough evaluations around their usability. I plan to continue iterating on my tools and prototypes by investigating how artists incorporate these tools into their practice, and what improvements I can make to make these tools more supportive for creative expression. Future evaluations could investigate what language is most natural for novices in thinking about designing interactive physical computing systems.

7.3 Empowering Artists to Exercise Their Creative Impulses

In the words of Patti Smith, "Every human being has a creative impulse, and we all have the right to exercise this creative impulse" [62].

DIY and maker technologies have been inspiring communities to take control over the production of their own projects, a power shift rooted in feminist ideals. While these technologies have done a lot to even the playing field for novices and creators, they still have a long way to go for enabling DIY artists to author their own expressive interactive systems for performance. These beginner-friendly tools have a great amount of potential to be useful and interesting tools for adult artists, but right now are primarily designed for children making basic interactive projects. I presented studies in which I explored the applications of these technologies in art and performance, demonstrating what happens when beginner-technologists try to implement their own designs for interactive systems. In doing so I discussed the gaps that exist between artists' goals and available technologies, and the challenges artists face when working to implement these goals. In addition, I discussed designs I have created in response to weaknesses in current tools, as well as possible design ideas and prototypes for building more accessible tools for DIY art and performance technology. My hope is that the work I have presented in this thesis inspires designers to create a more accessible tools for artists, and to really consider what it means to be accessible, in order to empower artists to exercise their creative impulses.

Bibliography

[1] Leon Anderson. Analytic Autoethnography. Journal of Contemporary Ethnography, 35(4):373–395, 2006.

[2] Catherine Ashcraft, Elizabeth Eger, and Michelle Friend. Girls in IT: The Facts. National Center for Women & Information Technology, 2012.

[3] Marije Baalman. Wezen — chrysalis. https://marijebaalman.eu/projects/wezen-chrysalis.html.

[4] Thomas Ball, Jonathan Protzenko, Judith Bishop, Michał Moskal, Jonathan De Halleux, Michael Braun, Steve Hodges, and Clare Riley. Microsoft Touch Develop and the BBC micro:bit. 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pages 637–640, 2016.

[5] Alan F Blackwell. First Steps in Programming: A Rationale for Attention Investment Models. Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), 2002.

[6] Paulo Blikstein. Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design. Foundations and Trends in Human-Computer Interaction, 9(1):1–68, 2015.

[7] Paulo Blikstein and Arnan Sipitakiat. QWERTY and The Art of Designing Microcontrollers for Children. 10th International Conference on Interaction Design and Children, pages 234–237, 2011.

[8] Elizabeth Bonsignore, June Ahn, Tamara Clegg, Mona Leigh Guha, Jason C. Yip, Allison Druin, and Juan Pablo Hourcade. Embedding Participatory Design into Designs for Learning: An Untapped Interdisciplinary Resource? Computer-Supported Collaborative Learning Conference, CSCL, pages 549–556, 2013.

[9] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. CHI '16 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 3485–3497, 2016.

[10] Benedict Du Boulay. Some difficulties of learning to program. Journal of Educational Computing Research, 2(1):57–73, 1986.

[11] Matt Bower and Daniel Sturman. What are the educational affordances of wearable technologies? Computers and Education, 88:343–353, 2015.

[12] Urie Bronfenbrenner and Pamela L. Morris. The Bioecological Model of Human Development. John Wiley & Sons, Inc., 2007.

[13] Leah Buechley, Mike Eisenberg, Jaime Catchen, and Ali Crockett. The lilypad arduino: Using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. Proceedings of the 2008 CHI Conference on Human Factors in Computing Systems - CHI '08, pages 423–432, 2008.

[14] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. Generation CS: The Growth of Computer Science. ACM Inroads, 8(2):44–50, 2017.

[15] Linda Candy and Ernest Edmonds. Modeling co-creativity in art and technology. In Proceedings of ACM Creativity & Cognition 2002, pages 134–141, 2002.

[16] Lori Carter. Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. SIGCSE '06 Proceedings of the 37th SIGCSE technical symposium on Computer science education, pages 27–31, 2006.

[17] Red Chidgey. Developing communities of resistance? maker pedagogies, do-it-yourself feminist, and diy citizenship. In Matt Ratto and Megan Boler, editors, DIY Citizenship: Critical Making and Social Media,. The MIT Press, 2014.

[18] Perry Cook. Principles for designing computer music controllers. Proceedings of the 2001 International Conference on New Interfaces for Musical Expression, pages 1–4, 2001.

[19] Adrienne Decker, Kurt Eiselt, and Kimberly Voll. Understanding and Improving the Culture of Hackathons: Think Global Hack Local. Proceedings - Frontiers in Education Conference, FIE, 2015.

[20] Kayla DesPortes and Betsy DiSalvo. Trials and Tribulations of Novices Working with the Arduino. Proceedings of the ACM SIGCSE International Computing Education Research, pages 219–227, 2019.

[21] Betsy Disalvo and Kayla Desportes. Participatory Design for Value-Driven Learning. In Betsy DiSalvo, Jason Yip, Elizabeth Bonsignore, and Carl DiSalvo, editors, Participatory Design for Learning: Perspectives from Practice and Research, pages 177–191. Routledge, New York, 2017.

[22] Betsy DiSalvo and Carl DiSalvo. Designing for Democracy in Education: Participatory Design and the Learning Sciences. Proceedings of International Conference of the Learning Sciences, ICLS, pages 793–799, 2014.

[23] Daniel C. Edelson and Diana M. Joseph. The Interest-Driven Learning Design Framework: Motivating Learning through Usefulness. Proceedings of the 6th international conference on Learning sciences, pages 166–173, 2004.

[24] Ron Eglash, Audrey Bennett, Casey O'Donnell, Sybillyn Jennings, and Margaret Cintorino. Culturally Situated Design Tools: Ethnocomputing from Field Site to Classroom. American Anthropologist, 108(2):347–362, 2006.

[25] Ron Eglash, Juan E. Gilbert, and Ellen Foster. Toward Culturally Responsive Computing Education. Communications of the ACM, 56(7):33–36, 2013.

[26] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. How to Design Programs: An Introduction to Computing and Programming. The MIT Press, 2008.

[27] Ann Ferguson, Rosemary Hennessy, and Mechthild Nagel. Feminist Perspectives on Class and Work. The Stanford Encyclopedia of Philosophy (Spring 2019 Edition), 2019.

[28] Rebecca Fiebrink. Wekinator example bundles.

[29] Rebecca Anne Fiebrink, Daniel Trueman, Cameron Britt, Michelle Nagai, Konrad Kaczmarek, Michael Early, MR Daniel, Anne Hege, and Perry Cook. Toward Understanding Human-Computer Interaction in Composing the Instrument. International Computer Music Association, pages 135–142, 2010.

[30] Gerhard Fischer and Kumiyo Nakakoji. Computational Environments Supporting Creativity in the Context of Lifelong Learning and Design. KBS Journal '97, pages 1–13, 1997.

[31] Allan Fisher and Jane Margolis. Unlocking the Clubhouse: The Carnegie Mellon Experience. inroads SIGCSE Bulletin, 34(2):79–83, 2002.

[32] Ellen K. Foster. Claims of equity and expertise: Feminist interventions in the design of diy communities and cultures. Design Issues, 35(4):33–41, 2019.

[33] Sarah Fox, Rachel Rose Ulgado, and Daniela Rosner. Hacking Culture, Not Devices: Access and Recognition in Feminist Hackerspaces. CSCW '15 - The 18th ACM Conference on Computer-Supported Cooperative Work and Social Computing, pages 56–68, 2015.

[34] Paulo Freire. Pedagogy of the Oppressed. Bloomsbury Publishing, 2000.

[35] Jill Gerhardt-Powals. Cognitive Engineering Principles for Enhancing Human-Computer Performance. International Journal of Human-Computer Interaction, 8(2):189–211, 1996.

[36] William Goddard, Richard Byrne, and Florian Floyd Mueller. Playful Game Jams: Guidelines for Designed Outcomes. ACM International Conference Proceeding Series, 2014.

[37] Google. Women Who Choose Computer Science — What Really Matters. pages 1–8, 2014.

[38] Shuchi Grover and Roy Pea. Computational Thinking in K-12: A Review of the State of the Field. Educational Researcher, 42(1):38–43, 2013.

[39] Mark Guzdial. Five principles for programming languages for learners. https://cacm.acm.org/blogs/blog-cacm/203554-five-principles-for-programming-languages-for-learners/fulltext. Accessed: 2019-09-13.

[40] Mark Guzdial. Why i say task-specific programming languages instead of domain-specific programming languages. https://computinged.wordpress.com/2019/05/27/why-i-say-task-specific-programming-languages-instead-of-domain-specific-programming-languages/. Accessed: 2019-09-13.

[41] Idit Harel and Seymour Papert. Constructionism, 1991.

[42] Joanie Harmon. Educating for equity and access in computer science, 2018.

[43] Lyndsey Havens. Women roar onstage but barely exist in the sound booth. Chicago Tribune, 2015. https://www.chicagotribune.com/entertainment/music/ct-recording-engineers-0924-20150923-story.html.

[44] Imogen Heap. the gloves. http://www.imogenheap.co.uk/thegloves/, 2012.

[45] Cindy E Hmelo and Guzdial Mark. Of Black and Glass Boxes: Scaffolding for Doing and Learning. ICLS '96 Proceedings of the 1996 international conference on Learning sciences, pages 128–134, 1996.

[46] Edwin L Hutchins, James D Hollan, and Donald A Norman. Direct Manipulation Interfaces. Human-Computer Interaction, 1(4):311–338, 1985.

[47] Jennifer Jacobs and Leah Buechley. Codeable Objects: Computational Design and Digital Fabrication for Novice Programmers. CHI '13 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 1589–1598, 2013.

[48] Karen H Jin. Coding while Making: using Blocks Language in a Physical Computing Setting. BLOCKS+, pages 110–111, 2018.

[49] Andrew Johnston. Keeping research in tune with practice. In Linda Candy and Sam Ferguson, editors, Interactive Experience in the Digital Age,, chapter 4. Springer, 2014.

[50] Sergi Jordà. On stage: the reactable and other musical tangibles go real. International Journal of Arts and Technology, 1(3-4):268–287, 2008.

[51] Yasmin Kafai, Chris Proctor, and Debora Lui. From Theory Bias to Theory Dialogue: Embracing Cognitive, Situated, and Critical Framings of Computational Thinking in K-12 CS Education. Proceedings of the 15th. ACM International Computing Education Research conference (ICER '19), pages 101–109, 2019.

[52] Caitlin Kelleher. Barriers to Programming Engagement. Advances in Gender and Education, 1:5–10, 2009.

[53] Annie Kelly, Lila Finch, Monica Bolles, and R Benjamin Shapiro. BlockyTalky: New programmable tools to enable students' learning of networks. International Journal of Child-Computer Interaction, 18:8–18, 2018.

[54] Annie Kelly, R Benjamin Shapiro, Jonathan de Halleux, and Thomas Ball. ARcadia: A Rapid Prototyping Platform for Real-time Tangible Interfaces. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pages 409:1—-409:8, 2018.

[55] Nazish Zaman Khan and Andrew Luxton-Reilly. Is computing for social good the solution to closing the gender gap in computer science? ACM International Conference Proceeding Series, 01-05-February-2016(February 2016), 2016.

[56] Amy Ko. Programming languages are the least usable, but most powerful human-computer interfaces ever invented. https://blogs.uw.edu/ajko/2014/03/25/programming-languages-are-the-least-usable-but-most-powerful-human-computer-interfaces-ever-invented/. Accessed: 2019-08-29.

[57] Karen Johanne Kortbek and Kaj Grønbæk. Communicating art through interactive technology: New approaches for interaction design in art museums. ACM International Conference Proceeding Series, 358:229–238, 2008.

[58] Brittany Ann Kos. The unique hackathon experience. University of Colorado Boulder, 2016.

[59] Tomaž Kosar, Marjan Mernik, and Jeffrey C Carver. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical Software Engineering, 17(3):276–304, 2012.

[60] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, and Maria João Varanda. Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. Computer Science and Information Systems, 7(2):247–264, 2010.

[61] Celine Latulipe. The Value of Research in Creativity and the Arts. In Proceedings of ACM Creativity & Cognition 2013, 2013.

[62] Louisiana Channel. Patti smith: We all have a creative impulse, February 2015. [Online; posted 8-February-2015].

[63] Lutron. Dmx-512 fundamentals.

[64] N Lytle, V Cateté, D Boulden, Y Dong, J Houchins, A Milliken, A Isvik, D Bounajim, E Wiebe, and T Barnes. Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. 2019 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2019, pages 395–401, 2019.

[65] Make and Intel. An In-depth Profile of Makers at the Forefront of Hardware Innovation. Technical report, 2013.

[66] C. Dianne Martin. Draw A Computer Scientist. SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education), 36(4):11–12, 2004.

[67] Julie Martin. A brief history of experiments in art and technology. IEEE Potentials, 34(December):13–19, 2015.

[68] David A Mellis and Leah Buechley. Scaffolding Creativity with Open-Source Hardware. Creativity and Cognition, 2011.

[69] David A. Mellis and Leah Buechley. Case Studies in the Personal Fabrication of Electronic Products. Designing Interactive Systems, pages 268–277, 2012.

[70] David A Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices. Designing Interactive Systems, pages 1270–1281, 2016.

[71] Microsoft MakeCode. Neopixel driver. https://makecode.microbit.org/pkg/microsoft/pxt-neopixel.

[72] Megan Molenti and Adam Rogers. The actual science of james damore's google memo, August 2017. [Online; posted 15-August-2017].

[73] Brad Myers, John F. Pane, and Amy Ko. Natural Programming Languages and Environments. Communications of the ACM, 47(9):47–52, 2004.

[74] National Center for Women & Information Technology. By the numbers, 2018. www.ncwit.org/bythenumbers.

[75] Jakob Nielsen. Enhancing the Explanatory Power of Usability Heuristics. CHI '94 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1994.

[76] Jukka K. Nurminen, Antony J.R. Meyn, Eetu Jalonen, Yrjo Raivio, and Raúl García Marrero. P2P Media Streaming with HTML5 and WebRTC. INFOCOM Demo/Poster Session, pages 63–64, 2013.

[77] Elizabeth Gillette Offensend. Crafting a Space: A Feminist Analysis of the Relationship Between Women, C Thesis, Portland State University, 2012.

[78] Paul G. Oliver. The DIY artist: issues of sustainability within local music scenes. Management Decision, 2010.

[79] Paul G. Oliver and Gill Green. Adopting new technologies: Self-sufficiency and the DIY artist. UK Academy for Information Systems Conference Proceedings, 2009.

[80] By Seymour Papert. An Evaluative Study of Modern Technology in Education. MIT Logo Memo 26, 1976.

[81] Seymour Papert. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, 1980.

[82] Elizabeth Patitsas, Michelle Craig, and Steve Easterbrook. How CS Departments are Managing the Enrolment Boom: Troubling Implications for Diversity. Proceedings of the 2016 Research on Equity and Sustained Participation in Engineering, Computing, and Technology, pages 1–2, 2016.

[83] Roy D Pea. The aims of software criticism: Reply to Professor Papert. Educational Researcher, 16(5):4–8, 1987.

[84] Kylie Peppler. STEAM-Powered Computing Education: Using E-Textiles to Integrate the Arts and STEM. Computer, 46(9):38–43, 2013.

[85] Radia Perlman. TORTIS – Toddler's Own Recursive Turtle Interpreter System. Massachusetts Institute of Technology A.I. Laboratory: LOGO Memo 9, 1974.

[86] Mitchel Resnick, John Maloney, Andrés Monroy, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for All. Communications of the ACM, pages 60–67, 2009.

[87] Mitchel Resnick and Stephen Ocko. Lego/logo: Learning through and about design. In Idit Harel and Seymour Papert, editors, Constructionism,. Norwood, NJ: Ablex Publishing, 1991.

[88] Mitchel Resnick and Brian Silverman. Some Reflections on Designing Construction Kits for Kids. Conf. Interaction Design and Children (IDC 05), pages 117–122, 2005.

[89] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2):137–172, 2003.

[90] Rosemary L Sallee. Femmage and the DIY Movement: Feminism, Crafty Women, and the Politics of Gende Dissertation, University of New Mexico, 2016.

[91] Albrecht Schmidt. Increasing Computer Literacy with the BBC micro:bit. IEEE Pervasive Computing, 15(2):5–7, 2016.

[92] Sue Sentance, Jane Waite, Steve Hodges, Emily MacLeod, and Lucy Yeomans. "Creating cool stuff": Pupils' experience of the BBC micro:bit. ACM Technical Symposium on Computer Science Education, pages 531–536, 2017.

[93] R Benjamin Shapiro, Matthew Ahrens, Rebecca Fiebrink, and Annie Kelly. BlockyTalky: A Physical and Distributed Computer Music Toolkit for Kids. Proceedings of the 17th International Conference on New Interfaces for Musical Expression, 2015.

[94] R Benjamin Shapiro, Annie Kelly, Matthew Ahrens, Benjamin Johnson, and Heather Politi. BlockyTalky: Tangible Distributed Computer Music for Youth. The Computer Music Journal. MIT Press., 41(2):1–30, 2017.

[95] Jennifer G Sheridan, Nick Bryan-Kinns, and Alice Bayliss. Encouraging Witting Participation and Performance in Digital Live Art. Proceedings of the HCI'07 Conference on People and Computers XXI, 1:2, 2007.

[96] Bruce Sherin, Andrea A. diSessa, and David Hammer. Dynaturtle revisited: Learning physics through collaborative design of a computer model. Interactive Learning Environments, 3(2):91–118, 1993.

[97] Elliot Soloway. Learning to Program = Learning to Construct Mechanisms and Explanations. Communications of the ACM, 29(9):850–858, 1986.

[98] James C. Spohrer and Elliot Soloway. Novice Mistakes: Are the Folk Wisdoms Correct? Communications of the ACM, 29(7):624–632, 1986.

[99] Monica Tan. Steve albini: the internet has solved the problem with music, 2014. https://www.theguardian.com/music/2014/nov/17/steve-albini-at-face-the-music-how-the-internet-solved-problem-with-music.

[100] Joy Teague. Women in Computing: What brings them to it, what keeps them in it? Gates, 5(1):45–49, 2000.

[101] Will Temple. First steps with blockytalky 3.

[102] Franklyn Turbak, Mark Sherman, Fred Martin, David Wolber, and Shaileen Crawford Pokress. Events-First Programming in App Inventor. Journal of Computing Sciences in Colleges, 29(6):81–89, 2014.

[103] Judy Wajcman. Feminism Confronts Technology, 1991.

[104] Susan Weinschenk and Dean T. Barker. Designing Effective Speech Interfaces. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[105] David Weintrop and Uri Wilensky. How Block-based Languages Support Novices: A Framework for Categorizing Block-based Affordances. Journal of Visual Languages and Sentient Systems, pages 92–100, 2017.

[106] Matthew Wright. Open Sound Control: an enabling technology for musical networking. Organised Sound, 10(3):193–200, 2005.

[107] Clement Zheng, Jeeeun Kim, Daniel Leithinger, Mark D. Gross, and Ellen Yi-Luen Do. Mechamagnets: Designing and fabricating haptic and functional physical inputs with embedded magnets. TEI 2019 - Proceedings of the 13th International Conference on Tangible, Embedded, and Embodied Interaction, pages 325–334, 2019.