ABSTRACT

Title of Thesis: HOW CAN DEBUGGING WITH PHYSICAL COMPUTING BE MORE PLAYFUL FOR CHILDREN?

Danyi Zeng

Master of Science, 2024

Thesis Directed By: Assistant Professor Caro Williams-Pierce

College of Information Studies

In response to the ongoing call for the education of computational thinking, I explored how debugging activities in a physical computing environment can be more playful and learnable for children. While a lot of studies have addressed the importance of debugging in generic programming learning, the benefits and challenges of physical computing implementation in classrooms, or the potential of playfulness in STEM education, few research focused on an

interdisciplinary conversation that sought design solutions to bring playfulness into the learning experience and to improve the user experience cohesively. In this study, based on a synthetical understanding of the relevant studies from computer science, human-computer interaction, and education, I situated the concept of *fragile knowledge* into the complex, multiple-object environment of physical computing. Accordingly, I designed two debugging projects on micro:bit for 8 participants at KidsTeam at the University of Maryland to understand their intuitive approaches to debugging in the physical computing environment. I analyzed the video data of the two 90-minute sessions and applied semantic coding to examine and compare the participants' earning experiences, including typical progress and failures. The qualitative findings revealed: 1) the differentiation in the process of debugging between the first-time and returning learners of programming, 2) the participants' passion for customizing after success by upgrading their projects or testing the limit of the physical chip, and 3) two forms of spontaneous collaborations. Across those experiences, I further identified the failures without feedback caused by the micro:bit's current coding environment and extended *Fish Tanks* and *Sandboxes*, two playful learning principles, to provide design insights for future physical debugging activities that support the findings above.

HOW CAN DEBUGGING WITH PHYSICAL COMPUTING BE MORE PLAYFUL FOR CHILDREN?

by

Danyi Zeng

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2024

Advisory Committee:
Dr. Caro Williams-Pierce, Chair
Dr. David Weintrop
Dr. Elizabeth Marie Bonsignore

Dedication

To my mother, Lizhen Mo, and my father, Li Zeng.

To all of us who enjoy playing and learning.

献给我的母亲莫莉珍、父亲曾李

献给享受游戏与求知的我们

Acknowledgments

My heartfelt gratitude to my thesis advisor and chair, Dr. Caro Williams-Pierce, for profoundly changing my perspective on the meaning, purposes, and structures of education. Tremendous appreciation for my committee members, Dr. Elizabeth (Beth) Marie Bonsignore and Dr. David Weintrop, for their incredible expertise and support throughout this transformative journey. Because of their collaborative advisement, I was able to conduct and complete an interdisciplinary study that not only responded to the academia but also fulfilled my pursuit.

My biggest shoutout to Kristina Anna Kramarczuk, a Clinical Assistant Professor & an inspiring Ph.D. candidate at UMD, for presenting me with a micro:bit on the unforgettable morning when we picked up kids for the STEM summer camps at the parking lot together. Without your recommendation, I could never have found such an interesting, suitable object for my research so fast.

Last but not least, thank you to my partner and friends who have always been ready to jump into thought-provoking conversations with me, push around those wild ideas together, and uplift my spirits in the past two years and beyond. Words cannot express enough how happy and fortunate I feel for their participation in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

This thesis is about seeking playfulness for children when they learn debugging in the environment of physical computing. I attempt to first understand how kids from ages 8 to 11 as young newcomers to programming approach the nature of debugging intuitively. Then, I explore corresponding design practices that could make use of the fun emerging from children's debugging process, with the aim to uncover future physical computing toolkit designs that may improve learning experiences. To address this goal, I designed two debugging mini-games on the micro:bit platform and observed how the children of KidsTeam at the University of Maryland interacted with them during two 90-minute participatory design sessions. I collected qualitative data on the participants' interactions with the codes and the physical chips of micro:bit, and used semantic coding to analyze their learning experiences. The outcomes verified the children's progress in debugging, revealed their passion for customization after success, and recorded their creative collaborations with each other. In this chapter, I will demarcate the background and my motivation to study the relevant questions, my positionality, the potential contributions of the results, as well as an overview of all the following chapters.

1.1 Background and Motivation

Debugging is a fundamental ability in Computational Thinking, yet it is always challenging to achieve especially for beginners in programming. Researchers from computer science studied the process of debugging and formed their disciplinary

perspective in the 1980s. In recent years, as computational literacy has become a necessity for people to learn, explore, and navigate in our time, many scholars from education and human-computer interaction have begun to tackle similar issues. Responding to the call to provide effective computational-thinking education for today's younger generation, some of the researchers became attentive to the learning process of debugging for K12 learners. One major and fascinating direction they look into is to apply physical computing for teaching debugging to deconstruct the complexity of the learning process and to provide a more hands-on experience for beginners of programming.

While the leading scholars have been trying to understand the process of debugging in the physical computing environment (DesPortes & DiSalvo, 2019; Fields, Kafai, et al., 2021; Jayathirtha et al., 2018, 2024; Lin et al., 2022) and provide better scaffolding for the process (Schneider, 2023) in middle schools, I am interested in extending the conversation to the elementary school level circumstances. Referring to the framework of *fragile* knowledge (Perkins & Martin, 1985) and Rich et al.'s analytical tools for evaluating dimensions of K-8 debugging curriculums (2019), I will understand how the participants approached debugging and what they lacked in their utility belts when encountering bugs in the physical environment created by micro:bit.

Then, I intend to find suggestions for bringing in playfulness into the picture. First, kids can learn programming knowledge – the use of language and linguistic understanding, particularly (Mccauley et al., 2008). Scholars have been exploring various learning mechanisms apart from traditional education. Mainly, Resnick & Rosenbaum, in computer science, developed the concept of tinkering to expand the potential of learning experiences for programming (2005). They argued that the "...playful, experimental, iterative style of engagement…" would enhance learners' abilities of adaptation and creativity (p. 164-166), and successfully applied the concept to their creation of *Scratch*, a now-popular computational construction kit (p. 168). In the same year, James Gee, a linguist and education researcher, published the principles of "learn by design" and argued that good-game-like learning experiences are powerful for learners to authentically utilize instead of memorizing knowledge (2005). Many of his discussions focused on STEM subjects, too.

Applying the overall ideology of increasing playfulness to the specific context of learning Computational Thinking through debugging, I tend to observe and find insights for creating both interesting and effective learning experiences for children. I believe that there is space for fun and improvement in physical computing for debugging and programming education. The study is a pragmatic exploration of the principles of playfulness within the particular context of Computational Thinking education, bridging multiple conversations of different scholar groups together.

1.2 Research Questions

My interest lied in combining physical coding with more playfulness. I started by examining the participants' reactions and strategies when they encountered bugs. Then, I sought achievements and failures in the overall learning process and implied the pros and cons in micro:bit's current environment. Lastly, I deducted insights for future designs. My research questions were:

1. How did the participants respond to the bugs they encountered and what they lacked if they failed to solve the bug?
2. What did the participants learn and develop through the successes and failures?
3. What are the implications for designing playful debugging activities?

1.3 Positionality Statement

Before I attended the program of the master's degree of human-computer interaction at the University of Maryland, I studied anthropology and practiced as a culture & user experience researcher in the consulting industry in China. I only started systemically learning programming before the summer I went back to the U.S. for the program. The first course I took was at *Codeacadmy.com* and I learned a surprising fact on day 1: a major part of professional programmers' work is debugging. During that whole summer and ever after, I kept repeating this fact in my own programming study and practice. So, I started wondering how beginners could learn to debug when they still know little about programming. What help and training

do beginners (or I, at that time and now still) want to be better at debugging and thus programming?

In my first year at UMD, I took a course called *Promote Rich Learning in Education with Technology* that introduced me to the imminent importance of Computational Thinking in current STEM education. Meanwhile, I started facilitating design and research sessions at KidsTeam on campus because I wanted to learn more about participatory design, a fascinating research method that prioritizes the voices and agency of children when designed for children. The two learning spaces, together, presented me with the diversity of educational environments, participants (beyond teachers in the formal education system), content, formats, and so many other factors in the discourse. Thus, I adopted a learner-centered perspective to design educational activities and, continuously, wondered how can learning be fun and playful if there is not always only one standard way to success.

My curiosity in gamification came into the picture finally. After seeing games or game-like projects at KidsTeam, I realized the potential of playfulness in education. It provides engaging, meaningful learning moments that support the learners' interests and emotional feedback – a form of interaction I regard fundamentally beneficial no matter how old the audience is. Therefore, I took another course called *Game as Emergent Experience*, seeking constructional knowledge of analyzing game experiences, the identity and mentality transformations of people as

players, the principles of designing game-like learning activities, and so on. The course successfully provided me with design and analytical tools for the prototypes I invented for this study.

My choice of this study was both a reflective hope and an inquisitiveness to knowledge. I wanted to find insights to solve this interesting problem that has happened to me and so many other learners and would happen to future learners of programming.

1.4 Contribution

My research will advocate an interdisciplinary conversation between the ongoing trend of debugging education for Computational Thinking and the long-existing focus on gamification in learning experiences. First, the study will supplement data on how elementary school learners entered a new physical computing environment and debugging. Second, the outcomes can disclose insights for increasing playfulness in current designs, based on theories like *learning by design*, *tinkering*, etc. from different academic fields. The discussion will consider how playfulness becomes more approachable for different types of learners. Lastly, the study suggests attention to additional advantages of learning physical programming. There could also be benefits for growing problem-solving and social-emotional skills that are widely applicable beyond debugging and STEM education.

1.5 Overview

Chapter 2 will draw literature to set the historical background of debugging in computer science, the current progress established for physical computing in education, and a miscellaneous theoretical source for "playfulness". Chapter 3 will introduce the theoretical standpoint and methodology of this study, including recruitment, research design, data collection and analysis, and logistics. Chapter 4 delivers the results of data collection, the debugging and programming outcomes from the two KidsTeam sessions, and the semantic codes derived from the video transcribing. Chapter 5 elaborates on the outcomes of the study, providing observations of the participants' successful learning progress and their interesting behaviors through debugging and creating new programs. This chapter also discusses potential ideas to improve playfulness for future debugging activities in physical computing. Chapter 6 concludes the thesis by drawing connections to the literature review, addressing limitations, and discussing future directions from the current study.

Chapter 2: Literature Review

In this chapter, I will first give a brief introduction to the concept of debugging in the historical development of computer science. Then, I will focus on the recent alignment of the benefits of physical computing for cultivating Computational Thinking among beginners and the emerging practice of turning debugging into a pedagogical process for learners. Lastly, I will discuss the definition of "playfulness" particularly for this study from an interdisciplinary standpoint. Collectively, the previous research achievements and my interpretations of them will draw the boundary of my academic focus and guide my research plan and analysis on debugging and playfulness later.

2.1 A Glimpse of Debugging In Computer Science

Debugging has always been a prevalent and time-consuming activity in most programmers' everyday work and a challenge for novices since the growth of computer science research and programming of all ages. Researchers have been studying the nature and causes of bugs over the years. In this section, I will introduce several key scholar groups as they set up the background for my research questions – how do programmers interact with the breakdowns in the computing system and what is a bug after all? Then I will explain my take on studying debugging in this project.

In the 1980s, researchers in computer science conducted several major studies to understand the nature of programming and how bugs happened. Soloway and their colleagues defined programming as a "goal-plan" processing and bugs happened when non-natural-language-related factors caused fallacies in programs (Mccauley et al., 2008). The researchers first scoped *preprogramming knowledge*: commonly among novices who had knowledge of natural languages and the logical habits to use the languages, they tended to apply these knowledge and habits to a computer language when they just started learning. And the plan could go wrong quickly – that was when bugs occurred (Bonar & Soloway, 1985). They also found that the majority of bugs were caused by *preprogramming knowledge* for beginners, but there were non-construct-based problems as well. Based on the seven causes they found, later researchers continued discovering more causes of bugs beyond the misconceptions of computer languages (Mccauley et al., 2008).

As many efforts have been invested into understanding the process of debugging afterward, a larger number of publications have classified bugs among college students and professional programmers since the 1980s (Mccauley et al., 2008). According to collective wisdom (Ahmadzadeh et al., 2005; Ford & Teorey, 2002; Hristova et al., 2003; Knuth, 1989; Mccauley et al., 2008; Robins et al., 2006), some bugs are related to specific computer languages or algorithms (thus *language liabilities*), while other bugs are ubiquitous to novices affected by the types of cognitive inconsistencies discussed above. From there, Robins et al. studied novice student programmers and separated three general types of bugs: *background*

problems, general problems, and language-specific problems (2006). *Background problems* refer to bugs that are "related to the use of tools, understanding the task, and very basic design issues". *General problems* include "difficulties with basic program structure, object concepts, naming things, and trivial mechanics, like mismatched parentheses and typos". And *language-specific problems* are specifically generated by the operation logic of computer languages[1], such as loops, conditions, variables, calculations, hierarchies, etc. (Mccauley et al., 2008).

The last piece I will refer to in computer science is Ko & Myers' comprehensive framework to identify bugs through "a chain of cognitive breakdowns" to address the cause of bugs from both cognitive and environmental aspects (2004). Based on the authors' summary of previous studies, as early as the 1970s and including some mentioned above, that attempted to classify bugs through different perspectives, Ko & Myers pointed out that bugs also happened due to non-cognitive external and internal factors in the environment (Ko & Myers, 2004; Mccauley et al., 2008). They adopted Reason's concept of *Human Error* and created the framework of *cognitive breakdown* to include both software-systemic and human-cognitive factors. Mainly, they identified three *programming activities*: *specification* (what the program should achieve), *implementation* (code to make the goal happen), and *runtime* (test and debug). And there are three *types* of breakdowns*: skill, rule,*

[1] The much longer list includes "issues related to control flow, loops, selection, Booleans and conditions, ……, data flow and method header mechanics, IO, strings, arrays, variables, visibility and scope, expressions and calculations, data types and casting, ……, hierarchies, ……." (Robins et al., 2006)

and knowledge breakdowns. A cognitive breakdown describes which *type* of breakdown happened during an *activity*[2] as well as the interface and the information that is being acted upon. A chain of cognitive breakdowns ultimately leads to software errors or, bugs.

Perkins & Martin's *fragile knowledge*, a concept Ko & Myers highlighted in their definition of a *bad rule* breakdown (2004), was an important pillar of theoretical support in my research, too. Back in the 1980s, Perkins & Martin coined the concept of *fragile knowledge* with their observations on how high school students debugged (1985). The knowledge was *fragile* whenever the knowledge was not complete, consolidated, remembered, etc. (Mccauley et al., 2008), so the students found it hard to utilize the knowledge to complete the process of debugging or coding. According to Perkins & Martin, there were four types of *fragile knowledge*: *partial/missing knowledge* that one never fully acquired, *inert knowledge* that one failed to retrieve, *misplaced knowledge* that one used the knowledge at the wrong place (usually a line of code), and *conglomerated knowledge* that one combined things incorrectly together (Mccauley et al., 2008; Perkins & Martin, 1985). In Ko & Myers' framework, therefore, they used *fragile knowledge* as the cause of a *bad rule breakdown* where the knowledge itself was correct but used inappropriately. The *bad rule breakdown*

[2] There are 6 types of activities: *design, creation, reuse, modification, understanding, and exploration*, but this will not be a focus of my study.

was a juxtaposition of a more common *rule breakdown – wrong rule* – where a rule was inapplicable due to the local code environment (2004).

2.2 Educate Debugging in Physical-Computing Environments

In the new century, Computational Thinking (CT) – "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Wing, 2006)"[3] – has been popularized and continuously studied for educational vocations (Shute et al., 2017), and debugging is a key activity in the fundamental scope (Yu & Roque, 2018, 2019). To promote and nurture those CT skills among learners, creating CT-inspired teaching and learning tools becomes a major focus naturally (Tang et al., 2020).

A compelling direction the researchers of CT and education have pointed out to cultivate debugging skills particularly is using physical computing. Physical computing, because of its tangibility, provides a unique environment for learners to expand their views of computing and further allows the learners to incorporate their interests in the learning process (Blikstein, 2013; DesPortes & DiSalvo, 2019; Y. B. Kafai et al., 2014). For example, in the summer of 2023 I facilitated the curriculums

[3] Many scholars have studied and redefined CT henceforth. I referred to the first formal definition only to introduce this concept broadly as a part of my study's context. Please see more variations in the articles of Shute et al. (2017) and Tang et al. (2020).

for several STEM summer camps at the Computer Science department at UMD, the students enthusiastically used Adafruit to create objects inspired by their hobbies or fictional characters.[4] As early researchers have proved, constructionist toolkits helped build personal as well as epistemological connections between the learner, the tool, and domain knowledge (Johnson et al., 2023).

While the physical coding environment empowers exploration and interest-driven learning, its unique environment also increases the complexity of debugging because more elements and subjects are involved. In the past few years, several groups of scholars have started understanding the nature and the process of debugging in physical computing environments. Because of the learner-centered nature of this topic, researchers have focused on the different groups of participants[5] (or factors) in the environment. Kafai and Fields focused on implementing physical debugging as a learning method in high school (Jayathirtha et al., 2018; Y. Kafai et al., 2020; Y. B. Kafai et al., 2014; Kaplan et al., 2011). Others like Elliott and DeLiema studied from a pedagogical standpoint for middle school students (DeLiema et al., 2024; Elliott et al., 2022, 2023). Recent scholars like Schneider began to look into informational and visual designs that can visualize the debugging process for learners (2022b, 2022a, 2023). Misirli and others led research on emerging debugging abilities in early

[4] A group of girls customized colors and sound effects for each of their Adafruit to build a Ninja Turtles team. A student created his version of Iron Man's glove. And, some students remixed some melodies they liked.

[5] Here, I mainly discuss the age ranges of K12 education. Research shows that most computational kits for young children (age 7 and under) have not incorporated debugging as a key element yet (Yu & Roque, 2018, 2019)

childhood (Misirli & Komis, 2023). Lastly, Kong and other scholars incorporated debugging as part of their wider research on CT (Chalmers, 2018; Kong & Abelson, 2019; Mukasheva & Omirzakova, 2021). I will elaborate separately in the following paragraphs.

Kafai and Fields, leading their research groups, focused on high school students in the past decade (Kaplan et al., 2011). They started after the appearance of the Maker Movement (Fields et al., 2016), applied the educational method of constructionism, designed "deconstruction kits" from electronic textiles (e-textiles), and inquired how high school students understand, explore, and solve problems in the debugging activities – sometimes collaboratively (Fields et al., 2016; Fields, Kafai, et al., 2021; Fields, Lin, et al., 2021; Jayathirtha et al., 2018, 2024; Y. B. Kafai et al., 2014). In recent years, they raised the concept of "Debugging by Design" (DbD) to extend the use of constructionism and encouraged students to create "failure artifacts" as a way to learn debugging (Fields, Lin, et al., 2021).

Addressing from a pedagogical standpoint, Elliott et al. discussed how middle school teachers' interactional grammar can better help support the complicated – and sometimes even new to the classroom educators – learning environment for students to understand debugging. Following the previous studies that developed the micro:bit-derived Data Sensor Hub (DaSH) for STEM education, Elliott et al. concluded that it was essential to debug *with* students as a teacher. Such as, teachers

should care for frustrations and joys emerging from the debugging process (2023). DeLiema, approaching debugging as a learning opportunity from failures, recently emphasized the divergences between teacher-student junctures during debugging as heterogenous learning moments (DeLiema et al., 2024). Also working with middle school classrooms, Schneider took the design-based stance and iterated the design of debugging scaffolds (2022, 2023). He proposed Circuit Check, a live sensor reader, available for several physical computing toolkits, for students to check both logical errors and test the hardware parts directly. The prototype can help teachers understand the situations from the pedagogical perspective as well. Researchers like Misirli focused on the debugging abilities of preschool children (4-6 years old) and found the children's construction and development of synaptic and semantic knowledge (Misirli & Komis, 2023).

While many scholars have studied the experiences and views of debugging in high school, middle school, and preschool, the majority of researchers focusing on elementary school studied debugging more as a part of CT than an independent discourse for its further potential as learning opportunities. For example, Kong, in his book *Computational Thinking Education*, incorporated debugging as a key skill of the CT process (Kong & Abelson, 2019). Other scholars from the same period demonstrated the same strategy of considering debugging (Chalmers, 2018; Mukasheva & Omirzakova, 2021). A decade ago, yet, Sipitakiat and Nusen confirmed that, among children of 8-9 years old, debugging is achievable by step-by-step program running and using passive objects could help identify the bugs (2012).

Joining the discussion of how to turn the debugging process into educational opportunities for children to learn Computational Thinking (Y. Kafai et al., 2020), my RQs will directly contribute to this context. This study will help understand the debugging process of particularly elementary school participants, generate insights into the advantages and insufficiency of a physical environment like the micro:bit, and locate the opportunities for playfulness along the way.

2.3 Playfulness as An Interdisciplinary Concept

Playfulness and gamification have been popular, interdisciplinary concepts in many fields, including STEM education and participatory design. Resnick & Rosenbaum, in their original advocacy of tinkering in programming learning, set "playful" as the overall tone for their engaging, experimental activities (2005, p. 180). Gee, a linguist, education researcher, and gamer, coined a set of design principles to transform learning experiences into game-like, challenge- and interest-driven activities that teach learners to practice knowledge instead of simply knowing. Participatory design, a useful and ethical research method especially in designing for children, also commonly addresses "fun" as part of the "technology design process" to situate children's cognitive needs (Bonsignore et al., 2013; Fails, 2012, p. 143).

A further interesting connection between the academic group who studied block and physical coding and those who studied playful learning principles was that

the indispensable human-computer interaction between the learner/player and the operable objects on the screen provided a space rather dynamic for gaining insights about the user experiences. Weintrop and Wilensky who studied block-coding applied *distribution cognitive theory* and pointed out that the blocks became the memory carriers of coding rules (e.g, the block name indicates its function, the shape indicates where the block could be inserted, etc.) and thus made the environment easier to use. From a cognitive perspective, these memory carriers not only released learners' memory load but also performed as hints and inspirations for the learners (Weintrop & Wilensky, 2015). Gee, in his discussion of playful learning earlier, also pointed out that, a good video game gave the virtual character and its player two separate sets of skills – the former can jump, dash, attack, etc. and the latter must give instructions and operate based on their reading of the environment (2005). The player must know their character well and initiate a simulation between the character and themselves to efficiently utilize both sets of skills to win. This *projective stance*, thus, to Gee, was an essential entry point to build deep, embodied experiences. Gee's finding proved that the objects in a learning environment could facilitate an embodied experience with fun. Weintrop and Wilensky's research verified that the code blocks already had the capacity to carry information both as rules and inspirations.

Bridging the ideas of the multiple groups above, I wondered how block coding could become an embodied playful experience for learning as if in a good game. What actually happened when a learner started moving their code blocks? If the block codes are already memory carriers, then how could the process be more

embodied than just efficient? What were the potentials of block-coding, if it became an embodied learning experience for children? And most importantly, where could playfulness become a booster in that experience?

Chapter 3: Methodology

3.1 Theoretical Framework

Responding to my literature review, I will address my standpoints on the key concepts introduced above for my research context: I took *programming knowledge* as a premise of my research design and was more concerned with *background* and *general* challenges than *computational language* challenges, given the physical coding environment. I also referred to and expanded the concept of *fragile knowledge* to situate the physical coding environment where learners must interact with multiple non-human objects. Finally, I used "playful" and "playfulness" as the overarching theme to describe my research context and the focus for future implications.

First, the *preprogramming knowledge* is a fundamental premise for studies involving children participants, including mine. Because children are still developing their cognitive capacity, their preprogramming knowledge itself could be unsettled. Particularly for my study where the participants were 8-12 years old, children of this age range (addressed as "tweens" in the following context) usually just start developing cognitive skills for independent thinking and acquiring more complicated mathematical skills of arithmetic, geometry, and sometimes formal algebra (Fisher, 2015). Therefore, it is necessary not to assume stability and coherence in the participants' pre-program logical thinking. My study did not ignore the bugs caused by *preprogramming knowledge*, yet I attentively focused on other cognitive and environmental factors that prevent children from debugging successfully.

Second, my study mostly observed the cognitive *background* and *general* problems while excluding the majority of *language liabilities* or *language-specific problems* in the micro:bit's block-coding environment. On the one hand, the micro:bit environment essentially as a coding platform was for users to create programs with certain functions, either practical or entertaining. Especially, my activities came with pre-designed goals and functions already. So, I expected that *background* and *general* bugs can naturally occur when my participants tried to fix the codes. On the other hand, the block-coding mechanism of micro:bit already reduced the occurrence of *language-specific* problems that were triggered by texting and formats. Except, the key elements, such as variables and functions, would show up across almost all common computer languages. These components, even in block coding, existed and acted as an indispensable factor once a program's *goal* was beyond pure mathematical calculation. Thus, the participants might experience these problems frequently as well.

To create a manageable environment for my participants, I decided to further avoid the complicated *language-specific* problems and observed the *background* and *general* problems if they naturally occurred. As I will elaborate in the following sections, I created one activity based on mathematical calculations that were already familiar to the tweens in their formal education and another based on an intuitive, imagination-driven setting. These were easy goals of programming for the participants to understand, and the code blocks were clean (yet sometimes

challenging) to read. I also only set up bugs related to variables and functions, considering most participants' limited familiarity with computer language. Through this well-balanced overseeing of the difficulties in the pre-existing bugs, I aimed to maintain fun and engagement in my study.

Third, extending Perkins & Martin's concept of *fragile knowledge* in Ko & Myer's context of examining cognitive breakdowns, I would like to incorporate the unique physical coding environment – block coding, a physical, interactive device, and other factors in the space – into the knowledge system and examine what *rules* are needed and learnable. To start with, from Ko & Myer's analytical framework, my research would mainly include *implementation* and *runtime* activities and I would concentrate on *rule breakdowns* to examine the participants' learning experience. However, different from the text programming in Ko & Myers' research, rule breakdowns in physical computing required a closer look given its complex combinations of codes, devices, etc. I also considered the fact that many of my tween participants would encounter a physical coding kit like micro:bit for the first time. They needed to learn block coding and the interaction with a physical device as new events, too. Thus, I wanted to flesh out the learning process with further awareness of the sources of the *rules*.

Therefore, I decided to detect *fragile knowledge* of different components in my research context: the block coding, the interactions with the physical chip, as well

as the common computing logic. When I observed the children's debugging behaviors and analyze the data, I distinguished in which component the *fragile knowledge* is situated. Then, I could use the analysis of the typical moments around *fragile knowledge* to trace back what was missing in the physical computing environment for the participants to establish non-*fragile* knowledge. My assumption was open: the facilitation, on-site guidance, particular programming environment of the toolkit I chose, activities' designs, etc. could all affect the emergence of *fragile knowledge.* As a result, I expected my findings to shine a light on future improvements for fewer *breakdowns* and more playfulness in debugging learning activities.

Finally, I used "playful" and "playfulness" as an overall description of the environment where I conducted my research and a starting point for the debugging projects that I designed for children to try debugging on micro:bit. KidsTeam at the University of Maryland, where I conducted my research, was a participatory design lab where children owned agency and leadership to play, explore, design, etc., without the traditional class evaluation system or standardized learning objectives. Thus, as demonstrated in Fails et al's article (2012, p. 114), the participants in my study would offer honest feedback, comments, ideas, and directions for technology innovations. This environment created a premise of "playfulness" that the participants knew they could and should try joyful actions even before they started any specific assignments. More explanations are in Chapter 3.4. Meanwhile, referring to Resnick & Rosenbaum's development of *Scratch* (2005) and Gee's design principles of game-like learning systems (2005), I created two debugging projects that looked like games

and invited the participants to “play detectives” within the scripted contexts of the projects. I will further elaborate in Chapter 3.3.

To summarize what I adopted from previous studies, although my research focuses on the very entry-level scenarios of computing and debugging among children, I situated the research context into the interdisciplinary conversations among the debugging research in computer science, the call for teaching computational thinking via debugging in HCI, as well as the advocacy for playfulness as a systemic innovation in education. First, I acknowledged *preprogramming knowledge* as a premise for me to observe young novice programmers’ interaction with coding and debugging. Then, I referred to the three types of bug problems (*background, general, and language-specific*) to delineate my research design, so that I could create a proper, playful yet challenging setting to collect data. To answer RQ1 (*How did the participants respond to the bugs they encountered and what they lacked if they failed to solve the bug?*), I extended the scope of *fragile knowledge* in the contemporary discussion to reversely examine the *rules* for debugging in micro:bit and study how the physical environment supplements. Lastly, I acknowledged the inherent playfulness in the research space and extended the same ideology in my debugging project designs. Combining their social and emotional capacity for fun, ethics, collaboration, etc, the children in this age range are ready to enjoy games for STEM thinking (Fisher, 2015, pp.107). The ultimate goal of my research was to inquire about improvements for children’s learning experience of physical debugging along the way.

3.2 Micro:Bit Environment

Before I give details on the debugging projects into which I intended to add playfulness so that the kids could enjoy the challenging learning experiences more, I will briefly highlight what features micro:bit shares with most physical coding kits, what makes it more accessible for beginners, and what it lacks in this project's context.

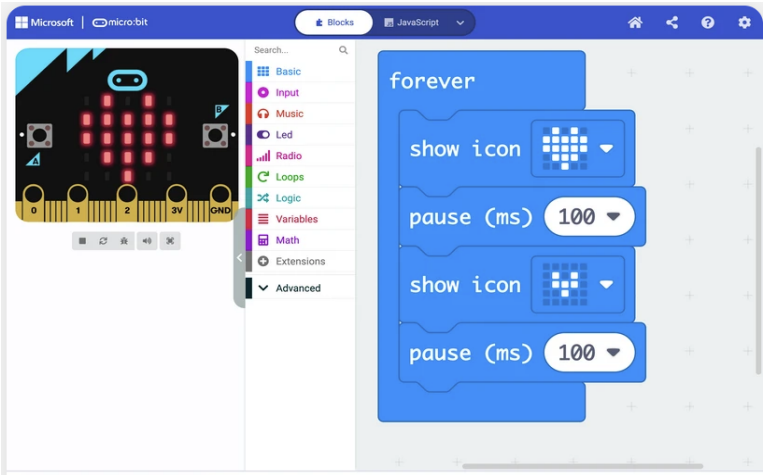

Figure 1: A typical micro:bit coding setup

Similar to other educational programming platforms and/or physical coding kits (*Scratch*, *Arduino*, *Adafruit*), micro:bit uses block coding as a predominant design to lower the threshold for learners to start programming or, more precisely, Computational Thinking. Compared to text-based coding, blocks-based coding looks more like natural languages for humans to understand, is easy to drag and drop for composition and tinkering, provides visual hints (e.g., the shapes of the blocks) to help learners draw connections, etc. (Weintrop & Wilensky, 2015). The platform of

micro:bit has satisfied all these features, too. In short, block-based programming allows learners to start practicing logical thinking to command a computer without worrying about the grammar, formats, syntax, etc. of computer languages.

What makes micro:bit attractive and playable for young beginners especially is its physical, multiple-way interactable micro:bit chip. Coming in with a small box, the chip is light and single-handedly playable by a kid. Yet, it has features including but not limited to a 25-LED light pad, buttons, accelerometer, light sensor, temperature sensor, speaker, pins, etc. (*Make It*, 2023). According to the tutorials and sample projects on the micro:bit's community website, a few blocks of codes in correct order and logic can quickly generate a digital name tag, a step counter, paired radio communicators, a dancing prop, so much and so on. In its design value, micro:bit aims at combining the advantages of block coding and physical coding so that the young, beginning learners of programming can be widely inspired and encouraged[6].

[6] However, since inadequate studies indicated how elementary school learners reacted to these designs, I would like to point out that my research would also help examine if mico:bit's environment and the features actually worked well for my participants.

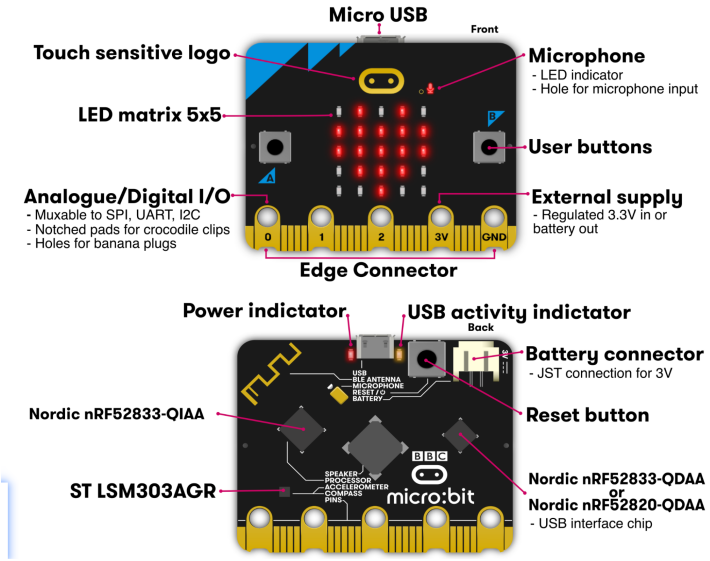

Figure 2: Micro:bit's official hardware instruction (*Hardware*, n.d.)

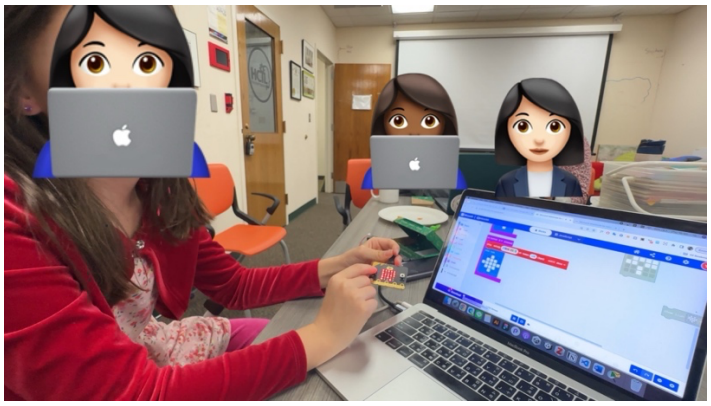

Figure 3: A participant pressed a button on the chip and showed its function.

Because the website of micro:bit aimed at providing instructions mainly for formal education environments, I needed to redesign the research activities for the unique setting of KidsTeam. After a quick examination, I realized that although the designers of micro:bit released short videos and tutorials for first-time users, those materials held a premise that their audience was teachers and educators who already

had pedagogical ideas and methods in their heads and would use micro:bit as a structuralized teaching tool for procedural, semester-long teaching. My study, however, did not have the capacity for such a long-time learning experience and I intended to treat programming on micro:bit as a sandbox activity. Therefore, I decided to design my own debugging projects and session plans. I must utilize the features of micro:bit that suited the environment of KidsTeam most and design activities that could be played in one or two short sessions to fulfill my research goal.

3.3 Debugging Projects Design

I created two block-code programs with pre-installed bugs in them, and the children at KidsTeam were encouraged to find those hidden bugs and solve the problems in their own ways. Applied Gee's principles, the children were given appealing and empowering identities in these "missions". There was no judgment on their failures. They could learn as they were playing with the codes. The solutions to the problems required them to use the just-learned knowledge but were also flexible enough to be customized by the participants' own journey of learning. Similar to coding on *Scratch*, there was space for the participants to tinker, explore, and make a lot of mistakes in their playing and playing. There were challenges but the participants could choose to opt out or in again at any time they would like to. The goal was not to finish a mission solely but to enjoy and grow on the journey (Resnick & Rosenbaum, 2013, pp. 179–180).

After several explorations and experiments, I finalized two debugging projects – *Counter* and *Floating Bubble* – for the participants to play around with. The overarching design idea was that each project was a seemingly completed project that could achieve a computational goal. However, the projects were in fact malfunctioning because some blocks had bugs in them. The participants, with the constrained support from the adult researchers, needed to first understand the goal of the project, find the bugs, and fix them.

Both projects shared the overall goal of providing an adequate context for the participants to go through the *specification* (what the program should achieve), *implementation*[7] (code to make the goal happen), and *runtime* (test and debug) (Ko & Myers, 2004) of the programming process, although the two projects delivered the context very differently in a form of problem-solving to create an interesting learning space (Thorn, 2013). *Counter* was a mathematical-geared project, in which the micro:bit worked as a numerical counter. Yet, to maintain the challenge relatable to the participants, I set the whole project in the context of recording the number of people at KidsTeam at that moment. *Floating Bubble,* on the other hand, invited the participant to use imagination and embodied experience. A red dot on the LED pad presented a drifting bubble on the water's surface and the bubble moved against the direction of gravity. Instead of calculating in math, the participants needed to draw

[7] The implementation for the participants in my study was not to write codes from scratch but to tweak or add blocks iteratively.

direct connections between the commands in the codes and the positions of a physical red dot.

Because I wanted the participants to approach the projects progressively in a game-play-like setting, and considering the KidsTeam participants varying in creativity interests, familiarity with programming, age, etc., I designed 3 levels for each project with the difficulties increased. The levels' designs ensured that participants would certainly encounter general and language-specific problems (Robins et al., 2006), and the first-time micro:bit environment meant the participants might encounter background problems and/or create new bugs in the environment as well. For example, *Counter* Level 1 included a bug of mathematical addition that belonged to language-specific problems. *Floating Bubble* Level 1 involved understanding how an imaginary object could be realized on micro:bit via codes. Moreover, some levels, like *Counter* Level 2, could be solved in multiple ways, so participants to approach the challenges intuitively and individually, as Gee proposed for designing a playful learning process (Thorn, 2013). These arrangements guaranteed me to collect diverse data on the kids' debugging experiences in a safe yet explorative space.

I will flesh out my designs for each level of the two projects in the following sections, please refer to the file *Workflow* in Appendix 2 as the counterpart manual I prepared for the adult researchers in my study. I also acknowledged that the

participants might encounter breakdowns during the activities because of the limits of the designs. Yet, the focus of this study was not to evaluate the capacity of the projects but to use them as debugging environments to understand the participants' experiences.

3.3.1 Level 1

Both projects started with a level 1 that introduced key elements throughout the whole activity and I designed only one bug in level 1. *Counter* started with introducing block coding and the concept of "variable" and connected the changes in the value of the variable by pressing the physical buttons on micro:bit in three different ways. Pressing button A should add the variable "count" by 1. Pressing button B should show the result of "count" immediately. Pressing A and B together will rest "count" to 0. Thus, the project required the participants to first understand the math and then draw the connection between the math and the ways to press the buttons. The process was more abstract compared to *Floating Bubble*.

Level 1 in *Floating Bubble* looked much simpler but the first challenge was to switch minds and understand the use of a variable differently from that in *Counter*. Instead of changing the value of a variable through calculations, *Floating Bubble* directly reflected the variable ("bubble")'s value, the position of the variable on the LED pad, visually. The participant needed not to press a button to check. The second challenge, or a hidden hint, was to recognize a second variable "play" and its

supporting logic “true”. This design was to mimic the situation beginners commonly encountered when they needed to navigate through unfamiliar, scaffolding codes and processes.

3.3.2 Level 2

The level 2s aimed to provide a full, coherent environment where the participants could apply their learning from level 1 to read and examine the code blocks. *Counter*’s level 2 increased the difficulty by adding another variable, but the mathematics remained the same. *Floating Bubble* became much more challenging by introducing the preset variable “xAxis” that detected the motion of micro:bit horizontally. It also required the “Bubble” to move accordingly when the micro:bit chip was tilted.

3.3.3 Level 3

The level 3s were “extra challenges” only for the participants who found the previous ones extremely easy or boring and wanted to try something much more complicated. *Counter*, at this level, changed the preset functions of the buttons and asked for adding subtraction at the same time. It *seemed* more difficult but turned out rather simple in coding if a solution was found. *Floating Bubble* added another direction of motion as a new variable into the picture and required the participant to think counter-intuitively this time – the bubble moved *along* with the direction of gravity.

3.4 Playful KidsTeam & Participatory Design

I conducted two 1.5-hour research sessions at KidsTeam at the University of Maryland in the Fall 2023. The study was approved by the Institutional Review Board (IRB) at the university with the submission of amendments for the KidsTeam project. 7 children and 6 adults attended the first session, and 8 children and 5 adults attended the second session. All the child participants were members of KidsTeam, from 7 to 12 years old during my research time. The adults, including me, were affiliated with iSchool, UMD and had been volunteering at KidsTeam for at least one academic semester. Thus, before my research started, all the participants were already familiar with the values and formats of KidsTeam, and the adults knew the children's personalities well.

KidsTeam has a strong tradition in participatory design[8] and the children are accustomed to exploring and creating in a playful, informal learning environment every week. Following the idea and methods of participatory design, KidsTeam works by centering the children's feedback and choices in the design process (Fails et al., 2013). Children can play many different roles in the process (Begnaud et al., 2020). For example, at KidsTeam, the children have participated in design activities such as brainstorming on future solutions to global issues, testing game or product

[8] When the users are children, some research used the more accurate term "cooperative inquiry" to describe the method. In my context, I will continue using "participatory design" as the generic term to represent the KidsTeam's environment.

prototypes, sharing feedback on current technologies, etc. The adults, supporting the participatory design process, respect the children's original voices and help articulate and synthesize them. The goal is to create a safe, empowering space for the young participants to experiment, create, and share.

A common key focus at the KidsTeam session is hack challenges. As Fails et al., stated, solving problems sits in the definition of developing technology (Fails et al., 2013, p. 91). At KidsTeam, a lot of sessions were structured for the participants to first understand a design challenge and the common benefits of solving it, and then provide proper tools (such as big paper, bags of stuff, sticky noting, big ideas, etc.) for the participants to hack the problem.

As a result, these elements of participatory design form a playful environment for kids to both create and learn, as Gee proposed for a game-design-inspired learning experience (Thorn, 2013). First, the kids usually feel empowered in the room, because they own their agency and their thinking processes. Their experiences at KidsTeam are customized by the diverse creativity tools available in the room as well as the close attention from the adults. The value that their opinions and designs are held in and the awareness of knowing that their input would pivot future designs for their friends and people around the world encourages them to take leading voices in the room. They also learn to respect that everyone can have a different opinion even on the same question – although the process may be full of banters and jokes.

Second, because of the nature of problem-solving in the participatory design sessions, the kids are used to encountering new questions and ideas and situating the questions in the socio-cultural context (Thorn, 2013). KidsTeam sessions are always open with clear communication to all participants that there are no right or wrong answers. They need not worry about making mistakes or getting punished because of their sometimes "goofy" ideas, so they are usually quite happy to participate in the activities. Moreover, they see, experience, and sometimes digest completely different research or design topics in the weekly sessions, so it becomes fairly common for them to feel a sense of diversity *and* be asked to connect the dots among their experiences.

All these features of a playful environment for creativity and learning supported me in situating my project at KidsTeam because debugging the process was new to most participants and itself is innately problem-solving. Acknowledging that debugging could be difficult and even daunting to young beginners in general, I believed KidsTeam provided a relatively safe and supportive space for me to introduce debugging the concept and the carefully designed programming projects to the kids. Because tweens are still developing skills to regulate their emotions and hard to persevere in adversary situations (Rich et al., 2019), KidsTeam is a flexible and tolerant space for the participants to encounter challenges and failures without harsh judgments.

Child (pseudonym)	**Age** (years)	**Gender**	**Identity**	**Session 1 Attendance**	**Session 2 Attendance**
Adrian	12	M	Returner	Yes	Yes
Bobby	12	M	Returner	Yes	Yes
Eddie	8	M	Returner	Yes	Yes
Helen	10	F	Newcomer	Yes	Yes
Oriana	9	F	Newcomer	Yes	Yes
Ray	9	F	Newcomer	Yes	Yes
Nova	9	F	Newcomer	Showed up but did not majorly participate	Showed up but did not majorly participate
Aaron	7	M	Newcomer	No	Yes
Adults	-	-	-	6	5

Table 1: Participants who attended the study

3.5 Session Plan & Implementation

3.5.1 Session Plan

KidsTeam follows a typical routine of participatory design for each week's activity. I proposed two sessions to give the participants abundant time to play and process while the adult team could both facilitate and collect data at a manageable pace. Both sessions' plans were typical at KidsTeam: the team started with a conversational snack session to re-address the equal power dynamic among everyone in the room – no matter their ages and genders. Then, everyone answered a "Question of the Day" (QoD) that was relevant to that day's design or research theme and would intrigue the participants to be further engaged with the following activities.

Afterward, the participants broke into smaller groups with 1-2 adults for that day's activity. In my session, each active participant had one adult's full attention when they programmed. Finally, at the end of the second session, everyone joined in a final group discussion and provided their feedback. After each session ended, the adults gathered together, exchanged observation notes, and offered suggestions and comments for the study.

For the first session, to begin with, my QoD was: "What is a bug…?", followed by probing questions: "Have you ever encountered a bug/glitch? What did you do?" The questions were to grasp a preliminary understanding if the participants had ever had an experience before the session and how they viewed debugging so far. Then, to warm up and introduce micro:bit – a new computing environment to all of the kids, I prepared a small game called *Dice* before the kids started the debugging projects: the pre-downloaded codes on micro:bits would show a random number from 1 to 6 whenever being shaken. Because the game was simple enough to understand, required physical interaction with the micro:bit chip, and contained surprises, it was an invitation to draw the kids' attention and excitement to learn about how micro:bit work and possibly build something similar.

After the QoD and the play of "dice", the participants started the debugging projects. Because the goal was to observe what bugs appeared and how kids debugged specifically in a playful setting of micro:bit, each kid was given a micro:bit

kit to work with and was accompanied by an adult researcher all the time. They started with *Counter* level 1 and could choose to level up or switch to *Floating Bubble* at will. The kids had the freedom of exploration and continuity, meaning they could try for as much as they wanted if they got stuck, but the adult researcher also had the volition to provide help whenever they decided the kid was too tired or lost in the process. For example, if a kid found *Counter* too mathematical, the adult could encourage them to try *Floating Bubble* instead. Or vice versa. Furthermore, the adults were instructed to actively solve operational problems of the computers and the micro:bit platform. For example, they could help locate a block the kid specifically asked from the category, retrieve a block the kid accidentally deleted, etc. If the pair needed troubleshooting beyond their knowledge, they could always find me at a close table. My purpose here was to provide the basic, necessary structure for playful bugging that guaranteed the participants could encounter and interact with bugs, and I also wanted them to stay engaged in the activities that were not too intimidating or boring to them. The levels of each project were created with a similar consideration.

For the second session, the QoD was: "What did you do on Microbit last week? What would you like to program today's session?", which aimed to refresh the participants' memories and allow them to share their experiences and feelings. For the main activity, originally, I planned to ask the participants to continue working on *Counter* and/or start *Floating Bubble*. Yet, after Session 1, I consulted with the adult team and decided to remodel Session 2 to satisfy the kids' strong autonomy of creativity through wide-wall programming activities (Resnick & Silverman, 2005).

This format provided much more play and less restrictions on the project goals. Then, I could examine the debugging moments within a coherent context where the participants decided the project goals. Tinkering, revising, and tailoring the code blocks when the chip could not generate the expected outcomes became a solid form of debugging as well.

As a result of the collective decision, I provided 4 diverse but shorter activities as inspirations for the participants to explore: *Name tag with sounds*, *Micro:bit chat, Tamagotchi*, and *Light sensor*. These projects had either similar logic or code blocks to *Counter* and *Floating Bubble*, but there was no bug. They were half-built, and the participants could choose to complete them or develop their own ideas away from the structures. Indeed, there were project goals and unused blocks scattered in the space as hints or suggestions, but none of them were mandatory. Consequently, debugging would appear when the codes could not realize the participants' expectations. They had to examine and revise the codes to make the program right, which naturally became moments I could study how they identified and solved the problems.

At the end of Session 2, all participants gathered together and shared their experiences, instant feelings, and thoughts. Particularly, I asked them to define debugging again and shared what bugs they encountered. In this way, I could detect the changes in debugging as a concept in their minds before and after the sessions. Then, I inquired what they found interesting, challenging, boring, etc. and how they

wanted the activities could be more playful, which allowed me to see what was fun from the participants' perspective and what could be future implications of improvements.

3.5.2 Preparation

To help other adult researchers better understand the debugging projects, I also prepared a workflow guidance, cheat sheet, and demonstration videos for each level's outcome. Please refer to Appendix 2 for details.

3.6 Data Collection

My main data source was the screen and audio recordings, and I also referred to the session field notes, photos, artifacts, etc. to acquire a holistic understanding of the participants' experiences. Because tweens in general do not tend to manage their emotions perfectly (Rich et al., 2019), capturing and analyzing the instant conversations and emotions, and following the decision-making process would help me better detect the subtle facts and cognitive changes in their successful or burdened debugging experiences. In order to identify key interactive moments, the adult researchers and I agreed to pay close attention to during the sessions: how the kids reacted to the digital or physical objects, elements, and outcomes at the different phases of a debugging process; how the participants explored, learned about the new "thingy" in the front of them, and decided what to do with it, etc; what they (dis)liked about the object and debugging itself. Yet, I also acknowledged the difficulty of

covering all these detail-oriented questions while supporting a tween to debug for the first time, and I understood the responsibility of the adult researchers to focus on the kid's needs continuously. Therefore, I acquired immediate feedback both from the adults and the kids after the sessions as well to secure fresh inspiration.

The adult team also collected ethnographic notes and brief discussions from the adult researchers as important supplements and collected the participants' voices after Session 2 for a collective thumbnail opinion. I took in the adults' first impressions after each session because they had the best opportunities to notice and record interesting moments when they spent a whole session observing and facilitating one kid individually. All the raw data were collected and uploaded to a shared repository through the adult team's collaboration.

3.7 Data Analysis

I first transcribed all the videos into verbatims and on-screen actions with time stamps. I also distinguished physical and cognitive actions, based on the conversations and the on-screen information, like cursor movements, code block movements, and typed texts. Then, I conducted semantic coding and built initial themes that addressed: how the participant reacted in terms of applying *fragile knowledge*, what strategies they used to debug, and what programming actions they took when they had the freedom to build their own projects. I also paid attention to original actions that I had not learned from the literature review so far. Afterward, I

reviewed the field notes and supplementary recordings to cross-check and refine my codebook. Finally, I summarized the repetitive themes from the sessions and across the participants. I synthesized them into proper categories that allowed me to answer my RQs accordingly.

Chapter 4: Results

In this chapter, I will first address the information of raw video and audio data I collected. Then, I will briefly list the outcomes the participants achieved or created in both sessions. I will also provide a list of semantic codes produced during my preliminary coding of the raw data, and my analysis strategies used to apply the codes to answer my RQs.

4.1 Raw Data

I received affluent raw data in screen recording and audio. A few adult researchers also shared their field notes and summaries after the sessions. The following 2 tables were the inventories of what forms of data I received for both sessions. Please notice that there were missing data due to technical issues or some participants did not choose or have enough time to explore certain activities.

November 02:

Child (pseudonym)	**Screen Recording**	**Audio**	**Ethnographic Notes**
Adrian (12, M, R)	Yes	Yes	/
Bobby (12, M, R)	Yes	Lost	Summarized the main actions, decisions, thinking processes about Bobby's debugging of *Counter* Levels.
Eddie (8, M, R)	Lost due to technical issues	Lost due to technical issues	Summarized Eddie's likes and dislikes but did not include the results of his debugging.

Helen (10, F, FT)	/	/	Summarized Helen's experience of creating the counter for hand-clapping sounds. Summarized Helen's likes and dislikes. Recorded key conversations and learning moments when Helen
Oriana (9, F, FT)	Yes	Yes	/
Ray (9, F, FT)	Yes	Yes	/
Nova (9, F, FT)	No full participation		
Aaron (7, M, FT)	Absence		

Table 2: Session 1 Raw Data

November 09:

Child (pseudonym)	**Screen Recording**	**Audio**	**Ethnographic Notes**
Adrian (12, M, R)	Yes (Collaborated with Eddie)	Yes (Collaborated with Eddie)	/
Bobby (12, M, R)	Yes	Partially	/
Eddie (8, M, R)	Yes (Collaborated with Adrian)	Yes (Collaborated with Adrian)	/
Helen (10, F, FT)	Yes	Partially	Summarized Helen's likes and dislikes. Recorded key conversations and learning moments when Helen.
Oriana (9, F, FT)	Yes	Yes	/

Ray (9, F, FT)	Yes	Yes	/
Nova (9, F, FT)	No full participation		
Aaron (7, M, FT)	Partially	Yes	/

Table 3: Session 2 Raw Data

Given the 2 sessions, most participants were deeply engaged with the projects and activities and enjoyed their unique learning journeys. There were certainly moments when the participants found it confusing or difficult. The adults in general provided abundant emotional support, but because the adults were also new to the micro:bit coding environment, there were situations that lacked clarification or directional instructions from for the participants to process the situation more easily. Judging by the participants' opinions in "Big Ideas" at the end of Session 2, I believe the participants had 2 safe and interesting experiences.

4.2 Debugging Results from Session 1

Session 1	*Counter* Level 1	*Counter* Level 2	*Counter* Level 3
Adrian (12, M, R)	Completed very fast	Solved with unnecessary changes	/
Bobby (12, M, R)	Completed	Solved with unnecessary changes	Half-way through due to end of the session
Oriana (9, F, FT)	Completed with the adult's step-by-step instructions. The adult used an older version of the project, but the	Completed fast Adult used an older version of the project, not the newest. (Need	/

	setup was mainly the same	to address the difference.)	
Ray (9, F, FT)	Failed, but explored the "music" section and added the melody function to the buttons	Did not try the project but used the interface to explore other functions mirco:bit provided, including drawing LED pads, music blocks and the "if else...then..." block	/

Table 4: Debugging Results in Session 1 - *Counter*

Session 1	*Floating Bubble* Level 1	*Floating Bubble* Level 2	*Floating Bubble* Level 3
Adrian (12, M, R)	Completed	Half-way through due to end of the session, almost figuring out the functions for xAxis rotation	/
Bobby (12, M, R)	/	/	/
Oriana (9, F, FT)	Completed without actual correct understanding	Did not start due to the end of the session	/
Ray (9, F, FT)	Completed	Half-way through due to end of the session, but removed the music blocks for indication as "unnecessary"	/

Table 5: Debugging Results in Session 1 – *Floating Bubble*

Session 1	Participants who lacked complete raw data.
Eddie (8, M, R)	Lost video and audio data due to technical issues. The adult provided ethnographic notes.
Helen (10, F, FT)	Helen was not able to access the projects due to a technical issue on site for the first 20-30 minutes. The adult used the micro:bit manual instead for Helen to explore. The adult provided written reflections.
Nova (9, F, FT)	Showed little interests in coding, thus did not work on a project coherently. Observed and participated in other participants' activities sometimes. For example, joined with Helen to test her hand-clapping counter.

Aaron (7, M, FT)	Absence

Table 6: Participants with missing data in Session 1

4.3 Programming Artefacts Generated from Session 2

Session 2	Everyone tried different things out of interests and/or challenged by the adults.
Adrian (12, M, R)	Worked on a program that play music and show 4 LED patterns repetitively. Collaborated with Eddie.
Bobby (12, M, R)	Completed a challenge called *Light Sensor*. The challenge provided a general framework and potentially useful variables. Bobby figured out the logic and used conditional blocks to succeed.
Eddie (8, M, R)	Worked on a program that play music and show 4 LED patterns repetitively. Succeeded in solving the challenge probed by the adult: repeat the music and LED patterns at the same time. Eddie used a specific function in playing music and the principle of parallelism to succeed. Collaborated with Adrian.
Helen (10, F, FT)	Created 4 LED letters “L”, “O”, “V”, “E”. Tried to add different melodies in the database.
Oriana (9, F, FT)	Tried to solve the challenge *Light Sensor* but became very confused without the adult explaining the goal of the program. Created a name tag with her initials showed by the LED letters, applying the sequence of codes.
Ray (9, F, FT)	Assigned a different melody and a different LED pattern to button A, button B, button A and B pressed simultaneously. Adjusted the melodies and patterns multiple times to achieve her expectation.
Nova (9, F, FT)	Showed little interests in coding, thus did not work on a project coherently. Observed and participated in other participants' activities sometimes. For example, saw Ray’s artifact and joined with Ray to tinker the codes.
Aaron (7, M, FT)	First solved *Counter Level 1* successfully. Then, adjusted the program into a “Pi Machine” that show the first 30+ digits of Pi on the LED pad. (He entered the numbers manually.)

Table 7: Participants’ activities in Session 2

4.4 Semantic Codes

Analyzing the raw actions and conversations, I summarized 4 sets of semantic codes that helped me understand and further study the interaction between the participants and the computing system. Appendix 1 provides full definitions and examples in detail.

4.4.1 Type of Fragile Knowledge

As mentioned in Chapter 2, I applied the original categories of fragile knowledge to the three specific aspects in my study: programming logic, block coding, and physical coding environment. The original categories are: **missing/partial** (the learner never learned the knowledge), **inert** (the learner failed to retrieve the knowledge for a specific context), **misplaced** (the learner used the knowledge at the wrong place), and **conglomerated** (the learner combined and applied multiple pieces knowledge incorrectly). Then, in my study, fragile knowledge could happen in all three aspects:

- **programming logic**: the knowledge of programming and logic co-shared with the common computer languages. For example, there is always a sequence of events to happen, a variable name is not necessarily equal to its meaning in a natural language, etc.
- **block coding**: the knowledge of how block coding works. For example, the blocks have different shapes for being able to connect each other, the colors indicate their functions, and predesigned

commands that only exist on the micro:bit's blocks such as "looping (melody) in the background", etc.

- **physical coding environment**: the knowledge of how digital and tangible components interact with each other in the physical coding environment. For example, the codes must be downloaded to the chip to show actual outcomes, the buttons could only respond at a limited speed, etc..

In total, there were potentially 12 types of fragile knowledge. A combined example of missing knowledge in programming logic was the learner did not know the idea and use of a variable in the program – a common phenomenon at the start of learning computational languages. Another example of misplaced knowledge in the physical coding environment that frequently showed up in my study was that some participants assumed pressing buttons A and B together meant resetting everything all the time, but the function only worked in Counter as coded. For more examples, please refer to the workflow in Appendix 2.

4.4.2 Process of Debugging

Mainly using raw data from the debugging activities in Session 1, I developed the codes for the process of debugging the participants demonstrated:

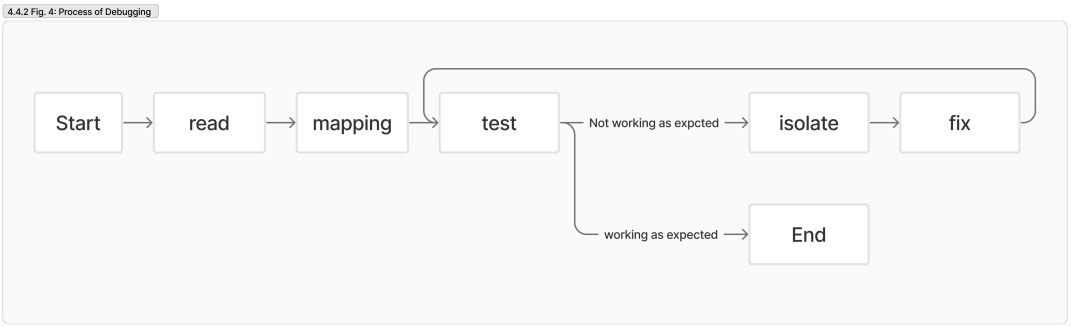

Figure 4: *Process of Debugging*

In this process, **read** means the participant the learner reads code blocks on the screen (usually for the first time) and tries to understand the meanings of each block and thus the coherent logic of the program. A typical learning and practicing action is when a learner reads out loud the texts on the block codes word by word. **Mapping** is the action that the participant draws connections between the commands of the program on the screen and the outcomes of the micro:bit. It starts with an overall understanding and later develops into step-by-step mapping. **Test** refers to the action when the participant the learner operates the physical chip (such as click, shake, tilt, etc.) and examines if the feedback of the chip is matched with the project goals as programmed or showing up at all. When the testing outcomes are not as expected, the learner intends to **isolate** the problem by finding the block(s) with error(s) on the screen, based on their mental model of the program and the feedback from the chip. At the beginner stage, they usually assume a single block carries the errors. Finally, the learner tries to **fix** the bugged block(s) by experimenting on probable solutions. The action is usually followed by another round of testing, which forms a full flow of iteration. The cycle ends whenever the test outcomes achieve the project goal.

4.4.3 Purpose of Interactions

Mainly based on Session 2 where several wide-wall creativities-focused artifacts showed up, I used the following codes to describe the participants' interaction with codes and micro:bit for the following purposes: **exploration** (looking for new blocks to expand their ideas), **creation** (setting a new project goal and code accordingly), **iteration** (running and adjusting the program to better deliver their ideas. A main activity within is the *process of debugging*), **organization** (aligning the grouped blocks in a logical sequence and deleting unused code blocks), **sharing/presentation** (showing other participants or adults their work and articulated their ideas). Due to the nature of computing in these artifacts, the types of *fragile knowledge* and the *process of debugging* inherently occurred, particularly during creation and iteration.

4.4.4 Issues of Interacting with the Physical Coding System

Driven by the results above, I then looked into the non-cognitive aspect (i,e., not caused by the *fragile knowledge* of programming logic) whenever errors or bugs showed up during the activities. I specifically focused on the design failures in the environment that prevented the participants from further engaging with the codes even when they knew what they would like to realize at a specific moment. The following codes revealed the key concerns I will further discuss in the next chapter:

- **Interactability**: (the visual clue of an element on the screen is ambiguous about its interactability
- **colored blocks**: blocks are colored-coded based on their categories, such as "Basic", "Input", "Music", "Led", "Logic", "Loop", etc. Blocks only become colored in the editing space after they are successfully connected with other blocks in the program.
- **shape inconsistency**: (different blocks have different outer shapes or inner holes, based on their properties and functions)

4.5 Analysis Strategy

The participants, because of their diverse personalities and backgrounds in computing, approached the projects and creative activities largely differently. Their experiences were not linear, either, but interestingly, some initiatives and strategies for solving problems and interacting with the physical coding environments are shared. To answer RQ1, I used the codes of the *process of debugging* to explain how the participants with and without previous experiences in programming approached my projects separately. For RQ2, I compared the data of the participants from both sessions and used the codes of the *purpose of interaction* to analyze the learning outcomes. Lastly, addressing RQ3, I analyzed the *issues of interacting with the physical coding system* and applied Gee's principles of "learning by design" in the context of this study to imply design improvements for playful debugging. The codes of *fragile knowledge* supported the analysis across situations.

Chapter 5: Discussion

In sections 5.1 and 5.2, I first discuss the initial reactions and tries of the participants on the debugging projects. I will distinguish the learning experience of two types of learners: newcomers and returners. Then I will present the learning progress of the learners in comparison between Session 1 and 2, addressing shared achievements in both groups. The sections respond to RQ1 and RQ2 cohesively. In sections 5.3 and 5.4, I will answer RQ3 and focus on the inadequacy in the micro:bit programming environment that prevented the participants from learning and successes of debugging. I will offer possible solutions for future improvements, based on the main issues I observed in the study.

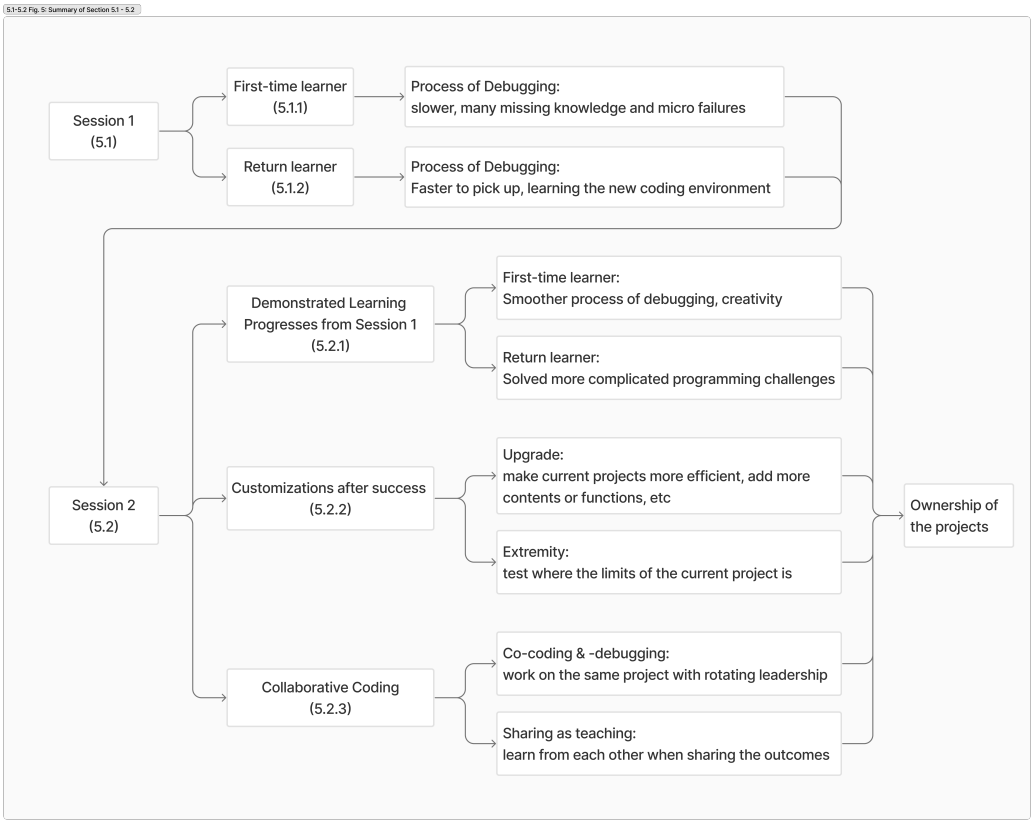

Figure 5: Summary of Section 5.1 – 5.2

5.1 Engage, but Challenging (Response to RQ1)

Most participants showed an intuitive process of iteration when they encountered the debugging projects, although the participants without programming experience took a longer process to be able to navigate in the setting. Responding to RQ1 (*How did the participants respond to the bugs they encountered and what they lacked if they failed to solve the bug?)*, I compared the types of *fragile knowledge* the participants with programming experience (returners) and without programming experience (newcomers) encountered in Session 1. The results presented that the newcomer participants spent more time facing missing knowledge in programming logic, block coding, and the physical coding environment. It meant that they were still at the phase of getting familiar with the environment, the systemic meaning of each digital or physical component, how the components interacted with each other, and what influence they as decision-makers and operators could have. Whereas, the returners already knew the essence of a programming activity so they encountered much less missing knowledge in programming logic but generated misplaced or conglomerated knowledge in block coding and the physical coding environment – the new conditions added to their already-known coding context. Yet, both groups' reactions conveyed a quick formation of iteration in response to the problem-solving nature of those activities and shared the difficulty of understanding where the codes actually live.

Before I go into detail about the two types of debugging processes, I want to note that a clear project goal was an indispensable premise to set great initiatives for the participants' attention. If the adult did not clarify the goal of the project and simply let the participant cluelessly wander around on the screen, the participant soon became confused and unconfident. For instance, in Session 1, when the adult showed Ray, a first-time learner[9], Counter Level 1 without explaining the purpose of the blocks, she started saying *"I don't understand. It's not doing anything. I don't understand why it won't change. This is hard.*" only two minutes after trying. Then, she soon claimed "*I'm not doing it again. I never liked math"*, although she presented great math skills in solving Floating Bubble Level 1 later. However, once another adult informed them of the goal for the level, she calculated quickly. Thus, what she actually disliked, based on her tone and wording, was being required to process a task without objectives.

5.1.1 Newcomers

Knowing the project goals, both groups of learners tended to form an intuitive process of iteration. The newcomers, however, needed a lot more time to first learn to treat block codes as cohesive programming elements that become a flow of commands and to establish a cognitive understanding of how the blocks on the screen would command the physical chip. They needed practice to "gather the necessary

[9] All returners in session 1 received clear instructions on the project goals, so no data was observed for this group in terms of their reactions to this issue. My assumption was that they probably would start asking for a project goal because they probably were aware of the existence and the need of a project goal, being familiar with programming in general.

information to understand what is happening within their...” physical coding “…system” (Schneider, 2023, p. 766). Meanwhile, their iterations of “*test – isolate – fix*” were slower, more cautious, and nonlinear. Basically, these participants were learning new knowledge in programming logic, block coding, *and* the physical coding environment for the first time, compared to the returners who focused more on the last two aspects.

In the *process of debugging*, the newcomers spent more time reading and establishing the mapping between the codes and the physical results on the chip before they started trying to isolate the problem by changing one thing on the screen each time. For example, when Oriana, a 9-year-old girl, worked on *Counter* Level 1, she first learned about the concept of coding, the fact that the codes on the screen and the micro:bit chip were both components that she could manipulate, and how the changes in the codes would command the changes on the chip.

[:04:08.000]

Adult: Yes, so, this is the code.

Oriana: Code?

Adult: Is this the first code?

Oriana: It's a code? (surprised) A code of what?

Adult: Yeah, so, have you ever used...I'm gonna show you [the demonstration video].

[:04:55.000]

Adult: So, see what happens, when Danyi presses the button? And you see what's happening?

Oriana: uh-huh...(processing)

Oriana: So you press both, it...?

Adult: If you press both, it resets to the beginning. So every time she pressed this (A) button...

Oriana: Nothing happened.

Adult: It does, it's just delayed. Oh no, I think she has to press B to get the number (to show the change).

Oriana: En-huh.

Adult: Let me see what I am allowed to tell you.

Oriana: So, all the numbers will come up here?

Then, she read the codes with the support of natural languages right after the adult encouraged her to “see what the codes say”. With such guidance, she realized that every time she clicked button A once, the number was changed by 2, which finally made her realize the problem in this context.

[:09:52.000]

Oriana: So (the code says) "on button A pressed"...press A...should I press A?

Adult: Sure.

Oriana: and B.

Adult: And what happens?

Oriana: It gave it 2.

Adult: En-huh, is this what we want it to do?

Oriana: No. It's supposed to give a 1.

Adult: So what do you think is wrong?

Oriana: It's a bug! That's not my answer.

She then began to try to debug. Because she had no experience in programming and no idea how programming worked, during the iterative process of *test – isolate – fix*, she developed a natural, cautious pattern in which she changed the program multiple times. For example, she clicked the variable "count" and saw the drop-down menu provided an unused variable "plus one", so she changed "count" to "plus one" for the functions of pressing button A, pressing button B, and pressing button A+B one by one each time. The "plus one" sounded right to her in the natural language, but the change was not an actual solution. Thus, she began to *hope* her changes solve the bug in the program. She started finger-crossing, leg-crossing, and even "cable-crossing"[10] for a fortunate result. As a result, her debugging process revealed a lot of missing[11] knowledge in programming logic, block coding, and the physical coding environment.

5.1.2 Returners

Compared to the newcomers, the returners could quickly apply their previous knowledge in the new context when they read, established the mapping, and entered the *process of iteration* with more confidence. Therefore, their experiences displayed much less missing knowledge in programming logic, at least until more advanced functions were required. However, the data also revealed diversity in misplaced or

[10] Because she so wanted a success, she became interestingly superstitious and crossed whatever she saw on site. She even crossed the cables of the adult's laptop charger and second screen.

[11] The use of "missing" here is to be aligned with the coherent application of *fragile knowledge*. As I mentioned at the beginning of this section, the newcomers were learning the knowledge freshly.

conglomerated knowledge in the block coding and physical coding environment when they tried to apply both internalized and new knowledge in the new space.

Returners initiated the first two steps, *read* and *mapping*, in the *process of debugging* fluently, and they also could *test*, *isolate*, and sometimes *fix* the problem – the following final three steps of the debugging process – with a more active awareness of what was going on, based on their understanding of the programming logic and comparing with code blocks on the screen. They also asked for less guidance from the adults during the process, and they would questioned details within their structural thinking. Adrian, a 12-year-old boy, when he first saw *Counter* Level 1, started reading the codes block by block and could build the mapping between the codes and the physical results immediately. After he found out the problem, he went to read the blocks again and located the bug almost immediately.

[:01:51.000]

Adrian: Well I think I figured out the bug. It's not going up.

Adult: Hmmm

Adrian: Huh, what the...ok, it just..32 (Pressed A and B in turns.)

Adrian: I'm just gonna reset. (Reset by pressing A+B immediately.)

Adrian: Oookay...it's giving me 22. (More pressing A and B in turns.)

Adrian: Oh! I figure out the bug. It multiplies everything by 2.

[:03:01.000]

Adrian: Haha, busted!! (He clicked "2" in the block "change count by" in the block "on button A", changed it to "1", and then looked through the rest of the code.)

Adrian was able to apply this process when he worked on *Floating Bubble* Level 1, too. Compared to those of the newcomers, his iterations of "*test – isolate – fix*" indicated logical thinking by trying potential solutions and collecting new feedback for further deduction after each run. And the cycles went fast. He was able to explain his solutions and the purposes of why he changed things in certain ways in his debugging process.

[:26:34.000]

Adrian: Danyi, I did it. Bug has been boosted! I fixed a bug in less than a minute. My brain is big. (Happy, show-off) This is too easy.

Adult: How? What did you do?

Adrian: So first, I just start writing in random numbers for X and Y. SoI did 10, 10. And it went here and here. And I thought that's weird. And I was like oh ok if it's 9 by 9...I think? Is it 9 by 9? No it's not. It's 4 by 4. Ok, I originally thought it's 9 by 9, so I put 3, 3. And it's here. (He pointed at a dot position on the LED pad.)

Adult: You originally thought what is 9 by 9? The whole thing (pad)?

Adrian: Yeah. So I thought this is 9 by 9, so I put 3, 3. And it was here. (Pointed at a different dot position.) Then I was like, what if I put 2, 3. And it went here. (Pointed at a different dot position.) So obviously it's 2, 2. Ok! So, that was easy.[12]

[12] He was able to solve *Counter* Level 2 in the same manner. For *Floating Bubble* Level 2, while he was able to apply the same set of skills, he did not finish it by the end of the session due to the increase in the number of variables and difficulty level.

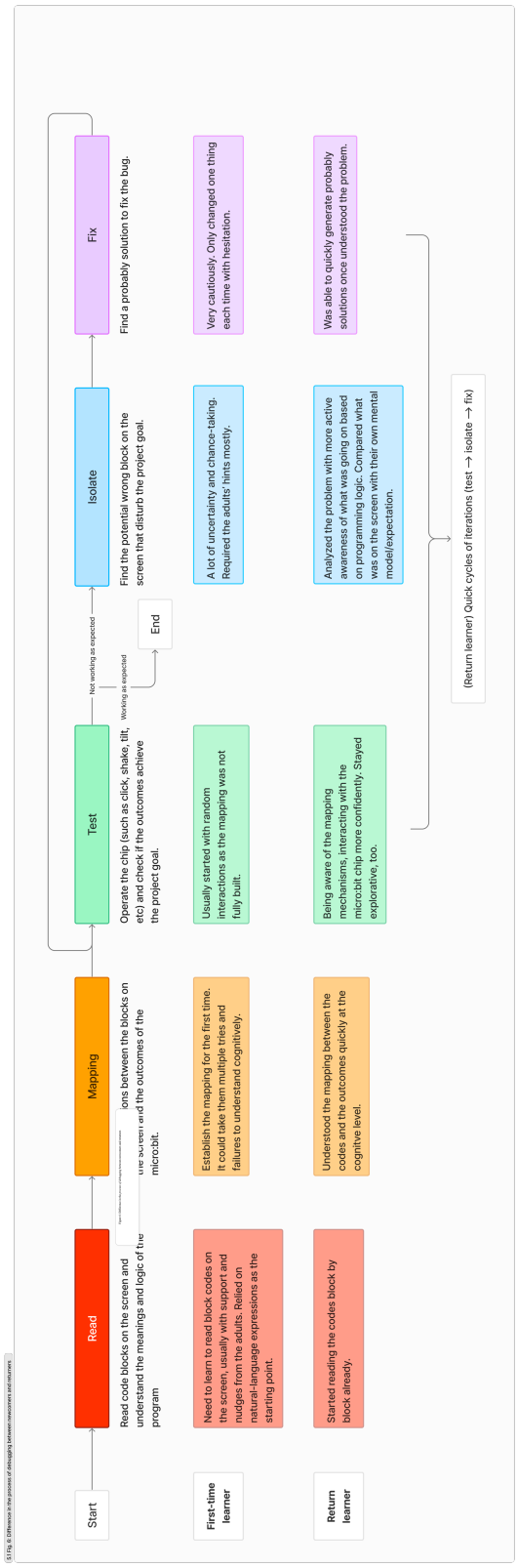

Figure 6: Difference in the *process of debugging* between newcomers and returners

5.1.3 Where do the codes live?

Both newcomers and returners showed an initial misunderstanding of the mapping mechanism at a more detailed level. For example, several participants, including Adrian (returner), Oriana (newcomer), and Aaron (newcomer), pressed buttons A and B together at the beginning of testing *Floating Bubble* or in other projects because they intended to reset the program as they succeeded in *Counter* Level 1. They did not realize that the function was commanded by *Counter*'s codes and thought it was already embedded in the chip. The action was caused by misplaced knowledge of the physical coding environment, as the participants did not realize what commanded the running chip and where some codes truly originated – in the chip or on the screen.

This was a salient moment of encountering a black box that, had the adult not provided an instant correct explanation, the children could easily struggle with confusion and impasse. Oriana, for example, never fully learned this piece of knowledge because she encountered the issue at the end of Session 1 and the adult was even equally confused by the actual location of the codes in Session 2. Adrian simply dropped the habit of pressing A and B together and switched to re-reading the codes of *Floating Bubble* as a new strategy of *isolating* the bug. Only Aaron was told immediately by the adult that there was no code in the chip permanently.

This black box moment indicated that the initial learning process in the physical coding environment required more specifications on what learners can grasp easily and what needs to be further clarified for the learners. Although my study did not provide adequate data on how the participants could learn that the physical chip was an empty container and it was the programmer's responsibility to pivot their project and import codes into the chip, it is a fundamental piece of knowledge that should and could be taught to children easily and engagingly at the beginning of learning in a physical coding environment. The code-emptiness of the chip is a key feature that allows programmers to (re)build programs and turn the chip into a useful object in different scenarios every time. Finding an efficient and attractive way to convey the knowledge and its meaning is indispensable for any progress in this context.

5.2 Progressing from Session 1 to Session 2 (Response to RQ2)

Based on the semantic codes and raw data, I concluded that almost all participants manifested progress in Session 2. They learned some Computational Thinking and specific functions of LEDs, music, etc. in Session 1 and successfully applied that knowledge in the wide-wall, creativity activities in Session 2. The newcomers, after their explorations in Session 1, were able to come up with interesting project goals and create programs using their newly developed knowledge about micro:bit and the iterative *process of debugging*. Meanwhile, some returners further solved more complicated programming goals with little technical support from the adults. More interestingly, many participants – whether first-time or return –

showed common passion in customizing their projects either through upgrading functions or testing the maximum capacity of their micro:bits. Lastly, the participants demonstrated a willingness to share their success and knowledge – either co-working or teaching their friends, while they held a strong sense of ownership of their creations as well as the programming processes during the whole time.

5.2.1 Learned Block-Coding Programming in the Physical-Coding Environment

Most participants learned to use code blocks for LEDs, buttons, and music-playing of micro:bit to create their projects in a diversity of ways. After the introductions of these functions that provided simple and instant feedback in Session 1, the participants in Session 2 easily recalled the exemplary use of those features. They came up with a clear goal for their programs and started block coding by assigning functions to the physically interactable features of micro:bit. Many of their projects included functions that showed LED patterns and/or played music after someone pressed a button, which implied that the participants actively used their experiences in *Counter* and *Floating Bubble* to create new ideas. The screen records disclosed that both newcomers and returners were invested in drawing specific LED patterns and selecting a favorite melody, given the freedom of creativity.

In such activities, the newcomers demonstrated a growing sophistication in the *process of debugging* and started learning more detailed functions of micro:bit. For example, when Oriana decided to make a name tag with the initials of her three

names, she already knew that the letters must be aligned in the right order under the block "forever", and immediately after she tested the outcomes on the chip, she concluded it was a success. This process proved that she set up a clear goal for her program, understood and followed the sequence of codes in blocking-coding, internalized the mapping between the codes and the outcomes on the chip, and knew when to stop coding and start testing based on the project goal.

As another example of newcomers, Ray constantly found new things at the brink of her comfort zone. She did not stay with the basic melody block she knew already but started trying other blocks under the "music" category. After she successfully[13] assigning a different melody to each of the functions of pressing A, pressing B, and pressing A and B – a structure largely similar to *Counter*, she extended her understanding of function assignments to the umbrella category *Input* where all the button-pressing conditional syntaxes belonged. She explicitly said "… I want to do one more 'Input'…2 more! I want 2 More!" when the adult suggested she try something new. Then, she patiently looked through the items in *Input*, which became an introductory learning moment for her. Later, when she experimented with auto-playing the music without pressing a button every time, she even tried *Loop*, another category, for the first time. It was another exciting success for her.

[13] In one of her tests, she found that the music was played not as expected. She immediately went back to check the codes line by line (a sign of understanding the sequence of codes) and realized the music was set up as played "in background" instead of "until done". She confidently isolated the problem, comparing between what was on screen and her mental model. This showed that she had become smoother in the *process of debugging.*

The returners, having understood the *process of debugging* already, learned the previously missing knowledge in block coding and started taking on more complicated challenges in Session 2. For instance, Bobby who mainly focused on *Counter* in Session 1 took up a completely new, semi-structural project called *Light Sensor*. (See the last link in Appendix 2 for the codes.) He was able to read the block codes on the screen, figure out what was missing in the codes, look through the unused blocks, and find the correct ones to fill in the unfinished codes in the correct logic. Then, he stayed on track with his potential solutions, referred to the project goal whenever needed, tested the outcomes, and debugged patiently without much support from the adult. Because the project demanded the micro:bit to show the light level of the environment, Bobby even improvised and used his cell phone light or covered the chip with his hand to change the environmental conditions for the testing. His example indicated that he had merged his past experiences with the new coding environment comprehensively, and the physical coding setup genuinely entertained his inquisitiveness in programming that generated interactive outcomes.

Although there were distinct differences in how newcomers and returners approached the physical coding projects, all participants who chose to participate in both sessions showed progress. They formed the concepts of debugging and physical programming at various levels and learned a few fundamental functions of micro:bit that allowed them to engage with programming at their own pace. At the end of Session 2, the participants gave clear definitions of debugging as “fix what’s wrong in

the code" and testing, meaning looking for correct outcomes on the chip, was an important step in the process of debugging. In the following subsections, I will discuss more shared features of the two groups of participants as signs of learning and creating as well.

5.2.2 Customization After Success

Looking into the codes of *Purpose of Interaction,* I realized that many participants were enthusiastic to tweak or upgrade their projects *after* they achieved success. Such interest-driven innovations were developed from what they had understood and created, and the process could be summarized by the *purpose of interactions* (exploration, creation, iteration, organization, sharing/presentation) mainly from Session 2. Yet, this tendency did not only appear in Session 2 where they could freely decide what to try, as even in Session 1 after they understood only a few blocks' functions, they would "mess around" and felt extremely happy when the changed version worked as expected. Specifically, there were 2 types of interest-driven customizations of the existing projects: 1) *upgrade* by making current projects more efficient, adding more contents or functions, etc., and 2) *extremity*, meaning the participants enjoyed testing the limits of their current project by giving it extreme commands to run.

Upgrade was common especially when the participants learned a new "trick"/ function for the first time, even when they did not know too much about micro:bit.

The environment was similar to what Gee described as "*Fish Tank*" (2005) where the participants had enough knowledge to make a change without being disturbed by too many choices. For example, Adrian (returner), after his first success in *Counter Level 1*, immediately suggested upgrading button A's function by transferring the current button B's function (show the counted number) to A, implying that then button B could do something else. This way, pressing A would "automatically" show the result and save an action of pressing B, which was actually also aligned with the design principle of providing was also a great design suggestion for creating an instant response! In Session 2, many newcomers were eager to add and edit more LED pads or melodies after their experiences with these features in Session 1: Ray and Helen combined music and LED, Oriana created a name tag with her initials, and Aaron (only attended Session 2) changed his *Counter Level 1* into a Pi Machine that show the digits of Pi. Eddie (returner) also worked on combining music and LED but challenged himself with a more complicated function of loop.

Extremity, another popular tendency among the participants when they customize their projects, referred to the actions when the participants pushed the limits of the current programs and challenged micro:bit to complete those seemingly crazy tasks. For example, Aaron coded his Pi Machine to count more than 30 digits and was impressed by the fact that micro:bit could *actually* realize his great plan. He ran to his friends and invited them to witness the success. Similarly, Adrian in Session 1, was modifying and testing multiple code blocks, aimed at making the *Counter* count to 1 million. Apart from creating large numbers, Ray edited her music

player to be super loud and fast, asking her friends to listen to it and laugh together. I found that the participants enjoyed such changes because they could see the shocking and usually goofy feedback immediately after their adjustments. The experience gave them a huge sense of achievement. They were amazed by the capacity of the small chip in their hand, and they also relished the moment when the goofy effects made their friends giggle, too.

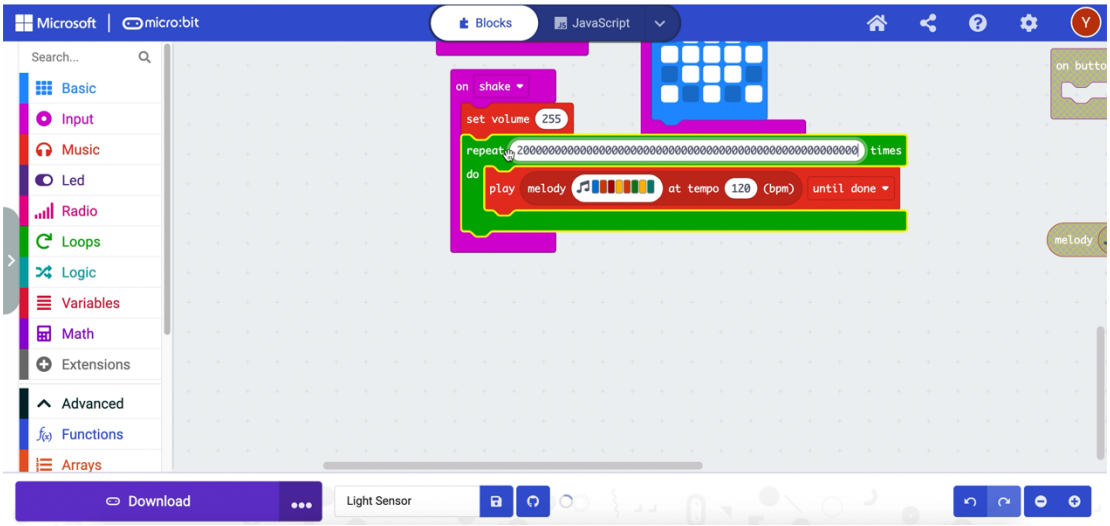

Figure 7: Example of *Extremity* – asking the chip to repeat so many times at the loudest volume.

These choices of customization, either *upgrade* or *extremity,* proved that the participants actively used their agency and sought fun in the learning process instinctively. In the carefree environment of KidsTeam, they were not afraid to create ridiculous yet interesting objects, and such explorations, in return, actually extended their knowledge of how programming and micro:bit could work.

5.2.3 Collaborative Coding

Another interesting phenomenon emerging from the *purpose of interaction* was that there were 2 spontaneous cases of collaborative programming and interactions with the physical coding components, given such a short period. There were no predominant instructions from the adults to arrange collaboration among the participants, but some kids either gradually paired up for programming or willingly invited their friend to join in at certain moments of the project.

The first case happened when Adrian and Eddie (brothers) were both working with one adult in Session 2. The original setting was each participant took turns to work on their separate projects, but they soon started to look at the codes together. When the adult raised an interesting challenge[14] promptly on Adrian's project, Eddie began to work on the challenge. He did not verbally confirm that he dismissed his own project, either. But, the result was Adrian and Eddie taking turns in leadership while they both worked on Adrian's project. Whoever sat in front of the laptop had the volition to edit the blocks. The adult and the supporter kid would provide ideas and inspirations, check the codes, see the results of testing together, and share the same sentiments within the group. When the leader ran out of ideas or the time was up, the supporter would take the leading position instead. The process formed friendly competitions between Adrian and Eddie who both wanted to impress the other, given

[14] The challenge was to play melodies and LED patterns repeatedly on micro:bit. Eddie, in the end, almost single-handedly solved the problem by applying his previously inert knowledge of block-coding!

their close relationship as both siblings and KidsTeam participants. Interesting, wild suggestions flew around, but they gave space for the other to think and try independently, too.

The second case happened when, in Session 2, Ray wanted to share her accomplishments, and the gesture of collaboration became a part of her demonstration of knowing the project confidently. After she decided the goal for her program and acquired the results on the micro:bit through her sequential blocks, Ray was confident to explain and present what she achieved. So, she called her friend Nova (the kid who initially decided not to participate in the activities at all) to come over and see the final artifact. The presentation and communication between the two kids, then, opened space for Nova to tinker around on the original project under Ray's "supervision". Because Ray clearly showed each button's interactive result to Nova and taught Nova how to modify a certain code block on the screen, Nova partially grasped the *mapping* between the codes and the outcomes effectively. Nova then discovered the feature "tempo" that Ray did not notice previously, which inspired Ray's learning reversely. Such collaboration based on peer teaching benefits both participants. For Ray, she was able to consolidate her knowledge through self-satisfying sharing and to continue her interest in creating on micro:bit. Nova, who verbally dismissed learning programming before, received the low-pressure, social-purpose-driven opportunities to take a first step out of her comfort zone.

From Session 1 to Session 2, the learning process of newcomers and returners, the customization driven by success, and the naturally occurring collaboration all suggested that the participants were learning and having fun in the bugging and physical coding activities. More importantly, they all showed strong ownership of their projects as they engaged more and more with their codes and micro:bit. They exclaimed "*what did you (do to my codes)? (grimacing)*" when they thought the adults messed up their codes. They used micro:bit to spell "L O V E" or their names as self-expressions. They exhibited their artifacts after success sincerely and proudly with their friends. Last but not least, they wanted their programs to work, feeling frustrated and confused yet *hopeful* whenever the little chips failed on them.

5.3 Failures without Feedback in the Micro:bit Environment (Response to RQ3)

Based on the semantic codes of *fragile knowledge* – particularly, those for block coding – in section 4.4.1 and issues of interacting in the physical coding environment from section 4.4.4, I also found a few inadequate user-experience designs that increased the complexity of learning for participants, either new to programming or new to physical coding. These designs prevented the learners from operating in the block coding space as they wished. Pointing out these issues structurally will guide a more comprehensive answer to RQ3 (*What are the implications for designing playful debugging activities?*). Unlike Scratch where most changes in codes can be seen directly and immediately on the screen (Resnick et al., 2009, p. 174), micro:bit had physical cables and carriers that "mystified" the process

of commanding from the screen. Even though micro:bit provided an emulator on the screen, the emulator did not present what a changed block meant within the program, either. Whether physical or digital, the feedback was always an overall final effect. Recall Oriana's switches between variable "count" and variable "plus one" for button A and button B: although each time she made a different change to the original program, the micro:bit always generated the same wrong number "0". The chip had no capacity to indicate the nuances between the differences among the *wrong* solutions.

The lack of differentiation among wrong solutions did not offer clearer clues to assist the participants, which I identified as "failures without feedback" (Williams-Pierce, 2020) in this context. In the current environment, the participants must learn through the failures that did not indicate what went wrong or why it was wrong and would have to deduce the reason why they failed, which requires more patience and guessing than the young participants could offer, especially at the beginning of the learning process. Here, I list the main types of *failures without feedback* I found through the analysis: 1) interactable elements or not? 2) unexplained visual clues for block connecting, 3) ambiguous category classification, and 4) natural or computational language? These failures were shared by both newcomers and returners, as the issues were mainly caused by the current user interface designs of micro:bit.

5.3.1 Interactable or Not?

It refers to the UI designs, although simple and friendly-looking most of the time, could confuse the participants when they do not know how to distinguish the interactable elements from the static information on the screen. For instance, Eddie clicked the text “melody”, hoping it would produce a list of interactions, but nothing happened on the screen at all. Oriana, furthermore, got stuck for more than 10 minutes on *Counter Level 1* because she did not know she could actually change the number “2” in the number box. Certainly, static information is indispensable for the communication between the participants and the codes, but the ambiguity between static and interactable elements hinders learners from knowing all possible options existing on the screen when they try to *isolate* problems during the *process of debugging* or programming in general.

5.3.2 Unexplained Visual Clues for Blocking Connecting

The UI of the micro:bit block coding space has a unified visual system corresponding to the nature of block coding, but the rules of this visual system are never taught directly to the learners. The key designs that frequently appear during programming, being supposed to support learner’s coding, are A) each block has an assigned color, and the color is a signal that the block has been successfully connected to another block. Or the block becomes grey. B) each block has a definite shape and can only be connected to another block that has a matched shape of edge or outline. Otherwise, the unconnected block will remain in grey as well.

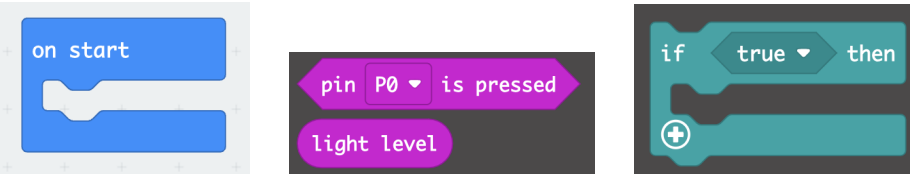

Figure 8. Code blocks of different shapes

These visual rules, also as knowledge of block coding, originally designed to help learners learn the knowledge of programming logic, are insufficient so far because the learners are not provided with explanations for any unsuccessful connections. Bobby, even when he confronted the challenge of *Light Sensor* in Session 2, asked "*Why is this (round block) not connected to the (square block)?*" after multiple failed tries. It was common that the participants tried multiple times before they gave up, so it implied that, although the animation and color changes were quite straightforward, the absence of an explanation pushed the participants to try several times because they strongly wanted to make the connection instinctively.

Even after they learned the visual rules on the surface, they did not understand the implicit reason derived from programming logic. For instance, when Ray failed to insert the variable "light level" under "on button A pressed", she learned the rule but she did not understand that it was because the variable must be combined with other blocks to become an achievable command/outcome, because "on button A pressed" was essentially a conditional syntax where a command/outcome lived in. Without an explanation as probing, nor did Ray consider alternatives to incorporate "light level"

under the condition of “on button A pressed”. The current visual rules of color-coded and shaped blocks actually intimidate the participants from continuous explorations.

5.3.3 Ambiguous Classification

Although micro:bit provided 10 categories of blocks (including *Basic*, *Input*, *Music*, *Led*, *Loop*, *Math*, etc.), the participants found that the naming and the content were not usually aligned with their own mental models. The main example appearing in my sessions was the location of the block of “show leds”. This block belonged to the *Basic* category, but the participants usually went to the *Led* category to find it at first because of the naming connections. Frequently, the participants needed to remember instead of recognizing (Nielson, 1994) the locations of the blocks, which was not intuitive or user-friendly in general. Meanwhile, because the *Led* category provided a completely different set of blocks for creating sprites on the LED pad of micro:bit, when they expected to see a direct drawing-pad-like block for them to directly create patterns of LED, their unsolidified understanding of the categories and functions went further challenged.

Similar to the interactable issue and unexplained visual codes, when there was no introduction to the categories on the screen, the participants tended to avoid completely new categories. The classification of the categories was designed to provide a laddered resource system of blocks for learners. Particularly, the *Basic* category, including “show leds”, “show number”, “show string”, “forever”, “on

start", etc., are introductory and friendly for new learners to start with. Yet, some categories' naming and content overlap with those of *Basic*. From the learners' perspective, the principles behind the classification became enigmatic. As a result, the fact that blocks under each category had a unique color did not provide any additional assistance for the learners either to understand the classification or to recognize a block immediately. In the typical example mentioned above, although the participants have seen the blue block "show leds" from *Basic* many times, they still clicked the purple *Led* category as their first instinct choice. As a result, they immediately ignored the content in *Led* after they failed to find "show leds", which indicates a lost chance of exploration and learning. The only exception was that, after Ray felt confident and accomplished about her project, she actively looked through different categories and intentionally sought originality for her purpose of upgrade

5.3.4 Natural or Computational Language?

As introduced at the beginning of the literature review, *preprogramming knowledge* is the most frequent cause of bugs among new learners of computing languages (Bonar & Soloway, 1985). Examining the *fragile knowledge* that occurred in the micro:bit environment, I found that similar issues – or more like a mental state – happened to the young participants, too. Most block codes carry texts, such as "show leds (the empty LED pad)", "repeat 4 times do (space for a block)", "if (space for a block) then (space for a block) else (space for a block)", etc., but in each of these text-embedded blocks, the text's degree of being natural or computational language is different. For instance, "show leds (the LED pad)", even in the context of

block programming, is easily understandable as if being a natural language expression. “Repeat 4 times do (space for a block)”, not correct in natural English’s grammar, is still comprehensible as a command of the code block[15]. “If (space for a block) then (space for a block) else (space for a block)” – the complete transplantation of “if…then…, else…” as a common programming syntax, however, is hard for the learners to understand and draw connections between those texts[16].

Grasping this variance of language property among the text blocks, never addressed explicitly in the coding space, largely relies on the learner’s previous knowledge of programming and/or adults’ assistance. In fact, the participants in my study tended to treat the texts as natural language, especially among the newcomers who were not aware of the existence of a difference between natural and computational languages. Certainly, these text-embedded blocks were designed to be more intelligible to help the participants navigate through the program line by line, but because they looked so natural the participants did not always realize their computational significance in the program. For example, when Oriana tried to decipher the blocks in *Light Sensor,* she never considered the probable existence of the logic connections of the empty blanks for “if (space for a block) then (space for a block) else (space for a block)”. While it is important to leave space for first-time learners to explore without considering much in terms of computational language,

[15] Both Eddie (returner) and Ray (newcomer) showed no difficulty in using this block for the first time in their projects.

[16] Oriana (newcomer) was stuck on this block for a long time in Session 2. Adrian and Bobby (returners), with internalized knowledge of programming, recognized this block immediately.

there could have been hints or hidden opportunities for them to be attracted and jump into a rabbit hole occasionally to learn the transition from natural to computational languages.

Figure 9: Example of misconceptions of block coding language.

(Oriana, with no understanding of the conditional syntax, tried to make "light level" (the variable) = "light level" (the string). But here it's a conditional statement, not an assignment, so the right side must be a numerical value.)

5.4 Implications for Designing Future Debugging Activities (Response to RQ3, continued)

5.4.1 Identify the Learners and Provide Corresponding Environments

Based on the findings of two types of learners, the children's willingness to customize after tasting success, unexpected collaboration among the participants, as well as the current failures without feedback in the micro:bit environment, I propose to further develop Gee's learning principles of *Fish Tanks* and *Sandboxes* to improve debugging activities in the current micro:bit environment: the programming space should consider a clearer distinction between the *Fish Tanks* space and the *Sandboxes* space, with each highlighting different features.

Proposed as two key principles of *Learn by Design*, *Fish Tanks* and *Sandboxes* are two concepts that cultivate a problem-based learning environment (Gee, 2005a; Thorn, 2013), which has the potential to support debugging as problem-solving learning opportunities in nature. Gee's framework starts with advocacy of shaping learning experiences as good-game-like journeys where learners/players willingly take challenges, voluntarily spend time internalizing and applying the knowledge in the context to overcome those challenges, and finally exercise the knowledge beyond into the real world comprehensively. Specifically in the aspect of problem-solving, as the framework emphasizes the importance of not just reciting facts but actively utilizing facts to identify and solve problems, *Fish Tanks* and *Sandboxes* provide two kinds of environments supporting the needs of learners to navigate through the complexity of knowledge systems at low risks. *Fish Tank*s are simplified learning spaces of real-world situations that contain only the key variables and their interactions initially, so learners/players can make small steps in understanding the basics and add new factors at one's own pace. *Sandboxes*, on the other hand, are open spaces that feel like the real world but the learners/players can explore and fail at new things safely. Both environments, essentially, give learners/players abundant time and scaffolds to interact with indispensable knowledge for their success without pressing standardized progress.

Situated in learning debugging on micro:bit, *Fish Tanks* are suitable for *newcomers* to first build the habits of *reading* blocks and *mapping* the relations

among the physical coding components, with more visual hints for learners to understand the knowledge system and the interactions within faster – which will be further discussed in the next section. Then, transferring into the *Sandboxes* environments, learners can have more opportunities for them to use the knowledge acquired in *Fish Tanks*. When debugging on the micro:bit as novices, both newcomers and returners need to learn different aspects of *fragile knowledge* on the screen and in the physical environment. Therefore, *Fish Tanks* will allow the participants to only interact with fewer and simpler elements at first and add one or two new elements in each round of learning. They can use the *Fish Tank* activities and environments to internalize information at a controllable pace. Particularly, there should be enough time and steady "challenges" for them to learn about *reading* the blocks in sequence and discover the *mapping* between digital and physical components micro:bit. The environment could first limit its visual and interactable choices but increase hints on the screen, so the learners can first understand the basic elements, programming logic, and the interaction between digital and physical components gradually, through which the participants would establish a good habit of understanding block coding as a computational language and support them to run iterations more smoothly in later challenges.

With a solid practice at *Fish Tanks*, the participants then can enter *Sandboxes* activities to continue applying their knowledge in a more diverse context and encountering new knowledge or challenges as the learning goes. Moreover, the environment can encourage customizations built upon success and collaborations, as

such intentions naturally emerge among the learners. After learners acquire fundamental ability of *reading* and *mapping*, they need space that is pliable and diverse enough for them to start debugging, completing the iteration of "*test – isolate – fix*". Because *Sandboxes* allow learners to try new things at low stakes, they are highly aligned with the nature of debugging as problem-solving experience as well. Particularly for returners who enjoy fast tries, *Sandboxes* will allow and encourage them to experiment with new ideas and seek solutions through failures without imposing risks or punishment accordingly. This environment appropriately reflects the iterative process of problem-solving but allows learners to follow their diverse methods to approach the challenges. Furthermore, as learners enjoy customizing their projects and/or collaborating with their peers after their initial success, *Sandboxes* have the capacity to offer more tools and elements that support the participants in developing their programming as well as social-emotional skills when responding to those needs. Therefore, the agency-determined process of learning in *Sandboxes*, will allow learners to develop fundamental knowledge and skills for computational thinking without following a unified structural learning routine. The experience itself, then, comprehensively supports the projective stance of Gee's view, as I discussed in the literature review.

In short, clarifying whether the learners are entering a *Fish Tank* or *Sandbox* should be a design awareness when planning the learning activities. There should be more visual hints to support the learners in familiarizing themselves with the coding environment and learning the initial steps of the *progress of debugging – reading* and

mapping in *Fish Tanks*. Then, more flexibility and support for customization after success and collaboration will help learners to continue learn and explore with their interests in *Sandboxes.*

5.4.2 Externalize Knowledge Through Instant Visualization

Adding step-by-step visual hints could better support the learners in cultivating their *process of debugging*, providing failures *with* feedback in *Fish Tanks.* Addressing that running programs step-by-step could slow the process of debugging down and was even fun to interact with (Sipitakiat & Nusen, 2012) , I further suggest that there should be visual informative scaffolds to support the mechanism of running programs step-by-step, too. As explained above, learners need to identify which elements on the screen are interactable, why shape variance exists, what are the differences among the categories, and how to read the texts from a Computational Thinking perspective. These are necessary conditions or fundamental skills for learners to engage in debugging activities. For example, when two blocks cannot be clutched together, a pop-up box with a warning sound effect – like in games – can provide a brief explanation on the functions of the two blocks, possible reasons why they cannot be combined, or suggestions of iterations, etc. For another example, the interactable and static texts could be visually different in font sizes, colors, or any other obvious clues for learners to identify the contrast.

Moreover, the participants sometimes knew something went wrong but they still had missing or inert knowledge to think through. A step-by-step visualization (Resnick et al., 2009, p. 147) of the outcome from each line of the codes would be easier for the participants to learn how to read and map the codes and outcomes. In the following example, when button A is pressed nothing will show on the emulator or on the micro:bit chip if the codes are downloaded. The variable "count", however, has already acquired a new value, 2, through the action.

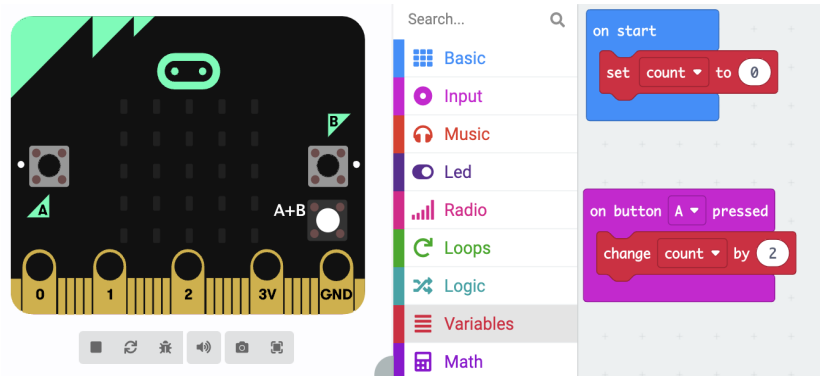

Figure 10: Example of no step-by-step visualization

It would be clearer for the young learners to actually *see* that the "count", as a variable, has a changed value now on the screen. Maybe in a side-by-side floating box, texts and pictures could illustrate how micro:bit takes and processes the orders in a linear fashion. Learners need more specific scaffolds to understand what happens *within* the black box instead of the sole last-step outcomes.

Finally, explicit explanations of the failures for the beginners of block coding will become new learning opportunities for the learners to know the environment as well as consider new possibilities to solve problems. A promising direction to

enhance the learning experience is to offer instant AI-supported analytical feedback. Learning through debugging pre-installed bugs in goal-settled projects is a perfect scenario. The AI can help analyze the codes, based on the project goals, and provide hints such as “check your spelling”, “think about the order of the blocks within on button A pressed’”, etc. that can help cultivate learners’ sophistication through the *process of debugging*. This facilitation will provide additional assistance especially when the adults lack adequate knowledge either in the physical coding environment or general programming, too.

5.4.3 Promote Customizations and Collaborations

Customizations after success and spontaneous collaborations are great opportunities for transitioning the learning context from *Fish Tanks* to *Sandboxes*. Because *Sandboxes* usually provide safe, diverse variables for the learners to experience real-world situations (Gee, 2005), creating spaces for customizations and collaborations allows the participants to start using their knowledge learned in *Fish Tanks* to continue learning in diverse situations. After the initial success, the participants prefer upgrading their projects or finding the limit of micro:bit. Thus, it would be appealing and educational to the participants if the activities tolerate or even encourage them to “do something silly”.

Similarly, allowing and encouraging collaborations will create opportunities for the participants to teach their peers – a fundamental way of consolidating

knowledge – and, reciprocally, to learn from their teammates. Different from *Fish Tanks* where the learners are still absorbing information, *Sandboxes* champion the plurality of solutions. Debugging, like most programming activities, usually ends up with multiple possible solutions, like in *Counter* Level 2, so collaborations with peers allow diverse perspectives to communicate with each other. Through both forms of engagement, the participants will have new chances to learn debugging, physical coding, and problem-solving in a broader sense.

In this chapter, I synthesized my observations and reflections of the progress and failures in implementing learning experiences in the current micro:bit environment. Then, using the findings, I provided a thread of suggestions for future activity development based on the design principles of *Fish tanks* and S*andboxes* for promoting playful learning experiences. These insights emerged from my 2-session research within a short period of execution, presenting one, among many, possible direction to address the advantages and issues for learners to debug in a current physical coding system.

Chapter 6: Conclusion

6.1 Key Findings

Joining the emerging trend of applying physical coding as a medium for Computational Thinking education, I inquired how tweens could learn debugging on micro:bit playfully. I designed playful projects with pre-installed bugs and offered wide-wall programming activities in order to collect qualitative data of 8 participants' learning experiences. As direct learning outcomes, both newcomers and returners found the LED patterns, music, and buttons from Session 1 easily learnable, and continued to create their projects based on these functions in Session 2. They demonstrated progress in utilizing the sequence and containment of codes in block coding and tinkering with individual blocks whenever needed. They also started combining those functions into layered commands, making their micro:bit more interactable and entertaining.

Through the semantic coding of the participants' learning experiences, I successfully applied and extended Perkins & Martin's *fragile knowledge* within the context of physical coding, further distinguishing the information and skills learners needed for the programming logic, block coding, and physical coding environment. I also identified two types of learners (newcomers and returners) and studied how they debug respectively via the *process of debugging*. I define the *process of debugging* as how young learners look for errors and try to fix them with iterations: they must first learn to *read* the block codes and establish the *mapping* between the codes on the

screen and the results on the micro:bit chip. Then, they initiate the cycle of "*test* the outcomes on the chip – *isolate* the bug on the screen – *fix* the bug by correcting one or multiple block codes". The newcomers took more effort to learn *reading* and *mapping* before they started the iteration, while the returners tended to jump into the iteration with their understanding of general programming. The participants' learning experiences resonated with the findings of previous research on how bugs involved conceptual errors in transferring their solutions into codes and the physical objects might pose as much challenging as interesting to the learners (Jayathirtha et al., 2018).

Moreover, I reported on the learners' passion for customization after success (*upgrade* and *extremity*) and collaborations (*co-debugging* and *sharing as teaching*) during the learning-creating process. *Upgrade* refers to the learners' interest in optimizing the current projects based on the learners' management of the projects' functions and their allocations on the chip. *Extremity* is the strong desire to witness the edge of the chip's calculation or hardware capacity, ubiquitous among young learners and frequently followed by a moment of jovial sharing. For collaborations, some participants started *co-debugging* with rotating leadership, and some participants unexpectedly learned from each other during the time of sharing. As most previous research concentrated on the direct impact of tangible objects and tools on debugging activities, my insights on the intangible actions and choices the learners created during their debugging process supplement the context and provide a different view to examine the benefits and inadequacy of physical computing.

Meanwhile, I discussed the issues – as failures without feedback from a learner's perspective – currently existing in the physical coding environment, the issues that prevented the participants from better understanding the program and smoothly debugging: the difficulty in identifying interactable elements on the screen, the hidden reasons behind the blocks with heterogeneous shapes, the ambiguity in the classification of the categories, as well as the unclarified mixture of natural-language expressions and computational syntaxes among the block coding elements. These difficulties reflected a similar situation several studies addressed in other physical computational kits, especially for novices having trouble connecting physical objects and the burdens caused by the low visibility of codes' outcomes (Booth & Stumpf, 2013; Sadler et al., 2017).

Lastly, I synthesized the successes and failures that occurred in the participants' learning experience and applied Gee's design principle of *Fish Tanks* and *Sandboxes* to explore improvements for the future design of debugging activities in a physical coding environment. The designers and educators of these debugging activities can start by considering their main audience being either newcomers or returners because their learning and practice curves of the *process of debugging* are different. Meanwhile, providing step-by-step visualization of what is happening inside the black box of the program and instant, contextualized feedback after failures will help learners learn the programming logic and how it is achieved through the physical coding environment. Last but not least, encouraging customizations after

success and collaborations not only can sustain learners' interests in explorations but also provide unexpected reciprocal learning moments. These suggestions add contributions to the ongoing discussion that addressed the "physical and visual affordance" in the environment and how to "organize work, externalize knowledge, and create new demands for problem-solving." (Deitrick et al., 2015; DesPortes & DiSalvo, 2019)

These findings contribute to understanding how children approach debugging and how future physical coding activities could be more playful, bridging the classical framework with the emerging trend of teaching programming through debugging. They are developed based on a respect for the diversity in the learners' personalities, interests, study curves, etc. The outcomes advocate for a learner-centered experience, seeking opportunities to increase fun in block coding and reduce frustrations caused by the computational environment. The study resonated with the ongoing request to educate CT for a broader spectrum of audiences and addressed multiple ongoing conversations in learning in physical computing.

Before the first session of KidsTeam after the winter break, we were picking up the kids at the loading deck as usual. I was surprised and brightened when a kid who participated in my study happily told me that they asked for and received a micro:bit as a Christmas gift. And, they had been trying to recreate *Counter* on their own micro:bit ever since.

6.1 Limitations

Several factors determined that my study conclusions and insights have their limitations. First, KidsTeam is an informal learning environment that accepts and even nurtures playfulness, wild ideas, open-ended answers, equality among educators and learners in a participatory design setting, etc. Thus, if one wants to apply my findings to a formal classroom or other more traditional education spaces, one must consider the social norms that might prevent the full transferability of the practice and its effects. Second, my designs of *Counter* and *Floating Bubble* were still introductory in terms of domain knowledge and playful mechanisms. They were, on the one hand, forthcoming and relatable but, on the other hand, insufficient to provoke every type of approach and strategy the participants might use for debugging. Nor did the designs consider accessibility, given the limited time and sources for preparation. More diverse designs should be provided to further examine the insights from this study. Third, because of the absence of several pieces of data and a small group of research subjects, my research could not provide more specific analysis on the programming concepts and skills, such as “loop”, “for”, etc. to evaluate the participants’ learning experiences more accurately. Last but not least, as all the participants reacted differently to those programming activities, the adult researchers also brought their diverse backgrounds and individual perspectives into the context. My research cannot reflect on the effects of such miscellany but can only acknowledges its potential influence.

6.2 Future Research

For future research, I find the following directions already interesting: 1) since I refined the *process of debugging* and the *process of interactions* from my data, the next step would be comparing them with the previous research and analyzing the differences and similarities, especially among different physical computing kits. 2) Digging into the design principles for *Fish Tanks* and *Sandboxes* activities, how could the activities best utilize the physical coding features responding to the respective needs of *Fish Tanks* and *Sandboxes*? What advantages does a combination of multiple physically interactable features (like, LED and music, pressing buttons and shaking the chip, etc.) have in this context? 3) Based on the difference between newcomers and returners, what content and support are more essential in those activities when they engage with physical coding for the first time? And, I believe there are more personas beyond these two types of learners, so who are they? 4) from a reflective standpoint, many projects on the micro:bit, including *Counter, Floating Bubble, Light Sensor,* etc., have the potential of being developed and experimented outdoors. How can we "break the wall" and merge this small gadget with outdoor activities safely and playfully? 5) because, like what happened in my session, the adults in an informal STEM education space do not always own the most expertise in programming (DeLiema et al., 2024; Elliott et al., 2022), how could we develop practical instructions to help our peer researchers and educators facilitate these activities? When and what should they provide when the learners need the support?

Appendix 1: Semantic Code

Type of fragile knowledge

Categories:

Missing/partial knowledge (n= 27)

Definition: the knowledge that learners have not obtained completely or correctly

Inert knowledge (n= 9)

Definition: the knowledge that learners fail to retrieve from their memory under certain contexts

Misplaced knowledge (n= 13)

Definition: the knowledge that learners use at the wrong place, although the knowledge itself is correct

Conglomerated knowledge (n= 4)

Definition: multiple pieces of knowledge that learners combine and apply incorrectly

Types:

Programming logic knowledge (n= 24)

Definition: the knowledge of programming and logic co-shared with other common computer languages, such as the sequence of events, the property of variables, conditional logics, etc.

Block Coding knowledge (n= 18)

Definition: the knowledge of how block coding works, including that blocks have different shapes for being able to connect to each other, the colors indicate their functions, and predesigned commands that only exist on the micro:bit's blocks such as "looping (melody) in the background", etc.

Physical Coding environment knowledge (n= 10)

Definition: the knowledge of how digital and tangible components interact with each other in the physical coding environment, such as, the codes must be downloaded to the chip to show actual outcomes, the buttons could only respond at a limited speed, etc.

Common co-appearing examples	Programming logic	Block Coding	Physical Coding environment
Missing/partial knowledge	Several participants did not know how "variable" worked. They thought the name of a variable was the function literally and treated them so.	Oriana, even though she realized the bug logically, did not know she could click the numerical element on the screen as a way to fix the bug.	Oriana could not isolate a block out of a chain and accidentally deleted it. She was scared and did not know how retrieve it.
Inert knowledge	Bobby found a bug happened because he forgot to add the key variable into the main block.	Eddie initially failed at achieving his project goal because he forgot a small function within a block commanding the outcome, until he went through the codes and fixed it.	Several participants forgot to "download" revised codes to the chip, and mistakenly concluded that their fixing was not correct.
Misplaced	Eddie applied the numerical rule of detecting the tilt degree of the physical chip wrongly to a block that commanded the position of a LED dot on the micro:bit chip.	Several participants could not drag a variable block directly under a conditional block (because of the shape consistency), which was counter-intuitive to their previous experiences.	Several participants thought the micro:bit chip had a default function of resetting by pressing button A and B simultaneously, due to their experience with the *Counter* projects.
Conglomerated	When Adrian tried to fix a bug, he changed a numerical element	Eddie, in one iteration, failed at success because, although he chose	Helen pressed a key on the keyboard when she tried to adjust the volume

	and a variable at the same time, thinking that together would work, but it just made the context a little bit more messier.	the blocks correctly, the order was not right due to some specific predesigned features of the code blocks.	sound of the micro:bit chip.

Process of Debugging

Nuance: due to the time and analysis limit, I did not count the number of occurrences for each step in the data. My judgment was the data reached saturation for this process.

Read

Definition: the learner reads code blocks on the screen (usually for the first time) and tries to understand the meanings of each block and thus the coherent logic of the program. A typical learning and practicing action is when a learner reads out loud the texts on the block codes word by word.

Mapping

Definition: the learner draws connections between the commands of the program on the screen and the outcomes of the micro:bit. It starts with an overall understanding and then develops into step-by-step mapping.

Test

Definition: the learner operates the physical chip (such as click, shake, tilt, etc.) and examines if the feedback of the chip is matched with the project goals as programmed or showing up at all.

Isolate

Definition: when the testing outcomes are not as expected, the learner intends to isolate the problem by finding the block(s) with error(s) on the screen, based on their mental model of the program and the feedback from the chip. At the beginner stage, they usually assume a single block carries the errors.

Fix

Definition: the learner tries to fix the bugged block(s) by experimenting on probable solutions. The action is usually followed by another round of testing, which forms a full flow of iteration. The cycle ends whenever the test outcomes achieve the project goal.

Purpose of Interactions

Exploration (n= 7)

Description: the learner looks for new blocks to expand their ideas

Example: Helen looked through the melodies blocks and the editing panel of the blocks to choose a piece of music for her micro:bit player.

Creation (n= 11)

Description: the learner sets a new project goal and codes accordingly

Example: Ray designed and programmed music and LED patterns to each of the buttons on the micro:bit chip.

Iteration (n= 8)

Description: the learner runs and adjusts the program to better deliver their ideas. A main activity within is the *process of debugging.*

Example: Eddie wanted the micro:bit to play and pause the music and rotating of LED patterns at the same time, so he tested different combinations of the blocks and commands in multiple runs.

Organization (n= 2)

Description: the learner aligns the grouped blocks in a logical sequence and deletes unused code blocks.

Example: Ray aligned her useful blocks on the top left corner of the screen, with close gaps between grouped blocks, and deleted the gray, unused ones.

Sharing/presentation (n= 4)

Description: the learner shows other participants or adults their work and articulates their ideas.

Example: Aaron excitedly showed his Pi machine to his friends nearby, explaining that the micro:bit could count the digits of Pi constantly.

Issues of Interacting with the Physical Coding System

Ambiguous interactability (n= 5)

Definition: the visual clue of an element on the screen is ambiguous about its interactability.

Example: Eddie clicked the word “melody” in a melody block, expecting it would provide a dropdown menu like “shake” in another block, but nothing happened to the melody block.

Colored blocks (n= 2)

Definition: the blocks are colored-coded based on their categories, such as “Basic”, “Input”, “Music”, “Led”, “Logic”, “Loop”, etc. Blocks only become colored in the editing space after they are successfully connected with other blocks in the program.

Example: Oriana

Shape Inconsistency (n= 10)

Definition: different blocks have different outer shapes or inner holes, based on their properties and functions.

Example: Bobby tried to insert one block into another, but he could not because the shapes were not compatible.

Appendix 2: Session Plans, Workflows for Session 1, Wide-Wall Activities of Session 2

Session 1 Plan:

https://docs.google.com/document/d/1ecHcvGy10jYcl47mD8Ow1MyndcrtZiK8l7yB7BNV_cE/edit?pli=1

Session 2 Plan:

https://docs.google.com/document/d/1gQyhzyH5xBCuKpxG3tTYUCmwrR_-lJibeezDCLHwJjE/edit#heading=h.83tbgrdeg72c

Workflow for *Counter* and *Floating Bubble*

https://docs.google.com/document/d/1ciJ0FY51U1F4G4E1VGGg6LDCphrhnk0FfqkqR4WlzQo/edit#heading=h.br2kascixdia

Wide-wall activities for Session 2:

https://docs.google.com/document/d/1oZCDImqgvB45uHYAykvYw2kFOYbfRpErt6PGQ72w6DA/edit

Bibliography

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, *37*(3), 84–88. https://doi.org/10.1145/1151954.1067472

Begnaud, D., Coenraad, M., Jain, N., Patel, D., & Bonsignore, E. (2020). *"It's just too much": Exploring Children's Views of Boredom and Strategies to Manage Feelings of Boredom*.

Blikstein, P. (2013). Digital Fabrication and 'Making' in Education: The Democratization of Invention. *Bielefeld: Transcript Publishers*.

Bonar, J., & Soloway, E. (1985). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers. *Human–Computer Interaction*, *1*(2), 133–161. https://doi.org/10.1207/s15327051hci0102_3

Bonsignore, E., Quinn, A. J., Druin, A., & Bederson, B. B. (2013). Sharing Stories "in the Wild": A Mobile Storytelling Case Study Using StoryKit. *ACM Transactions on Computer-Human Interaction*, *20*(3), 1–38. https://doi.org/10.1145/2491500.2491506

Booth, T., & Stumpf, S. (2013). End-User Experiences of Visual and Textual Programming Environments for Arduino. In Y. Dittrich, M. Burnett, A. Mørch, & D. Redmiles (Eds.), *End-User Development* (Vol. 7897, pp. 25–39). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-38706-7_4

Chalmers, C. (2018). Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, *17*, 93–100. https://doi.org/10.1016/j.ijcci.2018.06.005

Deitrick, E., Shapiro, R. B., Ahrens, M. P., Fiebrink, R., Lehrman, P. D., & Farooq, S. (2015). Using Distributed Cognition Theory to Analyze Collaborative Computer Science Learning. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, 51–60. https://doi.org/10.1145/2787622.2787715

DeLiema, D., Hufnagle, A., & Ovies-Bocanegra, M. (2024). Contrasting stances at the crossroads of debugging learning opportunities. *British Journal of Educational Psychology*, bjep.12666. https://doi.org/10.1111/bjep.12666

DesPortes, K., & DiSalvo, B. (2019). Trials and Tribulations of Novices Working with the Arduino. *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 219–227. https://doi.org/10.1145/3291279.3339427

Elliott, C. H., Chakarov, A. G., Bush, J. B., Nixon, J., & Recker, M. (2023). Toward a debugging pedagogy: Helping students learn to get unstuck with physical computing systems. *Information and Learning Sciences*, *124*(1/2), 1–24. https://doi.org/10.1108/ILS-03-2022-0051

Elliott, C. H., Chakarov, A. G., Bush, J. B., & Recker, M. (2022). *Debugging Pedagogies: Helping Middle School Students Learn to Get Unstuck With Physical Computing Systems*.

Fails, J. A. (2012). Methods and Techniques for Involving Children in the Design of New Technology for Children. *Foundations and Trends® in Human–Computer Interaction*, *6*(2), 85–166. https://doi.org/10.1561/1100000018

Fields, D. A., Kafai, Y. B., Morales-Navarro, L., & Walker, J. T. (2021). Debugging by design: A constructionist approach to high school students' crafting and coding of electronic textiles as failure artefacts. *British Journal of Educational Technology*, *52*(3), 1078–1092. https://doi.org/10.1111/bjet.13079

Fields, D. A., Lin, Y., Jayathirtha, G., & Kafai, Y. B. (2021). A Redesigned Reconstruction Kit for Rapid Collaborative Debugging and Designing of E-Textiles. *Proceedings of the FabLearn 2020 - 9th Annual Conference on Maker Education*, 98–101. https://doi.org/10.1145/3386201.3386207

Fields, D. A., Searle, K. A., & Kafai, Y. B. (2016). Deconstruction Kits for Learning: Students' Collaborative Debugging of Electronic Textile Designs. *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education*, 82–85. https://doi.org/10.1145/3003397.3003410

Fisher, C. (2015). *Designing games for children: Developmental, usability, and design considerations for making games for kids*. Focal Press/Taylor & Francis Group.

Ford, A. R., & Teorey, T. J. (2002). *Practical Debugging in C++*. Prentice Hall.

Gee, J. P. (2005a). *Learning by Design: Good video games as learning machines*. *2*(1).

Gee, J. P. (2005b). Pleasure, Learning, Video Games, and Life: The Projective Stance. *E-Learning*, *2*. https://doi.org/10.2304/elea.2005.2.3.2

Hardware. (n.d.). Retrieved February 25, 2024, from https://tech.microbit.org/hardware/

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, 153–156. https://doi.org/10.1145/611892.611956

Jayathirtha, G., Fields, D. A., & Kafai, Y. B. (2018). *Computational concepts, practices, and collaboration in high school students' debugging electronic textile projects*.

Jayathirtha, G., Fields, D., & Kafai, Y. (2024). Distributed debugging with electronic textiles: Understanding high school student pairs' problem-solving strategies, practices, and perspectives on repairing physical computing projects. *Computer Science Education*, *0*(0), 1–35. https://doi.org/10.1080/08993408.2023.2297738

Johnson, M. J., Castro, F. E. V., DiSalvo, B., & DesPortes, K. (2023). Chronicles of Exploration: Examining the Materiality of Computational Artifacts. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, *1*, 29–47. https://doi.org/10.1145/3568813.3600132

Kafai, Y. B., Lee, E., Searle, K., Fields, D., Kaplan, E., & Lui, D. (2014). A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles. *ACM Transactions on Computing Education*, *14*(1), 1:1-1:20. https://doi.org/10.1145/2576874

Kafai, Y., Hutchins, N., Snyder, C., Brennan, K., Haduong, P., DesPortes, K., DeLiema, D., Aalst, O. W., Flood, V., Fong, M., Fields, D., Gresalfi, M., Brady, C., Steinberg, S., Franklin, D., Eatinger, D., Coenraad, M., Palmer, J., Weintrop, D., … Bulalacao, N. (2020). *Turning Bugs into Learning Opportunities: Understanding Debugging Processes, Perspectives, and Pedagogies*.

Kaplan, E., Griffin, J., Kafai, Y. B., & Burke, W. Q. (2011). A Deconstruction Kit for the LilyPad Arduino: Designing Debugging Sets for Learning about Circuitry & Programming for High School Students. *The SIGCSE Conference*.

Knuth, D. E. (1989). The errors of tex. *Software: Practice and Experience*, *19*(7), 607–685. https://doi.org/10.1002/spe.4380190702

Ko, A. J., & Myers, B. A. (2004). Designing the whyline: A debugging interface for asking questions about program behavior. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 151–158. https://doi.org/10.1145/985692.985712

Kong, S.-C., & Abelson, H. (Eds.). (2019). *Computational Thinking Education*. Springer Singapore. https://doi.org/10.1007/978-981-13-6528-7

Lin, G. C., Schoenfeld, I., Thompson, M., Xia, Y., Uz-Bilgin, C., & Leech, K. (2022). "What color are the fish's scales?" Exploring parents' and children's natural interactions with a child-friendly virtual agent during storybook reading. *Interaction Design and Children*, 185–195. https://doi.org/10.1145/3501712.3529734

Make It: Code It projects. (2023). Make It: Code It Projects. https://microbit.org/projects/make-it-code-it/

Mccauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*. https://doi.org/10.1080/08993400802114581

Misirli, A., & Komis, V. (2023). Computational thinking in early childhood education: The impact of programming a tangible robot on developing debugging knowledge. *Early Childhood Research Quarterly*, *65*, 139–158. https://doi.org/10.1016/j.ecresq.2023.05.014

Mukasheva, M., & Omirzakova, A. (2021). Computational thinking assessment at primary school in the context of learning programming. *World Journal on Educational Technology: Current Issues*, *13*(3), 336–353. https://doi.org/10.18844/wjet.v13i3.5918

Perkins, D., & Martin, F. (1985). *Fragile Knowledge and Neglected Strategies in Novice Programmers. IR85-22*. https://eric.ed.gov/?id=ED295618

Resnick, M., & Rosenbaum, E. (2013). *DESIGNING FOR TINKERABILITY*.

Resnick, M., & Silverman, B. (2005). Some reflections on designing construction kits for kids. *Proceedings of the 2005 Conference on Interaction Design and Children*, 117–122. https://doi.org/10.1145/1109540.1109556

Rich, K. M., Strickland, C., Binkowski, T. A., & Franklin, D. (2019). A K-8 Debugging Learning Trajectory Derived from Research Literature. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 745–751. https://doi.org/10.1145/3287324.3287396

Robins, A., Haden, P., & Garner, S. (2006). Problem distributions in a CS1 course. *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, 165–173.

Sadler, J., Shluzas, A. L., & Blikstein, P. (2017). *Building Blocks in Creative Computing: Modularity Increases the Probability of Creative Prototyping Success | Lemann Center*. https://lemanncenter.stanford.edu/paper/building-blocks-creative-computing-modularity-increases-probability-creative-prototyping

Schneider, M. (2022a). *Scaffolding the Debugging Process in Physical Computing with Circuit Check*.

Schneider, M. (2022b). Where's the Bug?: Helping Students Find Errors in Physical Computing. *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, 1084–1084. https://doi.org/10.1145/3478432.3499059

Schneider, M. (2023). Designing scaffolds to support students in debugging e-textiles. *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*, 766–768. https://doi.org/10.1145/3585088.3593925

Shute, V., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*. https://doi.org/10.1016/j.edurev.2017.09.003

Sipitakiat, A., & Nusen, N. (2012). *Robo-Blocks: Designing debugging abilities in a tangible programming system for early primary school children*. https://doi.org/10.1145/2307096.2307108

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798. https://doi.org/10.1016/j.compedu.2019.103798

Thorn, C. (Director). (2013, November 13). *Jim Gee Principles on Gaming*. https://www.youtube.com/watch?v=4aQAgAjTozk

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children*, 199–208. https://doi.org/10.1145/2771839.2771860

Williams-Pierce, C. (Director). (2020, June 8). *Designing Feedback & Failure in Video games (AKA: Why You Should Sometimes Set Players On Fire)*. https://www.youtube.com/watch?v=YY3LpYCXtRw

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Yu, J., & Roque, R. (2018). A survey of computational kits for young children. *Proceedings of the 17th ACM Conference on Interaction Design and Children*, 289–299. https://doi.org/10.1145/3202185.3202738

Yu, J., & Roque, R. (2019). A review of computational toys and kits for young children. *International Journal of Child-Computer Interaction*, *21*(C), 17–36. https://doi.org/10.1016/j.ijcci.2019.04.001