ABSTRACT

LYTLE, NICHOLAS ALAN. Strategies for Designing, Scaffolding, and Leading Open-Ended Programming Projects within Core K-12 Classrooms. (Under the direction of Tiffany Barnes).

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills including learning to program. Considerable efforts have been made within the last two decades including the development and widespread use of novice-friendly block-based programming environments such as Scratch and Snap! as well as new K-12 curricula such as the College Board's Advanced Placement Computer Science Principles Course for high school in the USA. However, these developments have typically been towards improving CS education within elective CS Classes or extracurricular activities, where the majority of K-12 CS is taught. As prior experience with CS is a good predictor of both majoring and taking CS electives, efforts must be made to increase student exposure to CS by developing opportunities outside of these settings. Therefore, to truly broaden participation, computing must be integrated into required K-12 courses.

While there has been much research and work done to infuse computing into mainstream K-12 courses, more is still needed in order to increase widespread adoption of integrated computing assignments in core K-12 courses. First, a framework is needed to build lessons infused with computing in a way that teachers can lead them, since most K-12 teachers lack experience or training for teaching programming. While training of teachers through Professional Development is an important step, efforts must be made to additionally create lesson frameworks that are easily usable, and modifiable by teachers to work within their classrooms. Second, these lesson frameworks also must be able to help novice students learn to use the programming environment, since any computing-infused lesson may be a student's first experience with programming. Conversely, it is possible that many students are already very familiar with programming and as such, these lessons must be able to afford differentiation of both difficulty and task to appeal to these students. This differentiation must be built into the lesson framework so that teachers can easily implement the lesson for their classroom while also meeting the needs of diverse students. Third, infused computing lessons also need to be evaluated on metrics important to the context. While learning gains are important for traditional CS settings, our intent is to increase interest and enjoyment of programming in order to inspire students to take further classes. As such, student evaluations should focus on elements of their experience including

students ’perception of difficulty, their agency over their code, and what they were able to complete. There is also a need to study the impact of how teachers implement these new computing-infused lessons in their classrooms and what impact that has on what students are able to learn.

This dissertation answers the following research questions through the adaptation and study of a pedagogical lesson "Use-Modify-Create" framework for infused lessons:

RQ1: How do we create a pedagogical framework for infused computing activities that will scaffold students into working within block-based environments and reduce perceived difficulty of programming among novice students and teachers?

RQ2: How do we evaluate classroom implementations of these infused lessons in terms of both student experience and teacher instruction?

RQ3: How can a pedagogical framework allow for differentiation of tasks by interest and skill level to meet the needs of students, while being manageable for teachers to implement?

© Copyright 2020 by Nicholas Alan Lytle

All Rights Reserved

Strategies for Designing, Scaffolding, and Leading Open-Ended Programming Projects within Core K-12 Classrooms

by
Nicholas Alan Lytle

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2020

APPROVED BY:

Thomas Price	Eric Wiebe
Chris Martens	Tiffany Barnes Chair of Advisory Committee

DEDICATION

This Dissertation was written by the second Dr. Lytle. It is dedicated to the first Dr. Lytle, who along with my mother, gave me the love, support and guidance necessary to achieve all that is presented in this work. It is also dedicated to the future third Dr. Lytle, the love of my life, who has given me more strength and happiness than I could have possibly ever imagined.

BIOGRAPHY

Nicholas Lytle was born in Groton, Connecticut but grew up outside Washington D.C. in Herndon, Virginia. His passion for Video Games lead him to pursue a Computer Science Degree at the University of Virginia, and his positive experiences teaching and researching there contributed to his pursuit of graduate school at North Carolina State University. Nick got his PhD under the direction of Dr. Barnes in the Game2Learn lab. He was selected as a PostDoctoral Scholar through the CRA/CCC Computing Innovation Fellowship and will spend the next two years under the direction of Dr. Kristy Boyer at the University of Florida.

ACKNOWLEDGEMENTS

First and foremost, this dissertation would not be possible without my advisor, Dr. Tiffany Barnes. The mix of guidance and independence she gave throughout this process allowed me to develop and grow as a researcher, and to that I am in her debt. I'd also like to acknowledge Dr. Eric Wiebe who contributed much to my understanding of both Educational Research in Theory and Educational Research in Practice. Dr. Thomas Price and Dr. Chris Martens also provided tremendous support as committee members and I am excited to begin the translation to working with them as colleagues. I would like to thank everyone within the Game2Learn lab who helped me throughout this process, including, but not limited to: Behrooz, Aurora, David, Dave, Amy, Lauren, Rui, Yihuan, Christa, Mehak, Preya, Ruth, and Rachel. I'd like to individually call out Alex, who was my first friend and kept me on track and within the lab when I was about to leave, and Veronica, who was in many ways my 5th Committee member and also such more more than that. I would like to extend my thanks to everyone involved in the process for conducting this educational research, from members at the Friday Institute (especially Danielle Boulden) to those at the Citadel, to the teachers, faculty, and students at the middle schools in which the studies took place. In addition, to the other co-authors of the work presented in this document that have not been listed so far including Finally, I'd like to thank my family: my mother and father and my two siblings, Drew and Allie, who have always supported me. And Alyssa, who came into my life while this path was started, and I can't imagine spending my future without.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 Motivation

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills (98) including learning to program. Considerable efforts have been made within the last two decades including the development and widespread use of novice-friendly block-based programming environments such as Scratch(83) and Snap!(40) as well as new K-12 curricula such as the College Board's Advanced Placement Computer Science Principles Course for high school in the USA. However, these developments have typically been towards improving CS education within elective CS Classes or extracurricular activities, where the majority of K-12 CS is taught (61). As prior experience with CS is a good predictor of both majoring and taking CS electives (34), efforts must be made to increase student exposure to CS by developing opportunities outside of these settings. Therefore, to truly broaden participation, computing must be integrated into required K-12 courses.

While there has been much research and work done to infuse computing into mainstream K-12 courses (7; 23), more is still needed in order to increase widespread adoption

of integrated computing assignments in core K-12 courses. First, a framework is needed to build lessons infused with computing in a way that teachers can lead them, since most K-12 teachers lack experience or training for teaching programming. While training of teachers through Professional Development is an important step (45), efforts must be made to additionally create lesson frameworks that are easily usable, and modifiable by teachers to work within their classrooms. Second, these lesson frameworks also must be able to help novice students learn to use the programming environment, since any computing-infused lesson may be a student's first experience with programming. Conversely, it is possible that many students are already very familiar with programming and as such, these lessons must be able to afford differentiation of both difficulty and task to appeal to these students. This differentiation must be built into the lesson framework so that teachers can easily implement the lesson for their classroom while also meeting the needs of diverse students. Third, infused computing lessons also need to be evaluated on metrics important to the context. While learning gains are important for traditional CS settings, our intent is to increase interest and enjoyment of programming in order to inspire students to take further classes. As such, student evaluations should focus on elements of their experience including students 'perception of difficulty, their agency over their code, and what they were able to complete. There is also a need to study the impact of how teachers implement these new computing-infused lessons in their classrooms and what impact that has on what students are able to learn.

1.2 Research Questions and Hypotheses

This dissertation answers the following research questions through the adaptation and study of a pedagogical lesson "Use-Modify-Create" framework for infused lessons:

RQ1: How do we create a pedagogical framework for infused computing activities that will scaffold students into working within block-based environments and reduce perceived difficulty of programming among novice students and teachers?

RQ2: How do we evaluate classroom implementations of these infused lessons in terms of both student experience and teacher instruction?

RQ3: How can a pedagogical framework allow for differentiation of tasks by interest and skill level to meet the needs of students, while being manageable for teachers to implement?

I hypothesize that: “A ‘Use-Modify-Create’ Infused Lesson Framework can scaffold novice students and teachers into working within block-based programming environments, allows for differentiation of tasks based on interest and skill level, and can be evaluated at a class implementation level through a triangulated approach using code traces, exit tickets, and observations."

1.3 Dissertation Structure

In chapter 2, I introduce relevant prior literature necessary for understanding my work in context. Chapters 3 contains a study that act as supporting evidence for Research Question 1. Chapters 5 and 6 contain studies that act as supporting evidence for Research Question 2. Chapters 4 and 7 contain two studies that together work to support Research Question 3. Finally, Chapter 8 presents my conclusions of these studies and analyses as well as a theory of action for how the design, scaffolding, and instruction methods of programming projects affect students and teachers in core K-12 classrooms.

CHAPTER

2

LITERATURE REVIEW

This chapter provides a general introductory overview of the primary background literature necessary for placing this work in context. Each Study within Chapter 3 contains an introduction and background section that is useful for placing that individual work in context, but this section below serves as a beginning introduction. Please use the reference document attached in this folder to determine the associated reference.

2.1 Computing in Core K-12 Education

A full review of K-12 Computing Education is outside of the scope of this work, though Garneli provides a brief introduction of recent efforts here(32). Most of the efforts in research made over the past few years have been within elective programming courses and after school programs where the majority of CS education in K-12 occurs(61). A report by Google in 2014 focused on understanding the factors that went into women deciding to pursue computer science found that one of the major factors was academic exposure: prior experience with computing inside a course. This has called for increased efforts to create new opportunities for exposure within courses outside of elective Computer Science

to create interest in the major. A Survey of Educators found that over 70% believed that classrooms should integrate computer science within the core courses, though only 29% of teachers reported having done so within their classroom (34). Therefore targeted effort is needed to begin the process of integrating computing within the core K-12 curriculum.

One of the first elements to decide for integrating computing is what environment or programming language to use. In the age-range of middle grades to high school without prior experience, curricula that use block-based programming languages are often employed(38). Block-based programming languages or Block-based programming environments are designed to make learning to program easier by replacing the interaction of typing in functions, variables, and procedures with a manipulative "drag-and-drop" experience. An example of the block-based environment Cellular (51) is below. Most block-based environments follow a similar set up in which the set of available blocks to use (Called the "Pallette") is on the left most side of the screen. The middle section is where students drag these blocks down and order them in a way to construct a program. The right side of the screen, the "Stage" is where visual output is displayed to the user, usually in the forms of animated sprites that follow the instructions given to them by the code created.

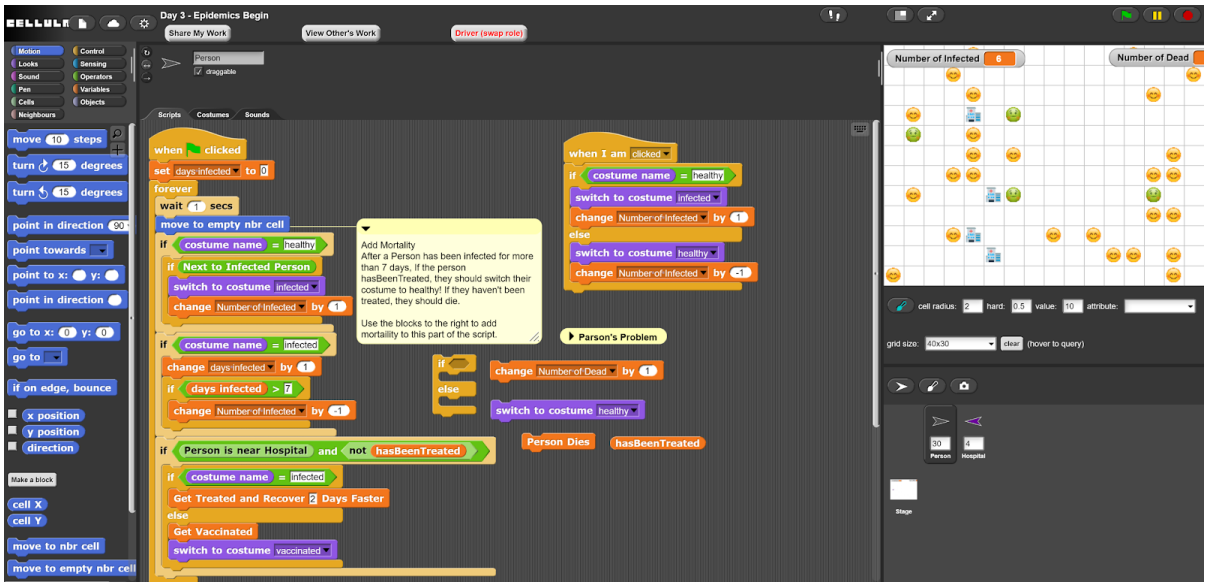


Figure 2.1: The Cellular Environment with Pallette (left) on the set of "Motion" blocks. Code for the Person Sprite is shown (middle). To the right, is the Stage with multiple sprites located in a grid.

Block-based programming languages such as Snap! (40) are increasingly popular and have been shown to scaffold novices into learning programming in STEM contexts better than traditional text-based languages (95). One of the most popular environments, Scratch (83) has millions of users from around the world and is used in research in K-12 computing. For example, Franklin et al. (**?**) have used Scratch for their curricula called Scratch Encore which is designed to help teach introductory programming concepts for K-5. Many environments have been developed or created in order to handle the set of use cases for integrated computing within a specific core K-12 course. NetLogo(97) is an agent-based modelling environment that has curricula developed for use in scientific inquiry such as modeling the diffusion of molecules based on volume or predator prey relations. It extends the popular Logo language, which uses turtle graphics to help teach mathematical concepts through programming (10). Similarly, the Cellular environment (51) was created as an extension to Snap! to be able to model agent-based phenomenon in a grid-world composed of cells. While some environments have been custom made for integration within a core context, others have used their natural affordances for specific programming projects. For example, the Alice programming language was designed for students to learn programming concepts through creating stories with actors (who act out the program code), so its integration into English classrooms has been well-received (84). Similarly, Dong in a review of teacher created material found that a majority of English and social studies teachers used Snaps animation features to create curricula that focused on telling interactive narratives (**?**).

While these integrated programming settings often involve the construction of programs, many researchers and practitioners have focused more broadly on activities that take domain concepts and view them through the lens of computer science, often called "Computational Thinking" (98). While having multiple definitions and components, Computational Thinking (CT) is the thought process of addressing problems, regardless of domain, as those that can be solved computationally, as in creating abstractions, developing algorithms, and analyzing solutions. In this more broad definition, CT activities don't necessarily have to feature programming, but can also include activities that have students use computational tools to do things like data analysis, and the use of these tools has been shown to enable deeper learning of STEM content areas for students (15) as well as increase student retention and learning performance in computing courses and outreach programs (31; 8)]. These CT activities do not even need to involve the use of computers, as the recent wave of "Unplugged" Computing Activities have shown. "Unplugged" Activities, popularized recently by Bell (1) have students engage in computational thinking activities such as learning sorting algorithms by physically engaging in the activity (such as using

a sorting network and having students actually walk through it). This can be a popular choice for teachers who may not have the computing resources necessary for all students to engage in programming or CT activities through devices. As CT is more nebulously defined than programming itself, several framework have been developed to aid novice teachers in addressing the various components necessary for integrating into their classrooms (23). These frameworks have been integrated into Professional Developments (45) which will be discussed at length later.

2.2 Pedagogical and Learning Theories in Computer Science Education

This need for specific curricula to be deployed in the classroom has led to a number of developments and strategies by researchers and teachers alike. Some curricula and activities are deliberately open-ended, following in the footsteps of the Constructionist environments of Logo, Scratch, and Snap (47). Constructionism is the pedagogical approach where students are in learning environments where they decide what learning objectives they wish to pursue and engage in an self-guided, experiential learning process (usually through the construction of artifacts) (68). Constructionism is not to be confused with constructivism, the learning theory in which argues that learning happens through a process of taking new experiences and either accommodating or assimilating the information gained based off of the current mental model to develop a new mental model, though they are often used in combination (24). While constructivism is a theory of learning, constructionism is a pedagogical approach for educational settings, and is quite popular in computer science, as much of learning computer science topics revolves around the construction of programs. Constructionist activities can take the form of fully open-ended creation tasks such as the video games developed for learning mathematical concepts in Minds in Play (46) where students came up with their own ideas and implemented them, or can take the form of more constrained problems. Many popular computing education tools and languages such as Scratch (83) were designed to afford“Wide-Wall, Low Floor and High Ceiling" constructionist learning, motivated by philosophy to give students an environment that enables them to create whatever they like. Using these environments have been shown to increase student engagement in their learning (47). This increased engagement and motivation can be explained further by Deci and Ryan’s Self-Determination Theory, which seeks to understand the development of intrinsic motivation and can be applied to educational set-

tings (19). Self-Determination Theory as applied to education states that students' intrinsic motivation to engage in learning is built through the satiation of three needs: autonomy, competency, and relatedness(19). Autonomy is the ability for a student to engage in tasks of their choosing, and the freedom of choice in constructionist learning environments allows for this feeling to develop. Competency is the need for a student to engage in tasks that develop their skills and allow them to achieve outcomes. Finally, relatedness is the need for a student to engage in learning that allows them to connect with others and build meaningful connections. While constructionist activities can be independent, learners can also work with others or share their creations to express themselves.

While popular and motivated, these fully open-ended create environments may be hard to implement in practice (80). A study of undergraduate project-based learning assignments found that these learning environments are sometimes difficult to facilitate since there are potentially limitless possibilities students may choose for projects and students also find difficulty reasonably scoping a project within their level of possible ability (80). This means that students can in the pursuit of autonomy not necessarily choose learning objectives that will satisfy their need for competency. Lead by educational researchers such as Kirschner and Sweller, there has been a push away from completely open-ended, self-guided, inquiry-based learning such as those promoted in constructionism pedagogies(49). The primary argument, motivated by research done by Sweller with Instructional Supports and Cognitive Load[Sweller1988] is that Direct Instruction, defined as a teacher or instructional material giving information about concepts and procedures that students need in order to accomplish learning tasks (49) is better suited for introductory settings and is more efficient for students learning new concepts than situations without scaffolded guidance (e.g. through instructional supports).

This is further supported by Sweller's cognitive load theory. Cognitive Load Theory suggests that learning, an active process, places three demands on working memory: Intrinsic Load which is the demand associated with the innate difficulty or complexity of the topic, Extraneous Load which is anything that is not useful to creating the new memories, and Germane Load which is the demand associated with the actual processes of turning short term memory into long term memory (92). It is possible that free exploration of a highly complex learning environment would increase extraneous load on the student and generate a heavy working memory load that is actually detrimental to learning, which could be alleviate through direct instruction. A study by Lin and Dalbey(18) shows medium and lower performing students do better with explicit instruction which has no negative impacts on high performing students. Husic argues that beginners lack a repertoire of useful approaches to

thinking about and learning programming. To build this, teachers of introductory classes need to be specific when providing integrated information, problem-solving support, and feedback (42). Direct instruction reduces the working memory load of the students and lets them concentrate on the tasks at hand (55). Thus, for introductory students, early in the learning curve, teachers should provide a supportive scaffolding to help students carry out a task. Accordingly, when teachers provide scaffolding, they generally carry out parts of the overall task that students cannot yet manage. This might be especially needed at the middle school level, as evidence from controlled studies demonstrate that direct, strong instructional guidance proves more efficient in terms of long-term learning than constructivist-based, minimal guidance (49).

2.3 Instructional Supports and Scaffolding Progressions

The need for scaffolded help in Programming has led to the adaptation of Instructional Supports from other domains into the domain of programming as well as the creation of new ones unique to programming. Worked Examples are one type of support in which students are given a problem that has been already solved (in this domain, a program that solves a problem) as an aid to use on further work(92). These have been shown to increase student efficiency while problem solving (102). A modification of Worked Examples is the Erroneous Worked Example, in which a problem is nearly fully completed for a student, save for a few errors in which students must identify and correct (37). As fixing a fully worked out problem maps to the act of debugging, common in programming, many environments have adopted this pedagogical framing of fixing bugs to teach such as GIDGET (54) and BOTS(101). A programming specific instructional support is Parsons Problems, which presents students with segments of code already written out that they must reorder in order to produce correct code (69). Recent research has demonstrated the efficiency of Parsons Problems vs writing code (25). Giving students Hints during problem solving is another means of providing support, though this usually relies on having an expert or teacher present. However, efforts have been made to create Intelligent Tutoring Systems for novice programming such as iSnap (77) which use prior student data to create on-demand hints. This analysis of prior data has also been the basis for data-driven instructional supports such as Data-Driven Worked Examples (102).

While many instructional supports have been demonstrated to be effective at increasing efficiency, little guidance is given in how to build assignments that sequence instructional

supports. One such approach to reduced cognitive load through a scaffolded sequence of interaction comes in the form of Lee et al's Use-Modify-Create (UMC). UMC as a learning progression has been touted as being able to promote the acquisition and development of CT while limiting the anxiety from activities that teachers may have previously perceived to be "too hard" for students (53). In the first phase, "Use", students inspect code and run existing models acting as "consumers of someone else's creation(53). This is similar to the Worked Example discussed above. In transitioning to the "Modify" phase, students go from solely using existing code to changing the code to suit their intended desires as designers. This act of modifying also brings about a change in perceived ownership as students move toward viewing the code as their own. In the final phase, "Create", students build their own intentions into the code and end having created a completely new model, having full ownership of the design and agency over its development[lee2011]. Lee promoted UMC not only as a lesson scaffolding framework, but also a way to create this sense of "ownership" in learners. It has been argued that in order to realize the full benefit of CT, students must develop a sense of ownership over the models underlying the CT concepts being taught (14). During creation, the final step of UMC, students increase engagement in learning and perceived agency of their learning, which is associated with behavioral, emotional, and cognitive engagement (82). UMC allows students to take increasing ownership of the learning by giving them progressively more complex tasks. This increased ownership empowers students to investigate CT and underlying assumptions behind the tasks.

Researchers have used UMC as a basis for a number of CT and CT-infused activities across the K-8 curriculum (52; 96; 36). Werner et al. employed UMC in the creation of an elective game programming course(96), finding that students demonstrated understanding of several CT and CS concepts through developing games. Furthermore, Grizioti et al. developed a game-specific adaptation in which players first play then modify/fix a "half-baked" version of the game, and then create a new version (36). Sentance and Waite extend UMC in their PRIMM (Predict, Run, Investigate, Modify, Make) model for teaching text-based programming(90). Initial workshops suggest that teachers are willing to adopt this model, but like Werner and Grizioti, this work is situated in a pure CS context. Weintrop et al. used UMC as a basis for developing a measurement of CT understanding (94), dividing students into three hierarchical levels of CT understanding (based off of whether they just *used* provided code blocks, *modified* the provided code, or *created* their own). More recently, Franlin et al developed a progression for the "Use to Modify" step entitled TIPP&SEE in which students go through an explicit guided inquiry of a pre-built Scratch project, looking at each section of code described in the acronym (Sprites, Events, Explore) (28).

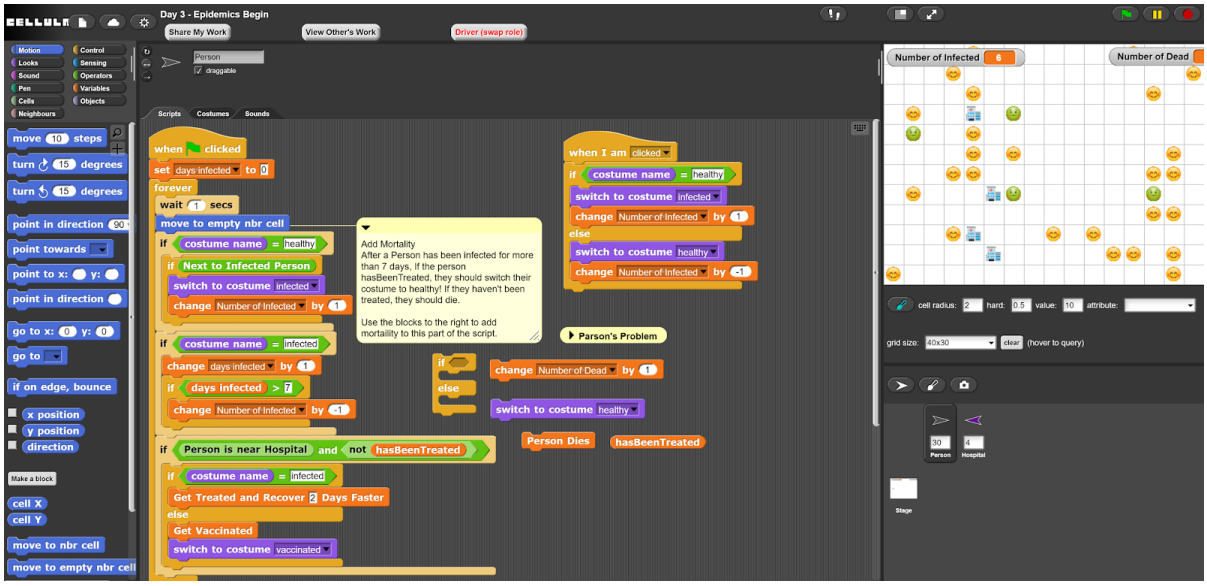


Figure 2.2: Diagram taken from Lee's Computational Thinking for Youth in Practice that explains Use-Modify-Create.

2.4 Integration into K-12, Teacher Focused

Prior work has found that teachers comfort in adopting and instructing material is a barrier for wide-spread use of computational thinking material (7). Besides a purely practitioner-lead route, there have been several strategies adopted by researchers to aid teachers in integrating computing into K-12 core classrooms. One method is through Research practice partnerships (RPPs). RPPs are long-term collaborations between practitioners and researchers that are organized to investigate problems of practice and solutions for improving schools and school districts (12). One type of RPP focuses primarily on design-based implementation research (26). This research aims to study solutions implemented in real world contexts, typically utilizing a cycle of developing and testing instructional activities and curricula. The work in this dissertation proposal is situated in this context, extending work originally started by Catete (7). Working with middle grade teachers around the Raleigh area, Catete identifies numerous barriers to successful CT integration including teacher comfort with the curricula, the need to scaffold both teachers and students into working with block-based environments, and the need for differentiation of assignments for learners of varying skill (7).

While RPPs can provide iterative and intimate contexts for refining and collaboratively creating material, it is limited in scale to the availability of the researchers. One avenue to

combat this problem is the use of Teacher Professional Development. Teacher Professional Development has been used to instruct teachers at scale about curricula and best practices for computing education, such as through the large scale Beauty and Joy and Computing professional development (73) which instructs 100s of teachers each year about how to teach the BJC curriculum for the AP CS Principles Course. More recently, Professional Developments have been developed to focus specifically on instructing a variety of non-CS core teachers on how to integrate CT and programming into their core courses. One such Professional development is the Infusing Computing PD, which has been attended by hundreds of NC and SC teachers over the past three years (45). The Infusing Computing PD uses the Code Connect Create model in which teachers learn the basics of programming in a block-based environment (Code), ways to connect and identify CT in their subject matter (Connect) and finally are aided in creating their own lesson plans to be used within their classrooms (Create). These PDs have resulted in dozens of lessons for a variety of subjects (23), but solely having lessons made and available may not be enough to get computing into classrooms.

Teacher's self-efficacy and discomfort are important factors to consider when helping core subject teachers add computing into their classes. According to Pajares, efficacy beliefs help determine how much effort people will spend on an activity, how long they will persevere when confronting obstacles, and how resilient they will prove in the face of adverse situations - the higher the sense of efficacy, the greater the effort, persistence, and resilience (67). Research has shown that a teacher's self-efficacy is closely linked to their level of adventurousness in teaching. As research by Frykholm suggests, a teachers self-efficacy in combination with tolerance for discomfort will affect their willingness to adopt new materials and curricula (30). In order to better support teachers, they not only need 'more training' but also curricular resources that promote and scaffold their ability to adopt new teaching practices.

Frykholm's theory of discomfort is broken down into four categories. The categories most relevant to introducing a new computing activity into science classrooms are pedagogical discomfort and emotional discomfort. Pedagogical discomfort can occur as a result of managing an active learning environment and loss of authority (as the knowledge keeper) as students gain autonomy (in constructing their own knowledge) (30). Emotional discomfort can occur as a result of the changing role from teacher towards facilitator, the loss of ritual in the classroom, and teachers' vulnerability because they no longer 'know' the content being explored, or have to become active problem solvers along with the students (30). Many of these factors revolve around the openness of the activity and the uncertainty of

the teacher, both of which are present in our partnering teachers, who are unfamiliar with computing. To mitigate discomfort, teachers tend to employ more instructional-based practices such as focused or guided instruction that contrasts the constructionist and open-ended practices loudly promoted in computing (49). This suggests K-12 teachers might benefit from adopting the pedagogical technique of Live Coding. Live Coding, often found in university programming courses, has an instructor start from scratch (or starter code) and build a program related to the material in front of the students(3). Unlike code snippits, displaying the construction of code gives students insight into the programming process (3) and can usually be done in a manner that allows students to follow along creating an active learning context. Studies have demonstrated the benefits of live coding for instructing in block-based programming languages (87). Recent work has focused on intelligent dashboard support for Live Coding (9), though Chen et al. focus on making the process easier rather than analyzing the dynamics between instructor and students. It should be noted that nowhere in definitions for live coding does it specify that students follow along.

Although, infusing CT directly into a STEM course improves the teacher's mastery of their disciplinary concepts with new instructional approaches (93), in courses where students solve open-ended problems, teachers lose much of the control they traditionally have over the learning process and may become uncomfortable (15). Teacher's self-efficacy and discomfort are important factors to consider when helping core subject teachers add computing into their classes. According to Pajares, efficacy beliefs help determine how much effort people will spend on an activity, how long they will persevere when confronting obstacles, and how resilient they will prove in the face of adverse situations - the higher the sense of efficacy, the greater the effort, persistence, and resilience (67). Research has shown that a teacher's self-efficacy is closely linked to their level of adventurousness in teaching. As research by Frykholm suggests, a teachers self-efficacy in combination with tolerance for discomfort will affect their willingness to adopt new materials and curricula (30). In order to better support teachers, they not only need 'more training' but also curricular resources that promote and scaffold their ability to adopt new teaching practices.

2.5 Programming Trace Data and Analytics

There are dozens of systems that analyze and collect student programming data (an introductory review can be found here: (43)). In programming data-mining, The sequence of all

interactions within the environment is called a *trace*, which comprises a list of all *states* a student environment is in and the interactions that connect them. An important subset of this trace is all interactions that result in differing code (e.g. adding, deleting, and relabeling code), called a *code-trace* and all the different *code-states* (i.e. unique code) that a user progresses through on the way to complete a problem. This code trace is used for intelligent programming environments as input for the creation of data-driven next-step hints (77). While useful for data-driven algorithms, the size of the space of all possible code-states is in practice incredibly large even for a small programming problem (Zhi et al.). As such, methods have been developed to collapse the state-space of programming problems to something more manageable for analysis. Prior work has represented the varying pathways of student attempts using a *feature-state* representation (Zhi et al.). In this manner, an assignment can be seen as a set of *features*, and a student's current *state* in the assignment can be represented by which features the student has present or absent in their code. Rui et al (Zhi et al.) demonstrated techniques to visualize student pathways through a programming assignment by representing which features students progressively added to to their code until reaching a final solution.

Additional effort has been placed recently in using student feature completion information to help the *instructor* during classroom implementation (20). Dashboards have been developed that use programming trace or compilation information to aid instructors in finding students in need of help(20). Diana et al's dashboard system is able to use features of code-trace data to display to an instructor which students are lagging behind in the lesson (having fewer features complete) in order to aid teachers in selecting which students to help or group together to peer-help (20). While useful, this requires that the dashboard be visible to the instructor at all times which may take away from the time spent live coding. We wish to extend these methodologies in order to develop ways to provide post-hoc analysis of a classroom implementation as well as in the future, provide unintrusive immediate support to instructors live coding in K-12 environments.

geometry pdflscape fancyhdr rotating

CHAPTER

3

USE, MODIFY, CREATE: COMPARING COMPUTATIONAL THINKING LESSON PROGRESSIONS FOR STEM CLASSES

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., ... & Barnes, T. (2019, July). Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (pp. 395-401).

3.1 Introduction

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills (98). However, these skills have typically been taught within elective Computer Science classes or outside of school activities (61). To reach *all* students, CT must be integrated into required K-12 courses, such as science and math. This CT integration will pose several challenges. First, lessons must not only focus on key

CT concepts but must also include and integrate domain knowledge, though research demonstrates that CT topics can be integrated without detracting from the learning of the core domain material (5). Further, these activities must be designed with the understanding that they may be the first introduction to programming or CT for many students and teachers. Successful integration depends on equipped and capable teachers, though many do not have the prerequisite background required to teach CT or computer science (17). Professional development can give teachers experience with CT skills (**?** 74**?**), however, to reach all students, we must develop solutions that can be readily adopted by both experienced and inexperienced teachers and that can improve student learning in both CT and course content.

This study compares two separate design implementations of a 4-day computing infused science lesson across multiple classrooms. One condition received a lesson including 3 days of coding while the other received a scaffolded curriculum modeled after Lee's Use-Modify-Create (UMC) (53). Through our quasi-experimental design, we aim to investigate how UMC sequencing impacts:
1. Student-perceived *difficulty* of the lesson.
2. Student-perceived *ownership* of the used and developed programs.
3. Teacher-perceived *difficulty* of the lesson and *ability* to teach it.

3.2 Related Work

As demand increases for incorporating computing into core K-12 subjects, so does the need for classroom activities that are tailored for non-computing focused teachers and students. Through adding computing activities directly into a STEM course, teachers gain improved mastery of their discipline when using new instructional approaches (93). However, reporting suggests that teachers lose much of the control they traditionally have over the learning process and may become uncomfortable when students pose and solve open-ended integrated STEM and Computing problems (16). For teachers to support these students engaging in self-directed collaborative processes, they require an ability to diagnose difficulties and give hints, rather than supply solutions.

A prior case study by Cateté et al. (7) on infusing CT into science classrooms highlights how teachers were hesitant to lead new programming activities, and were afraid of misguiding students, even with professional development and classroom support. These concerns are also identified in a 2017 report for UK teachers facing computing infusion (89). The most common challenges mentioned included subject knowledge, differentiation, lack

of time, approaches to teaching topics, students' understanding, and ability to problem solve. This report also lists successful teacher strategies towards teaching computing such as unplugged activities(1), computational thinking, contextual learning, and scaffolding of programming tasks. In order to improve the incorporation of computing into science classrooms, we attempt to utilize the above findings to create more supportive materials for both teachers and students facing computing.

One such approach to improving CT acquisition with reduced cognitive stress comes in the form of curricular materials that follow a scaffolded intensity of interaction. Research by Lee et al, suggests that using a Use-Modify-Create (UMC) learning progression can promote the acquisition and development of CT while also limiting the anxiety from activities that teachers may have previously perceived to be "too hard" for students (53; 52). In the first phase, **Use**, students inspect code and run existing models acting as "consumers of someone else's creation(53)." In transitioning to the **Modify** phase, students go from solely using existing code to changing the code to suit their intended desires as designers. This act of modifying also brings about a change in perceived ownership as students move toward viewing the code as their own. In the final phase, **Create**, students end up in a state where they have created a completely new model, having full ownership of the design and agency over its development(53).

Lee promoted UMC not only as a lesson scaffolding framework, but also a way to create this sense of "ownership" in learners. It has been argued that in order to realize the full benefit of CT, students must develop a sense of ownership over the models underlying the CT concepts being taught (14). During creation, the final step of UMC, students increase engagement in learning and perceived agency of their learning, which is associated with behavioral, emotional, and cognitive engagement (82). UMC allows students to take increasing ownership of the learning by giving them progressively more complex tasks. This increased ownership empowers students to investigate CT and underlying assumptions behind the tasks.

Researchers have used UMC as a basis for a number of CT and CT-infused activities across the K-8 curriculum (52; 96; 36). Werner et al. employed UMC in the creation of an elective game programming course (96), finding that students demonstrated understanding of several CT and CS concepts through developing games. Furthermore, Grizioti et al. developed a game-specific adaptation in which players first play then modify/fix a "half-baked" version of the game, and then create a new version (36). Sentance and Waite extend UMC in their PRIMM (Predict, Run, Investigate, Modify, Make) model for teaching text-based programming(90). Initial workshops suggest that teachers are willing to adopt this

model, but like Werner and Grizioti, this work is situated in a pure CS context.

We believe the UMC framework can extend into core domains, alleviating the burden of learning to program while simultaneously learning domain material. This will allow students to ease into the activity, become familiar with the programming environment, and explore how smaller changes affect the code. We assume these same benefits extend to teachers who also might not be familiar with coding and would welcome scaffolded lessons. We finally posit that as students go from users to creators in the Use-Modify-Create lessons, their sense of ownership of the project will increase to match that of students who participate in lessons where they always create code from scratch. We test out these assumptions using an A/B study across multiple classrooms as described in the section below. If these hypotheses are supported, these benefits can reduce teachers' fears of being CT novices as well as students' frustration with the difficulty of learning to program, potentially increasing adoption of materials developed using the UMC progression.

3.3 Methods

3.3.1 Context

The study took place in two separate middle schools (School 1, School 2) in the mid-atlantic United States. The classrooms were all 6th grade (age 11-12) science classrooms taught by one of 4 teachers (1 from School 1, 3 from School 2). None of the teachers had experience instructing programing lessons. Each teacher was responsible for multiple class *periods* with different students. Teachers in School 2 had 5 class periods each while the teacher in School 1 taught 2 class periods. Each period averaged 20 - 30 unique students. A total of 394 students participated in the study, but we only analyze data from the 160 consenting students who provided data for every day. Demographic information for these students is reported in Table 3.1. Chi-Square tests show no significant differences in gender or race between the two populations.

We further surveyed students on their previous programming experiences. Responses are shown in Table 3.2 ranging from Never to Daily. Chi-Square tests show no significant differences in prior programming experience between the two populations. In an open response follow up to question 1, many students reporting "Rare" or "Occasional" described participating in an Hour of Code activity (48). Those marking "Frequent" or "Daily" report being in a computing club or technology elective course.

Table 3.1: Gender and race/ethnicity by condition.

		UMC (N=95)	Control (N=65)	Total (N=160)
Gender	Female	42.1%	41.5%	41.9%
	Male	47.4%	55.4%	50.6%
Race Ethnicity	Black	19.0%	13.9%	16.9%
	Caucasian	26.3%	26.2%	26.3%
	Hispanic	15.8%	18.5%	16.9%
	Asian	14.7%	15.4%	15.0%
	Multi-racial	3.2%	3.1%	3.1%
	N/A	21.1%	23.1%	21.9%

Table 3.2: Participants' computing background self-ratings.

	Never	Rare	Occasional	Frequent	Daily
Q1	Previous participation in computing activities				
UMC	7.4%	9.5%	54.7%	11.6%	6.3%
Control	6.2%	15.4%	47.7%	20.0%	7.7%
Q2	Previous experience writing a computer program				
UMC	15.8%	21.1%	41.1%	7.4%	4.2%
Control	13.8%	23.1%	40.0%	15.4%	4.6%

3.3.2 Curriculum

The activity was designed to be a 4-day, CT lesson. Though teachers were trained on the material, on programming days, a research team member taught the first period as part of a "Faded Instructor Scaffolding Model" designed to help teachers understand the curriculum from a student's perspective. Each programming day took place within the Cellular environment (51), an extension of the block-based programming language Snap! (31) that provides a good method for agent-based modeling and has been used in similar initiatives with infusing CT into STEM curriculum (7).

For use in the 6th grade science classroom, the topic of "Food Webs" was chosen. In the food web curriculum, students learn about how energy is transferred from producers to primary and secondary consumers. The computing-infused activity let students explore the transfer of energy in a simplified food web developed using the block-based programming environment, Cellular (51). We describe each daily segment of the activity below, and a breakdown by condition is visualized in Figure 3.1. The Use-Modify-Create (UMC) version was adapted from a previous version of the food web activity used in classrooms which acts

as our control lesson in this study.

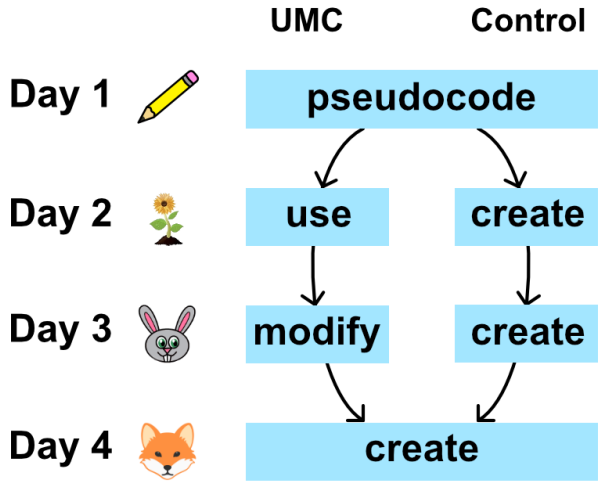


Figure 3.1: Programming methods for Food Web agents (plants, bunnies, foxes) by day and condition

Day 1 - Both conditions completed an “Unplugged" activity (1) in which students reviewed definitions and components of a Food Web (e.g. primary and secondary consumer, how energy is transferred through the system, etc.). This ended with completing a worksheet lead by the instructor where students described the behavior of agents in the model through pseudo-code. This was done to prepare students for developing these ideas within the programming model.

Day 2 - Day 2’s focus was the “Plant" agent (the producer), which would grow based on the solar energy given by the “Sun" (code provided for both conditions). For the control condition, students had to develop code for the Plant to be able to transition between stages of its life cycle using the amount of ‘Solar Energy’ received over time. This was represented in code as sequential conditionals, checking both the plant’s current state and energy before transitioning into the next state. The students in the UMC condition had plant code provided and instead inspected and read through the working code in order to become familiar with the different conditions. The instructor led students in exploration by changing the initial input (the solar energy intensity), the cutoff conditions (how much energy is needed to transition) and the amount of energy lost through transitioning. Students then used a worksheet to record how those changes affected the speed in which flowers changed state.

Day 3 - Day 3's focus was the "Bunny" agent (the model's primary consumer). Control condition students wrote code to add the new agent to their working model. Meanwhile, the UMC condition had Bunny code provided at the outset. However, the given bunny behavior did not conform with their idea of the actual model (e.g. bunnies never ate when they got low on energy, flowers transitioned to an incorrect state after being eaten etc.). Thus, students during this class period *modified* the existing code in order to make it conform to the existing ideas they had discussed on Day 1's activity of how the model should behave. This is similar to the activity found in prior studies of fixing the "half-baked microworld"(36).

Day 4 - The agent focus for the final lesson was on this model's secondary consumer, the "Fox". Both conditions had to develop the entire Fox code (in a sense, *creating* a new model with this additional agent) and update the "Bunny" code to react to the new agent and implement the desired final behavior in their model. Once complete, as shown in Figure 3.2, students were tasked with changing some functionality in their code and comparing how their new simulations behaved differently from their previous one.

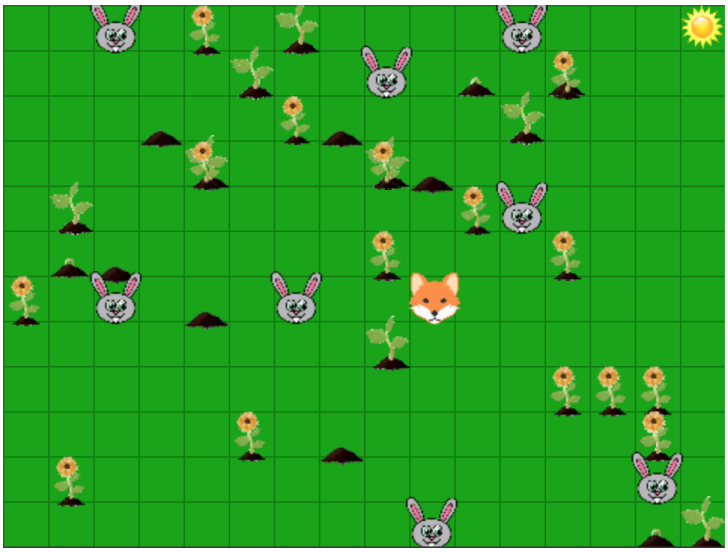


Figure 3.2: Food Web in Cellular showing how plant, bunny, and fox agents appear on the grid at one time step.

Although each programming day builds on previous topics, students begin with the same starter code for their condition to reduce effects from student absences or incompletion from a prior day. The topic of Food Webs and the sequencing of our curriculum affords

the exploration of our research questions for a number of reasons. First, as a life science topic found in common core guidelines, Food Web is an exemplar STEM lesson that could be taught throughout the United States. Second, as the topic of Food Web focuses on the interaction between different actors in an environment, it lends itself nicely to a computing task focused on developing an agent-based model or simulation like prior UMC-developed lessons (53). Finally, as we use a multi-day assignment, we can segment each programming day to be on a single Food Web actor, allowing us to frame the UMC conditions' activity to match each phase of UMC.

3.3.3 Data Collection

Evaluation of the initiative was recorded through a number of data collection methods. For every period, at least one research team member was present taking observation notes, focusing on the students' interactions within the environment as well as how the teacher was teaching the lesson. After the conclusion of each day's activity, students took an end-of-activity "Exit Ticket" in which they answered a series of questions about the activity. In order to study student-perceived difficulty, we asked students to rate the difficulty of the days' activity on a 1-5 Likert scale (Very Easy to Very Difficult). For student-perceived ownership, two questions were included that addressed the ability to express one's ideas and the belief the code was their own creation. These questions were on a 7-point Likert scale from Strongly Agree to Strongly Disagree.

For teacher-perceived difficulty, we first analyzed classroom observations that focused on the teacher's ability to teach the lesson. Additionally, a member of the research team conducted interviews using a semi-structured interview protocol with each of the participating teachers at the conclusion of the lesson sequence. The protocol consisted of questions targeted to elicit general teacher feedback about each of the days of the lesson sequence and their impact on students (e.g., What are the strengths of the lesson? Weaknesses?) Interviews lasted approximately twenty to thirty minutes and were audio recorded and transcribed for analysis. A constant comparison analysis (65) of the interview transcripts provided insights on the teachers' perceptions of the lessons from a pedagogical standpoint, comparing teacher interviews within groups and between groups.

3.4 Results

3.4.1 Student-perceived difficulty

Each day, students completed the exit ticket question “Please use the [scale of 1 to 5 (Very Easy to Very Difficult)] to rate how difficult or easy the lesson was today." The daily average values for students in each condition are given in Table 3.3. A Friedman Test, similar to a parametric repeated measures ANOVA but for non-parametric data (29), was performed in order to determine if there were differences found in the average values for each of the days. For the Use-Modify-Create (UMC) condition, no significant difference was found among the four days $\chi^2_3 = 1.879, p = 0.598$. However, for the control condition, $\chi^2_3 = 9.984, p = 0.019$ showing significance. Therefore, a post-hoc Wilcoxon Signed Rank Test, a non-parametric equivalent to a paired T-Test (99), was performed between each of the pairwise groups. No significant differences were found between days 1 and 3, 1 and 4, nor 3 and 4. However, for each pairwise comparison with day 2, a significant difference was found (1-2: $V = 240, p = .002$; 2-3: $V = 770, p = .003$; and 2-4: $V = 218, p = .004$). An additional Mann Whitney U Test, a non-parametric test similar to an unpaired T-Test(63), was performed between groups for each of the 4 days to determine differences in difficulty responses between conditions for the same day. No significant differences were found in the comparisons between conditions for Days 1 ($W = 3187, p = 0.72$), 3 ($W = 3162, p = 0.79$), or 4 ($W = 3144, p = 0.84$). However, comparing Day 2 (the first coding day) between the two conditions found a significant difference with UMC being significantly easier ($W = 2238, p = .002$).

This difference in difficulty is backed by classroom observations. Researchers found that in the UMC classrooms, students were often able to finish their designated tasks more quickly, especially on the third day where the UMC group modified a bunny while the control group coded one from scratch. This time difference means that the UMC group had additional time to add elements or engage in teacher-led discussions about connections to class topics. Further, researchers found in some control condition classrooms (especially on Day 2 and to some degree Day 3) that students had difficulty finishing the task. As a result, teachers in the control group on Day 2 either forged ahead leaving many students behind, or slowed the lesson so all students could keep up, but were unable to complete the full lesson to add flowers. During the follow-up interviews, teachers in the control condition commented on the need for more scaffolding, while teachers in the UMC condition did not express this concern. Teachers in the UMC condition indicated that the progression of the curriculum served as an effective scaffold for students’ conceptual understanding of

the programming environment that better prepares them for creating their own program models. One of the teachers articulates this below:

"...like day [two], when we were on the computer. You really understand the beginning part. And then day [three] it builds a little bit more and you're building the code. You're playing with it. And day [four] is really copying the bunny code, just tweaking it a tiny bit. So at that point they've done so much with it already. They've got it. I mean, they were playing with all kinds of things."

Table 3.3: Reporting of Average and Standard Deviation for student responses to Exit Ticket questions.

	Use-Modify-Create Condition (N=95)				Control Condition (N=65)			
Likert Questions	Day 1	Day 2	Day 3	Day 4	Day 1	Day 2	Day 3	Day 4
Rate how difficult or easy the lesson was today: (Very Easy) 1 - 5 (Very Difficult).	$\overline{\chi} = 2.11$ $\sigma = 0.93$	$\overline{\chi} = 2.04$ $\sigma = 1.18$	$\overline{\chi} = 2.15$ $\sigma = 1.03$	$\overline{\chi} = 2.25$ $\sigma = 1.09$	$\overline{\chi} = 2.03$ $\sigma = 0.85$	$\overline{\chi} = 2.58$ $\sigma = 1.13$	$\overline{\chi} = 2.11$ $\sigma = 1.03$	$\overline{\chi} = 2.25$ $\sigma = 1.09$
“I was able to express my ideas in the model today." (Strongly Agree) 1 - 7 (Strongly Disagree)	N/A	$\overline{\chi} = 2.89$ $\sigma = 1.45$	$\overline{\chi} = 2.64$ $\sigma = 1.48$	$\overline{\chi} = 2.68$ $\sigma = 1.54$	N/A	$\overline{\chi} = 3.13$ $\sigma = 1.18$	$\overline{\chi} = 2.98$ $\sigma = 1.32$	$\overline{\chi} = 2.99$ $\sigma = 1.32$
“The code I ended the lesson with is my own creation." (Strongly Agree) 1 - 7 (Strongly Disagree)	N/A	$\overline{\chi} = 3.54$ $\sigma = 1.87$	$\overline{\chi} = 2.79$ $\sigma = 1.64$	$\overline{\chi} = 2.93$ $\sigma = 1.72$	N/A	$\overline{\chi} = 3.39$ $\sigma = 1.42$	$\overline{\chi} = 3.24$ $\sigma = 1.44$	$\overline{\chi} = 3.21$ $\sigma = 1.43$

3.4.2 Student-perceived ownership

Two questions were added to the coding day exit tickets (Days 2, 3, and 4) to address student-perceived ownership. These were: “To what extent do you agree with the following statement: I was able to express my ideas in the model today" and “To what extent do you agree with the following statement: The code I ended the lesson with is my own creation" both on a scale from 1 (Strongly Agree) to 7 (Strongly Disagree). The average values for these answers are shown in Table 3.3. A Friedman Test was performed in order to determine if there were differences found in the average values for each of the days. No significant difference was found among the three days for ‘expressing ideas’ for both the control group, $\chi_2^2 = 0.0231, p = 0.989$ and for the UMC group $\chi_2^2 = 1.1421, p = 0.565$. However, in performing Mann Whitney U Tests between the 2 groups, significant differences were found in Days 3 ($W = 3389, p = 0.03$) and 4 ($W = 3425, p = 0.04$) though not for Day 2 ($W = 3683, p = 0.21$). For the statement “The code I ended the lesson with is my own creation", a Friedman Test finds no significant difference in the control condition answers: $\chi_2^2 = 0.562, p = 755$. However, for the UMC condition, the Friedman Test found a significant difference among the three: $\chi_2^2 = 9.637, p = 0.008$. A follow-up pairwise Wilcoxon-Signed Rank Test was performed between each of the days and while no significant difference was found between Days 3 and 4 ($V = 769, p = 0.38$), significant differences were found between Days 2 and 3 ($V = 1973, p < 0.001$) and Days 2 and 4 ($V = 1000, p = 0.006$) within the UMC group. Additional Mann-Whitney U Tests were performed between the two conditions. While no difference was found between Day 2 ($W = 4207, p = 0.8$) or Day 4 ($W = 3485, p = 0.07$) between the conditions, significant differences were found between the Day 3 ($W = 3284, p = 0.02$) responses between groups.

While not as direct as the student-perceived difficulty differences, there were key classroom observational differences between the conditions that corroborate the findings from the exit tickets. First, as stated before, students in the UMC condition were often able to finish tasks faster and were therefore able to explore more within the code, and add their own additional features. It is possible that adding their own touches after the guided part of the lesson led to an increased sense of artifact ‘ownership’. Second, researchers observed (and teachers commented during follow-up interviews) that students in the UMC condition seemed more engaged in the activity, but it was actually difficult to keep students engaged in the control condition. This disconnect with the material and the monotony of the tasks in the control condition might have contributed to the students’ sense that the code was not their own.

3.4.3 Teacher-perceptions

Teacher interview transcripts were analyzed using a constant comparative method (65) that entailed searching for themes amongst the teachers within each condition and then comparing data from the teachers across the two conditions. Results reflected the difficulties that teachers in the control group faced. The two teachers implementing the coding-intensive control version of the curriculum expressed concerns that their students needed more scaffolding to complete the lessons and concerns about their students' daily engagement. When asked about potential improvements to the curriculum, both teachers suggested giving students more time to "explore" and "play" with the code prior to creating their own programs. UMC provides this opportunity for students. As one of the teachers in UMC condition explains, "the kids [during the 'use' day] were understanding the coding; they were understanding why it was changing and they were starting to play around with some of that as well." Additionally, results from teacher interviews demonstrated that teachers in the control condition perceived a decrease in student engagement each day. Teachers themselves suggested changing the approach each day, e.g. "I got a lot of comments that they were bored with it because it was the same thing day after day. So I can't really think of it right now, but if there is, like some kind of way to mix it up and still have the same information, but maybe have them do it in a different way." Our data and teacher interviews suggest that approaching coding through a variety of tasks, as the UMC approach does, can improve student engagement. It is also possible that the structured UMC sequence makes the Create day more purposeful and engaging for students. One teacher explains her students' reactions to the Create day, "They're like 'day three was very fun'. They really got a chance to understand how the whole thing is connected."

The teacher interviews also corroborated that a UMC sequence offers benefits to teachers, as it supports their learning and confidence with the materials. One of the teachers revealed to us that, researcher support "wasn't needed the last day because I knew it. At this point I was like, 'I'm comfortable. I know where you're going with this.'" Both of the teachers in the code creation sequence commented that the lessons were "exhausting," as one teacher described that implementing the lessons entailed "standing in front of a room and talking and basically having you being the first person [they're] gonna ask questions to for five hours."

The findings above are supported by observations of teachers working through the curriculum. Like the teacher above noted, researchers observed that teachers gained confidence in teaching the lessons in both conditions, but this was more marked for UMC

teachers. The two UMC teachers, in addition to following the guided material provided to them, were able to add new tasks to class periods where time was still available, and led students in guided discussions about the connections between the CT concepts and the scientific concepts modeled within the environment. As previously discussed, teachers in the control condition had difficulty engaging and keeping all students on task as observed by the researchers, and often needed to pause (especially on Days 2 and 3) to check which task they needed to be doing or what the code was supposed to look like. While this was also observed with one teacher of the UMC condition, this behavior was occurring later in the assignment sequence (on Days 3 and 4) and not to the same frequency.

3.5 Discussion

For student-perceived difficulty, **students perceived the introduction to coding on Day 2 as significantly harder in the control group than any other day**. While there was the same procedure for each of the coding days in this condition (i.e. students had to create an agent on each day) having to do this for the first time might have been difficult without prior knowledge or experience in the environment. Days 1, 3, and 4 having similar reported levels of perceived difficulty, suggests that while the Day 2 activity was harder, getting through it and understanding the procedure prepared students for the Day 3 and 4 coding activities. No difficulty spike, however, was found in the sequence for the UMC condition. The benefits of this sequence can be explained using James Paul Gee's principles of good learning (33). It could be that first "using" the new Cellular environment was "pleasantly frustrating" – challenging but perceived to be easily done, and then modifying it and then creating a new agent results in "well-ordered problems" that allow students to develop mastery (33). In contrast, Day 2 for the control condition combined the need to learn the new environment while also learning the basics of programming and how they work within the environment, potentially increasing the cognitive load of the students (92), which could result in a higher perceived difficulty on Day 2.

Two questions were assessed to measure student perceived *ownership* of the models each day. While there are differences between conditions for the question regarding expression of ideas, the most pronounced differences between days and conditions is found for the question on whether the code was their "own creation". While the difference was not significant, the average value for the response was higher in Day 2 of the UMC condition, indicating that UMC students did not feel as much ownership over the final code as stu-

dents in the control condition. This was expected, as Day 2 represents the “Use" day where students changed parameters and input variables, but did not create any new code. It is only on Day 3, where students in UMC were “Modifying" existing Bunny code, that there is a difference in perceived ownership between groups. We were somewhat surprised that students in the UMC condition felt significantly more ownership of their code, since they made fewer programmatic changes (code adds, deletes, edits) than the control group. It could be that the framing of the activity as modifying existing code to make it perform the ‘correct’ behaviors played into this mindset. In addition, the large number of program edits needed in the control condition may have made more creative activities, such as augmenting the model behavior, blend in with more mechanical changes, like dragging in pre-specified blocks. In some cases, these creative activities may not have even occurred, since teachers struggled to help students complete the Day 3 and Day 4 activities in the control group. The strengthening of artifact ownership in the UMC condition that began on Day 3 carried into Day 4, with the **UMC group agreeing significantly more than the control group that their final code was their own, despite both conditions doing the exact same task**.

In addition to student reported data, reflections from teachers also suggest that they would benefit from and prefer the UMC Condition. Teacher expertise is in supporting student learning, and their perceptions confirm that a strict code creation approach is not as effective for their classrooms. Since many teachers are novices to programming and CT, and their courses are focused on other topics, it is not realistic to expect disciplinary K-12 teachers to be able to support such an intensive coding approach to integrating CT. The UMC model helps teachers gradually learn how programs represent their disciplinary knowledge, enabling them to make those connections just in time with students. As stated by teachers, having time to be able to “explore" the environment by first reading and understanding code, then performing minor edits, and finally being ready to add independent features, gives both students and teachers an easier progression of tasks. This means that teachers can adopt integrated CT curricula more readily, letting them learn CT and programming along with their students.

3.6 Conclusion

In this paper, we present results from an A/B study of Use-Modify-Create (UMC) versus a control group implementation of a CT-infused Food Web activity. With student reports corroborated by classroom observations and teacher interviews, we were able to confirm

previous research results showing that UMC sequencing provides students a natural progression to learn computational thinking within a science course, while giving students more ownership over the artifacts they create. We also found that teachers using our UMC curriculum felt it was easy to teach, and that it promoted student engagement and exploration, while teachers using a code-intensive control curriculum desired more scaffolding and features to improve student engagement.

Limitations of this work include potential population bias, instructor effects, and our interpretation of the UMC model. Participants are from two middle schools where many students have prior exposure to learning computing. This population bias could have improved students' ability to go through the curriculum and the ease in which they learned and experienced the topics. However, as the daily difficulty ratings are discussed in relative terms, we assume that a middle school with less access to CT and computing education would show even greater differences in perceived difficulty between the UMC and control conditions. It is not clear how more or less programming experience would impact student ownership ratings. Four teachers participated in the study, with two leading instruction in each condition. As such, there is no way for us to separate instructor effects from the curricular content/sequencing. Though Kruskal-Wallis Tests (62) and Mann-Whitney U Tests find no significant difference in student perceived difficulty by teacher group, it is still possible that instructors played a role in the student perception of difficulty and ownership. Finally, while Lee's original paper on the UMC model defined "Creation" as students making their own designs (53), we interpret it specifically to mean that students should develop all of their own code for an agent. In future studies, we hope to design activities that facilitate more open-ended student exploration and creativity.

CHAPTER

4

FROM ‘USE’ TO ‘CHOOSE’: SCAFFOLDING CT CURRICULA AND EXPLORING STUDENT CHOICES WHILE PROGRAMMING

Lytle, N., Catete, V., Isvik, A., Boulden, D., Dong, Y., Wiebe, E., & Barnes, T. (2019, October). From'Use'to'Choose' Scaffolding CT Curricula and Exploring Student Choices while Programming (Practical Report). In Proceedings of the 14th Workshop in Primary and Secondary Computing Education (pp. 1-6).

4.1 Introduction

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills (98). However, these skills have typically been taught within elective Computer Science classes or outside of school activities (61). To reach *all*

students, CT must be integrated into required K-12 courses. This type of CT integration poses several challenges. First, lessons must not only focus on key CT concepts but must also integrate domain knowledge. Further, these activities must be designed with the understanding that they may be the first introduction to programming for many students. Successful integration depends on equipped and capable teachers, though many do not feel they have the prerequisite background required to teach computing (17). Professional development gives teachers much needed experience with CT skills (74), however, to reach all students, we must develop solutions that can be readily adopted by both experienced and inexperienced teachers while keeping students actively engaged.

This report investigates the iterative design of a 4-day computing-infused curricula developed for a grade 6 middle-school science classroom. Through multiple iterations of content refinement, we have identified the need for integrating student-choice through differentiated options in our curricula. This allows students with varying levels of prior programming experience to complete the assignment to a level of their own choosing, focusing on the aspects that interest them, thus fostering their interest in STEM and computing. This design also balances the teachers need to have a sense of control/comfort over the classroom in order to continue the adoption of CT and computing into their core content classroom.

This paper reviews related initiatives and our prior work and findings on developing computing infused curriculum for science classrooms. The paper then describes our latest curriculum feature on adding student choice, investigating both student programming behavior and teacher affect. Our data is triangulated using a combination of code traces, exit tickets, and classroom observations (59). We further support our findings with post-hoc teacher interviews.

4.2 Related Work

CT and the use of computational tools has been shown to enable deeper learning of STEM content areas for students (16) as well as increase student retention and learning performance in computing courses and outreach programs (8; 31). Middle grades has been identified as a critical age range to study the potential for developing CT. In this age-range, block-based programming curricula (38), and visual programming environments have been shown to improve student performance and affect(73).

Many introductory activities are motivated at the pedagogical and psychological level

by constructionism (68), and self-determination theory (19). Many popular computing education tools today like Scratch (83) use “Wide-Wall" learning environments motivated by constructionist philosophy to give students an environment that enables them to create anything. These environments have been shown to increase student engagement in their learning (47), which can be further explained by Self-Determination Theory, specifically the three needs of autonomy, competency, and relatedness required for developing intrinsic motivation in students (19). Freedom of choice in constructionist learning environments allows for greater development of autonomy and competency, and sharing of personal artifacts in the social setting of the classroom can help develop relatedness. While popular and motivated, these fully open-ended create environments may be hard to implement in practice (80).

Although, infusing CT directly into a STEM course improves the teacher’s mastery of their disciplinary concepts with new instructional approaches (93), in courses where students solve open-ended problems, teachers lose much of the control they traditionally have over the learning process and may become uncomfortable (16). Teacher’s self-efficacy and discomfort are important factors to consider when helping core subject teachers add computing into their classes. According to Pajares, efficacy beliefs help determine how much effort people will spend on an activity, how long they will persevere when confronting obstacles, and how resilient they will prove in the face of adverse situations - the higher the sense of efficacy, the greater the effort, persistence, and resilience (67). Research has shown that a teacher’s self-efficacy is closely linked to their level of adventurousness in teaching. As research by Frykholm suggests, a teachers self-efficacy in combination with tolerance for discomfort will affect their willingness to adopt new materials and curricula (30). In order to better support teachers, they not only need ‘more training’ but also curricular resources that promote and scaffold their ability to adopt new teaching practices.

One promising avenue to improved support resources for teachers is curricular materials that follow a scaffolded intensity of interaction. Research by Lee et al. suggests that using a Use-Modify-Create (UMC) learning progression can promote the acquisition and development of CT while also limiting the anxiety from activities that teachers may have previously perceived to be “too hard" for students (53). Sentance et al. takes this a step further by specifying learning practices to engage in at the Use level. Their new model, Predict, Run, Investigate, Modify, Make (PRIMM) “incorporates activities that scaffold learning for students and provides a structure for lessons" (91). By dedicating a portion of the lesson on reading and planning before modifying code, students can develop a sense of what the code is doing. As Lister shows, a student’s ability to accurately follow code is highly correlated

with student confidence to independently write code (57). The next step, modify is used by both Sentance and Lee and is a scaffolded way to encourage small code changes, or focus on recognizing misconceptions between code logic and world logic (36). The Create step of these two frameworks is still open-ended and can leave stress for teachers. Teachers often find it difficult to facilitate the potentially limitless possibilities students may choose and students also find difficulty reasonably scoping a project within their level of possible ability (80). In order to alleviate both students lack of direction and teachers inability to support all extensions, our prior findings suggest implementing a more limiting Choose phase to the curricula. The sections below describe our intervention.

4.3 Context

4.3.1 Background

In our prior work developing infused computing curricula for STEM courses, we found that the largest differentiator in both teacher and student engagement in our integrated science-CT curricula was the degree to which teachers embraced a learner and facilitator role (7). Teachers who became comfortable with open-ended assignments with diverse solutions were able to promote student engagement in the curricula. Having these open-ended assignments initially also created challenges as lower-skilled students struggled without having some support on how to use the environment. Conversely, higher-performing students needed differentiated challenges as finishing their tasks early lead many to engage in 'off-task' behavior.

Based on these results, we set two primary goals for our next years implementation. One was to create lesson designs that better eased novice students and teachers into programming environments. The second was to create lesson designs that promoted student self-guided exploration, differentiated challenges, and scaffolded teachers adopting a facilitator role.

In follow-up work, we adapted the "Use-Modify-Create" (UMC) model (53) to create an assignment design that scaffolded students into developing simulations. Teachers lead students in 4 days of an infused lesson in which they first do an unplugged activity (1), then use code, modify it, and finally create their own code. In an experimental study, teachers and students found the UMC version to be more engaging and less difficult than students who created code all three days(58). While successful at scaffolding students, we found having all students program the same thing did not afford the goal of having open project-based

learning. We describe the original curriculum below then detail our changes in section 4.4.1.

4.3.2 Curriculum

Figure 4.1: The Food Webs Cellular Stage with each animal 'Choice' present.

We designed a 4-day, CT lesson about Food Webs - a scientific topic for 6th grade students. In the food web curriculum, students learn about how energy is transferred from producers to primary and secondary consumers. They do this through developing a simulation in a block-based environment, Cellular (51), an extension of the block-based programming language Snap! (31). We describe each daily segment of the activity below.

Day 1 - The curriculum began with an "Unplugged" activity (1) in which students re-

viewed definitions and components of a Food Web (e.g. consumer types, energy transfer). This ended with students completing a worksheet, lead by the instructor, describing the behavior of agents in the model through pseudo-code. This prepared students for developing these logic models within the programming environment.

Day 2 - Focused on the “Plant" agent (producer), which grows based on the solar energy given by the “Sun". Students had plant and sun code provided and *used* and read through the working code in order to become familiar with the different conditions. The teacher led students in exploration by changing the initial input (the solar energy intensity), the cutoff conditions (how much energy is needed to transition) and the amount of energy lost through transitioning. Students recorded how those changes affected the speed in which flowers changed state on a worksheet.

Day 3 - Focused on the “Bunny" agent (primary consumer). Some bunny code was provided at the outset, though the given bunny behavior did not conform with their idea of the actual model (e.g. bunnies never ate when they got low on energy). Teachers led the students in *modifying* the existing code in order to make it conform to the existing ideas they had discussed on Day 1.

Day 4 - Focused on *Create* in the UMC model. Teachers lead the class through how to develop the “Fox" code, the model’s secondary consumer, with students following along. The class ended with students modifying the initial conditions, bunny and fox code at-will in order to determine how it affected the final model.

This version of the curricula (further referred to as **‘Fox-Only’**) was successful at easing students and teachers into the programming environment and into coding (58). However, students on the final day all code the *same* feature lead by the teacher. As this moves us away from more project-based, student-guided learning, we adapted this curricula to afford student-choice.

4.4 Methods

4.4.1 Adding Student Choice

We modified the final day of the activity to promote student self-guided exploration and move teachers towards acting as facilitators. This curricula, further referred to as **‘Student Choice’** substituted the activity where students changed variables in the simulation on Day 4 with time for students to make extensions from a set of options. Each student was given a piece of paper with a list of possible extensions. The first two options were to extend the

functionality of the *bunny* or *fox* code to include the new feature of reproduction. The third option was to extend the *sun* code such that every couple of days, the weather changed. The final three options were adding new agents to the Food Web: another herbivore - *mouse*, another carnivore - *eagle*, or an omnivore -*bear*. These animal options are shown in Figure 4.1.

We developed these extensions by taking into account prior student requests for code extensions in previous implementations of the curricula. We ensured that each choice had a meaningful connection to the topic content and that each addition to the student's model would change the food web (new animals interacting with each other/weather changing the plants etc.) in a manner to promote scientific inquiry. Every new animal code had the same programming tasks shown in Figure 4.2 as the Fox, differing only in what type of food they ate based on their consumer type (feature 4). In this way, students could develop a complex simulation (with different actors interacting) without having to learn many more code blocks. In a manner similar to a Parson's problem (70), we listed the necessary blocks for completing the code for each option to ease student cognitive load (92).

Our intent was to promote teachers acting as facilitators during programming sessions. With this assignment design, students were more likely to have widely varying code, meaning teachers could not employ the standard teaching strategy of instructing at the front, forcing them to act as facilitators. However, teachers were supported in the fact that student options are limited to a pre-set list. Each teacher would also be given a "Cheat-Sheet" showing what each of the different animal codes should look like.

4.4.2 Implementation/Data Collection

In order to evaluate the differences between our original Fox-Only and new Student-Choice versions, we tested both versions in a quasi-experimental design. Two 6th-grade science teachers from the same school were recruited to teach both the Food Webs curriculum. Neither had prior experience teaching the activity. Teachers were trained on each curricula prior to implementation. In the Winter, both teachers taught one class each using the Fox-Only model. 3 months later, in the Spring, they each taught one class using the Student-Choice model. Each classroom contained around 25 students, though we only have consenting information from 13 and 15 from Spring (28 total) and 17 and 21 from Winter (38 total) respectively. Teachers instructed the curricula while researchers observed, but the research team made themselves available if teachers or students needed programming help. All class periods were the exact same length: 50 minutes.

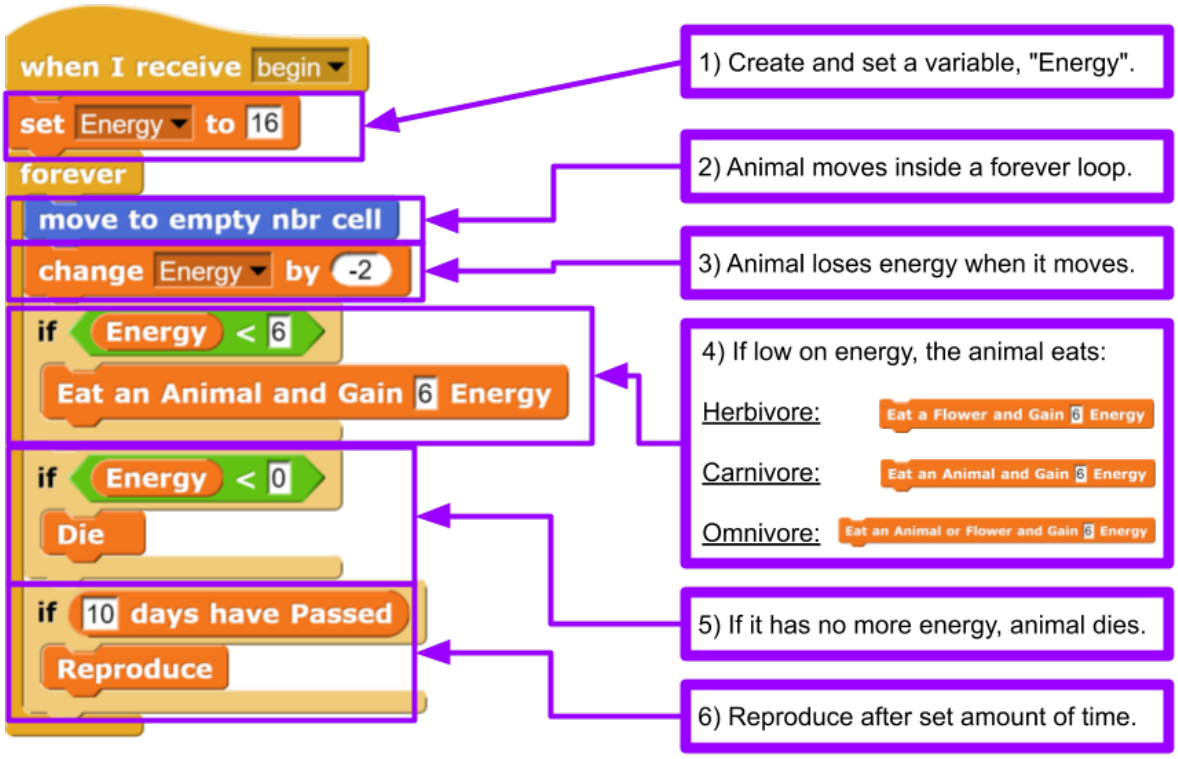


Figure 4.2: Final Fox Code with one extension choice (Reproduction). Teachers lead students in programming features 1 through 5. Every animal choice had similar code except for Feature 4 where animals eat different things based on their agent type (e.g. herbivores eat plants).

The initiatives were evaluated through a triangulation of multiple data sets including student programming traces, student exit-tickets, observations, and semi-structured interviews with teachers (59). For every period, at least one research team member was present taking observation notes, focusing on the students' interactions within the environment as well as how the teacher was teaching the lesson. After the conclusion of each day's activity, students took an end-of-activity "Exit Ticket" in which they answered a series of questions about the activity. As we were focused on difficulty perceptions between curricula, we focus only on student responses to the question, "Please use the following scale to rate how difficult or easy the lesson was today." Responses were on a 1-5 Likert scale ranging from Very Easy to Very Difficult. After both implementations concluded, we interviewed teachers in a group semi-structured interview asking about their experience teaching both versions of the curricula.

In addition, every consenting student's interaction data within the programming en-

vironment was recorded. From this, we were able to collect a series of code 'snapshots' for each student, what student programs looked like at any given point in time during the activity. The series of snapshots in order produces a student's code trace for the assignment. We focus primarily on student code traces for the fourth day as this is where the assignments differed. For each of the requirements for the Fox code as well as for all of the possible choice options on Day 4, we developed auto-grading code to determine if a given student snapshot had completed a task or not. We then ran this auto-grader on each snapshot in each student code trace to determine *what* choices students made in developing their simulations and *when* they finished each choice option.

4.5 Results

4.5.1 Classroom Observations

In all 4 classrooms, teachers spent the initial half of the classroom leading students in the creation of the Fox. There were instances during each session of teachers forgetting a step, but this did not greatly impede leading the classroom and all teachers were able to correct. After walking the students through each of the steps of the Fox code, the teachers then walked around the room helping individual students with the respective next part of the assignment. For the Fox-Only condition, this involved debugging some students' code and answering questions about different run conditions. For Student-Choice, teachers walked around the room helping individual students. Teachers did not seem to have difficulty handling the variety of student choices. Both, in fact, elected to not use their cheat sheet, instead just memorizing the differences each animal choice had with the Fox code they already knew how to do.

Similar to our previous studies, researchers observed many students who finished early in the Fox-Only condition engaging in off-task behaviors. This would happen for upwards of 20 minutes as students who finished the Fox tended to do so with plenty of time remaining. This 'off-task behavior' was rarely seen in the Student-Choice classrooms. Researchers observed many instances of students trying to get other students, their teachers, or even the researchers to see what progress they had made in their environment. As no students had finished all the features, both classes doing the 'Student-Choice' condition asked their teachers if they could continue working on this the following day.

4.5.2 Student Programming Data

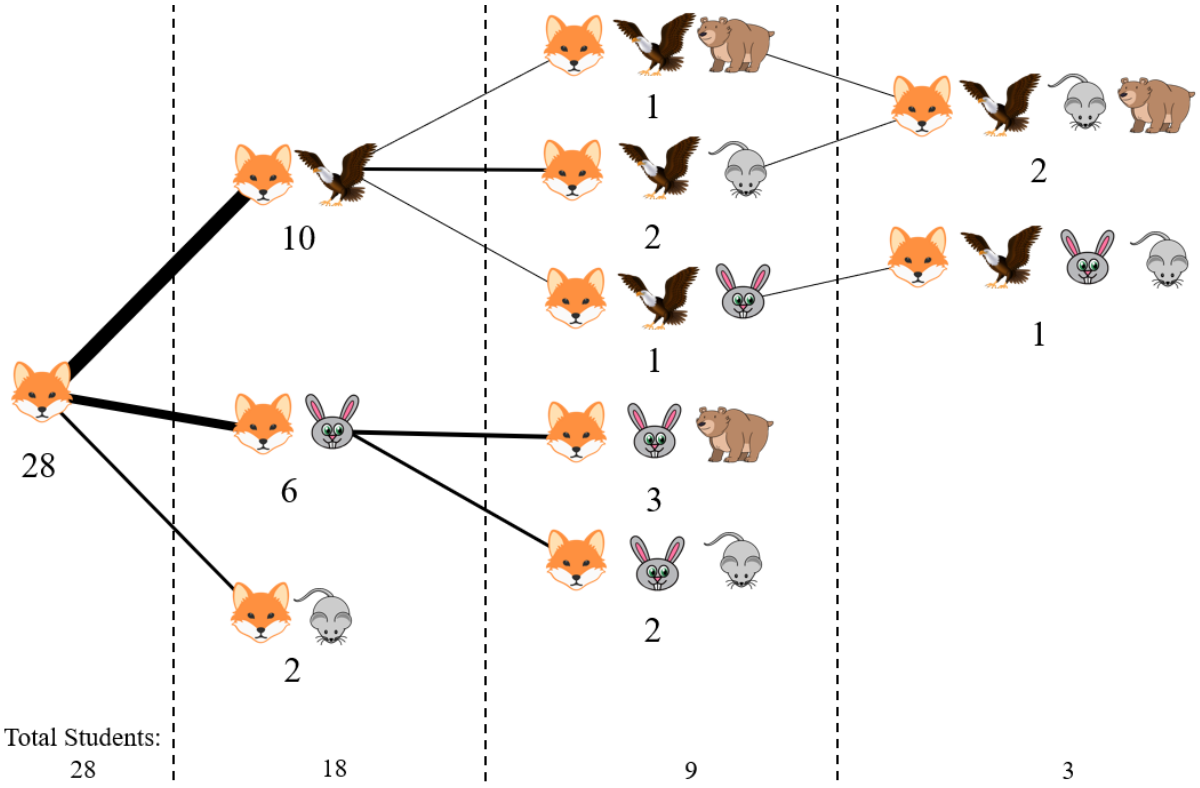


Figure 4.3: The code trace of how students choose to implement each extension option.

We analyzed 66 student traces - 38 from those who did Fox-Only and 28 from those who had the Student-Choice version. As both had to complete the Fox code before their conditions differed, we examined their Fox code completion rate. The students in the Choice condition had a higher rate of completing the Fox code (15/28) than the Fox-Only Condition (15/38) though a Chi-Square with Yates correction does not find this to be significant ($p = .38$).

We then turn to the programming patterns of the 28 students in the Student-Choice condition beginning with the 10 who made no extension. In stepping through the code trace data, 4 of these students did complete the fox code and finished with much time to spare (20 minutes left in the class on average). However, these students elected to instead tweak Fox code, switch the number of foxes in the world, and try different run-time conditions for testing out their simulations. The other 6 spent the entire time attempting, but not

completing, the Fox code. Most of these students had difficulty completing tasks relating to the “Energy" variable including setting it in the beginning, and using it to check if the animal should eat or die.

We found 30 extensions developed by 18 students who made at least one. All of these were animals (no one chose the weather task). 7 students extended Bunny Code from Day 3, 10 students elected to make Eagles, 8 worked on Mice, and 5 made Bears. Of the 18 students that made an extension, half (9) programmed 2 or more additional animals and 3 programmed 3 additional. Students in the “Choose" activity took a variety of paths towards developing their simulations. We outline this in Figure 4.3. Here, the number under each state represents how many students attempted to complete that animal and each edge describes a transition to a different simulation. Here, we can see that the most common first additions were to add Eagle (10 students) or Bunny (6 students).

4.5.3 Student-Perceived Difficulty

On each of the four days for both initiatives, students were asked to fill out an end-of-day exit ticket. 31 of the 38 consenting students in Fox-Only responded for all 4 days, though only 15 of the 28 consenting students in Condition 2 did. Students were asked each day how difficult they thought the lesson was. A Friedman Test for the responses for the Fox-Only condition does not give a significant difference in student perceived difficulty across the 4 days ($p = .10$). This corroborates our previous study which found no perceivable difference in difficulty across days(58). For the Choose condition, a Friedman test finds no significant difference in difficulty as well ($p = .92$). Mann-Whitney U Tests also find no significant differences in difficulty responses *between* conditions in Days 1 ($p = .22$), 2 ($p = .34$) 3 ($p = .52$) or 4 ($p = .91$).

4.5.4 Teacher Interviews

An end-of-initiative semi-structured group interview with teachers was conducted to understand their thoughts on how the two implementations compared. Both teachers felt that the UMC progression found in the curricula was beneficial for students, especially since many had never used the programming environment before. They did suggest that the ‘Unplugged’ and ‘Use’ day be combined as they found those the easiest to get through and finished both early.

Both teachers expressed preference for the Student-Choice condition saying it was

“cool that [students] could decide how many of each thing they could add". They thought the students were more engaged in the Choice condition remarking that “[students] kept running up to me the whole class, ‘look, look what mine’s doing!’ ‘Look what I’ve got.’ They were pretty excited about it." The Choice condition, having more species in the simulation, also better aligned with scientific standards. A Food **Web** is defined as having multiple actors of different types interacting (i.e. multiple primary/secondary consumers) while a Food **Chain**, only has one of each type, like the Fox-Only condition. Teachers “really liked [the Choice condition] how it was more like a food web than a food chain where you add a whole bunch of other little things." When asked, both teachers said they would do the choice version again.

Neither teacher expressed discomfort with acting as a facilitator for individual student projects even if they didn’t have an answer for a student question immediately. A teacher remarked that she was able to answer questions “most times" with only “one I couldn’t figure out but then one of the other kids figured it out". While they felt they could teach the Choice version without researchers present next time, they acknowledged having done the Fox-Only condition before the Choice version might have helped in their preparation. In addition to combining the Unplugged and Use days, teachers expressed the need for more time for students to program on their own finding the final 20 minutes of the last class insufficient. Both teachers, as previously described, allowed students to continue making their simulations on the day after the initiative was done.

4.6 Discussion

Finding no significant differences in student difficulty perceptions, having the majority of students (18/28) complete the base activity, and neither seeing nor hearing teachers report difficulty teaching the new condition, we find that the Student-Choice design was not more difficult for either students or teachers. Thus, we believe we still maintain all the affordances that our original curriculum strategy, UMC, has in easing students and teachers into programming. In addition, the choice at the end gives students a chance to select the option(s) they wish, promoting their autonomy, and choose as many options as they need to satiate their need for competency as defined in Self-Determination Theory (19). While not explicitly designed for, the emergent sharing of materials by students seemed to promote student relatedness. We can imagine further curricula explicitly adding a section for student sharing to make sure that this relatedness is met. In analyzing the diversity of

student artifacts and the pathways students took to make their models along with how engaged students were as noted by researchers and teachers, we believe that the Choice version better engaged students in creating individualized programs which aligns with previous research in the design of constructionist learning environments (47). Following advice from teachers, we believe that further versions of this activity should include *more* time for student differentiation. Further, to more closely resemble the definitions of Create found in UMC, PRIMM, and other models, we believe students can be given opportunities to choose their own extensions after they complete some of the choices from the set provided by the instructor. It is important to note that unlike these full constructionist models, our design is situated in an infused context where it is important to match computing and domain knowledge and creating pre-set choices for students ensures they create things that are scientifically accurate. Furthermore, these infused contexts involve novice teachers and students, so we must not forgo easing them into a new learning environment. As such, we feel 'Use' and 'Modify' as necessary steps on our path towards open-ended, constructionist 'Create'.

In framing our teacher observations and responses to interview questions through Frykholm's theory, we did not find teachers experienced pedagogical or emotional discomfort through our activities. As teachers discussed and our prior studies have shown, creating the scaffolded progression of moving from 'using' code to 'creating' their own seemed to reduce teacher discomfort in teaching content. Teachers also reported no difficulty in adopting a facilitator model. This could help scale our research practice partnership efforts to help expand CT activities into additional teachers' classrooms.

4.7 Conclusion/Lessons Learned

Despite its exploratory nature, this case study supports that a curricula designed for freedom of choice is useful for both teachers and students. As we did not measure CT competency, we make no argument that this design works better or best at *teaching* programming and CT concepts. The small sample size of teachers (2) and students (68) does not allow us to extrapolate that this method is best for all teachers, contexts, or classrooms. Additionally, teachers first leading the non-choice version prior to the choice version could have greatly influenced their curricula preference as well as their development of competency in teaching the material. The activities' warm reception, however, leads us to believe that content developers could benefit from creating activities that end in perceived student freedom

of choice. To aid with this, we provide the following list of recommendations for creating these activities:

Scaffold Students and Teachers - Providing the necessary programming blocks students need to complete a choice ala a Parson's Problem greatly reduces cognitive load, especially in activities where lots of choices means lots of blocks. Similarly, giving teachers a "Cheat Sheet" of answers for each choice, scaffolds their ability to act as a facilitator and debugger.

Differentiate Choices by Difficulty - This time, challenge came from adding *more* choices, but each choice was relatively the same difficulty. In the future, we wish to create choice systems that have varying difficulty to give targeted tasks for each student skill level.

Create Choices that Show Visible Change - Prioritize choices that produce immediate changes in the run of the simulation (e.g. a new animal appearing in the environment).

Create Choices that Promote Content Inquiry - Our decision to primarily focus on adding more animals came from the fact that each new animal made the Food Web more complex.

Make things Complex, not Complicated - As demonstrated in Figure 4.2, each animal had relatively the same set of code blocks. In this way, a lot of choices were available without adding many additional necessary blocks for students.

Draw from Student Desires - Some of the choice ideas (e.g. bear) came from responses from students in the 'Fox-Only' condition on how they wished to extend their model. We suspect if students are asked in the beginning what they'd like in their model and these choices are implementable at the end, students will engage more with the material, feeling like the creations are their own.

In the future, we will describe how to develop these student-choice, CT, activity types for professional development for K-12 teachers. We hope that in promoting successes from our implementation, we can encourage teachers to adopt, adapt, and create their own activities that make them feel comfortable facilitating and allowing students to engage in self-guided inquiry and learning.

CHAPTER

5

CEO: A TRIANGULATED EVALUATION OF A MODELING-BASED CT-INFUSED CS ACTIVITY FOR NON-CS MIDDLE GRADE STUDENTS

Lytle, N., Cateté, V., Dong, Y., Boulden, D., Akram, B., Houchins, J., ... & Wiebe, E. (2019, May). CEO: A Triangulated Evaluation of a Modeling-Based CT-Infused CS Activity for Non-CS Middle Grade Students. In Proceedings of the ACM Conference on Global Computing Education (pp. 58-64).

5.1 Introduction

Computer science and computing technologies have become recognized vehicles for economic growth and development across the globe. In order to meet the demands of this growing workforce with well-qualified employees, more countries are focusing on increas-

ing their primary and secondary grade computing courses. Computational knowledge has far-reaching benefits beyond standard technology companies. Computational thinking (CT) (98) and other aspects of computing are being used to solve advanced, multi-faceted, problems around climate change, marine ecosystems, and beyond. Introducing computing into the science classroom provides equitable access to computing training as well as demonstrates to students the usefulness of computing in outside fields.

Previous research has examined teacher preparedness and professional development on the impact of classroom implementation and teacher confidence (7; 75; 15). Studies show that students with teachers more interested in facilitating CT exhibit higher levels of time on task and engagement than those with disengaged teachers. While previous research focuses primarily on teacher outcomes, more research is needed on student outcomes (i.e. learning and programming behaviors) in order to successfully gauge the curriculum and support materials.

We propose a CEO model to assess the implementation of a CT-infused science curriculum. The model examines student **C**ode traces, **E**xit tickets, and field **O**bservations in order to triangulate the effectiveness of curriculum implementation and student learning outcomes. In order to effectively infuse computing, students must achieve both an understanding of computer programming as well as reinforcement of the scientific concepts being taught in class.

Using the CEO model, we attempt to identify improvements for the existing curriculum, specifically:

1. How are students completing the assignment tasks?
2. What emergent behaviors or observations can we identify in order to better engage students in future iterations?
3. How do students perceive the science vs CT learning goals?

5.2 Background & Related Work

Research practice partnerships (RPPs) are long-term collaborations between practitioners and researchers that are organized to investigate problems of practice and solutions for improving schools and school districts (12). One type of RPP focuses primarily on design-based implementation research (26). This research aims to study solutions implemented

in real world contexts, typically utilizing a cycle of developing and testing instructional activities and curricula.

We detail the cycle of creating, piloting, and re-designing a life-sciences curriculum in a prior case study (41). The grounding decisions for activity changes included realizations that the cognitive demands for learning both CT and science concepts could be too high for the diverse range of middle-grade students. Additionally, students without prior programming exposure would benefit from tutorials and scaffolded directions. Thirdly, interface distractions could derail students and cause extra demands to cognitive load(92). Many of these realizations however, came from short lab-controlled settings or subjective team reflections and discussion.

As Fishman points out, there is a broad challenge to gather and interpret evidence of effectiveness in the field as this differs greatly from a controlled setting (26). Strategies for gathering evidence include student assessments, self-report surveys, rigorous observations (66), along with trace data analyses and video-recorded sessions. Although many of these practices are common in the field of educational research, they are often difficult to set up and take place primarily in closed lab environments (85; 88).

Upon entering the implementation cycle of design-based research, additional data collection methods should be used. In a previous report, educational researchers described the implementation of computing oriented science activities in middle grades classrooms (7). This research focused primarily on the teacher's willingness to adopt the activity and effects of just-in-time professional development, noting that teacher buy-in had a large impact on student perception of usefulness and active on-task engagement. Although this research is well documented, it relied solely on qualitative data, including teacher interviews and observations. The report did little to convey the student learning outcomes or other evidence of student success.

Conversely, work by Grgurina et al. focuses on assessing modeling activities in a secondary grade computing classroom (35). The assessment uses a combination of the Revised Bloom's Taxonomy (50) and SOLO taxonomy (4) to evaluate students' written answers to a number of questions regarding their models on multiple dimensions ranging from prestructural (information makes no sense) to extended abstract (generalization and transfer) for areas of design, experimentation, and reflection on the model. This assessment, however, is summative in nature using a multi-week modeling and research homework assignment as it's main vessel. The assessment provides less formative feedback on their specific skills and knowledge and instead measures larger concepts.

As our current research concerns middle grades students who are still developing their

scientific knowledge and coding abilities, more formative feedback is needed. Consequently, we need to be able to better assess their programming behaviors so that we may provide more support or scaffolding for emergent behaviors in future iterations. Prior research in assessing science and modeling activities in secondary classrooms have focused on teaching, adoption, and summative assessments. In our new research, we look at evaluating an experimental curriculum through both empirical and qualitative evidence of student understanding.

5.3 Methods

We created and utilized a CEO model to assess student outcomes, our curriculum as a whole, and the effectiveness of our implementation. Each of the elements of CEO describe a different data set and methodology for curricula analysis. Code Traces (C) describe the actions within the coding environment used by the students, Exit Tickets (E) describe post-activity written responses, and Observations (O) are researcher notes taken during implementation. In the following sections, we describe the curriculum and implementation, followed by our data collection methods.

5.3.1 Curriculum & Implementation

The curriculum development team, composed of educational psychologists and computing education researchers, created an 8th-grade life science lesson on Epidemic Diseases infused with computational thinking. The lesson was designed to be aligned to national and state-level science standards as well as the k12cs.org Computational Thinking Framework. This five-day unit focused on modeling the spread of epidemic diseases like the flu, see Figure 5.1.

Individual activities are described as “plugged" if students used a programming environment, or “unplugged" if students were learning without a computer. Students were given a block-based programming tutorial prior to the run of the unit. The unit features 2 unplugged days focused on modeling agent-host relations in the transmission of disease. There were also 3 plugged days: 2 of which had students develop a simulation based off the model discussed on the “unplugged" day using the block-based programming environment, Cellular (51), and 1 day where students used the environment to solve scientific research questions about the spread of diseases. An overview of the activities are shown in Table 5.1.

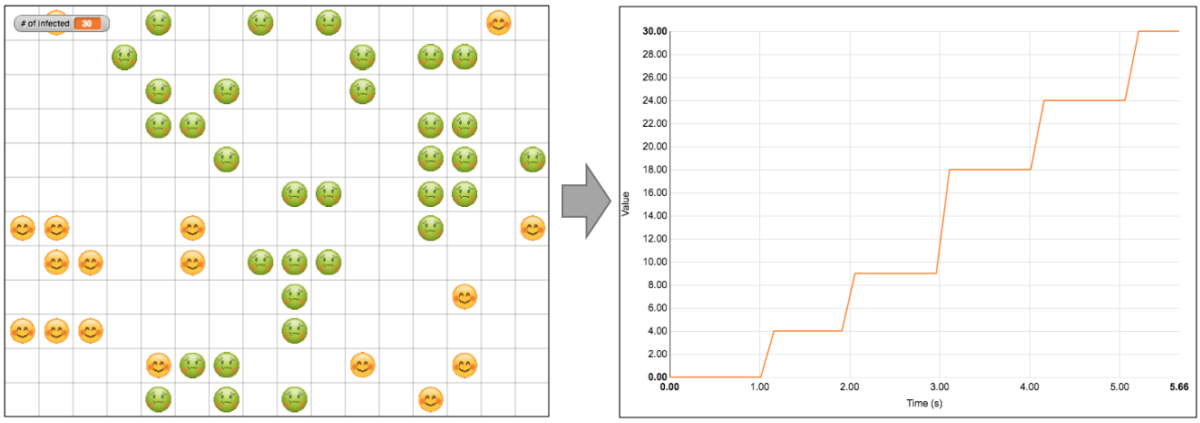


Figure 5.1: A Cellular representation of an epidemic.

Table 5.1: Epidemics outline, P: plugged, U: unplugged

Day	Learning Goals
1: U	Able to explain a simple model. Understand Hosts and Agents share properties with modified values.
2: P	Define infection and infection rate. Demonstrate understanding of agent properties. Understand and use loops and conditionals.
3: P	Understand disease spread and rate of transmission/infection. Use variables to maintain count. Analyze trends in data to identify patterns. Demonstrate understanding of how interaction properties can affect simulations.
4: U	Understand Morbidity/Mortality rates and their influence on spread. Understand and use Finite State Machines to model logic flow. Model algorithmic thinking through transition modeling.
5: P	Understand how environmental factors affect disease spread; Learn experimental procedure for hypothesis testing. Visualize data; Use simulations to test hypotheses.

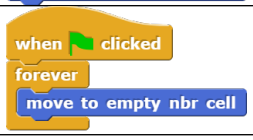
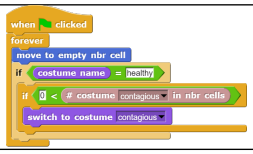
We tested the 5-day Epidemics activity with two 8th grade science teachers at a local middle school. Each teacher taught 5 sections of students with class sizes ranging from 20 to 25 students. Each of the teachers' classes were presented the same activity on the same day. Teachers taught every class period on unplugged days of the unit. On plugged days (2, 3, and 5), teachers had one observation period at the start of the day where they experienced the content as a student, following along as the researchers led the first class. They then taught the remaining 4 periods of the day, guiding students through program implementation. 61 students returned consent forms for us to analyze their data.

5.3.2 Code Traces

On days 2 and 3, students' actions within the Cellular environment were logged in a database for later examination. We focus our analysis on the actions that make changes to their code. Specifically, we call a student's code at any point a *code state*, and the sequence of code states that leads to the final solution a *code trace*.

For days 2 and 3, students were given instructions with individual tasks to complete within the environment. These tasks were presented sequentially, but the order in which students could complete the assignment did not have to match the intended order. These tasks are outlined in Tables 5.2 and 5.3. In examining student code traces, we can identify whether or not a specified task has been completed at any given point in the code trace. We refer to the instance in which a specified task has been completed as a *task completion state*. The second column of Tables 5.2 and 5.3 provides one of the many solutions that would result in the completion of that task.

Table 5.2: Task milestones for day 2 of the epidemics activity

Task/Feature	Example Solution
(1) Write a program that moves your sprite to an empty neighbor cell.	
(2) Use a control loop to have your sprites move to empty neighbor cells forever.	
(3) Add an if control block to your main script that checks whether your sprite is healthy before checking its neighbor cells for an infected sprite.	
(4) Add a script to your forever loop that makes your sprite infected if an infected sprite (a sprite with infected costume) is in a neighbor cell.	
All	

Task completion states were individually tagged by a member of the research team and build sequentially as students complete additional tasks. When a student has a task

completion state with all tasks present, they have fully completed the assignment. To illustrate, take two students who completed day 2's activity, one by entering completion state "1234" and another by entering completion state "1243". The presence of all four tasks (i.e. tasks 1,2,3,4) in both states signify both students completed the full assignment. The order of the numbers in the state tells the sequence of tasks the students completed in the assignment (notice the order difference in how they completed the last two tasks - 3 and 4).

Using code traces to identify student programming behaviors gives light to RQ1 as they can illustrate common solution strategies, most difficult tasks, and common pitfalls in student implementations. As these states and actions are also time-stamped, specific strategies by students can be compared to see whether or not certain strategies are easier (i.e. faster to complete) than others.

To complement our task completion state analysis, we went through traces and made qualitative observations of other behaviors in the environment we found interesting. These included behavior taken by the students after the assignment was complete, resets of the environment (giving up and starting over again), spots where students found difficulty in completing tasks, and off-task behavior.

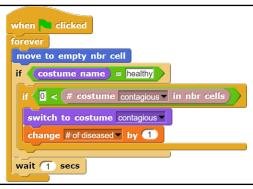
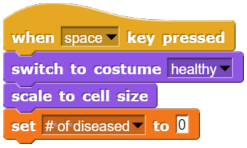
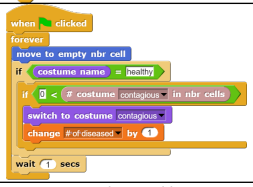
5.3.3 Exit Tickets

On plugged days, students were asked to take an end of class "Exit Ticket". This series of questions measured self-reported affect, student engagement, and perception of what they learned. The 61 consenting students generated 127 responses over three days of activity. The relevant questions are listed below:

1. What did you learn today?
2. What was the most helpful to you: the tutorials, your classmates, your teacher?
3. Did you find anything difficult or frustrating? Please explain.
4. Did the lesson go too fast, too slow, or just right?
5. Do you have any suggestions to improve the activities?

For question 1, our aim was to answer RQ3 by analyzing student perception of the intent of the activity. As this was a CT-infused science lesson, we aimed to see whether or not the perception of the activity leaned more towards computing or science. Our approach consisted of having two researchers individually tag responses as being more focused on

Table 5.3: Task milestones for day 3 of the epidemics activity

Task/Feature	Example Solution
(1) Create a variable to hold number of infected sprites and name it # of diseased. Set this variable to zero in the beginning of your program (just after the “when green flag clicked" block)	
(2) Write a script that increases the value of the infected people by one when a sprite gets infected.	
(3) Write a script that restarts your simulation when you hit space. The restart script should make all sprites healthy and set the number of infected to 0. Add your restart script to this part of the code on the screen.	
(4) Write a script that changes the value of the variable: # of diseased when you click on a sprite.	
(5) Remove the set # of diseased from your main script (the script after “when green flag clicked")	
All	3,4, and 5 all co-present.

computing, science, both, or neither. After individual tagging, the researchers compared their tags and discussed cases where conflict existed until agreement could be reached. For questions 2 and 4, our aim was to see which of the three choices or combinations thereof were most present within an assignment. Finally, for questions 3 and 5, researchers counted and grouped the most common answers to use as feedback for the next development cycle.

5.3.4 Observations

During the implementation, at least one member of the research team acted as an observer on each plugged day. Each observer had prior classroom experience leading computing activities with middle and secondary age students (with three of the observers having 5+

years of experience). Our goal with classroom observations was to experience and record how the activity was conducted across different classrooms and to gather both insights for improving teacher training practices as well as improving the actual activity and its implementation environment. In most activity sessions, we assigned one observer to watch the teacher, and others to observe the students. We also had additional support to help students stuck on programming tasks, so that observers could stay focused on the field. Observers recorded student affect, behaviors, and interactions of note. After each session, observers conferred and noted interesting results to follow up with.

5.4 Results

5.4.1 Code Traces

In Figures 5.2 and 5.3, we represent the paths students took to complete the assignment as a transition of progressive task completion states. These states correspond to the subset of tasks that have been completed thus far (e.g. state “12" represents finishing task 1 and then 2). The size of the shape corresponds to the number of individuals who entered that specific task completion state, and the size of the edges corresponds with the number of individuals who made that unique transition. Task completion states can often be stuck states (octagons in this representation) meaning that some students made no further progress in the assignment after reaching this state. This representation was adapted from a similar analysis of student programming paths(see (100)).

Day 2

The code traces for 61 participants on day 2 are shown in Figure 5.2. The task completion was mostly uniform, with participants only differing on whether or not they completed task 3 or 4 first. Most students, 33, completed the tasks in the intended order of 1234 while 22 completed the ordering as 1243. The third feature was the most missed (7 students omitted).

Timing trends were also recorded and analyzed within the environment. The average time spent in the environment was 36 minutes. However, average time in environment varied considerably by class period. Periods 2 and 3 spent the most amount of time (41 and 39 minutes respectively) and period 4 spent the least (31 minutes). The last two periods spent 36 and 33 minutes within the environment. This difference in timing is important as

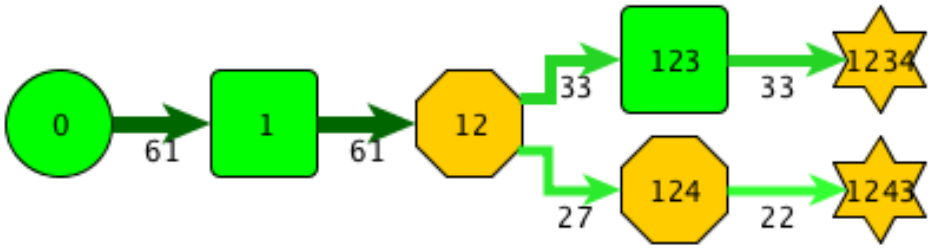


Figure 5.2: Task completion strategies for day 2.

all students within period 4 completed the features in the order 1243.

In looking through the traces, there is clear evidence for student confusion. 14 students reset the environment and started over at least once. Many students tried to complete their implementation with *distractor blocks* (blocks not useful in final solution). The most common distractor block was the “Touching" block, which was used to check if two sprites were overlapping, instead of checking if an agent was in a “neighboring cell". Another common block that students attempted to use in the solution was the “Move Step" block which was used in lieu of the “move to empty neighbor cell" block.

Students largely were able to complete the assignment with relative ease as there was 17 minutes on average left from the last feature was completed to close of the system. Due to the amount of time left at the end, many students elected to complete additional extension features with the most common (39/61) being a “reset functionality" that resets the simulation to its initial position.

Day 3

Among the 59 code traces, the average time spent in the environment was 39 minutes, ranging from 34 to 42 minutes. There were markedly fewer resets (only 6 students) than the previous day’s programming assignment and many of these occurred after students made an error such as creating a new sprite (and losing track of their current sprite).

What is easily apparent by comparing the graphs of this assignment with the other is the incredibly varied paths that students take through the assignment. 11/59 students were unable to finish the assignment with five of these students missing all but Task 5. The 48 students who did finish the assignment took a myriad of paths. The majority of these solutions (34) did NOT include tasks 1 and 5. In fact, 44 students did not complete task 1 at all and of the 15 that did, nearly half (7) did not complete task 5 afterwards (and thereby

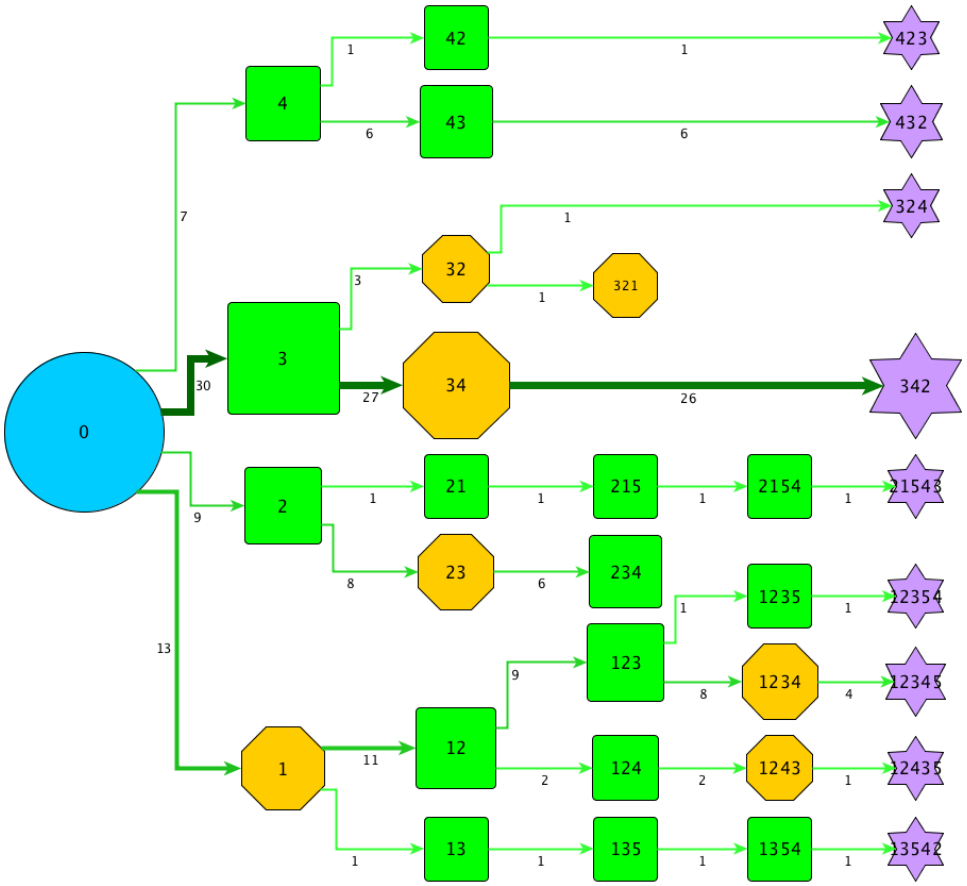


Figure 5.3: Task completion strategies for day 3.

not having correct final code).

Students who completed the assignment had less time on average (10 minutes) between finishing and close of the environment than those working on assignment 2, but this split also varied by solution strategy with those who did tasks 1 and 5 having less time (around 8.5 minutes) than those who just did only tasks 2, 3, and 4 (around 11 minutes). With the remaining time, students attempted a number of self-created extensions. Some students attempted to make other sprites in their model such as hospitals. Others attempted to extend the logic of their modeling by either stopping the simulation once the entire population was infected or keeping track of and graphing the remaining healthy population. Another student attempted to add interactivity into their model with key press events.

5.4.2 Exit Tickets

In addition to the main breakdown of students reporting learning science vs computing (see Table 5.4), we also wanted to examine how students articulate their new computing knowledge. We classified their learning statements by whether they mention a computing term or mention the term in connection to a specific goal. On day 2, 15 students mention sprites and costumes, 10 with specific goals in mind. Seven students mention programming blocks, but only two with a specific use in mind. On day 3, 23 students mention variables and plotting, 13 of which with a specific goal for use. This time, only four students mentioned sprites or costumes. Day 5, showed the biggest swing with most students reporting science, though 5 students also mention programming in general.

Table 5.4: Student responses to Exit Ticket questions 1, 2, and 4. Answers of none are omitted from the table.

Day	Balance	Most Helpful	Pacing
2 N=39	Computing: 23 Science: 3 Both:12	Teacher: 28 Tutorial: 5 Classmates:2	Perfect: 29 Slow: 5 Fast: 3
3 N=49	Computing: 34 Science: 5 Both:7	Teacher: 32 Tutorial: 8 Classmates:6	Perfect: 30 Slow: 13 Fast: 2
5 N=39	Computing: 0 Science: 26 Both: 7	Teacher: 24 Tutorial: 3 Classmates:6	Perfect: 23 Slow: 4 Fast: 5

Most students on each day reported that they did not find anything frustrating with the activity. Of those that did, eight found programming errors and debugging frustrating, while five students found system related errors (problems saving, crashing, etc.) each day frustrating to deal with. Five students on day 2 and day 3 specifically mentioned adding the hospital was difficult, but this was an extension activity not explicitly supported in the curriculum. Finally, day 5's specific activity brought new challenges for five students as some were frustrated and confused about the terms introduced (e.g. independent variable) as well as using the tools in order to find the relationships in the assignment.

Finally, students had a number of suggestions on how to improve the curriculum. Suggestions can be broadly categorized as having to do with the instruction (providing more

support or improving the pacing); fixing technical problems with the programming environment (e.g. crashes, bugs); and activity-related suggestions (e.g. more extensions, time for group work, creative time).

5.4.3 Observations

Observers identified general patterns on all three days. First, concerns were raised about the task sequence in both days 2 and 3. For day 3, observers noted that the action of setting # of diseased to 0 under the Green Flag in task 1 didn't seem to serve a purpose and caused confusion in some cases. In later periods, teachers skipped this step entirely.

Observers noted that the ordering of tasks in day 2 was off. When comparing periods, the class seems to run smoother when students work to iteratively solve a problem and the nested-if (task 3) does not produce any visible change. Observers also noted that teachers adopted different strategies in different periods, resulting in the difference in pacing of the assignments. In some periods, the activities were led in an incredibly instructionist manner, leading students through each step. As teachers got more comfortable with the assignment, they began to add their own pacing and scaffolding approaches such as having students explain previous days code or creating parsons problems (69) for students to reason through as a group. Towards the end periods, teachers found ways to explicitly connect science concepts into the programming instruction.

Researchers observed the majority of students actively engaged during all three days of instruction with minimal off-task behavior. While students were generally able to code within the environment, some confusion was observed during day 5 of the activity. Students struggled to setup the simulation environment in a way needed to answer their group's research questions. This was alleviated in later periods with deliberate instructor support.

5.5 Discussion & Limitations

In addressing the question "*How are students completing the assignment tasks?*", we examined the connection between code traces and observations. Overall, day 2 and 3 completion rates were strong with 55/61 and 42/59 students fully completing the assignment respectively. Those that completed the assignment tended to have extra time left, suggesting mid to high performing students need additional activities and extensions. Initial observer concerns about task sequencing were corroborated by examining students progression through the completion states. For day 2, the quicker completion time as well as observa-

tions suggest that the task sequencing should be 1243 instead of 1234. Task 4 is the behavior that actually visually changes the sprites to infected, which as observers noted, is critical in engaging students. Moreover, task 3 requires a doubly nested if, while task 4 only requires a single if, which is a better scaffolding strategy and smoother sequence than the original task sequence (as noted by field observations of teachers guiding the instruction). The final recommendation for day 2's sequence is informed by the large number of field observed and code trace observed end of day behaviors of adding in the reset functionality. This should be explicitly added as a day 2 task, as the behavior and functionality is necessary for day 3's activity.

In examining day 3's sequencing, the most apparent result was that the placing of the set block under the green flag in task 1 and the removal of this block in task 5 were unnecessary additions to the assignment. Not only did it impede progress in many cases as evident by the code traces, observations from researchers showed that teachers and students would often forego actually completing the tasks at all. This lead to faster overall completions of the assignment as well as fewer stuck states. As the plurality of students completed the task by completing task 3 first and without doing tasks 1 and 5 fully, the results strongly support removal and reordering of the tasks.

In answering the question "*What emergent behaviors or observations can we identify in order to better engage students in future iterations?*", we identify several behaviors of actionable interest. First, blocks that are identified as distractors impede progress and can lead to student frustration and confusion, noted by both observers and exit tickets. Within the stuck states, it is also observable when students don't know which block to use. Since the goal of the curriculum is for the student to build the correct logic for the epidemics model instead of identifying the difference between blocks, we will hide the distractor blocks in future iterations in order to focus attention solely on blocks necessary for completing the assignment. Second, student exit tickets, observations and code traces all support the idea that students want more opportunities for open-ended exploration. With many extensions attempted in the extra time on days 2 and 3, the curriculum should explicitly provide additional scaffolding for students to be able to complete these extensions, including the reset functionality completed by the majority of students on day 2. Supporting these extra activities will also alleviate pacing issues with students finishing early and displaying disruptive/bored behavior.

In answering the question "*How do students perceiving the science vs computational learning goals?*", we primarily examine exit tickets and observations. Students self-reported learning topics correlate strongly with the amount of programming on a particular day.

On days 2 and 3, which focused creating the simulation to determine rate of infection, the majority of students reported learning computing (89.7% and 83.7% respectively). On these days, students were able to articulate the connections between the sprites/costumes and the resulting agent behavior in the simulation as well as the usefulness of variables in being able to track the rate of infection. On day 5, which focused on solving research questions within the simulation the majority of students (84.6%) reported learning science. This trend in perceived learning, is further supported by observations that suggest teachers make deliberate breakpoints to connect the computing to science concepts previously covered in class as teachers introduced more guided reflections and connections with material as the day progressed. These moments of connection, created by teachers, should be explicitly supported within the curriculum in order to be able to truly promote the lesson as an infused activity.

Furthermore, several students reported the tutorial being helpful for each day, however, the majority of students report help from the teacher being most useful. Observers noted the teacher had a good sense of pace of the students on programming days, and helped students construct hypotheses and explain terms like susceptibility which they only briefly covered prior. The teacher also provided logical reasoning and discussions for students to understand why the simulation behaved they way it did.

The scope of this study is limited by mainly focusing on plugged days and not unplugged. We also did not record the action logs of students during the final day as no coding occurred in the environment. An issue that was not addressed in this study was student assessment of scientific concepts as this was also beyond the scope of our research. Finally, as is the case in much classroom research, an observer effect may have influenced findings.

5.6 Conclusions & Future Work

In this analysis of a CT-infused life science lesson, we created and used the CEO model to triangulate data from code traces, exit tickets, and field observations to evaluate the success of student outcomes in the experimental curriculum. Using these data measures, we identified popular student programming pathways that lead to successful completion of the assignment and pathways that lead to stuck states. The code traces provide empirical support for the subjective field observations and student exit tickets.

Other observations and suggestions corroborated by code traces include staging tasks in such a way that each milestone has a visual component or perceivable outcome by the

students. Additionally, on days where students complete the main tasks quickly, additional content can be included, specifically, the functionality (such as reset) that most students already take upon themselves to complete. In general, these findings suggest a role for code traces as a initial attempt to empirically corroborate purely qualitative-based evidence. Thus, promoting a more robust pathway for iteratively improving computing-infused activities for non-computing students. While this process was done through inspection by a member of the research team, this process could be automated through the creation of unit tests for intended code behavior for each task.

A natural progression of this work is to investigate the transferability of these activities. Following Means and Penuel 2005, we want to further identify, “what works where, when, and for whom" (64). We have piloted the epidemics curriculum in two additional schools within the same county. However, we would like to investigate if the lessons will also work at another institution, with a different school culture at differing times in the school schedule (i.e. before/after lesson plan; exploration vs. review activity).

CHAPTER

6

DATA-DRIVEN APPROACHES FOR EXPLORING THE EFFECTS OF TEACHER INSTRUCTION ON STUDENT PROGRAMMING BEHAVIORS

Lytle, N., Cateté, V., Dong. Y. 2020. Data-Driven Approaches for Exploring the Effects of Teacher Instruction on Student Programming Behaviors. In Proceedings of the 4th Computer Science Educational Data Mining Workshop (CSEDM).

6.1 Introduction

To address inequities of access to computer science education (61), teachers and researchers have begun partnerships to put computing assignments in required STEM courses. Though we have found success introducing these new materials and working with in-service teachers, we often find that many, especially those self-described as novices, exhibit levels of

pedagogical and emotional discomfort with programming assignments which reduces their confidence in teaching with the new curricula. Teachers often adopt instructional styles for our lessons that are more aligned with their traditional teaching methods, which for middle school grades is often direct instruction. Direct instruction is often done in the form of 'Live-Coding', showing the process of developing the final code. If teachers chose to live code, they often do not have a sense of how the class is doing, whether or not they are following along, or whether or not what they as a teacher are programming is correct. Though our classroom observations provide teachers with information on how students are following along with the lesson, this methodology is not scalable and we seek more quantitative data-driven methodologies to investigate student programming behavior.

To better understand the effects of teacher instructional practices observed by the research team, we take a closer examination of student and teacher coding artifacts, utilizing the logged interactions of the coding environment. Specifically, we are interested in *how students follow along with teacher instruction.* We perform exploratory analyses to identify interesting patterns in student behavior *concerning completion timing, paths, and rates*. Using task feature visualizations, we present case studies of teachers exemplifying the various teaching patterns we found echoed in student code. We use classroom observation notes to triangulate, ground, and explain the findings we illustrate through these novel methods. We believe that a combination of these data-driven approaches could lead to both scaleable offline and online analysis of programming instruction, informing the development of intelligent dashboards designed to aid both novice and expert programming teachers during live coding instruction.

6.2 Background

6.2.1 Computing in STEM Education

It is becoming increasingly necessary for every child to have experience with 21st-century Computational Thinking (CT) skills (98) which are typically taught alongside programming in elective Computer Science classes or outside of school activities (61). To reach *all* students, computing must be integrated into required STEM K-12 courses. This will have an added benefit to the teaching of STEM content as using computational tools has been shown to enable deeper learning of STEM content areas for students (16). Middle grades has been identified as a critical age range to study the potential for developing CT. In this age-range, curricula that use block-based programming languages are often employed (38).

Block-based programming languages such as Scratch (83), Snap (40), and Cellular (51) are increasingly popular and have been shown to scaffold novices into learning programming in STEM contexts better than traditional text-based languages (95). While block-based environments can support novice students, teachers must be trained in computing topics and integrated computing lessons must be developed for their classrooms. These are usually both accomplished during teacher professional development seminars such as Infusing Computing described in Jocius et al (44), though this might not be accessible for all teachers. Another avenue, like the work our research is situated in, is through research practice partnerships (11), where we collaborate to iteratively improve new curricula. Our research follows design-based implementation research (26), resulting in live field tests in which we observe teachers and students using the new materials in an active classroom. When collaborating with in-service teachers to iteratively build and refine a computing-integrated activity, we actively focus on professional development and teacher comfort. Using a faded-scaffolding approach and gradual release of responsibility (72), we model the desired instruction of the teacher's own class in their native setting. We choose the pedagogical approach of live coding based off of the following arguments below.

6.2.2 Instructing Programming in K-12

Teacher's self-efficacy and discomfortness are important factors to consider when trying to get core subject teachers to add computing into their regular classes. As research by Frykholm suggests, a teachers self-efficacy filter in combination with tolerance for discomfort will affect their willingness to adopt new materials and curricula (30) such as computing. In order to mitigate discomfort, teachers tend to employ more instructional-based practices such as focused or guided instruction that contrasts the constructivist and open-ended teaching practices which are loudly promoted in computing (2). However, at the middle school and young learner level, evidence from controlled studies demonstrate that direct, strong instructional guidance proves more efficient in terms of long-term learning than constructivist-based, minimal guidance (49). This is further supported by Sweller's cognitive load theory which suggests that free exploration of a highly complex learning environment may generate a heavy working memory load that is actually detrimental to learning. This suggestion is particularly important in the case of novice learners, who lack proper schemas to integrate the new information with their prior knowledge (92).

These prior general findings apply directly to programming instruction. A study by Lin and Dalbey (56) shows that medium and lower performing students do better with explicit

instruction and have no negative impacts on the high performing students. This type of instruction tended to minimize the influence of access to computers outside of school and the influence of student ability on outcomes for those of medium and high ability (56). Husic further argues that beginners lack a repertoire of useful approaches to thinking about and learning programming. To build this critically important knowledge base, teachers of introductory classes need to be specific when providing integrated information, problem-solving support, and feedback (42). Direct instruction reduces the working memory load of the students and lets them concentrate on the tasks at hand (56). Thus, for introductory students, early in the learning curve, teachers should provide a supportive scaffolding (13) to help students carry out a task. Accordingly, when teachers provide scaffolding, they generally carry out parts of the overall task that students cannot yet manage.

The research above suggests that K-12 teachers might benefit from adopting the pedagogical technique of Live Coding. Live Coding, often found in university programming courses, has an instructor start from scratch (or some starter code) and build an entire program related to the instructional material in front of the students(71). Unlike static code snippits, displaying the construction of code gives students insight into the programming process (3) and can usually be done in a manner that allows students to follow along with the material, creating an active learning context. Various academic studies have demonstrated the benefits of live coding as an instructional technique (81; 86) even for instructing in block-based programming languages (87). As it is a popular teaching technique, intelligent dashboard support for Live Coding has become popular (9), though Chen et al. focus on making the process of Live Coding easier rather than analyzing the dynamics between instructor and students. It should be noted that nowhere in definitions for live coding does it specify that students follow along. However, given the research into middle grade computing described above, we feel that active, follow-along participation driven by instructors live coding will provide the best platform for students to learn computing in these contexts. Additionally, those who have some prior experience will be able to independently move on without the instructor, and those who are more novice can follow along intently step by step. Given this instructional technique and the classrooms that have already participated in this style, we seek to use additional advances in programming trace data analytics to understand classroom dynamics during live coding sessions.

6.2.3 Programming Trace Data and Analytics

There are dozens of systems that analyze and collect student programming data (an introductory review can be found here: (43)). These include systems for block-based languages like iSnap (77) which can be used to evaluate how students solve programming problems by logging interactions students have within the environment (e.g. running their code, deleting blocks, selecting menus, etc.). In programming data-mining, The sequence of all interactions within the environment is called a *trace*, which comprises a list of all *states* a student environment is in and the interactions that connect them. An important subset of this trace is all interactions that result in differing code (e.g. adding, deleting, and relabeling code), called a *code-trace* and all the different *code-states* (i.e. unique code) that a user progresses through on the way to complete a problem. This code trace information is used for intelligent programming environments as input for the creation of data-driven next-step hints (76) as well as other instructional support like worked examples (100).

While useful for data-driven algorithms, the size of the space of all possible code-states is in practice, incredibly large, even for a small programming problem (100) and has the possibility to be infinite. As such, methods have been developed to collapse the state-space of programming problems to something more manageable for analysis. Prior work has represented the varying pathways of student attempts using a *feature-state* representation (100; 21). In this manner, an assignment can be seen as a set of *features*, and a student's current *state* in the assignment can be represented by which features the student has present or absent in their code. Rui et al (100) demonstrated techniques to visualize student pathways through a programming assignment by representing which features students progressively added to to their code until reaching a final solution. Lytle et al extended this methodology to be useful in informing curricula design by looking at common pitfalls in student programming pathways (60). These feature-state representations are also useful in generating data-driven help such as worked examples based off of students completed code and their assumed next step (102). However, individualized next step help might not be sufficient as studies of help seeking behavior in intelligent programming environments find students rarely take hints (79). This is why additional effort has been placed recently in using student feature completion information to help the *instructor* during classroom implementation (22; 20). Dashboards have been developed that use programming trace or compilation information to aid instructors in finding students in need of immediate hel(27; 39). Diana et al's dashboard system is able to use features of code-trace data to display to an instructor which students are lagging behind in the lesson (having fewer

features complete) in order to aid teachers in selecting which students to help or group together to peer-help (20). While useful, this requires that the dashboard be visible to the instructor at all times which may take away from the time spent live coding. We wish to extend these methodologies in order to develop ways to provide post-hoc analysis of a classroom implementation as well as in the future, provide unintrusive immediate support to instructors live coding in K-12 environments.

6.3 Methods

6.3.1 Data Collection

The dataset used in this analysis came from observation and trace data acquired during several implementations of a 4-day computing-integrated science lesson. Nine instructors across three schools led 19 middle-grade classes in a 'Food Webs' unit using a block-based programming environment, Cellular (51) (example code is shown in figure 2). Cellular, like Scratch uses a block-based language to scaffold novices learning to code, and uses an agent-based environment to aid in the modelling of scientific phenomena. Approximately 480 students participated in the integrated activity, however, we are only analyzing data from 287 consenting students with complete participation over the four days.

Table 6.1: Data participants by instructor

Instructor	School	In-Service	Sections	Usable N
Teacher A	School I	N	1	10
Teacher B	School I	Y	1	9
Teacher C	School II	N	2	27
Teacher D	School II	Y	4	72
Teacher E	School II	Y	3	61
Teacher F	School II	N	1	11
Teacher G	School III	Y	1	16
Teacher H	School II	Y	5	62
Teacher J	School III	Y	1	19
Total	I-2, II-15, III-2	6Y,3N	19	287

A breakdown of participating instructors and schools is available in Table 6.1. Demographic data for each of the three schools is provided in Table 6.2 below. Data for each of

the science classes mirrors the overall school demographics.

Table 6.2: Student demographics at each of the studied schools.

School	#	White	Asian	Black	Hispanic	Multi	Female	Low-Income
I	526	31.4%	6.7%	30.8%	24.3%	6.3%	48.3%	47.7%
II	815	39.5%	5.9%	22.5%	25.8%	5.8%	48.1%	51.0%
III	919	22.0%	2.6%	42.3%	28.1%	4.7%	52.2%	65.6%

Instructors were all trained in the Food Webs course materials before teaching their class. In schools I and II, Research instructors (Teacher A, C, and F) led the first class period each morning. Teacher instructors led the remaining class periods. There were a total of 6 researcher-led (or Non In-Service instructors) class sessions and 13 teacher-led (In-Service Instructors) class sessions.

6.3.2 Curriculum

The Food Webs unit is a 4-day activity designed to meet the state-level education standards for 6th-grade science classrooms on the topic of Energy Transfer in a food web. On the first day of the activity, students use pseudocode to complete scientific worksheets on defining terminology and relationships between organisms in a food chain. Students work as a class to abstract the behaviors that an organism in the food chain might exhibit (eat, grow, move, etc.). On day two, teachers lead students in using their previous pseudocode to model the behavior of sunlight as an abiotic factor in the food web and its effect on producers (plants) using the Cellular programming environment. Using a companion worksheet, students manipulate energy values in the code to see its effects. On day three, teachers help students add in code for a primary consumer (bunny), again exploring the propagation of energy in the food chain. Finally, on day four, teachers instruct students on how to create the basic functionality for a secondary consumer (fox) as shown in Figure 6.2. Students then set up their own scientific experiment determining an independent and dependent variable, as well as their hypotheses on what will happen. Students are encouraged to run several trials of their experiments before declaring a definitive conclusion. A final version of the simulation can be seen in Figure 6.1.

The Food Webs curriculum was chosen as we have data from the largest set of unique teachers as well as the largest number of student participants. We choose the 4th day rather

than the other two programming days for our analysis because as part of our design-based implementation research, we implemented different versions of Days 2 and 3 for different teachers to experiment with different instructional scaffolding techniques. However, every teacher implemented the same final activity. As Day 4 is also the final day of the activity, teachers have had multiple class periods of experience instructing programming assignments and Day 4 serves as a culmination of their final abilities.

Figure 6.1: The completed Food Web simulation, with agent-based organisms (Fox, Bunny, Plant) and abiotic sun.

6.3.3 Data Analysis

The primary data used for our analyses is environment trace data from the consenting students working within the assignment. The 19 classroom implementations of the assignment resulted in over 90,000 interactions comprising 287 unique students. As part

of our partnership, instructor guides were developed for teachers in order to learn what was necessary to complete the assignments. These included images of example final code as well as a suggested order of how to add code in their environment to complete the assignment. Using theses instructor guides, we decompose the assignment into five unique, mutually exclusive *features*. These features demonstrate the five additions that teachers led students into creating in the Cellular Environment. These features can be detected using an auto-grader type system and we can determine at any given point in a students programming trace whether or not that feature is present or absent in the code. Figure 6.2 presents a short description of each feature alongside the corresponding code blocks.

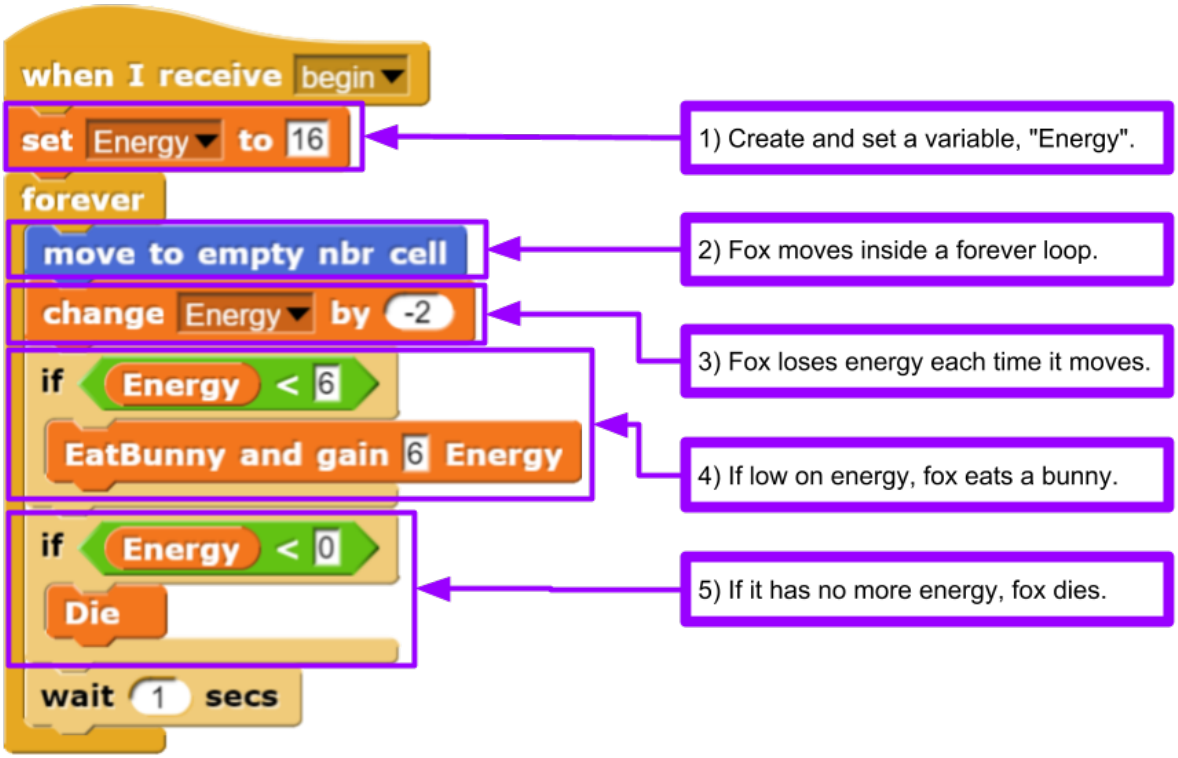


Figure 6.2: Food Webs Day 4 student coding tasks.

Feature 1 tasks students to create a local variable, Energy, that will represent how much energy a Fox agent has in the Food Web. The instructions explicitly state that the variable **must** be local as it represents an individual fox's current energy. Furthermore it **must** be named "Energy" due to a case-sensitive dependency in custom blocks introduced later in the program. Feature 2 tasks students to make the Fox move within a forever loop, as

to simulate a Fox moving around in a biome. The 3rd feature has students decrement the Energy variable inside the forever loop to simulate the Fox losing energy with each move. The 4th feature introduces the first If block - If the energy of the Fox is low (but not 0), the Fox "eats a bunny" in the environment and gains some energy. The final feature, Feature 5, is a second If block where if the Fox's energy is less than 0[1], the Fox will die.

We use code feature detection to determine, for each code state in a student's code-trace, whether or not each feature is present or absent in a student's code. We then use this information to develop the following three analyses:

1. **Feature-Time-Completion** - A histogram that visualizes when students complete a given feature relative to the feature completion time of the instructor.

2. **Cumulative-Feature-Completion** - A stair-step graph indicating when students add more features to their code during the course of the class relative to the instructor.

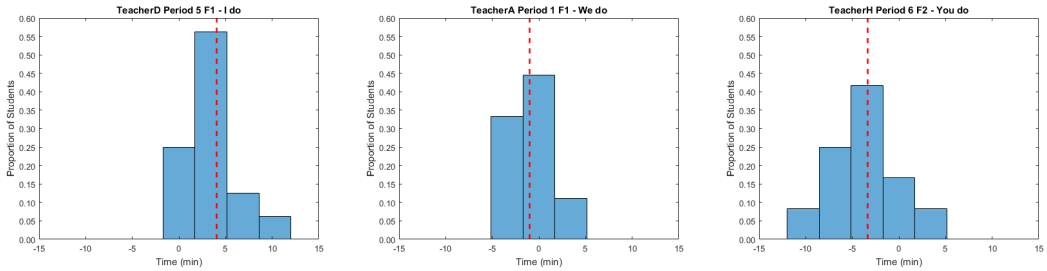
3. **Feature-Path-Graph** - A state-space representation outlining what features students add in what order, showing the varying paths students take to complete the assignment compared to the path taken by the instructor.

6.4 Results

6.4.1 Feature Time Completion

Our first methodology, Feature Time Completion, produces a histogram showing the timing in which students complete a feature relative to when the instructor completes the feature. In the three graphs in Figure 6.3, we present exemplar cases of the "I do, We do, You do" instructional strategies introduced in the Gradual Release of Responsibility (GRR) model (72). This is a common instructional strategy taught to teachers, that progresses a teacher from modeling a step to students (I do) to completing a step together with students (We do) to finally allowing students to try on their own (You do), giving the correct answer after their attempts. The histograms shown show the timing of when the students complete the feature relative to when the teacher completes the feature (minute 0). In subfigure 6.3a we can see evidence of focused instruction, or "I do", where the teacher demonstrates procedures before students attempt to solve problems on their own. The graph is right-skewed and close

[1]We write in our instructions "less than 0" and not "equal to 0" to mitigate issues with the variable decreasing past 0 without registering the if.



(a) TeacherD - I do (b) TeacherA - We do (c) TeacherH - You do

Figure 6.3: Gradual Release of Responsibility as shown by student interaction data

to the center signifying students completing the task directly after the instructor. The second strategy, guided collaborative learning, or “We do", is characterized by students working as a class to complete the feature while being guided along by the instructor. An exemplar of this strategy is shown in graph ‘6.3b’. This has students completing the feature very close (within 5 minutes) of the instructor centering around the instructor’s completion time. This suggests an instructional strategy where students are following along and completing the assignment with the instructor in a “we do" fashion. Finally, the last graphs depict independent learning, or a “You do" strategy where students attempt to solve the problem before reviewing as a group. Graph ‘6.3c’ shows the majority of students completing the feature prior to the instructor demoing, with a subset of students (around 25%) completing the feature after the instructor reviews the solution. The median student (the red-dashed line) is either right, near-center, or left-adjusted for the three cases respectively. Classroom observations confirm that the Teacher scaffolded students by providing students with the blocks they would need to complete the feature (but not giving the full answer or how to combine them). Many students, given the appropriate building blocks, were then able to assemble the answer very quickly (10 minutes before the demonstration).

We turn to Figure 4, which provides an alternative representation of the same information for one class period taught by Teacher B. Here, colored points are shown in order to be able to track students across different features. It should be noted that outlier points are removed for scaling purposes (5 total features are completed by 3 students past the 100 second marker). Teacher B completed 4 features in intended sequential order (F1,F2,F3,F4) which is important to note as this was not always the case (as we will see later). Most students complete the same feature within a minute of when Teacher B completes the feature

(minus the visibile and non-visible outliers).

6.4.2 Cumulative Feature Completion

Our second representation shows how students cumulatively add features to their code relative to each other as well as relative to the instructor. Using the timestamps of when students add their next feature to their code, we are able to chart out in a Stair-step fashion students progressing through the assignment, going from having nothing complete to having more and more of the 5 features added. We are also able to see based off the final placement of each line, how many features students ended the lesson with. In the 3 cases and 4 classrooms described below, each student trace is an individual stair-step function. Instructor feature completion is denoted by the dashed vertical lines, and comparing when teachers and students complete features (in combination with the additional classroom observation data) allows us to understand different instructional patterns and their consequences.

Teacher G

Figure 6.4 describes one teacher doing the Food Web Assignments with their class. After the teacher completes their first feature (with the majority of the class following very closely - within 2 minutes of him completing it) a long 15 minute drought of progress happens. A strand of students make progress independently of the teacher, but the majority are stuck at only completing one feature. From classroom observations, we know that the majority of students had followed the instructor in naming their variable “energy" (though the instructional guide says to name it “Energy" as custom functions expect that name and are case sensitive). A strand of students are able to recognize the error from their student guides and are able to independently make progress. At around 10:00, the teacher fixes the error, and announces it to the class). The wide range of students jumping in progress at varying different times reflects a change in instructional style adopted by the teacher. As it was a simple naming fix that affected every other Feature, fixing it resulted in making progress on multiple features near simultaneously (seen in the tall jumps in each stair step graph).

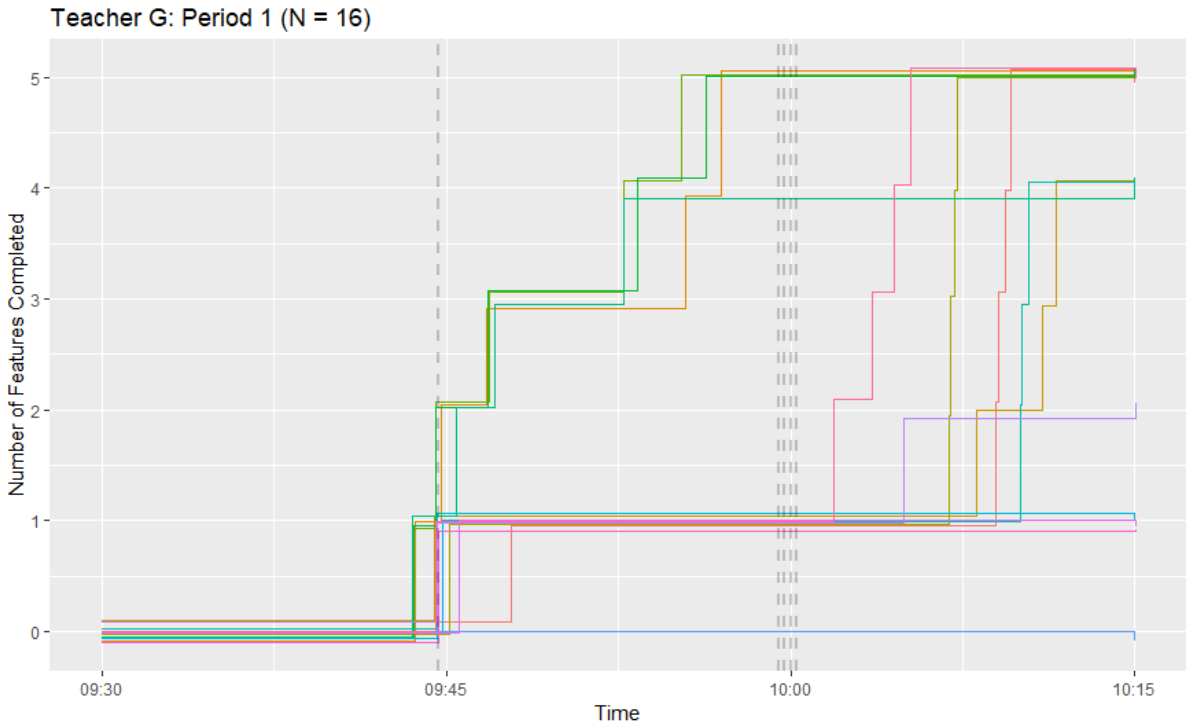


Figure 6.4: The flat-line after Teacher G's first task completion shows students getting caught in the same error that the Teacher gets in while live coding.

Teacher E: Period 4 vs 6

We now present a case study of the same instructor teaching two different periods. As part of the faded scaffolding teaching approach, a research member (Teacher C) taught the first class period (Period 2) where Teacher E observed. Teacher E then taught Period 4. Teacher E requested that Teacher C instruct the next period (Period 5) to observe once more how the lesson should go and then afterwards, the teacher instructed Period 6. From Figure 6.5, we can see the difficulties that Teacher E had with their first attempt teaching. Teacher E completes all of her features for the class after nearly 45 minutes of instruction with sometimes nearly 8 minute gaps between feature additions. Many students are intently following Teacher E adding a feature when she does, but the number of bands that seem to be moving independently before the teacher suggests that the students were at varying paces not at the instructors. This is backed by the classroom observations with researchers noting that some students were ahead of the teacher and the teacher actually asked students for help on what to add next in the sequence. In the next attempt, Figure 6.6, Teacher E was able to complete features much faster, adding 4 of the 5 necessary features in an 8 minute

span of time. This latter period dramatically changes the pace at which the students add the features, and the closeness of bands suggests that students were more intently following the instructors pace.

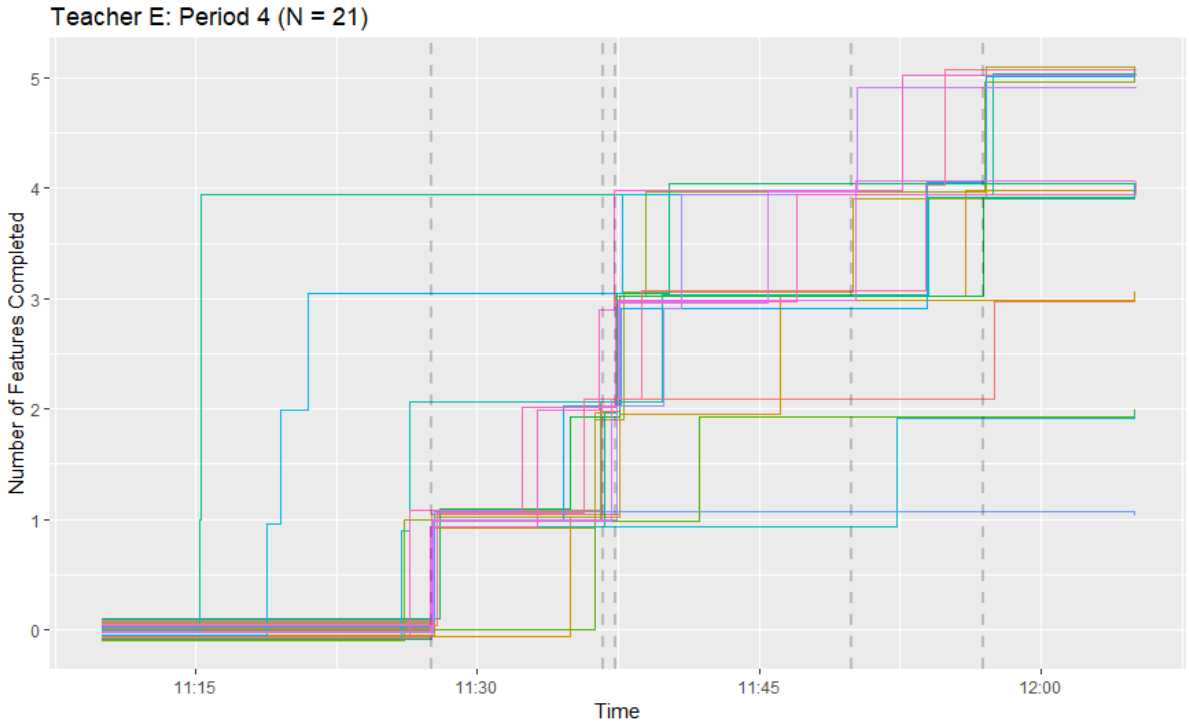


Figure 6.5: Teacher E's first attempt teaching the integrated computing lesson.

There are common patterns that can be seen across both graphs. One is the presence of outlier students who complete things at a dramatically faster pace than the teachers. From the two graphs, it also should be noted that the teacher does not demonstrate the last feature to the class (unlike in her previous teaching period), and many of her students therefore do not get all 5 features in their code. We do acknowledge two outliers, one who jumps 4 features (around 11:15) and the other who more gradual progress, but still faster than the group. Classroom observations confirm that that student was copying code that the previous instructor had left on the projector during the transition period.

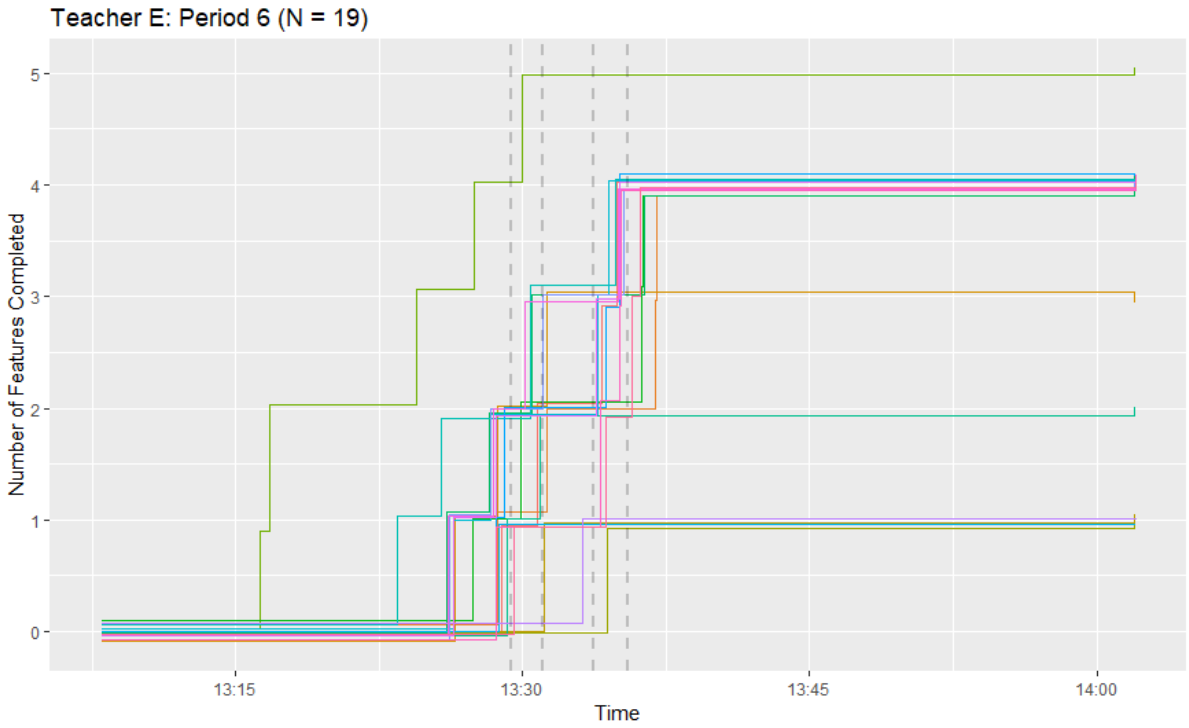


Figure 6.6: Teacher E's second attempt teaching the integrated computing lesson.

Teacher F

We now highlight a period taught by Teacher F, a research instructor with a large amount of prior informal computer science teaching experience. Teacher F elected to lead the assignment *without* demonstrating to the students what the code should look like. Instead, she instructed students in what features they should be adding next, facilitating, and helping students by walking around the room. After each feature, she would then recap verbally with the whole class to review what they needed to have done in order to add the feature before moving on to the next one. We can see that student progress is very independent of one another, with the timing of each successive feature addition (i.e. when students go from having 1 to 2 features) varying widely. While students add their first feature within 5 minutes of each other, Students add their 2nd, 3rd, and 4th feature within 10 minutes of each other. This reflects the choice of instruction style, with Teacher F walking around the room independently helping students. In contrast to the end of Teacher G's lecture, progress in Teacher F's class is limited, usually, to *only* adding the next feature and nothing else. This reflects the teaching style of only giving students limited information about what the next tasks are, focusing the instruction on one feature at a time. Teacher F's students

that complete their fifth feature (adding death), do it near simultaneously, reflecting the change in instruction style recorded where the teacher instructs the class in completing the feature together.

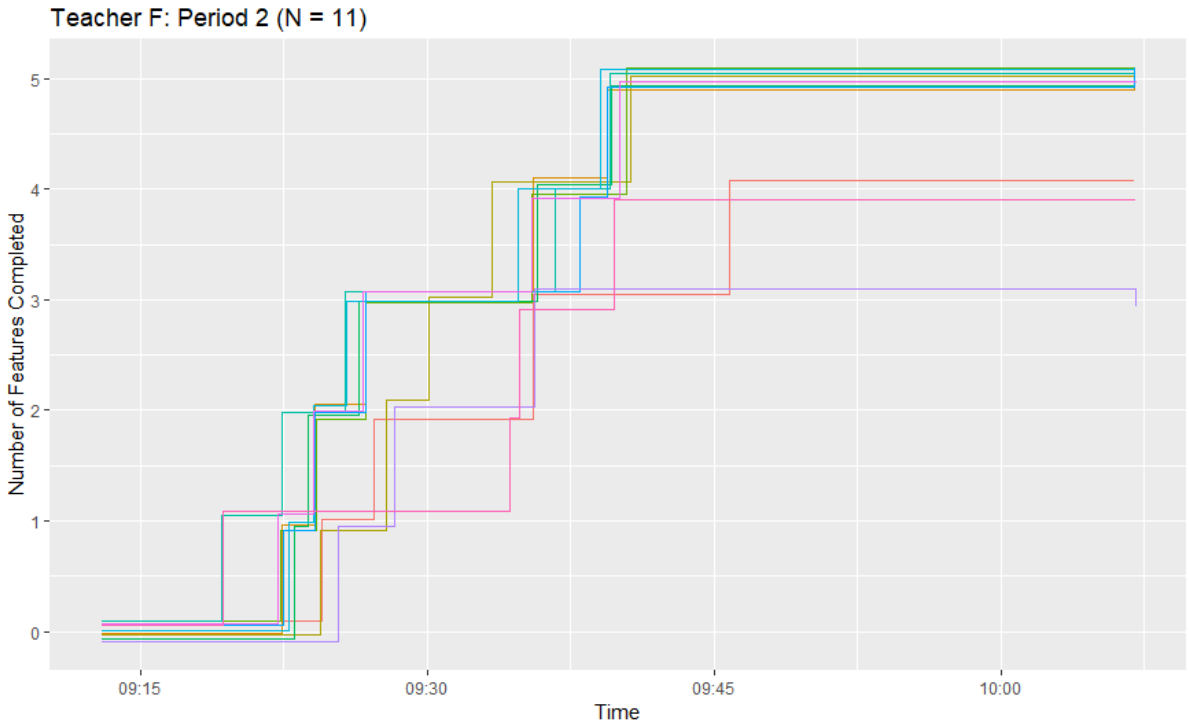


Figure 6.7: Teacher F leads class instruction without demoing code steps.

6.4.3 Feature-Path-Graph

In our final visualization, we remove the temporal information of the last two graph-types and focus solely on Feature-paths, the order in which students add features to their code. In Figure 6.8, we highlight three Feature State representations of three classes taught by three different instructors (Teacher C, B, and H). Each state in the graph represents a unique set of features complete by a student. For example, following the path of Teacher B (the states marked 'TB') in the second graph, the states progress from the one with 0 features to the one having only feature 1, to having both feature 1 and 2, to having features 1,2, and 3, and ending having features 1 through 4. Conversely, the state spaces for Teachers H and C show different sequences of feature additions (for C in the order 2,1,3,4,5 and for H in

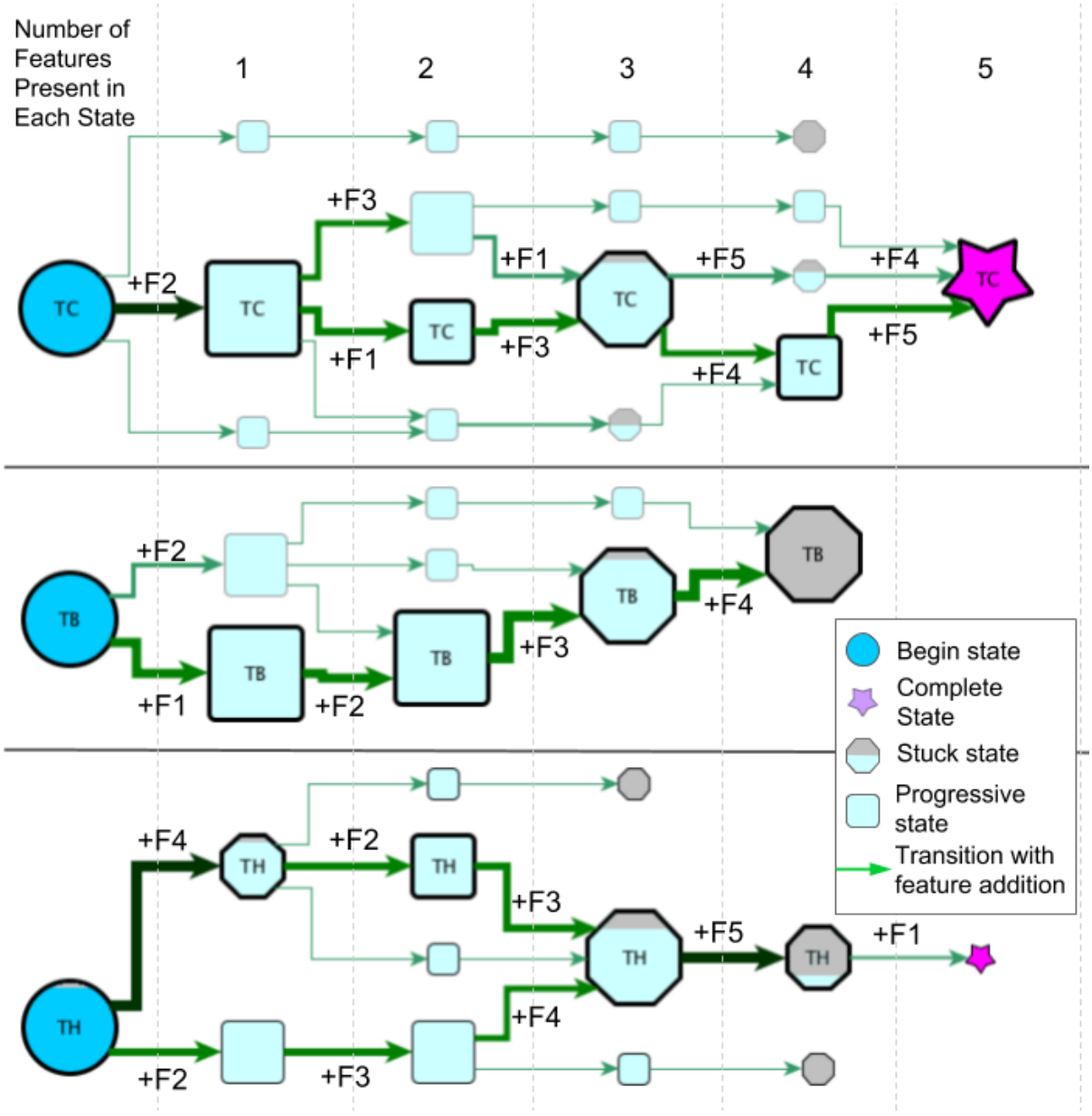


Figure 6.8: Feature-Path Graphs for Students (and teachers) within sample classes for Teachers C, B, and H.

the order 4,2,3,5). Common state transitions, (those where a large proportion of students transitioned) are marked with the Feature that was added during the transition. The size of each state represents the proportion of students in the class that visited that state. As some students do not finish the assignment fully (getting “stuck" at certain points) we represent these as “stuck-states" in which a grayed proportion of the state represents how

many students who visited that state did not progress. We also denote the beginning and end state with a special circle state and star-shaped state respectively. This is similar to the representation introduced in Rui et al (100).

The three feature graphs show different instructional strategies at play. The first, TC, show most students following the path of the teacher, adding features as they do. However, it also includes many individual paths taken by students not necessarily following the teachers instruction sequence explicitly, but following the general direction, denoted by the number of 'confluence' points or rejoining back into the Teachers state of various different paths. This classroom had the largest proportion of students actually finish the assignment compared to the other two classrooms. Teacher B's more narrow state-space with the largest states being those that the teacher traversed suggest that most students were following along with the instructors code additions (corroborated by Figure 4 as well as observations), adding in the order that they did. While Teacher H had a subset of students add features in the order they did as well as a subset chosing a separate path, these paths seemed to converge in state 2,3,4 then leading to the addition of Feature 5. While Teacher H and C both show variance in student feature paths, it should be noted that both Teacher H and B did *not* add all 5 features to their code (Teacher H did not add Feature 1, and Teacher B did not add feature 5). As such, all students in Teacher B's classroom and all but one student in Teacher H's class failed to complete all features of the assignment highlighting scenarios in which intent direct mimicry did not fully benefit students.

6.5 Discussion

Our three methodologies provide different lenses for analyzing the question, *"How do teachers influence student programming behavior?"* Our first method, showing the relative timing, focuses more on the *When* and *How* aspects. Instruction and influence goes beyond students copying or echoing teacher focused direction. As demonstrated in Figure 6.3, while there are indeed examples of "I do" sequences, where a teacher demonstrates a task and then students replicate it, we also find examples of "We do" and "You do" sequences rounding out Pearson's Gradual Release of Responsibility model. Our observations triangulate the "We do" pattern to actually involve the class working together with the teacher to solve a problem, as opposed to Pearson's more traditional interpretation of the teacher working with just a small group of students.

Another notable observation is derived from the successful "I do" and "We do" strategies;

these are from initial task features in the assignment, i.e. the first features that students work on together in the assignment with the instructor. The successful "You do" exemplar completions are from features *later* in the assignment (the last features students work on). This would corroborate teaching progression strategies pushed for in the GRR model, transitioning students from "I do" to "We do" to "You do" feature completion.

Our second methodology, the cumulative feature stair-step graph, focuses on the *When* and *Who* lenses. By separating out the individual traces of each student into their own function, we can see how a student temporally progresses through the assignment in relation both to the teacher and the other students in the class. From the student perspective, we see many examples of students who work in lock-step with the instructor (or with each other), adding features as they do. But we also see students who trail and lag behind as well as students who rapidly add features before the instructor has even introduced them. This differentiation is to be expected in a core classroom where students have varying access to computing outside of school and therefore different experiences with block-based programming environments. The ability to identify students who are not following along with the instruction, either as the pace is too fast or too slow, can aid instructors in identifying when to create more extension activities to high achievers or when to give more support to students in need.

Our final methodology, state-space, focuses more on the *What* of the question. As the state-space focuses on *which* of the features are being added in what order, it can grant information about the way teachers set up the instruction (i.e. how they choose to pace the students through the task). Very narrow state-space graphs like the ones shown for Teacher B can demonstrate a classroom that is explicitly following along with the instructions as sequenced by the teacher (even if all of them might be working on the tasks at different times). Conversely, wider state-spaces like those shown for Teachers C and H can be more indicative of classrooms where students had more agency over the order in which they choose tasks. We make no argument that more explicit instruction, or a higher path "follow" rate leads to either more or less students completing the assignment. Teacher B and Teacher H had a high proportion of students following along with their path, however, this might have left students unable to complete the assignment independently as neither teacher progressed into adding the fifth and final feature.

Looking through the teacher perspective, we can identify patterns indicative of increased teacher discomfort. As mentioned in Teacher G's case study, as he erred in making the variable, he and consequently his following students, all flat-lined in their coding progress. This highlighted the loss of the teacher as the "keeper" of knowledge and exposed the

vulnerability that they had to problem solve along with the students, thus incurring both pedagogical and emotional discomfort. Similar signs of discomfort are visible in Teacher E's case study where there are long pauses between teacher actions (and student reactions) showing that Teacher E was unfamiliar with the content leading to discomfort. These graphs contrast Teacher F's, the experienced computing instructor, which had students clustered on the same features, moving at a steady interval. Teacher F did not feel the need to explicitly direct students with live coding, but instead was able to guide their exploration through facilitation. Conversely, we also discovered evidence of *change* in instructional practice, both within a class period and between them. In many cases, these changes were for the positive, leading to more students following along and getting through the material. As the anxieties that novice teachers tend to have tend to focus on not doing things 'correctly', showing evidence of positive change between implementations could potentially inspire confidence in novice teachers that they are capable of instructing in this new context.

The drawn conclusions that we outline in our case studies are supported in part by our classroom observations, corroborating what we find within the data-driven visualizations. Because of the nuanced and varied ways in which classroom behavior can manifest within the data, it is important that we frame these results with our understanding of what happened within the classroom. As such, we understand the reliance our methodology has on observational data, and make no advocacy that our method acts as a complete replacement for it. In the same way that no single one of the three graphs tell the full set of information of a classroom, we believe that these quantitative analysis methods act as an additional tool to act alongside qualitative data collection methods, ones that scale much better though than the observational-based ones. We also understand that other limitations present themselves with aspects of our methodology such as only focusing on one lesson with only 5 possible features. In addition, only having the set of consenting students out of the entire classroom trace data could have created a sampling bias that might alter any and all of our graphs and therefore our analysis. We intend to try and offset these effects by replicating this with the multitude of lessons that are being developed by our teacher partners in the upcoming academic year, and we encourage other scholars to do so as well. Our methodology described is agnostic to the assignment, the features, and how they are tested for (though we acknowledge that the feature state points are only as good as the tests you run on the code). Further, many logging systems exist for more traditional text-based programming environments like BlueJ through Black Box (6) and frameworks for logging and analyzing programming trace data have become more standardized through systems like ProgSnap2(78) meaning it is becoming increasingly easier for our methodology

to be **practically implemented and adopted**. We also believe that this methodology can be adapted to any direct instruction in intelligent environments that are procedural and loggable such as going through a worked math example together in an online system.

6.6 Future Work

Our initial results provide a foundation for further analysis into instructional methods for programming in K-12 contexts. As a broader range of K-12 classrooms move towards integrated programming assignments, it's important to investigate how students within these contexts learn and complete tasks. Further, as teachers with lower confidence tend to gravitate towards very instructional methods of teaching, they serve as the primary source of information for students on how to work within these programming environments, meaning it is important to investigate best practices for teachers to effectively communicate, scaffold, and support students in learning environments where students learn as a class as well as independently. For instance, as more schools begin to adopt pair programming models of instruction, it would also be worth investigating these collaborative interaction effects as well, doing similar modeling between two partners rather than student and teacher.

This current work has **immediate practical** applications for our context in teacher training. As these graphs, charts, and other figures can be generated rather efficiently (in a few minutes after an implementation), these can serve as additional means of going over with novice teachers how the experience went and how the classroom acted. This additional data stream can help teachers inform their teaching practice and could result in them changing the pace or tasks or the sequence they go through them for their next class of kids. Further, through additional processing, we can translate many of these graphs into *animations* that can give a better 'live feel' of how the instruction played out in real time. We believe though that in the near future, these types of instructor-student feature comparisons can be implemented within real-time dashboards designed to be used during live coding like Improv (9). Setting times for feature completion targets for an instructor can keep them on pace in tight classroom periods, and windows for instructor-student feature completion can be used to determine whether or not the class is following along intently. Often, the novice instructor participants looked to the Research Observer for a simple 'Thumbs Up or Thumbs Down' measure to see whether or not they were doing things correctly and kids were following along. If a simple notification such as the one described were implemented into a dashboard system, We can see near-term **practical**

and scholastic benefits for both novice and expert teachers using such a system while live coding. We imagine **improvements must be made** to our data analysis to create intelligent systems to determine when notifications for pacing and help should be given to instructors. Thankfully, the data problem could be mitigated through individual teacher help given the sheer popularity of live coding as an instructional practice around the world. We begin the process by sharing our code and data sets for this implementation, hoping to inspire others to examine how we as instructors can shape our classrooms and the students we teach.

CHAPTER

7

EPIDEMIC CHOICES

7.1 Introduction and Motivation

Study 2 demonstrated how a 'low floor, high ceiling environment' can be created for students by presenting them with a set of choices for them to implement. Teachers facilitated these lessons by walking around and helping students with their individual choices. As all choices used effectively the same code (save for one block that was specific to the type of animal that was chosen), the set of choices had neither variation nor difficulty differentiation. In this study, I expand on this design for "Choices" to investigate how a classroom behaves when choices vary more widely, e.g. in terms of perceived difficulty, sprite use, and block use. A core tenant of the "Choice" element in this Use Modify Create model is to have students pursue Choice Extensions that are meaningful to them. However, students may choose an extension that is outside of their current level of ability - either too easy or too difficult. To explore this possibility, this study will investigate the relationship between student choice and extension difficulty in the Create/Choose phase for the Epidemics assignment. First, students are asked which of the extensions they would like to pursue, that is, which extensions are motivating and meaningful to them. Then, students attempt their

chosen extensions or perhaps choose others during class. I will use log data and student preferences to determine how their interests, and difficulty, relate to which extensions students attempted and completed. Finally, using classroom observation and follow up interviews, we will also determine whether this formulation of Choices is still manageable for teachers to implement within their courses. This will comprise Study 5 which is detailed below. Our Dissertation Research Question is once again:

RQ3. How can a pedagogical framework allow for differentiation of tasks by interest and skill level to meet the needs of students, while being manageable for teachers to implement?

Our research questions for Study 5 are:

RQ1: Do students attempt or implement extensions other than their original intent and can this deviation be explained by difficulty of the choices? RQ2: Do students who implement their desired extensions report higher levels of ownership than students who do not? RQ3: How do teachers evaluate the "Use-Modify-Choose" sequencing with Choice extensions?

7.2 Epidemics Assignment

The Epidemics Assignment detailed in Study 3 was 5 days long and was designed and sequenced prior to the work done in Study 1, where we demonstrated that a Use-Modify-Create sequencing was preferable. For this study, we re-designed the Epidemics Curricula to align with the UMC framework. Students could only have three days to work on it, so we decided to have the following setup for the coding portion of the lesson:

Use: Day 1 had students explore a version of the epidemics curriculum with basic functionality - students could click on agents to make them sick, had code such that if a sick agent was near a healthy one it would become sick, and had a variable implemented that counted that number of infected over time. Students' main goals for the day were to inspect the code and try to understand how it worked through creating different run time conditions. This could include setups such as only one agent being sick, half being sick, all agents being very spread out, and so on. The goal was to get students familiar with the Cellular Environment and see how these run time conditions affected the rate of spread of disease. For example, having the agents spread out would most likely result in a slower rate of spread than a situation in which they were all close together. Again, students were encouraged to use the built-in graphing feature in Cellular to chart the number of infected over time. In terms of the set of code features available to them compared to the

prior version, the "Use" day had all the code features up to Day 3 of the prior assignment (having a variable for tracking the number of sick people, having a way for people to become infected, etc). We decided to give this amount of code as a starting point for Use, as it nicely bridges to the code they would complete in Modify (below), and the space of potential choices for their "Create" portion.

Modify: Day 2 tasked students to write or edit code in the environment for the first time. Students' simulations were extended with three new concepts: incubation period (a local variable that would track how long an agent had been sick), immunity (a new state that agents would transfer to after being sick for a certain amount of time), and hospitals (an additional sprite that, if agents were nearby and sick, could speed up their recovery). Hospitals were a popular extension choice in the previous implementation, and immunity and incubation afforded the ability for other Choice extensions on the third day that will be discussed below. Teachers were once again told to directly instruct the class through implementing the changes necessary for the code to work through Live Coding. The task sequence is included within the Appendix.

Create: Day 3 had students select from a list of possible extensions and implement at least one to extend their simulation. Teachers would walk around and facilitate the classroom, helping any individual student on their selected extension choice. Students were given a worksheet listing each of the possible extensions. Students were explained that the front side of the worksheet contained the "Easier" extensions , while the back side of the sheet contained the "Harder" extensions. On the worksheet, each extension had an explanation of what the extension should do, and had the set of recommended code blocks necessary to complete the feature. Extensions were given the label of either easy or difficult by the researchers based on the number of code edits necessary to implement the extension into the simulation (with the easier set requiring fewer edits). The extension choices C1-C6 are listed below:

C1. Chance of Getting Sick [Easier] - Students need to change the logic of how agents become sick by requiring not only for the healthy agent to be next to a sick agent, but also for a random number generator to output a value greater than some threshold. C2. Death [Easier] - Students modified the code developed in Day 2 that would move an agent to the immunity state if they were infected for a certain amount of time. Instead of moving to immunity, agents would die and be removed from the simulation adding to a counter of how many agents had died that simulation. C3. Moving Toward Hospital [Easier]- Students would modify the logic for how agents move such that, if they are healthy, they will move around randomly as normal. Otherwise, if they are sick, they will move towards the nearest

hospital. C4. Vaccine Preventing Illness [Harder] - This extension had students work on extending the agent code such that if it was healthy and was by a hospital, it would be able to get a treatment in the form of a "vaccine" that would move it to the immune state. C5. Limited Number of Treatments [Harder] - Students extended the logic of the hospitals such that there were only a limited number of treatments available to hand to sick individuals. This involved creating and keeping track of a variable for number of treatments and decrementing each time it was used. If a hospital ran out of treatments, sick individuals could no longer seek treatment from the hospital. C6. Mobile Hospital Services [Harder] - An additional ambulance costume was available in the simulation for the hospital sprite. For this extension, Students had to create the logic that would toggle a hospital sprite between the hospital and ambulance costumes if it was clicked, as well as the logic for moving the hospital sprite around the stage if it was in the ambulance costume.

The extension set was chosen such that there would be no dependencies between the extensions. The goal was to also create a diverse set of interactions and extensions that would hopefully have at least one extension choice that would motivate each student. Teachers were also given a cheat sheet of all of the extensions to help them with facilitating the Choice Day (included in the Appendix).

7.3 Population, Experimental Setup, and Data Collection

We once again recruited teachers from the local magnet middle school to participate in this study. Two teachers had previous experience leading Infused Computing Activities, specifically the prior version of the Epidemics Curriculum detailed in Study 3, and one teacher had no experience with leading Epidemics. Each teacher taught multiple class periods in a one week period in February 2020 (5 for the two experienced teachers, 3 for the novice) with a total of over 200 students participating fully in all 3 days. For the final dataset, we included only students who consented to the activity, had a program trace on Day 2 and Day 3 and had submitted Exit tickets on both Day 2 and Day 3 resulting in 96 students. We collected the following sets of data to evaluate the activity and study:

Exit Tickets: At the end of each day, students were given an exit ticket with a set of questions. On Day 2, we gave students the set of possible extensions and asked them to select the ones they were interested in trying to implement on Day 3. The question shown to them is below. Notice how the difficulty labels were not attached to the choices as they will be on the worksheet for Day 3. We wanted to see what choices students were interested

in without having the potential for a labeling of "easy" or "hard" affecting their choice.

Students had the entirety of Day 3 to implement as many of the choices they wanted into their code. As students had 50 minutes for the Day 3 assignment, and only had one prior day of coding, we set up the choices such that all students could at least complete one within the timeframe, and we expected most students to complete at least two. Students also were free to not choose extensions they previously chose at the end of Day 2. At the end of the activity, students were asked to fill out a survey with the following questions:

Difficulty: Please use the following scale to rate how difficult or easy the lesson was today. (Likert 1 to 5, Very Easy to Very Difficult)

Ownership: To what extent do you agree with the following statement: "The code I ended the lesson with is my own creation." (Likert 1 to 7, Strongly Agree to Strongly Disagree).

Observations: Our observations once again focused on the classroom experience of each of the three days. Specifically, we observed for the purpose of being able to answer these questions: how engaged were students in the tasks?, which tasks were difficult for students?, what teaching styles and strategies were employed?, and how was the classroom experience managed and carried out? Observers also noted how teachers engaged with students on Day 3 to see if they felt comfortable facilitating the lesson.

Teacher Interviews: Teachers were asked to participate in a follow-up interview about the classroom experience. Teachers were asked about their experience on teaching the activity overall and what changes, if any, they would make to the curricula.

Code Traces: All code snapshots within the Cellular Environment were logged and recorded. These code traces were used to understand what choices students actually made within the environment, and the process they took to implement these choices. For each of the code choices, a set of features were created such that having all features present within a student's code state represented them having completed the Choice. When this occurred in the code trace for a student, that specific timestamp was marked as the first occurrence of that Choice being present in the code and the student was marked as having completed the choice. Students who completed only a subset of the features necessary for having completed the choice were marked as "attempted" for that specific choice.

Table 7.1: Student Choices by of Students who attempted, desired, and fully completed the extension.

Choice	Desired	Attempted	Completed
Chance Getting Infected	39	50	37
Moving Toward Hospital	39	50	11
Death	81	71	14
Easier	**159**	**171**	**62**
Vaccines	60	44	13
Limited # of Treatments	51	25	5
Ambulances	58	30	2
Harder	**169**	**99**	**20**

7.4 Results

7.4.1 Choices

The results for the Code Trace choices are shown in the table below. In comparing Students desired choices to the ones they attempted, the number of easy choices compared to harder choices that were desired were around a 50/50 split (159 vs 169 respectively). However, the number that were attempted for each choice category leaned more heavily towards the easier choices: (171 vs 99). This skew is even more pronounced in the number that students actually completed, with more than 3 times more Easier Choices completed by students (62) vs Harder Choices (20). Already this suggests that though there was a more even spread of desired choice extensions between harder and easier in the beginning, students when presented with difficulty labels began to skew in their attempts and actual completed activities towards easier choices.

As we had the set of choices attempted and desired for each student, I looked at the average rating of the set on a scale that ranged from only including easier choices (-1) to only including harder choices (1). As it was an average of the set, whether or not a student only desired one easy choice or all three easy choices would not matter, and still result in a -1. The average score for the student's desired set of choices was -0.01, very close to 0 or a score representing an equal amount of easy and hard choices. However, the average score for the set attempted by students was much lower at -.338. Matching individual scores for students (attempted vs desired) a Wilcoxon Signed Rank test finds this difference significant at $p < .001$.

We find more evidence for this skew towards easier choices in diving deeper into the

data. The number of students who only attempted Easy choices was 30 compared to 7 for only attempting harder choices, and the number of students who attempted more easy than hard choices was 63 compared to only 14 who attempted more harder choices. Observations of classrooms further support a general lean towards easier choices. Observers found that teachers often put some explicit pressure classroom-wise that might have contributed to students choosing easier over harder choices. This would include telling students that if they were a novice or if they felt uncomfortable coding to choose one of the easier choices.

What is also noticeable from the chart is the very low completion rates for each of the choices. On average, the easier choices were completed at a higher rate than harder choices (compared to ¼), but both represent very low actual completion rates.The highest completion rate for a choice, at nearly ¾ for chance of infected, stands as an outlier to the rest of the chart and choices. The picture begins to become more clear as we look at the number of choices desired, attempted and completed. More than half of all students (53/96) desired between 1 - 3 extensions and around the same number attempted between 1-3 extensions (58/96). However, the majority of students are not completing any extension fully, with 52 completing none. Students, while attempting a number of choices within the environment, are more often than not, fully completing them.

A clearer picture emerges as we step through the traces of students as they work on their code. While there are instances in which students continue to work on a choice and just can't get it before time elapses for the period, there are a lot more instances of students making partial work on the choice, and then just moving on to another choice. This corroborates the finding of a much larger number of attempts than number of actual completions. There could be many reasons why students move on to another feature prematurely including being bored of the current task (or conversely, excited by the idea of a different task), but another explanation could be found in students' perception of whether or not they completed the task. This would explain common patterns of partial-completion across multiple students. For example, many students did not fully complete death, with a set of students either using the "change sprite" code block and switching to the Dead costume (and thus still having the agent move around the simulation) and a separate set of students neglecting to include elements of the code necessary for tracking the number of dead (placing the "Set of Dead to 0" block at the beginning or not changing the number of dead when someone dies).

What connects many of these absences or partial completes is there is usually a demonstrable effect on the visual runtime of the code after the partial feature is added which might be enough to signal to the students that they are done. If a student sees a hospital

moving around or dead sprites being removed from the simulation, it might be enough for them to believe they have completed the Choice and can move on. Students also might either not know they have not completed or not care to add in ancillary elements of the Choice such as tracking the number of dead and instead feel ready to move on.

In comparing students perceived difficulty between Day 2 and Day 3, we find that students express day 3 as more difficult. The difference between Day 2 (2.29) vs (2.56) is significant using a Wilcoxon Signed Rank Test (p= .02). We also find that students' answer to "The code I ended the lesson with is my own creation" moves towards stronger agreement between Day 2 (3.82) and Day 3 (3.40) with 1 being Strongly agree and 7 being strongly disagree. Again, a Wilcoxon Signed Rank Test finds this difference significant (p = .014) In dividing students up roughly in half by those who attempted 0-2 activities (42) and those who attempted 3-6 activities (54) we find that there is a significant difference between the difficulty perception of the 0-2 group (2.26) and the 3-6 group (2.80) which a mann whitney u finds significant (p= .025). However, using the same categories, we find no difference in the response to the expression question between the 0-2 group (3.33) and 3-6 group (3.44) (p-value is .73).

We then took the set of the Desired and Attempted choices and computed a difficulty score. This score would be -1 if students chose only easier choices and 1 if they only chose harder choices. It did not matter how many of the choices were chosen (i.e. 1 easier choice vs all easier choices), only the relative difficulty of the set of choices. The average score for students' difficulty of their desired choices was -.01, incredibly close to the value of having an even number of easy and hard choices (1). However, the set of attempted choices skewed more towards easy, with a value of -.338. Using a Wilcoxon Signed Rank Test, this difference is found to be significant at $p < .0001$. We can subdivide students further by specific type in order to get a clearer picture of this spread. For example, the number of students who only attempted easy choices was 30 compared to the 7 who only did harder choices. The number of students who attempted more easy than hard choices was 63 as compared to 14 who did more harder (and 19 who did the same).

To understand the relationship between students desired and attempted choices, we calculated two proportions using the two sets for each student, the Set of Choices they initially Desired (D) and the set of Choices they actually Attempted (A). From this, we can define the proportion of choices of those they attempted which were ones they initially desired (PA,D) and the proportion of choices of those they desired that they actually attempted (PD,A). For clarity, the formulas for these are shown below.

For PA,D 5 students didn't attempt anything at all, 44 did a mix of things they desired

and did not desire to do and 47 did only things they desired. The students who did half or fewer of their desires had an average score of 3.51 for the Code ownership question as opposed to the average of 3.36 for those who attempted more than half of what they desired. A mann U whitney finds this difference significant (p=.728). Similarly, for PD,A, comparing the expression score between those who did not attempt most of their choices (n=50, avg = 3.22) vs those who attempted most (n = 45, avg = 3.51) there is no significant difference in a Mann Whitney U test (p = .34). A oneway ANCOVA was conducted to determine the effect of the different groups on student's Ownership score on Day 3 controlling for their ownership score on Day 2 and found it non-significant for both groupings of PA,D (p=.68) and (PD,A).

7.4.2 Classroom Observations

In line with past research, student engagement with the assignment varied from class period to class period, often affected by the level of teacher engagement with the learning process [Catete2018]. All teachers, even the novice Teacher C, were observed walking around the room able to debug and help students work on their choices, but their confidence depended on their prior experience. More often, Teacher C with much less experience than the other two would call over other researchers to help debug, but it is also the case that Teacher C had fewer periods to attend to and learn the material than Teachers A and B (3 vs 5 each). Observations noted markedly more off-task behavior during the Create Day than the prior Modify Day in which students were led by the instructor. Observations also noted more students off-task within Teacher C's room than in Teacher A or B. Teachers adopted a number of actions in order to manage and facilitate their room during the activity. Students in later periods in Teacher A's classrooms grouped into pairs to work on choices. Teachers also used other measures to scaffold students such as announcing in the beginning that newer students to coding should start with the easier choices. Some class periods also began with Teachers demoing and demonstrating how to do the "Chance Infected" Choice which accounts for the large number of students who successfully completed this extension. Observers marked that students engaged in "trial-and-error" type programming, running the code after a set of code changes to see if the Choice was implemented correctly. Most students observed, if they engaged in the programming activity at all, would add features up until the end of the class period, with fewer stopping after only implementing 1 or 2 features. Examples of students working on independent ideas such as drawing in their own sprites or creating their own code features (e.g. a time delay for when a vaccine becomes available)

were also observed, backing up these code traces that also saw this type of activity.

7.4.3 Teacher Interviews

Teacher A noted that the Create day was the second best (behind the unplugged) in terms of student engagement: "I feel this year's create day was the best out of all the years...They had a lot of activities to do and I like how they could pick and choose. And then also they just seemed more comfortable than the students have in the past with being able to create on their own. So I think that was a really successful day." She recognized that death was one of the more favorite choices among students, but added that she liked and encouraged students to do vaccines as well as that was important to their Science Unit. Teacher B also liked the extensions and choices especially related to death and vaccines/limited of treatments. Teacher B believed that the create day the students were the most engaged "... Because by then they'd already used it, they already modified it, they saw things they could do to it and then they had these six options of, 'What do you want to do?' And of course, they all want to make all the people die, so that was so engaging and wonderful for them." Teacher B also believed that the scaffolding through the suggested blocks on the worksheet were helpful: "[The students] said, 'Okay, if I use these blocks, I put them somewhere, this will happen'. And so I think that the students felt most confident by then." Teacher B also detailed that "... the create was easy because it's kind of open-ended. But the modify, I remember I was always putting the if blocks inside the wrong other if block it should have been under that if block. And so, I myself was struggling with remembering exactly where to place certain things."

Teacher C was new to this curriculum and expressed difficulties with doing each of the days, including the create. "... I definitely did not feel confident in my skills on helping students, like what they would run into an issue with the code, I didn't know the program well enough to be able to offer them assistance without telling them what to do, because there was like one big issue with where stuff was located." She expressed that she relied much more on the researchers who acted as helpers but felt she could lead the activities next year. She remarked that the choices were all good and connected to the science concepts especially "...the limited treatments, because that is a very common thing now. But I really enjoyed that because it kind of ... I thought it really drove the point home that this stuff doesn't just make itself...I felt that was a really good learning point for them, to see that even in the game, you have to simulate real life situations and circumstances."

7.5 Discussion

We now synthesize each source of data from the results in order to address our three research questions which are repeated below:

RQ1: Do students attempt or implement extensions other than their original intent and can this deviation be explained by difficulty of the choices? RQ2: Do students who implement their desired extensions report higher levels of ownership than students who do not? RQ3: How do teachers evaluate the "Use-Modify-Choose" sequencing with Choice extensions?

Research Question 1: From the distribution of choices both attempted and desired as well as in our method of comparing the relative difficulty of the two sets, there seems to be evidence for the idea that students both implement extensions other than their original intent and that these attempted extensions are significantly easier than those that were originally desired. There are a number of potential reasons as to why students might have engaged in this behavior. First, there were documented examples of teachers suggesting to students that they should, if they are a novice to programming, start on the easier side. If a students sense of competency in terms of working within the environment for the past two days was low, they would probably have heeded the teacher's advice and remained on the easier choices. The fact that two very common choices, Chance of Infected and Death, were both on the easier side also contributed to the Easier choices being more attempted than harder. Death was by far the most popular choice in terms of number of students wanting to implement it and number of students attempting to implement it, and Chance of Infected, though not initially popular, was highly attempted and completed by students as that was the choice adopted by some teachers as a demonstration. While there is evidence to suggest students skewed towards easier choices, It is harder however, to characterize why students engaged in such behavior. Work avoidance could be one explanation, but the number of students who engaged in adding feature after feature until the end of the period frames the behavior more as work "delay" (students doing the easier choices first rather than doing only the easier choices).

Research Question 2: Analysis using the constructed values of the proportion of Attempted Choices that were Initially Desired and the Proportion of Desired Choices that were Attempted both did not evidence to suggest that students who completed more of their desired choices reported higher ownership. There are a number of reasons as to why this relationship was not found in this study, but still possibly could exist. First and foremost, the fact that students answered the question on Day 2 about their desired choices and only

implemented choices on Day 3 meant there was an entire day between expressing their opinions and acting on them. It is possible that students do not remember their expressed preferences or even have changed their preferences within that time frame, especially in response to seeing how these choices actually work within the environment. It is also possible that purely selecting the set of choices they are interested in doing did not provide sufficient enough information about relative preference. Further studies could expand upon this methodology in order to capture this information through asking students to do things like rank the top 3 they would like to do, for example. While there was no evidence to suggest that getting to implement initially desired choices had an effect on student perceived ownership, there was evidence that students increased their sense of ownership over the code between Day 2 and Day 3. This, in conjunction with the previous result, suggests that the most important effect may come from students being able to implement choices at all rather than the effect coming from acting on any specific choice.

Research Question 3: Teacher expression of their comfort with teaching the lesson fell mostly along the lines of prior experience, with the two teachers having done a similar version of this feeling more comfortable than the teacher who expressed discomfort doing this their first time. Still, Teacher C expressed they felt they would be able to teach this assignment next year. It is important to recognize the necessity of materials that helped teachers implement these assignments including teacher and student guides as well as the cheat sheet that was given to help facilitate choices. These materials in conjunction with teacher prior ability seemed to lay the foundation for successful initiatives which has been the case throughout the studies within this dissertation. Of the two teachers with prior experience running the Epidemics assignment, both expressed preference for and seemed comfortable with facilitating the Choices. Reasons Teachers gave for supporting the Choice framework included allowing students to differentiate, be independent, and explore scientific concepts in more detail. These preferences mirror the original ones expressed in Study 2 with re-implementing the Food Webs assignment. It will be important in research focused on teachers developing their own Use Modify Create assignments to determine to what extent they feel comfortable not only leading such "Choice" activities but also creating them. Teachers recognized that the different choices offered different levels of challenges for students, but Teachers being able to make these challenges at different levels might prove a challenge for future work.

One finding that was not captured completely within any one research question was the incredibly low rate of completion for Choices. The low completion rate for choices seems to suggest that stronger scaffolding is needed for students when they begin the transition

working independently after previously working through direct instruction from the teacher. Though the suggested blocks seemed to be sufficient scaffolding for some students (or this level of scaffolding plus feedback from the instructor as they walked around), it did not seem to be sufficient for the majority of students working on the assignment. Our prior studies have suggested that these novice students often rely on support from the teacher throughout the assignment, either in terms of understanding code, completing actions within the environment, or determining whether or not things are correct [Study3,Study4]. Without the aid of the teacher, it might be necessary to either place more instructional support within the system itself (through data-driven means of supporting the students while they work) or within the instructional material (more explicit directions).

7.6 Conclusion

Using this study in combination with the prior 4, we can conclude that while students can benefit from adding "Choice" as the endpoint to Use Modify Create assignments, there are several things that must be considered beforehand. Proper scaffolding must be put in place in order to help students work on choices independently. This could come in the form of instructional material (most likely in more depth than just the suggested blocks we gave to students in this experiment) or other social supports such as access to help from other students (working together) or knowledgeable teachers. As students are seeking ways to validate and check their work, incorporation of data-driven systems that check assignment features could be utilized during these assignments, such as the ones currently in development for Snap programming assignments [Marwan2020]. In addition, as students seemed to gravitate towards easier choices within their perceived level of ability rather than their original intent, there seems to be motivation to incorporate individualized scaffolding within this assignment framework. This could come in the form of different instructional support for students coming from different levels of ability or experience such as more explicit directions, access to data-driven help within the environment, or differing starter code. In this way, students could work on the choices that they are more motivated to do without worrying about which choices are within their ability.

Thought Choices is not "Create" as originally intended or expressed in the original formulation of "Use Modify Create" [Lee2011], this variant is much closer to the original intent of student self-direction than other versions of the progression (e.g. Study 1). It is clear that students still need help and scaffolding as they transition from Modify to

Create in a sequence that only occurs over the course of 3 days, but longer timeframes or more frequent occurrence of these activities could allow students, once comfortable, to engage in truly self-directed learning, fully realizing "Create" as originally designed. In the meantime, this 3 day framework for assignment design ending in Choice provides a middle ground for students and teachers who want to have student agency and self-direction in the assignment while still maintaining the level of scaffolding and limitation of scope necessary for successful implementation with core K-12 courses.

CHAPTER

8

CONCLUSIONS

8.1 Introduction

In the previous chapters, we have examined how variations in the "Use-Modify-Create" curricula sequencing and design affected classroom outcomes. Notably, affecting student engagement with the activities, teacher's roles and their actions during the activities, students completion of tasks within the environment, and their changing sense of perceived difficulty and ownership throughout the activity. We begin this chapter by reviewing again the purported claims of Use Modify Create. We then synthesize the results from the 5 studies within this dissertation to determine which of the purported claims this research has supported and which requires more evidence within the context of future work. We also will detail how though "Use Modify Create" began as more of a theory of Computational Thinking engagement, this work has reified it into a specific framework for assignment designs to be used within core K-12 contexts. While we discuss how this translation has lost some of the specific original intent of some of the key components, we detail using evidence from the 5 studies how this reinterpretation can potentially better serve students and teachers in creating effective CT experiences within Core K-12 classrooms as well as

how this framework can be extended to bring it more in line with the original intent. We end this chapter with creating an understanding of the relation between different aspects of a classroom CT experience and how they affect student engagement throughout the activity. We connect these findings and these relationships with the educational theory of motivation, Self-Determination Theory, in order to explain these findings, create an understanding of how Use Modify Create provides motivating contexts for students, and how it can be extended to further promote student motivation to engage in computing activities.

8.2 Research Questions

RQ1: How do we create a pedagogical framework for infused computing activities that will scaffold students into working within block-based environments and reduce perceived difficulty of programming among novice students and teachers?

Study 1 demonstrated that a Use Modify Create assignment design can help students maintain a constant level of perceived difficulty as opposed to a non-scaffolded equivalent. In that Study, we found students who participated in the non-scaffolded activity saw a significant increase in their perceived sense of difficulty of the assignment when going from the Unplugged to Plugged Activity (i.e. when they had their first day in the programming environment). Further, we also found that students who participated in the Use Modify Create activity saw a significant increase in their perception that the code was their own creation over the course of the activity and more so than the equivalent change in the non-scaffolded assignment. Observations from Study 1 also demonstrated that many students had a difficult time keeping pace with the teacher in order to follow the coding instructions, especially on the first day of coding. In addition, the teachers who were given the Use Modify Create activities felt comfortable leading the class and felt the pace was good for their students. This reaction runs counter to the one expressed by teachers who participated in the non-scaffolded form, who wished their activity had been better scaffolded for students (like Use Modify Create had been) and had wished for variation in teaching than the monotony that the non-scaffolded version gave.

RQ2: How do we evaluate classroom implementations of these infused lessons in terms of both student experience and teacher instruction?

Study 3 introduced the CEO method, a triangulated approach using Code Traces, Exit Tickets and Observations useful for evaluating student performance and engagement dur-

ing infused lessons. Demonstrated using the context of an infused epidemics lesson, CEO was useful in identifying variation in student engagement and teaching style throughout the days, understanding which sources of help are most important to students (the teacher), documenting the shifting perception of the type of activity students were engaging with (science vs programming vs both), and identifying weak points within the assignment steps that are useful for researchers and activity designers in addressing assignment issues and setting a course for assignment redesign. Study 4 extended one element of the CEO model, code traces, to examine more the effects of teacher instruction during these lessons. Through multiple representations using the actions of students in relation to actions of teachers and comparing these representations to documented observations from that class period, we were able to identify instructional styles employed by the teachers as well as understand the demonstrable effect teachers have on the progress of their students. This CEO method is lightweight enough such that it can be easily employed within classrooms without researcher presence (though either teachers will need to self observe or enlist another individual) and further data-driven systems created by researchers can be developed to use information from these sources to give feedback and suggestions for improvement.

RQ3: How can a pedagogical framework allow for differentiation of tasks by interest and skill level to meet the needs of students, while being manageable for teachers to implement?

Study 2 began the process of translating the original Use Modify Create assignment design to one that afforded more opportunity for student differentiation and choice. In a study comparing two variants of Use Modify Create, one in which students explored the activity as it was in Study 1 and one variant of the activity that had students end in a set of Choices they could implement into the model, student code traces demonstrated that students were able to find the appropriate amount of work within the model (i.e. number of animals) to still maintain a constant sense of difficulty (as compared to the original group). Moreover, the teachers, who taught both models, expressed a strong preference for the variant with choices as they found students to be more engaged and did not find it to be more difficult to implement. Study 5 extended on this idea of choices by introducing into the design an endpoint of Choice with multiple difficulty levels and much more variation in code to implement than the ones presented in Study 2. Results from this Study demonstrate that students will often select easier choices than the ones they originally intended to implement and that teachers can often encourage this behavior. Though they choose easier choices, results from the study suggest that students are not actually able to complete these extensions fully without more scaffolding or help from the instructor. Further, while students increased their sense of ownership over the code by engaging in choices, we do not

have evidence to suggest that engaging in choices that one originally wanted to implement increased the sense of ownership over the code. Teachers expressed comfort with teaching the Choice variant so long as they had comfort over teaching the lessons in general. These results suggest that students, who shy away from their original choices, could be scaffolded through instructional supports or individualized versions of their Choices such that they can actually work towards something they would like to do at the appropriate level of difficulty to actually complete the choice within the time allotted.

8.3 Review of Use Modify Create Claims and Evidence from Dissertation

Irene Lee et al. originally proposed "Use-Modify-Create' as a " three-stage progression for engaging youth in CT within...rich computational environments" [Lee2011]. As initially stated, UMC was designed to be able to "transform learners from users to creators of computational artifacts", promote "students' agency and independence as learners", and " maintain a level of challenge that supports growth while limiting anxiety " [Lee2011,Martin2020]. Recent shifts by the original authors also include the goals of learning domain topics (e.g. Science) through CT activities and promoting creative expression [Martin2020]. Later work building upon the framework has also focused on the ideas that this progression can also help students learn programming and computational thinking concepts [Lao2019,Salac2020]. We itemize these claims and connect the supporting evidence from our studies below.

Maintain Level of Challenge/Reduce Difficulty: Study 1 demonstrated in a quasi experimental design that the "Use-Modify-Create" version of the Food Webs activity provided a significantly easier introduction to working within the Block-based programming environment than the version of the activity where students started programming on Day 2. Study 2 also found this to be the case, with no reported spike in perceived difficulty between the days. Study 5 found a significant spike in difficulty progressing from "Modify" to "Choose" which was not found within Study 2. We explain in Study 5 that this is most likely due to the lack of stronger scaffolding or instructional material for students, In addition, compared to Study 2 which just had students create different animals with almost identical code, Study 5 had students engage in different programming tasks that involved differing levels of code edits, types of code blocks, and used different sprites. While the primary measurement for all three of these studies were student's self-reporting on the perceived difficulty of the assignment, Study 1 also contained observed differences between versions both in stu-

dent engagement and teacher's comfort teaching the materials that also point to evidence that "Use Modify Create" might be better at maintaining a level of challenge necessary for students. As originally hypothesized by Lee et al [Lee2011], the goal of the "Use" and "Modify" stages would be to create challenges within a students ability in the beginning, and gradually scaffold them to progressively more difficult challenges. In a similar way, this work's "Use" day gets students comfortable working within the environment using a worked example led by the teacher. Similarly, the "Modify" task has students work on making code additions to the model, often aided through instructional scaffolding (e.g. Parsons Problems) and teacher direction. These appropriately scaffolded tasks lead through teacher instruction seemed to be able to provide the sufficient conditions to reduce the initial difficulty of working within these environments and provide an appropriate level of challenge for students throughout the activity confirming the original belief. Further, extending the model using "Choices" allows students to differentiate based on skill which can create more opportunities for students to select an appropriate challenge based on their perceived competency.

Transform Students from "Users to Creators" - Use Modify Create was designed to progress students from being consumers to creators of computational artifacts. This progression mirrors the three stages listed in which students begin as users of someone else's code, can modify the code to the point where their own ideas can take hold, and finally work towards creating their own works. Study 1 demonstrated that Students within the Use Modify Create condition increased their sense of ownership over the code throughout the course of the assignment with students having a higher sense of code ownership after Modify and Create than during the Use activity. Study 1 also demonstrated that compared to the assignment sequencing that had students program their models on all three days, students who first experienced "Use" and then experienced "Create" developed a higher sense of ownership over the code. The addition of Choices also seemed to increase student's sense of ownership over the code, with some mild support from observations in the form of Study 2 and stronger evidence in the form of higher reported scores during the "Choose" Day in study 5. These studies combine to provide strong evidence that students do progress from "Users to Creators" through a Use Modify Create sequencing. This instructional framing of working from a base code and slowly modifying and adding new features also served as a preferred and useful pedagogical sequence for teachers who generally felt comfortable leading the Use and Modify days through direct instruction and successfully transitioned during the Create day to a facilitation role.

Increasing Student Agency/Enabling Personalization - As originally designed, Use

Modify Create's "Create" section was more centered on student self-directed learning [Lee2011]. In this way, a student's sense of agency and autonomy over their learning was preserved as they could decide to pursue whatever was meaningful to them. This promotion of student agency falls in line with constructionist educational theory [Papert1991] and the block-based programming languages and environments that were designed around these ideas such as Scratch [Resnick2009]. However, it is clear that the Use Modify Create presented within this study as currently designed provides more limitations on student agency than its original intent. Students no longer have the ability to choose their own pathway for Create and are bounded during the "Use" and "Modify" times by what the instructor is leading them through. Still, the positive evidence of student engagement and progression through coding activities from Study 2 and 5 suggest that the "Choice" dynamic can offer an intermediary. Moreover, this Choice stage might work well for instructors as they are able to limit the amount of pathways an individual student may go, thereby making their role as a facilitator easier. It is also important to note that reframing Create to "Choose" is not an action solely taken by this work; Natalie Lao also adopted this within the context of a ML class so that students focused on a set of possible algorithms to use for their final project [Lao2019].

Students Engaging with Computational Thinking within Core Courses: While the studies that lead to the development of Use Modify Create occurred within after school programs, one of the stated goals of the research was to move towards Computational Thinking activities being embedded within core courses during the school day [Lee2011] .All 5 studies in this dissertation have been concerned with how to create assignment designs that not only engage students, but allow for easy adoption within Core K-12 courses. In all studies, teachers expressed comfort with being able to lead these assignments within their classroom barring the assignment conformed to more elements of a Use Modify Create design (i.e. the views expressed in Study 1) or had equivalent preparation time to those who had had some prior experience teaching these types of activity (i.e. the views expressed by teacher C in study 5). It is important to note how much of a modification to the original intent was necessary in order to get Use Modify Create within K-12 classrooms. These are detailed in the further section entitled "Translation into Assignment Design" but overall, modifications needed to be made in order to shift from the original, student self-directed design, to one that had a more heavily involved teacher role. This serves multiple purposes. The first being a sense that the teacher is also engaging in these learning activities with the student. The second being the role of the teacher as a source of knowledge that can be used to help struggling students and for teaching them concepts of how to engage in

the environment (as reported by Study 3 as being vitally important and as demonstrated further from classroom analysis within Study 4).

Increase Programming Ability - Later Use Modify Create work such as those done by Natalie Lao [Lao2019] and Jean Salac [Salac2020] have used UMC as a means of teaching programming or improving students learning of programming concepts. While our methodology of marking students actions within the programming environment provides evidence of students actually being able to complete these tasks given during these assignments, we performed no analysis or pre-post tests to determine whether or not students had an increased ability or understanding of programming topics after the activity. Intuitively, one can take the evidence that students without prior experience were able to create and work within these block-based programming environments as evidence suggesting that these activity designs can produce learning gains. However, whether or not this can be empirically demonstrated or whether or not a UMC sequence does the best job at increasing a student's programming ability compared to other potential progressions remains for future work.

Learning Gains (Domain/CT): Finally, more recent work using Use Modify Create is designed to determine whether given the infused context, learning gains can be produced for either Computational Thinking or for the domain subject [Martin2020]. Much like how increased programming ability was not addressed through this work, determining whether UMC promotes learning gains is an area for future work. Using a similar study design as Study 1, one could determine whether or not a more "UMC-style" activity was better at promoting student learning of either topic within the activity.

8.4 Translation into Assignment Design

The original authors of "Use-Modify-Create" cautioned against taking the three stages "too literally" [Lee2011]. However, much of the contributions of this work has come from mostly ignoring this advice. In all studies except for Study 3, students participated in activities segmented by separate days in which they either were engaging in a "Use" activity, "Modify" activity, or "Create" activity. This was initially done within Study 1 as an explicit study design measure in order to examine and ultimately confirm the proposed claim that this progression serves as a less difficult introduction to working with these Computational Thinking environments. However, throughout the rest of the studies, the idea of segmenting into separate days allowed for an explicit framing for future assignment designs usable by teachers and students.

The table below outlines the changes made from the original Use Modify Create to the one adopted within this study. The primary change from its original intent was the structuring of activities away from solely student lead endeavors to ones that gradually progressed from teacher instructed to student-independent initiatives from a set of choices. Originally proposed, the model "Use-Modify-Create" had for student engagement with the rich computational environments fell more aligned with the Constructionist ideas of more completely self-directed learning. This work has moved away from that for a number of reasons. First and foremost, prior work indicated that teacher engagement during infused activities within core K-12 contexts tended to promote more student engagement with the activities [Catete2018]. This work, building off of these findings, has also found evidence that not only do teachers help students engage with the activities, but serve as an incredibly important instructional support for these novice students. This was most evidently found within Studies 3 and 4 which demonstrated that students rely on teachers for information on how to progress and complete actions within the environment. Moreover, the framing of the activity as something led by the teacher in which all students are involved with allows for an additional layer of motivation for students who might not be fully internally motivated to engage in computational activities alone. This importance of relatedness is discussed further in the section on Self-Determination Theory.

It is important to note that this work is not the only reinterpretation or expansion on the original Use Modify Create in order to work for explicit domains. Sue Sentance] through PRIMM (Predict, Run, Investigate, Modify, Make) [Sentance2019] expanded the original "Use" into the PRI steps listed in order to give explicit scaffolding and instructing for students. This expansion was based on work done by Lister et al on the importance of reading code before writing [Lister2004]. This explicit sequence of steps for the "Use" stage is also found in work done by Jean Salac in the TIPPSEE strategy [Salac2020], which walks students through the Use stage through a guided set of things to pay attention to (Title, Instructions, Purpose, Play, etc). In a similar fashion, this work's "Use" Step involves direct instruction from the teacher who is able to field questions, ask questions to students, and generally walk through how to read and understand how the code blocks work in a manner very similar to PRIMM. Natalie Lao in work on teaching ML added a "Choose" stage at the end to limit the student's selections of models [Lao2019]. Similar to how Choice was implemented in this research, this limitation of the "Create" phase to a set of options limits the number of pathways students can veer off to, aiding in the facilitation of the lesson on an instructor standpoint. While others exist, these examples serve to highlight how other researchers working off of Use Modify Create have already engaged in reinterpretations or

expansions of the original intent to suit specific contexts.

8.5 Use Modify Create and Self-Determination Theory

We now take time to try and determine what educational theories can explain the findings over the past 5 studies and how better adherence to the consequences of these theories can allow for future activities that will be more effective for students and teachers. Prior work such as that done by Price [Price2018] and Zhi [Zhi2019] have framed the studies of students working with block-based programming environments around cognitive load theory. This theory is useful for explaining the differing effects of certain levels of scaffolding (e.g worked examples or buggy code) on learning gains, increased programming efficiency, or increased completion rates. However, while it is possible that the gradually increased scaffolding from Use Modify Create can potentially increase these learning outcomes, they were not measured or the focus of the studies of this dissertation and so remain as future work. It is also important to note that unlike past research which has focused on student learning within established computing courses, within the Infused Context that these studies and activities have taken place, the pedagogical goal was to expose, but not necessarily to teach, computational thinking, computer science topics, and programming skills. The learning objectives of the course do not include programming skills or computer science topic knowledge and as such, a focus for this research on learning gains would be misplaced. Moreover, as this is a class context in which a range of prior student experiences with programming is to be expected, judging student's achievements in these contexts will more than likely reflect initial stratification of student experience than change occurring as a result of the initiative.

It is better to frame these infused activities as being designed to engage students while working within block-based programming environments, create a well scaffolded experience that does not lead to high perceptions that coding is difficult (as originally stated in [Lee2011]) and to motivate students to seek out additional computing educational opportunities. This could be elective courses, after school camps,or outside of school activities such as where most students currently get their CS education prior to college [Margolis2010]. These follow-up contexts like elective courses can be appropriately scaffolded and tailored to the groups of learners and contain explicit learning objectives. Before students can enroll in these elective courses, however, students have to develop a sense of self-efficacy from prior programming experiences (such as through infused contexts) and believe they can

succeed within these elective courses. This frames this dissertation around how Use Modify Create can develop students' motivation within these contexts. Students, through well-designed infused activities, can develop intrinsic motivation to direct their own learning of computational thinking and programming topics outside of the initial context. As such, it is appropriate to determine a theory of motivation that can explain past results and motivate how to design better infused activities using Use Modify Create.

Self-Determination Theory is a theory of motivation proposed by Edward Deci and Richard Ryan [Deci1991]. The theory attempts to explain the driving factors towards an individual being intrinsically motivated to complete tasks or engage in behaviors (such as a sense that the activity is fun or satisfying) rather than being extrinsically motivated, where the motivation is in pursuit of some external or outside reward (such as receiving a high-mark on a grade). Self-Determination identifies three psychological needs necessary for building or sustaining internal motivation. The first is a sense of competency, or an individual's drive or need to achieve or have mastery or complete tasks. The second, autonomy, is the individual's need to have control over their actions and to be able to be the one initiating and making choices. The final one, relatedness, is the sense of belonging and feeling cared for and caring for others[Ryan2002]. In the context of education, this usually refers to connections between an individual and the teacher (such as feeling respected by the teacher) or connections between individual students.

Self-Determination Theory research within the context of education has demonstrated that a student's sense of competency and autonomy are crucial for internal motivation. These satisfied psychological needs are predictive of learning effort [Reeve2002], motivation to achieve academically [Chirkov2001] lower anxiety of learning material and increased interest[Deci2000], higher satisfaction with learning experience [Jang2007], greater effort to persist [Standage2006] among others. While much work has been done to examine the interplay and effect of competency and relatedness, work done by Allison Mithcin focused on women in Computer Science suggests that Relatedness might have an outside effect on underrepresented groups as a motivating factor [Mishkin2019]. This outsized effect on Relatedness was also found in a study focused on redesigning a Second-year Computer Engineering Course in which the authors found that the team project nature fostered relatedness among the college students and without the feeling of relatedness and competence, motivation would decline [Trenshaw2016]. This aligns with other work demonstrating that many underrepresented groups are motivated by social factors such as giving back to the community and that educational initiatives that are framed around these social dynamics might be more appealing and effective for these individuals [Isvik2020].

Strategies for fostering student autonomy include giving students more choice and control over the learning process. Strategies identified for fostering competency include giving students optimally challenged tasks and providing constructive feedback. Finally, strategies identified for fostering relatedness include respecting students and supporting them as a teacher during the learning process [Niemiec2009].

Using this information, below, I present a Theory of Action of all of the factors that I believe contribute to the development of Autonomy, Relatedness, and Competency of students within Use Modify Create Assignments. First, prior student experience with working in block-based environments or programming in general is most likely very helpful in developing a students perception that they can complete tasks during infused activities. However, as stated in the goals of these types of activities, we cannot assume that students have any prior experience or knowledge and therefore, must find other means of satiating student competency. One source is within the environment itself, as it is not necessarily necessary to have these assignments within block-based programming environments. However, multiple studies have demonstrated that using block-based environments over traditional text-based programming might be helpful in reducing student error rate, increasing their speed of assignment completion, or promoting better understanding of cs topics [Weintrop2017]. Moving away from student competency, student relatedness can be first developed in the fact that these assignments are occurring within infused contexts within their core K-12 classrooms. This is because students will now have a social learning context where all their peers, regardless of past experiences, will be working together on building computational thinking models. Further, the teacher will be leading said activity which will create a positive connection of support between the student and their teacher.

The UMC Sequence works at explicitly defining a student and teacher role during the activity. Both the role of the student engaging in the activity and the role of the teacher leading the activity help in supporting the development of student motivation. First, the UMC sequence defines a set of scaffolded tasks that students without prior knowledge can engage with. This will allow students to maintain a sense of competency in the beginning of the assignment. If students have more prior experience than assumed, these tasks can fall way below their Zone of Proximal Development, but it is important to note that they can still achieve a sense of mastery and completion with the easier tasks, even if they are not appropriately challenged. The defined role of the teacher in the beginning as instructor also aides in developing student competency. The teachers direct instruction acts as a source of feedback and information necessary for novice students to complete tasks. As the assignment moves away from direct instruction during Use and Modify to

open-ended independent learning during Create or Choose, students are able to first develop and satiate their sense of autonomy by choosing which elements they would like to engage with. By adopting a facilitator role and allowing students to pursue what they want, teachers also encourage this development of autonomy. Students will be able to choose an extension or set of extensions that fall within their sense of competency, which will aid all students ranging from those with high prior experience to those with no prior experience. Throughout this entire set of activities, teachers are not only able to support students autonomy and competency but their relatedness by becoming an active participant in the computational thinking activity and supporting students work during the activity. Finally, as intrinsic motivation might not be present for all students in the beginning, by directly involving the teacher and creating a class context where these activities are necessary or graded, teachers are able to supply a source of extrinsic motivation for students that while possibly necessary in the beginning, can hopefully be supplanted by student intrinsic motivation to engage in the activities as the course of the days progress.

8.5.1 Use Modify Create and Fostering Student Motivation

In the table below, I present an overview of the main ways in which Use Modify Create assignments affect student's competency, autonomy and relatedness. I detail how each study supports each of the claims.

First, properly sequencing and scaffolding the activity from a highly-scaffolded "Use" activity to a less scaffolded "Modify" activity and finally a lower scaffolded "Create" allows students to maintain a sense of competency throughout the learning process if they have no to low prior experience. Study 1 demonstrated this through the analysis of student's perceived difficulty in the Use Modify Create version vs the non-scaffolded version and demonstrated that student's found the introduction of programming to be more difficult in the non-scaffolded version. Study 2 demonstrated this maintenance of student perceived difficulty further, while Study 5 showed that students can have an increased sense of difficulty if the "Choice" section is not properly scaffolded for each individual student. Studies 3 and 4 demonstrated that students heavily rely on teachers as an aid for instructional support. Study 3 showed students identify teachers as their primary source of help and the multiple representations in Study 4 showed that student progress is often reliant on or completely at the mercy of teacher involvement. Finally, Studies 2 and 5 demonstrated that differentiation of tasks through choices can allow students to self select into an assignment activity within their perceived abilities. However, more scaffolding is needed in order for

students to work on even the easier of choices independently.

This inclusion of choices also is responsible for creating a sense of autonomy amongst students. Study 2 demonstrated from observations that students were more engaged in the task and walked around sharing their creations with other students and the teacher when given the option to Choose their final animal rather than only doing the Fox. Responses from teachers in both Study 2 and Study 5 identified teacher preference for activities that facilitate this level of agency and choice among students as well. Study 5 also demonstrated that students may need additional scaffolding to engage in the tasks that they express interest in, and if they do not receive that might shy away from difficult tasks and pursue easier labeled ones. The entire assignment framing of progressing from "Not My Code" to "My Own Code" may help in developing student autonomy, giving them a sense of control over their learning. Study 1 demonstrated that the assignment framing of progressing from "Using" to "Creating" developed a higher sense amongst students that the final code was their own than the non-scaffolded progression. Study 5 also showed that students who got to engage in choices also reported a much higher sense that the code was their own creation than the prior day spent modifying.

Finally, the explicit role of the teacher within the activity and the embedding of the computational thinking activity within the core K-12 course helps to develop students' sense of relatedness. Rather than just passively managing the classroom, teachers adopt in this Use Modify Create activity a more active role, either as direct instructors or facilitators. This sense that teachers care for and are a part of the learning process could serve to benefit students. The fact that these activities are also placed within the core classroom also gives students connections between each of the classmates in their period. Instead of framing the activity as a solitary pursuit, the computational thinking activity is one in which all students, regardless of past background, are engaged with and learning together.

It is important to note that within each of our studies and initiatives, there were still individuals not engaging with the prescribed learning objectives, either using the environment but not programming at all (instead using the environment to paint new sprites), not using the environment at all (instead using the computer to do other activities) or using the environment and programming but not doing the listed activities (for example, making their own objectives and trying to program that within the environment). The first two examples are most likely never to be completely removed from a classroom environment, and external motivation sources such as punishment, withholding of reward, or threat of a bad mark or grade might be employed by the teacher to keep students back on track. The last example sometimes represents a student with sufficient intrinsic motivation but not

employing that to complete the class objectives and instead engaging in a "Create" activity as it is usually traditionally described [Lee2011]. Teachers might feel comfortable allowing students to engage in further activities beyond the initial set, but the more open-ended these become, the more likely it is that a teacher might not know how to aid a student with a specific activity.

8.5.2 Use Modify Create and Teacher Motivation for Assignment Adoption

While the primary focus of this research has been on how a more "UMC" assignment design benefits students, the ways that UMC promotes the necessary components of autonomy, relatedness, and competency in teachers cannot be left out. Teacher's sense of competency is allowed to be satiated as the activities are framed within the topics they teach within their core classroom context. Each of the 2 activities from the 5 studies were framed around a scientific topic such that the science teachers had an easier time creating a motivating reason for engaging in the activities. In addition, teachers were often observed using this framing to create connections between the programming and the domain topic, using the activity as a review of what they had previously learned and applying it to a new context. Teachers also reported that they enjoyed extensions to the activity design that promoted exploration of the scientific topics (e.g. the multiple animals in the food web of Study 2 or the ability to explore effects of vaccines in Study 5). It is also important to note that the pedagogical sequence of the three days (Use, Modify, Create) seemed to also benefit teachers, as there was less that they had to instruct or teach on each day and could learn the material more easily as demonstrated by Study 1.

Teacher's sense of autonomy over their classroom is paired nicely with one affordance of UMC: teacher's ability to scaffold/differentiate leading the activity. Throughout each activity mentioned in the study, observers noted how teachers could vary instruction in order to help struggling students or control the pace of the classroom in their own manner. This could include things such as pairing students together if they anticipated those students needed help or varying the way they introduced steps in constructing the model (such as the variants in "I do, You do, We do" shown in Study 4). This afforded flexibility might be useful for teachers as if they feel they do not have to conform to a rigid style of teaching computational thinking and can feel good varying the curriculum to address their own needs or the needs of their classroom, they may be more willing to adopt the curriculum. Finally, the direct nature of the involvement of teachers within the Use Modify Create

sequence can strengthen a teacher's sense of relatedness in the same way it did for a student. Creating a context where teachers can directly engage and facilitate student learning of the context probably helps in establishing motivation for adopting the activities at all.

8.6 Future Directions for Use Modify Create Research

Using the endpoint goals of further trying to increase student and teacher's sense of autonomy, competency, and relatedness, there are several future directions this work can take. First, further support can be given to students as they are independently working on choices. This will serve two functions. First, methods of bringing the task within students' level of ability to complete within a class period will aid in their sense of competency as the likelihood of them completing it independently and within the class period will increase. Methods that can accomplish this could be authoring different levels of scaffolding for the same task as discussed in Study 5, or employing a data-driven approach of using prior student data to determine what level of scaffolding a task should have for an individual student. In a similar manner to systems like the ones developed by Ericson for dynamic parsons problems [Ericson2019], a task can be scaffolded by giving more information about the final structure of the blocks and where it needs to go in the model. Other systems could involve methods of giving student feedback without instructor involvement. A "Check my work" function similar to those developed by Price [Price2017] and further expanded upon by Marwan [Marwan2020] could allow students to get positive constructive feedback on their progress, satiating their sense of competency.

The above discussed systems would also aid in developing a students sense of autonomy. If a task could be scaffolded within a students ability to complete it within the period, then the tasks that students would like to do could be pursued over other tasks. We have seen from Study 5 that students often engage in doing tasks that are within their perceived limits, which usually involves easier tasks. If the harder choices are more appealing to students, and thus if completed could better satiated their sense of autonomy, then the systems previously discussed could help students work towards building these more desired choices rather than settling for ones within their reach. Another avenue for building student autonomy would be bringing back the original intent of the "Create" portion of Use Modify Create and allow students to develop any idea, not just choices, in the model. As this would create a larger burden on teachers in terms of facilitating as well as be best suited for students who already have enough of an ability to realize their ideas in the model, it is probably best

for teachers to adopt this as an activity students can engage with after already completing choices or reserving it for students who they deem able to do this alone.

Teacher Competency and Autonomy could be further developed through high quality teacher professional development. Currently, Professional Development for teachers trying to embed computational thinking in their classrooms, such as the Infusing Computing Professional Development, already focuses on becoming competent in block-based programming languages and includes developing an assignment usable during the school year [Jocius2020]. The work in this dissertation suggests that teachers could benefit from designing assignments around a Use Modify Create framework. In addition, if students benefit from learning through this progression, it is possible that teachers could also benefit during professional development through modules designed to teach them programming that also conform to the Use Modify Create progression. Teacher Autonomy could also be developed through new data-driven systems for assignment deployment. The studies of this dissertation demonstrate that there was considerable variation in how teachers taught individual classrooms. Teachers who believed that students needed more help often paired students together or gave them additional help in the beginning. Rather than put more pressure on teachers to teach the activity in a standardized way, we can instead try to satiate their desire to have more control over the material by allowing the assignment deployment to be flexible. Systems could be developed that can take instructor input on how a certain class period should run (e.g. paired or unpaired, easier version of the assignment or harder, etc). Further systems could also be developed to utilize the information collected from Study 4 on student-teacher programming behavior to help teachers as they instruct. This could include feedback in terms of how many students are following along, current student progress, or indicators to either slow down or speed up the instruction. In addition to systems that can provide teacher feedback during implementation, systems that review classroom behavior and student's individual progress after the fact could also be beneficial for teachers.

Finally, it is important to hypothesize on ways that we can further promote both students and teachers' sense of relatedness while engaging in these activities. The block-based programming environments and simulations or activities that can be designed using them provide meaningful contexts for students to be able to creatively express themselves. Kate Compton defined a "Casual Creator" as tools that allow individuals to engage in "the fast, confident, and pleasurable exploration of a possibility space, resulting in the creation or discovery of surprising new artifacts that bring feelings of pride, ownership, and creativity to the users that make them." [Compton2015]. Two of the important design principles for

a "Casual Creator" is the ability to easily share and export one's work and engage easily with communities of other creators. These functions are already within block-based environments such as Snap and Scratch, but further emphasis on sharing your work with other students, the teacher, and individuals outside of the school context (e.g. their parents) could further develop a student's sense of relatedness. This could take the form of explicit modifications of the activity progression so that students have explicit time for sharing their work during class and getting positive feedback from others. Further tools for sharing and exporting assignments would also be beneficial for developing teacher relatedness. If teachers are able to share their experiences on infusing computing within their classroom, learn from others doing the same, and easily share resources and materials, a community of practice could form around infused assignments and could help not only satiate teacher relatedness, but further the adoption and spread of materials.

8.7 Contributions

1. Studies that motivate the use of UMC as a pedagogical framework because of reduced difficulty, increased ownership, and ease of teacher implementation.

2. Studies demonstrating how Code Traces, Exit Tickets, and Observations can be used to redesign curricula, compare class implementations, understand student perceptions during activities, and explore teachers instructional style.

3. Studies that motivate the use of "Choice" as an endpoint for UMC, creating a bridge between open-ended, self-motivated tasks and scaffolded tasks teachers can facilitate and adapt to a student's individual skill level.

4. Data-driven methodologies for exploring teacher-student relationships while programming that can be used to develop feedback systems for teachers as they instruct students.

5. A Framework for understanding how various factors influence student motivation and engagement with computing activities within core K-12 courses and how a Use Modify Create assignment design works to better engage students.

REFERENCES

[1] Bell, T., Alexander, J., Freeman, I., and Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1):20–29.

[2] Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73.

[3] Bennedsen, J. and Caspersen, M. E. (2005). Revealing the programming process. In *ACM SIGCSE Bulletin*, volume 37, pages 186–190. ACM.

[4] Biggs, J. B. and Collis, K. F. (2014). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.

[5] Boyce, A. K., Campbell, A., Pickford, S., Culler, D., and Barnes, T. (2011). Experimental evaluation of beadloom game: how adding game elements to an educational tool improves motivation and learning. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 243–247, New York, NY. ACM, ACM.

[6] Brown, N. C. C., Kölling, M., McCall, D., and Utting, I. (2014). Blackbox: a large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 223–228. ACM.

[7] Cateté, V., Lytle, N., Dong, Y., Boulden, D., Akram, B., Houchins, J., Barnes, T., Wiebe, E., Lester, J., Mott, B., and Boyer, K. (2018). Infusing computational thinking into middle grade science classrooms: Lessons learned. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, WiPSCE '18, pages 21:1–21:6, New York, NY, USA. ACM.

[8] Cateté, V., Wassell, K., and Barnes, T. (2014). Use and development of entertainment technologies in after school stem program. In *Proc. of the 45th ACM technical symposium on Computer science education*, pages 163–168. ACM, ACM.

[9] Chen, C. and Guo, P. J. (2019). Improv: Teaching programming at scale via live coding. In *Proceedings of the Sixth Annual ACM Conference on Learning at Scale*, L@S '19, New York, NY, USA. ACM.

[10] Clements, D. H. and Meredith, J. S. (1993). Research on logo: Effects and efficacy. *Journal of Computing in Childhood Education*, 4(4):263–290.

[11] Coburn, C., Penuel, W., and Geil, K. (2013a). Research-practice partnerships at the district level: A new strategy for leveraging research for educational improvement. *Berkeley, CA and Boulder, CO: Univ. of California and Univ. of Colorado*.

[12] Coburn, C. E., Penuel, W. R., and Geil, K. E. (2013b). Practice partnerships: A strategy for leveraging research for educational improvement in school districts. *William T. Grant Foundation*.

[13] Collins, A., Brown, J. S., Newman, S., and Resnick, L. (1989). Knowing, learning, and instruction: Essays in honor of robert glaser. *Cognitive apprenticeship: Teaching the craft of reading, writing, and Mathematics. Hillsdale, NJ: Lawrence Erlbaum Associates*, 8(1):453–494.

[14] Coulter, B., Lee, I., and Martin, F. (2010). Computational thinking for youth.

[15] Council, N. R. et al. (2011a). *Report of a workshop on the pedagogical aspects of computational thinking*. National Academies Press.

[16] Council, N. R. et al. (2011b). *Successful K-12 STEM education: Identifying effective approaches in science, technology, engineering, and mathematics*. National Academies Press, Washington, D.C.

[17] Cuny, J. (2012). Transforming high school computing: a call to action. *ACM Inroads*, 3(2):32–36.

[18] Dalbey, J. and Linn, M. C. (1986). Cognitive consequences of programming: Augmentations to basic instruction. *Journal of Educational Computing Research*, 2(1):75–93.

[19] Deci, E. L., Vallerand, R. J., Pelletier, L. G., and Ryan, R. M. (1991). Motivation and education: The self-determination perspective. *Educational psychologist*, 26(3-4):325–346.

[20] Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., and Basu, S. (2017). An instructor dashboard for real-time analytics in interactive programming assignments. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, pages 272–279. ACM.

[21] Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., and Basu, S. (2018a). Measuring transfer of data-driven code features across tasks in alice. In *Proceedings of SPLICE 2018 workshop Computing Science Education Infrastructure*, volume 5.

[22] Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., and Basu, S. (2018b). Peer tutor matching for introductory programming: Data-driven methods to enable new opportunities for help. In *ISLS 2020*. International Society of the Learning Sciences, Inc.

[23] Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., Joshi, D., Robinson, R., and Andrews, A. (2019). Prada: A practical model for integrating computational thinking in k-12 education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 906–912, New York, NY, USA. ACM.

[24] Duffy, T. M. and Cunningham, D. J. (1996). 7. constructivism: Implications for the design and delivery of instruction.

[25] Ericson, B. J., Margulieux, L. E., and Rick, J. (2017). Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, pages 20–29.

[26] Fishman, B. J., Penuel, W. R., Allen, A.-R., Cheng, B. H., and Sabelli, N. (2013). Design-based implementation research: An emerging model for transforming the relationship of research and practice. *National society for the study of education*, 112(2):136–156.

[27] Fonseca, N. G., Macedo, L., Marcelino, M. J., and Mendes, A. J. (2018). Augmenting the teacher's perspective on programming student's performance via permanent monitoring. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9.

[28] Franklin, D., Salac, J., Crenshaw, Z., Turimella, S., Klain, Z., Anaya, M., and Thomas, C. (2020). Exploring student behavior using the tippsee learning strategy. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, page 91–101, New York, NY, USA. Association for Computing Machinery.

[29] Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701.

[30] Frykholm, J. (2004). Teachers' tolerance for discomfort: Implications for curricular reform in mathematics. *Journal of Curriculum and Supervision*, 19(2):125–149.

[31] Garcia, D., Harvey, B., and Barnes, T. (2015). The beauty and joy of computing. *ACM Inroads*, 6(4):71–79.

[32] Garneli, V., Giannakos, M. N., and Chorianopoulos, K. (2015). Computing education in k-12 schools: A review of the literature. In *2015 IEEE Global Engineering Education Conference (EDUCON)*, pages 543–551. IEEE.

[33] Gee, J. P. (2007). *Good video games+ good learning: Collected essays on video games, learning, and literacy*, volume 27. Peter Lang, New York, NY.

[34] Google (2018). What is computational thinking?

[35] Grgurina, N., Barendsen, E., Suhre, C., Zwaneveld, B., and van Veen, K. (2018). Assessment of modeling and simulation in secondary computing science education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, WiPSCE '18, pages 7:1–7:10, New York, NY, USA. ACM.

[36] Grizioti, M. and Kynigos, C. (2018). Game modding for computational thinking: an integrated design approach. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, pages 687–692, New York, NY, USA. ACM.

[37] Große, C. S. and Renkl, A. (2007). Finding and fixing errors in worked examples: Can this foster learning outcomes? *Learning and instruction*, 17(6):612–634.

[38] Grover, S., Pea, R., and Cooper, S. (2016). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM technical symposium on computing science education*, pages 552–557. ACM, ACM.

[39] Guo, P. J. (2015). Codeopticon: Real-time, one-to-many human tutoring for computer programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM.

[40] Harvey, B., Garcia, D., Paley, J., and Segars, L. (2012). Snap!:(build your own blocks). In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 662–662. ACM.

[41] Houchins, J. K., Boulden, D. C., Akram, B., Wiebe, E., Cateté, V., Dong, Y., Lytle, N., Milliken, A., Barnes, T., Lester, J., Mott, B., and Boyer, K. E. (2019). Designing a computational modeling unit for middle grades science classrooms: Grounding decisions in practice. In *2019 American Educational Research Association Annual Meeting*, Toronto, Canada.

[42] Husic, F. T., Linn, M. C., and Sloane, K. D. (1989). Adapting instruction to the cognitive demands of learning to program. *Journal of Educational Psychology*.

[43] Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., et al. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM.

[44] Jocius, R., Albert, J., Andrews, A., Catete, V., Dong, Y., Joshi, D., Robinson, R., Barnes, T., and Lytle, N. (2019). Infusing computing through professional development: Shifts in content area teachers' understandings of computational thinking integration. In Graziano, K., editor, *Proceedings of Society for Information Technology & Teacher Education International Conference 2019*, pages 302–305, Las Vegas, NV, United States. Association for the Advancement of Computing in Education (AACE).

[45] Jocius, R., Joshi, D., Dong, Y., Robinson, R., Cateté, V., Barnes, T., Albert, J., Andrews, A., and Lytle, N. (2020). Code, connect, create: The 3c professional development model to support computational thinking infusion. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE 20. ACM.

[46] Kafai, Y. B., Kafai, Y. B., and Kafai, Y. B. (1995). *Minds in play: Computer game design as a context for children's learning*. Routledge.

[47] Kafai, Y. B. and Resnick, M. (2012). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Routledge.

[48] Kalelioğlu, F. (2015). A new way of teaching programming skills to k-12 students: Code. org. *Computers in Human Behavior*, 52:200–210.

[49] Kirschner, P. A., Sweller, J., and Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, 41(2):75–86.

[50] Krathwohl, D. R. (2002). A revision of bloom's taxonomy: An overview. *Theory into practice*, 41(4):212–218.

[51] Lane, A., Meyer, B., and Mullins, J. (2012). *Simulation with Cellular A Project Based Introduction to Programming*. BlockBooks Series. Monash University, Melbourne, Australia, first edition. Online: https://github.com/MonashAlexandria/snapapps.

[52] Lee, I., Martin, F., and Apone, K. (2014). Integrating computational thinking across the k–8 curriculum. *Acm Inroads*, 5(4):64–71.

[53] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., and Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, 2(1):32–37.

[54] Lee, M. J. (2014). Gidget: An online debugging game for learning and engagement in computing education. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 193–194. IEEE.

[55] Linn, M. C. and Dalbey, J. (1985a). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, 20(4):191–206.

[56] Linn, M. C. and Dalbey, J. (1985b). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, 20(4):191.

[57] Lister, R., Fidge, C., and Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin*, 41(3):161–165.

[58] Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., Isvik, A., Bounajim, D., Wiebe, E., and Barnes, T. (2019a). Use, modify, create: Comparing computational thinking lesson progressions for stem classes. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 395–401. ACM, ACM.

[59] Lytle, N., Cateté, V., Dong, Y., Boulden, D., Akram, B., Houchins, J., Barnes, T., and Wiebe, E. (2019b). Ceo: A triangulated evaluation of a modeling-based ct-infused cs activity for non-cs middle grade students. In *Proceedings of the ACM Conference on Global Computing Education*, pages 58–64. ACM, ACM.

[60] Lytle, N., Cateté, V., Dong, Y., Boulden, D., Akram, B., Houchins, J., Barnes, T., and Wiebe, E. (2019c). Ceo: A triangulated evaluation of a modeling-based ct-infused cs activity for non-cs middle grade students. In *Proceedings of the ACM Conference on Global Computing Education*, CompEd '19. ACM.

[61] Margolis, J. (2010). *Stuck in the shallow end: Education, race, and computing*. MIT Press, Cambridge, MA.

[62] McKnight, P. E. and Najab, J. (2010a). Kruskal-wallis test. *The corsini encyclopedia of psychology*, 4:1–1.

[63] McKnight, P. E. and Najab, J. (2010b). Mann-whitney u test. *The Corsini encyclopedia of psychology*, 4:1–1.

[64] Means, B. and Penuel, W. R. (2005). Scaling up technology-based educational innovations. *Scaling up success: Lessons learned from technology-based educational improvement*, pages 176–197.

[65] Miles, M. B., Huberman, A. M., and Saldana, J. (2014). *Qualitative data analysis*. Sage, Washington DC, USA.

[66] Ocumpaugh, J. (2015). Baker rodrigo ocumpaugh monitoring protocol (bromp) 2.0 technical and training manual. *New York, NY and Manila, Philippines: Teachers College, Columbia University and Ateneo Laboratory for the Learning Sciences*.

[67] Pajares, F. (1996). Self-efficacy beliefs in academic settings. *Review of educational research*, 66(4):543–578.

[68] Papert, S. and Harel, I. (1991). Situating constructionism. *Constructionism*, 36(2):1–11.

[69] Parsons, D. and Haden, P. (2006a). Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 157–163, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[70] Parsons, D. and Haden, P. (2006b). Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 157–163, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[71] Paxton, J. (2002). Live programming as a lecture technique. *J. Comput. Sci. Coll.*, 18(2):51–56.

[72] Pearson, P. D. and Gallagher, M. C. (1983). The instruction of reading comprehension. *Contemporary educational psychology*, 8(3):317–344.

[73] Price, T. W., Albert, J., Catete, V., and Barnes, T. (2015). Bjc in action: Comparison of student perceptions of a computer science principles course. In *Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 2015*, pages 1–4. IEEE, IEEE.

[74] Price, T. W., Cateté, V., Albert, J., Barnes, T., and Garcia, D. D. (2016a). Lessons learned from bjc cs principles professional development. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 467–472, New York, NY. ACM, ACM.

[75] Price, T. W., Cateté, V., Albert, J., Barnes, T., and Garcia, D. D. (2016b). Lessons learned from “bjc" cs principles professional development. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE ’16, pages 467–472. ACM.

[76] Price, T. W., Dong, Y., and Barnes, T. (2016c). Generating data-driven hints for open-ended programming. *International Educational Data Mining Society*, 9(1).

[77] Price, T. W., Dong, Y., and Lipovac, D. (2017a). isnap: towards intelligent tutoring in novice programming environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM.

[78] Price, T. W., Hovemeyer, D., Rivers, K., Becker, B. A., et al. (2019). Progsnap2: A flexible format for programming process data. In *The 9th International Learning Analytics & Knowledge Conference, Tempe, Arizona, 4-8 March 2019*.

[79] Price, T. W., Liu, Z., Cateté, V., and Barnes, T. (2017b). Factors influencing students’ help-seeking behavior while programming with human and computer tutors. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 127–135. ACM.

[80] Pucher, R. and Lehner, M. (2011). Project based learning in computer science–a review of more than 500 projects. *Procedia-Social and Behavioral Sciences*, 29:1561–1566.

[81] Raj, A. G. S., Patel, J. M., Halverson, R., and Halverson, E. R. (2018). Role of live-coding in learning introductory programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, page 13. ACM.

[82] Reeve, J. and Tseng, C.-M. (2011). Agency as a fourth aspect of students’ engagement during learning activities. *Contemporary Educational Psychology*, 36(4):257–267.

[83] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. S., Silverman, B., et al. (2009). Scratch: Programming for all. *Commun. Acm*, 52(11):60–67.

[84] Rodger, S. H., Bashford, M., Dyck, L., Hayes, J., Liang, L., Nelson, D., and Qin, H. (2010). Enhancing k-12 education with alice programming adventures. In *Proceedings of the*

fifteenth annual conference on Innovation and technology in computer science education, pages 234–238.

[85] Rowe, J. P., McQuiggan, S. W., Robison, J. L., and Lester, J. C. (2009). Off-task behavior in narrative-centered learning environments. In *AIED*, pages 99–106.

[86] Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 651–656. ACM.

[87] Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., and Saulters II, C. (2010). Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 351–355. ACM.

[88] Sawyer, R., Smith, A., Rowe, J., Azevedo, R., and Lester, J. (2017). Enhancing student models in game-based learning with facial expression recognition. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, UMAP '17, pages 192–201, New York, NY, USA. ACM.

[89] Sentance, S. and Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, 22(2):469–495.

[90] Sentance, S. and Waite, J. (2017). Primm: Exploring pedagogical approaches for teaching text-based programming in school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pages 113–114, New York, NY. ACM, ACM.

[91] Sentance, S., Waite, J., and Kallia, M. (2019). Teachers' experiences of using primm to teach programming in school. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 476–482, New York, NY, USA. ACM.

[92] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.

[93] Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., and Wilensky, U. (2014). Defining computational thinking for science, technology, engineering, and math.

[94] Weintrop, D., Hansen, A. K., Harlow, D. B., and Franklin, D. (2018). Starting from scratch: Outcomes of early computer science learning experiences and implications for what comes next.

[95] Weintrop, D. and Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1):3.

[96] Werner, L., Campe, S., and Denner, J. (2012). Children learning computer science concepts via alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 427–432, New York, NY. ACM, ACM.

[97] Wilensky, U. and Rand, W. (2015). *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. Mit Press.

[98] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

[99] Woolson, R. (2007). Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3.

[100] Zhi, R. (2018). Exploring data-driven worked examples for block-based programming. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, pages 294–295, New York, NY, USA. ACM.

[101] Zhi, R., Lytle, N., and Price, T. W. (2018). Exploring instructional support design in an educational game for k-12 computing education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 747–752, New York, NY, USA. Association for Computing Machinery.

[102] Zhi, R., Price, T., Marwan, S., Dong, Y., Lytle, N., and Barnes, T. (2019). Toward data-driven example feedback for novice programming. *International Educational Data Mining Society*.

[Zhi et al.] Zhi, R., Price, T. W., Lytle, N., Dong, Y., and Barnes, T. Reducing the state space of programming problems through data-driven feature detection.