Bringing Computational Thinking to K-12 and Higher Education

by

Joshua Levi Weese

B.S., Kansas State University, 2011
M.S., Kansas State University, 2013

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Abstract

Since the introduction of new curriculum standards at K-12 schools, computational thinking has become a major research area. Creating and delivering content to enhance these skills, as well as evaluation, remain open problems. This work describes different interventions based on the Scratch programming language aimed toward improving student self-efficacy in computer science and computational thinking. These interventions were applied at a STEM outreach program for 5^{th}-9^{th} grade students. Previous experience in STEM-related activities and subjects, as well as student self-efficacy, were surveyed using a developed pre- and post-survey. The impact of these interventions on student performance and confidence, as well as the validity of the instrument are discussed. To complement attitude surveys, a translation of Scratch to Blockly is proposed. This will record student programming behaviors for quantitative analysis of computational thinking in support of student self-efficacy. Outreach work with Kansas Starbase, as well as the Girl Scouts of the USA, is also described and evaluated.

A key goal for computational thinking in the past 10 years has been to bring computer science to other disciplines. To test the gap from computer science to STEM, computational thinking exercises were embedded in an electromagnetic fields course. Integrating computation into theory courses in physics has been a curricular need, yet there are many difficulties and obstacles to overcome in integrating with existing curricula and programs. Recommendations from this experimental study are given towards integrating CT into physics a reality. As part of a continuing collaboration with physics, a comprehensive system for automated extraction of assessment data for descriptive analytics and visualization is also described.

Bringing Computational Thinking to K-12 and Higher Education

by

Joshua Levi Weese

B.S., Kansas State University, 2011
M.S., Kansas State University, 2013

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Approved by:

Major Professor
William H. Hsu

Copyright

Abstract

Since the introduction of new curriculum standards at K-12 schools, computational thinking has become a major research area. Creating and delivering content to enhance these skills, as well as evaluation, remain open problems. This work describes different interventions based on the Scratch programming language aimed toward improving student self-efficacy in computer science and computational thinking. These interventions were applied at a STEM outreach program for 5^{th}-9^{th} grade students. Previous experience in STEM-related activities and subjects, as well as student self-efficacy, were surveyed using a developed pre- and post-survey. The impact of these interventions on student performance and confidence, as well as the validity of the instrument are discussed. To complement attitude surveys, a translation of Scratch to Blockly is proposed. This will record student programming behaviors for quantitative analysis of computational thinking in support of student self-efficacy. Outreach work with Kansas Starbase, as well as the Girl Scouts of the USA, is also described and evaluated.

A key goal for computational thinking in the past 10 years has been to bring computer science to other disciplines. To test the gap from computer science to STEM, computational thinking exercises were embedded in an electromagnetic fields course. Integrating computation into theory courses in physics has been a curricular need, yet there are many difficulties and obstacles to overcome in integrating with existing curricula and programs. Recommendations from this experimental study are given towards integrating CT into physics a reality. As part of a continuing collaboration with physics, a comprehensive system for automated extraction of assessment data for descriptive analytics and visualization is also described.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to extend thanks to Nathan Bean and Russell Feldhausen who have helped organize, develop, and contribute material to the outreach efforts. I would like to thank Jesse Peters, Brooke Snyder, Kent Hildebrand, and Chris Herald for their work and assistance in the summer STEM Institute. I would also like to thank Beck Caitlyn and staff at Kansas Starbase Manhattan for there dedication to STEM and working with me to integrate computer science into their program. This work would not be possible without the support of my wife, Chelsea, my advisor, Dr. William H. Hsu, and my doctoral committee.

Chapter 1 - Introduction

Computer science has become one of the most relevant, fastest growing, and highest paying fields; however, Kansas and many other states are behbeind the curve for being an advocate of computer science. According to Code.org, 9 out of 10 parents want their children to study Computer Science (CS), but only 1 in 4 schools teach computer programming. There are 2,980 open computing jobs in Kansas, but only 338 CS college graduates (Code.org, 2017). The core problem in Kansas is that CS is not widely offered in K-12 schools, where there are no CS curriculum standards, CS does not count towards high school graduation, and no state-level funding for computer science professional development for teachers (Code.org, 2017). Nationwide, there has been a 17% increase to 54,379 students in 2016 who took AP CS; 23% of which were female and 16% were from underrepresented minorities (Code.org, 2017). Kansas also saw an increase in the number of students in AP CS; however, the number of students from underrepresented minorities decreased to only 7% and only 7% female (Code.org, 2017).

The ever-growing popularity of computer science has fostered the need for computational thinking (CT), especially in K-12 education. Creating and delivering content to enhance these skills, as well as evaluation, remain open problems. In recent years, countries have begun to develop and incorporate computing in the K-12 education system. From these curricula and reports, succinct definitions of CT provide broader impacts in terms of education. Currently, the US does not have country wide K-12 CS education standards. Organizations have been dedicated in creating CS standards that incorporate CT, although these standards are not widely adopted by the states. Since Jeanette Wing's ACM Viewpoint in

2006, CT has gained traction as an essential 21st century learning skill (Wing J. M., Computational Thinking, 2006). CT draws from computer science fundamentals; however, few definitions of CT give a succinct synopsis of what fundamentals CT includes. This is an important distinction to make as computer science standards are being formed for K-12 education in the United States. CT in K-12, as well as higher education, gives students a firm foundation for a higher level of thinking not only in computer science, but also a plethora of other STEM fields. As Barr and Stephenson state, an interpretation or definition of CT "must ultimately be coupled with examples that demonstrate how computational thinking can be incorporated in the classroom [2]."

The novel contributions of this work center on a clear synopsis of CT and what computer science principles are included therein as described in Computational Thinking: Concepts, Practices, and Perspectives. This report also includes new surveys for qualitatively assessing student self-efficacy in CT. Current survey work is presented in the Summer STEM Institute 2015 chapter, followed by adaptations of that survey in Summer STEM Institute 2016. Novel curricula, supported by the student self-efficacy survey, are also presented, designed for K-12 outreach and embedding CT. Apart from CT in K-12, results from an experimental study incorporating CT into an undergraduate physics course is presented. A new method for automatically extracting semi-structured assessment data in physics is presented. Although this methodology is first being applied to the domain of physics, this approach is designed to be applicable across domains for future use in CS. This also provides capability for advanced analytics with assessment data. Finally, future work, including proposed work for quantitatively analyzing CT by going beyond the portfolio and static analysis of completed or compiled programs is presented. This research is published in part by

(Weese, 2016; Weese & Feldhausen, 2017; Weese & Hsu, 2016; Weese, Feldhausen, & Bean, 2016). The goals of this research are aimed at answering the following questions:

1. Can the attitude surveys created reliably assess student ability in computational thinking? This is answered by the survey analysis in the summer STEM Institute 2015 and 2016.
2. Are the underlying computer science principles being taught through outreach curricula reflected in student performance? Similarly, this mostly comes out in the analysis of survey work done for the summer STEM institutes. Some anomalies were found and are described in the findings.
3. Is there a link between self-efficacy in computational thinking and self-efficacy in problem solving skills? An initial answer to this question was found during the 2016 summer STEM Institute; however, further studies will need to be done to thoroughly answer this research question.
4. Does introducing CT to physics students without any training in computer programming lead to gains in students' abilities in computational thinking? Initial experiences and recommendations are made, but a more formal and larger experiment will be needed to study the effects of CT on students in the physics classroom.

Chapter 2 - Computational Thinking

The ability to define, incorporate, and assess computational thinking is the main purpose of this research. The Next Generation Science Standards (NGSS) (NGSS Lead States, 2013) and Common Core Standards (CCSS) (National, 2010) have influenced a STEM movement with ever-increasing needs for computational thinking. CT has been defined in a variety of ways, but discussion between researchers on what the definition of CT should include was born from Wing's vision to make CT a fundamental skill for everyone, not just computer scientists (Wing J. M., Computational Thinking, 2006). *Computational Thinking: A Digital Age Skill for* emphasizes the importance of Wing's vision and notes the significance of CT as a vital 21st century skill, which is noted in the P21 Framework for 21st Century Learning (P21 Partnership for 21st Century Learning, 2015). Not all definitions of CT are created equal; however, among various definitions (Barr & Stephenson, 2011; Barr, Harrison, & Conery, 2011; National Academy of Sciences, 2010; Sengupta, Kinnebrew, Biswas, & Clark, 2013; Wing J. M., Computational Thinking, 2006), abstraction and algorithms are two main concepts that everyone agrees upon. More so, Barr and Stephenson point out that whatever an educator's interpretation or definition of CT includes, it "must ultimately be coupled with examples that demonstrate how computational thinking can be incorporated in the classroom" (Barr & Stephenson, 2011). This leads to a conclusion that educators do understand the importance of CT skills; however, they lack a clear, practical definition with established pedagogy to help bring CT to their classrooms (Barr, Harrison, & Conery, 2011; Barr & Stephenson, 2011). Solving this problem would also remove the preconceived notion that computational thinking is CS or computer programming, and show educators that CT is a skill used across many disciplines (Wing J. M., Computational Thinking, 2006; Sengupta,

Kinnebrew, Biswas, & Clark, 2013; Bennett, Ioannidou, Repenning, Kyu Han, & Basawapatna, 2011; Repenning, et al., 2015).

Defining Computational Thinking

"Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing J. M., Computational Thinking, 2006). As mentioned earlier, this is not intended to be interpreted as equating computer science outright to computational thinking; rather the essence of CT comes from thinking like a computer when faced with problems from any discipline (Grover & Pea, 2013). In 2011, Wing expanded the definition of CT, mentioning that CT is "the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information processing agent" (Wing J. M., 2011). The inclusion of intelligent agents in what embodies CT creates a pathway to fostering CT in multiple disciplines by means of scientific simulation and real-world problem sets. This is the key point when defining CT: the ability to connect core computer science concepts to non-computer science domains.

In recent years, countries have begun to incorporate computing in the K-12 education system. From these curricula and reports, succinct definitions of CT provide broader impacts in terms of education. The U.S.A. does not have country wide CS education standards; however, some organizations, like the CSTA, have been dedicated in creating CS standards that incorporate CT, although their standards are not widely officially adopted. The CSTA defines CT as solving problems in a way that can be implemented with a computer, including concepts like abstraction, data, recursion, and iteration (Seehorn, et al., 2011; Seehorn, et al., 2016). CSK12 is also a collaborative effort by CS educators and professionals aiming for

creating a standardized CS curriculum in the U.S.A. Many states have begun to adopt their own CS standards; however, there is no uniformity among them. Countries like the United Kingdom have had a large push for reforming their educational system to incorporate computer science and CT. The U.K. has had CS standards in their education system in the past encompassing ICT (Information and Communication Technology). However, in a report by the Royal Academy, the ICT curricula has left students only coming out of the school system with digital literacy skills and not a true handle on computer science (Royal Society, 2012). The report encourages to move away from ICT and to use a combination of information technology, digital literacy, and computer science. The report also emphasizes the need to incorporate CT, defined as "the process of recognizing aspects of computation in the world, and applying tools and techniques from computer science to understand and reason about both natural and artificial systems and process" (Royal Society, 2012). This encompasses the vision of CT, drawing from traditional computer science to encourage new ways of thinking about the world.

Computational Thinking: Concepts, Practices, and Perspectives

Computational thinking can be expanded by defining in terms of concepts, practices, and perspectives (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012). CT concepts have been a popular target for research and curricula development (Barr, Harrison, & Conery, 2011; Grover & Pea, 2013; Sengupta, Kinnebrew, Biswas, & Clark, 2013; Royal Society, 2012; National Academy of Sciences, 2010; Seehorn, et al., 2011); however, the concepts vary across domains. Some, such as Brennan and Resnick, present CT concepts as referenced in their problem domain (Scratch), while others, such as the Computer Science Teachers

Association (CSTA), present CT concepts for application across K-12 curricula. Even from within the general domain application, what authors include in their definition of CT concepts vary (Grover & Pea, 2013; Seehorn, et al., 2011; Bort & Brylow, 2013; Chuang, Hu, Wu, & Lin, 2015; Google, 2016). Brennan and Resnick define computational concepts (also referred to as CT concepts as the "concepts that designers employ as they program." To encompass more fields, CT concepts are generalized as the usage of one of the following computer science principles in solving a problem: algorithmic thinking, abstraction, problem decomposition, data, parallelization, and control flow. These concepts are first generalized, and when necessary, broken into subtopics. Many computer science principles could arguably be included as or fall under one or more of these CT concepts; however, for purposes of this research, domain-independent language is used where possible, and definition of the concepts are kept finite.

Algorithmic Thinking – Algorithms are sequences of steps used to solve anything from simple to complex tasks. Procedural knowledge, such as algorithms, apply to a plethora of domains, like chemistry, math, and physics. An example applicable to everyday life is cooking. Recipes provide a well-formed plan for cooking food. This is the quintessence of algorithmic thinking: the ability to clearly define a sequence of steps. One sub-concept that is included under algorithmic thinking is operations. *Operators* have been presented as a CT concept in Scratch (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012), but have been absent in others. Operations are defined in context of algorithmic thinking as logical, mathematical, symbolic, and textual expressions. While expressions contain some layers of

traditional abstraction and usage of data concepts, they require a systematic, step-by-step approach to implement and solve.

Abstraction – An abstraction is a generalized representation of a complex problem. Often, this representation is not holistic in nature. Abstraction allows one to ignore certain aspects or details in order simplify and understand a difficult problem, although what gets included or excluded in the abstraction should be done with great care. The traditional definition of abstraction differs from that of computational abstraction, such that computational abstractions may generalize beyond mathematical and physical science properties and tend to be more symbolic in nature (Wing J. M., Computational Thinking and Thinking about Computing, 2008). Wing continues to show that the abstraction process involves multiple layers. A cake itself, for example, is an abstraction that contains several interconnected layers of abstractions. The layers of sponge, separated with layers of frosting or other fillings, are results of detailed recipes by various combinations of ingredients and cooking. The relationship between each layer must also be understood, otherwise the cake may not be structurally sound or taste very good. This idea leads us to argue that the "nuts and bolts of computational thinking are defining abstractions, working with multiple layers of abstraction, and understanding the relationships among the different layers" (Wing J. M., Computational Thinking and Thinking about Computing, 2008).

Problem Decomposition – Problem decomposition involves breaking down a problem into smaller, more manageable parts where each part can be solved independently of each other. These can then be recombined to solve the problem as a whole. Problem decomposition is not to be confused with abstraction. Abstraction can help in decomposing problems, because each decomposed part, while smaller and more manageable, is not

guaranteed to be conceptually simpler. Abstraction is intended only for generalizing or simplifying, whereas problem decomposition is used to separate a problem into independent sub problems.

Data – Data can reference an extremely wide variation of information, things, and ideas. Data in computational thinking can be broken down into three parts: collection, representation, and analysis (Google, 2016; Chuang, Hu, Wu, & Lin, 2015; Bort & Brylow, 2013; Seehorn, et al., 2011). Data collection is the act of gathering information. For example, students could measure the height of everyone in their class and record it in a variety of mediums, like on a digital spreadsheet or a simple piece of paper. This leads into data representation: the depiction and organization of data (Google, 2016). Students could choose to use metric, standard, or an arbitrary form of measurement. The heights can also be organized, maybe by being sorted or even grouped by male or female. This prompts data analysis: gaining an understanding or developing insight from data (Google, 2016). Students should formulate questions to answer from the heights they collected. The way they chose to represent their data can have a direct impact here as well. If the question is "Who is the shortest and tallest person in the class?," it will take longer to discover if the data is not organized. Some data that is collected may be irrelevant to the questions that need answered. Analysis of the data, which can be a primitive form of abstraction, brings focus to data that is relevant.

Parallelization – Parallelization is the "simultaneous processing of smaller tasks from a larger task to reach a common goal" (Google, 2016). For example, a restaurant will often have different cooks for different parts of the menu (a person that fries food, one who grills, one who bakes, and different people for prepping the food before it gets cooked). Parallelization

also includes simultaneous processing of the same task to improve efficiency. In the restaurant example, adding more than one person at each station can improve the speed at which food can be sent from the kitchen. Synchronization is also an import concept in parallelization. Synchronization is the coordination of the tasks that are being executed in parallel.

Control Flow – Without control flow, algorithms are simple, linear problem solving traces. In other words, control flow directs an algorithm's steps and when an algorithm completes. This includes allowing the algorithm to repeat steps several times, complete steps under certain conditions (based on observed data or events), skip steps, or even stop before all steps are completed. Control flow can make algorithms more efficient by reducing duplicated or redundant steps, and ultimately, this can improve the way problems are solved using algorithms, giving one the ability to solve a problem in a non-linear manner.

Brennan and Resnick argued that assessing CT only by computational concepts does not provide proper representation the learning and participation that occurs (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012). Thus, computational practices and perspectives are included in assessment (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012; Seehorn, et al., 2011; Royal Society, 2012). "Computational practices focus on the process of thinking and learning, moving beyond *what* you are learning to *how* you are learning" (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012). From this framework, practices include: being incremental and iterative, testing and debugging, and reusing and remixing.

- Being incremental and iterative: Instead of finishing a problem all at once, being incremental or taking the problem one step at a time can decrease the difficulty of complex problems. This is in relation to the engineering design process where problems are cycled through multiple times, often making incremental improvements before settling on a final solution.
- Testing and debugging: Testing and debugging refers to troubleshooting a problem solution. Often, the solutions are not perfect the first time they are implemented. Troubleshooting is usually test driven, where a set of standards or conditions are checked each time something in the solution is modified.
- Reusing and remixing: Many problems are rooted in ones that have already been solved. This practice refers to taking an existing solution to a similar problem and adapting it to solve your own.

Abstraction is also included as a practice from the framework done by Brennan and Resnick; however, as abstraction is a concept that is rooted deeply in many of the CT concepts, it is included as a CT concept in this research, rather than a practice. Computational perspectives are the way students understand the effect of computation and technology in the world around them, as well as with their relation to others and themselves (Brennan & Resnick, Using artifact-based interviews to study the development of computational thinking in interactive media design, 2012). Brennan and Resnick address the following perspectives:

- Expressing: Especially with visual languages, computer science provides people with a way to interact through various media. This perspective looks at when someone sees computation as a medium. This allows students who grasp this perspective to use computation as an opportunity to create, express, and implement their ideas.

- Connecting: This perspective points out the value of networking, where learning is social practice. This gives further motivation for students and creators, such that see the value of creating with and for others. Whether it is by entertaining, engaging, or equipping (examples solutions or parts of solutions meant to be reused) others, having some form of connection with others makes the learning process more valuable.
- Questioning: The idea of this perspective is to get learners to inquire and investigate the role that computation has in their life and world around them. It prompts students to go from being able to use technology to considering how those technologies and devices work and how they can take part in it.

Problem Solving and Computational Thinking

Incorporating CT into existing curricula and programs is not always an easy task, especially in K-12 where problem solving (PS) skills are traditionally emphasized. Most non-computer science educators see the importance of CT; however, it is difficult to distinguish CT from PS so it is common for CT to be seen as PS within computational domains. Many of the CT concepts and practices described in the previous section are not necessarily unique to computer science nor are they restricted to being used with technology (computers, electronics, etc.). They have roots in many other disciplines, as well as problem solving; however, using these concepts and practices collectively in mutually-reinforcing ways to solve problems is typically limited to computer scientists.

Authors Voskoglou and Buckley define PS as "an activity that makes use of cognitive or cognitive and physical means to overcome an obstacle (problem) and develop a better idea of the world that surrounds us" (Voskoglou & Buckley, 2012). This definition originates from Polya: "solving a problem means finding a way out of a difficulty, a way around an obstacle,

attaining an aim that was not immediately understandable" (Polya, 1973). Critical thinking, though there are a plethora of definitions, can be defined as "the systematic evaluation or formulation of beliefs or statements by rational standards" and can be viewed as "how we think" not "what we think about" (Vaughn, 2008).

Critical thinking is often included when discussing problem solving; however, there isn't a consensus among philosophers and educators on what the relationship between the two really is (Giannakopoulos, 2012). The discussion ranges from wondering if critical thinking is a form of problem solving, a part of problem solving, should it include problem solving, or is it complementary to problem solving (Papastephanou & Angeli, 2007; Giannakopoulos, 2012). For this work, critical thinking is treated as its own distinctive process, but complementary to problem solving. As such, one who has sufficient critical thinking skills and knowledge can solve a problem. These aspects can be seen in Figure 1 with addition of cognitive processes.

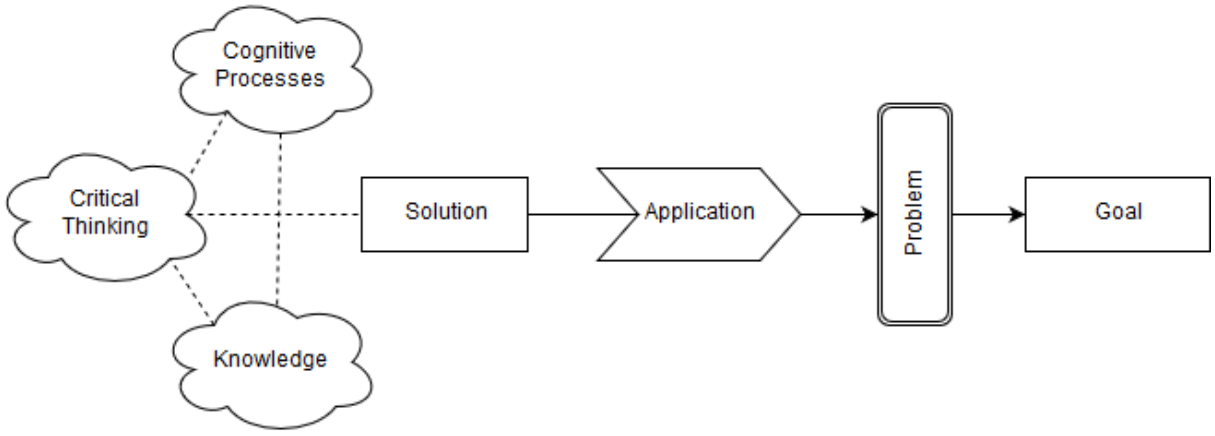


Figure 1 Conceptual model of the aspects and processes involved in problem solving, adapted from (Giannakopoulos, 2012; Voskoglou & Buckley, 2012).

Cognitive processes are leveraged to inform problem solving process. But like critical thinking, these processes are distinct, high-ordered levels of thought. Problem solving uses specific strategies in developing solutions that overcome a problem to reach the goal state. These strategies are informed by a person's knowledge, critical thinking, and cognitive

abilities. Wang and Chiew offer a summative collection of problem solving strategies can be seen from:

- Direct facts – finding a direct solution path based on known solutions.
- Heuristic – adopting rule of thumb or the most possible solutions.
- Analogy – reducing a new problem to an existing or similar one for which solutions have already been known.
- Hill climbing – making any move that approaches closer to the problem goal step by step.
- Algorithmic deduction – applying a known and well defined solution for a problem.
- Exhaustive search – using a systematic search for all possible solutions.
- Analysis and synthesis – reducing a given problem to a known category and then finding particular solutions (Wang & Chiew, 2010).

Arguably, some of these strategies could be considered critical thinking and vice versa. The same could also be said about computational thinking. For example, hill climbing can be related to being incremental, analogy to abstraction, algorithmic deduction to algorithmic thinking, etc. It is important to note, however, that these comparisons can be true only within a finite set of defined problems. Abstraction, for example, will typically only be used in one or two layers in most disciplines and cognitive processes. However, CT typically emphasizes many layers of abstraction, as well as the relationship between them. For example, the OSI model used by all internet-connected technologies involves *seven* distinct layers of abstraction. The real essence of CT is that it is about the idea and way of thinking, not the application, product, or artifact. Voskoglou and Buckley describe CT as a hybrid mode of thinking, leveraging logical, abstract, and constructive thinking. By synthesizing critical thinking and

existing knowledge, these modes of thinking are essential in solving real-life, complex problems (Voskoglou & Buckley, 2012). Thus, PS can be described as an activity that *combines* various components of cognition (Voskoglou & Buckley, 2012; Green & Gilhooly, 2005). CT, as well as critical thinking, can be seen as a prerequisite to overcoming problems in certain problem spaces. Concluding, CT is treated as a distinctive cognitive process used to overcome types of problems in conjunction with existing knowledge and critical thinking skills.

Computational Thinking Assessments

Clearly and definitively defining computational thinking within existing methods and domains (as with problem solving) remains to be an open problem. As such, there is no clear pathway to measuring computational thinking. Several approaches, however, exist, ranging in theme and methodology. Many focus using evidence-centered design (Snow, Haertel, Fulkerson, Feng, & Nichols, 2010) when creating assessments, while others study the psychological state of students using cognitive science and self-efficacy (Ramalingam, LaBelle, & Wiedenbeck, 2004). Developed curricula vary for assessments, but video games seem to be a very common theme as it provides a high level of interest for students, especially in K-12. For example, Wilson evaluated an eight-week course on game programming in Scratch for primary grades in Scotland (Wilson, Hainey, & Connolly, 2012). Games were encoded under three main categories: programming concepts, code organization, and designing for usability. Each of these contained sub categories which were coded for the presence/absence of or to the extent of which that sub category was used ranging from zero to three. Another study looked at a semester long gaming course using Alice by analyzing the presence of abstraction, control structures, and events. Denner et al. also looked for presence

of computer science concepts in an after-school program for game programming in Stagecast Creator, a rule-based visual programming language (Denner, Werner, & Ortiz, 2012). Repenning et al. measured student's learning of CT as patterns rather than concepts using Agent Sheets (Koh, Basawapatna, Bennett, & Reppening, 2010). This led to an automatic analysis tool using latent semantics to determine student growth in CT. Recently, the same group created a system entitled Real-Time Evaluation and Assessment of Computational Thinking (REACT), a real-time assessment tool allowing teachers to get immediate feedback on what students are struggling with or where they are succeeding (Basawapatna, Repenning, & Koh, 2015). Seiter developed the Progression of Early Computational Thinking model (PECT) (Seiter & Foreman, 2013). The PECT model combined evidence of programming concepts in Scratch projects with levels of proficiency (basic, developing, and proficient) in a set of design patterns to understand student ability in CT. Other works focused on assessing concepts through rubrics (Franklin, et al., 2013) and creating and validating traditional computer science assessments within the visual programming domain (Buffum, et al., 2015).

Apart from evidence-centered assessments, Yadav et al. created an open-ended questionnaire, as well as an attitude survey to understand if introducing computational thinking material in pre-service education courses influenced pre-service teachers' understanding of CT and attitudes toward computing (Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). Bean et al. developed a two-part self-efficacy survey for a pre-service teacher training program (Bean, Weese, Feldhausen, & Bell, 2015). The first measured the pre-service teachers' confidence that they are capable of incorporating computer programming into their classroom, as well as recognizing how programming concepts relate to NGSS and CCSS. The second survey delved into their self-efficacy in their understanding of CT concepts in relation to programming. In

another related project, Bell conducted a CT intervention as part of an art-based program component of a summer STEM institute; his experimental approach served as a basis for this work (Bell, 2014).

Chapter 3 - Outreach

Outreach programs vary in theme, as well as target audiences, ranging from primary and secondary education to in-service and preservice teachers. These programs are essential for both student and professional development, exposing them to topics and training they otherwise would not have access to. This chapter outlines effective teaching methods and curricula in STEM, targeting computational thinking. Some, such as Brennan and Resnick, present CT as referenced in their problem domain (Scratch), while others, such as the Computer Science Teachers Association (CSTA) and the College Board for AP CS, present CT concepts for application across K-12 curricula (Seehorn, et al., 2011; College Board, 2016; Seehorn, et al., 2016). These definitions of CT that are described in detail in Chapter 2 serve as the foundations for the curricula and instruments presented in this chapter. Effects of a STEM institute for 5^{th}-9^{th} grade students are presented, as well as the adaptation for the next iteration of that institute. Outreach efforts with the Girl Scouts of America and a STEM program called *Starbase* are also discussed.

Visual Based Programming

Visual based programming tools have become largely popular due to their ease of use for beginner programmers in not only K-12, but also higher education. These block-based programming languages have made their way into many Science, Technology, Engineering, and Math (STEM) outreach programs to train both students and educators. Code.org (Code.org, 2015) has been a major leader advocating for CS in the K-12 classroom by

providing materials for educators, as well as providing interactive tutorials on programming using Blockly (Fraser, 2015). Most recently, Code.org released two new programming tutorials themed around the popular game, Minecraft, and up and coming release of *Star Wars VII*. These were in preparation for the Hour of Code, an event encouraging children and adults alike to program for one hour during computer science education week. Touch Develop, Agent Cubes, codeSpark, Lightbot, Tynker, and Scratch are popular visual programming platforms that also developed Hour of Code exercises for kids as young as four years old.

The Hour of Code has been the largest computer science outreach effort with nearly 200,000 events around the world and over 240 million participants since its inception. CS outreach programs have a large range of focus. Scalable Game Design (SGD), for example, developed CT tools using AgentSheets and AgentCubes which enabled middle school students to develop video games (Repenning, et al., 2015). The tools increased student understanding of CT concepts, which then allowed them to apply their new skills on scientific simulations, not just video games. While still using block-based programming tools, another outreach program, GK12 INSIGHT, worked with K-12 teachers and graduate students to incorporate embedded systems and sensor technology with emphasis on CT in K-12 curriculum (Neilsen, Shaffer, & Johnson, 2015). Other researchers have focused on creating various outreach programs, such as Computer Science for High School (CS4HS), that emphasize training teachers in computer science (Blum & Cortina, 2007). Another CS4HS workshop focused on the measurement of the ability of teachers to incorporate CT concepts into lesson plans (Bort & Brylow, 2013). While Bort & Brylow developed a rubric for general CT concepts (i.e. abstraction, algorithms, etc.), they did not measure the teacher's own understanding of CT. Bean et al. created a workshop targeting pre-service teachers (Bean, Weese, Feldhausen, &

Bell, 2015). During this workshop, pre-service teachers went through Scratch exercises to learn not only CT concepts, but how to incorporate CT into their future classrooms. Similarly, Bell created a Scratch curriculum for Music in a STEM outreach program for 5^{th}-9^{th} grade students (Bell, 2014). Pre-service teachers also participated in this program by observing and assisting in the Scratch activities (Bell, 2014).

Methods

While the interventions vary in theme, both center on the same learning and scaffolding theory. One of the major challenges of teaching CT concepts through computer programming in both K-12 and higher education environments is that students quickly become overwhelmed with learning a new language. By starting with text-based languages, beginning programmers spend more time struggling with the syntactic structure of programming languages instead of learning the core concepts like algorithms, abstraction, and data analysis. By using Scratch, a block-based language, students can learn the language quickly. This provides more focus on teaching CT concepts, rather than giving drawn out instructions on how to use the programming language. This is especially important because, like most outreach programs and school districts, time to work with students about CT and CS is extremely limited.

Using a block-based programming language effectively reduces the students' cognitive load. Cognitive Load Theory (CLT) (Plass, Moreno, & Brunken, 2010) supports the hypothesis that an individual's ability to learn is compromised when the intake of a learning task exceeds their working memory capacity. Morrison et al. developed sub goal labels in worked examples which reduced cognitive load in text-based programming (Morrison, Margulieux, & Guzdial, 2015). Aa similar approach is used by using seed Scratch programs

for the intervention activities. The seed programs are partially completed. This skeleton outlines the structure of the program and complete low learning potential portions of the program (for example, have sprites, costumes, and backgrounds already created), which focuses on any new CT skills or computer science principles in a limited timeframe while reducing cognitive load.

The cognitive load continues to be reduced, specifically extraneous load (Morrison, Margulieux, & Guzdial, 2015), by putting scaffolding in place for each activity or project in the interventions. Scaffolding is a support structure put in place for learners to accomplish tasks that they could otherwise not complete (Bliss & Askew, 1996). The approach of instructional scaffolding, which correlates to programming tutorials, is used. However, as Repenning notes, direct instruction can limit student motivation, especially in females (Repenning, et al., 2015). Problem-based Learning (PBL) (Savery, 2009), alongside Inquiry-based Learning (IL) to keep students motivated, are used. PBL is a 'learner-centered approach that empowers learners to conduct research, integrate theory and practice, and apply knowledge and skills to develop a viable solution to a defined problem." (Savery, 2009). IL is like the PBL approach, but in IL, the facilitator acts as a provider of information. In PBL, this is left to the student. Kirschner argues that PBL and IL do not provide enough guidance for students to learn based on human cognition (Kirschner, Sweller, & Clark, 2006); Hmelo-Silver refutes this statement by providing evidence that PBL and IL have enough scaffolding to be effective learning practices (Hmelo-Silver, Duncan, & Chinn, 2007). The coding activities in day 1 use mostly direct instruction scaffolding (step-by-step instruction of what blocks to use), but as the class progresses into later projects, guided discovery or inquiry-based learning is used, which has been shown to increase student abilities in scientific literacy (Wu & Hsieh,

2006; Gormally, Brickman, Hallar, & Armstrong, 2009) as well as students' motivation to learn (Repenning, et al., 2015). As the intervention progresses, PBL is also utilized. This allows the researchers to let students who are progressing quickly in activities to work ahead or on their own while assisting others who are struggling. By asking the students questions about the task, often relating it to real world or previous classroom experiences, they will often discover how to use the blocks available to them in Scratch to solve the task. Throughout the intervention, scaffolding is removed until the last day where students are tasked with their final project.

The goal of the methodology is to maximize the increase in student self-efficacy. Self-efficacy can be defined as "an individual's belief that they can accomplish a particular task" (Bean, Weese, Feldhausen, & Bell, 2015). Measuring self-efficacy relative to CS and even more so in CT is required, because there does not yet exist any widely-adopted standardized assessments which measure student progress (apart from AP CS). Bandura notes that self-efficacy can by improved through enactive attainment, vicarious experiences, verbal persuasion, and psychological state (Bandura, 1982). PBL enables the students to achieve tasks on their own without direct instruction. This relates to enactive attainment (individual mastery of skills), although the problems are structured carefully so that they are not too easy (students will get bored) or too difficult (increased anxiety). This is also referred to as the zones of proximal flow and development (Repenning, et al., 2015). Verbal persuasion occurs in the intervention through IL. Though IL is indirectly guiding the student (asking the right questions), students are convinced that they can solve the tasks at hand, whether it's difficulty with a CT concept or a technical problem with placing the correct blocks in Scratch. Vicarious experiences are achieved through group activity. During activities, students are encouraged to

talk to their neighbors about how they solved the programming tasks, and in others, students are partnered up for group projects. Finally, a stable psychological state is achieved through the scaffolding and use of a block-based language to reduce the cognitive load. By designing the interventions around Bandura's methods of improving self-efficacy, powerful and achievable learning experiences are crafted.

Summer STEM Institute 2015

This section, published in part by (Weese, Feldhausen, & Bean, The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking, 2016) describes a summer STEM institute where the Manhattan-Ogden Unified School District 383 has partnered with the Department of Education at Kansas State University. This program lasts four weeks and is designed to expose STEM careers and subjects to 5th-9th grade students through hands-on activities. The program covers a large range of areas, including robotics, computer programming, agriculture, food science, unmanned aerial vehicles, clean energy, and construction science. Educators, who are experts in the subject matter, are paired with small groups (2-4) of pre-service teachers to run each class (maximum size of 18). This allows pre-service teachers to get practical, hands-on experience, as well as to learn new STEM activities to include in their own future classrooms. This also gives an excellent teacher to student ratio, providing a one-on-one learning experience for program participants.

This chapter will cover new interventions with similar pedagogy implemented for the institute which focuses on video game design and robotic agents. Each intervention used the visual programing language Scratch (Resnick, et al., 2009) as a tool to seed CT and CS concepts in both institute participants and pre-service teachers. A self-efficacy instrument used to measure STEM experiences, 21st century learning skills, and CT is described. The

importance of this research is to discover whether past STEM activities and experiences will transfer to student self-efficacy in CT or not, as well as develop a method for delivering and measuring CT skills in the K-12 environment.

Mission to Mars

Students in the lower grade levels (5th, 6th, and 7th grade) attended a program called Mission to Mars. The goal of this intervention design was to introduce students to CT through many different activities revolving around tasks that must be completed to send an autonomous rover to Mars. Each session of the program consisted of four days of activities (each day being 3 hours long).

The first day consisted primarily of an introduction to the Scratch visual programming language. The students were led through many short activities to familiarize themselves with the language, culminating in a challenge to draw regular polygons (Bean, Weese, Feldhausen, & Bell, 2015) using methods very similar to the turtle graphics features of the classic Logo programming language. As the students slowly built shapes with more sides, concepts such as iteration, variables, user input, and mathematical operators were introduced, leading to a generalized program that could draw any regular polygon. This program also demonstrated a fundamental theory of calculus.

The second day focused on using computers to simulate real-world ideas. The students began by playing with human-powered compressed air rockets (Stomp Rockets). While doing so, they plotted the distance each rocket traveled and discussed reasons for the wide variance of results. This led to a discussion of the scientific method, independent and dependent variables, and how to design an accurate experiment. Afterwards, students were led through an activity to simulate a rocket's trajectory in Scratch, using the launch angle as the independent

variable. With that knowledge, students were introduced to high-performance computing as a way to solve even bigger problems, such as the trajectory of a real rocket, and were given a guided tour of a nearby supercomputer. Students then learned how to create a simple acceptor finite state machine that accepts a secret key though a series of clicks.

The third day introduced the concept of artificial intelligence (AI). First, students were led through a project to recreate three of the four enemy AI ghosts from the classic Pac Man arcade game. In doing so, they were introduced to a two-step artificial agent pattern of perceiving the environment and acting based on that perception. Following that, students were assigned an activity to explain how more complex AI, such as neural networks, can be trained. After completing that activity, the students were introduced to the final project: building an AI for an autonomous Mars rover. The concept was first shown to them as a game, where they were challenged to get the highest score possible. This required planning ahead to find the best path and learning how the rover operates. These activities drew on many areas of CT, including modeling and simulation, abstraction, and data representation.

On the fourth day, the rover was re-introduced as a game, but this time the rover could only see the squares immediately adjacent to it. This required students to "sense" their surroundings and act based on limited information, just as the rover would. This helped reinforce the "perceive" phase of an artificial agent and forced the students to adjust their thought process to match that of an algorithm. Finally, the students were given a rover project that allowed them to build an AI following the perceive-and-act model previously used. They worked independently but with some guidance on how to build the best rover AI possible, and compared their results with other students.

Game Design

The second intervention was for the 8th and 9th grade students. This intervention focused on video game design within Scratch. While game design contains a significant amount of established theory, the focus was only on a small subset of common principles of game design inspired by *100 Principles of Game Design* (Despain, 2013) and a popular YouTube series (snomaN, 2015). While the delivery focus of this intervention was game design principles, the development of different games in Scratch was used to teach CT concepts. Like *Mission to Mars*, this program consisted of four days of activities.

Day one began with an introduction to game design principles. These consisted of seven principles: (1) Principle of isolation: introducing new elements in a way that allows players to familiarize with new enemies or mechanics before they are set in a real situation. (2) Principle of accomplishment: gives players a sense of motivation and direction either through story progression or the mastering of skills. (3) Teach without teaching principle: help players learn by doing instead of relying on step-by-step tutorials. (4) Growing stronger principle: a game storyline can often be rewarding alone; however, progression can be improved by letting the player grow stronger and accomplish tasks that they could not earlier in the game. (5) Silent storytelling principle: allow the player to experience the story for themselves instead of having it spelled out. (6) Hidden reward principle: give the player extras (bonus levels, collectables, etc.) to add an extra feeling of accomplishment beyond the original gameplay/story. (7) Balance principle: gameplay must have a good balance between boredom and anxiety to keep the player interested and coming back. These principals were chosen due to their relation to educational theory and how the scaffolding is constructed. Examples of these principles were discussed in popular video games. Students were also asked to give

examples of the principles from games that they play at home. This discussion was followed by an introduction to scratch using shapes as mentioned in day one of the *Mission to Mars* intervention. Students were then asked to split off into pairs or groups of three to brainstorm their own game for as a final project of the course.

The second day focused on introducing basic AI concepts, an important aspect of video games. Students were asked to think about what it means to be intelligent. Most responses tended to be things like "smart." After describing intelligence as reasoning, problem solving, ability to construct knowledge, planning, learning, and perception (of which all relate back to the core concepts of CT), students were presented with the Turing test and how computers could be considered "intelligent." The importance of AI in video games is emphasized, and then, the first game tutorial called Cat and Mouse was started. This is a partially completed game where students are walked through implementing a basic AI for a cat that chases a mouse, the player, which tries to eat pieces of cheese. After they had a working game, students were presented a problem to improve the AI to exhibit more complex behavior. As a follow up, students were worked with to complete the starter AI for the game *Strikers 1945*. The day ended with time for students to complete storyboards for their final project.

The theme of day three was dungeon crawlers, a classic game style. To demonstrate this, students were given a starter project for *One Tap Quest*, a simple, yet popular dungeon crawler/RPG. This game was used to illustrate all the game design principles taught since the first day. *One Tap Quest* requires only a single click from the player and their hero starts off on a quest through a randomized set of enemies to slay for experience and power-ups to collect before reaching the boss. Researchers walked students through setting up randomization of the first level of monsters. They were then tasked with adding another level of monsters, as

well as a power-up. The rest of the day was left for students to work in their group on their final project. Before students left for the day, a discussion was led on career options in the video game industry.

The final day was reserved time for groups to work on their projects while researchers walked around to assist. At the end of the day, groups got up in front of the class to demonstrate their games and describe what game design principles they used. Groups could use any of the seed projects used any of the previous days, as long as they added additional content or mechanics. Some groups did use the seed projects, but most designed their own game and used what they learned from programming the seed projects as the basis for their mechanics. To encourage the students to continue to collaborate, all projects from each week were added to a Scratch studio.

Instrument Design

A hybrid instrument was developed, combining the questions 6-17 from the Self-Efficacy for Computational Thinking (SECT) survey (with the addition of a question about Boolean operations) (Bean, Weese, Feldhausen, & Bell, 2015) with questions extracted from the math (27, 28, 31,and a new question: "I can apply math concepts to other subjects"), science (35-37, 40, 42), engineering and technology (44 – changed products to things, 45, 50-52), and 21st century skills (38, 44, 46, 48, and a new question "I am confident I can manage my time wisely when working in a group") sections in a survey built for measuring attitudes towards STEM (Faber, Unfried, Corn, & Townsend, 2012). The new questions were added for better coverage of the interventions. Additional questions were asked about the student's previous experience in STEM activities (if they attended this institute before or any other STEM-related outreach activities), as well as whether they had previous experience

programming in the Hour of Code, Scratch, Blockly, TouchDevelop, text-based languages, or any other computer language. A teacher survey was also created in a similar fashion by extending the Teacher Self-Efficacy for Computational Thinking (TSECT) survey from (Bean, Weese, Feldhausen, & Bell, 2015) to include teachers' background in STEM activities, interest in teaching STEM, and experiences with programming languages. The full surveys can be found in the appendix STEM Institute 2015 Survey.

	Mission to Mars				*Game Design*			Total
Grade Level	5th	6th	7th	Total	8th	9th	Total	
Week 1	0	8	5	13	8	2	10	23
Week 2	3	7	3	13	6	3	9	22
Week 3	1	9	6	16	6	5	11	27
Week 4	0	5	3	8	5	6	11	19
Total	4	29	17	50	25	16	41	91

Table 1: Number of students (after survey exclusion) in each intervention

Pre-surveys were administered online at the beginning of each week-long session. The post-survey (excluding initial background questions) was given on the last day of each session (day four) after ending discussions. While both the student survey and the teacher survey were optional, the student survey was administered during each session, and the teacher survey was only emailed each session. Out of 94 surveys sent to all educators and pre-service teachers involved with the institute, only 33 responded to the pre-survey and fewer than 10 responded each week for the post-survey. For this reason, results for the teacher survey are excluded from analysis. Student response rate (after exclusions) can be seen broken down in Table 1. Out of 101 student respondents, 7 were excluded for not taking the post-survey (absent those days), one was excluded for not taking the pre-survey (absent), and two were excluded for

incomplete surveys (the missing data in these responses were classified as MCAR). The reliability of the instrument was confirmed with a Chronbach's Alpha of .908.

Findings

Part of the instrument was to gather background information in STEM, including programming experience as seen in Table 2. Surprisingly, over 30% of students had been exposed to a text-based programming language. Over 59% of students had experienced some sort of block-based programming language, and half had participated in the Hour of Code. Less than half of those who participated in the Hour of Code (which is written using the Blockly language) knew that they were using Blockly. From students who had previously attended some sort of STEM program before the institute, 70% of them had used Scratch. This shows that most outreach programs in this geographic area highly favor the Scratch language. With more than 80% of students having used some programming language, it shows that all students are being exposed as much to computer programming at home or school as those who participated in outreach programs. However, the low level of exposure is reflected in the self-efficacy in CT concepts. Students who had previously attended outreach programs improved more in CT concepts such as algorithms, procedures, parallelization, data collection, and data representation, as shown in Table 3(this includes students who attended this institute before as well as those who attended other outreach programs). This hints that even though both samples of students had about the same amount of experiences using programming languages, STEM outreach programs have better success in seeding CT skills in students, compared to exposures in school or at home.

	Any Language	Hour of Code	Scratch	Blockly	TouchDevelop	Text-based	Other

No Previous Attendance in STEM Programs (37 students)	83.78%	51.35%	56.76%	18.92%	16.22%	35.14%	35.14%
Previous Attendance in STEM Programs (54 students)	81.48%	51.85%	70.37%	20.37%	18.52%	35.19%	31.48%
Overall – *Mission to Mars*	84.00%	52.00%	58.00%	18.00%	16.00%	32.00%	34.00%
Overall – *Game Design*	80.49%	51.22%	60.98%	21.95%	19.51%	39.02%	31.71%

Table 2 Programming experience before the interventions

Abbreviated Question	No Previous Attendance in STEM Programs	Previous Attendance in STEM Programs	Overall	
			Mission to Mars	*Game Design*
21st Century Learning—Math				
Math is my worst subject	-.132	0	-.117	.024
Consider a career that uses math	.264	.127	.156	.215
Perform well in other subjects, but not math	-.193	.050	-.125	.044
Apply math to other subjects	.058	.020	.129	-.079
Consider a career in math	.138	.151	.143	.149
21st Century Learning—Engineering				
Like to imagine creating new things	.036	.098	-.106	.291
If I learn engineering, I can improve things people use everyday	.191	.196	.165	.230

Would like to use creativity and innovation in my future work	.087	.199	.064	.262
Math and science together will help me invent useful things	.251	.064	.255	0
I can be successful in a career in engineering or technology	.113	-.039	-.021	.077
21st Century Learning—Leadership				
Lead others to accomplish goals	.501	-.023	.198	.181
Work well with others who have different backgrounds and opinions	-.033	.135	.097	.030
Make changes when things don't go as planned	.036	.098	.132	0
Manage my time wisely when working on my own	.027	.074	.060	.048
Manage my time wisely when working in a group	.088	.141	.153	.0798
Computational Thinking				
Executes a sequence of commands	.352	.632	.496	.546
Uses loops to repeat commands	.641	.785	.683	.779
Responds to events	.259	.539	.584	.231
Parallelism	.482	.656	.556	.621
Conditional commands	.498	.508	.582	.408
Perform math operations	.265	.387	.481	.162
Perform Boolean operations	.606	.626	.626	.608
Store, update, and retrieve values	.405	.550	.429	.568
Ask user for input	.292	.694	.537	.522

Iterative development	.331	.417	.322	.456
Frequent tests/debugging	.519	.481	.533	.452
Share and collaborate with programs	.337	.573	.445	.517
Break program into parts	.537	.412	.448	.482

Table 3 Comparison of effect sizes for the 21st century learning and CT focused questions

One could say as well that because the students who attended STEM programs previously had more exposure to Scratch (70.37% vs. 56.76%), they could move more quickly through the activities and focus more on learning CT concepts rather than the language itself. Students who had not attended STEM programs previously showed higher pre-survey self-efficacy than those who had. It is hypothesized that since they may not have been exposed to CT as much, this led to overconfidence, which is reflected in the amount of improvement when looking at post-surveys. This shows evidence of the Dunning-Kruger Effect (Kruger & Dunning, 1999). By comparison with the overall program, the two interventions were less distinguishable, though students in *Mission to Mars* had a strong improvement in self-efficacy in writing programs that respond to events and for being able to perform math operations in their programs. However, it was found that the mean pre-survey self-efficacy for *Mission to Mars* in these questions was much lower than that of the *Game Design* intervention, although the mean post-survey responses were nearly equivalent. This verifies that even though the topic of interest in each intervention is different (as well as the age groups), the end results for both are comparable.

When looking at 21st century learning skills, improvements were less noticeable as most of the students came in with high confidence in these areas. For example, over 80% of students came into the sessions highly confident in math, which led to little improvement. However, students who had previously not attended STEM outreach programs showed

stronger improvement for their value of math and science in inventing new things. Leadership also showed a strong improvement in these students, which reveals that the STEM outreach programs are doing well in improving student confidence in leading others to accomplish goals. Only weak improvements are present when comparing the two interventions, though *Game Design* had slightly stronger results in imagination and creativity. This is because the *Game Design* intervention offered more room for students to create and implement their own ideas in their final project video game.

Conclusions

Though both interventions differed in topics (video games vs. Mars rovers), they showed similarly strong improvements in student self-efficacy in CT concepts. This pedagogy shows that it has a positive impact how CT concepts are delivered through CS and computer programming at the K-12 level. Likewise, by expanding the survey done by Bell, more insight into specific CT concepts learned by students in a similar environment was gained (Bell, 2014). Furthermore, some lasting impacts that STEM outreach programs have on students who continue to stay active in science, technology, engineering, and math activities were revealed. These students who have participated in the outreach programs show greater capacity in improving their CT skills over those who have not. This important finding cannot be explained completely with the data collected, though conjectures were made, and warrants further investigation through revised instruments or longitudinal studies. Background survey questions also revealed that the STEM outreach in the areas (apart from this summer institute) does not have large participation by upper middle school and high school students (19 students in 5^{th}-7^{th} grade vs 4 students in 8^{th}-9^{th}).

Summer STEM Institute 2016

This section describes a continuation of the STEM program described in Summer STEM Institute 2015 and is published in part by (Weese & Feldhausen, 2017). The focus is on measuring the impact of two interventions on the program participants. Each intervention employed similar pedagogy and the Scratch (Resnick, et al., 2009) programming language; however, one relied heavily on computer science theory and Mars as a theme, while the other used micro controllers as the basis for the activities. The goals of this research continue from the 2015 STEM Institute and are as follows: 1. Develop effective curricula for improving student self-efficacy in CT, 2. Develop a reliable and effective way of measuring student self-efficacy in CT, and 3. Enforce the notion that CT is not PS, but a component of cognition.

Methods

Teaching programming can be a difficult task when involving students who have no background in foundational computer science skills. The curriculum emphasizes reducing cognitive load through scaffolded examples and the Scratch programming environment which eliminates complex syntax and programming errors. Problem-based learning and inquiry learning was utilized as described in Summer STEM Institute 2015, effectively improving student self-efficacy through vicarious experiences, verbal persuasion, enactive attainment, and psychological state (reducing cognitive load) (Bandura, 1982). From these methods, two different curricula were created as part of the summer STEM outreach program. The Saving the Martian (Mars) class was an intervention focused on 5th and 6th grade students and introduced CT using the Scratch programming environment. Many of the activities were modeled on situations or ideas taken from *The Martian*, by Andy Weir, to make them more interesting and exciting for the students. The Mighty Micro Controllers (MMC) class was an

intervention for 7th-9th graders focused on teaching CT through programming Arduino Uno micro controllers using Scratch, as well as a small exposure to the Arduino IDE and text-based language. Overall, the format of this class included guided examples on how to create certain circuits and programs, followed by problem driven exploration to help enforce programming, electrical, and CT skills. MMC heavily utilized pair programming. Each class consisted of four days of activities lasting three hours each.

Saving the Martian

Students were first to the Scratch environment and some basic ideas involved in programming. The main activity was to build a computer program that could draw an n-gon (regular polygon with n sides). At first, students were shown a sample program that drew three lines and were asked to modify it to create a triangle. From there, they further modified the program to draw a pentagon by adding more lines and adjusting the angles. As the number of sides grew larger, students were introduced to iteration as a way to reduce the amount of code in the program. In addition, mathematical concepts such as the geometric formula to calculate exterior angles of a polygon were used to determine the angles between each line. Finally, the students modified the program to accept user input, and were encouraged to try and draw shapes with many sides. As students determined that the shape would appear to be a circle, they were introduced to other concepts such as the fact that a smooth shape such as circle can be approximated by an n-gon with a sufficiently large number of sides, which is how most computer games represent such objects.

For the second day, students were introduced to sorting algorithms. First, the students participated in a sorting network activity where they followed lines on the floor that intersected. At each intersection, students would change their direction based on some factor,

with all students having a higher value going one direction and all students with a lower value going the other. Afterwards, students were given some hands-on experience by learning how to sort decks of playing cards into sorted order using several common algorithms such as insertion sort and bubble sort. While doing so, students recorded the number of steps needed to perform the algorithm, and their results were shared on the board. Students were then asked to use that data to determine which algorithm might be faster than the other, and then were introduced to a simple form of algorithm analysis which showed that both algorithms performed similarly. Students were shown the merge sort algorithm, and given information about why it takes fewer steps than the other two. As part of the discussion around bubble sort, students were also introduced to the concept of "swapping" two variables in computer programming by using a third temporary variable. After discussing the algorithms, students were lead through the first activity of the day to write a computer program in Scratch to perform the bubble sort algorithm. This built upon their knowledge of iteration from the previous day while adding in a conditional statement as well. For the second activity, students were given a situation from The Martian where the main character must determine how to grow enough food on the surface of Mars with limited resources. In the book, he creates water by a chemical reaction involving hydrazine (rocket fuel) and oxygen. For the activity, students were shown how to create a simple simulation program in Scratch to demonstrate how the chemical reaction would alter the presence of different materials in the atmosphere of the Mars habitat. The simulation had several pre-built parts that would help visualize the results. Once the simulation was started, plants would slowly grow, but if the presence of certain materials became too high, the simulation would stop due to a failure. Students were encouraged to

adjust the variables of the simulation to see if they could find a way to grow plants fast enough to sustain life.

On the third day, students were introduced to the concept of a binary number system. The activity started with students using small cards to represent different place values in the binary number system, and they slowly learned how to convert numbers from decimal to binary and back. The students were also briefly introduced to the way addition in binary works similarly to addition in a decimal system. Once the students were comfortable with binary, they were also introduced to the hexadecimal representation of simple binary numbers, as well as how more complex data such as text or images can be expressed in binary. For the activity, students were shown the scenario from The Martian where the main character must communicate with others using only a camera that rotates. He does so by placing sixteen signs around the camera representing hexadecimal values, and recording the signs that the camera points to and converting each pair of values to its equivalent ASCII value. Students were given a similar situation in Scratch, and were lead through the process of translating the data to ASCII. This involved calculating the angle of the camera, converting it from a degree value to a hexadecimal value, and then converting a sequential pair of values into an ASCII character. The activity itself mainly focused on using nested conditional statements as a decision tree to determine the ASCII character. As an added learning experience, the original version of the program contained an intentional typo in the message received, leading to ambiguity in the message. Students were encouraged to describe ways the system could be improved to minimize or eliminate ambiguous messages.

On the last day, students were introduced to several concepts in artificial intelligence. The students discussed the Turing Test and how it works, and then participated in an activity

that simulates how a neural network learns using training data. Afterwards, students were lead through an activity to create simple AI agents for an arcade video game following a simple perceptron model that required decision making using conditional statements based on the sensed inputs. The final activity built upon that structure by using a situation from The Martian, where the main character must plot a course around several terrain obstacles while driving across the surface of Mars. The students were given a program that randomly generated simple obstacles on a terrain, and upon reaching an obstacle, students had to use a simple perceptron model to determine how to get around the obstacles while still moving toward the goal.

Mighty Micro Controllers

Students were first introduced to the basic principles of electricity. Most had their first exposure to what electricity really is, as well as the concepts of conductivity and insulation. By using an example of marbles in a tube (Kuphaldt, 2014), students could visualize and understand the flow of electricity. Once this basic principle was established, circuits were introduced. For this intervention, students were introduced to resistance, voltage, and current (Ohm's Law), as well as digital, analog, and pulse width modulation (PWM) signals. After this introduction to electronic circuits, students learned how to make their first basic circuit using a solderless breadboard, the Arduino Uno, and Scratch. This was the blinking LED tutorial that most complete their first time using Arduino. Although, before wiring their circuit, students were required to create a circuit diagram using Fritzing to visualize how the circuit should be laid out (Knörig, Wettach, & Cohen, 2009). This introduced the idea of the engineering design process. After everyone completed the blinking LED example, students were introduced to an activity called "Resistance is Futile." Students received five resistors

ranging in strength from 220 ohms to 1 million ohms. They were challenged to rank the resistors in order of strength. By using the previous blinking LED circuit and program, students could visualize the effects of resistors and how they impede the flow of electrons with the dimming of the LED.

Then, the single blinking LED was expanded to introduce the idea of abstraction and other high-level programming and CT skills. Students were challenged to change the single LED circuit to include five LEDs of different colors. On their own, students needed to create the circuit, as well as get each LED to blink in succession. This led to a discussion about abstraction and problem decomposition. The programs created to achieve this task were duplicated code from the original blinking LED program. The class was guided through the process to recognize patterns in the program to reduce the number of blocks that were repeated. This led to using Scratch custom blocks which imitate functions. Students were left to identify which variables changed between each blink and which ones could be kept the same. The next activity extended this program to include pushbuttons. Students were given a circuit diagram and materials to wire a circuit with an LED and a pushbutton. This was a guided activity focused on teaching analog signals, pull-down circuits, open/closed circuits, and control flow.

On day three, activities for PWM signals that were introduced on the first day were done, as well as time to plan for a final project where students could design, build, and program their own circuits. The first activity of this day utilized RGB LEDs. First, students wired the circuit from a given diagram. Before being able to program, students needed to learn how to convert colors to traditional RGB format and how that was translated to the RGB LED connected to the Arduino. Common anode LEDs were used as well, so the students

discovered that they needed to invert the color before sending it to the Arduino. Abstraction and custom blocks were emphasized to make setting the different intensities of red, green, and blue simple. Students were guided through this process until they could turn on the red, green, and blue colors individually. Students were then given a large amount of discovery time to see what kind of colors they could produce using this circuit. At the end of the activity, a complete program that gradually changed through all the colors the LED could make was demonstrated. A video of an RGB LED matrix was also used to inspire students with more ideas for a final project. The day concluded with a guided activity using small motion sensors, and the applications of how the sensor could be used in their everyday lives.

The last day featured a guided activity using ultrasonic sensors. This activity used the Arduino IDE because Scratch was not able to accurately detect distance with the sensor. A side-by-side comparison was used with Scratch and the Arduino IDE to demonstrate how the text-based language translated into Scratch blocks. After this activity, students were given time to complete their group projects. However, they had to produce a design document, containing a circuit diagram and materials list, before they could start building or programming. This helped emphasize the engineering design process, as students had to keep revising their design when they discovered flaws in their original circuit. At the end of the day, students presented their projects to the class.

Instrument

To measure student learning, a self-efficacy survey to collect attitudes towards students' ability to think computationally was developed. The survey largely expands work done in the 2016 Summer STEM Institute by adjusting question language to be more age appropriate with the audience, as well as with the addition of questions assessing student self-

efficacy in problem solving (Weese, Feldhausen, & Bean, The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking, 2016). These questions are framed to correlate to appropriate computational thinking skills. As such, the survey is organized in four main sections: problem solving, computer programming skills, computer programming practices, and computer programming impact. Within each section, questions are categorized by relevant CT concept, practice, or perspective as seen in Table 4. Each of these questions measured self-efficacy on a five-value Likert scale: strongly agree, somewhat disagree, not sure, somewhat agree and strongly agree. Apart from these questions, the survey also contained questions collecting information about gender, participation in STEM activities/camps, and background in computer programming.

When solving a problem I...			I can write a computer program which...		
1	create a list of steps to solve it	Algorithms	10	runs a step-by-step sequence of commands	Algorithms
2	use math	Algorithms	11	does math operations like addition and subtraction	Algorithms
3	try to simplify the problem by ignoring details that are not needed (3)	Abstraction	12	uses loops to repeat commands	Control Flow
4	look for patterns in the problem	Abstraction	13	responds to events like pressing a key on the keyboard	Control Flow
5	break the problem into smaller parts	Problem Decomposition	14	only runs commands when a specific condition is met	Control Flow
6	work with others to solve different parts of the problem at the same time	Parallelization	15	does more than one thing at the same time	Parallelization

7	look how information can be collected, stored, and analyzed to help solve the problem	Data	16	uses messages to talk with different parts of the program	Parallelization
8	create a solution where steps can be repeated (8)	Control Flow	17	can store, update, and retrieve values	Data
9	create a solution where some steps are done only in certain situations (9)	Control Flow	18	uses custom blocks	Abstraction
When creating a computer program I...			When creating a computer program I...		
19	make improvements one step at a time and work new ideas in as I have them	Being Incremental and Iterative	22	break my program into multiple parts to carry out different actions	Problem Decomp.
20	run my program frequently to make sure it does what I want and fix any problems I find	Testing and Debugging	Impact		
21	share my programs with others and look at others' programs for ideas	Reuse, Remix, Connecting	23	I understand how computer programming can be used in my daily life.	Questioning

Table 4 The four core sections of the self-efficacy survey, denoting which CT skill each question falls under.

The experiment was carried out in a pre-post survey format. Pre-surveys were administered online on the first day of each week-long session before any class material was given. The post-survey, which did not contain demographic or STEM participation questions, was given on the last day of each session once all projects were finished. Survey participation was voluntary. Out of 110 students between both interventions, one student was excluded for opting out of the survey, one student was excluded for missing the pre-survey, and three students were excluded for having incomplete responses. A Chronbach's Alpha of .872 on the pre-survey and .908 on the post-survey shows that the survey described in Table 4 is reliable.

	% of Students	Avg. Pre Mean	Avg. Post Mean	Avg. Std. Dev.
Mars	53.33%	3.763	4.187	1.016
MMC	46.67%	3.619	3.884	1.056
Male	65.71%	3.71	4.083	1.048
Female	34.29%	3.661	3.978	1.032
No-STEM	29.52%	3.642	4.016	1.074
STEM	70.48%	3.718	4.058	1.026
OUTSIDE STEM	35.24%	3.673	4.108	1.043
ANON	27.62%	3.671	4.165	0.988
STEM INST	35.24%	3.764	4.008	1.006

Table 5 The distribution of students included in the survey within each compared group as well as average pre and post self-efficacy.

Cat	Skill	Mars	MMC	Male	Female	NO-STEM	STEM	OUT-SIDE STEM	STEM INST	Star-base
PS	Algorithms	0.400	0.061	0.194	0.306	0.245	0.229	0.269	0.186	0.344
PS	Abstraction	0.215	-0.082	0.101	0.036	0.056	0.089	0.058	0.117	0.153
PS	Control Flow	0.434	0.230	*0.299*	0.427	0.279	0.371	*0.558*	0.173	*0.645*
PS	Data	0.082	0.291	0.166	0.178	0.028	0.251	*0.349*	0.157	*0.387*
PS	Parallel.	0.181	0.037	0.165	0.000	-0.086	0.192	0.203	0.181	0.135
PS	Prob. Decomp.	*0.254*	0.154	*0.270*	0.059	0.090	*0.268*	*0.367*	0.173	0.310
CT	Algorithms	**0.828**	0.370	**0.667**	*0.448*	0.723	**0.538**	**0.702**	0.373	0.878
CT	Abstraction	0.501	**0.625**	**0.513**	0.639	0.362	**0.653**	0.545	**0.769**	0.692
CT	Control Flow	0.480	*0.353*	0.361	*0.574*	0.444	*0.408*	***0.583***	0.232	*0.682*
CT	Data	**0.728**	**0.537**	**0.629**	0.642	0.716	**0.603**	**0.818**	*0.395*	**0.892**
CT	Parallel.	**0.628**	**0.513**	**0.633**	*0.468*	0.706	**0.521**	**0.653**	*0.381*	0.704
CT	Prob. Decomp.	**0.530**	0.196	0.371	0.380	*0.432*	*0.344*	**0.560**	0.111	0.621

CT	Being Incremental and Iterative	0.229	*0.269*	*0.278*	0.189	0.295	*0.224*	0.274	0.169	*0.344*
CT	Questioning	**0.631**	0.083	**0.478**	0.203	0.429	0.339	**0.540**	0.141	**0.752**
CT	Reuse, Remixing, Connecting	*0.248*	0.198	*0.305*	0.054	0.248	0.211	0.024	0.412	0.132
CT	Testing and Debugging	0.091	0.230	0.167	0.143	0.309	0.093	0.126	0.056	0.176

Table 6 The effect size for each survey question, broken into each comparison group. Italicized indicates a p-value of ≤ .05, italicized, underlined indicates a p-value of ≤ .01, and bolded indicates a p-value of ≤ .001.

Findings

In the analysis of survey results, the 8 groups outlined in Table 5 were investigated: Saving the Martian (Mars), Mighty Micro Controllers (MMC), male, female, no previous participation in STEM activities/groups (No-STEM), previously attended this STEM program (STEM INST), previously attended a different STEM program (Outside STEM), previously attended Starbase (Star-base), and previously attended any STEM program (STEM). Average over questions 1-23 pre- and post-means can also be observed in Table 5 Table 6 shows the effect size (Sullivan & Feinn, 2012), calculated using pooled standard deviation, for each question, broken down by group. Note that effect sizes of 0.2 are considered small, 0.5 are medium, and 0.8 are large.

Overall, 70.42% of students had previously attended some STEM related group activity or program, while nearly all students in the program had used a visual-based programming language, mostly though Scratch, Hour of Code, and Lego robotics. These results show that in the area, outreach efforts are beginning to spread through the local K-12 population. Due to small sample sizes, the groups in Table 6 were not broken down into language background

(Scratch vs Lego robotics for example). Students who had never attended any type of STEM program showed to perform just as well as those who had attended a STEM program, apart from problem solving skills. The inverse applies when comparing the outside STEM group to those who had previously attended this STEM program. It is hypothesized that this result is partially due to a selection bias with this STEM camp and Starbase, which accounts for 78% of the outside STEM group. Those who had participated in Starbase in the past showed significantly higher effect sizes on most CT concepts. This could be explained by the difference in the two programs. The summer camp focuses on getting students having fun in STEM. While Starbase includes many fun, hands-on activities, it has a richer, deeper focus in STEM learning outcomes. Also, the Starbase participants are from complete classes, whereas this STEM camp contains participants who volunteered.

When comparing the two interventions, Saving the Martian had a larger positive effect on student self-efficacy in all four question sub-areas. This is further confirmed when looking at the average initial self-efficacy in Table 5, where MMC could not capitalize on its students' higher potential to learn (lower initial self-efficacy compared to Mars). It is hypothesized that this was caused by additional overhead from making and controlling circuits, even though the activities were designed to reduce cognitive load. Breaking down the results to specific concepts, the Mars curriculum emphasized algorithmic thinking through sorting algorithms, using CS unplugged to explain sorting before implementing in Scratch. This is shown to be highly effective compared MMC, which focused on using LEDs as a method to teach algorithms. MMC focused creating and programming parts of each circuit one component at a time. For example, to make a circuit with five LEDs, students first had to make a circuit with only one. Surprisingly, MMC had no significant result for problem decomposition. It was

also surprising to see no effect on MMC students' understanding on how programming can be used in their daily lives. The MMC curriculum leveraged physical computing to provide tangible results from programming lights and sensors that could be used at home in practical applications; however, it was unsuccessful in the context.

Across all groups, the curriculum did not perform well for questions 19-21, when compared to CT concepts. This shows that CT practices need to be balanced more alongside the other CT skills. Problem solving skills also performed poorly overall, revealing little to no effect in many areas. Skills like algorithms and control flow show some relation the effects in CT skill questions; however, there is no discernable pattern. Results between male and female were interesting, particularly with conditionals in control flow, though sample sizes were too small to make conjectures.

Conclusions

In this section, the application of two interventions applied to a 5^{th}-9^{th} grade STEM outreach program was discussed. From reviewing the survey results, it was discovered that using micro controllers as a tool for teaching CT was less effective than a pure computer science related curriculum. Although MMC was effective at fostering improvement in CT skills, the curriculum has room to improve when using physical computing. By adjusting the language from the 2015 summer institute (Weese, Feldhausen, & Bean, The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking, 2016) to be more age appropriate, as well as a smaller Likert scale, a survey that was more consistent within student responses and effective at measuring self-efficacy in CT was created. Finally, it is shown that CT framed inside PS is largely decoupled from CT. This may be the starting indicators toward supporting the goal to show that CT is not PS; however, due to the small sample sizes of the

experiment, future studies would be needed to fully confirm that goal. Due limitations of the STEM program, the research was conducted as transparently as possible. Knowledge-based assessments would likely provide more insight into the research questions; however, this would make it feel like a normal classroom and not a summer camp. More so, self-efficacy has been shown to be a good predictor of student learning outcomes (Lishinski, Yadav, Good, & Enbody, 2016). Apart from knowledge-based assessments, static analysis on code produced by students. Since many the activities were heavily scaffolded, most solutions that students produced were identical leaving little information to be gained.

Starbase

Starbase is a STEM outreach program open to 4th-6th grade students, primarily funded by the Department of Defense. Since its inception in 1992, it has engaged thousands of students in STEM topics. In a study by Wilder Research, long-term effects of the Minnesota *Starbase* program showed that high school students who had participated in *Starbase* showed high interest in technology and science, as well as engineering and math, when compared to a control group (Mohr & Mueller, 2012). Moreover, students who had participated in *Starbase* were more likely to graduate high school on time and enroll in college or interest in joining the military. The authors note that while this was not statistically significant, it reveals a potential longer term track the program may take. For the purposes of this research, only the *Starbase* program in Manhattan, KS will be considered, though there are four other *Starbase* programs in Kansas.

The Manhattan *Starbase* consists mainly of 5th grade students, though some 4th and 6th graders participate in the program later in the academic year. Students who participate in a week (spread throughout a month) of hands-on experiments. These cover the engineering

design process (EDP), Newton's Laws of Motion, fluid mechanics, aerodynamics, basic chemical reactions, nanotechnology, rocketry, computer aided drafting using Creo (CAD program made for primary grades), robotics with Lego EV3, and many other experiments. However, the Manhattan *Starbase* have a unique partnership with Kansas State University's Department of Computer Science. This partnership gives participants an opportunity to learn Scratch programming. This activity focuses on algorithmic design and parallel computing by having students translate the song "Ode to Joy" by Beethoven to Scratch (Bean, Weese, Feldhausen, & Bell, 2015). Students are shown how to program the first couple notes in Scratch, then they are tasked with translating as much of the song as they can with their partner (pair programming). Students are given scales with note letters if they are unfamiliar with reading music and are asked to swap roles with their partner during the activity (one programs while the other translates). They are also split into groups and get guided tours of Beocat, the supercomputing cluster at KSU. At the end of the activity, students are asked "Are you computer programmers?" Many are hesitant to raise their hands, but with some encouragement, they usually all say yes. The motivation behind this question is to help students connect computer programming to more than just lines of code. Whether they are making their robot move (they also program in their EV3 challenges) or telling their computer how to play Ode to Joy, they are still programmers. This helps students connect computing to the technology people use in their daily lives and realize the endless applications of computing.

Findings

Starbase is assessed in a pre- and post-test with questions over material covered in the experiments done by participants. The surveys also contain a small number of self-efficacy

questions, some of which were done in consultation with the group at KSU. These questions can be found in the appendix: *Starbase* Post. The goals of this research are complimentary with the work done with the USD 383 summer STEM Institute, looking at the effects of experience and knowledge in STEM on student ability in CT. Data is collected anonymously from the Manhattan *Starbase* as existing data, as such, the KSU researchers do not have control over survey questions or administration. Data has been collected during the spring 2016, fall 2016, and spring 2017 semesters and contains anonymized student demographics and responses from individual survey questions. Data from 309 students have been transcribed; 35 were excluded for not taking the pre- or post-surveys, leaving 274 students for data analysis. Gender and demographic information of these students can be found in Table 7. Findings between comparisons of genders and race, while informative, are unable to be explained with current data. Future work will be needed, such as student interviews, to investigate these findings.

Male	Female	Caucasian	African-American	Asian	Native American	Hispanic	Multinational	Other
131	143	176	23	3	1	17	37	17

Table 7 Distribution of Starbase student democraphics and race.

Male	Score	Math	Science	Tech	Prog	Group Work	Prog. Robot	Outside CS
Pre	8.786	1.191	1.260	1.191	1.351	1.076	1.336	1.305
Post	14.145	1.687	1.267	1.168	1.229	1.191	1.130	1.183
STDV	1.819	1.819	1.819	1.819	1.819	1.819	1.819	1.819
Effect	2.946	0.273	0.004	-0.013	-0.067	0.063	-0.113	-0.067
P	0.00000	0.00000	0.87948	0.61396	0.02593	0.00493	0.00005	0.02593

Table 8 Starbase test results for male students.

Female	Score	Math	Science	Tech	Prog	Group Work	Prog. Robot	Outside CS

Pre	8.028	1.168	1.112	1.168	1.490	1.028	1.406	1.392
Post	14.706	1.874	1.224	1.238	1.343	1.252	1.196	1.140
STDV	1.904	1.904	1.904	1.904	1.904	1.904	1.904	1.904
Effect	3.507	0.371	0.059	0.037	-0.077	0.118	-0.110	-0.132
P	0.00000	0.00000	0.01533	0.23329	0.01910	0.00000	0.00032	0.00002

Table 9 Starbase test results for female students.

Caucasian	Score	Math	Science	Tech	Prog	Group Work	Prog. Robot	Outside CS
Pre	8.443	1.210	1.148	1.210	1.415	1.040	1.403	1.313
Post	14.955	1.778	1.216	1.199	1.250	1.210	1.136	1.148
STDV	1.846	1.846	1.846	1.846	1.846	1.846	1.846	1.846
Effect	3.528	0.308	0.037	-0.006	-0.089	0.092	-0.145	-0.089
P	0.00000	0.00000	0.08326	0.80340	0.00216	0.00001	0.00000	0.00149

Table 10 Starbase test results for Caucasian students.

ALL Other	Score	Math	Science	Tech	Prog	Group Work	Prog. Robot	Outside CS
Pre	8.296	1.122	1.245	1.122	1.439	1.071	1.316	1.418
Post	13.510	1.796	1.296	1.214	1.357	1.245	1.214	1.184
STDV	1.888	1.888	1.888	1.888	1.888	1.888	1.888	1.888
Effect	2.761	0.357	0.027	0.049	-0.043	0.092	-0.054	-0.124
P	0.00000	0.00000	0.42615	0.16095	0.21879	0.00027	0.05839	0.00032

Table 11 Starbase test results for all races except Caucasian.

In Table 8, Table 9, Table 10, and Table 11, pre- and post-scores are calculated by grading the knowledge-based questions 1-20 in the Starbase test shown in the Appendix: Starbase Pre-test and Starbase Post-test Overall, an effect size of 2.946 shows a significant improvement in students understanding of STEM topics after participating in the Starbase program. Breaking this down by gender, females saw better improvement compared to males in understanding of STEM topics. Female pre-scores indicated that they came in with less knowledge, but ended up scoring higher on the post-test compared to their male classmates.

An opposite trend is seen when comparing the scores of Caucasian students and students of other nationalities. Non-Caucasian students had similar levels of understanding to Caucasian students; however, they scored significantly lower on the post-test.

All students devalued group/teamwork at the end of the Starbase program. There are many activities that are done in pairs and small groups, but this may indicate that the method of assigning groups needs to be changed. Likewise, these activities may need to be adjusted to better fit group work over the individual. All students seemed to dislike Math even more after the Starbase program. Most groups of students' attitudes towards science and technology did not change, indicating that they enjoy these areas of STEM; however, female students disliked Science more after the program. All students showed a significant (except non-Caucasian students) increase in attitudes toward computer programming, indicating that exercises done with Lego robotics and Scratch are effective at getting young students interested in computer programming. All students indicated a better understanding that computer science can be applied in a variety of areas. This allows students to not see computer science as a closed field, but as a gateway to many STEM disciplines or even the arts. This is the core philosophy of computational thinking. Leading students to the realization that they do not have to be computer scientists to use those tools and techniques, and that they can continue to use what they learned in whatever they decide to pursue as a career.

Conclusions

Like the summer STEM Institute, the Starbase program is set in getting young minds excited about and interested in STEM, but more so in educating them. There are many activities in Starbase that begin with lecture material and followed up with hands-on experiments. These always have learning goals in mind, which are apparent in the strong

performance from students in the post-test knowledge assessment. However, this traditional approach in teaching STEM does not seem to captivate student interest in some areas in STEM, particularly math (all groups) and science (only for females). There was also evidence of this in comments made by students in the post-survey (see below). As part of the Starbase program, students are exposed to programming through Lego robotics, as well as Scratch. Both activities are designed as challenges, rather than lectures or experiments, for students to overcome with a wide breadth for tinkering and creativity, and have been shown to increase student self-efficacy as well as attitudes towards programming. These activities also contain the base constructs for CT, which gives the students from Starbase who participate in the summer STEM Institute a greater capacity to learn CT. Moving forward, Starbase will need to change an adapt activities, possibly by taking from the successful model from the STEM Institute, to captivate student attitudes towards more areas of STEM. When students at Starbase were asked "When I talk about STARBASE to my family and friends I say…", responses were mostly positive, though some did not enjoy the program. Here are a few samples:

> *"Starbase was fun and a little challenging but I like it! I would go back. Starbase is awesome I want to do an experiment at home!"*
>
> *"We learn about programming engineering, mathematics, we learn about science and they make learning fun"*
>
> *"It was amazing to learn about STEM because all the activities told me about new stuff that my teacher could not tell me"*
>
> *"I do not like it. It was too much instructions and did not let us be free and do what we want. But I liked it and liked the rockets"*

“I like Starbase because we get to use Lego robots, and launch and make a rocket and we got to go to k-state”

“What I learned that day and please pass the sodium chloride.”

Chapter 4 - Computational Thinking in Physics

As stated in Chapter 2 - computational thinking is intended to be a fundamental 21st century skill; a tool envisioned by Jeannette Wing to be accessible by everyone in a day of modern technology (Wing J. M., Computational Thinking, 2006; P21 Partnership for 21st Century Learning, 2015). However, it is important to know where computational thinking plays its roll in integration with STEAM disciplines and modern 21st century learning competencies (Dede, Mishra, & Voogt, 2013). Computational thinking should be something students can learn as a fundamental skill, and not be forced into computer science as a discipline. Rather, students should learn the roles CT plays in other domains. Hemmendinger also argues that the aim of CT is not trying to force everyone to think like a computer scientist, but to "teach them how to think like an economist, a physicist, an artist, and to understand how to use computation to solve problems, to create, and to discover new questions that can fruitfully be explored" (Hemmendinger, 2010). Hemmendinger continues that maybe computer scientists have turned too much focus on computational *thinking* and less towards computational *doing* – being able to implement and carry out new ideas through computational tools (Hemmendinger, 2010). This also leads into more of the original ideas of Seymour Papert who promoted computational literacy through "micro worlds" and the LOGO programming language (Papert, 1980). Scratch continues Papert's work by emphasizing creative computing, enabling a wide breadth of access to the ability to create personalized computational artifacts and ultimately allowing the "development of personal connections to computing" and the development into "computational thinkers" (Brennan, Balch, & Chung, Creative Computing, 2014). This begins to bring together the idea that CT extends ordinary

human-computer interactions, extending human creativity and intuition into a modern age (Mishra & Yadav, 2013).

Leveraging its constructionist approaches to teaching programming, Scratch has been and still is an excellent tool for introducing CT in various disciplines, especially in the arts (Brennan, Balch, & Chung, Creative Computing, 2014). Scratch is commonly used to introduce CT through storytelling, letting new learners focus on being creative and telling a story, all-the-while learning CT concepts like algorithmic thinking, iteration, and conditionals (Burke & Kafai, 2010). Scratch has also been successful embedding CT in the music domain (Bean, Weese, Feldhausen, & Bell, 2015). Creativity in engineering though robotics has also been a prominent area for embedding CT for new learners (Leonard, et al., 2016), particularly due to robotic systems like LEGO that are controlled using visual-based languages. Computational thinking has even made its way into the Ethics classroom, making student think critically and systematically on the choices machines are programmed make, particularly smart cars, and the moral and ethical ramifications that are associated with those choices (Seoane-Pardo, 2016). CT has also begun making progress integrating into science and mathematics through modeling, simulations, and video game design (Weintrop, et al., 2016; Wilensky, Brady, & Horn, 2014; Sengupta, et al., 2015; Repenning, et al., 2015).

Computation in physics, however, has been very slow to start in higher education. Content is often tightly packed into courses and pressure on faculty to deliver is high, making it exceedingly difficult to work in new computational physics material (Roos, 2006). Likewise, teaching loads, alongside research and service requirements, leaves faculty and departments with no room to add new courses that emphasize computational thinking (Roos, 2006). Researchers are advocating for the need of introducing CT into physics and looking for

various ways to embed it in undergraduate curriculum (Chonacky & Winch, Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments, 2008). The Partnership for Integration of Computation into Undergraduate Physics (PICUP) is community for educators who are looking for ways to incorporate computation into their physics classrooms (Chonacky & Winch, PICUP, 2017). PICUP continues to create and publish a large amount of curriculum and resources, ranging from models, simulations, problem sets, and programming exercises that emphasize computational thinking in physics.

Others have taken the challenge to embed computation into existing courses. Roos discusses that "one obvious drawback of integrating computational physics into the traditional format of physics courses is its potential interference with the course's core subject matter" (Roos, 2006). CT in STEM is not necessarily intended to be its on subject matter of a course, but a tool to enhance students learning and comprehension of course content. Still, depending on students' previous experience with technology or programming, faculty may only have time for shallow coverage, relying on student self-study to learn the selected computational tools (Roos, 2006). In their university, Roos first looked at using commercial software; however, it "produced a barrier between students and the direct calculations." Using black-box algorithms in languages like MatLab add a level of user friendliness, but also prevents students comprehending what is really going on in the background. This lead Roos to conclude that computational physics can be adequately learned using "direct programming," which "maximizes computational learning and helps students achieve maximum understanding and control in numerically solving equations." On the other end of the spectra, Austin Peay State University reorganized their physics major to include three new courses that progress from

theoretical, experimental, and computational methods (Taylor & King, 2006). Computational tools introduced to the major were not unique to the new courses, but added throughout the major's course offerings. Unlike Bradley University (Roos, 2006), many commercial and GUI tools were used, such as LabView, MatLab, and Electronics Workbench (Taylor & King, 2006). The curriculum also leveraged the direct programming approach with C++, Fortran, and C#; however, students are not left to self-study. They required to take two levels of introductory programming and one object oriented programming course (Taylor & King, 2006).

CT has also made it into K-12 physics. Dukeman et. al. introduced the C3STEM framework to teach students CT skills in the traffic domain (Dukeman, et al., 2013). Through C3STEM, students worked to identifying patterns by analyzing real traffic data. The students had a support system where they had the opportunity to communicate with traffic engineers, city planners, and members of the research team for guidance and feedback on data analysis and their initial models. Students then created agent-based models and introduced interventions in a simulation to study how their decisions affected traffic flow. Farris and Sengupta used agent-based modeling to bring physics to 5th and 6th graders. In this study, students learned about kinetics by creating models and simulations in ViMAP, a custom visual-based programming language that supports "domain specific learning in kinetics" (Farris & Sengupta, 2014).

Exploratory Study: Computational Thinking in Electromagnetics

This section outlines an exploratory study in integrating computational exercises into PHYS 532, Electromagnetic Fields I, at Kansas State University. Embedding computational thinking into physics classrooms has taken on a slow start. Given the previous examples of

attempts in undergraduate courses, there is not a clear consensus on what is the best approach. The past integrations of computation into physics, including the integration of direct programming using Fortran and C++ (Roos, 2006) into existing courses and a broader approach to complete physics major overhaul outlined in (Taylor & King, 2006). Neither of these approaches have any formal measure to provide empirical justification for one approach over the other. The approach chosen was to use direct programming using the Python language through a series of computational problems to reinforce topics covered in lecture. Python was chosen over the dominant languages Fortran, C++, and MatLab due to its exceptionally low entry barrier. Python has previously gained popularity as an introductory language and scientific computing language for many reasons, including:

- Pseudocode-level syntax makes Python English-like, making it much easier to read for novice programmers
- Free and open source, making it more accessible compared to other popular commercial tools
- Cross-platform compatibility
- Wide breadth of libraries and modules built for visualization and scientific computing, including NumPy, SciPy, VPython, and PyQt
- Dynamically typed, allowing fast development and prototyping through a variety of programming paradigms like imperative, object-oriented, and functional programming

Others have also expressed Python's viability as a scientific computing language (Oliphant, 2007) and as a viable language in computational physics (Backer, 2007; Borcherds, 2007). The purpose of this study is to investigate the feasibility of incorporating the Python

programming language into an undergraduate physics classroom and how it effects student self-efficacy in computational thinking.

Experiment Design

This exploratory study was introduced to the Fall 2016 PHYS 532 course that lasted sixteen weeks. As part of the traditional course, students were given five computational thinking problems:

1. CT1 – Pi: Students were asked to calculate pi. Three methods of calculating pi were given: a) Use of the area of a unit circle and generation of random points between 0 and 1 and comparing the number of points within the circle ($x^2 + y^2 < 1$) to those outside the circle. b) Averaging the perimeters of a circumscribed square. c) The MacLaurin series expansion given $\arctan(x) = \sum_{n=0}(-1)^n \frac{x^{2n-1}}{2n+1}$. Students were tasked to write pseudocode for method (a) and for method (b) or (c). Then, by using the pseudocode as comments, implement the two chosen methods in python to be accurate to at least six digits.
2. CT2 – Dipole: Students receive a base starting file where they must add comments to explain what is happening in the code. Afterwards, they make various adjustments to the plot, different dipole fields, charges, and e-field.
3. CT3 – Bars and Planes: Adjust the code from their solution in CT2 to find and plot the electric field everywhere from a bar of length L and total charge Q. Then they must show that the field is appropriate in the limits of infinite length and constant non-charge density. The code must allow for a location based charge density and able to find the electric field from a disc.

4. CT4 – Capacitors: Using the method of relaxation, find the potential everywhere near a finite disc capacitor. Then show that the results are reasonable in the limits of large radiuses, separation, and zero charge. Plot the equipotential and electric field lines.
5. CT5 – Toroidal Ion Trap: Design a toroidal trap which uses electric potential to keep charged particles in a ring. Show that the trap has a potential profile that could trap positively charged ions in a ring.

Due to flexibility issues with course curriculum, students were not given training in programming in Python and were required to rely on any prior experiences and self-study. Each CT problem lasted approximately one to two weeks, and students could collaborate with each other and use online resources albeit proper citations. For each problem, students were required to add sufficient documentation to explain the code they wrote. All CT problems were assigned and completed by week eight of the course.

Part of the evaluation included qualitative analysis of the code submitted for the CT problems, this included the course population, which consisted of 20 students: 12 physics majors (the majority were seniors) and 8 students from 6 other majors including physics education, computer science, mathematics, and electrical, mechanical, and nuclear engineering. Other demographic data was excluded to preserve anonymity. The second part of the evaluation includes results from a modified self-efficacy survey from Summer STEM Institute 2016 and can be seen in Table 12.This survey was used to measure attitudes towards computational thinking, problem solving, and the use of computer programming. The survey was administered before the first CT problem was assigned and at the end of the course. All 20 students responded to the pre-survey; however, 8 were excluded from analysis for not completing the post-survey and two students opted-out for using their survey data in research.

When solving a problem I...			I can write a computer program which...		
1	create a list of steps to solve it	Algorithms	10	runs a step-by-step sequence of commands	Algorithms
2	use mathematics	Algorithms	11	does math operations	Algorithms
3	try to simplify the problem by ignoring details that are not needed	Abstraction	12	uses loops to repeat commands	Control Flow
4	look for patterns in the problem to create an efficient solution	Abstraction	13	takes input from a user	Control Flow
5	break the problem into smaller parts	Problem Decomposition	14	only runs commands when a specific condition is met	Control Flow
6	work with others to solve parts of the problem in parallel	Parallelization	15	runs commands in parallel	Parallelization
7	look how data can be collected, stored, and analyzed to help solve the problem	Data	16	uses messages and other information to talk with different parts of the program	Abstraction
8	create a solution where steps can be repeated	Control Flow	17	can store, update, and retrieve data	Data
9	create a solution where some steps are done only in certain situations	Control Flow	18	uses custom functions	Abstraction
When creating a computer program I...			When creating a computer program I...		
19	make improvements one step at a time and	Being Incremental and Iterative	22	break my program into multiple parts to	Problem Decomp.

	work new ideas in as I have them (1)			carry out different actions (4)	
20	run my program frequently to make sure it does what I want and fix any problems I find (2)	Testing and Debugging	Impact		
			23	I understand how computer programming can be used in my daily life.	Questioning
21	share my programs with others and look at others' programs for ideas (3)	Reuse/Remix, Connecting	24	I am confident I can use/apply computer programming to my field of study.	Questioning

Table 12 Self-efficacy survey given to students in the E&M class.

Results

This study was not setup to study impact of computational thinking on a student's understanding of physics, nor a student's ability to program in Python. As such, evaluation of student submissions was done qualitatively and not through static analysis, unit testing, or grading rubrics. First, levels of abstraction were looked for in student submissions. Abstraction was chosen over other CT concepts due to its difficulty compared to concepts like iteration and conditionals, which nearly all students showed basic understanding of. Code efficiency is considered for CT4 and CT5. No students demonstrated efforts in parallelism. Comments made in the code were also analyzed, coding for levels of understanding of the code and the physics/computational concepts covered in the CT problem. Comment codes were not exclusive. These codes can be found in Table 13. Completion rate of each CT problem were 85%, 95%, 75%, 70%, and 60% respectively.

General Label	Description
Abstraction	Python functions, apart from any starter code, were defined and used.
Efficiency	Poor algorithmic design that causes the code to run long, mainly due to inefficient/nested loops.

No Comments	No effort was made to generate comments.
Code Comments	Sufficient comments were made, but only described actions of the code and not the physics concepts.
Pseudocode Comments	(basic) The comments made gave an overall outline to what was happening in the code, along with general description of the physics involved. (depth) The comments made gave in-depth detail on the physics concepts and linking them to how the code was executed.
Commented Code	Some actual code was commented out, leaving behind evidence of tinkering.

Table 13 Annotation labels and descriptions used to analyze student code submissions.

Most students did not show understanding of abstraction in their code. The first CT problem was simple with only two students using functions. These were only single use functions; however, it indicated that these students had a better understanding of Python and the ability decompose and organize a problem into smaller, reusable parts. For the second CT problem, students were given starter code to modify. The starter file contained one function, but four students introduced more to better organize and reuse code. The upward trend of students understanding abstraction and the use of functions continued into CT3 with eight students utilizing functions. Nine students implemented functions for CT4; however, most (6/9) used only a simple accessor function that was more suitable to be a lambda expression instead of a function. Students could work together on each of the CT assignments, with each one having 23.5%, 10.5%, 53.3%, 50%, and 75% of the students who had turned in code working in small groups, respectively. This ended up having a negative impact on CT4 with the unnecessary use of abstraction. The negative impact of group work continued into CT5 where only two students used functions. Two other students had poorly optimized solutions that had many triple and quadruple nested loops, although the method of relaxation used was inherently inefficient so this finding was not surprising. The rest of the students for CT5 created mostly flat scripts and many hard-coded values. It is believed that group work was one

core part of the degradation in quality in both code and CT concepts in CT5; however, many students did not show mastery of the physics concept in comments. Only 15% of the students in CT5 had comments that indicated understanding of the physics problem, compared to 60% and 42.85% in CT3 and CT4 respectively as seen in Table 14. The prevalence of few/simple comments was also present in CT2, where most of the work done was modifying existing code. While this gave in to higher completion rate, there was much less effort given to commenting and understanding the existing code. This was flipped in CT3, where students had to modify their solution to CT2 but were given more abstract directions. This resulted in a higher number of in-depth comments explaining the physics concepts that were happening in code.

Label	CT1	CT2	CT3	CT4	CT5
None	0	2	0	2	5
Only Code	3	6	6	3	1
Basic Pseudocode	3	8	3	6	4
In-depth Pseudocode	10	3	9	6	2
Commented Out Code	4	4	3	5	0

Table 14 Number of labels annotated for each degree of commenting made by students in their CT problem submissions.

Category	Skill	Pre-Mean	Post-Mean	Pooled Stdv	Average Gain	Effect Size	P Value
PS	Algorithms	3.625	3.667	0.911	0.042	0.029	0.545
PS	Abstraction	4.083	3.667	0.919	-0.417	-0.464	0.193
PS	Control Flow	4.167	4.083	0.867	-0.083	-0.096	0.777
PS	Data	3.292	3.583	0.948	0.292	0.320	0.372
PS	Parallel.	3.083	3.333	0.917	0.250	0.273	0.191

PS	Prob. Decomp.	4.167	4.083	0.867	-0.083	-0.096	0.777
CT	Algorithms	3.625	4.167	1.215	0.542	0.443	0.129
CT	Abstraction	2.458	3.625	1.132	1.167	**1.022**	0.002
CT	Control Flow	3.361	4.056	1.222	0.694	0.572	0.086
CT	Data	2.917	3.500	1.238	0.583	0.471	0.027
CT	Parallel.	2.333	3.000	0.959	0.667	0.696	0.039
CT	Prob. Decomp.	3.083	3.667	1.178	0.583	0.495	0.306
CT	Being Incremental and Iterative	2.750	3.917	1.167	1.167	**1.000**	0.019
CT	Questioning	4.083	4.375	1.015	0.292	0.288	0.392
CT	Reuse, Remixing, Connecting	2.500	3.667	1.128	1.167	**1.034**	0.023
CT	Testing and Debugging	3.833	4.083	1.163	0.250	0.215	0.571

Table 15 Self-efficacy pre- and post-survey results.

In the analysis of survey results, most students had no direct-programming background; however, three students had a class on Java programming and showed high initial self-efficacy in CT. Otherwise, background in MatLab or Octave accounted for 75% of the prior programming experience among students and had no correlated effect in reported self-efficacy. Results showing the pre-post self-efficacy in CT and PS concepts and practices can be found in Table 15. CT concepts reported mostly medium effect sizes; however, abstraction had a significantly large effect. Abstraction is arguably one of the most important and difficult CT concepts, yet without being explicitly taught, students demonstrated varying levels of abstraction in code as well as their self-efficacy. Breaking down abstraction, self-efficacy for creating a program that uses custom functions was 1.195 ($p = .002$). Couple this with question 16 (using messages and information to talk with different parts of the program)

that reported an effect size .849 ($p = .002$) and it shows that students not only understood how to create functions, but also understood parameter passing and return values. This could also be attributed to the starter code given in CT2 where students were required to provide comments to explain the function given, as well as the number of libraries students used to create and visualize models. This also works well with using variables (store, modify, and retrieve data) that had significant results; however, with the large emphasis on models and usage of variables in the code turned in, a larger post self-efficacy score was expected. A dynamically typed language, like Python, can be a great benefit to new programmers because data types are not required and variables are quite flexible. This can also have a negative impact, particularly in complex applications or problems, where students are not explicitly forced to learn data structures or types and how to utilize them as in statically typed languages. Iteration had significant, large effect in student self-efficacy, but some of the later CT problems showed inefficient use of for-loops. There were no significant results from CT concepts framed as problem solving skills, though data and iteration had a medium and large effect size respectively. Similar disconnection between PS and CT was seen in the Summer STEM Institute 2016 survey, but further studies, including student interviews that target problem solving skills, will be required to make any significant claim.

Apart from CT skills, students reported significantly higher self-efficacy in CT practices like being iterative and incremental, as well as reusing, remixing, and connecting. This is supported by student collaboration on the CT problems as previously noted. Consequently, by the end of the CT problems, students began to rely too heavily on the collaborative aspect and hindered their understanding of the problems and the uniqueness of their solutions. Iterative development can also be seen in student solutions through code that

students comment out. Students expressed that they had good understanding of how computer programming could be used in daily life, but experienced a significant increase in self-efficacy in apply programming to their field. This is corroborated by student's expectation of using programming in the future where students reported that they would sometimes use programming as part of a class, work or hobby (3.5 $\sigma = 1.17$, 3.5 $\sigma = 1.09$ and 3.08 $\sigma = 1.44$) and would usually use programming (3.92 $\sigma = 1.17$) as part of research. Some students expressed that they liked having programming as part of the class; however, most students struggled learning a programming language:

> *"I think that getting us involved in programming is a great idea but the way that we were taught how to do so was an unmitigated disaster. When over 60% of our class had never programmed before this class and we weren't even taught basic languages or why we need to import files to make our program work it was unbelievably frustrating. I accept the fact that programming is/will be very important for some of our futures but this is not the correct class to introduce us to it. There simply wasn't enough class time."*

This was a common theme among student, both the lack of class time dedicated to the CT problems and little to no support given to learning how to program in Python:

> *"It would have been much more beneficial to me if we did more in class work on the computational problems rather than being left to figure it out ourselves."*

> *"More guidance, maybe in the form of an online video series, would be welcome. The lack of information at the beginning of the course made the projects very difficult."*

Teaching yourself how to program in a new language, especially if you have never programmed before, can also be very difficult to overcome. Some students expressed a mix of success in trying to find help online for their problems:

> *"It would be nice if we actually had time in class to be taught the syntax and functions we need to know to complete the assignment. Only one person in the class had seen Python before, but the homework assignments were pretty much Google it yourself and figure out how to do it."*

> *"While I did learn a lot of the basics and could really easily write pseudocode. I found that I had an incredibly hard time making the actual code. While google did solve a few problems I often didn't know what to search to find the answer that I was looking for."*

Another student found it difficult not only to be able to learn programming, but also the added difficulty and overhead in applying programming within physics:

> *"I think it would be better to take a course on programming before dealing with it in a physics context. It was difficult to learn a new language (and programming in general for some people) and apply physics without coercion, which was using physics to avoid programming. I think a class before would have helped me know what to do and be more efficient/comfortable. Also, we spent a total of two class days for five assignments which took, at a minimum, five hours. This made them feel unimportant."*

However, not all students had a completely negative experience:

"Programming is awesome, I can't wait until I actually become natural at putting this knowledge to use."

Conclusions

Introducing computational thinking into undergraduate curriculum has been shown to be a difficult task. Similar problems found in the literature, particularly being able to fit new concepts and techniques into an already bloated curriculum. The approach of using direct programming has promise. Students could demonstrate levels of abstraction and algorithmic thinking, as well as CT practices like reusing/remixing and being incremental and iterative in their Python code, as well as their self-efficacy. While students expressed a general understanding of the benefits of incorporating computational physics into the class, they found the lack of support for learning the Python programming language debilitating. To continue incorporating CT into existing physics curricula, supplemental material, such as video lectures, will be required to alleviate some of the cognitive load from the students. Another area for future study is focusing on how introducing CT effects student comprehension of the physics content. In the experiment conducted, some levels of this were present in the comments of student code; however, this does not capture how CT effects student learning outcomes in other homework, exams, or labs.

Chapter 5 - The Data Explorer

This chapter, published in part in (Weese & Hsu, Work in Progress: Data Explorer – Assessment Data Integration, Analytics, and Visualization for STEM Education Research, 2016), describes the primary components of an analytics system for STEM education research, developed for the American Association for Physics Teachers (AAPT). The purpose of this data exploration system is to allow instructors to comparatively assess student performance in intraclass, longitudinal, and interinstitutional contexts. The interface allows instructors to upload course data including student demographics and exams to a secure site, then retrieve descriptive statistics and detailed visualizations of this data. For Physics Education Research (PER), the Data Explorer will be one of the largest repository of assessment data. This enables research on significantly larger populations and diverse groups, while providing users of the Data Explorer detailed comparisons and analysis, as well as expert recommendations tailored for teaching physics in their own classrooms.

Automated Assessment Extraction

While some work focuses on automatic analysis of programs (Koh, Basawapatna, Bennett, & Reppening, 2010) for evaluating student performance, others use sentiment analysis and topic modeling of things like student comments to predict student performance (Sorour, Goda, & Mine, 2015). However, in CS, little work is done modeling students by using student assessments. This section presents various approaches from table extraction schema inference for the assessment extraction research in this dissertation. One approach, exemplified in the previous work of Doan, Domingos, and Halevy, uses machine learning to produce classifiers for schema matching (Doan, Domingos, & Halevy, 2003). Cafarella et al. extend this approach by targeting relational schema and using constraints on relational well-

formednesss (Cafarella, Halevy, Wang, Wu, & Zhang, 2008). More recently, Venetis et al. infer semantic properties of web data by using observed weak typing constraints (is-A relations, also known as *hyponymy*) in online knowledge sources (Venetis, et al., 2011). In a variation on this general approach, the research done in this chapter also uses pattern matching heuristics and constraints, but restricts the matching to type constraints such as enumerative types on multiple-choice questions.

Another approach, holistic information extraction from tables, is characteristic of systems such as that of Nagy et al., which use syntactic elements of tables – header paths in particular – to extract relational tuples (Nagy, et al., 2011). This approach subsumes tabular data cleaning. For example, Fang, Mitra, Tang, and Giles use supervised inductive learning to learn the concept of a genuine table (as opposed to spacers and decorative elements), and empirically validate heuristics for physical structure analysis (table segmentation, which is obviated in the task) and logical structure analysis (Fang, Mitra, Tang, & Giles, 2012). Suchanek and Weikum examine how to capture such tables in the wild, e.g., as embedded in articles on the web or in print; some relevant ideas from this approach are how to use rule-based data transformations to segment uploaded data (remove headers, trim extraneous elements) and validate them against known good tuples (Suchanek & Weikum, 2013). Adelfio and Samet specifically address the chief problem of schema extraction for tabular data by using a conditional random field (CRF) classifier learned from data; this approach has achieved marked success in shallow parsing tasks such as named entity recognition in text (Adelfio & Samet, 2013). Finally, Zhang re-examines the problem of capturing relations in tables using a combination of named entity recognition and the kinds of semantic constraints applied by the second approach (Zhang, 2014).

Data Flow

The first component consists of a rule-based system for pattern analysis that infers multiple common assessment formats with minimal metadata, and in some cases without headers. This paper describes the incremental development of a priority-based inference mechanism with matching heuristics, based on real and synthetic sample data, and further discusses the application of machine learning and data mining algorithms to the adaptation of probabilistic pattern analyzers. Early results indicate potential for user modeling and adaptive personalized recognition of document types and abstract type definitions.

The second component is an information retrieval and information visualization module for comparative evaluation of uploaded and preprocessed data. Views are provided for inspection of aggregate statistics about student scores, comparison over time within one course, or comparison across multiple years. These visualizations include tracking of student performance on a range of standardized assessments. Assessments can be viewed as pre- and post-tests with comparative statistics (e.g., normalized gain), decomposed by answer in the case of multiple-choice questions, and manipulated using prespecified data transformations such as aggregation and refinement (drill down and roll up). The system is designed to support inclusion of a range of supervised inductive learning methods for schema inference, unsupervised learning algorithms for similarity-based retrieval, supervised learning for regression-based time series prediction, and Bayesian models for causal inference on the decision support end.

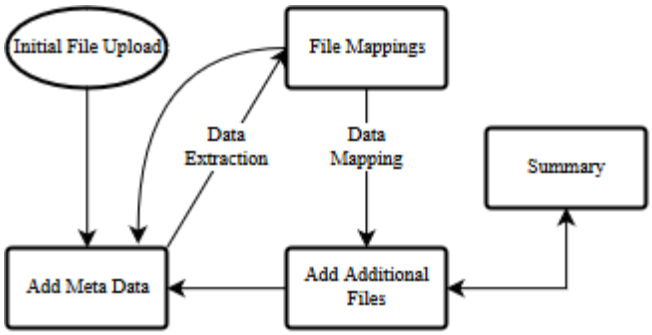


Figure 2 Data flow for importer of Data Explorer.

The users of the system who are usually Physics educators upload their historical assessments through an iterative data upload interface. The data upload interface accepts assessment files that are in a limited set of formats in the current system. The accepted file formats are xls, xlsx, and csv. Simplistic file requirements, which include having a header row and one student per data row, help assure extraction of the correct headers and student data while allowing users to upload a wide range of data formats. Apart from accepting and verifying the integrity of the uploaded files the data upload interface prompts the user to specify meta information ("Add Meta Data" in Figure 2), such as approximate number of students that took the assessment and whether the file contains either pre-, post-, or pre- and post-test assessment data. Some of these assessment features are required, while others are optional. The assessment specific information, such as assessment name and assessment type (belief survey or standard multiple choice), provide a rough estimate of the number of questions (usually represented as columns) that are present within the uploaded, whereas the number of students gives an estimate of the number of rows with student scores. The data upload interface checks the integrity of the file and extracts all the data that is present within the various file types. The extracted data is saved as a data frame, a two-dimensional data structure, where the atomic data items present in the input file are stored in individual cells of

the data frame. The row-column relationships of the data items in the uploaded files are preserved in the data frame.

Parsing Data

The objective of the file parser is to identify the boundaries of the assessment scores within the data frame, as well as identify the location of the headers. The presence of other extraneous legacy information within the data makes the task of extracting payload data from the data frame a complicated exercise. Some of the various kinds of information that is available within these files, apart from the payload, could be the rubric or the scoring criteria for the particular assessment; it could also have information dealing with aggregate student demographic information and other extraneous data. Considering all these variabilities, a heuristics based parser that takes the meta information that is provided during the file upload process to extract the valid assessment payload from the test data is created. The presence of both pre- and post-assessment scores within the same data frame is another degree of freedom that adds to the complexity of the parsing approach.

Heuristic (α)	Description	Condition to Count (σ)	Contributed Value (γ)
String cells	The number of cells in a row that are text.	$> thresh$	1
Integer cells	The number of cells in a row that contain integers.	$> thresh$	1
Float cells	The number of cells in a row that contain floating-point numbers	< 0	-1
Duplicate cells	The number of duplicate cells in a row	$> thresh$	1

Unique cells	The number of unique cells in a row	$< numberOfQuestions$	-1
Pre/Post	Detects whether or not the row contains “pre” or “post”	> 0	1
Long question number	Detects the number of large question numbers (helps when assessment data is outputted by online tools)	$> numberOfQuestions - 10$ $numberOfQuesions > 0$	1
Max consecutive number	Detects the largest consecutive number series in a row after stripped of alpha characters (Q1, Q2, Q3, etc.)	$> thresh$	3
Unique markers	The number of unique known headers (Student ID, Gender, etc.)	> 1	2
Repeated markers	The number of repeated known headers (question, ques, q, pre, post)	$> (thresh - 3)$	2

Table 16 Heuristics for identifying the header row

To identify the boundaries of the payload within the data, the header row of the payload is identified. The header row consists of column names of the various columns available in the assessment scores. These could be student particulars such as name, identifier, or gender, or the particular assessment information, such as grade, question number, or aggregate score. The model consists a series of heuristics that score rows and columns for identifying which row contains column headers, and which rows contain the student data. This helps eliminate user added calculations and miscellaneous data, and extracts relevant student information. Table 16 shows the heuristics for determining the header row, where

$numberOfQuestions$ is equal to the number of questions in the assessment (collected in the add metadata phase) and $thresh = \lfloor numberOfQuestions - (numberOfQuestions * .2) \rfloor$.

This threshold gauges an approximate number of columns to expect for questions; the buffer adds tolerance for poorly formatted files. From Table 16, the header row is defined as $\forall r \in rows$ max($\sum_{\alpha_{i,r}}^{n} \gamma_i \; if \; \sigma_i$ where $\alpha_{i,r}^{m}$ is the heuristics for row r. The header row is then used to determine the table boundaries for relevant student data by comparing each row to row markers from known templates; otherwise, in the case a row is absent of markers, the length of the row (number of non-empty cells) is compared to $thresh$, as defined for Table 16. If a row is blank, a combination of 80% of the class size (given by the user as metadata) and a two-row margin to allow small gaps in student data is used. If this margin is exceeded, and the number rows in the current block of data parsed is less than 80% of the class size, the start of the student data is moved after the blank rows and parsing continues. This allows the parser to skip over blocks of precomputed statistics and other user specific information; however, if the user gives a greatly over or under estimate on class size, files with more than two row gaps in the data underneath header will be unsuccessfully parsed.

The schema inference model can successfully parse 77/80 testing files (a mixture of sanitized real data submitted to the project and synthetic data). A file is parsed successfully if it identified the header row and included all rows of student data. If the parser includes miscellaneous columns of data, the test can pass as these columns can be excluded in post processing; 23 tests were passed in this manner. The last three tests failed due to the assessment answer keys being included as part of the block of student data. This problem can be solved for templated files; however, for semi-structured files, the answer keys cannot be differentiated from student responses. Accuracy of the schema inference during beta testing

and future production deployment is partly dependent on user feedback (missing student rows or columns), as well as the headers that are verified by the user (columns thought to be student data but was not).

Guesser

The guesser module (position in system as "File Mappings" in Figure 2), uses a hybrid similarity measure to detect approximate matches between candidate header strings and template strings. This consists of a convex combination of two edit distance functions (Levenshtein and Jaro-Winkler), both computed by dynamic programming. The weights are calculated using a generalized logistic function:

$$\boldsymbol{w = Y(t) = A + \frac{K - A}{(C + Qe^{-B(t-M)})^{\frac{1}{\nu}}}} \qquad \textbf{Equation 1}$$

where $K = C = 1$, $A = 0.3$, $Q = \nu = M = 5$, $B = 2.7$, and t is the Levenshtein distance. A is the lower asymptote, K is the upper asymptote, B is the growth rate, M is the baseline distance (input), $\boldsymbol{\nu}$ is a skew parameter (for controlling the inflection point), and Q is the baseline weight (output). The final distance measure for strings $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$can then be defined as:

$$\boldsymbol{dist(s_1, s_2) = wd_1 + (1 - w)d_2} \qquad \textbf{Equation 2}$$

where d_1 and d_2 are normalized Jaro-Winkler and thresholded Levenshtein edit distances, respectively, d_{JW} is the raw Jaro-Winkler distance and:

$$\boldsymbol{d_1 = \left(1 - d_{JW}\right)^{\frac{t(M-B)}{M}}} \qquad \textbf{Equation 3}$$

The confidence of a column header labeled as a given class is then given by:

$$\boldsymbol{conf = 1 - dist(header, label)} \qquad \textbf{Equation 4}$$

If the header and the class label both contain numeric parts (i.e. "Question 24"), then distance of the numeric and alpha parts are separately compared combined with weights .75 and .25 respectively. This increases the likelihood of labeling alphanumeric question columns with the

correct question number. If the confidence of the best candidate label for a column header is less than .45, the inferred header in the File Mappings is presented to the user as "Unknown, otherwise the inferred header is shown.

From initial beta testing, inference of column headers shows strong positive results. Although it can match columns in the synthetic data, the model's performance is judged on the data which users have uploaded and completed the file mappings process. In order measure performance, true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) are framed within the problem. If the inferred column header is verified as correct by the user, it is counted as a TP. However, if the inferred header was verified as something different (inferred header is overridden), it is counted as a FP. This incorrect guess would normally be counted as a TN; however, while the task is to infer column headers, excluding columns of extraneous data mingled in with student data is also required. For this reason, if the inferred column header is "Unknown," and the user verifies the header as "Do Not Import," it is counted as a TN since this column is confirmed to be unnecessary for analysis and visualization. If a column header is "Unknown," and the user verifies the column as actual student data, it is counted as a FN.

Assessment	Data Sets	Columns	TP	TN	FP	FN	Accuracy	Precision	Recall	F1
CLASS-Chem	3	34	16	15	1	2	0.9118	0.9412	0.8889	0.9143
CLASS	66	659	516	60	54	29	0.8741	0.9053	0.9468	0.9256
BEMA	38	204	117	38	25	24	0.7598	0.8239	0.8298	0.8269
FCI	198	1193	711	113	249	120	0.6907	0.7406	0.8556	0.7940

MEAT	2	4	2	0	0	2	0.5000	1.0000	0.5000	0.6667
ChCI	1	8	8	0	0	0	1.0000	1.0000	1.0000	1.0000
FMCEv98	10	64	54	3	5	2	0.8906	0.9153	0.9643	0.9391
FMCE	68	317	245	13	35	24	0.8139	0.8750	0.9108	0.8925
MPEX	6	14	9	0	4	1	0.6429	0.6923	0.9000	0.7826
CSEM	18	141	56	2	53	30	0.4113	0.5138	0.6512	0.5744

Table 17 Results showing the performance of the base guesser model by assessment.

The results from the initial user testing are found in Table 17. Data was collected through 84 unique users who have uploaded 410 datasets spread across ten different assessments. Data shown in Table 17 shows the performance of the guesser module on student metadata only (demographics, student records, etc.). Question columns were excluded from analysis due to the system's ability to verify question columns in batches. Once the first question is verified, the rest of the set for that assessment are automatically verified. There is also a significantly larger quantity of question columns compared to student metadata, which inflates the results to be more positive. Still, most assessments maintain a high f1-score. Some of the assessments cannot be evaluated sufficiently due to lack of datasets. After inspecting some of the poorly performing files, most of the incorrectly guessed headers are from ill formatted header rows that contain text not related to the contents of the column. Though, in some cases, the modified edit distance model does not perform well among target headers that contain similar text. For example, a raw header with the text ‘Student’ may end up matching a variety of target headers (‘Student ID’, ‘Student Full Name’, etc.), depending on the columns that were matched earlier.

To overcome this weakness, an active learning approach is used to leverage historical guesses that are overridden by the user. This adds learning to the edit distance model where guesses towards a target label are overridden if the raw header has been seen several times (α) before. If this occurs, the guess is overridden with the most frequently verified target label for that raw header text. This new model, with sufficient data, can be trained to recognize headers that are unique to a specific user. However, less than 50% of beta users have uploaded more than two files and even fewer have sufficient student metadata. Training a model for each user does provide better results when compared to the original model that has not been trained, but until the Data Explorer receives more data, models are trained per assessment. Figure 3 compares the original model that has no override to results from training models requiring a minimum of 1, 2 and 3 historical guesses before override occurs. Leveraging a single historical guess provided over 10% increase in f1-score. Requiring 2 or 3 historical guesses still provides an improvement over the original model; however, this increases the number of datasets to be uploaded before the model is trained. One drawback with this method is when there are many duplicate raw headers in a dataset. This increases the number of target headers associated which decreases the accuracy of correctly overriding the original guess. This same situation arises from generic raw headers like ‘Name’ and ‘Student,’ which can be inaccurate if they have been verified differently within the same user. This is less likely as the number of samples seen increases.

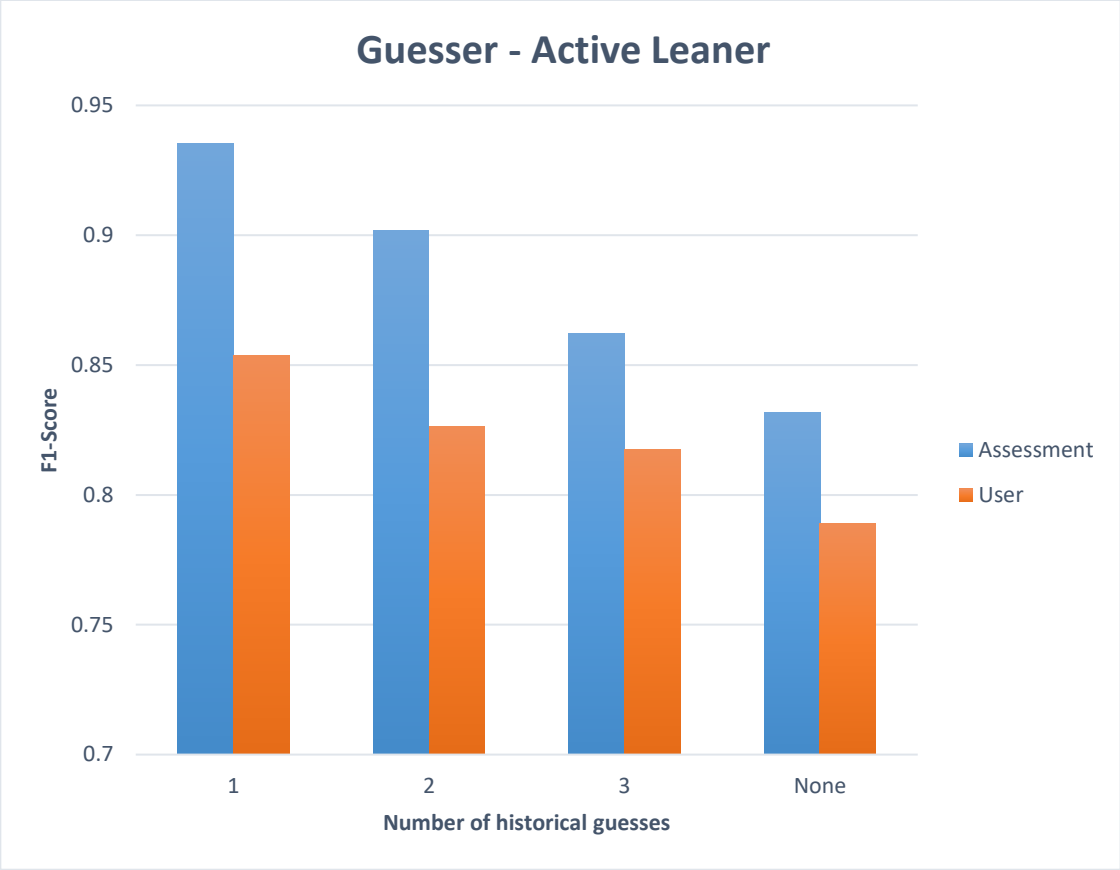


Figure 3 F1-Score of the base model to the active learning model.

Visualization

The information visualization facility of the Data Explorer contains a variety of functions implemented using the D3.js JavaScript library (Bostock, Ogievetsky, & Heer, 2001). Figure 4 shows how normalized (Hake) gain is plotted, with order statistics (mean and median) and standard deviation, for a class's performance on an assessment. Figure 5 shows how the visualization services also allow drill-down ("breakdown") by question, an important type of analytical query that results in the display of a distribution of answers for each question and facilitates comparative analytics for pre- and post-instructional assessments. The objective of these visualizations is to provide instructors with actionable insight concerning: topics covered; the impact of instruction and classwork on student learning as assessed formally

using tests such as FCI, FMCE, and BEMA; and longitudinal trends of concern. In continuing work, additional ways to drill down into multidimensional assessment data, such as using the *TableLens* visualization (Rao & Card, 1994), are being explored.

Figure 4. Data visualizer component of the Data Explorer, displaying a histogram of normalized gain for a hypothetical class on the Force Concept Inventory (FCI) assessment.

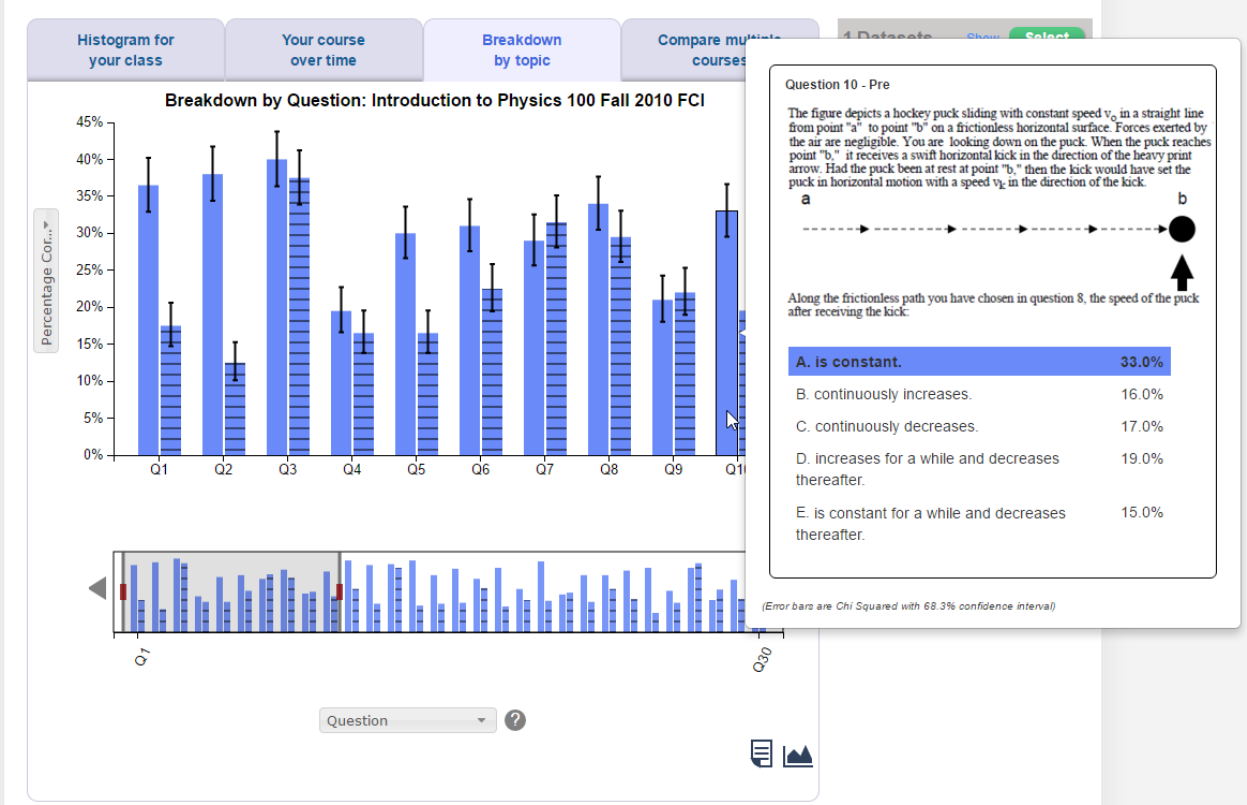


Figure 5. A "Breakdown by Question" view, showing drill-down for a single question and multiple-choice responses, together with the distribution of student responses, on a post-instructional assessment question (also for the FCI).

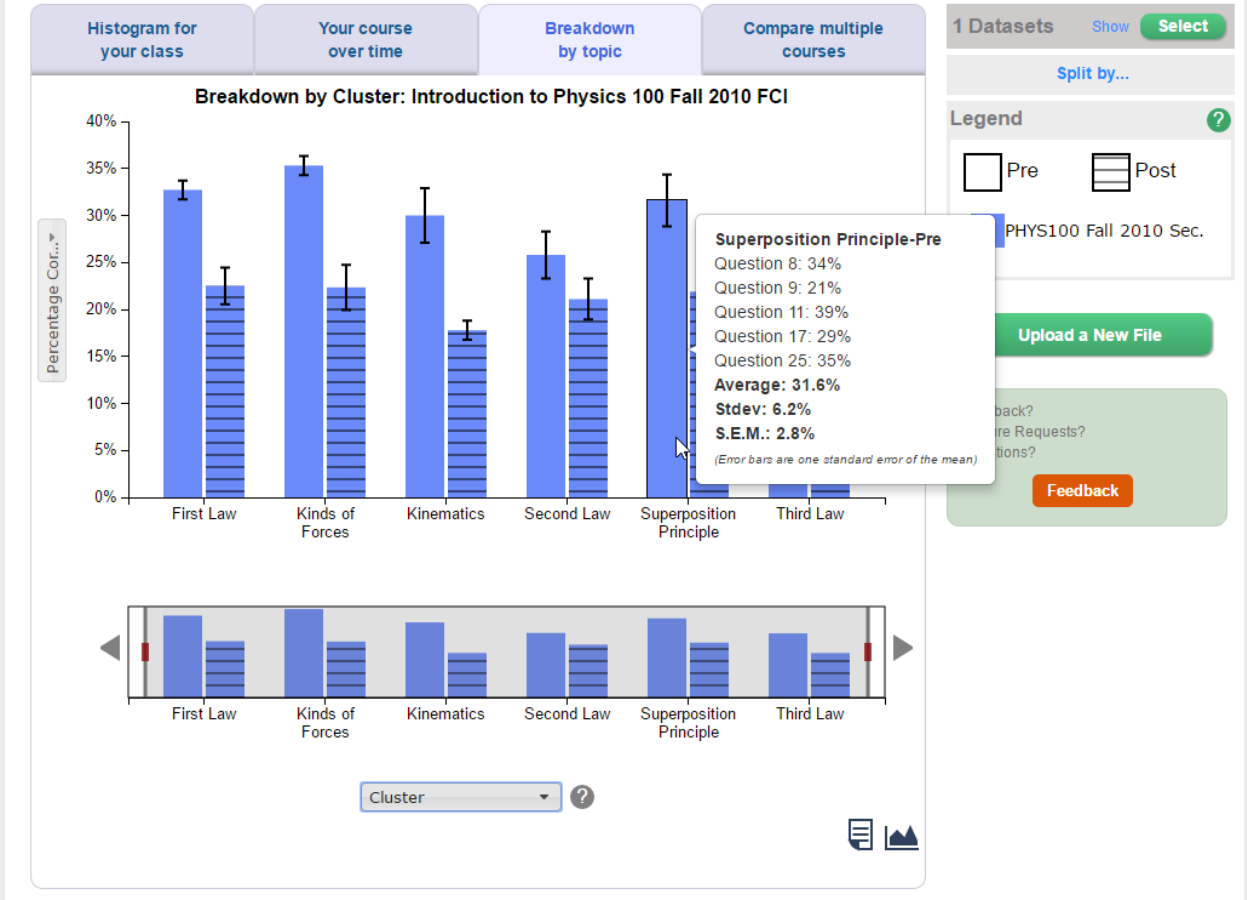


Figure 6. Visualization of student performance on pre- and post- assessment, organized by classification of question. Class labels are assigned by subject matter experts (physics education researchers).

Continuing Work: Information Retrieval and Data Mining

A further capability, designed to facilitate instructor exploration of assessment data, is that of grouping questions by known or discovered category. Figure 6 shows the results of visualizing hand-labeled categories (which are known as *classes* in machine learning, *clusters* in statistics, and *segments* in business analytics). Work in progress aims at using unsupervised learning to perform clustering of assessment questions (by topic modeling or by other similarity-based learning). The key capability that this future work aims at is that of retrieving *classes like mine* relative to longitudinal data (short time series) and similarity measures adapted to such time series. Meanwhile, clustering can also enable similarity-based queries for

time series data (Rafiei & Mendelzon, 1997). The time series consists of student assessment scores and normalized gain measures, and thus admit the same kind of dimensionality reduction and indexing (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001). Ultimately, the goal is to develop a data-driven approach towards concept similarity in assessment data in STEM education, as Madhyastha and Hunt could do to some degree for diagnostic assessments (Madhyastha & Hunt, 2009).

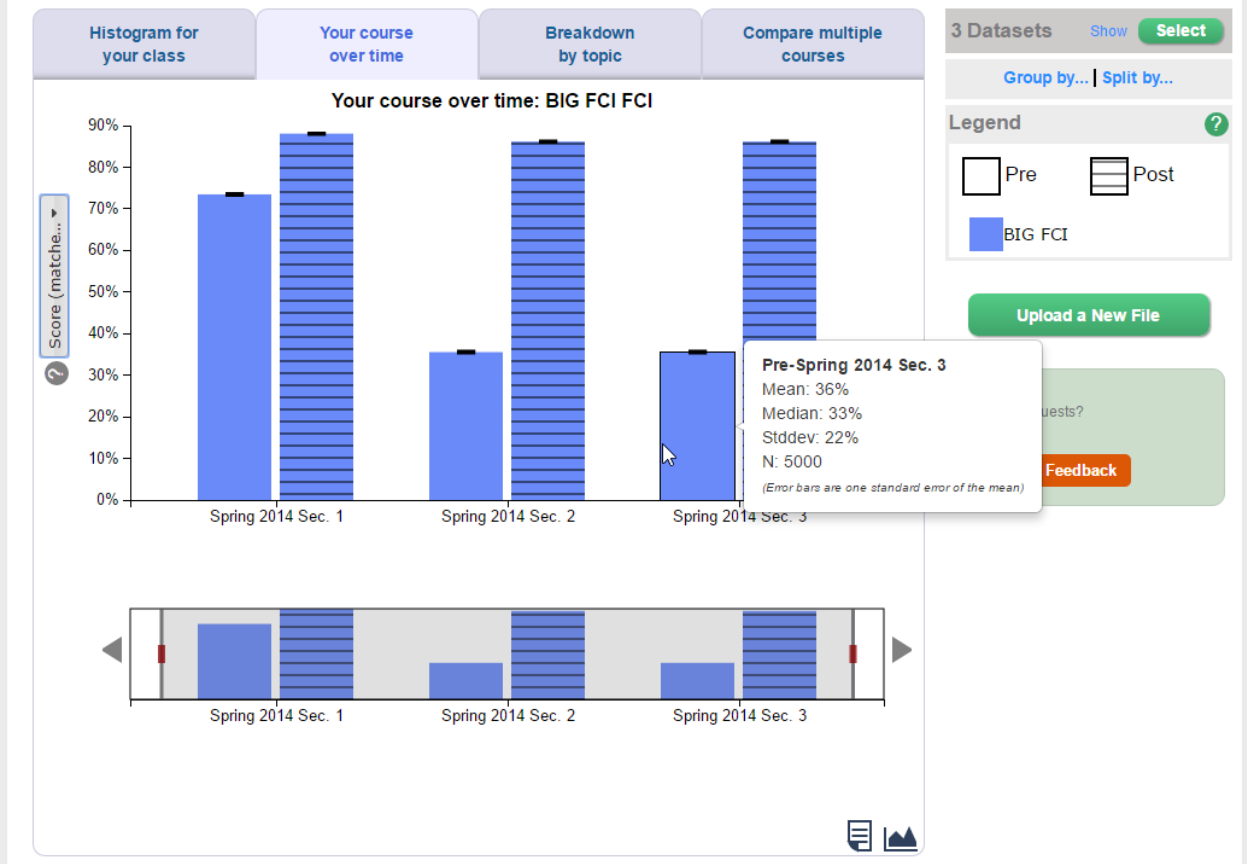


Figure 7. Visualization of courses over time: tracking performance across classes in multiple offerings (semesters and sections) in a longitudinal study.

Future Work: Instructional Decision Support and Adaptive Recommendation

Figure 7 includes a visualization of assessments across multiple courses taught at a single institution, typically by a single instructor under whose login the data are grouped for

multiple semester combinations. The visualization subsystem also provides a facility for drilling down by section. This provides the analytical setting for long-term objectives: to progress from interactive visualization within this federated display to adaptive decision support systems and tutoring systems (Brusilovsky & Millan, 2007).

Conclusion

This chapter presented a data integration and information management system for STEM education research. The functionality outlined in the example screen captures is focused around the continuing research regarding schema inference and educational data mining from student assessments. The key novel contributions with respect to data integration are intelligent systems components for schema inference where columns and other elements are unlabeled, nonstandard, and may include missing data. The novel contribution with respect to analytics are the interactive information visualization components that both provide insights into assessment data and generate requirements for similarity-based retrieval and comparative evaluation of student performance.

Chapter 6 - Summary and Future Work

Summary

Computer science undoubtedly plays a role in all disciplines and daily life, making computational thinking a necessary 21st century learning skill. As a country, the United States still lacks the support to embed this skill into the K-12 curriculum. This leaves CS and education professionals room to only advocate, promote, and create. As part of the research in this report, multiple curricula and pedagogy have been employed and tested in K-12 outreach environments. These methods have shown effectiveness at fostering CT, yet maintain an accessible level for educators who do not have a CS background. Even moving into undergraduate education, this work shows the possibilities and benefits of incorporating CT into other discipline like physics.

Future Work

The work done in this dissertation has room to grow. Work done with the summer STEM Institute will continue. To further the conclusions drawn from self-efficacy, knowledge-based questions are going to be introduced to the camp lessons. These will be introduced through trivia-like games to prevent the class from feeling like 'school' and more like a 'camp.' Furthermore, adjustments will be made to some of the microcontroller curriculum to further reduce cognitive load and to put more emphasis on CT apart from the electronics. Student interviews will also be needed in the future to corroborate findings in links or non-existing links between CT and PS. No changes are currently slated for Starbase; however, the girl scout outreach event is going to be expanded to be a full day event to cover more topics and will include brownies (2nd-3rd), juniors (4th-5th), and cadettes (6th-8th). CT work done for undergraduate physics will also continue to be pursued. The work done in this

dissertation was only an experimental study to look at feasibility and possible ways of approaching the incorporation of CT in physics. Moving forward, the direct programming approach will still be used, but online lecture and support materials will be created to help and support students as they learn the programming language. Rubrics will be created for the CT problems to help link the effects of CT on students' ability to learn physics. Work with physics and the Data Explorer will also continue as the user feedback from the open beta shapes and improves the visualizations, features, and user experience. Future research in the Data Explorer includes automatic clustering of assessment data through topic modeling and student responses, identifying students-like-mine so users can compare their students to similar data across the entire site, and expansion of assessment support, including custom assessments that can be made within the Data Explorer.

Girl Scouts

This section discusses material used for a three-hour, artificial intelligence-themed Scratch event for Girl Scout Juniors (4^{th} and 5^{th} grade), as well as observations made during the event and future goals for the program. Underrepresented groups in STEM have been a large focus in college recruitment, as well as outreach programs. Females in particular, are a largely underrepresented group in computer science (Pivkina, Pontelli, Jensen, & Haebe, 2009; Vekiri & Chronaki, 2008). Research has shown that girls cope equally well with computer science as boys; however, self-efficacy in computers and value in computers is significantly lower in females vs males (Vekiri & Chronaki, 2008). Some outreach programs, designed specifically for young women, have had success in teaching computer science concepts (Pivkina, Pontelli, Jensen, & Haebe, 2009; Denner, Werner, & Ortiz, 2012; Webb & Rosson, 2013). Webb and Rosso also designed an outreach program for young women and showed

that scaffolded examples are an efficient method for teach computational thinking to women (Webb & Rosson, 2013). The Girl Scout event drew from a mixture scaffolding theory and direct instruction, very similar to the work done at the summer STEM Institute as K-State. Attendance consisted of 25 Girl Scout Juniors, as well as three Brownies (2nd and 3rd grade) and one Daisy (Kindergarten and 1st grade). The younger scouts were brought by parents with Juniors and were allowed to participate. Only five or six scouts mentioned they had used Scratch before.

Activities

The first lesson involved a guided exploration of Scratch. In this activity, students were shown how to place, move, and delete blocks, how to move, draw, and use basic loops, and how to modify, add, and delete Sprites and Backdrops. Scouts who had used Scratch before, as well as those who were catching on much faster than others, were encouraged to work ahead and explore what they could find in the language. After this introduction, scouts participated in an activity to draw regular shapes similar to the activity outlined in (Bean, Weese, Feldhausen, & Bell, 2015). At this time, the class was split into two groups. One went on a guided tour of K-State's computing cluster, Beocat, while the others worked on the shape exercise.

Once the groups got to complete the program and the tour, the scouts participated in a CS unplugged activity centered on artificial neural networks (ANNs). In this activity, students were given a paper with segments of ten different pictures. Their task was to guess, solely on their fragment, if a picture was a cat or a dog. Ten numbers were lined around the room with everyone starting at one number in the beginning. If they guessed the picture correctly, they moved up a number, if wrong, they moved down a number. This slowly introduced the basic

understanding of how the brain works. As the activity moved on, different groups started to form that knew certain knowledge of cats and dogs (similar segments of the picture). In the middle of the activity, the students are tricked by being shown a picture of a bear. This initiates a discussion about how humans and computers are only able to make conjectures based on current knowledge, and in this case, students were only given knowledge that they were looking at a picture segment of a cat or a dog. At the end of the activity, there are generally many smaller groups and a couple large groups. Discussion about how the larger groups (generally the students who get segments of pictures with more details like a nose) are more reliable in guessing if a picture is a cat or dog and that their vote carries more weight if the vote for pictures was as a whole, rather than individually.

The theme of artificial intelligence was carried into the last activity, where scouts programmed an AI in a game called Cat and Mouse. The lesson plan for this activity can be found in the Appendix: Introduction to AI: Cat and Mouse Lesson Plans. Before scouts started to program, they were asked what it means to be intelligent. While the most common answer is "being smart," it opens discussion about constructed knowledge, learning, planning, and sensing. Scouts were then asked to link human intelligence to artificial intelligence. After the opening discussion, scouts were directed through how to program a basic completed Cat and Mouse game. Then they were prompted to openly discuss how they could improve the AI (the cat) to make the game more interesting. The event was then wrapped up with the video from Code.org entitled "What Most Schools Don't Teach," allowing a brief discussion on what computer science can do for them as a career.

Summary

Overall, this event was received way beyond what was originally planned. The girls, as well as parents who stayed, were highly engaged and maintained that excitement throughout the time they were there. Many scouts went above and beyond the activities by exploring what they could do in Scratch to make the programs their own. One observation with the Brownies and Daisy that attended was that they did not require any special attention to stay caught up in the activities. This was unexpected, especially with the Daisy scout, although the younger scouts did exhibit some shyness when compared to their older peers. Because this event went over so well, a larger, full day workshop is being organized for 2017. The Girl Scouts of America, especially the size of the organization, provides a great partnership to get girls excited about STEM.

Computational Thinking: Qualitative to Quantitative

Current work for this dissertation has centered on mostly qualitative analysis of CT though self-efficacy. While self-efficacy has been shown to be an accurate measure for computational thinking (Bell, 2014; Bean, Weese, Feldhausen, & Bell, 2015) and computer programming (Ramalingam, LaBelle, & Wiedenbeck, 2004), this report proposes to combine self-efficacy with quantitative analysis, such as static analysis of programs. The motivation for this connection is that previous self-efficacy work (discussed in the Summer STEM Institute 2015 chapter) showed that attitude surveys are not a reliable measurement for CT when students have not had previous exposure to computer science. Students were overconfident in their abilities in the pre-survey, leading to a net loss in the post-test after being introduced to the topics. Some inconsistencies are also present in student answers from within each survey and from pre to post, most likely due to confusion in language or not understanding the question. With these observations in mind, the proposed research will use

quantitative analysis of student problem solving traces and programs. The hypothesis here is that quantitative analysis will reveal students' overconfidence and provide a basis for mapping students' actual ability throughout the course to students' self-efficacy in the post-test. In other words, this will most likely lead to opposing qualitative and quantitative results early in a class, with quantitative results fluctuating as concepts are introduced and plateauing once the concept is learned, and finally converging with post-test self-efficacy.

Scratch to Blockly

Program logging and analysis, including compilation and programming process, is present in current text-based programming literature, such as Blackbox (Altadmri & Brown, 2015), a data logging system from the popular IDE BlueJ used in CS0 and CS1 courses, and CloudCoder (Spacco, et al., 2015), a web based programming exercise delivery system. With these systems in mind, a translation from Scratch to Blockly is proposed. While Scratch is a mature visual programming tool, it is not supported by mobile devices or iPads (a fairly common device in K-12 classrooms). Blockly is developed using JavaScript which is cross platform and runs well on mobile devices. While containing the basis for a lot of visual programming language functionality, it does not provide similar sprite and animation functionality that is present in Scratch and other popular visual based languages. Touch Develop by Microsoft provides cross platform and mobile support, as well as sprite functionality; however, after inspecting source code of both Blockly and Scratch, Blockly's code base was a better option for reskinning and tooling for the purposes of this research. Blockly also implements blocks as a 'toolbox.' This makes it a great platform for not only creating custom blocks, but also gives the ability to limit blocks, providing educators a way to target CT concepts without distracting students with extraneous options. Once Scratch

functionality has been added into Blockly, a new layer will be added to allow fine-grained logging. This will include user interactions with the Blockly interface (button clicks, opening and closing of the block toolbox, switching views, etc.), and more importantly, logging block placement, deletion, movement, and length of time between these actions. The overarching goal of this data logger is to gain an understanding on how students solve programming problems and to detect when they may be having difficulties with a particular concept. This quantitative analysis will provide the basis for interpreting student self-efficacy, as well as a potential predictor of performance.

The current development progress of this tool has been done by a group of senior undergraduate CS students. While the tool has not been completed, the following are some of the major features that have been implemented:

- Blocks have been skinned to look more like scratch. This gives it a more vibrant color and a familiar feel.
- Motion, Pen (drawing), and Scratch control loop blocks have been added
- Basic Stage and Sprite functionality, as well as a console for program output

A major component missing are message blocks. These are used to communicate between scripts and sprites in Scratch and are essential for a multithreaded program. Since Blockly generates JavaScript from blocks and executes it within a sandboxed environment (unlike Scratch which uses Adobe Flash), it will require some major improvements to its interpreter. JavaScript is natively a single threaded programming language, so to execute a Blockly program, the Blockly interpreter must share the execution environment with the web browser. The proposed solution to increase the efficiency of Blockly's execution is to implement a scheduling policy, as well as execute blocks in chunks rather than one at a time. By also using

a timer, this will allow more execution time for Blockly (currently each block command is queued and yields until the browser stack is empty) without starving the UI thread.

Once this functionality is complete, the Scratch to Blockly translation will be piloted in the CS0 course at Kansas State University which currently uses Scratch as its programming language. Self-efficacy will be collected using pre- and post-surveys at the beginning and end of the semester, respectively. Data will be logged from the course programming assignments and tracked throughout the semester. This data will be analyzed using a modified PECT framework (Seiter & Foreman, 2013) to fit the programming assignments, as well as compared to student course grades. Machine learning tools like regression models and Bayesian networks will also be used with features generated from the modified PECT framework to study whether student performance the course and CT can be predicted. To dive deeper in understanding transfer and self-efficacy of CT, future work will also include studying the move from block-based to text-based programming language (Weintrop & Wilensky, Using Commutative Assessments to Compare Conecptual Understanding in Block-based and Text-based Programs, 2015), which is also why Blockly was chosen as the base tool as it provides a block to text-based language translation. This, as well as potential intelligent tutoring system implementations, will be reserved for future work.

Chapter 7 - References

Adelfio, M. D., & Samet, H. (2013). Schema Extraction for Tabular Data on the Web. *39th International Conference on Very Large Data Bases*, (pp. 421-432).

Altadmri, A., & Brown, N. C. (2015). 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 522-527).

Backer, A. (2007). Computational Physics Education with Python. *Computing in Science & Engineering, 9*(3), 30-33.

Bandura, A. (1982). Self-Efficacy Mechanism in Human Agency. *American Pyschologist, 37*(2), 122-147.

Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking : A Digital Age Skill for Everyone. *Learning & Leading with Technology, 5191*, 20-23.

Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community ? *ACM Inroads, 2*(1), 48-54.

Basawapatna, A., Repenning, A., & Koh, K. H. (2015). Closing The Cyberlearning Loop. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 12-17).

Bean, N., Weese, J. L., Feldhausen, R., & Bell, R. (2015). Starting From Scratch: Developing a Pre-Service Teacher Program in Computational Thinking. *Frontiers in Education*.

Bell, R. S. (2014). *Low Overhead Methods for Improving Capacity and Outcomes in Computer Science.* Manhattan, KS: Kansas State University.

Bennett, V., Ioannidou, A., Repenning, A., Kyu Han, K., & Basawapatna, A. (2011). Computational Thinking Patterns. *American Educational Research Association*.

Bliss, J., & Askew, M. (1996). Effective Teaching and Learning: Scaffolding Revisited. *Oxford Review of Education, 22*(1), 37-62.

Blum, L., & Cortina, T. J. (2007). CS4HS: An Outreach Program for High School CS Teachers. *Proceedings of the 38th SIGCSE technical symposium on Computer science education* , (pp. 19-23).

Borcherds, P. (2007). Python: a language for computational physics. *Computer Physics Communications, 177*, 199-201.

Bort, H., & Brylow, D. (2013). CS4Impact: measuring computational thinking concepts present in CS4HS participant lesson plans. *Proceeding of the 44th ACM technical symposium on Computer science education.*

Bostock, M., Ogievetsky, V., & Heer, J. (2001). D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics, 17*(12), 2301-2309.

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design.* Vancouver, BC, Canada: American Educational Research Association.

Brennan, K., Balch, C., & Chung, M. (2014). *Creative Computing.* Harvard Graduate School of Education.

Brusilovsky, P., & Millan, E. (2007). User Models for Adaptive Hypermedia and Adaptive Educational Systems. In *The Adaptive Web: Methods and Strategies of Web Personalization* (pp. 3-53). Berlin: Springer.

Buffum, P. S., Lobene, E. V., Frankosky, M. H., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2015). A Practical Guide to Developing and Validating Computer Science Knowledge Assessments with Application to Middle School. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 622-627).

Burke, Q., & Kafai, Y. B. (2010). Programming & Storytelling: Opportunities for Learngin About Coding & Composition. *Proceedings of the 9th International Conference on Interaction Design and Children - IDC '10.* Barcelona, Spain.

Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., & Zhang, Y. (2008). Uncovering the Relational Web. *Proceedings of the 11th International Worksop on Web and Databases.*

Chonacky, N., & Winch, D. (2008). Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments. *American Journal of Physics, 76*(4), 327-333.

Chonacky, N., & Winch, D. (2017, March 8). *PICUP*. Retrieved from Compadre: http://www.compadre.org/PICUP/webdocs/About.cfm

Chuang, H.-C., Hu, C.-F., Wu, C.-C., & Lin, Y.-T. (2015). Computational Thinking Curriculum for K-12 Education-A Delphi Survey. *International Conference on Learning and Teaching in Computing and Engineering* (pp. 213-214). IEEE Computer Society.

Code.org. (2015). *Code.org*. Retrieved from https://code.org

Code.org. (2017, 03 28). *Dig Deeper into AP Computer Science.* Retrieved from Code.org: https://code.org/promote/ap

Code.org. (2017, 3 28). *Promote Computer Science.* Retrieved from Code.org: https://code.org/promote/ks

College Board. (2016). *AP Computer Science Principles Including the Curriculum Framework.* New York, NY: CollegeBoard. Retrieved 04 02, 2017, from https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf

Dede, C., Mishra, P., & Voogt, J. (2013). Working Group 6: Advancing computational thinking in 21 st century learning. *EDU Summit 2013 TWG6.*

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240-249.

Despain, W. (2013). *100 Principles of Game Design.* New Riders.

Doan, A., Domingos, P., & Halevy, A. (2003). Learning to Match the Schemas of Data Sources: A Multistrategy Approach. *Machine Learning, 50*(3), 279-301.

Dukeman, A., Caglar, F., Shekhar, S., Kinnebrew, J., Biswas, G., Fisher, D., & Gokhale, A. (2013). Teaching computational thinking skills in C3STEM with traffic simulation. In H. A., & P. G., *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data. Lecture Notes in Computer Science.* (pp. 350-357). Berlin, Heidelberg: Springer.

Faber, M., Unfried, A., Corn, J., & Townsend, L. W. (2012). *Student Attitudes toward STEM: The Development of Upper Elementary School and Middle / High School Surveys.* Friday Institute for Educational Innovation.

Falkner, K., Vivian, R., & Falkner, N. (2015). Teaching Computational Thinking in K-6: The CSER Digital Technologies MOOC. *Australian Computing Educaiton Conference*, 27-72.

Fang, J., Mitra, P., Tang, Z., & Giles, C. (2012). Table Header Detection and Classification. *Proceedings of the National Conference on Artificial Intelligence*, (pp. 599-605).

Farris, A. V., & Sengupta, P. (2014). Perspectival Computational Thinking for Learning Physics : A Case Study of Collaborative Agent-Based Modeling. *Proceedings of the International Conference of the Learning Sciences*, (pp. 1102-1106).

Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., . . . Waite, R. (2013). Assessment of Computer Science Learning in a Scratch-Based Outreach Program. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, (pp. 371-376).

Fraser, N. (2015). Retrieved from Blockly: https://developers.google.com/blockly/

Giannakopoulos, A. (2012). *Problem solving in academic performance: A study into critical thinking and mathematics content as contributors to successful application of knowledge and subsequent academic performance*. unpublished Ph.D. Thesis, University of Johannesburg, South Africa.

Google. (2016, April 1). *Computational Thinking Concepts Guide.* Retrieved from Google for Education: https://docs.google.com/document/d/1i0wg-BMG3TdwsShAyH_0Z1xpFnpVcMvpYJceHGWex_c/edit#heading=h.ld02iaxpskpn

Gormally, C., Brickman, P., Hallar, B., & Armstrong, N. (2009). Effects of Inquiry-based Learning on Students' Science Literacy Skills and Confidence. *International Journal for the Scholarship of Teaching and Learning, 3*(2).

Green, A. J., & Gilhooly, K. (2005). Problem Solving. In N. Braisby, & A. Gellatly, *Cognitive Pychology.* Oxford: Oxford University Press.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher, 42*(1), 38-43.

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads, 1*(2), 4-7.

Hmelo-Silver, C. E., Duncan, R. G., & Chinn, C. A. (2007). Scaffolding and Achievement in Problem-Based and Inquiry Learning: A Response to Kirschner, Sweller, and Clark (2006). *Educational Psychologist, 42*(2), 99-107.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems, 3*(3). doi:10.1007/PL00011669

Kirschner, P. A., Sweller, J., & 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.

Knörig, A., Wettach, R., & Cohen, J. (2009). Fritzing – A tool for advancing electronic prototyping for designers. *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction - TEI '09* (pp. 351-358). New York, NY, USA: ACM Press.

Koh, K. H., Basawapatna, A., Bennett, V., & Reppening, A. (2010). Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. *IEEE Symposium on Visual Languages and Human-Centric Computing*, (pp. 59-66).

Kruger, J., & Dunning, D. (1999). Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of personality and social psychology, 77*(6), 1121-1134.

Kuphaldt, T. R. (2014, Feb). *Lessons in Electric Circuits.* Retrieved 2016, from allaboutcircuits: http://www.allaboutcircuits.com/textbook/

Leonard, J., Buss, A., Gamboa, R., Mitchell, M., Fashola, O. S., Hubert, T., & Almughyirah, S. (2016). Using Robotics and Game Design to Enhance Children's Self- Efficacy, STEM Attitudes, and Computational Thinking Skills. *Journal of Science Education and Technology, 25*(6), 860-876.

Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016). Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)* (pp. 211-220). New York, NY, USA: ACM.

Madhyastha, T., & Hunt, E. (2009). Mining Diagnostic Assessment Data for Concept Similarity. *Educational Data Mining, 1*(1), 72-91.

Mishra, P., & Yadav, A. (2013). Of Art and Algorithms: Rethinking Technology and Creativity in the 21st Century. *Tech Trends, 57*(3), 10-14.

Mohr, C., & Mueller, D. (2012). *STARBASE Minnesota long-term follow-up study STARBASE Minnesota long-term follow-up study.* Wilder Research.

Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15* (pp. 21-29). Omaha, NE: ACM.

N. G. (2010). *Common Core State Standards.* Washington, DC: Authors.

Nagy, G., Seth, S., Jin, D., Embley, D. W., Machado, S., & Krishnamoorthy, M. (2011). Data Extraction from Web Tables: the Devil is in the Details. *Proceedings of the 20011 International Conference on Document Analysis and Recognition*, (pp. 242-246).

National Academy of Sciences. (2010). *Report of a Workshop on The Scope and Nature of Computational Thinking.* National Academies Press.

Neilsen, M. L., Shaffer, J., & Johnson, N. (2015). Time Lapse Photography for K-12 Education. *Int'l Frontiers in Education: CS and CE*, 201-207.

NGSS Lead States. (2013). *Next Generation Science Standards: For States, By States.* Washington, DC: The National Academies Press.

Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering, 9*(3), 10-20.

P21 Partnership for 21st Century Learning. (2015, November). Retrieved from http://www.p21.org

Papastephanou, M., & Angeli, C. (2007). CT Beyond Skill. *Educational Philosophy and Theory, 39*(6), 604-621.

Papert, S. A. (1980). *Mindstorms: Children, Computers, And Powerful Ideas.* New York: Basic Books.

Pivkina, I., Pontelli, E., Jensen, R., & Haebe, J. (2009). Young Women in Computing: Lessons Learned. *SIGCSE Bull., 1*(41), 509-513.

Plass, J. L., Moreno, R., & Brunken, R. (2010). *Cognitive Load Theory.* Cambridge University Press.

Polya, G. (1973). *How I solve it: A new aspect of mathematical method.* New Jersey: Princeton University Press.

Rafiei, D., & Mendelzon, A. (1997). Similarity-Based Queries for Time Series Data. *Proceedings of thje ACM SIGMOD International Conference on Management of Data*, (pp. 13-24).

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and Mental Models in Learning to Program. *ACM SIGCSE Bulletin, 36*(3), 171-175.

Rao, R., & Card, S. K. (1994). The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. *Proceedings of the SIGCHI Concerence on Human Factors in Computing Systems*, (pp. 318-322).

Repenning, A., Webb, D. C., Kho, K., Nickerson, H., Miller, S. B., Brand, C., . . . Repenning, N. (2015). Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools though Game Design and Simulation Creation. *ACM Transactions on Computing Education*, 1-31.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., . . . Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM, 52*(11), 60-67.

Roos, K. (2006). An Incremental Approach to Computational Physics Education. *Computing in Science & Engineering, 8*(5), 44-50.

Royal Society. (2012). *Shut Down or Restart? The way Forward for Computing in UK Schools.* The Royal Academy of Engineering. Retrieved from https://royalsociety.org/topics-policy/projects/computing-in-schools/report/

s. G. (2015). *Snowman Gaming: Home of Good Game Design*. Retrieved from YouTube: https://www.youtube.com/channel/UCmY2tPu6TZMqHHNPj2QPwUQ

Savery, J. R. (2009). Overview of problem-based learning: Definitions and Distinctions. *Interdisciplinary Journal of Problem based Learning, 1*(1), 269-282.

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., . . . Verno, A. (2011). *CSTA K-12 Computer Science Standards.* New York: Association for Computing Machinery. Retrieved March 31, 2016, from http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf

Seehorn, D., Pirmann, T., Lash, T., Batista, L., Ryder, D., Sedgwick, V., . . . Uche, C. (2016). *Interim CSTA K-12 Computer Science Standards.* New York, NY: Association for Computing Machinery.

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, (pp. 59-66).

Sengupta, P., Dickes, A., Farris, A. V., Karan, A., Martin, D., & Wright, M. (2015). Programming in K-12 Science Classrooms: Introducing students to visual programming as a pathway to text-based programming. *Communications of the ACM, 58*(11), 33-35.

Sengupta, P., Kinnebrew, J. S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380.

Seoane-Pardo, A. M. (2016). Computational thinking beyond STEM: an introduction to "moral machines" and programming decision making in Ethics classroom. *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM '16* (pp. 37-44). Salamanca, Spain: ACM.

Snow, E., Haertel, G., Fulkerson, D., Feng, M., & Nichols, P. (2010). *Leveraging Evidence-Centered Assessment Design in Large-Scale and Formative Assessment Practices.* Denver, CO: National Council on Measurement in Education.

Sorour, S. E., Goda, K., & Mine, T. (2015). Correlation of Topic Model and Student Grades Using Comment Data Mining. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 441-446).

Spacco, J., DennyPaul, Richards, B., Babcock, D., Hovemeyer, D., Moscola, J., & Duvall, R. (2015). Analyzing Student Work Patterns Using Programming Exercise Data. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*, (pp. 18-23).

Spradling, C., Linville, D., Rogers, M., & Clark, J. (2015). Are MOOCs an appropriate pedagogy for training K-12 teachers computer science concepts? *Computing Sciences in Colleges, 30*(5), 115-125.

Suchanek, F., & Weikum, G. (2013). Knowledge Harvesting in the Big-Data Era. *Proceedings of the ACM SIGMOD International Conferance on Management of Data*, (pp. 933-938).

Sullivan, G. M., & Feinn, R. (2012). Using Effect Size-or Why the P Value is Not Enough. *Journal of Graduate Medical Education, 4*(3), 279-282.

Taylor, J., & King, B. (2006). Using Computational Methods to Reinvigorate an Undergraduate Physics Curriculum. *Computing in Science & Engineering, 8*(5), 38-43.

Vaughn, L. (2008). *The Power of Critical Thinking: Effective Reasoning about Ordinary and Extraordinary Claims.* N.Y.: Oxford University Press.

Vekiri, I., & Chronaki, A. (2008). Gender issues in technology use: Perceived social support, computer self-efficacy and value beliefs, and computer use beyond school. *Computers and Education, 51*(3), 1392-1404.

Venetis, P., Halevy, A., Madhavan, J., Pasca, M., Shen, W., Wu, F., . . . Wu, C. (2011). Reovering Semantics of Tables on the Web. *Proceedings of the 37th International Conference on Very Large Data Bases*, (pp. 528-538).

Voskoglou, M. G., & Buckley, S. (2012, September). Problem Solving and Computers in a Learning Environment. *Egyptian Computer Science Journal, 36*(4), 28-46.

Wang, Y., & Chiew, V. (2010). On the cognitive process of human problem solving. *Cognitive Systems Research, 11*(1), 81-92.

Webb, H. C., & Rosson, M. B. (2013). Using Scaffolded Examples to Teach Computational Thinking Concepts. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, (pp. 95-100).

Weese, J. L. (2016). Mixed Methods for the Assessment and Incorporation of Computational Thinking in K-12 and Higher Education . *Proceedings of the 12th International Computing Education Research Conference (ICER 2016).* Melbourne, VIC, Australia.

Weese, J. L., & Feldhausen, R. (2017). STEM Outreach: Assessing Computational Thinking and Problem Solving. *Proceedings of the 124th American Society for Engineering Education Annual Conference and Exposition (ASEE 2017).* Columbus, OH.

Weese, J. L., & Hsu, W. H. (2016). Work in Progress: Data Explorer – Assessment Data Integration, Analytics, and Visualization for STEM Education Research. *Proceedings of the 123rd Annual American Society for Engineering Education Annual Conference and Exposition (ASEE 2016).* New Orleans, LA, USA.

Weese, J. L., Feldhausen, R., & Bean, N. H. (2016). The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking. *Proceedings of the 123rd American Society for Engineering Education Annual Conference and Exposition (ASEE 2016).* New Orleans, LA, USA,.

Weintrop, D., & Wilensky, D. (2015). Using Commutative Assessments to Compare Conecptual Understanding in Block-based and Text-based Programs. *Proceedings of*

the eleventh annual International Conference on International Computing Education Research - ICER '15, 101-110.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology, 25*(1), 127-147.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Education Fostering Computational Literacy in Science Classrooms: An agent-based approach to integrating computing in secondary-school science courses. *Communications of the ACM, 57*(8), 24-28.

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. *Proceedings of the European Conference on Games Based Learning*, (pp. 549-558).

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2008). Computational Thinking and Thinking about Computing. *22nd IEEE International Parallel and Distributed Processing Symposium* (pp. 3717-3725). The Royal Society.

Wing, J. M. (2011, March 6). Research Notebook: Computational Thinking--What and Why? (J. Togyer, Ed.) *The Link Magazine*. Retrieved March 29, 2016, from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wu, H.-K., & Hsieh, C.-E. (2006). Developing Sixth Graders' Inquiry Skills to Construct Explanations in Inquiry-based Learning Environments. *International Journal of Science Education, 28*(11), 1289-1313.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education, 14*(1), 1-16.

Zhang, Z. (2014). Towards Efficient and Effective Semantic Table Interpretation. *The Sematic Web - ISWC*, 487-502.

Appendix A - STEM Institute 2015 Survey

Student Survey

	A * notes questions inspired by or used from (Faber, Unfried, Corn, & Townsend, 2012), and ** notes questions inspired by or used from (Bean, Weese, Feldhausen, & Bell, 2015).					
	Background – STEM Camp					
0	Which grade will you be in this coming school year?	5th	6th	7th	8th	9th
1	How many ***previous*** summers have you attended the STEM camp?	0	1	2	3	4
1.1	What STEM activities did you participate in?	Free text				
2	Which classes are you taking during this STEM camp?	Four drop down boxes or a list of check boxes for the sessions: ----5-6th grade ------ CSI: Undercover Electronic Textiles GPS and Secret Hideouts Hollywood Science Mission to Mars Monster Storms Outdoor Biology Robotics 1 at STARBASE Robotics 2 at STARBASE Rockin' and Rollin' Coasters Solar Construction Science of Sports Vet Med Wind Energy				-----7-9th grade------ 3D Printing Chemistry of Candy City of Minecraft Exploring Drone Technology Feed Your Head with Tech Ed Flour, Food, and Fido Need for Speed Game Design

3	Which classes have you already taken during this STEM camp?	Same as above question.				
4	Have you participated in other STEM camps, groups, or activities outside of this program?	Yes	No			
4.1	If so, please list.	Free text				
	Background – Computer Technology					
5	Please check all that apply with the location associated. Have you …					
5.1	participated in the hour of code?	School Activity	In the Classroom	Home	Outside Event	Other
5.2	programmed using Scratch?	School Activity	In the Classroom	Home	Outside Event	Other
5.3	programmed using Blockly?	School Activity	In the Classroom	Home	Outside Event	Other
5.4	programmed using Touch Develop?	School Activity	In the Classroom	Home	Outside Event	Other
5.5	programmed using a text based language?	School Activity	In the Classroom	Home	Outside Event	Other
5.6	programmed using languages not listed above?	School Activity	In the Classroom	Home	Outside Event	Other
	Background – Math	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
6	*Math has been my worst subject.					
7	*I would consider choosing a career that uses math.					
8	*I can handle most subjects well, but I cannot do a good job with math.					

9	I can apply math concepts to other subjects.					
10	I would consider a career in math.					
	Background – Science	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
11	*I am sure of myself when I do science.					
12	*I would consider a career in science.					
13	*I expect to use science when I get out of school.					
14	*I know I can do well in science.					
15	*I can handle most subjects well, but I cannot do a good job with science.					
	Background – Engineering and Technology	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
16	*I like to imagine creating new things.					
17	*If I learn engineering, then I can improve things that people use every day.					
18	*I would like to use creativity and innovation in my future work.					
19	*Knowing how to use math and science together will allow me to invent useful things					
20	*I believe I can be successful in a career in engineering or technology.					

	Background – Leadership and Group ….. (21st Century Learning**)	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
21	*I am confident I can lead others to accomplish a goal.					
22	*I am confident I can work well with students with different backgrounds and opinions.					
23	*I am confident I can make changes when things do not go as planned.					
24	*I am confident I can manage my time wisely when working on my own.					
25	I am confident I can manage my time wisely when working in a group.					
	Programing – I can write a computer program which …	0	25	50	75	100
26	**Executes a step-by-step sequence of commands					
27	**Uses loops to repeat commands					
28	**Responds to events like pressing a key on the keyboard					
29	**Does more than one thing at the same time					
30	**Only executes commands when a specific condition is met					

31	**Perform math operations like addition and subtraction					
32	Perform Boolean operations like 5 < 10 < 15					
33	**can store, update, and retrieve values					
34	**can ask for input from the user					
	Programing – When creating a computer program I …	0	25	50	75	100
35	**Make improvements one step at a time, and work new ideas in as I have them					
36	**run my program frequently to make sure it does what I want, and fix any problems I find					
37	**share my programs with others and look at others' programs for ideas					
38	**break my program into multiple parts to carry out different actions					

Teacher Survey

	A * notes questions inspired by or used from (Faber, Unfried, Corn, & Townsend, 2012), and ** notes questions inspired by or used from (Bean, Weese, Feldhausen, & Bell, 2015).					
	Background – STEM Camp					

0	Check those that apply: I am a …	College Student	Pre-Service Teacher	Primary Teacher	Secondary Teacher	Collegiate Instructor /Professor	Administrator
1	Is this your first time teaching at the summer STEM camp?	Yes	No				
1.1	If not, how many previous summers have you taught?	1	2	3	4	5	
1.2	What STEM activities did you teach?	Free text?					
2	Which classes are you teaching during this STEM camp?	Four drop down boxes or a list of check boxes for the sessions: ----5-6th grade ------ CSI: Undercover Electronic Textiles GPS and Secret Hideouts Hollywood Science Mission to Mars Monster Storms Outdoor Biology Robotics 1 at STARBASE Robotics 2 at STARBASE Rockin' and Rollin' Coasters Solar Construction Science of Sports Vet Med Wind Energy				-----7-9th grade------ 3D Printing Chemistry of Candy City of Minecraft Exploring Drone Technology Feed Your Head with Tech Ed Flour, Food, and Fido Need for Speed Game Design	
3	Which classes have you already taught during this STEM camp?	Same as above question.					
4	Have you participated in other STEM	Yes	No				

	camps, groups, or activities outside of this program?					
4.1	If so, please list.	Free text				
	Background – Computer Technology					
5.1	participated in the hour of code?	School Activity	In the Classroom	Home	Outside Event	Other
5.2	programmed using Scratch?	School Activity	In the Classroom	Home	Outside Event	Other
5.3	programmed using Blockly?	School Activity	In the Classroom	Home	Outside Event	Other
5.4	programmed using Touch Develop?	School Activity	In the Classroom	Home	Outside Event	Other
5.5	programmed using a text based language?	School Activity	In the Classroom	Home	Outside Event	Other
5.6	programmed using languages not listed above?	School Activity	In the Classroom	Home	Outside Event	Other
5.1	participated in the hour of code?	School Activity	In the Classroom	Home	Outside Event	Other
	Background – Teaching	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
6	I am comfortable in teaching Math.					
7	I am comfortable in teaching Science.					

8	I am comfortable in teaching Technology.					
9	I am comfortable in teaching Engineering.					
10	Which areas do you teach?					
11	Which areas would you like to teach?					
	Background – Engineering and Technology	Strongly Disagree	Disagree	Neither Agree or Disagree	Agree	Strongly Agree
12	*If I learn engineering and technology, then I can I integrate it into my classroom.					
13	*I would like to use creativity and innovation in my classroom.					
14	*Knowing how to use math and science together will allow me to better integrate engineering and technology into my classroom.					
	Background – Leadership and Group ….. (21st Century Learning**)	0	25	50	75	100
15	*I am confident I can lead others to accomplish a goal.					

16	*I am confident I can teach students with different backgrounds and opinions.					
17	*I am confident I can make changes when things do not go as planned.					
18	*I am confident I can manage my time wisely when working on my own.					
19	I am confident I can manage my time wisely when working in a group.					
	Teaching – Programming	0	25	50	75	100
20	**I feel confident writing simple programs for the computer.					
21	**I know how to teach programming concepts effectively.					
22	**I can promote a positive attitude towards programming in my students.					
23	**I can guide students in using programming as a tool while we					

	explore other topics.					
24	**I feel confident using programming as an instructional tool within my classroom.					
25	**I can adapt lesson plans incorporating programming as an instructional tool to meet my students' learning level.					
26	**I can create original lesson plans incorporating programming as an instructional tool.					
27	**I can identify how programming concepts relate to Common Core Standards.					
28	**I can identify how programming concepts relate to Next Generation Science Standards.					

Appendix B - STEM Institute 2016 Survey

	** notes questions inspired by or used from (Bean, Weese, Feldhausen, & Bell, 2015).					
	Background – STEM Institute					

0	What is your gender?	Male	Female			
1	Which grade were you in at the time you enrolled for the STEM Institute	5th	6th	7th	8th	9th
2	How many ***previous*** summers have you attended the STEM Institute?	0	1	2	3	4
2.1	What STEM activities did you participate in?	Four drop down boxes or a list of check boxes for the sessions: ----5-6th grade ------ CSI: Undercover Electronic Textiles GPS and Secret Hideouts Hollywood Science Mission to Mars Monster Storms Outdoor Biology Robotics 1 at STARBASE Robotics 2 at STARBASE Rockin' and Rollin' Coasters Solar Construction Science of Sports Vet Med Wind Energy		-----7-9th grade------ 3D Printing Chemistry of Candy City of Minecraft Exploring Drone Technology Feed Your Head with Tech Ed Flour, Food, and Fido Need for Speed Game Design		
3	Which classes are you taking this year at the STEM Institute?	Four drop down boxes or a list of check boxes for the sessions: ----5-6th grade ------ Biomechanical Engineering CSI: Uncovering the Truth Electronic Textiles Hollywood Science Monster Storms Robotics 1 at STARBASE Robotics 2 at STARBASE Rockin' and Rollin' Coasters		-----7-9th grade------ 3D Printing Chemistry of Candy City of Minecraft Engineering with Nanotechnology Exploring Drone Technology Fill Your Toolbox Introduction to Passive Architecture		

		Rube Goldberg Challenge Simulating the Martian Solar Construction Science of Sports Vet Med Wind Energy			Grain and Bakery Science Mighty Micro Controllers Robotic Design and Programming	
4	Have you participated in other STEM camps, groups, or activities outside of this program?	Yes	No			
4.1	If so, please list.	Free text				
	Background – Computer Technology					
5	Please check all that apply with the location associated. Have you …	At School	At Home	At a STEM camp, group or activity		Other
	participated in the hour of code?					
	programmed using Scratch?					
	programmed using Blockly?					
	programmed using Touch Develop?					
	programmed using a text based language?					
	programmed using languages not listed above?					
	When solving a problem I …	Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
	Create a list of steps to solve it.					
	Use math operations.					
	Try to simplify the problem by ignoring					

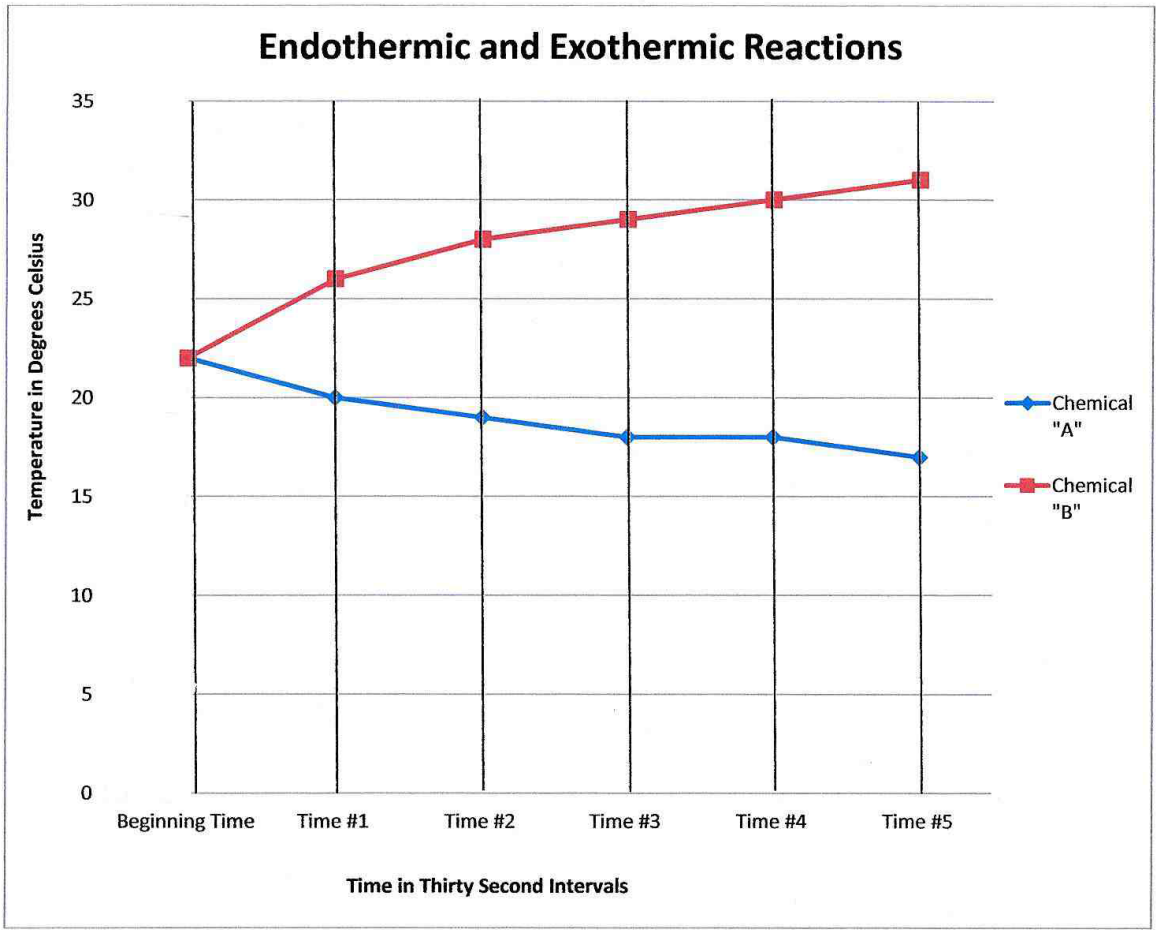
	details that are not needed.					
	Look for patterns in the problem.					
	Break the problem into smaller parts.					
	Look at how information can be collected, stored, and analyzed to help solve the problem.					
	Break problem into parts to be solved by different people at the same time.					
	Create a solution where steps can be repeated					
	Create a solution where certain steps are only done when a condition is met					
	Programing – I can write a computer program which …	Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
	**runs a step-by-step sequence of commands					
	**Uses loops to repeat commands					
	**Responds to events like pressing a key on the keyboard					
	**Does more than one thing at the same time					
	**Only executes commands when a specific condition is met					
	**Perform math operations like addition and subtraction					

	Uses messages to communicate with different parts of the program					
	**can store, update, and retrieve values					
	**can ask for input from the user					
	Programing – When creating a computer program I …	Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
	**Make improvements one step at a time, and work new ideas in as I have them					
	**run my program frequently to make sure it does what I want, and fix any problems I find					
	**share my programs with others and look at others' programs for ideas					
	**break my program into multiple parts to carry out different actions					

Appendix C - Starbase Pre-test

1. What is the first step in the engineering design process?
   A. Create a prototype
   B. Choose the best solution
   C. Define the problem
   D. Select the best Computer-Aided Design (CAD) software to use

2.



Based on the graph above, which prediction is the most logical for the reaction shown?

A. The temperature of Chemical “A” and Chemical “B” will both increase.
B. The temperature of Chemical “A” will increase; Chemical “B” will decrease.
C. The temperature of Chemical “A” will decrease; Chemical “B” will remain constant.
D. The temperature of Chemical “A” will decrease; Chemical “B” will increase.

3. When measuring the volume (or amount) of a liquid, what standard unit of measurement would you use?

A. Liter
B. Gram
C. Celsius
D. Meter

4. Which of the following statements is NOT true about air?

A. Our air is made up of mostly oxygen.
B. Air takes up space.
C. Air has weight and mass.
D. Air can easily change its shape, be separated, poured, and flows; therefore, it is a fluid.

5. Which of the following statements is NOT true about the Periodic Table of Elements?

A. It is a list of all the chemical elements found in our universe.
B. The elements are arranged in alphabetical order.
C. Everything we know of is made of some combination of the elements on the periodic table.
D. It is a way of organizing the elements based on their properties.

6. A scientist is studying an object on the nanoscale with a scanning electron microscope. Objects on the nanoscale measure 1 to 100 nanometers in size. 1 nanometer = 10^{-9} meter. Which object would the scientist be viewing?

A. A stick of gum
B. The width of a DNA helix
C. A human hair
D. A dust mite

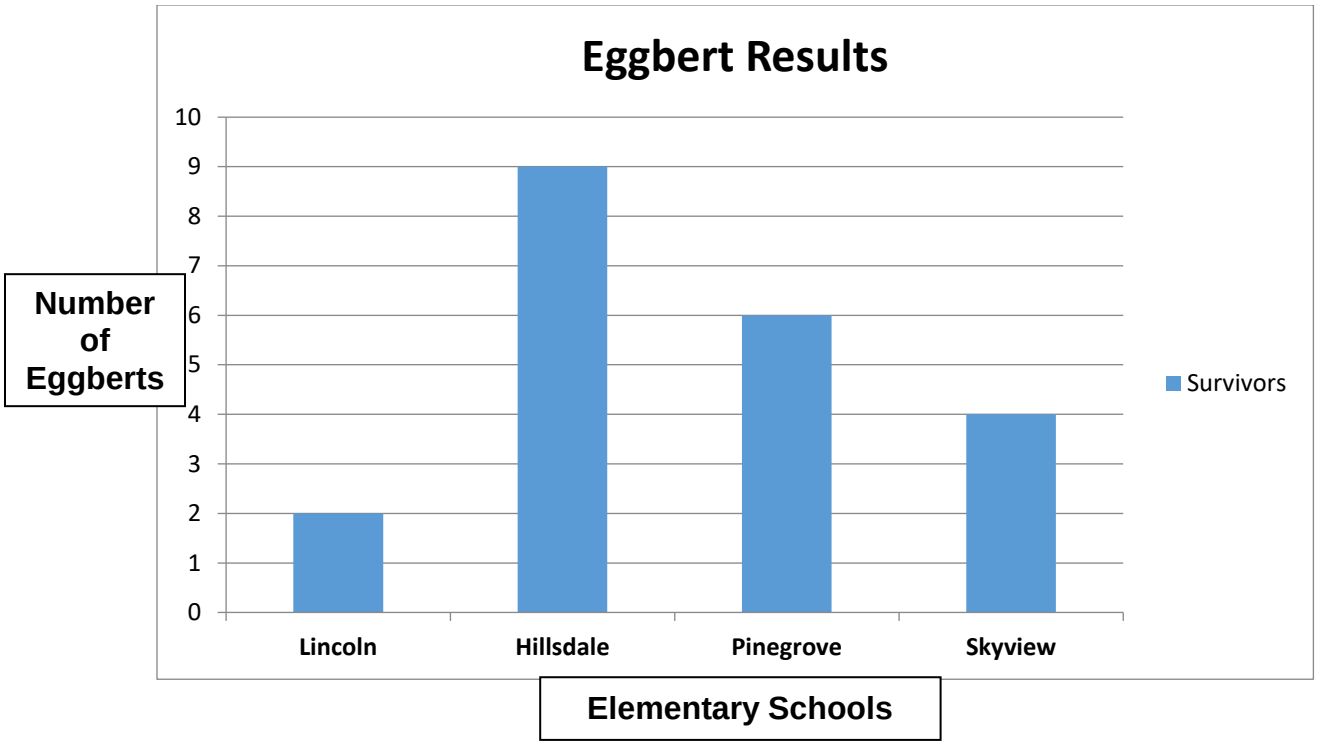
7. Which of the following would be a good reason to use latitude and longitude coordinates?

A. Find a specific point or location
B. Analyze the weather pattern of a region
C. Determine the terrain of an area
D. Determine the depth of a canyon

8. One carbon atom (C) and two oxygen atoms (O) chemically bond together to form carbon dioxide (CO_2). CO_2 is best described as…

A. a nucleus.
B. a compound or compound molecule.
C. an atom.
D. an element.

9. Examine the bar graph below. Determine which two schools combined had one fewer Eggbert survivor than Hillsdale Elementary School.



A. Lincoln and Skyview
B. Pinegrove and Skyview
C. Lincoln and Pinegrove
D. Skyview and Hillsdale

10. Which of the following is NOT a chemical reaction, but rather is a physical change?

A. Baking a cake
B. Mixing two compounds together producing bubbles
C. Freezing water
D. Burning paper

11. At sea level air presses down 14.7 pounds on every square inch of our bodies. The reason we don't feel this pressure is…

A. the air is thinner closer to the ground than up in space.
B. the atmosphere cushions the weight of the air.
C. our bodies push out 14.7 pounds on every square inch to equalize the pressure.
D. we are in a building so we don't feel it.

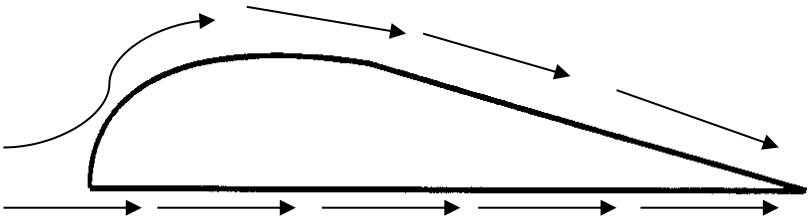
12. If you launched three similar rockets, using the same force, which rocket would go the highest?

A. The rocket weighing 50 grams goes the highest.
B. The rocket weighing 100 grams goes the highest.
C. The rocket weighing 150 grams goes the highest.
D. They all go the same height.

13. Which of the following Newton's Law of Motion makes it important to wear a seat belt?

A. Newton's Law of Motion which explains that the greater the mass of an object, the greater the force needed to accelerate it.
B. Newton's Law of Motion (Inertia) which explains that an object in motion will stay in motion unless acted upon by an outside force.
C. Newton's Law of Motion which explains that for every action there is an equal and opposite reaction.
D. None of the Above

14. One reason an airplane is able to produce lift is because the air moving across the top of the wing…



A. exerts more pressure on the wing than the air moving along the bottom.
B. exerts the same amount of pressure on the wing as air moving along the bottom.
C. exerts less pressure on the wing than the air moving along the bottom.
D. does not exert any pressure on the wing.

15. When you sprain an ankle, you need to apply an activated cold compress to relieve the swelling. Which of these does the activated cold compress produce?

A. a hydrophobic reaction
B. an endothermic reaction
C. an exothermic reaction
D. a hydrophilic reaction

16. Which of the following machines is NOT a robot?

A. Television digital video recorder (DVR)
B. da Vinci Surgical System
C. Electric pencil sharpener
D. Dimension 3-D printer

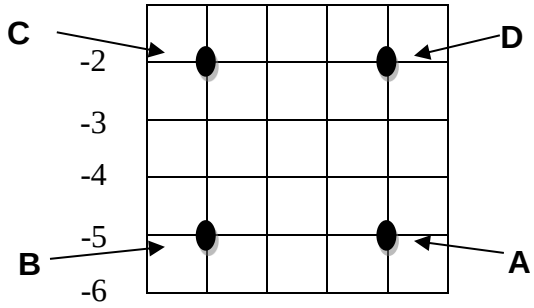
17. Two boats are heading upstream and water is flowing downstream. Based on the Bernoulli Principle, fast moving water between the boats will cause the boats to...

A. move further apart.
B. come closer together.
C. continue on the same path.
D. none of the above

18. Which point is located at the coordinates (5, -5)?

1 2 3 4 5 6
-1

A. A
B. B
C. C
D. D



19. In which way would an engineer most likely use a Computer-Aided Design (CAD) program?
A. To write a proposal for a project
B. To determine the cost of labor for a building project
C. To design a bridge
D. To communicate with other engineers via email

20. H_2O can exist in several states of matter. In which form do H_20 molecules have the most kinetic energy or motion?
A. Ice
B. Water
C. Steam
D. Snow

21. Right now,
A. I enjoy math.
B. I think math is okay.
C. I don't enjoy math.

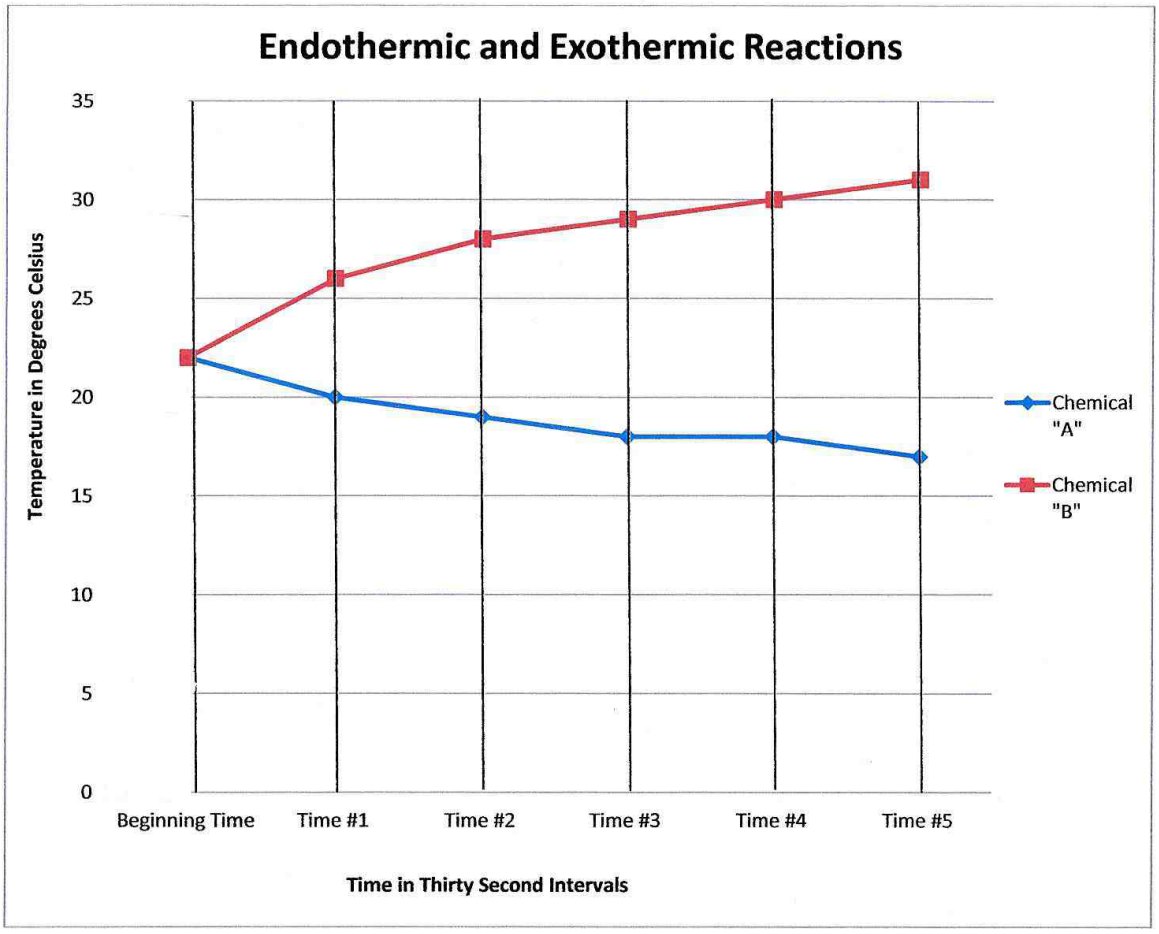
22. Right now,
A. I enjoy science.
B. I think science is okay.
C. I don't enjoy science.

23. Right now,
A. I enjoy technology.
B. I think technology is okay.
C. I don't enjoy technology

Appendix D - Starbase Post-test

1. What is the first step in the engineering design process?
   E. Create a prototype
   F. Choose the best solution
   G. Define the problem
   H. Select the best Computer-Aided Design (CAD) software to use

2.



Based on the graph above, which prediction is the most logical for the reaction shown?

   E. The temperature of Chemical “A” and Chemical “B” will both increase.
   F. The temperature of Chemical “A” will increase; Chemical “B” will decrease.
   G. The temperature of Chemical “A” will decrease; Chemical “B” will remain constant.
   H. The temperature of Chemical “A” will decrease; Chemical “B” will increase.

3. When measuring the volume (or amount) of a liquid, what standard unit of measurement would you use?

E. Liter
F. Gram
G. Celsius
H. Meter

4. Which of the following statements is NOT true about air?

E. Our air is made up of mostly oxygen.
F. Air takes up space.
G. Air has weight and mass.
H. Air can easily change its shape, be separated, poured, and flows; therefore, it is a fluid.

5. Which of the following statements is NOT true about the Periodic Table of Elements?

E. It is a list of all the chemical elements found in our universe.
F. The elements are arranged in alphabetical order.
G. Everything we know of is made of some combination of the elements on the periodic table.
H. It is a way of organizing the elements based on their properties.

6. A scientist is studying an object on the nanoscale with a scanning electron microscope. Objects on the nanoscale measure 1 to 100 nanometers in size. 1 nanometer = 10^{-9} meter. Which object would the scientist be viewing?

E. A stick of gum
F. The width of a DNA helix
G. A human hair
H. A dust mite

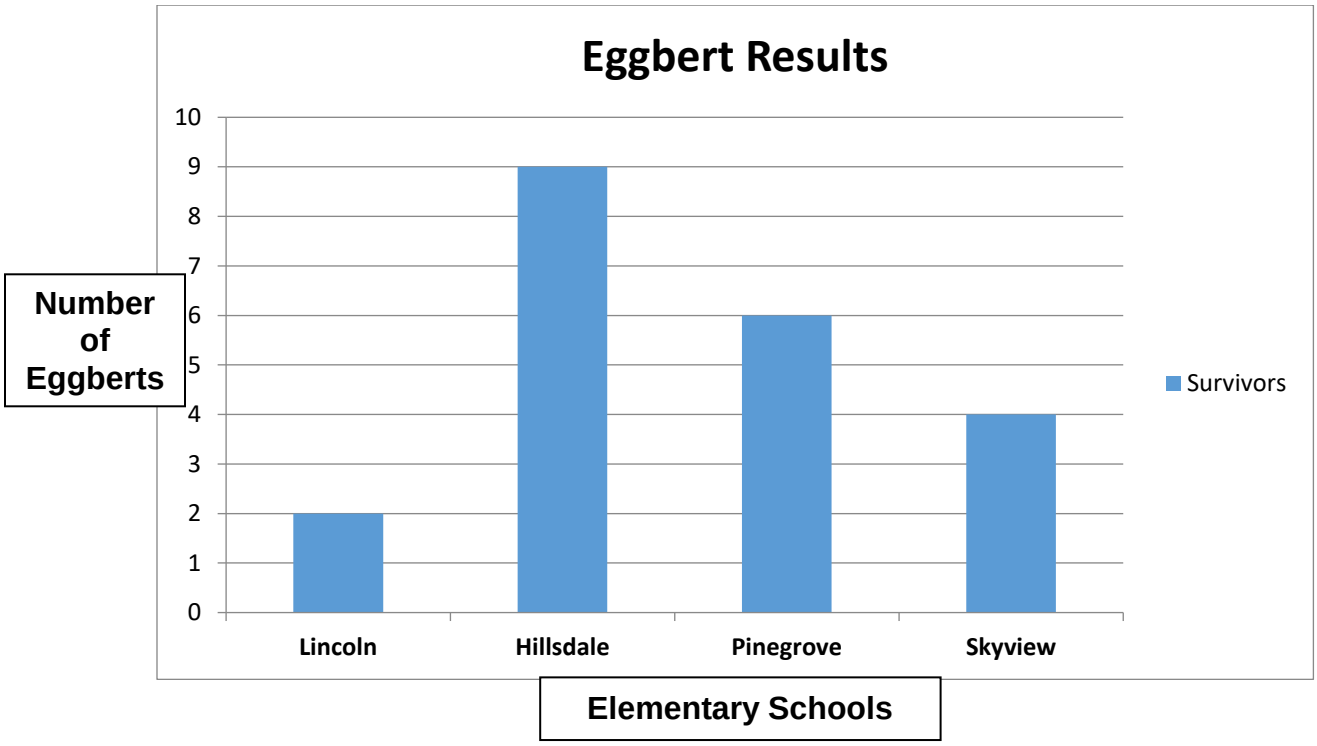
7. Which of the following would be a good reason to use latitude and longitude coordinates?

E. Find a specific point or location
F. Analyze the weather pattern of a region
G. Determine the terrain of an area
H. Determine the depth of a canyon

8. One carbon atom (C) and two oxygen atoms (O) chemically bond together to form carbon dioxide (CO_2). CO_2 is best described as…

E. a nucleus.
F. a compound or compound molecule.
G. an atom.
H. an element.

9. Examine the bar graph below. Determine which two schools combined had one fewer Eggbert survivor than Hillsdale Elementary School.



E. Lincoln and Skyview
F. Pinegrove and Skyview
G. Lincoln and Pinegrove
H. Skyview and Hillsdale

10. Which of the following is NOT a chemical reaction, but rather is a physical change?

E. Baking a cake
F. Mixing two compounds together producing bubbles
G. Freezing water
H. Burning paper

11. At sea level air presses down 14.7 pounds on every square inch of our bodies. The reason we don't feel this pressure is…

E. the air is thinner closer to the ground than up in space.
F. the atmosphere cushions the weight of the air.
G. our bodies push out 14.7 pounds on every square inch to equalize the pressure.
H. we are in a building so we don't feel it.

12. If you launched three similar rockets, using the same force, which rocket would go the highest?

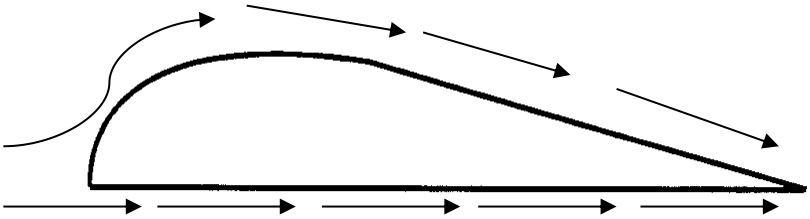
E. The rocket weighing 50 grams goes the highest.
F. The rocket weighing 100 grams goes the highest.
G. The rocket weighing 150 grams goes the highest.
H. They all go the same height.

13. Which of the following Newton's Law of Motion makes it important to wear a seat belt?

E. Newton's Law of Motion which explains that the greater the mass of an object, the greater the force needed to accelerate it.
F. Newton's Law of Motion (Inertia) which explains that an object in motion will stay in motion unless acted upon by an outside force.

G. Newton's Law of Motion which explains that for every action there is an equal and opposite reaction.
H. None of the Above

14. One reason an airplane is able to produce lift is because the air moving across the top of the wing…



E. exerts more pressure on the wing than the air moving along the bottom.
F. exerts the same amount of pressure on the wing as air moving along the bottom.
G. exerts less pressure on the wing than the air moving along the bottom.
H. does not exert any pressure on the wing.

15. When you sprain an ankle, you need to apply an activated cold compress to relieve the swelling. Which of these does the activated cold compress produce?

E. a hydrophobic reaction
F. an endothermic reaction
G. an exothermic reaction
H. a hydrophilic reaction

16. Which of the following machines is NOT a robot?

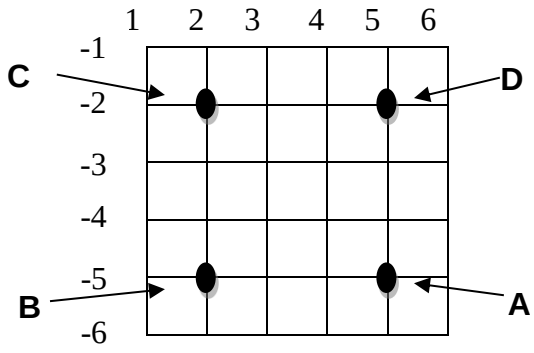
E. Television digital video recorder (DVR)
F. da Vinci Surgical System
G. Electric pencil sharpener
H. Dimension 3-D printer

17. Two boats are heading upstream and water is flowing downstream. Based on the Bernoulli Principle, fast moving water between the boats will cause the boats to...

E. move further apart.
F. come closer together.
G. continue on the same path.
H. none of the above

18. Which point is located at the coordinates (5, -5)?

E. A
F. B
G. C
H. D



19. In which way would an engineer most likely use a Computer-Aided Design (CAD) program?

E. To write a proposal for a project
F. To determine the cost of labor for a building project
G. To design a bridge
H. To communicate with other engineers via email

20. H_2O can exist in several states of matter. In which form do H_20 molecules have the most kinetic energy or motion?

E. Ice
F. Water
G. Steam
H. Snow

21. Right now,

D. I enjoy math.
E. I think math is okay.
F. I don't enjoy math.

22. Right now,

A. I enjoy science.
B. I think science is okay.
C. I don't enjoy science.

23. Right now,

A. I enjoy technology.
B. I think technology is okay.
C. I don't enjoy technology

24. STARBASE made learning fun:
A. Yes
B. No

25. I want to come back to STARBASE:
A. Yes
B. No

26. After studying math and science at STARBASE, I want to learn even more about math and science:
A. Yes
B. No

27. Finish the following sentence. "When I talk about STARBASE to my family and friends I say… _____

_____."

1. The Engineering Design Process (EDP) can be repeated more than once for the same problem
    a. Yes
    b. No
2. I think working in groups:
    a. Helps me solve problems and complete tasks
    b. Does not help me solve problems or complete tasks
3. I am confident that I can program a robot to navigate and complete a task.
    a. Yes
    b. No
    c. I'm not sure
4. I can apply and use computer programming in areas other than Computer Science and Engineering.
    a. Yes
    b. No
    c. I'm not sure
5. Right now
    a. I enjoy computer programming
    b. I think computer programming is OK

c. I don't enjoy computer programming

Appendix E - Introduction to AI: Cat and Mouse

Lesson Plans

Lesson

Cat and Mouse – An Introduction to Artificial Intelligence

Time

45-60 minutes, depending if the optional extra material is presented.

Group Size

Can be done with any number of students. It is intended for individuals; however, it can be done in pairs.

Materials

- www.scratch.mit.edu
    - An offline version can also be used
- The starter project is located here: https://scratch.mit.edu/projects/21341974/
- The completed project is located here: https://scratch.mit.edu/projects/21319246/
- A slightly expanded completed project is located here: https://scratch.mit.edu/projects/23787898/
    - This project is an example of what students can do to expand and enhance the game from the basic, completed program.

Objectives

- Introduce the basics of Artificial Intelligence (AI) and what it means to be intelligent
- Introduce what an agent is in computer science
- Computational Thinking concepts:
    - Data – Students will learn how to keep score and control the speed of the sprites by using variables. An advanced concept in Scratch, known as cloning, will also be introduced. This creates copies of sprites which makes it difficult to conceive.
    - Parallelization – Students will have to manage multiple sprites as well as maintain a consistent state.
    - Flow control – Students will be introduced to message passing, along with other concepts like conditionals, loops, and blocking (wait blocks).

    - Algorithmic thinking – Students will need to recognize the order in which events have to happen to make the game work as intended (when to get points, when the game is over, when to create clones, etc.)
    - Patterns – Students will need to recognize the patterns present in the game's design in order to enhance them if the time permits.
- Students will get a feel for basic game design and how to write AI in Scratch

Introduction

The goals of this lesson is to introduce students to the idea of AI, the study and design of intelligent agents (software that exhibits intelligent behavior). Advanced concepts in Scratch are also introduced, such as message passing (broadcast) and cloning. To start this lesson, present the class with the question "What does it mean to be intelligent?" The responses to this question will vary, but a general consensus will usually be that to be intelligent, you must be smart. This is a common misconception across many students. While it's common to use smart and intelligent interchangeably, smart is generally a status achieved once someone has had advanced experience in a subject matter. On the other hand, intelligence refers back to our primal skills. Intelligence can include many different skills and traits, but the following are the main abilities:

- Reasoning and problem solving – Our ability to overcome complex situations and to think logically
    - A good running example of this is the classic children's puzzle with the different blocks to place in various shaped holes. Use this as a running example when explaining the rest of the intelligent concepts.
- Perception – Our ability to sense things from our environment
    - See and feel the difference in shapes. As a side example, we can also link touching a hot pan on the stove to constructing knowledge and learning.
- Construct knowledge – Our ability to gather and store information we perceive (our experiences).
    - We begin to store information on each block we pick up and attempt to put through the different shaped holes. Likewise, we remember that the pan on the stove is hot, as well as the associated pain with it.
- Learning – Our ability to learn from the knowledge and experiences. We can now recognize that each block is different. We can recognize the difference between shapes (circles, squares, etc.) from our experience in trying to place blocks in the wrong shaped holes. Likewise, we learn not to touch things that are extremely hot, like a pan on the stove.

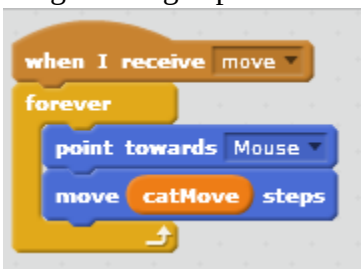
    - Planning – Our ability to premeditate our actions. From our learned concepts/experiences, we can construct detailed instructions to achieve certain tasks, like putting various blocks in the correctly shaped hole. Another example to include in this description would be going to the grocery store. As kids, we go with our parents to buy groceries and slowly learn what it entails. Discuss how we start out by making a

list, drive to the store, and collect the items we need at the store, pay for them, drive home, and put those items away.

Procedure

1. Open up the starter project and give the class a minute to run it to see what it does.
    a. Talk through the game: control the mouse using the arrow keys to eat as many pieces of cheese before the cat gets you
    b. Discuss the starter script in the mouse sprite. The controls for the mouse are already programmed, to save time; however, these can be removed and done during the activity if time permits.
    c. Point out the variable that controls the mouse's speed
    d. When I receive Move acts as the starting point for the game. The green flag acts as an initialization step for the game
2. Switch to the Cat sprite and add a "forever" block from the *control* pallet to "when I receive move" and add a "point towards" block (make sure you point towards the *mouse* sprite) inside the "forever" block:

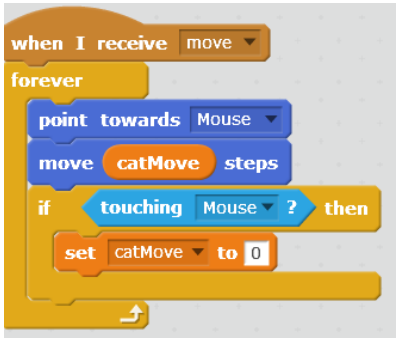
    a. Stop and let the students run the program to see how it works now. The cat sprite should now always face the mouse wherever it goes on the stage
3. Next, connect a "move 10 steps" block to the "point towards" block. Then place the "catMove" variable block from the *data* pallet in the move block. Make sure help the class recognize that "catMove" is a variable (data storage). The value is set to 5 when the green flag is pressed.



    a. Pause to make sure the cat sprite chases after the mouse. The cat may look like it glitches if it catches the mouse, but this will be solved in the next steps.
4. Once the cat catches the mouse, it needs to stop! Place an "if" block from the control pallet after the move block. If the class has not been introduced to "if" blocks before, discuss what a conditional is (The diamond portion of the "if" block will be evaluated it be true or false. If it is true, the blocks inside the "if" portion of the conditional will be executed).
    a. Place a "touching ?" block in the diamond of the "if" and select "Mouse"

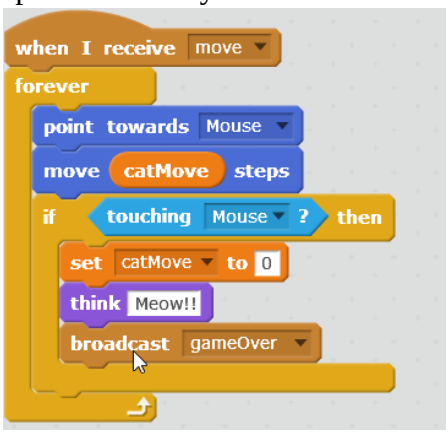
b. Inside the "if" block, put a "set to" block from the data pallet. Set the variable *catMove* to 0

c. Pause and run the program. The cat should now stop once it touches the mouse.



5. From the *looks* pallet, place a "think" block after the "set *catMove* to 0" block and change "Hmm" to "Meow!!"

a. After the "think" block, place a "broadcast" block from the *events* pallet and select the *gameOver* message. This sends the *gameOver* message to **all** the sprites, which can choose to either catch the message or ignore it. The mouse sprite is the only one that catches the message and stops the game.

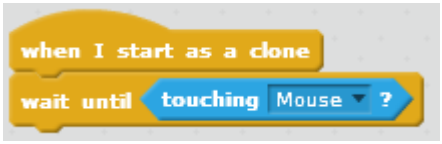


b. Pause to allow students to test their program. Make sure that if the cat catches the mouse, the game properly ends.

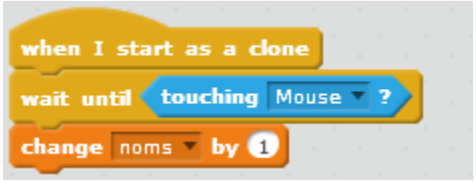
6. Now we will work on letting the mouse eat pieces of cheese to gain points. Switch to the cheese sprite and discuss the "when green flag clicked" script. This is another example of variables. We use *cheeseX* and *cheeseY* to set the coordinates of the cheese sprite on the stage. Note that the stage is an XY coordinate system where X ranges from -240 to 240 and Y ranges from -170 to 170. This is why we use these numbers when picking random numbers for the cheese location. It allows the cheese to appear randomly anywhere on the stage. The ranges can be slightly reduced in order to prevent the cheese spawning on the edges; however, that is not necessary for this exercise. After picking a random XY location, the "go to x: y:" block changes the location of the cheese sprite on the stage. Then the sprite is made visible *only* to be cloned, and then hidden for the remainder of the game. This applies to the *original* cheese sprite. The created clone is a copy of the original, including its visibility and variables. Having the mouse interact with a clone of the cheese sprite allows us to

continuously generate new pieces of cheese as the mouse eats it; all the while introducing students to a complex mechanism in Scratch eats them. Make sure the students connect to the notion that "when I start as a clone" only modifies properties and variables for that clone (in CS the term *this* is used), not the original sprite.

7. Connect a "wait until" block from the control pallet to "when I start as a clone." This block forces the script to pause until the condition inside the diamond becomes true. This can also be referred to as *blocking* in CS terms and is a commonly used technique in parallel programming.
    a. Place a "touching ?" block inside the diamond of the "wait until" block and select the Mouse sprite from the dropdown menu. This pauses the script until the Mouse sprite touches the cloned cheese.

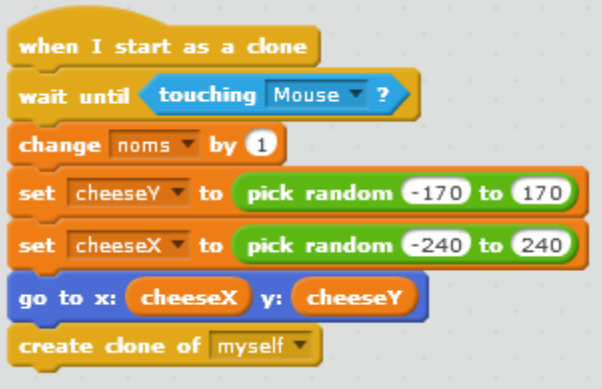


8. This is the point where we want to give the Mouse a point for "eating" a piece of cheese. Place a "change by" block from the data pallet right after the "wait until touching Mouse?" block. Select the *noms* variable from the dropdown menu of the "change by" block and make sure the value is 1. Remember that the code *after* the "wait until touching Mouse?" block will execute once the Mouse sprite touches the cloned cheese.



    a. After the score (*noms*) is updated, we want to set a new random location for the cheese since the mouse is "eating" this piece. Luckily, we already have most of this code done underneath the "when green flag is clicked" hat block in the cheese sprite. To make copies of code blocks in Scratch, right click the first block of the portion of code you wish to copy, and then click "duplicate." In our case, right click the first "set to" block under the "when green flag is clicked" hat block and duplicate. Connect this cloned code to the "change *noms* by 1" block and delete the "show," "hide," and "stop this script" blocks from the cloned code. This can be done by dragging the blocks to the area

underneath the block pallets on the left of the script area.



b. We now should be getting new pieces of cheese showing up at random locations once the mouse touches the cheese; however, after the first time the mouse gets a piece of cheese, the clones won't disappear. Place a "delete this clone block" to the end of the "when I start as a clone" script in order to simulate the mouse "eating" the cheese once it catches one.

9. The game is now complete! Let the student run the program to test, and play as time permits.

Assessment

This simple activity allow students to expand and improve the game. If time permits, encourage students to improve the AI. Before letting them modify the game, discuss possible changes. For example, the advanced cat and mouse game (listed in the Materials section) works by increasing the difficulty of the game after the player has 10 noms. The speed of the cat gradually increases every nom after 10 until the cat moves as fast as the mouse. This greatly reduces the room for error by the player and gives the cat a better chance in catching the mouse. Give the students 15 mins (more or less depending on constraints) to modify their Cat and Mouse game to see what they can improve or change about the AI behavior. After time is up, openly discuss each student's solution.

Appendix F - STEM 2015 Lesson Plans

These lesson plans were generated in by the co-teachers from USD 383 for the camp.

Game Design

Day 1

Teacher: Mr. Peters and Mr. Weese

Date:

Subject / grade level: 7-9

Materials:
Computers, paper, pencils,

NC SCOS Essential Standards and Clarifying Objectives

Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.

Lesson objective(s):

1. **Identify the basic principals of game design.**
2. **Perform basic operations in the Scratch program, including move, sound, control, and sensing blocks.**

ENGAGEMENT

- Icebreaker: "Fact or Fiction" – (Instructors will of course introduce themselves to the students) Students will be asked to write down two facts about themselves and one fiction. They will read those things to the class and the rest of the students will determine which one is fiction.
- Set expectations for students during the week.
- Students will also introduce themselves to the class in general: Name and school they attend at least.
- Student Survey on Computer Coding

EXPLORATION

- Whole Group: Instruction on principles of game design.
  - What makes a game…a game? Students will be given instruction on story, characters, mechanics, and other game design principles.
  - Principals (presentation)
- Basic Overview / Review of Scratch
  - Hopefully most of the students have some familiarity with Scratch. We will do a basic review of the Scratch components and programming blocks needed.
  - Stage, blocks, variables

EXPLANATION

- Artificial Intelligence Focus
    - One of the most important aspects of a video game are the opponents. Most enemies or opponents in games are not played by another human character, but by the computer. How does that work?
    - We will use a simple AI program called "Cat and Mouse" to have students work specifically with the control, motion, and sensing blocks in the program.
        - How can we use the motion and sensing blocks to get the cat to chase the mouse on its own? What kind of Artificial Intelligence is this?

ELABORATION

- Game Design Teams (Random assign because they will take forever to pick someone and others will be left out.)
    - Students will be placed in a team of 2-3 depending on the size of the class.
    - These groups will begin discussing what kind of games they play and what kind of game they want to build.
        - How will it look? What will the story be? What main sprits / characters will be needed?
        - What category of game do you want to make?

EVALUATION

- We will have student groups share out their ideas on their games and provide feedback to any potential problems that we can see.
- Students will have written their ideas on paper and turn them in at the end of the day so they can be reviewed.

Day 2

Teacher: Mr. Peters and Mr. Weese

Date:

Subject / grade level: 7-9

Materials:
Computers, paper, pencils, storyboard templates, Scratch block laminated pages

NC SCOS Essential Standards and Clarifying Objectives

Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.

Lesson objective(s):

3. **Identify the basic principals of game design.**
4. **Perform basic operations in the Scratch program, including move, sound, control, and sensing blocks.**

ENGAGEMENT

- Review from yesterday's lesson
    - How does the game Pacman work? How does that game use artificial intelligence?

EXPLORATION

- Pacman: "Intelligent Agents"
    - Students will play the Pacman game on Scratch. We will have them identify the game design components.
    - We will then look over the code with the students to see how the game works.

EXPLANATION

- Strikers 1945 "Scrolling Shooter"
    - We will have students play this new type of game. They should now fully understand the required components of a game.
    - How does the AI work in this game? What is different about this game?
    - We will walk through helping students complete the different portions of this game.

ELABORATION

- Game Design Teams
    - Game teams will be working on their video game storyboards. Teachers will circulate around the room to assist students in their development.

EVALUATION

- Student teams will turn in their video game storyboards for review. If there is time we will have the groups share out their plans so far for their games.

Day 3

Teacher: Mr. Peters and Mr. Weese

Date:

Subject / grade level: 7-9

Materials:
Computers, paper, pencils, storyboard templates, Scratch block laminated pages

NC SCOS Essential Standards and Clarifying Objectives

Lesson objective(s):

5. **Create a video game that implements the core game design principles.**
6. **Identify careers that are related to the computer science field.**

ENGAGEMENT

- What is a Dungeon Crawler? We will show examples of dungeon crawler games.

EXPLORATION

- Game Teams
  - Students will work on their video games that they have planned.
  - Teachers will move around the different groups to help where needed and to remind students that they need to meet the basic criteria of game design.

EXPLANATION
N/A

ELABORATION

- Career Discussion
  - We will show a video that explains to students what opportunities are in the computer science field.
  - We will also explain that most jobs that involve coding are not related to video games.

EVALUATION

- Students will continue to work on their games and we will ask each group what they have left to accomplish for tomorrow.

Day 4

Teacher: Mr. Peters and Mr. Weese

Date:

Subject / grade level: 7-9

Materials:
Computers, paper, pencils, storyboard templates, Scratch block laminated pages

NC SCOS Essential Standards and Clarifying Objectives

Lesson objective(s):

7. **Create a video game that implements the core game design principles.**
8. **Identify careers that are related to the computer science field.**

ENGAGEMENT

- Students will use the first part of class time to finish their video games.

EXPLORATION

EXPLANATION
N/A

ELABORATION

EVALUATION

- Students will present their video games to the rest of the class.
- STEM Survey

Coding to Mars

Day 2

Teacher: Brooke Snyder Class: Coding/Mission to Mars

Lesson Title: Rocket Trajectory Simulation
Lesson Objectives: Students will create a computer based simulation on Scratch to further explore trajectory modeled by stomp rockets

Anticipated Learner Outcomes: Students will describe the relationship between angle and trajectory and explain them as dependent and independent variables. Students will realize that many variables need to be constant and the use of a controlled environment (simulation) will create more reliable and valid data.
KCCR Standards Math: CCSS 8.F.B.4 Use functions to model relationships between quantities. , CCSS 7.EE.B.4 Solve real life and mathematical problems using numerical and algebraic expressions and equations. KCCR Standards Science: NGSS Grades 9-12 Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.
Materials: Stomp Rockets, open field, computers with internet access, Scratch (scratch.mit.edu), Women in Computer Engineering presentation, pencils, paper, sticky notes.
Learning activities
Engage (describe how teacher will capture students' interest): Ask students to do a round robin and provide as many characteristics, abilities, attributes of rockets. List student responses on the board. Tell them they will be doing one type of simulation, stomp rockets.
Explore (describe hands-on/minds-on activity students will be doing): Take students to an open field. They will line up behind two stomp rockets. Ask students what considerations we should make in order to have successful trajectory. All students several times to try their stomp rocket, encouraging manipulation of the angle. Have them count in paces the distance their rocket traveled. After a significant amount of exploration, have students return to the room. Ask them to talk at their tables the variables that accounted for a successful or unsuccessful flight. Start to discuss the independent and dependent variables (angle, applied force, and trajectory) and the other variables that should remain constant (position of flight pad, wind/resistance, applied pressure, paces to measure).
Explain (list questions & strategies that allow students to communicate what they learned): Click here to enter text.
Elaborate (describe how students expand on concepts & make connections): Students will sense that it takes a lot of computers and programming to create a simulation and record data for numerous rockets. Ask students how that could be possible. Elicit conversation, then support thinking about a super computer. Tour BeoCat , a super computer located in Nichols Hall. Show them the many components and the research tracking capabilities of this computer.
Evaluation (describe how students will demonstrate their learning): Students will do a stand up/hand up, pair up (Kagan Structure) to discuss what they learned for the day. We will visit the Parking Lot/schema-misconception chart.

Vocabulary or topics discussed
Science = variables, independent, dependent, constant, trajectory, acceleration
Technology = simulation,
Engineering = coding, infinite machines.
Mathematics = x,y variables, functions, equations to determine acceleration

Which STEM careers were highlighted? = Computer Programming and Coding (Women in Computer Engineering)

Day 3

Teacher: Brooke Snyder Class: Coding/Mission to Mars

Lesson Title: Artificial Intelligence
Lesson Objectives: Students will compare and contrast the abilities of Artificial Intelligence and Human Sources and Receivers
Anticipated Learner Outcomes: Students will code an AI on a drafted PacMan stage. Students will experiment with their created scripts to determine success. Scripts contain conditionals and variables.
KCCR Standards Math: CCSS 8.F.B.4 Use functions to model relationships between quantities. , CCSS 7.EE.B.4 Solve real life and mathematical problems using numerical and algebraic expressions and equations. KCCR Standards Science: NGSS Grades 9-12 Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.
Materials: clip of Big Hero Six (?) Turing Test diagram/picture, Chinese room example, picture clips with corresponding cell table on computer, paper signs with numbers 1-5, pencils, paper, computers, Scratch (scratch.mit.edu)
Learning activities
Engage (describe how teacher will capture students' interest): Ask students what artificial means and follow up with what it means to be intelligent. Show them a trailer from Big Hero Six. Ask then how BaMax may fit the profile of an AI. Have them discuss in their groups. Tell then that coders can create AI's. What questions would you ask to determine if who/what you are dealing with is an AI? (look for questions that elicit emotion, consistent answers the entire time, many possible outcomes). Introduce them to the "turing test" and Chinese Room experiment (on saved presentation).

Explore (describe hands-on/minds-on activity students will be doing): Tell students they are going to take a 'turing test' in which they will process input as an AI would. Pass out picture grid cards. Tell students the cards contain cut up pictures of a cat or dog. Tell them they will look at their clip, determine if it is a cat or dog. Then, tell them to close their eyes and raise their hand if it is a cat or dog. Write down how many of each got cat dog, then reveal the entire image. Continue this process for all of the images. One image is of a bear. Why did you only answer cat or dog? (they were "coded" at the beginning to only choose cat or dog). Why did you believe this image could be a cat or dog (fur, beady eyes, nose, etc). Tell them because an AI is coded to determine only certain outcomes, it did not pass the turing test. After looking at the data on the board from cat/dog responses, coordinate responses on a corresponding grid. Mark grids for correct responses with Y and N for incorrect. Ask students to determine any patterns in the grid. Hopefully they will see there are more Y's in the middle of the grid (where the animal's most determining features would be) and N's in the borders (vague details). How did this simulation resemble a turing test?
Explain (list questions & strategies that allow students to communicate what they learned): Explain to students that an AI will be able to navigate through a game independently if coded correctly. Give them the link for the PacMan game. Tell them the basic controls for each sprite. Allow them to play the game. After some time, they will notice that only PacMan moves and the ghosts stay in their spot. Have them code Blinkly (Russ has the step by step for this, along with Pinky and Clyde). Students will be creating variables, move, perceive, and act blocks for each ghost.
Elaborate (describe how students expand on concepts & make connections): Students will then be introduced to the Mars Rover Game. The Game has been scripted for the operator to move their sprite to collect samples. The operator can only collect samples on smooth terrain (light grey) and then they can transmit after 3 processes. However, the only 85 moves are allotted, so the operator must think ahead before moving. After allowing time to experiment with the game, allow students to share strategies for how they obtained high scores. Some may have said they recoded the amount of moves allotted for their rover. Ask them if there is a way to code their rover to navigate on its own throughout the mission. Pay attention to any students that mention artificial intelligence or coding it to do so.
Evaluation (describe how students will demonstrate their learning): Students will do an inside/outside circle to discuss the following: What are three careers you think AI's will be doing more than humans in the next 50 years? What are three careers/jobs you think humans will still do more than computers?

Vocabulary or topics discussed

Science = attributes, characteristics

Technology = Artificial Intelligence,

Engineering = perceive, motion, act, variable blocks

Mathematics = Coordinates, outliers, data, probability

Which STEM careers were highlighted? = Coding artificial intelligences, turing test creators, robotics, game coding

Appendix G - STEM 2016 Lesson Plans

These lesson plans were generated in by the co-teachers from USD 383 for the camp.

Mighty Micro Controllers

Day 1

Day One Lesson Title: Getting into Circuits

Lesson Objectives:
*Students will identify an arduino and how it functions.
*Students will examine the Fritzing program and its connection to circuit boards

Anticipated Learner Outcomes:
*Students will understand the role of waves in information transmission *Students will understand the role of resistors in a circuit

KCCR Standards Math:

KCCR Standards Science: NC SCOS Essential Standards and Clarifying Objectives

*Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.
*Develop or modify a model based on evidence to match what happens if a variable or component of a system is changed.
* A wave model of light is useful for explaining brightness, color, and the frequency-dependent bending of light at a surface between media. (MS-PS4-2) However, because light can travel through space, it cannot be a matter wave, like sound or water waves. (MS-PS4-2)
*HS-PS4-5. Communicate technical information about how some technological devices use the principles of wave behavior and wave interactions with matter to transmit and capture information and energy
*Systems can be designed to cause a desired effect. (HS-PS4-5)

Materials: Computers, paper, pencils, arduinos, LEDs
Learning activities
Engage (describe how teacher will capture students' interest): *Icebreaker: "Fact or Fiction" – (Instructors will of course introduce themselves to the students) Students will be asked to write down two facts about themselves and one fiction. They will read those things to the class and the rest of the students will determine which one is fiction. *Set expectations for students during the week. *Students will also introduce themselves to the class in general: Name and school they attend at least. *Student Survey on Computer Coding
Explore (describe hands-on/minds-on activity students will be doing): *Basic Overview / Review of Scratch Hopefully most of the students have some familiarity with Scratch. We will do a basic review of the Scratch components and programming blocks needed. Stage, blocks, variables * Resistance is Futile Experiment what happens when you use stronger resistors *Students will create a Blinking LED with their arduino.
Explain (list questions & strategies that allow students to communicate what they learned): *Students Draw diagram for 4-5 LEDs, create their own blinking LED *What role do the resistors play? *What evidence do we have of wave behavior?
Elaborate (describe how students expand on concepts & make connections): *Students experiment with what happens to LEDs when you use stronger resistors
Evaluation (describe how students will demonstrate their learning): *We will have student groups share out their ideas on their games and provide feedback to any potential problems that we can see. *Students will have written their ideas on paper and turn them in at the end of the day so they can be reviewed.

Vocabulary or topics discussed

Science =resistors, Ohm's Law, electricity, circuit, waves

Technology =arduino, Fritzing software, LED

Engineering = Click here to enter text.

Mathematics = Click here to enter text.

Which STEM careers were highlighted? = Programmer

Day 2

Lesson Title: Day 2—Digital vs. Analog, Open vs. Closed Circuit

Lesson Objectives:
*Students will compare and contrast analog and digital input
*Students will examine how ultrasonic and and motion sensors function
*Students will compare and contrast an open and closed circuit

Anticipated Learner Outcomes:

*Students will understand the role of RGBs in determine color of LEDs *Students will understand how waves are used in sensors

KCCR Standards Science: NC SCOS Essential Standards and Clarifying Objectives

*Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.

* MS-PS4-3. Integrate qualitative scientific and technical information to support the claim that digitized signals are a more reliable way to encode and transmit information than analog signals.
* MS-PS4-2. Develop and use a model to describe that waves are reflected, absorbed, or transmitted through various materials.
* PS4.C: Information Technologies and Instrumentation
* Digitized signals (sent as wave pulses) are a more reliable way to encode and transmit information. (MS-PS4-3)
* HS-PS4-2. Evaluate questions about the advantages of using a digital transmission and storage of information.
* Information can be digitized (e.g., a picture stored as the values of an array of pixels); in this form, it can be stored reliably in computer memory and sent over long distances as a series of wave pulses. (HS-PS4-2),(HS-PS4-5)
* PS4.C: Information Technologies and Instrumentation
* Multiple technologies based on the understanding of waves and their interactions with matter are part of everyday experiences in the modern world (e.g., medical imaging, communications, scanners) and in scientific research. They are essential tools for producing, transmitting, and capturing signals and for storing and interpreting the information contained in them. (HS-PS4-5)

Materials: Computers, paper, pencils, arduinos, LEDs, sensors

Learning activities

Engage (describe how teacher will capture students' interest): *Icebreaker
*Fading LED demonstrations
*Experimenting with sensors
*Review from yesterday's lesson

Explore (describe hands-on/minds-on activity students will be doing): *Students will use analog input to fade LEDs *Students will begin constructing ultrasonic and motion sensors * Students will work with buttons to open and close circuits
Explain (list questions & strategies that allow students to communicate what they learned): *What is the difference between an open and closed circuit? What are the advantages of each? *Students draw diagram on Fritzing to demonstrate fading LEDs
Elaborate (describe how students expand on concepts & make connections): *Students use knowledge of circuit design from Day 1 to create open/closed circuits *Students use prior knowledge of digital input to compare with analog input
Evaluation (describe how students will demonstrate their learning): *Students demonstrate their fading LEDs *Students create and demonstrate an open and closed circuit

Vocabulary or topics discussed

Science =open/closed circuit

Technology =sensor, analog vs. digital, pulse width modulation

Engineering =

Mathematics =

Which STEM careers were highlighted? =Traffic controllers, Security engineers

Day 3

Lesson Title: Day 3: Putting it all together

Lesson Objectives:
*Students will use prior knowledge of class content to create their own project, demonstrating ability to design a circuit, provide coded input, use Scratch code and Fritzing diagramming software

Anticipated Learner Outcomes:

* Students demonstrate ability to use prior knowledge of Scratch and Arduino to create their own project incorporating use of LEDs, RGB lights, push-button circuit, motion or ultrasonic sensors.

KCCR Standards Math: Click here to enter text.

KCCR Standards Science:
*HS- ETS1-2. Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.
* NC SCOS Essential Standards and Clarifying Objectives
Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.
HS-ETS1-4. Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Materials: Computers, paper, pencils, arduinos, LEDs, sensors

Learning activities

Engage (describe how teacher will capture students' interest):
*Icebreaker with pre-service teachers
*Simon Says game
*Individual time to create project

Explore (describe hands-on/minds-on activity students will be doing):
*Students will complete their sensor project and fading LEDs if not already finished *Students will begin work on their traffic light/Piezzo buzzer/self-determined projects

Explain (list questions & strategies that allow students to communicate what they learned): *Why is order of coded commands important? *How is the timing of a system determined? *What are the smaller components of a large system and how do they fit together?
Elaborate (describe how students expand on concepts & make connections): *Students use traffic and buzzer project to connect their prior knowledge of circuits and sensors and apply it to solve new problems
Evaluation (describe how students will demonstrate their learning): *Students begin to design their own individual project demonstrating their mastery of the skills and content covered on Days 1&2.

Vocabulary or topics discussed

Science =

Technology = Click here to enter text.

Engineering = Click here to enter text.

Mathematics = Boolean phrase, conditional statements

Which STEM careers were highlighted? = Click here to enter text.

Day 4

Lesson Title: Day 4: Ultrasonic sensors and Freestyle: Your own project

Lesson Objectives: *Students demonstrate ability to apply previous knowledge of coding, circuitry, sensors, resistors, etc., to solve a problem with their own individual plan in a culminating project.
Anticipated Learner Outcomes: *Students make connections between small-scale circuits from class and larger-scale, real-world examples of circuits at work
KCCR Standards Math: KCCR Standards Science: * ETS1.C: Optimizing the Design Solution Criteria may need to be broken down into simpler ones that can be approached systematically, and decisions about the priority of certain criteria over others (trade-offs) may be needed. (HS-ETS1-2) * NC SCOS Essential Standards and Clarifying Objectives Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.
Materials: Computers, paper, pencils, arduinos, LEDs, sensors
Learning activities
Engage (describe how teacher will capture students' interest): *Ice breaker activity with pre-service teachers
Explore (describe hands-on/minds-on activity students will be doing): *Students code and wire ultrasonic sensor, Students continue work on final project *Students interview a professional programmer over Skype
Explain (list questions & strategies that allow students to communicate what they learned): *How effective was my design strategy? *How did I use circuitry and coding to solve a problem? *How did we use math and formulas to determine distance with out sensor?
Elaborate (describe how students expand on concepts & make connections): *Students connect use of sensors to application in the real-world, including echo-location

Evaluation (describe how students will demonstrate their learning): *Students will present their final projects to the class

Vocabulary or topics discussed

Science = echo-location, sonar, sound waves

Technology = Click here to enter text.

Engineering =

Mathematics = conversion of units, using algebraic formulas

Which STEM careers were highlighted? = Computer programmer

Simulating the Martian

Day 1

Lesson Title: Introduction to Scratch and Its Components
Lesson Objectives: Students will be able to successfully navigate through the scratch program and its components. Anticipated Learner Outcomes: Students will be able to navigate and use Scratch with a basic knowledge, build scripts, and draw within the program.
KCCR Standards Math: **5.OA.A.1** Use parentheses, brackets, or braces in numerical expressions and evaluate expressions with these symbols. **5.G.A.2** Represent real world and mathematical problems by graphing points in the first quadrant of the coordinate plane, and interpret coordinate values of points the context of the situation. KCCR Standards Science: **ETS1** Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem –How well did the solutions meet the criteria and constraints of the design problem-communicate with peers about proposed solutions, share ideas that can lead to improved designs.
Materials: SlidesSlides: http://people.cis.ksu.edu/~russfeld/presentations/stem2014_day1.html * Code.org video: https://www.youtube.com/watch?v=nKIu9yen5nc

* Scratch Website: http://scratch.mit.edu
* Scratch Wiki on Blocks: http://wiki.scratch.mit.edu/wiki/Blocks
* Scratch Spirograph: http://scratch.mit.edu/projects/21326308/

Learning activities
Engage **Brooke Teach** Teachers will introduce themselves and share the learning objectives/outcomes of the day. Go around the room and have each student introduce his or herself to the group. Present the question prompt of, " During camp, I wish to learn----------." Give them think time. Then, do hand up stand up, pair up (Kagan Structure) Ask students to then share their ideas or the ideas of someone they talked to. Direct their attention to the parking log (area designated for sticky notes). Invite them to put questions, comments, concerns, ah hah's in the parking lot to visit during wrap up/reflection time.
Explore: Tell students that one of the reasons we are able to do many things we do is because of computers. Share that Computer Science is a growing field and computers are vital to the enhancement of future. Tell them they will be viewing a video from a non-profit organization entitled, Code.org. Direct their attention to the guiding questions (What are your thoughts on this video, Do you think learning how computers work is important? Explain your thinking.

What are some of the things you do every day that use computers? Can you do them without computers? Show the video via youtube. Direct their attention back to the questions. Allow students time to jot down any notes for reflection or further investigation. Facilitate a think, pair, share. Then, have the students locate the Scratch website (scratch.mit.edu) and click on the "create" button. Explain the different components of the scratch editor (stage, sprites, palette, block shapes, and the menu). Allow students to ask questions for clarification before they are able to explore these components.
Explain (Brooke and Russ Co-teach) Introduce the various blocks in the scratch program. Show students the properties of each block. Then, allow students to build a simple program. They will start by gliding the sprite across the screen with motion blocks. Have the sprite bounce if it hits and edge. Then program it to make a sound when it hits the edge. Create a second sprite and have it say something when it touches the main sprite. Allow students time to manipulate and navigate this simple program.
Elaborate (Brooke and Russ Coteach) Now that students are familiar with the basic functions and components of Scratch, they will be able to draw using the pen and numeric values in scratch. After experimenting with the drawing functions, they will be directed on how to create a Spirograph using mathematical equations to create a design. They will be able to manipulate values and variables using prime numbers. They will be able to share effective values with the group.
Evaluation Visit the parking lot/evidence of learning spot (On Lino-virtual sticky note) Ask students to share responses to the following prompts: What did you learn today? What can we do with this new knowledge? What do we want to learn next? Any other questions?

Vocabulary or topics discussed

Science = manipulating variables.

Technology = Scratch, spirograph

Engineering = Spirograph, wheels, rotational motion, gears interlocking and moving together.

Mathematics = Discussion of coordinate planes, ranges, X, Y coordinates , prime numbers

Which STEM careers were highlighted? = Computer Science and Engineering

Day 2

Lesson Title: **Interactive Sorting Networks and Finite Machines**

Lesson Objectives: Students will create a computer based simulation on Scratch to further explore how computers interpret and sort input and data.

Anticipated Learner Outcomes: Students will explain how computers can complete parallel processing with conditional statement commands

KCCR Standards Math: CCSS 8.F.B.4 Use functions to model relationships between quantities. , CCSS 7.EE.B.4 Solve real life and mathematical problems using numerical and algebraic expressions and equations.

KCCR Standards Science: NGSS Grades 9-12 Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Materials: colored electrical tape, cards or dice, open space (tile floor or smooth carpet) , computers with internet access, Scratch (scratch.mit.edu), Lino-Virtual sticky note

Learning activities

Engage (describe how teacher will capture students' interest): Ask students " How do you sort items, objects, numbers? Do you remember sorting buttons in Kindergarten?" Choose 6 students and hand them a number card. Ask these students to discuss a way to sort their cards. Ask for class suggestions. Present the term parallel sorting or processing

Explore (describe hands-on/minds-on activity students will be doing): The 6 sorter network will be previously created before students arrive to sorting area. Explain the sorting network (directions available mathmaniacs.org) Have the students who had the cards following the sorting rules as explained.

Explain (list questions & strategies that allow students to communicate what they learned): Students will utilize If, Then, Else blocks on scratch to code a simulation with 2 or more variables. (Discuss with Russ for more details on this component). Students will also continue to add insights into Canvas? or Lino.

Elaborate (describe how students expand on concepts & make connections): Students will sense that it takes a lot of computers and programming to process and sort data. Ask students how that could be possible. Elicit conversation, then support thinking about a super computer. Tour BeoCat , a super computer located in the Computer Engineering Department. Show them the many components and the research tracking capabilities of this computer.

Evaluation (describe how students will demonstrate their learning): Students will do a stand up/hand up, pair up (Kagan Structure) to discuss what they learned for the day. We will visit the Parking Lot/schema-misconception chart.

Vocabulary or topics discussed

Science = variables, independent, dependent, constant, trajectory, acceleration

Technology = simulation, parallel sorting, finite machine

Engineering = coding, infinite machines, defining a situation that determines need of sorting.

Mathematics = x,y variables, functions, data and quantity comparison.

Which STEM careers were highlighted? = Computer Programming and Coding (Women in Computer Engineering)

Day 3

Lesson Title: **Data Encoding and Message Decoding-Binary Codes and Communication**
Lesson Objectives: Students will utilize binary code within a Scratch simulation to communicate with NASA
Anticipated Learner Outcomes: Students will explain how computers are coded with binary codes and patterns and these binary code patterns can be translated into meaningful data.
KCCR Standards Math: CCSS 8.F.B.4 Use functions to model relationships between quantities. , CCSS 7.EE.B.4 Solve real life and mathematical problems using numerical and algebraic expressions and equations. KCCR Standards Science: NGSS Grades 9-12 Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.
Materials: Find Someone Who handouts, dot binary cards , computers with internet access, Scratch (scratch.mit.edu), Lino-Virtual sticky note or Canvas

Learning activities
Engage (describe how teacher will capture students' interest): Tell students that engineers collaborate and share ideas. Pass out Find Someone Who… sheets. Go over each square in the sheet. Model how to Find Someone Who with students understanding they are to share and receive information from peers. After time is up (12-15 min) they will be asked to share their own knowledge or that learned from someone else highlighting each square. It is anticipated students will be stuck on the binary code square. Show the students teacher ages in binary.
Explore (describe hands-on/minds-on activity st-udents will be doing): Pass out binary cards. Each card has powers of 2 represented in dots (1,2,4,8,16,32,64). Tell students that binary code has 0 and 1. Tell students to place their dot cards in order with 64 being the farthest left and 1 on the right. Write these digits on the board. Tell them that patterns with 0 and 1 can be used to communicate data. For example1=1, 10=2, 11=3,100=4. Model how you flip the dot cards over and if they are flipped you use a 1, if not, 0. Continue working through base ten numbers with binary.
Explain (list questions & strategies that allow students to communicate what they learned) Challenge students to figure the binary code for 63. Hopefully they realize they need all numbers below 64 so it would be 111111. How is binary different from base 10? Are there any similarities? Can computers use binary code to interpret and communicate with words?

Elaborate (describe how students expand on concepts & make connections): This part is co=taught with Russ. Students will sense that codes can be combined into infinite possibilities. Binary codes create numbers and numbers can be combined to represent words –hexadecimal and ASCII table http://www.asciitable.com/ Students will be directed to a link that has a simulation of a Mars Rover and Camera. Students will run the program to have the camera communicate direction and . The output will just be numbers indicating a direction related to degrees on a coordinate grid. Students will use their knowledge of binary codes and ASCII symbols to rewrite the code (adding 8 to make positive values) and joining inputs to create a message. The final message will be MOVE~EEST~MARK.
Evaluation (describe how students will demonstrate their learning): Students will be left wondering if they were Mark Watney, would they move West or East. Ask students what we do a when we don't receive a clear message. We would ask for a repeat. Also, we would expect reliability as Mark is navigating Mars with only a camera, camera angles, and input from NASA. If students run the program again, it should read MOVE~WEST~MARK. Students should make the connection that computers can miscommunicate or have glitches (autocorrect).

Vocabulary or topics discussed

Science = variables, independent, dependent

Technology = encoding, decoding, programming

Engineering = coding, infinite machines, defining a solution for effective communication using code

Mathematics = binary codes, hexadecimal, powers of 2, powers of 10,

Which STEM careers were highlighted? = Computer Programming and Coding

Day 4

Lesson Title: Artificial Intelligences and Rover Simulations
Lesson Objectives: Students will compare and contrast the abilities of Artificial Intelligence and Human Sources and Receivers. Students will create a simulation of a Mars Rover within Scratch.
Anticipated Learner Outcomes: Students will code an AI on a drafted PacMan stage. Students will experiment with their created scripts to determine success. Scripts contain conditionals

and variables. Students will create bits of code manipulating data and variables to successfully navigate a Mars roving mission in a simulation

KCCR Standards Math: CCSS 8.F.B.4 Use functions to model relationships between quantities. , CCSS 7.EE.B.4 Solve real life and mathematical problems using numerical and algebraic expressions and equations.

KCCR Standards Science: NGSS Grades 9-12 Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Materials: Pictures of "artificial" objects –wig, turf, Cheetos, food coloring. Kahoot quiz, Turing Test diagram/picture, Chinese room example, picture clips with corresponding cell table on computer, paper signs with numbers 1-5, pencils, paper, computers, Scratch (scratch.mit.edu)

Learning activities

Engage (describe how teacher will capture students' interest): Students participate in Kahoot quiz with questions based on content learned from the last three days. Then, show students the pictures (of artificial items) Ask them to talk to the people around them and share what all the pictures have in common. Then ask them what are things that cannot be artificial (water, humans, landforms) Guide students to understand that things that are not artificial can change with the environment and are ever changing. Ask students what artificial means and follow up with what it means to be intelligent. Tell then that coders can create AI's. What questions would you ask to determine if who/what you are dealing with is an AI? (look for questions that elicit emotion, consistent answers the entire time, many possible outcomes). Introduce them to the "turing test" and Chinese Room experiment (on saved presentation).

Explore (describe hands-on/minds-on activity students will be doing): Tell students they are going to take a 'turing test' in which they will process input as an AI would. Pass out picture grid cards. Tell students the cards contain cut up pictures of a cat or dog. Tell them they will look at their clip, determine if it is a cat or dog. Then, tell them to close their eyes and raise their hand if it is a cat or dog. Write down how many of each got cat dog, then reveal the entire

image. Continue this process for all of the images. One image is of a bear. Why did you only answer cat or dog? (they were "coded" at the beginning to only choose cat or dog). Why did you believe this image could be a cat or dog (fur, beady eyes, nose, etc). Tell them because an AI is coded to determine only certain outcomes, it did not pass the turing test. After looking at the data on the board from cat/dog responses, coordinate responses on a corresponding grid. Mark grids for correct responses with Y and N for incorrect. Ask students to determine any patterns in the grid. Hopefully they will see there are more Y's in the middle of the grid (where the animal's most determining features would be) and N's in the borders (vague details). How did this simulation resemble a turing test?
Explain (list questions & strategies that allow students to communicate what they learned): Can AI's actually 'learn'? What are some reasons AI's make errors? How do AI's affect exploration and travel? Do you think AI's will ever be able to understand emotion?
Elaborate (describe how students expand on concepts & make connections): Students will then be introduced to the Mars Roving Simulation. Students will be given the object to get their rover to the 'target' . Their rover will sense a crater and communicate that it 'can't go there'. It has been coded to not move and respond if it senses red (color of craters). Students will need to manipulate the code with operations blocks, controls –if, then else- and move blocks to move their rover to the target. Once they reach their target, challenge students to change the values of the move blocks and rotate blocks for less moves and to conserve power. Pay attention to any students that mention artificial intelligence or coding it to do so.
Evaluation (describe how students will demonstrate their learning): Students will do an inside/outside circle to discuss the following: What are three careers you think AI's will be doing more than humans in the next 50 years? What are three careers/jobs you think humans will still do more than computers? These questions will also be posted on Canvas.

Vocabulary or topics discussed

Science = attributes, characteristics, artificial and natural

Technology = Artificial Intelligence, simulation

Engineering = perceive, motion, act, sensing, variable blocks

Mathematics = Coordinates, outliers, data, probability, angles, greater/less than

Which STEM careers were highlighted? = Coding artificial intelligences, turing test creators, robotics, game coding