Mission to Mars: A Computer Science Curriculum for Middle School STEM Camps

by

Russell A. Feldhausen

B.S., Kansas State University, 2008

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Daniel A. Andresen

Copyright

Abstract

This thesis presents a curriculum designed for 5th and 6th grade students attending a summer camp for science, technology, engineering, and mathematics (STEM) disciplines. The curriculum uses several concepts from educational theory and computer science education research. It also uses techniques such as cognitive apprenticeship, expansive framing, and scaffolded lessons to increase student learning outcomes. It was taught during two cohorts of a STEM summer camp.

The curriculum is analyzed through self-efficacy surveys both before and after the class, measuring how students judged their own capability to use skills learned during the class. Analysis of the data shows that the increase in student self-efficacy has a medium to large effect size overall, as well as student self-efficacy with many computational thinking skills. Data from various population groups based on gender, previous STEM experience, and socio-economic status indicators is also analyzed. Finally, many areas of future work and improvement are presented and discussed.

The outcome of this work is to demonstrate the effectiveness of the curriculum presented in increasing student self-efficacy with computational thinking skills, specifically by showing the links between content in the curriculum and specific computational thinking skills.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I wish to thank Dr. Andresen first and foremost for all of his help and advice throughout my years as his student. He still stands as the only one to ever give me a terrible grade, but it was the grade that I fully deserved at that time. I hope I've improved since then.

Thank you also to Dr. Hsu and Dr. DeLoach for always offering advice and support whenever needed, Nathan Bean, Dr. Valenzuela and Dr. Weese for their advice and expertise in the realm of computer science education, Brooke Snyder and Dr. Valenzuela for their help teaching the course and refining its design, and all current and former faculty at K-State who put forth time and effort to teach me. In addition, thanks are due to Dr. Soh and Dr. Cooper from UNL, who provided some initial direction and contacts for my literature review, and to Dr. Dringenberg for her valuable feedback on a draft of this work.

Finally, I wish to thank my family for their never ending love and support as I've continued on my quest through life, all of my students who have had to put up with my terrible jokes and quirky activities, and anyone else who has helped me in her or his own unique way toward this goal. Thank you!

Dedication

For my wife, Miriam. Without her unending love and support, I would not be able to do the work that I do. I love you!

Chapter 1

Introduction

Today's modern world increasingly relies on technology, and young students today must be equipped with the skills needed to thrive in such a society. Computational thinking is one important aspect of understanding how these various technological devices function, in essence a "fundamental skill for everyone"[1] in the 21st century. We must find a way to integrate teaching and learning that skill into educational curricula to develop the next generation of science, technology, engineering, and mathematics (STEM) graduates.

Research published in the journal *Science* has shown that introducing students to STEM careers during middle school has a large effect on their choice of future careers.[2] To that end, this thesis presents and analyzes a curriculum developed for a summer camp for middle school students promoting STEM careers. The overall goal of this curriculum, named *Mission to Mars*, is to introduce students to computational thinking skills through the field of computer science, and encourage them to consider computer science it as a future STEM career path. This can be done by increasing their knowledge, experience, and self-efficacy within the field.[3]

To accomplish that goal, *Mission to Mars* models modern educational theories such as cognitive apprenticeship to promote knowledge transfer,[4] and focuses on improving student self-efficacy in computational thinking through hands-on activities.[5] The curriculum has also been annotated to describe the levels of student achievement with computational thinking

skills desired using the revised Bloom's Taxonomy.[6;7]

Using data collected from two cohorts of the summer camp, this work seeks to answer the following research questions:

RQ1) What is the impact of this curriculum on student self-efficacy with computational thinking skills?

RQ2) What is the relationship between the computational thinking skills covered in the curriculum and the student self-efficacy with those skills?

RQ3) What is the relationship between student factors such as gender, previous STEM experience, or socio-economic status indicators and the observed student self-efficacy with computational thinking skills?

Before discussing the curriculum, a review of current research literature related to this topic is presented in Chapter 2. Following that, the curriculum design is discussed in Chapter 3. Chapter 4 discusses the structure and format of the camp and how the class is taught, and Chapter 5 discusses the design of the assessment tools and techniques used. The results of the surveys and other assessment data are presented in Chapter 6 along with a discussion of those results in Chapter 7. Chapter 8 lists possible future work identified to improve this project, with Chapter 9 providing a conclusion.

The curriculum and results described in this work are part of a larger project discussed in several other publications.[5;8–10]

Chapter 2

Literature Review

This work builds upon existing knowledge from many different areas. This chapter covers relevant research literature in the areas of computational thinking, educational theory, computer science curricula, and student assessment.

2.1 Computational Thinking

In 2006, Wing wrote "Computational Thinking," a seminal Viewpoint article in the *Communications of the ACM* magazine, advocating for computational thinking as a "fundamental skill for everyone" [1] and starting a vast discussion of what that term actually entails and how to accomplish that goal.[‡] She later refined her own definition in 2011, stating "computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." [12] She also drew a very clear distinction between learning computer programming and computational thinking, underscoring that learning to program isn't enough to ensure students are learning computational thinking skills. [1;12] In both articles, she strongly argued that computational thinking is a necessary skill set for anyone

[‡]As of this writing, Google Scholar lists over 2800 citations of this article, and over 400 citations are listed in the ACM Digital Library. Computational thinking was also previously discussed by Papert, but in a much more limited scope as it related to enhancing mathematics education with computers, and is therefore not relevant to this discussion. [11]

to learn, and it should be taught throughout our educational system.[1;12] This clearly informs the need for the creation and wide use of computational thinking curricula, which is a primary goal of this work.

Lee et al. furthered this call by discussing how computational thinking could be integrated into primary and secondary (kindergarten through 12th grade, commonly referred to as K-12) education more directly.[13] They chose to focus on three major areas of computational thinking: "abstraction, automation, and analysis,"[13] showing how each of those concepts could be applied in many different learning scenarios, such as modeling and simulation, robotics, and game design.[13] Unfortunately, they also noted that there are many obstacles to directly including computational thinking in K-12 curricula due to, in their words, "existing curriculum standards, lack of opportunities for teachers to learn [computational thinking] as part of their professional development, and lack of access to necessary infrastructure."[13] Instead, they remarked that computational thinking is much more prevalent in activities outside of the normal classroom, such after-school clubs, weekend activities, and STEM camps such as the one this curriculum was created for.[13]

Recently, Yadav et al. discussed how to integrate computational thinking into K-12 education.[14–16] In those articles, the authors made the case for including computational thinking and related pedagogical strategies in K-12 teacher training.[15;16] This can be done by teaching non-computer science teachers how to use tools and techniques to engage students in computational thinking within a different subject, such as the use of modeling and simulation tools in a science classroom.[15] In another instance, they discussed how vital it is to introduce teachers to computational thinking concepts throughout their education, perhaps in an educational technology course already required in their curriculum.[16] To further their argument, they conducted an experiment with two groups of pre-service teachers, one receiving instruction in computational thinking, and another control group.[14] The results showed that pre-service teachers who were exposed to about two hours of training in computational thinking were much more likely to define computational thinking as more than just including technology in the classroom, and slightly more likely to agree that it should be included in the curriculum.[14] This is similar to the program described by Bean et al.[17] when

training pre-service teachers to use computational thinking in the classroom at Kansas State University. Through the partnership with local teachers and pre-service teachers offered in the STEM camp environment, this project allows those teachers to become more familiar with computational thinking and encourages them to bring that knowledge back to their classrooms.

Grover and Pea provided a "state of the field" review of computational thinking in K-12 education in 2014.[18] They discussed many of the problems inherent to teaching computational thinking, namely that it has been very difficult to find an agreeable definition of what computational thinking is and how best to integrate it into K-12 curricula.[18] Much of the work they presented revolves around the creation of tools and environments that aid in learning computational thinking or computer programming, but most haven't achieved broad curricular integration.[18] However, they did provide a large list of areas ripe for future research and improvement, such as applying modern educational theories, including situated cognition and cognitive apprenticeship, to computational thinking curricula; using computational thinking within other subjects; and dealing with student perceptions of computational thinking and computer science in general.[18] Much of the design of the *Mission to Mars* curriculum was informed by these ideas.

Mannila et al. provided a much deeper look at computational thinking in primary and middle school education (kindergarten through 9th grade, or K-9) that same year.[19] They discussed how K-9 education is different than other areas, namely that it is much more generalized and teachers tend to teach many subjects at once in an integrated fashion, instead of being subject-matter experts.[19] The authors then provided a review of several different countries' previous attempts at including computational thinking in K-9 curricula, informal initiatives to introduce computational thinking outside the classroom, and broader initiatives aimed at increasing computational thinking in national curriculum standards.[19] They also discussed results from an international survey of teachers, collecting information about how they integrated computational thinking into their subject areas.[19] Finally, they also gave several explicit examples of computational thinking successfully implemented into educational contexts.[19] Many of those examples provided inspiration for some of the activities

included in the *Mission to Mars* curriculum. In addition, the curriculum's use of expansive framing and links to concepts in a variety of subject areas further builds on this concept.

Many previous research publications from Kansas State University also dealt with computational thinking in a K-12 setting. Bean et al. discussed a program to introduce pre-service teachers to computational thinking concepts using Scratch,[20] along with a survey to assess student and teacher learning.[17] It also included a prototype of the shapes activity discussed in section 3.1.[17] Weese et al.[5] discussed how a previous version of the curriculum discussed in this thesis impacted student self-efficacy in computational thinking and computer science at the 2016 *STEM Summer Institute*.[5] Weese and Feldhausen, and Feldhausen et al., presented further results from the 2016 and 2017 cohorts at the *STEM Summer Institute* using the current version of the curriculum.[8;10] The assessment methods used will be discussed in detail in section 2.4 and Chapter 5. Finally, Weese further discussed many of these outreach programs and assessment methods in his doctoral dissertation.[9]. While these works primarily focused on assessing student gains in computational thinking ability, this work will cover the design aspects of the *Mission to Mars* curriculum. It will, however, rely on many of the same assessment methods and data to support the effectiveness of the curriculum's design.

Brennan and Resnick provided a concrete definition for computational thinking (CT) in terms of computational concepts, practices, and perspectives through their study of the Scratch[20] programming environment.[21] Similarly, Google also uses a list of concepts and skills to further define computational thinking.[22]. Weese and Feldhausen created a list of computational thinking skills and related CS principles based on those works, adapted below in Table 2.1, which informed the curriculum design and assessments in this work.[8]

2.2 Educational Theory

The design of the *Mission to Mars* curriculum makes use of several concepts from the field of educational theory. This includes expansive framing to aid with knowledge transfer to other domains, the cognitive apprenticeship model for teaching how a master thinks when performing a task, and the revised Bloom's taxonomy, which is used to describe the expected

Table 2.1: *CT Concepts and Related Computer Science Principles*[8]

Abbr.	Description
ALG	**Algorithmic thinking** - sequence of steps that complete a task. Operators and expressions are also included
ABS	**Abstraction** - generalized representation of a complex problem, ignoring extraneous information
DEC	**Problem decomposition** - breaking a problem into smaller, more manageable parts that can be solved independently of each other
DAT	**Data** - collection, representation, and analysis of data[22]
PAR	**Parllelization** - simultaneous processing of a task[22]
CON	**Control flow** - directs an algorithm's steps when to complete
IAI	**Incremental and iterative** - building small parts of the program at each step instead of the whole program at once
TAD	**Testing and debugging** - performing intermediate testing and fixing problems while developing
QUE	**Questioning** - working to understand each part of the code instead of using code that is not understood well
USE	**Reuse and remix** - making use of other people's work and resources to solve a problem

learning outcomes from each lesson.

2.2.1 Framing and Knowledge Transfer

Vygotsky introduced the concept of the *zone of proximal development*, which he defined as the "distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers." [23] Likewise, Piaget also wrote extensively about the cognitive development of children.[24] In his work, he discussed a learning paradigm called *constructivism*, where students must "construct" new knowledge from what is being taught, while challenging their own view of the world and understanding of how it works.[25]

In 1980, Papert published *Mindstorms: Children, Computers and Powerful Ideas*, which discussed the creation and use of the Logo programming language and computer technology

to teach students.[26] In that seminal book, he provided a strong argument for his view of what he termed *constructionism* in education.[26] Similar to Piaget, Papert argued that students must build their own understanding, but they can do so through the "construction" of artifacts such as computer programs, while working directly with the tools and ideas they are exploring.[25] He compared it to learning a language by spending time among native speakers of the language instead of just learning it from a textbook.[26] This supports the use of hands-on learning of computational thinking skills

Brown et al. continued this work by discussing what they called *situated cognition*.[27] This theory posits that the material being taught, the process by which it is taught, and the larger context in which it is taught, all play a role in learning.[27] This is in contrast to the notion that knowledge gained is fundamentally independent from the context and situation wherein it is learned, thereby assuming that the problem of knowledge transfer from school environments to work environments is trivial.[27] In their work, they discuss the differences between how "just plain folks" (JPFs), students, and practitioners learn a skill. Their comparison shows that JPFs and practitioners both learn skills within the context those skills are to be employed, leading to much more effective learning.[27] They argue that learning is best accomplished through authentic activity, using the "tools of the trade," while immersed in the culture of the subject area.[27]. The *Mission to Mars* curriculum is designed to provide that authentic feel by using precise computer science terms and concepts wherever possible, while providing background information to students about the larger context in which those activities are placed.

This is further supported by Pea, who suggested that educators could improve knowledge transfer "by making everyday situations and school situations part of the same classification scheme for problem types, making explicit the links the student is now expected to draw spontaneously."[28] Furthermore, he suggests that schools should spend time bridging the gap between in-school and out-of-school experiences and situations.[28] He would later assist Grover et al.[29–31] in the creation of their computer science curriculum discussed in section 2.3.

Engle et al. discussed how the *framing* of a learning experience affects how well that

knowledge transfers to other situations.[32;33] Framing refers to the larger societal context in which the learning experience is situated.[33] This should not be confused with scaffolding, as described below in relation to cognitive apprenticeship, but is similar to the concepts related to the contexts surrounding a learning environment as discussed by Brown et al.[27] and Pea.[28] above. From Engle et al., "[a] teacher can frame a lesson as a one-time event of learning something that students are unlikely to ever use again,"[33] called *bounded framing*, "or as an initial discussion of an issue that students will be actively engaging with throughout their lives,"[33] called *expansive framing*. Their initial research showed that expansive framing resulted in students who were able to more easily transfer knowledge between related subject areas when compared to bounded framing.[32] They theorize that students exposed to expansive framing are more likely to make better use of their time and abilities to learn the material and generalize it for future use because expansive framing implies that they may need this information in the future.[32] Whenever possible, the *Mission to Mars* curriculum encourages instructors to provide expansive framing and discussion to assist students in learning the material within a larger context, while employing an encompassing scenario to link the individual lessons together.[†]

Tai et al. presented research showing that students expecting to have a science-related career in 8th grade were up to 3.4 times as likely to graduate with a physical science or engineering degree as their peers.[2] This implies that introducing students to computer science in middle school can have a very strong effect on their future career choices.[2] This is supported by independent articles by Greening,[35] Carter,[36] and Grover et al.,[3] who each focused on student perceptions of computer science prior to college. They collectively showed that many of the issues preventing students from choosing computer science as a possible college major or career choice involved incorrect perceptions about the field and the types of people who succeed in it.[3;35;36] The *Mission to Mars* curriculum helps present computer science in a positive light while hopefully mitigating previous poor perceptions students may have of the

[†]Interestingly, the authors reference a unique study by Hart and Albarracín, which demonstrated that simply using the imperfective aspect instead of the perfective aspect when describing a previously undertaken action—i.e "I was doing something" instead of "I did something," made participants more willing to resume a task or continue a previously learned behavior, indicating that simply how we discuss learning experiences after the fact could also have an impact in retention and transfer.[34]

field, encouraging students to consider it as a possible future career path.

2.2.2 Cognitive Apprenticeship

Brown et al.[27] (including Collins as a co-author), and other works by Collins et al.[4;37] introduce the concept of *cognitive apprenticeship* as an educational paradigm. Based on the traditional apprenticeship method of learning a task or trade, cognitive apprenticeship directs students to first observe a master of a cognitive task at work, with the master clearly describing the internal thought processes and methods being employed.[4] The student is then slowly given larger and larger tasks to complete under the watchful eye of the master, who provides brief correction and instruction as needed while completing the more difficult parts for the student, a process called *scaffolding*.[4] Over time, the student assumes more responsibility, with the master providing projects of appropriate skill level until the student is able to complete entire projects independently.[4] The overall structure of cognitive apprenticeship is divided into four areas: the content to be taught, the methods by which it is taught, the sequence in which skills are learned, and the social aspects,[4] linking back to Brown et al.'s situated cognition.[27] Furthermore, Collins also discussed how the growth of computer technology could help cognitive apprenticeship become more approachable and cost-effective, making this paradigm very useful in modern education.[37]

That discussion was continued by Ghefaili, who discussed how cognitive apprenticeship could effectively be used as an instructional technique.[38] He took the approach of clearly describing the theories behind cognitive apprenticeship, how it differs from traditional notions of apprenticeship, and, more specifically, the teaching methods that can be used to achieve it.[38] In addition, he also discusses how educational technology can make cognitive apprenticeship a reality in modern classrooms, by bringing real-world problems to students, allowing students to interact with experts, and making the thought processes and techniques used to solve the problems more visible.[38] Much of the design of the *Mission to Mars* curriculum uses concepts and methods from cognitive apprenticeship discussed by Ghefaili. Specifically, when leading students through a guided activity, the instructors are encouraged

to clearly talk through any thought processes they use to complete the task, helping the students understand not only what is being done, but why it is done that way.

2.2.3 Bloom's Taxonomy

Bloom et al. created a taxonomy for describing expected student learning outcomes, now known as Bloom's taxononmy, which was later revised by Anderson et al. and Krathwohl.[6;7] The revised Bloom's taxonomy uses two dimensions to describe the expected learning outcome using both a noun, to describe the type of knowledge, and a verb, to describe the cognitive process level at which students should be competent with that knowledge.[7] A summary of these two dimensions is presented in Table 2.2, adapted from Krathwohl.[7] This taxonomy is used in Chapter 3 to annotate the expected learning outcomes in the *Mission to Mars* curriculum.

2.3 Related Computer Science Curricula

There have been many efforts to include computer science and computational thinking in K-12 education. Some efforts focus on entire curricula or standards to be adopted broadly at all levels, while others represent single classes or interventions designed for a specific audience, venue, or technology. For this work, the most relevant literature focuses on small interventions and single activities, as discussed below.

2.3.1 Activities and Lesson Plans

There have been many efforts to create online resources for K-12 computer science lesson plans. One of the most notable examples is CS Unplugged, a project from the University of Canterbury, New Zealand, co-sponsored by Google.[39] CS Unplugged provides "a collection of free learning activities that teach Computer Science through engaging games and puzzles that use cards, string, crayons, and lots of running around."[39] The overall goal of the project is to demonstrate that learning computer science and computational thinking does not require a

Table 2.2: *Revised Bloom's Taxonomy Dimensions*[7]

Knowledge Dimension	
Factual Knowledge	The basic elements that students must know to be acquainted with a discipline or solve problems in it
Conceptual Knowledge	The interrelationships among the basic elements within a larger structure that enable them to function together
Procedural Knowledge	How to do something; methods of inquiry, and criteria for using skills, algorithms, techniques, and methods
Metacognitive Knowledge	Knowledge of cognition in general as well as awareness and knowledge of one's own cognition
Cognitive Process Dimension	
Remember	Retrieving relevant knowledge from long-term memory
Understand	Determining the meaning of instructional message, including oral, written, and graphic communication
Apply	Carrying out or using a procedure in a given situation
Analyze	Breaking material into its constituent parts and detecting how the parts relate to one another and to an overall structure or purpose
Evaluate	Making judgements based on criteria and standards
Create	Putting elements together to form a novel, coherent whole or make an original product

computer, and to allow students to experience the field without first learning programming.[39] Several of the activities in the *Mission to Mars* curriculum are adapted from or inspired by CS Unplugged activities.

Google also provides an online repository of computer science lesson plans on their Exploring Computational Thinking page.[40] In addition, they provide a guide showing how their lessons align with various international standards, including the CSTA K-12 Computer Science Standards in the United States and the Computing at School standards from the United Kingdom.[41–44] The activities in this resource are not used directly within the *Mission to Mars* curriculum, but they were reviewed at the time of writing and may have brought forward ideas that are present in the curriculum.

Code.org, along with Computer Science Education Week, hosts the Hour of Code each year to encourage students to learn at least one hour of programming.[45;46] The Code.org website currently lists over 170 activities for a variety of grade levels and programming environments.[45] Some related activities developed prior to the *Mission to Mars* curriculum were used for Hour of Code events held at Kansas State University, and were subsequently adapted to fit the curriculum.

Many individual lesson plans published online are built around the use of a specific programming language such as Alice,[47] or a specific technology such as robots[48;49] or other devices.[50] While there are many to choose from, of particular interest to this project is research using the Scratch programming environment from MIT.[20] Meerbaum-Salant et al. discussed how early research working with Scratch showed it to be a viable platform for teaching students computer science concepts.[51;52] However, they also cited that learning to program in Scratch lead to the development of some habits outside accepted good programming practices, and they were concerned that this could cause issues as students progress to other languages.[52] Specifically, they note that because of the low barrier to entry with Scratch, many students only learn the basics of creating scripts, and don't progress toward creating more advanced and complex scripts; instead, they engage in what the authors call "extremely fine-grained programming,"[52] whereby they create a large number of very simple scripts to accomplish a task instead of a single, more complex script.[52]

More recently, Moreno-León and Robles performed a literature review of research related to the use of Scratch to teach programming.[53] In their analysis, they narrowed the field down to seven papers of interest, and concluded that there is a "very promising outlook" for using Scratch to integrate computer science and programming into the K-12 curriculum, but that much of the research is not as rigorous as it should be, with small sample sizes and a lack of quantitative data.[53] Still, the Scratch programming environment is a very powerful tool, and was chosen for the *Mission to Mars* curriculum due to its flexibility and ease of use. In addition, many students were already familiar with the tool from other STEM outreach activities, making it a good choice for use with this project.

2.3.2 Educational Theory and Computer Science Curricula

There are also some examples of educational theory, specifically cognitive apprenticeship and scaffolding as discussed in section 2.2, being used to create lesson plans and curricula. Shabo et al. describe an early attempt at bringing cognitive apprenticeship to computer science by creating an online repository of notes and resources for various problems in order to make visible the thought processes and techniques used to solve them.[54]

Larkins et al. describe a project applying the cognitive apprenticeship framework to a middle school robotics camp.[55] They specifically note how they designed activities around each teaching method described in the cognitive apprenticeship principles from Collins et al.[4;55] In their results, they note that they did not achieve significant gains in student perceptions of STEM fields, mainly due to the already high interest of participants before the camp, but they did observe an improvement in the variable isolation reasoning skills of the students.[55] Many of the activities in the *Mission to Mars* curriculum use the same teaching methods from cognitive apprenticeship as applied in this project.

Webb and Rosson used scaffolding extensively, along with the Scratch environment, to teach computational thinking to middle school girls.[56] Their activities were centered around programs developed in Scratch and designed to be usable by the participants, but with a flaw that could be investigated and corrected, encouraging exploration and reflection.[56] Based on their qualitative review of student work and reactions, they concluded that "scaffolded examples in Scratch are an effective way to convey [computational thinking] concepts and skills in a short amount of time."[56] Many of the activities in *Mission to Mars* take advantage of scaffolding in a similar way to tackle more difficult problems than could normally be addressed during a short learning session.

Repenning et al. provided one of the largest and most comprehensive projects for bringing computational thinking skills to K-12 classrooms.[57;58] Their Scalable Game Design intervention uses a project-first approach to scaffolding, where students immediately start working on a project, with concepts and skills taught along the way.[58] They were able to conclude that, by properly scaffolding the activities, they can take advantage of Vygotsky's zone of

proximal development to push students beyond what they could accomplish on their own.[58] The authors also note that over 80% of teachers who used their program chose to continue it beyond the first year, demonstrating its effectiveness in the eyes of K-12 teachers.[58]

Grover et al. presented another project for bringing computational thinking to middle school students.[29–31] The Foundations for Advancing Computational Thinking (FACT) intervention was created using a design-based research approach to review and enhance the curriculum across several iterations.[29–31] In designing the curriculum, they focused on using both scaffolding and cognitive apprenticeship techniques, combined with expansive framing to encourage transfer to other areas.[29–31] In addition, the course was first taught as a combination of in-person lessons and accompanying videos, then converted to a completely online version.[29–31] Their work successfully showed that students could transfer some computational thinking skills to other areas, and also that the students participating in the online course performed at least as well as students in the earlier course with in-person lessons.[29;30] Much of the design and desired outcomes of the *Mission to Mars* curriculum align with this work.

2.4 Student Assessment

Another major aspect of designing a curriculum is determining how to assess student learning outcomes and the quality of the curriculum itself. Most techniques for assessment of computer science and computational thinking fit into three broad categories: knowledge assessment, code analysis, and self-efficacy assessment. Chapter 5 discusses in detail why self-efficacy assessment was chosen for this project, with information on possible future work to include knowledge assessment and code analysis discussed in Chapter 8

2.4.1 Knowledge Assessment

Knowledge assessments include typical classroom activities such as exams, quizzes, and homework to assess a student's knowledge of the subject. Bransford et al. provided a strong case for assessment in curriculum design.[59] The authors argued that schools should

be centered around four areas: the learner, knowledge, assessment, and community.[59] As part of an assessment-centered classroom, they stated that teachers should "continually attempt to learn about their students' thinking and understanding."[59] However, they also discussed that many common views of assessment are troublesome, such as the focus on rote memorization of facts and procedures, and tricky quiz questions instead of assessing for deeper understanding of the subject.[59]

Whalley et al.[60] and Thompson et al.[61] described two different projects to create computer science assessments based on commonly used educational models, such as the revised Bloom's taxonomy[6;7] and the SOLO taxonomy.[62] In both projects, they used the frameworks and their associated categories to devise questions for exams that are designed to measure aspects of student learning.[60;61] While they noted that both projects yielded useful exam questions, they also both discussed the difficulty in agreeing on what level of learning should be expected from students while consistently underestimating the difficulty of the assessment.[60;61] The *Mission to Mars* curriculum makes use of these taxonomies to describe the level of student learning desired at various points in the curriculum, but does not use them directly for assessment. Other works describing knowledge-based assessment of computational thinking include Bienkowski et al.[63] and Grover.[64]

2.4.2 Code Analysis

Code analysis refers to the process of discovering information about a student's learning outcomes by analyzing projects created by the student, either at the time of submission or during the creation of the project. Werner et al. described a process they've used to analyze games developed in the Alice programming language.[65–67] While successful, they also note that this process was very time consuming.[66;67] Meerbaum-Salant et al. also performed detailed analysis of student work in Scratch.[52] Their analysis was performed manually, and was primarily driven by their own observations of student work while working on a different research topic.[52] Franklin et al. presented an effort to automate analysis of Scratch programs.[68] They collected student projects from a summer camp very similar to the *STEM*

Summer Institute and analyzed them based on desired computer science concept outcomes using a custom developed tool.[68] Moreno-León et al. developed a more advanced code analysis tool named Dr. Scratch to measure how well students implemented various computational thinking concepts in their Scratch projects through static code analysis.[69;70]

Fields et al. took a slightly different approach to code analysis, using techniques from big data to sift through code snapshots collected over time.[71] They collected JSON representations of student Scratch projects as they were being edited, with a snapshot created each time a student switches context in the program, or at least every two minutes.[71] They then analyzed the number of instances of different programming aspects across the time students spent editing the project.[71] By correlating the data with their own qualitative analysis of student progress at various checkpoints throughout the course, they were able to get a deeper view into student learning and thought processes that would not be visible by simply analyzing the code after the fact.[71]

2.4.3 Self-efficacy Assessment

Bandura provided another insight into how to measure student success, or in this case, student confidence.[72] *Self-efficacy* is described by Bandura as "how people judge their capabilities"[72] and the effect that judgement has on their own "motivation and behavior."[72] He argued that students who have a high level of self-efficacy with a task will tend to work harder and persist in the face of obstacles, whereas someone with a lower sense of self-efficacy might give up.[72] Conversely, someone with high self-efficacy might also find simple tasks boring and put forth less effort than someone with low self-efficacy.[72] He also noted that self-efficacy seems to play a role in career choice, citing the difference between males and females and their own self-efficacy regarding different possible careers, when each group has similar measured capability to perform the job.[72]

While measuring self-efficacy is as simple as asking a student to rate herself or himself, Bandura did describe some possible concerns and his own attempts to remedy them.[72] In one instance, he addressed the notion that by asking someone to rate his or her self-efficacy,

it created an expectation that the person must live up to that rating.[72] In his explanation of related research, he noted that this is indeed a myth, and no relation has been found between subjects who were asked to rate themselves before a task and those who were not, or those whose ratings were made publicly or privately.[72]

Ramalingam et al. applied the concept of self-efficacy to computer science education.[73] In their work, they studied the self-efficacy of students before and after a programming class, and compared that to the measured mental models, or subject knowledge, of the participants.[73] They found that not only do most students experience significant gains in self-efficacy, but student self-efficacy itself was a significant predictor of student performance, along with the student's own mental model.[73] In essence, while knowledge in the subject is important, student self-efficacy is also an essential part in determining a student's success in a class.[73]

Lishinski et al. furthered this work by studying how self-efficacy relates to course performance over time, as well as many other "self-regulated learning" aspects.[74] They confirmed that self-efficacy is the strongest predictor of student performance in classes when compared to other predictors such as student goals and student metacognitive strategies.[74] However, they noted that their research shows that self-efficacy alone isn't enough, because, in their words, "metacognitive strategies and goal orientation impact self-efficacy, which impacts performance, and then performance impacts self-efficacy, which then impacts performance again."[74] Thankfully, they described self-efficacy as "very malleable" compared to other aspects of student learning, so there is still value in designing learning experiences around improving student self-efficacy.[74]

Much of the previous work at Kansas State University related to this project also used self-efficacy as a primary means of assessing student learning.[5;8–10;17] Those assessments were also used during this project.

Chapter 3

Curriculum Design

The *Mission to Mars* curriculum was designed for the *STEM Summer Institute*, hosted by the Manhattan-Ogden USD 383 school district.[75] It was first designed during the 2015 summer camp, and was subsequently refined before the 2016 session. The curriculum was again used in 2017, with only minor changes from the previous year.

The primary goal of the curriculum is to introduce 5th and 6th grade students to computational thinking skills through computer science, increasing their knowledge and self-efficacy within the field. By doing so, the desired outcome is an increase in the number of students who consider STEM majors in college and related careers.

The theme was chosen to relate many of the activities to the field of space exploration, one that has proven popular with many students at that level. Additionally, inspiration for the revised version was taken from the popular book *The Martian* by Andy Weir and its film adaptation.[76;77] The use of an overarching theme for the lessons helps provide expansive framing for these activities within the broader world.

Many of the activities and lessons included in the curriculum were adapted from a variety of sources, including previous outreach activities, courses at Kansas State University, CS Unplugged,[39] and other online repositories. The sources are cited below wherever possible.

In this chapter, the activities and learning objectives for each day of the four-day curriculum are outlined, as well as relevant notes regarding the design aspects of each part. The

complete curriculum and notes can be found in Appendix A.

3.1 Day 1

The first day begins with an icebreaker activity. During the activity, students introduce themselves and are encouraged to ask the instructors any questions they may have, no matter how strange or difficult. This allows the students and instructors to get to know each other, and also begins an open and friendly dialog between students and instructors. By showing that any questions will be answered honestly and truthfully, it establishes a classroom norm of open communication, a very important aspect of constructive student learning and cognitive apprenticeship.

Following the icebreaker, students complete a brief survey to measure previous computer science and STEM experiences, as well as their current self-efficacy with computer programming and computational thinking. The assessments used are discussed in Chapter 5.

Before starting any formal activities, the students are shown two videos as part of the expansive framing of the curriculum. The first video is a promotional video from Code.org,[78] describing the importance of learning computer programming by featuring various famous figures and celebrities discussing their experiences. The second is a short clip from the film adaptation of *The Martian*,[77] explaining how the main character found himself stranded on Mars and setting the stage for several of the activities in the course.

To begin the lessons, students are introduced to the Scratch programming environment.[20] While many students have previously worked with programming tools such as Scratch, they may not all have a similar experience level. Therefore, it is important to ensure each student has the proper foundation. The lesson starts by introducing the vocabulary of terms used to refer to various parts of the Scratch interface, and then briefly demonstrates how to manipulate blocks and sprites to create very simple programs. The programs typically involve moving sprites within the Scratch stage, causing them to respond to each other through the use of special blocks to detect collisions. However, the amount of instruction is purposely kept to a minimum, and students are encouraged to experiment with the many

blocks available in Scratch during a large block of unstructured time. This follows the constructivist ideal of allowing students to build their own world-view and understanding of the field through hands-on experience. Students are encouraged to constantly test their program after adding new blocks, introducing the skills of testing and debugging when a block does not function properly, as well as iterative and incremental development, both core skills in computational thinking.

Once the students have some basic experience with Scratch, the lesson continues with a couple of activities to demonstrate more advanced capabilities of the tool. The first activity involves drawing regular geometric polygons using a moving sprite in Scratch. It is adapted from a set of Scratch activities published online by Bean.[79] The activity starts by providing a scaffolded example demonstrating how to draw three lines connected by obtuse angles on the screen. A screenshot of the example is shown in Figure 3.1. The students must modify the example to draw an equilateral triangle, easily done by simply modifying the angles. However, most students immediately try to set the angles to 60 degrees, correctly recalling the measurement of internal angles of such a shape, and are surprised when that doesn't produce the desired result. In fact, the angles in the code represent the external angles of the shape, and not the internal angles as expected. Through trial and error, most students can complete the activity given enough time. Students who complete the activity quickly are encouraged to help their neighbors if they are stuck, effectively turning them into teachers and creating a more collaborative learning environment.

As the activity continues, students are directed to modify their program to draw regular polygons with progressively more and more sides. Typically, the first modification is difficult for students to grasp, but once they see that it involves simply adding more lines and modifying the angles, they quickly catch on to the process. However, soon they realize that the program becomes bigger and more complex as they modify it. Students are then introduced to the concept that "programmers are lazy" and good programmers will find ways to simplify their code. Following the model of cognitive apprenticeship, emphasis is placed on discussing the thought processes a programmer would use to analyze this problem. After observing that many blocks of code are repeated, students learn about iteration to

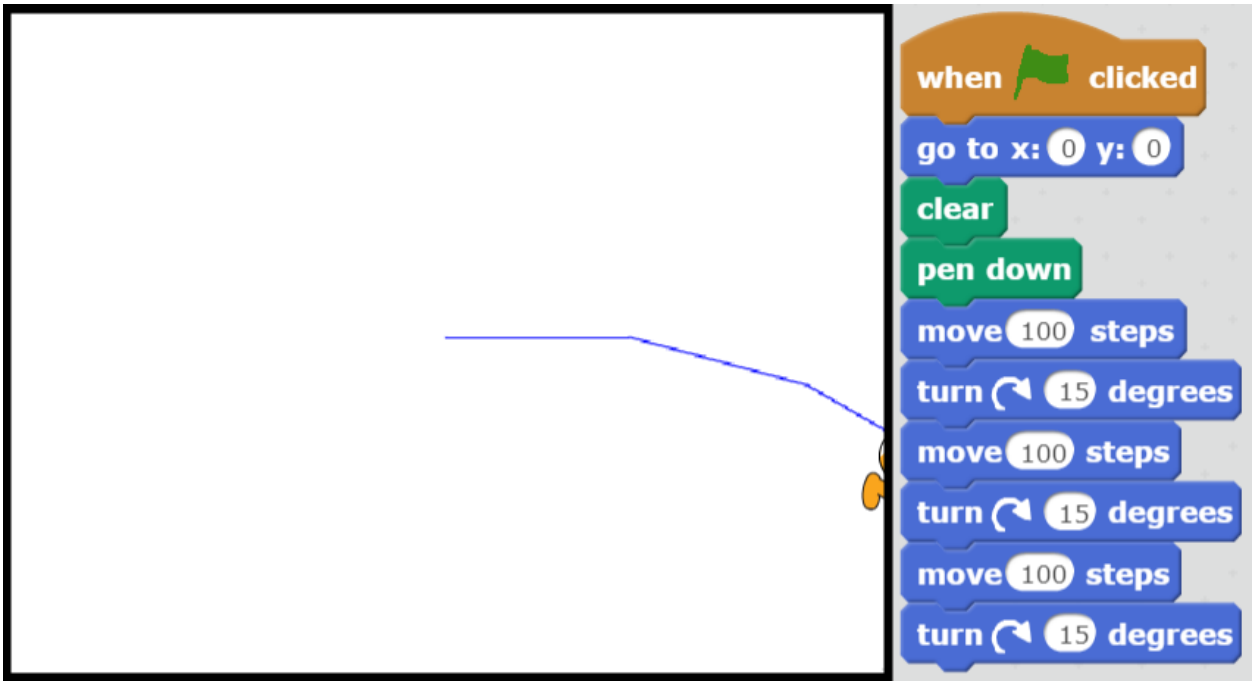

Figure 3.1: *Scratch Shapes Activity Example*

simplify the code significantly. Students quickly realize that, with iteration, adding more sides involves simply changing the number of iterations.

Similarly, once students create a shape with many sides, they realize that the angles may no longer be whole integers. Instead of manually calculating those values, students learn how to use the math operators in Scratch to calculate the size of each angle, using a formula from geometry that many of them are already aware of. Finally, once students see that their code only involves the number of sides in two places, they are shown how to replace those two values with a variable, just as a programmer would in a real-world program. By using a variable, students only have to modify one number to direct the program to draw a shape with any number of sides. Finally, the program can be modified to accept user input to set the value of the variable, creating a fully developed and usable application. At each step, students are encouraged to modify their program, test it, and debug any problems, further emphasizing both the testing and debugging skill and the iterative and incremental development skill.

Additionally, with one small modification to make the length of sides inversely propor-

tional to the number of sides, students can create a program to draw figures with tens or hundreds of sides while shrinking them to a size that is still visible on the screen. Students quickly notice that, as the number of sides increases, the shape looks more and more like a circle. As part of the goal of expansive framing, this observation is explicitly linked to how circles are represented in computer games, animation, and graphics programs. Additionally, these concepts can be linked to the fundamental laws of calculus, showing that a smooth shape can be approximated by a shape made of infinitely many infinitely small sides.

The final activity of the day is a Scratch program simulating a Spirograph drawing toy. It is meant to demonstrate another usage of the drawing capabilities of Scratch, while giving the students something fun and engaging to interact with at the end of the day. The students can modify the values of variables controlling the simulation, and are challenged to experiment and find values that create interesting designs. Some students struggle initially due to the fact that they are likely to use composite numbers, being more familiar with them after learning multiplication tables. However, using prime numbers, or combinations of coprime numbers, yields the best results. Again, the connections between computer science, aesthetics, and mathematics are explicitly described as part of the expansive framing of the lesson.

At the end of the day's activities, the students are asked to respond to some simple reflection questions to help build a lasting memory of the day by calling attention to the aspects that were most important in the lesson. In addition, the instructors can quickly gauge what students felt was most important as well as any areas they wanted to focus on in the future. This stems from the work of Bransford et al.,[59] who encourage teachers to continually assess student thinking and learning as they teach.

The major topics covered in day 1, annotated with the related computational thinking skills from Table 2.1, are listed below. They are also placed within the revised Bloom's taxonomy in Table 3.1.

1) Understand how programming is used in the broader world (QUE)
2) Understand what several of the Scratch blocks do in a script (QUE)
3) Create working scripts in Scratch (CON)

4) Test programs after adding each block and debug as needed (TAD)
5) Build programs in small chunks (IAI)
6) Write a computer program based on a given specification (DEC)
7) Apply the formula for external angles of a regular polygon from geometry (ALG)
8) Understand that "programmers are lazy" and look for ways to deconstruct a problem into repeatable steps (DEC)
9) Understand the use of iteration in computer programs (CON)
10) Understand the use of variables in computer programs (ABS)
11) Learn how circles are represented and displayed in a computer (DAT)
12) Learn how a Spirograph can be simulated in Scratch (ABS)

Table 3.1: *Revised Bloom's Taxonomy Analysis of Day 1 Outcomes*

	Remember	**Understand**	**Apply**	**Analyze**	**Evaluate**	**Create**
Factual	11	2	7			
Conceptual		1	6			
Procedural	12	9,10	3			
Metacognitive		8		4,5		

3.2 Day 2

To begin the second day of activities, the students cover some simple questions about the Scratch interface and what some of the blocks can be used for. This is designed to refresh the students' mindset and get them ready to work with Scratch again. They are also able to ask any questions about the activities from the previous day before moving on.

The first lesson of the day involves sorting. Sorting was chosen as a representative algorithm for computer science since it is easily understood by students of all ages. In addition, there are many different forms of sorting algorithms that are well known and studied in computer science, and they can easily be discussed by anyone familiar with programming.

Finally, sorting lends itself to several unplugged-style activities much better than many other algorithms. Clearly, a large part of this day's activities are devoted to the algorithmic thinking skill in computational thinking.

The discussion begins by asking students why sorting is important in the real world. This includes descriptions of how words are ordered in the dictionary, how phone numbers are listed in the phonebook, and many others. In addition, students are asked to think about which attributes of real-world objects can be used for sorting, such as by height, weight, color, name, age, etc. In doing so, students are encouraged to start thinking of the world in terms of data to be manipulated, a key part of the abstraction skill in computational thinking.

The next activity involves the use of a sorting network, as shown in Figure 3.2. This activity was inspired by a similar activity from CS Unplugged.[39] The instructors create a sorting network on the floor of the classroom using colored tape, and describe to students how it works. At each intersection where 2 students meet, the one with the higher value being sorted goes right, and the student with the lower value goes left. At the end, the students should be in sorted order, no matter where they started in the beginning. It is recommended to avoid duplicate sorting values at the start, so items such as age are not optimal; instead values that are easily discernable such as height or the number of letters on a nametag work well for this activity. As students learn how the process works, groups are encouraged to see if they can perform the task faster and with fewer errors that the previous group. Once students are comfortable with the activity, they are asked to describe how this activity relates to computer programming. Instructors can use open-ended questions to help lead the discussion, such as: "is the sorting network a computer program? Why or why not?"

Following that activity, the day continues with a discussion of algorithms from the perspective of a computer scientist. Much of this activity is adapted from slides used in the *Introduction to Computing Science* class at Kansas State University, taught by the author of this work.[80] First, the instructors perform an in-class demonstration of how difficult it can be to create an algorithm that works. To do this, two students are asked to demonstrate

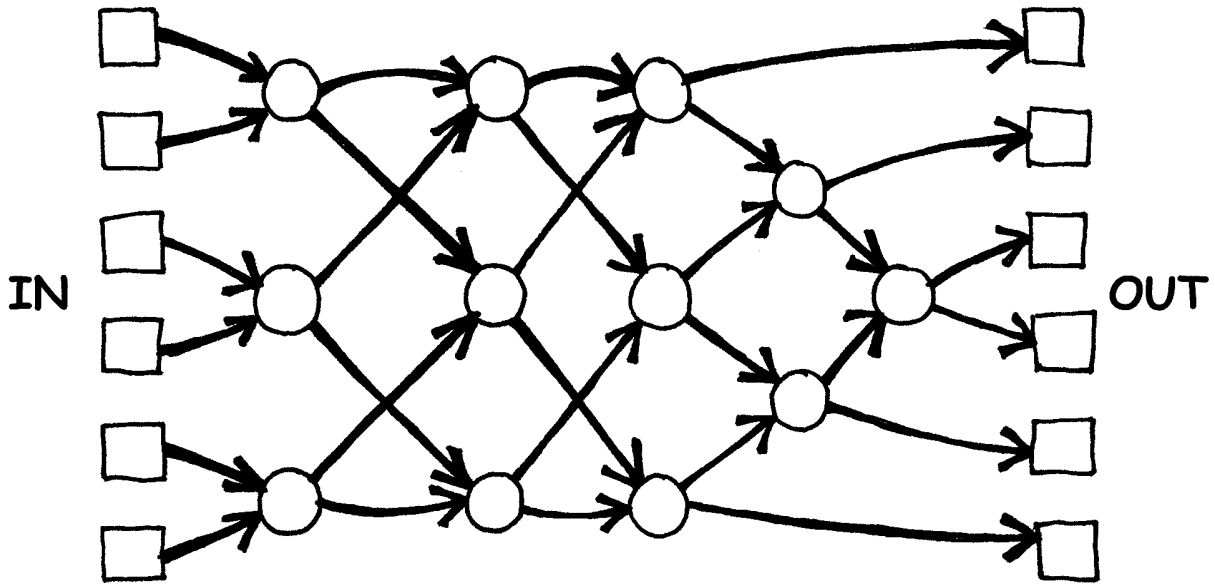

Figure 3.2: *Sorting Network*[39]

how to shuffle a deck of cards. One student tries to give detailed instructions, while the other student, aided by the instructor, performs those steps explicitly while outside of the view of the first student. This is very similar to a well-known activity of creating a peanut butter and jelly sandwich. Usually students find it very difficult to describe the exact steps to shuffle a deck of cards. However, at the end the instructor can demonstrate a very simple method to accomplish the task to show it can be done: place the cards into two separate piles, then repeatedly take one card from each pile and place it on top of a third pile. While it isn't a method someone would normally use to shuffle cards because it is slow, it still works. In doing so, students are encouraged to "think outside the box" and carefully consider their own preconceptions of how to perform a task. This leads to a discussion about why this method might be ideal for computers to implement and why it is counter-intuitive for a person to do in reality.

The activity then shifts to a more formal discussion of algorithms, including the origin of the name and a simple formal algorithm, such as Euclid's algorithm for finding greatest common divisor of two numbers. This helps support the notion that an algorithm is a simple list of steps that anyone can follow to accomplish a task.

To give students hands-on experience with algorithms that computers perform, each student is given a suit of 13 cards from a standard deck of playing cards which they can use to practice computer sorting algorithms. Students are first directed to shuffle and sort the cards however they naturally would, then describe the steps they followed to get the cards into sorted order. Typically most students perform a variant of insertion sort or selection sort, but a few students have demonstrated a bucket sort algorithm or even something resembling quicksort. These activities are designed to give students further experience both with abstraction and data manipulation in the realm of computational thinking.

The first formal sorting algorithm demonstrated is insertion sort. The students hold the suit of cards in their hand, then place the first card face up on the table. Then, they take the second card, hold it before the first card on the table, and ask "does this card go here?" If so, they place it on the table there, if not, they place it behind the last card. They repeat that process for each card, starting with the first opening and asking themselves if it fits there, then moving to the next one until the correct place is found. While doing so, students are directed to keep track of the number of times they ask that question as a simple way of tracking the number of steps they perform. Once they are done, they shout out that number while it is recorded on the board.

The same process is done for bubble sort. Student start with all cards in a line face-up on the table in front of them. Then, they look at the first two cards and ask themselves "do I need to swap these?" If so, they swap those cards, then look at the second and third cards and ask the same question. They repeat this process, starting over at the beginning each time they reach the end, until the cards are sorted. Again, they are asked to keep track of the number of times they swap two cards, and then shout out the result at the end so the numbers can be recorded on the board.

After these two steps, there are two sets of numbers on the board. The instructors help the students determine the average number of steps for each, and use that to determine which algorithm used fewer steps. Typically they are similar, but each time the result is a bit different. From there, the students are asked to then discuss which algorithm would be faster on a computer, based on their observations. Eventually, the instructors lead the discussion

toward the need for some way to evaluate these algorithms numerically, introducing simple algorithm analysis and the concept of worst-case inputs. This is done to help increase the link between mathematics and computer science.

Once they are comfortable with sorting algorithms, students are then led through an activity to create a program that uses the bubble sort algorithm in Scratch. First, the instructor describes how to work with lists in Scratch, introducing a new set of blocks. Then, they are shown how to create a list that has a few unsorted numbers. It is important to carefully show them how to set up the list to be the same each time instead of filling it with random numbers each time, so that it is easier to debug errors later on. Once they have a list, students then build the bubble sort algorithm from the inside out, first creating an "if" block to check if the first two consecutive numbers should be swapped, then filing in the block with the standard three-step swap procedure, introducing the concept of a temporary variable. Then, they can use variables to generalize the "if" block to work with any two consecutive numbers, while wrapping it in a "repeat" block and a "forever" block to loop through the list multiple times until it is sorted. This process of building a program from the inside-out uses the skills of problem decomposition, iterative and incremental development, and testing and debugging from computational thinking. A version of this program is shown in Figure 3.3.

The last activity of the day links back to the space exploration theme and uses an activity inspired by the events in the book *The Martian*. In this instance, the main character must figure out how to grow enough food to survive on Mars for over a year. He has a few real potatoes available, but must somehow obtain the other requirements for plant growth, namely water. The activity starts with a brief discussion of how to get water on Mars using the available oxygen in his living quarters and the hydrogen in the rocket fuel. This leads to a short explanation of the chemical reactions at work, how much of each ingredient is needed, and what the byproducts are. It also links back to programming by discussing how to simulate these reactions in a computer program. The instructors can also generalize this discussion to include the use of computer programs to simulate many real-world interactions, another key concept in abstraction.

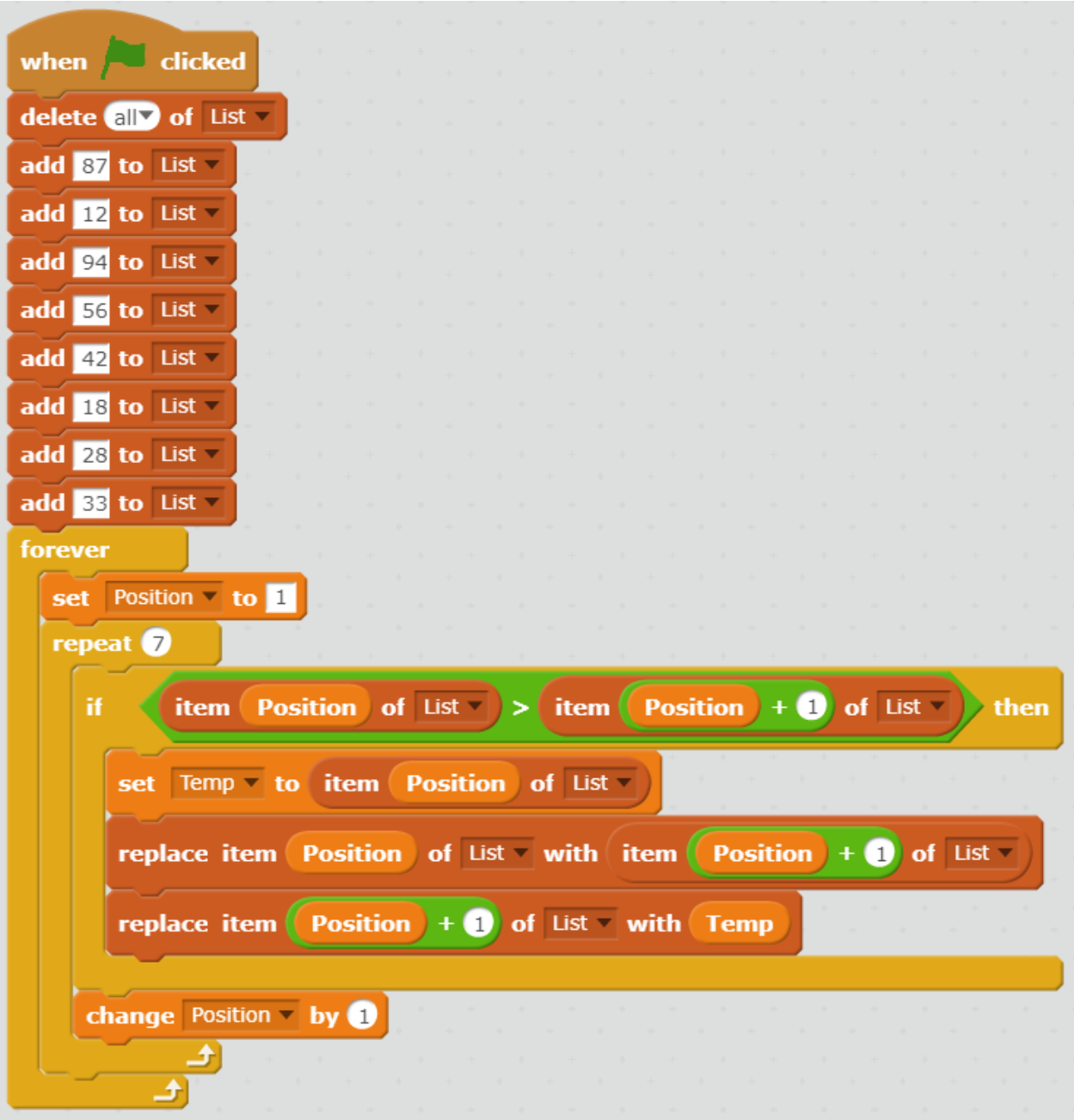

Figure 3.3: *Bubble Sort in Scratch*

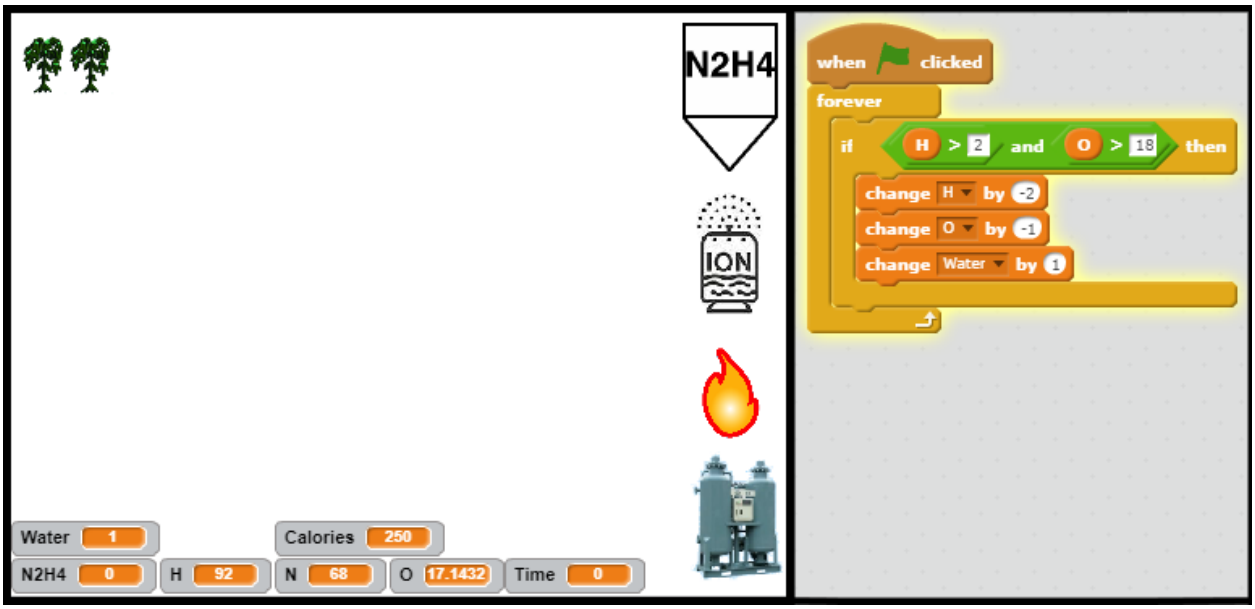

Figure 3.4: *Growing Potatoes Simulation in Scratch*

The students are then given a scaffolded simulation program in Scratch which they must complete. It includes sprites for the potatoes to be grown, hydrazine (rocket fuel), oxygen reclaimer, ionizer, and a flame. In addition, there are variables for the amount of hydrazine, hydrogen, oxygen, nitrogen, and water in the atmosphere of the simulation. On each sprite, students must add a few blocks to produce the proper ingredient or use available ingredients in a reaction. In doing so, they affect the atmosphere in some way. Their goal is to produce water quick enough to grow food, while maintaining safe levels of hydrogen in the atmosphere. If too much hydrogen is released, it will cause the simulation to stop and explode. A screenshot of this simulation, including the code for the flame sprite which creates water from hydrogen and oxygen in the air, is shown in Figure 3.4.

Finally, there is some time set aside for asking the students reflection questions at the end of the day, similar to the end of the first day.

The major topics covered in day 2, annotated with the related computational thinking skills from Table 2.1, are listed below. They are also placed within the revised Bloom's taxonomy in Table 3.2.

1) Understand the importance of working with data (ALG)

2) Use a sorting network to sort data (ALG, ABS)
3) Understand that an algorithm is a series of steps (ALG, DEC)
4) Use insertion sort to sort playing cards (ALG)
5) Use bubble sort to sort playing cards (ALG)
6) Compare bubble sort and insertion sort (ALG, QUE)
7) Write a program to sort numbers using bubble sort (ALG, IAI, TAD)
8) Learn how water can be produced through chemical reactions (ABS)
9) Create a working simulation of water synthesis (ABS, TAD, IAI, PAR)
10) See that computers and humans may use different algorithms for similar tasks (ALG, QUE)

Table 3.2: *Revised Bloom's Taxonomy Analysis of Day 2 Outcomes*

	Remember	**Understand**	**Apply**	**Analyze**	**Evaluate**	**Create**
Factual	8					
Conceptual		3	9	6		
Procedural		1	2,4,5,7			
Metacognitive	10					

3.3 Day 3

The third day of activities begins by quickly reviewing what students have done so far to get them back in the mindset of working with computer science. They are also asked for any questions or unclear ideas covered so far, so those can be resolved before moving forward.

The first part of this day deals with numbers and how computers store and represent data, another major concept in computational thinking. It starts with an activity from CS Unplugged[39] that introduces binary numbers using cards and worksheets from the Computer Science and Engineering for K-12 website.[81] Each card has a number of dots on it representing a power of two. Each student has a set of cards for the first eight powers of two. The students

are then shown how to construct binary numbers using the cards. For example, to represent the decimal number 11 in binary, the cards for eight, two and one are turned face up, while the others are face down. Then, they write out the number by using a 0 for a face down card, and a 1 for a face up card. Therefore, decimal 11 becomes 00001011 in 8-bit binary. This exercise is repeated for several numbers, while students convert from binary to decimal and from decimal to binary. These exercises can also be used to teach simple addition in binary, showing that it follows the same process and rules that they've already learned for decimal addition.

Once students are familiar with binary numbers, they are then introduced to the hexadecimal numbering system. Since hexadecimal is simply an adaptation of binary to a larger set of symbols, it has a direct one-to-one translation from binary. Students are given a worksheet that helps them make the transition from decimal to binary and hexadecimal. Once they have completed that worksheet they can use it as a resource for later activities.

The discussion then moves to how computers could represent other data types. Students are asked to describe the types of files they store on their computer, and then discuss how the computer might store those files using binary. For example, the American Standard Code for Information Interchange (ASCII) can be used for storing text as numbers, which can be expanded to include items such as documents and spreadsheets which could be stored as text represented as numbers using ASCII. Similarly, images can be stored as text in the form of vector images, or as an array of pixels that use numbers to record the intensity of each of the three primary colors as an RGB bitmap image. This can be further developed by explaining that videos are just images that change over time, following the same concept as bitmap images. This discussion is supplemented by slides from the *Introduction to Computing Science* class.[80] This is typically a very fun and engaging discussion for students, and many times students are observed after class talking with their peers about binary numbers and ASCII codes. This activity further enforces the abstraction and data skills in computational thinking.

The next activity once again pulls from the theme and a situation found in the accompanying book. In this instance, the main character on Mars is trying to establish a

communication channel with Earth. While the folks on Earth can see what is happening on Mars using a camera, the main character on Mars has no way of getting information from Earth. The only thing that his companions on Earth can do is rotate the main camera they are using to view Mars. This leads the main character to place sixteen signs around the camera in a circle, marking them with the sixteen symbols used in hexadecimal. He remembers that two hexadecimal symbols can represent a single character in ASCII, so if the camera points at the symbols he can record them and convert that information to text.

The students are provided with another scaffolded activity in Scratch that simulates this situation. When students first run the simulation, they see a camera randomly rotate back and forth, with no direct indication of what it means. A secondary backdrop is provided which has the hexadecimal symbols marked on a circle to help start the decoding process. From there, students follow along and answer guided questions as the instructor describes how to find the angle the camera is facing and how to convert that angle to a value corresponding to a hexadecimal symbol. Once they have a value, students must store it and wait for a second value, since two hexadecimal values are required to decode one ASCII character. With both values, the students use a chain of "if" blocks to determine which character is represented. Unfortunately, in Scratch there is no direct way to convert from a value to a character as in most programming languages, so a large chain of "if" blocks is the only way to go. This requires students to be very careful when arranging blocks, and helps to highlight the importance of understanding how code blocks fit together and how many ways it can produce an incorrect result if a minor change is made. Because of this limitation, it further emphasizes the importance of testing and debugging as well as incremental and iterative development to avoid problems when working on a large program. An example of this project is shown in Figure 3.5.

In addition, once students have completed the project, they discover that the initial message received, "go eest mark," is very confusing. This leads to another discussion about data and how sometimes it can be misinterpreted. The instructor can show how this can be mitigated, and link it back to situations in real world, such as asking someone to repeat what they said or using a phonetic alphabet. By pressing a second button on the simulation,

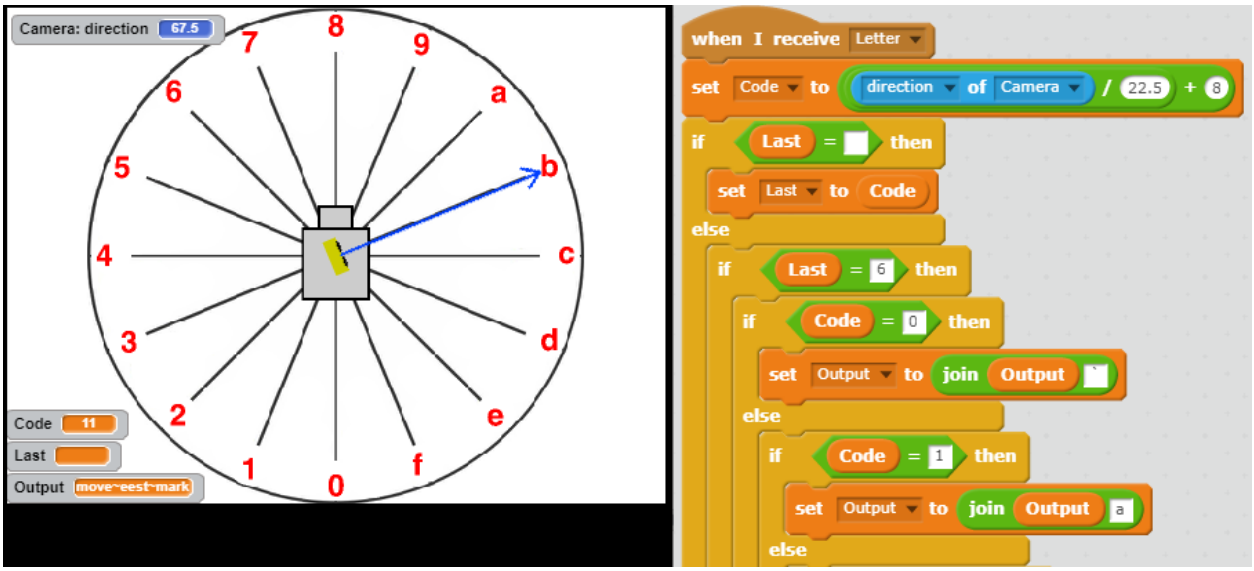

Figure 3.5: *Hexadecimal to ASCII in Scratch*

it will replay the message with a corrected output, so the students can see how important it is. This further builds on the skill of working with data in computational thinking, as well as the questioning skill related to how the program functions and still returns an invalid result.

As with the other days, some time is set aside at the end to ask the students some questions to reflect on what they've learned.

The major topics covered in day 3, annotated with the related computational thinking skills from Table 2.1, are listed below. They are also placed within the revised Bloom's taxonomy in Table 3.3.

1) Understand the binary and hexadecimal numbering systems (ABS, DAT)
2) Convert numbers to and from binary, decimal, and hexadecimal (ALG, DAT)
3) Observe simple addition in binary (ALG)
4) Understand how other data types are stored on a computer in binary (ABS, DAT)
5) Convert an angular measure to a hexadecimal value in Scratch (ALG, ABS, DAT)
6) Store a value for use later in a program (DAT)
7) Use a large set of conditional statements to decode a value (ABS, DAT)
8) Develop a large computer program without errors (TAD, IAI)
9) Understand how data can be misinterpreted or incorrect (ABS, DAT)

Table 3.3: *Revised Bloom's Taxonomy Analysis of Day 1 Outcomes*

	Remember	Understand	Apply	Analyze	Evaluate	Create
Factual		4	5			
Conceptual		1,9				
Procedural		3	2,6,7			
Metacognitive			8			

3.4 Day 4

The fourth day uses the field of artificial intelligence to continue teaching students about computer science. As with the previous days, it begins with a brief discussion of topics covered earlier in the week to make sure students are in the correct mindset as well as time to answer any questions they may have before starting on new material. In addition, in 2016 students were asked to play a brief online quiz game consisting of several simple review questions, in order to determine how well each student was retaining the knowledge gained from previous activities. The questions used and responses obtained from students are discussed in Chapter 6. This may also form the basis of a future extension to this work that includes knowledge-based assessments, which will be discussed in Chapter 8.

Following that, slides from the *Introduction to Computing Science* class[80] are used to introduce the topic of artificial intelligence. The slides include a few videos on the topic, and briefly discusses the background and history of artificial intelligence. This culminates in a lively discussion of the classic Turing test and its counterpoint, the Chinese room experiment. This gives the instructor a chance to ask students their own thoughts on what would make a computer seem intelligent, and how to determine if a computer is truly acting with a human level of intelligence. Typically, through this discussion most students find it very difficult to determine exactly what intelligence is and how to determine if a computer is intelligent, mirroring many experts' views on the subject. This process continues to build upon the abstraction, problem decomposition, and questioning skills in computational thinking.

Next, students are introduced to the concept of artificial neural networks as one method for building an artificially intelligent computer. They also are shown the early belief that a computer structured like the human brain, with neurons and synapses, should be able to think like a human brain to support this concept. The discussion also describes how a neural network is trained, and how that mirrors learning in the brain.

The first activity of the day is an unplugged activity originally designed by the author for the *Introduction to Computing Science* class,[80] simulating how to train a neural network to identify if a picture contains a cat or a dog. Each student is given a small section of a larger image, and then asked to look only at their section and vote if the whole image contains a cat or a dog. Some students will have sections from the center of the image, making it simple. Other students will have sections near the edge, containing very little if any useful information. As students vote, the results are counted and presented to the class. Students who vote correctly are given more heavily weighted votes for the next round, typically by grouping them in the classroom based on the number of correct votes. In addition, one of the images is revealed to be a bear, prompting a discussion about what would happen if the training data is incorrect or misleading.

After a few rounds, the students should be split into several groups. Students then reveal which section of the pictures each student has, observing that students with sections near the center of the picture typically have guessed correctly more times than students with sections near the edge. This helps demonstrate that a neural network can be trained to be correct more often by changing how much weight each neuron is given.

Following the activity, students are presnted information about a famous incident where neural networks failed, as well as shown a video of a neural network learning how to play a video game, which gives a very clear view of how a neural network can be built over time using evolutionary programming. While it is a bit advanced for most middle-school students, it clearly links the field of artificial intelligence to video games, helping to connect it with an area of interest to many students. The overall goal of these hands-on activities is to provide additional framing for the field of computer science within the larger scope of the world.

The next activity involves creating artificially intelligent agents for a video game. Stu-

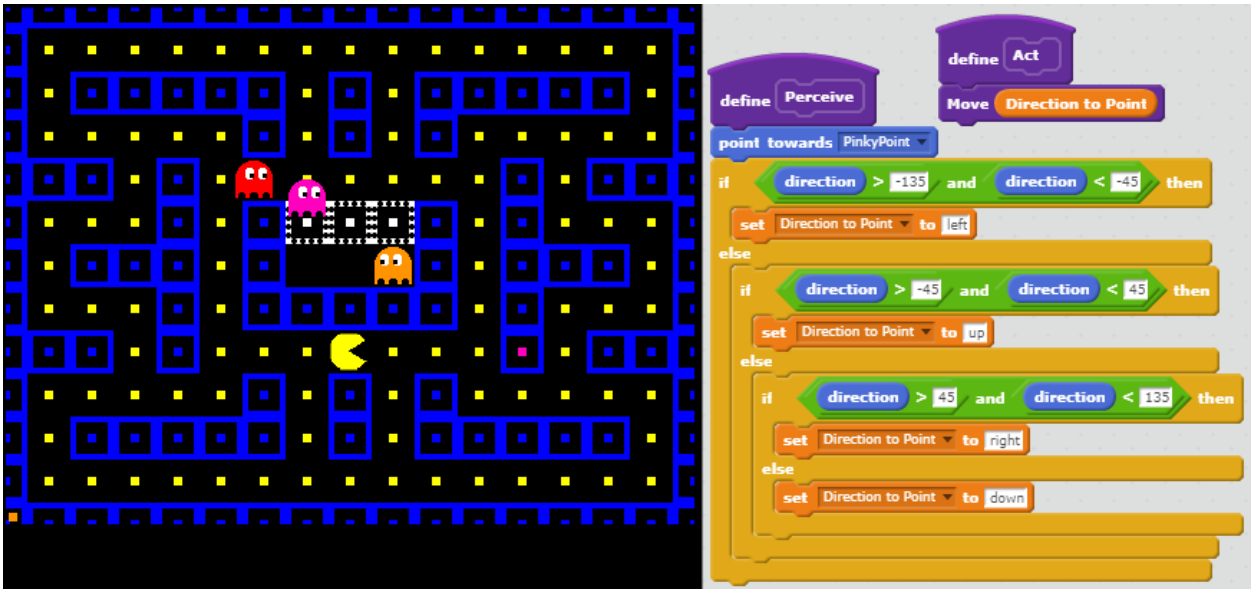

Figure 3.6: *Pac-Man in Scratch*

denst are provided with a scaffolded Scratch program that resembles the classic Pac-Man arcade game.[82] In the original game, each of the four ghost enemies had a unique artificial intelligence algorithm controlling its movement.[83] For this activity, the instructor leads the students through creating three of them using a two-phase intelligent agent framework, consisting of *perceive* and *act* phases. In the perceive phase, each agent will determine the location or locations it intends to move toward. In the act phase, it will choose a single option and move in that direction. In doing so, students are once again shown how they can use problem decomposition to make large programs simpler to build, as well as reiterating many skills they've covered earlier in the week. A screenshot of this activity is shown in Figure 3.6.

The final activity uses one more situation from the source material for the theme. In this instance, the main character must traverse the surface of Mars to get to a new area. While doing so, he must avoid several roadblocks along the way. For this activity, students start with a scaffolded Scratch program that simulates a Martian surface with three randomly placed obstacles between the vehicle and the goal. Students use what they've learned to build an artificially intelligent agent to avoid the obstacles, following the same two-phase model as in the previous activity. Once they have successfully created a program to reach

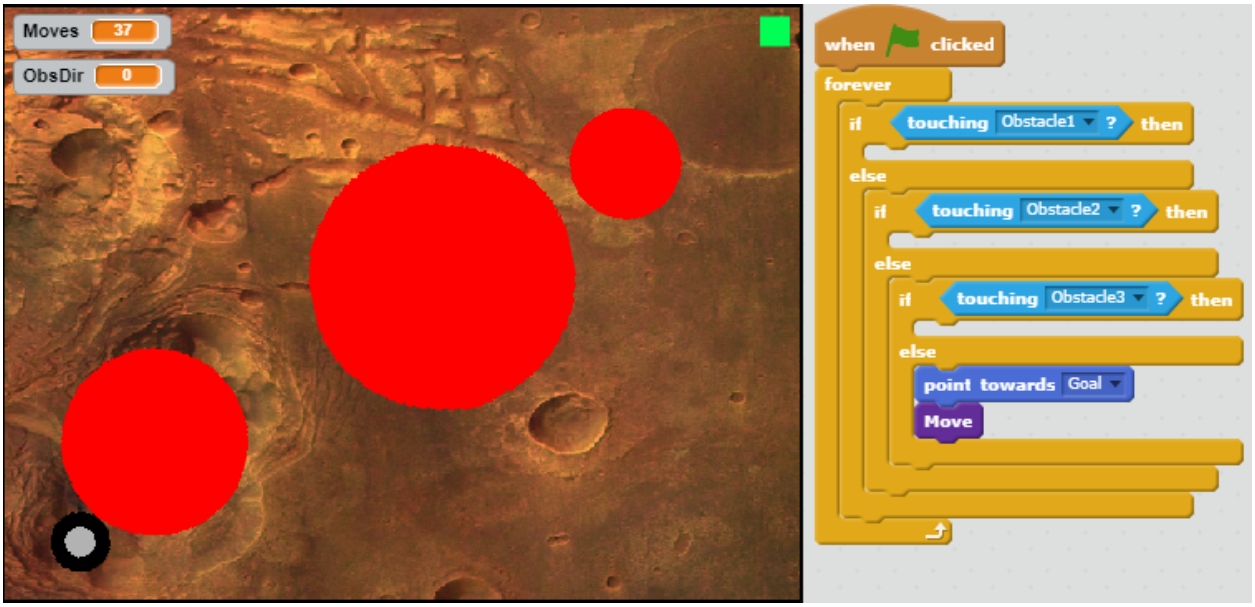

Figure 3.7: *Mars Pathfinding AI in Scratch*

the goal, students are encouraged to modify their program to try and reduce the length of the path taken by adjusting how the agent makes decisions. This culminating activity also reinforces many previously covered skills from computational thinking. A screenshot of this activity is shown in Figure 3.7.

At the end of the week, each student once again completes the survey given at the beginning of the week. That data is used to demonstrate any changes in student self-efficacy and familiarity with computer science concepts. The assessments are discussed in Chapter 5 and the results are given and discussed in Chapters 6 and 7.

The major topics covered in day 4, annotated with the related computational thinking skills from Table 2.1, are listed below. They are also placed within the revised Bloom's taxonomy in Table 3.4.

1) Understand what the field of artificial intelligence entails (QUE)
2) Discuss the Turing test and Chinese room experiment (ABS, DEC, QUE)
3) Learn how a neural network is trained (ABS, QUE)
4) Implement a two-phase artificial agent for a video game (ALG, ABS, DEC, PAR, CON, TAD, IAI)

5) Implement and optimize a pathfinding artificial agent (ALG, ABS, DEC, PAR, CON, TAD, IAI)

Table 3.4: *Revised Bloom's Taxonomy Analysis of Day 4 Outcomes*

	Remember	**Understand**	**Apply**	**Analyze**	**Evaluate**	**Create**
Factual		2				
Conceptual		1				
Procedural		3	4,5			
Metacognitive						

Chapter 4

Implementation

The *Mission to Mars* curriculum was taught to students during the 2016 and 2017 sessions of the *STEM Summer Institute*,[75] hosted by the Manhattan-Ogden USD 383 school district. This chapter discusses the structure of the camp and the process of delivering and refining the lessons over time.

4.1 STEM Camp Structure

STEM Summer Institute is a summer camp for students in 5th grade through 9th grade, coming primarily from the Manhattan-Ogden USD 383 school district. It was originally funded through a grant from the United States Department of Defense. The stated goal of the camp is "to raise student achievement levels and increase enrollments in science, technology, engineering and math, or STEM, careers."[84]

The camp takes place during four weeks each year in June, and is located primarily on the Kansas State University campus. Students are able to select four classes from a list of classes available each year, with each class lasting for one week. Most class groups each week are between 15 and 20 students in size. The camp meets Monday through Thursday during the week, with students in attendance from 8:30 to 11:30 AM each day. Each Friday is reserved for teacher activities and planning time.

Each class is typically marked as appropriate either for younger students in 5th and 6th grades, or older students in 7th through 9th grades. *Mission to Mars* was designed for the younger group of students.

4.2 Teacher Duties

Each year, *Mission to Mars* was taught cooperatively between a teacher from the Manhattan-Ogden USD 383 school district, acting as an expert in teaching young students, and a Computer Science faculty member, acting as a subject-matter expert. During the spring semester before each summer camp, those individuals would typically meet several times to discuss ideas for the class and build or refine existing curricula and activities.

In addition, several years the group would also include one or more pre-service teachers from Kansas State University's College of Education. They would be assisting with the camp as part of a teaching experience class for college credit in their program. Their goal was to gain experience teaching young students in a STEM field, while working with and observing a more experienced teacher and a subject-matter expert. As part of their responsibilities, the pre-service teachers were directed to lead at least one lesson each summer, typically during the last week of camp.

During each camp, the subject-matter expert would typically lead the class for most of the first week, introducing the curriculum to both the students and the teachers she or he was partnered with. During the next three weeks, the subject-matter expert would typically take the lead on fewer and fewer lessons, with the partnered teacher taking the lead as his or her comfort level allowed. Whenever possible, the teaching group would try to coordinate their teaching efforts, joining in when appropriate on each lesson with comments or feedback. In addition, the pre-service teachers would select a lesson and start learning how to present it. They would typically do so for a grade during the last week of camp.

While presenting the lessons, each teacher and pre-service teacher would constantly move about the classroom, observing students as they worked. Whenever a student was confused, had a question, or was unable to keep up with an activity, a nearby teacher would usually

jump in and help without disrupting the rest of the class. In addition, other teachers were encouraged to contribute to the class at any time if they had a comment, suggestion, or correction. Many times they were able to observe students acting confused much more quickly than the teacher leading the lesson. Since the class itself is very short, it is important to always keep the students on task and deal with issues quickly.

At the end of each week, the teachers would typically meet as a group to discuss the class and any changes that needed to be made. Through that process, the curriculum was continually changed and improved as questions arose or ideas were presented. In addition, teachers were required to submit formal lesson plans each year, which would typically be written by the USD 383 teacher in coordination with the pre-service teachers.

During the summer of 2016, recordings were made while teaching several of the lessons. This was primarily for students who were unable to attend class or if they wanted to review what was covered later on. The video recordings also served as a valuable aid for teachers who were asked to lead the lessons in 2017, when the primary author of the curriculum was unavailable to work as a subject-matter expert. In addition, prior to the 2017 session, the author formalized the curriculum by creating lesson plans for each day that could be shared with future teachers. Those lesson plans, along with the associated activities and slides, serve as the primary artifacts for the curriculum.

Chapter 5

Assessment

As discussed in section 2.4, there are three major methods to assess student learning. The first method, knowledge assessments, can produce some of the strongest results. By directly assessing student learning and retention, it is trivial to demonstrate that the class was effective in changing the students' understanding of the field. Unfortunately, knowledge assessments can be very time-consuming to construct and perform, and typically take the form of a test or quiz that students must complete. Because this curriculum was developed for a summer STEM camp, it was decided that performing knowledge assessment through tests or quizzes was not in keeping with the fun spirit of the camp. However, Chapter 8 will discuss one possible area of future work to informally include knowledge assessments.

Another method of student assessment is code analysis. While the students create several code artifacts throughout the camp, many of them are heavily scaffolded and do not require students to perform much independent work to complete the projects. Therefore, most student submissions would be so similar as to invalidate any observations of actual student performance. This method may be applicable for future versions of the curriculum targeted at older students.

Therefore, the third type of assessment presented, self-efficacy assessment, was used for this project. By measuring changes in student confidence and self-efficacy with computational thinking skills and programming concepts, this project seeks to demonstrate that the

course has had a positive impact on students. Each year, pre- and post-surveys of student experiences were conducted to gather this data.

5.1 Surveys

The survey instrument used for this project is primarily detailed in Weese and Feldhausen.[8] That paper details a self-efficacy survey that collects information about student attitudes regarding their own ability to think computationally and create computer programs. While the original survey was designed to determine if student self-efficacy in computational thinking and problem solving are correlated, the data regarding computational thinking is still relevant for this project. Therefore, the questions regarding problem solving are not included in this presentation of the survey and results, but were originally included in the survey given to students.

The questions are linked to accompanying skills in computational thinking and computer science detailed previously in Table 2.1. A summary of questions and their related computational thinking skills are presented in Table 5.1.

Cronbach's Alpha was used to measure the internal consistency of the survey questions on computational thinking skills. The Cronbach's Alpha of the 2016 pre-survey was 0.859, the 2016 post-survey was 0.866, the 2017 pre-survey was 0.882, and the 2017 post-survey was 0.836. All of those values show a good internal consistency, demonstrating that the survey is reliable.

Along with the self-efficacy information, the survey collected some information about students' previous experiences with computer programming, STEM outreach events, and previous attendance at the *STEM Summer Institute*, as well as some very basic demographic information. In addition, in 2017 questions were added to elicit information about students' socio-economic status, based on the works of Entwisle and Astone,[85] Currie et al.,[86] and Hauser.[87] This information can be used to group the results based on some student characteristics that are of interest to this project.

Table 5.1: *Self-Efficacy Survey Questions and Related CT Skill*[8]

	I can write a computer program which...	
10	runs a step-by-step sequence of commands	ALG - Algorithms
11	does math operations like addition and subtraction	ALG - Algorithms
12	uses loops to repeat commands	CON - Control Flow
13	responds to events like pressing a key on the keyboard	CON - Control Flow
14	only runs specific commands when a specific condition is met	CON - Control Flow
15	does more than one thing at the same time	PAR - Parallelization
16	uses messages to talk with different parts of the program	PAR - Parallelization
17	can store, update, and retrieve values	DAT - Data
18	uses custom blocks	ABS - Abstraction
	When creating a computer program I...	
19	make improvements one step at a time and work new ideas in as I have them	IAI - Incremental and Iterative
20	run my program frequently to make sure it does what I want and fix any problems I find	TAD - Testing and Debugging
21	share my programs with others and look at others' programs for ideas	USE - Reuse and Remix
22	break my program into multiple parts to carry out different actions	DEC - Problem Decomposition
	Impact	
23	I understand how computer programming can be used in my daily life	QUE - Questioning

Chapter 6

Results

The results from the surveys for the 2016 and 2017 summer camps are presented here. A full discussion of the results and inferred information can be found in Chapter 7.

6.1 Survey Result Data

For each survey, Cohen's d effect size was calculated as a measure of the size of the change in student scores from the pre-survey to the post-survey.[88] Entries with significant p-values less than 0.01 are shaded. Significant effect sizes are marked according to Cohen's recommended reference points as follows:[88]

- *Small effect $\geq$0.2 is italicized*
- **Medium effect $\geq$0.5 is bolded**
- ***Large effect $\geq$0.8 is bolded and italicized***

Rows are marked by the computational thinking skills listed in Table 2.1. The results also grouped by how each skill was framed in the survey instrument presented in Table 5.1, with "CT" representing skills framed within computational thinking from survey questions 10-23. The columns are marked with the following abbreviations for various student groupings:

- **Male** - Male-identifying students

- **Female** - Female-identifying students
- **NoS** - Haven't previously attended a STEM event
- **STEM** - Have previously attended a STEM event
- **OutS** - Have previously attended a STEM event outside of *STEM Summer Institute*
- **SI** - Have previously attended *STEM Summer Institute*
- **SB** - Have previously attended STARBASE[89]
- **>$10** - More than $10 in weekly spending money
- **≤$10** - $10 or less in weekly spending money
- **Cell** - Have a cell phone of their own
- **NoC** - No cell phone of their own
- **MS** - At least one parent has MS degree or higher
- **NoMS** - No parent has MS degree or higher

Results from the summer 2016 cohort are given in Table 6.1, and similarly results from the summer 2017 cohort are presented in Table 6.2. Unfortunately, the post-survey was not given to one of the classes during the summer 2017 cohort, so only valid surveys from the other three weeks are included, leading to a smaller number of student responses overall. Combined results for both summer sessions are given in Table 6.3.

Table 6.1: *2016 Survey Results*

Skill	All	Male	Female	NoS	STEM	OutS	SI	SB
CT - ALG	**0.590**	**0.592**	**0.601**	**0.686**	**0.531**	**0.588**	*0.327*	**0.626**
CT - ABS	**0.501**	*0.480*	**0.586**	*0.492*	**0.514**	**0.511**	*0.369*	**0.563**
CT - DEC	**0.530**	*0.479*	**0.731**	**0.592**	*0.490*	**0.685**	0.131	**0.680**
CT - PAR	*0.444*	*0.450*	*0.423*	**0.586**	*0.355*	*0.445*	*0.279*	*0.444*
CT - DAT	**0.728**	**0.795**	**0.520**	**0.639**	**0.786**	**0.779**	***0.859***	***0.828***
CT - CON	*0.275*	*0.214*	*0.477*	*0.272*	*0.283*	*0.384*	0.110	*0.394*
CT - IAI	*0.229*	*0.228*	*0.233*	*0.295*	0.186	*0.388*	-0.343	*0.384*
CT - TAD	0.091	0.094	0.082	0.138	0.060	0.180	-0.279	0.197
CT - USE	*0.248*	0.191	**0.509**	*0.394*	0.147	0.140	*0.333*	*0.232*
CT - QUE	**0.631**	**0.682**	*0.498*	**0.519**	**0.708**	**0.705**	**0.515**	**0.789**
# Students	56	42	14	21	35	29	16	26
Pre Mean	3.763	3.734	3.850	3.640	3.837	3.779	4.032	3.719
Post Mean	4.187	4.151	4.296	4.104	4.237	4.243	4.270	4.219

Table 6.2: *2017 Survey Results*

Skill	All	Male	Female	NoS	STEM	OutS	SI	SB	>$10	≤$10	Cell	NoC	MS	NoMS
CT - ALG	**0.519**	**0.544**	**0.516**	0.066	**0.622**	**0.688**	**0.717**	**0.740**	***1.038***	*0.467*	**0.654**	*0.266*	**0.557**	***1.189***
CT - ABS	*0.452*	**0.639**	*0.305*	*0.381*	*0.469*	**0.676**	*0.363*	***0.893***	-0.206	**0.552**	*0.481*	*0.395*	0.108	***1.189***
CT - DEC	*0.408*	*0.236*	**0.673**	*0.321*	*0.425*	*0.485*	*0.384*	*0.498*	0.000	*0.460*	*0.424*	*0.366*	**0.551**	***0.841***
CT - PAR	**0.539**	**0.636**	*0.379*	-0.060	**0.672**	**0.716**	**0.706**	***1.071***	**0.603**	**0.534**	**0.676**	*0.253*	*0.355*	**0.728**
CT - DAT	**0.757**	***0.826***	**0.605**	-0.167	***0.948***	***0.961***	***0.899***	***1.061***	**0.683**	**0.762**	**0.685**	***0.905***	*0.266*	***1.189***
CT - CON	*0.419*	*0.498*	*0.399*	0.073	*0.499*	**0.581**	**0.545**	**0.677**	***0.938***	*0.371*	**0.562**	0.132	**0.519**	**0.627**
CT - IAI	0.120	-0.047	*0.273*	-0.188	0.179	0.132	*0.246*	*0.295*	**0.586**	0.067	*0.225*	-0.091	0.000	***0.841***
CT - TAD	**0.576**	**0.574**	***0.821***	0.000	**0.694**	***0.817***	**0.700**	***0.827***	0	**0.644**	**0.741**	*0.212*	*0.309*	***0.841***
CT - USE	0.057	0.045	0.079	*0.323*	0.000	0.084	0.123	0.069	-1.000	0.195	0.000	0.187	0.115	***0.841***
CT - QUE	*0.378*	**0.605**	0.083	***1.226***	*0.259*	0.094	*0.363*	*0.351*	**0.500**	*0.385*	*0.449*	*0.210*	*0.406*	0
# Students	34	21	12	6	28	22	16	13	4	30	23	11	17	2
Pre Mean	3.742	3.704	3.790	3.826	3.724	3.694	3.666	3.766	3.967	3.712	3.720	3.787	3.698	4.239
Post Mean	4.133	4.116	4.174	3.942	4.174	4.178	4.171	4.304	4.380	4.100	4.191	4.012	4.054	4.652

Table 6.3: *Combined 2016 and 2017 Survey Results. Columns marked with an asterisk include data from 2017 only*

Skill	**All**	**Male**	**Female**	**NoS**	**STEM**	**OutS**	**SI**	**SB**	**>$10***	**≤$10***	**Cell***	**NoC***	**MS***	**NoMS***
CT - ALG	**0.564**	**0.576**	**0.563**	**0.561**	**0.570**	0.629	**0.520**	**0.662**	***1.038***	*0.467*	**0.654**	*0.266*	**0.557**	***1.189***
CT - ABS	*0.483*	**0.533**	*0.447*	*0.467*	*0.493*	**0.584**	*0.365*	**0.675**	-0.206	**0.552**	*0.481*	*0.395*	0.108	***1.189***
CT - DEC	*0.484*	*0.400*	**0.702**	**0.532**	*0.461*	**0.598**	*0.258*	**0.617**	0.000	*0.460*	*0.424*	*0.366*	**0.551**	***0.841***
CT - PAR	*0.478*	**0.509**	*0.403*	*0.454*	*0.491*	**0.558**	*0.493*	**0.624**	**0.603**	**0.534**	**0.676**	*0.253*	*0.355*	**0.728**
CT - DAT	**0.739**	***0.806***	**0.559**	*0.474*	***0.860***	***0.861***	***0.879***	***0.911***	**0.683**	**0.762**	**0.685**	***0.905***	*0.266*	***1.189***
CT - CON	*0.328*	*0.304*	*0.441*	*0.232*	*0.381*	*0.471*	*0.336*	*0.490*	***0.938***	*0.371*	**0.562**	0.132	**0.519**	**0.627**
CT - IAI	0.189	0.138	*0.252*	*0.205*	0.182	*0.268*	-0.032	*0.351*	**0.586**	0.067	*0.225*	-0.091	0.000	***0.841***
CT - TAD	*0.266*	*0.252*	*0.384*	0.111	*0.336*	*0.452*	*0.210*	*0.413*	0	**0.644**	**0.741**	*0.212*	*0.309*	***0.841***
CT - USE	0.177	0.144	*0.286*	*0.378*	0.079	0.115	*0.224*	0.173	-1.000	0.195	0.000	0.187	0.115	***0.841***
CT - QUE	**0.535**	**0.657**	*0.306*	**0.623**	*0.499*	*0.417*	*0.436*	**0.637**	**0.500**	*0.385*	*0.449*	*0.210*	*0.406*	0
# Students	90	63	26	27	63	51	32	39	4	30	23	11	17	2
Pre Mean	3.755	3.724	3.822	3.681	3.787	3.742	3.849	3.735	3.967	3.712	3.720	3.787	3.698	4.239
Post Mean	4.167	4.139	4.240	4.068	4.209	4.215	4.221	4.247	4.380	4.100	4.191	4.012	4.054	4.652

6.2 Knowledge Assessment Results

In addition to the survey results, during the summer of 2016 an online trivia game creator was used to collect some rudimentary data regarding student knowledge retention. The game consisted of seven to eight multiple-choice questions, with each student responding individually on her or his computer or mobile device. While the trivia game awarded more points to students who answered quickly, students were encouraged to take their time to come up with the correct answer.

The questions used, as well as the correct response rate for students each week, are given in Table 6.4.

Table 6.4: *Knowledge Assessment Questions and Correct Response Rate*

Question	**Wk1**	**Wk2**	**Wk3**	**Wk4**
1. What is a character called in Scratch?	100%	88%	100%	100%
2. How do you start the action in Scratch?	94%	91%	87%	100%
3. What is the difference between a list and a variable?	94%	94%	80%	69%
4. Which tool allows you to duplicate code?	76%	100%	100%	100%
5. Who came up with the GCD algorithm?	71%	88%	53%	62%
6. Which blocks would you use to compare numbers?	29%	94%	93%	92%
7. What is the number 8 in binary?	41%	47%	40%	31%
8. What is the purpose of a simulation?	N/A	94%	100%	100%

Chapter 7

Discussion of Results

This chapter contains the analysis and discussion of the results presented in Chapter 6. It will address each research question individually.

7.1 Research Question 1

RQ1) What is the impact of this curriculum on student self-efficacy with computational thinking skills?

The 2016 results in Table 6.1 show a small or medium effect size for each computational thinking skill except for testing and debugging (CT-TAD). In addition, all but two of those skills also have a p-value less than 0.01, confirming their significance. This clearly demonstrates an increase in student self-efficacy with computational thinking skills in 2016.

The 2017 results in Table 6.2 show a small or medium effect size for each computational thinking skill except for iterative and incremental development (CT-IAI) and reuse and remix (CT-USE). Of those, only 4 have p-values less than 0.01: algorithms (CT-ALG), parallelization (CT-PAR), data (CT-DAT), and control flow (CT-CON). While there are not as many significant results in this year's data, it still shows an increase in student self-efficacy in 2017.

The combined results from 2016 and 2017 in Table 6.3 are similar, with each compu-

tational thinking skill except for iterative and incremental (CT-IAI) and reuse and remix (CT-USE) showing a small or medium effect size. Of those, all except testing and debugging (CT-TAD) have a p-value less than 0.01. Those results also support the claim that students experience an increase in self-efficacy with computational thinking skills as a result of the curriculum.

Based on this analysis of the results, the curriculum does appear to cause an increase in student self-efficacy with several computational thinking skills, answering the first research question.

7.2 Research Question 2

RQ2) What is the relationship between the computational thinking skills covered in the curriculum and the student self-efficacy with those skills?

The curriculum design given in Chapter 3 describes the computational thinking skills covered each day, as well as the level at which it is covered according to Bloom's revised taxonomy. Using that information, the computational thinking skills covered most often and at depth are algorithms (CT-ALG) and abstraction (CT-ABS). Other skills that are covered often are data (CT-DAT), iterative and incremental development (CT-IAI), and testing and debugging (CT-TAD). These five computational thinking skills comprise the core skills of the curriculum for the purpose of this analysis.

In the 2016 results given in Table 6.1, three of the five computational thinking skills with medium effect size are represented in the core skills given above: algorithms (CT-ALG), abstraction (CT-ABS) and data (CT-DAT). Each has a p-value less than 0.01, indicating a significant result. Iterative and incremental development (CT-IAI) has a small effect size, while testing and debugging (CT-TAD) does not have an effect size large enough to be considered meaningful. So, while three of the five core skills clearly show an increase in student self-efficacy, two of them do not.

The 2017 results given in Table 6.2, three of the four computational thinking skills with medium effect size are represented in the core skills given above: algorithms (CT-ALG),

data (CT-DAT), and testing and debugging (CT-TAD). With the exception of testing and debugging (CT-TAD), each has a p-value less than 0.01, indicating a significant result. Abstraction (CT-ABS) has a small effect size, while iterative and incremental development (CT-IAI) does not have an effect size large enough to be considered meaningful. Once again, three of the five skills clearly show an increase in self-efficacy, while two of them do not.

The combined results from 2016 and 2017 in Table 6.3 are similar to those given above. Two of the three computational thinking skills with medium effect size are represented in the core skills given above: algorithms (CT-ALG) and data (CT-DAT). Each has a p-value less than 0.01, indicating a significant result. Abstraction (CT-ABS) and testing and debugging (CT-TAD) have small effect sizes, with abstraction (CT-ABS) also possessing a significant p-value. Iterative and incremental development (CT-IAI) does not have an effect size large enough to be considered meaningful.

Based on the analysis given above, there is a loose correlation between the computational thinking skills presented in the curriculum and the increase in student self-efficacy with those skills. Algorithms (CT-ALG) and data (CT-DAT) have a medium effect size in each data set with a significant p-value, and a majority of the computational thinking skills with medium effect sizes in each data set are represented in the five core skills listed earlier in this section. The representation of the core skills in the data set varies each year, but in general each set shows that the core skills are represented well among the skills with the highest effect size.

7.3 Research Question 3

RQ3) What is the relationship between student factors such as gender, previous STEM experience, or socio-economic status indicators and the observed student self-efficacy with computational thinking skills?

Each of the surveys also collected demographic data from each student, allowing the creation of population groups to further analyze the data. While some of the population groups are very small, they can provide additional insight into the types of students attending the camp, what effect the curriculum has, and how that could possibly relate to their

background.

The first population grouping analyzed was based on self-reported student gender. As shown in Table 6.3, there were 63 males and 26 females across both cohorts. The effect size for many skills was similar between the two groups, with the largest differences observed in problem decomposition (CT-DEC), where females outperformed males, and data (CT-DAT), where males outperformed females. In addition, males had a much larger overall effect size in the impact area (CT-QUE). While it may be tempting to draw conclusions about this result, it is difficult to find any research that supports a clear relation between gender and ability to perform computational tasks. Instead, much of the difference is attributed to how each gender is encouraged or discouraged to pursue certain careers or interests. Additionally, the major differences can be attributed to a small sample size and individual variation between students and their previous experiences.

The next population group considered includes students with previous STEM experience (STEM) and students without previous STEM experience (NoS). There were very few major differences between these two groups, with the only large difference in the data (CT-DAT) area where students with previous STEM experience had a much larger effect size. Interestingly, students without previous STEM experience had a somewhat larger effect size in the overall impact area (CT-QUE). This could be indicative of students with previous experience being more able to learn new concepts easily by building on their existing knowledge and interest, such as when working with data representation on the third day of the curriculum. This is a key expected outcome from both the constructivist and constructionist paradigms in education. Students without previous experience gained more overall by working with computers and becoming much more comfortable with computer programming itself, outside of any particular skill.

Similarly, groups of students who had previously attended the *STEM Summer Institute* (SI), students who had attended another STEM event outside of *STEM Summer Institute* (OutS), and students who had previously attended STARBASE (SB), a structured STEM program for 5th graders in the area,[89] were analyzed. Once again, there was not a large difference observed between the three groups. The most notable difference was observed

in the *STEM Summer Institute* group, who had a lower effect size in both the abstraction (CT-ABS) and problem decomposition (CT-DEC) skills. This difference could be due to the fact that those students have already gained some experience working with those areas at previous *STEM Summer Institute* classes.

In 2017, some questions were added to the survey to learn more about the students' socioeconomic status. The three groupings are based on student weekly spending money (>\$10, ≤\$10), whether the student had a cell phone of their own (Cell, NoC), and whether one or more parent had at least a master's degree (MS, NoMS). The groupings based on spending money and parent's education are inconclusive due to small sample size. Only 4 students reported having more than \$10 in weekly spending money, and only 2 students reported neither parent with at least a master's degree. One interesting result is that a vast majority of students attending the course had at least one highly-educated parent. It could indicate a connection between the parents' education level and the participation of students in optional educational camps such as this one.

The third grouping, based on student cell phone possession, does provide a bit of information. Generally, students who had access to a cell phone of their own had much larger effect sizes than students who didn't. This could be attributed to a couple of factors. One factor is that students who have easy access to a cell phone are much more comfortable and familiar with technology, making it easier for them to learn computer programming and computational thinking skills. Similarly, students who have a cell phone tend to have a higher socioeconomic status, so there may be additional outside factors giving them an advantage over their peers.

7.4 Knowledge Retention

The knowledge assessment results in Table 6.4 also provide a few interesting insights. While the questions asked are in no way intended to be a true measure of student knowledge, they serve as a good first-step in that direction. For most questions, students responded correctly at a rate of 80% or higher. The questions students struggled with the most were questions

5 and 7. In the case of question 5, it requires remembering a name that was only briefly mentioned a couple of times on the second day of class. For question 7, it shows that their familiarity with binary does not hold very well the next day. However, because students may have been under pressure to respond quickly, they could have selected an option that appeared correct without careful consideration. This will be one area of interest in future knowledge assessments.

Also, worth noting is the result from question 6 on week 1. During the first week, the options for that question were "sensing," "operators," "data," and "operators and data." The last option was correct, but many students incorrectly chose "operators" based on their extensive experience using those blocks. For the later weeks, that question was adjusted to accept either "operators" or "data" as a correct answer, resulting in much better results from students.

Chapter 8

Future Work

While this project has yielded many unique artifacts and outcomes, there are still several areas that could be improved. This chapter discusses some areas of future work and improvement based on this project.

8.1 Rewrite using Educational Theory and Previous Results

The first major area of future work would be to rebuild the curriculum from the ground up, following what has been learned researching educational theory and analyzing the results of the previous curriculum. When the *Mission to Mars* curriculum was first created, it was done to fill the immediate need for a curriculum for an upcoming summer camp. At that time, the author did not have much experience with educational research, and relied primarily on experience as a teacher and intuition from helping with this and other STEM outreach events.

The next goal would be to continue this work by recreating the curriculum, using concepts from cognitive apprenticeship, extensive framing, and scaffolded lessons to reach the zone of proximal development, as discussed in Chapter 2. Linking the curriculum to these concepts will strengthen the connection between observed results and the exact ways that

the curriculum has achieved those outcomes through the use of these concepts.

In addition, many students attend the *STEM Summer Institute* multiple years, so several students end up attending the same class more than once. By having additional curricula available, it helps avoid using the same curriculum in sequential years.

Finally, by building on the results from the self-efficacy surveys, it will help identify areas of weakness in the current curriculum so they can be addressed more directly in the design of lessons and activities.

8.2 Knowledge Assessments

The second significant area of future work involves creating formal knowledge assessments. While the current self-efficacy instrument is very useful and should continue to be used, additional instruments to measure student knowledge gains as a result of the curriculum would also be helpful. By linking student knowledge gains to changes in self-efficacy, it will further establish the link between student outcomes and their own confidence in working with the field.

In addition, it will help reveal more about students' prior knowledge coming to the camp. Using a pre-test to determine what areas students already have experience with, the curriculum can be adapted to meet their unique backgrounds and interests.

8.3 Languages

Currently, the curriculum is designed to use the Scratch programming environment.[20] However, there are many additional tools and environments available that could be considered. Some possible areas would be moving to a text-based programming language such as Python, or building a new environment based on Scratch or Google's Blockly library.[90]

In addition to using a different language, it would increase the amount data that can be collected from students based on their programming environment. Currently, Scratch does not have a way to collect data beyond the resultant projects that students create. However,

many other environments will collect intermediate data as students work on their projects, providing a greater ability to analyze student thought processes and techniques used to complete the projects.

8.4 Other Venues

In addition to the *STEM Summer Institute*, in the future this curriculum could be adapted for other venues. Some possible examples would be the Hour of Code,[91] other STEM outreach events, or even more in-depth programming classes for young students.

Beyond that, many of the activities in this curriculum could be adapted into stand-alone lessons for teachers at various grade levels. A few activities were inspired by or adapted from activities already used in the author's own college teaching experience, and they could easily be used in many other settings.

8.5 Code Analysis

The last area of future work identified is related to code analysis. By collecting additional information from the programming environment, code analysis techniques from the field of big data can be used to gain further insight into the knowledge and thought processes of students as they complete the projects. Fields et al.[71] provides a good starting point for this area of further research, as they collected and analyzed data from a Scratch-based summer camp very similar to the *STEM Summer Institute*.

Chapter 9

Conclusion

This thesis presents a curriculum, named *Mission to Mars*, for 5th and 6th grade students attending a STEM summer camp. The curriculum aims to increase student self-efficacy working with computational thinking skills through computer programming with hands-on activities and scaffolded programming projects. It also employs expansive framing through the use of a common theme from popular culture to increase student interest.

By collecting data from two cohorts of the summer camp, the following research questions were addressed.

RQ1) What is the impact of this curriculum on student self-efficacy with computational thinking skills?

RQ2) What is the relationship between the computational thinking skills covered in the curriculum and the student self-efficacy with those skills?

RQ3) What is the relationship between student factors such as gender, previous STEM experience, or socio-economic status indicators and the observed student self-efficacy with computational thinking skills?

Regarding RQ1, analysis of the results shows that this curriculum increases student self-efficacy with several computational thinking skills. For RQ2, the data shows that the core skills addressed in the curriculum make up a majority of the skills with the highest measured

effect size for the increase in student self-efficacy. This provides evidence that those core skills are correlated with measured increases in student self-efficacy with those skills. To answer RQ3, many of comparisons made between different student groupings do not result in significant differences between the groups, showing that there may not be a relationship between student factors and observed self-efficacy. While some differences are present, many of them could be the result of individual student differences.

With these results, the curriculum does appear to result in an increase in student self-efficacy with many computational thinking skills, and the skills focused upon in the curriculum generally show larger effect sizes in student self-efficacy than other skills. This demonstrates the effectiveness of the curriculum overall, making it a useful result of this work.

Bibliography

[1] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006. ISSN 0001-0782. doi: 10.1145/1118178.1118215. URL http://doi.acm.org/10.1145/1118178.1118215.

[2] Robert H. Tai, Christine Qi Liu, Adam V. Maltese, and Xitao Fan. Planning Early for Careers in Science. *Science*, 312(5777):1143–1144, 2006. ISSN 0036-8075. doi: 10.1126/science.1128690. URL http://science.sciencemag.org/content/312/5777/1143.

[3] Shuchi Grover, Roy Pea, and Stephen Cooper. Remedying misperceptions of computer science among middle school students. *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14*, pages 343–348, 2014. doi: 10.1145/2538862.2538934. URL http://dl.acm.org/citation.cfm?doid=2538862.2538934.

[4] Allan Collins, John Seely Brown, and Ann Holum. Cognitive Apprenticeship: Making Thinking Visible. *American Educator*, 15(3):6 – 11, 38–46, 1991.

[5] Joshua Levi Weese, Russell Feldhausen, and Nathan H. Bean. The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking. In *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016. ASEE Conferences. doi: 10.18260/p.26179. URL https://peer.asee.org/26179.

[6] Lorin W. Anderson and David R. Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York, 2001.

[7] David R Krathwohl. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice*, 41(4):212–218, 2002. doi: 10.1207/s15430421tip4104_2. URL http://dx.doi.org/10.1207/s15430421tip4104_2.

[8] Joshua Levi Weese and Russell Feldhausen. STEM Outreach: Assessing Computational Thinking and Problem Solving. In *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. ASEE Conferences. URL https://peer.asee.org/28845.

[9] Joshua Levi Weese. *Bringing Computational Thinking to K-12 and Higher Education*. PhD thesis, Kansas State University, 2017. URL http://hdl.handle.net/2097/35430.

[10] Russell Feldhausen, Joshua Levi Weese, and Nathan H. Bean. Increasing student self-efficacy in computational thinking via stem outreach programs. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 302–307, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5103-4. doi: 10.1145/3159450.3159593. URL http://doi.acm.org/10.1145/3159450.3159593.

[11] Seymour Papert. An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning*, 1(1):95–123, 1996. doi: 10.1007/BF00191473. URL https://doi.org/10.1007/BF00191473.

[12] Jeannette M. Wing. Research notebook: Computational thinking - What and why? *The Link Magazine*, 2011. URL http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why. [Online; accessed 21 July 2017].

[13] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. Computational Thinking for Youth in Practice. *ACM Inroads*, 2(1):32–37, February 2011. ISSN 2153-2184. doi: 10.1145/1929887.1929902. URL http://doi.acm.org/10.1145/1929887.1929902.

[14] Aman Yadav, Chris Mayfield, Ninger Zhou, Susanne Hambrusch, and John T. Korb. Computational Thinking in Elementary and Secondary Teacher Education. *Trans. Comput. Educ.*, 14(1):5:1–5:16, March 2014. ISSN 1946-6226. doi: 10.1145/2576872. URL http://doi.acm.org/10.1145/2576872.

[15] Aman Yadav, Hai Hong, and Chris Stephenson. Computational Thinking for All: Pedagogical Approaches to Embedding 21st Century Problem Solving in K-12 Classrooms.

TechTrends, 60(6):565–568, Nov 2016. ISSN 1559-7075. doi: 10.1007/s11528-016-0087-7. URL https://doi.org/10.1007/s11528-016-0087-7.

[16] Aman Yadav, Chris Stephenson, and Hai Hong. Computational Thinking for Teacher Education. *Commun. ACM*, 60(4):55–62, March 2017. ISSN 0001-0782. doi: 10.1145/2994591. URL http://doi.acm.org/10.1145/2994591.

[17] Nathan Bean, Joshua Weese, Russell Feldhausen, and Richard Scott Bell. Starting from Scratch: Developing a Pre-service Teacher Training Program in Computational Thinking. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2015. doi: 10.1109/FIE.2015.7344237. URL https://doi.org/10.1109/FIE.2015.7344237.

[18] Shuchi Grover and Roy Pea. Computational Thinking in K12. *Educational Researcher*, 42(1):38–43, 2013. doi: 10.3102/0013189X12463051. URL http://dx.doi.org/10.3102/0013189X12463051.

[19] Linda Mannila, Valentina Dagiene, Barbara Demo, Natasa Grgurina, Claudio Mirolo, Lennart Rolandsson, and Amber Settle. Computational Thinking in K-9 Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, ITiCSE-WGR '14, pages 1–29, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3406-8. doi: 10.1145/2713609.2713610. URL http://doi.acm.org/10.1145/2713609.2713610.

[20] MIT Media Lab. Scratch - Imagine, Program, Share, 2017. URL https://scratch.mit.edu/. [Online; accessed 21 July 2017].

[21] Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, 2012.

[22] Google. Computational Thinking Concepts Guide, 2017. URL https://docs.google.com/document/d/1i0wg-BMG3TdwsShAyH_0Z1xpFnpVcMvpYJceHGWex_c/edit. [Online; accessed 24 August 2017].

[23] Lev Semenovich Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1980.

[24] Rheta DeVries. Vygotsky, Piaget, and Education: A Reciprocal Assimilation of Theories and Educational Practices. *New Ideas in Psychology*, 18(2):187–213, 2000.

[25] Edith Ackermann. Piaget's Constructivism, Papert's Constructionism: What's the Difference. In *Constructivism: Uses and Perspectives in Education*, pages 85–94, 2001.

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

[27] John Seely Brown, Allan Collins, and Paul Duguid. Situated Cognition and the Culture of Learning. *Educational Researcher*, 18(1):32–42, 1989. doi: 10.3102/0013189X018001032. URL http://dx.doi.org/10.3102/0013189X018001032.

[28] Roy D. Pea. Socializing the Knowledge Transfer Problem. *International Journal of Educational Research*, 11(6):639 – 663, 1987. ISSN 0883-0355. doi: 10.1016/0883-0355(87)90007-3. URL http://www.sciencedirect.com/science/article/pii/0883035587900073.

[29] Shuchi Grover and Roy Pea. Design-Based Research for Deeper Learning in an Online Introductory CS Course for Middle School Students. *Presented at the Annual Meeting of the American Educational Research Association*, 2015. URL https://www.sri.com/work/publications/design-based-research-deeper-learning-online-introductory-cs-course-middle-school.

[30] Shuchi Grover, Roy Pea, and Stephen Cooper. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25 (2):199–237, 2015. doi: 10.1080/08993408.2015.1033142. URL http://dx.doi.org/10.1080/08993408.2015.1033142.

[31] Shuchi Grover. *Foundations for Advancing Computational Thinking: Balanced Designs*

for Deeper Learning in an Online Computer Science Course for Middle School Students. PhD thesis, Stanford University, 2014. URL http://purl.stanford.edu/cc869py7832.

[32] Randi A. Engle, Phi D. Nguyen, and Adam Mendelson. The Influence of Framing on Transfer: Initial Evidence from a Tutoring Experiment. *Instructional Science*, 39(5): 603–628, Sep 2011. ISSN 1573-1952. doi: 10.1007/s11251-010-9145-2. URL https://doi.org/10.1007/s11251-010-9145-2.

[33] Randi A. Engle, Diane P. Lam, Xenia S. Meyer, and Sarah E. Nix. How Does Expansive Framing Promote Transfer? Several Proposed Explanations and a Research Agenda for Investigating Them. *Educational Psychologist*, 47(3):215–231, 2012. doi: 10.1080/00461520.2012.695678. URL http://dx.doi.org/10.1080/00461520.2012.695678.

[34] William Hart and Dolores Albarracín. What I Was Doing Versus What I Did. *Psychological Science*, 20(2):238–244, 2009. doi: 10.1111/j.1467-9280.2009.02277.x. URL http://dx.doi.org/10.1111/j.1467-9280.2009.02277.x. PMID: 19170935.

[35] Tony Greening. Computer Science: Through the Eyes of Potential Students. In *Proceedings of the 3rd Australasian Conference on Computer Science Education*, ACSE '98, pages 145–154, New York, NY, USA, 1998. ACM. ISBN 1-58113-018-X. doi: 10.1145/289393.289415. URL http://doi.acm.org/10.1145/289393.289415.

[36] Lori Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 27–31, New York, NY, USA, 2006. ACM. ISBN 1-59593-259-3. doi: 10.1145/1121341.1121352. URL http://doi.acm.org/10.1145/1121341.1121352.

[37] Allan Collins. Cognitive Apprenticeship and Instructional Technology. Technical Report. *BBN Systems and Technologies Corporation*, 1988.

[38] Aziz Ghefaili. Cognitive Apprenticeship, Technology, and the Contextualization of

Learning Environments. In *Journal of Educational Computing, Design & Online Learning*, pages 1–27, 2003.

[39] CS Unplugged. CS Unplugged, 2017. URL http://csunplugged.org/. [Online; accessed 1 August 2017].

[40] Google. Google For Education: Computational Thinking, 2017. URL https://edu.google.com/resources/programs/exploring-computational-thinking/. [Online; accessed 1 August 2017].

[41] Computer Science Teachers Association. CSTA K-12 Computer Science Standards, Revised 2017, 2017. URL http://www.csteachers.org/standards. [Online; accessed 31 July 2017].

[42] Miles Berry. Computing in the national curriculum. A guide for primary teachers. 2013. URL http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf.

[43] Peter Kemp. Computing in the national curriculum. A guide for secondary teachers. 2014. URL http://www.computingatschool.org.uk/data/uploads/cas_secondary.pdf.

[44] Google. International CS Education Standards, 2017. URL https://docs.google.com/spreadsheets/d/1SE7hGK5CkOlAf6oEnqkODPr8OOSdyGZmRnROhr0XHys/edit#gid=218360034. [Online; accessed 1 August 2017].

[45] Anybody Can Learn — Code.org, 2017. URL https://code.org/. [Online; accessed 1 August 2017].

[46] Computer Science Education Week, 2017. URL https://csedweek.org/. [Online; accessed 1 August 2017].

[47] Susan H. Rodger, Maggie Bashford, Lana Dyck, Jenna Hayes, Liz Liang, Deborah Nelson, and Henry Qin. Enhancing K-12 Education with Alice Programming Adven-

tures. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 234–238, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-820-9. doi: 10.1145/1822090.1822156. URL http://doi.acm.org/10.1145/1822090.1822156.

[48] R. Brook Osborne, Antony J. Thomas, and Jeffrey R.N. Forbes. Teaching with Robots: A Service-learning Approach to Mentor Training. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 172–176, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0006-3. doi: 10.1145/1734263.1734321. URL http://doi.acm.org/10.1145/1734263.1734321.

[49] Rachel Goldman, Amy Eguchi, and Elizabeth Sklar. Using Educational Robotics to Engage Inner-city Students with Technology. In *Proceedings of the 6th International Conference on Learning Sciences*, ICLS '04, pages 214–221. International Society of the Learning Sciences, 2004. URL http://dl.acm.org/citation.cfm?id=1149126.1149151.

[50] G. Nugent, B. Barker, N. Grandgenett, and V. Adamchuk. The use of Digital Manipulatives in K-12: Rrobotics, GPS/GIS and Programming. In *2009 39th IEEE Frontiers in Education Conference*, pages 1–6, Oct 2009. doi: 10.1109/FIE.2009.5350828. URL https://doi.org/10.1109/FIE.2009.5350828.

[51] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. Learning Computer Science Concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, pages 69–76, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0257-9. doi: 10.1145/1839594.1839607. URL http://doi.acm.org/10.1145/1839594.1839607.

[52] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. Habits of Programming in Scratch. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE '11, pages 168–172, New York,

NY, USA, 2011. ACM. ISBN 978-1-4503-0697-3. doi: 10.1145/1999747.1999796. URL http://doi.acm.org/10.1145/1999747.1999796.

[53] J. Moreno-León and G. Robles. Code to learn with Scratch? A systematic literature review. In *2016 IEEE Global Engineering Education Conference (EDUCON)*, pages 150–156, April 2016. doi: 10.1109/EDUCON.2016.7474546. URL https://doi.org/10.1109/EDUCON.2016.7474546.

[54] Amnon Shabo, Mark Guzdial, and John Stasko. Computer Science Apprenticeship: Creating Support for Intermediate Computer Science Students. In *Proceedings of the 1996 International Conference on Learning Sciences*, ICLS '96, pages 308–315. International Society of the Learning Sciences, 1996. ISBN 1-880094-23-1. URL http://dl.acm.org/citation.cfm?id=1161135.1161177.

[55] D. Brian Larkins, J. Christopher Moore, Louis J. Rubbo, and Laura R. Covington. Application of the Cognitive Apprenticeship Framework to a Middle School Robotics Camp. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 89–94, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445226. URL http://doi.acm.org/10.1145/2445196.2445226.

[56] Heidi Webb and Mary Beth Rosson. Using Scaffolded Examples to Teach Computational Thinking Concepts. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 95–100, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445227. URL http://doi.acm.org/10.1145/2445196.2445227.

[57] Alexander Repenning, David Webb, and Andri Ioannidou. Scalable Game Design and the Development of a Checklist for Getting Computational Thinking into Public Schools. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 265–269, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0006-3. doi: 10.1145/1734263.1734357. URL http://doi.acm.org/10.1145/1734263.1734357.

[58] Alexander Repenning, David C. Webb, Kyu Han Koh, Hilarie Nickerson, Susan B. Miller, Catharine Brand, Ian Her Many Horses, Ashok Basawapatna, Fred Gluck, Ryan Grover, Kris Gutierrez, and Nadia Repenning. Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools Through Game Design and Simulation Creation. *Trans. Comput. Educ.*, 15(2):11:1–11:31, April 2015. ISSN 1946-6226. doi: 10.1145/2700517. URL http://doi.acm.org/10.1145/2700517.

[59] John D. Bransford, Ann L. Brown, and Rodney R. Cocking. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academy Press, 2000.

[60] Jacqueline L. Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P. K. Ajith Kumar, and Christine Prasad. An Australasian Study of Reading and Comprehension Skills in Novice Programmers, Using the Bloom and SOLO Taxonomies. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 243–252, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-34-1. URL http://dl.acm.org/citation.cfm?id=1151869.1151901.

[61] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. Bloom's Taxonomy for CS Assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc. ISBN 978-1-920682-59-0. URL http://dl.acm.org/citation.cfm?id=1379249.1379265.

[62] John B Biggs and Kevin F. Collis. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, New York, 1982.

[63] Marie Bienkowski, Eric Snow, Daisy Rutstein, and Shuchi Grover. Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science: A First Look . SRI Technical Report, SRI International, Menlo Park, California, 2015. URL http://pact.sri.com/resources.html.

[64] Shuchi Grover. Systems of Assessments for Deeper Learning of Computational Thinking in K-12. In *Proceedings of the 2015 Annual Meeting of the American Educational Research Association*, pages 15–20, 2015.

[65] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 215–220, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1098-7. doi: 10.1145/2157136.2157200. URL http://doi.acm.org/10.1145/2157136.2157200.

[66] Linda Werner, Shannon Campe, and Jill Denner. Children Learning Computer Science Concepts via Alice Game-programming. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 427–432, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1098-7. doi: 10.1145/2157136.2157263. URL http://doi.acm.org/10.1145/2157136.2157263.

[67] Linda Werner, Jill Denner, and Shannon Campe. Children Programming Games: A Strategy for Measuring Computational Learning. *Trans. Comput. Educ.*, 14(4): 24:1–24:22, December 2014. ISSN 1946-6226. doi: 10.1145/2677091. URL http://doi.acm.org/10.1145/2677091.

[68] Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, Chelsea Laird, Felicia Lopez, Christine Pham, Jessica Suarez, and Robert Waite. Assessment of Computer Science Learning in a Scratch-based Outreach Program. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 371–376, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445304. URL http://doi.acm.org/10.1145/2445196.2445304.

[69] Jesús Moreno-León and Gregorio Robles. Dr. Scratch: A Web Tool to Automatically Evaluate Scratch Projects. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, pages 132–133, New York, NY, USA, 2015. ACM.

ISBN 978-1-4503-3753-3. doi: 10.1145/2818314.2818338. URL http://doi.acm.org/10.1145/2818314.2818338.

[70] Jesús Moreno-León, Marcos Román-González, Casper Harteveld, and Gregorio Robles. On the Automatic Assessment of Computational Thinking Skills: A Comparison with Human Experts. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 2788–2795, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4656-6. doi: 10.1145/3027063.3053216. URL http://doi.acm.org/10.1145/3027063.3053216.

[71] Deborah A. Fields, Lisa Quirke, Janell Amely, and Jason Maughan. Combining Big Data and Thick Data Analyses for Understanding Youth Learning Trajectories in a Summer Coding Camp. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 150–155, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844631. URL http://doi.acm.org/10.1145/2839509.2844631.

[72] Albert Bandura. Self-efficacy Mechanism in Human Agency. *American Psychologist*, 37 (2):122–147, 02 1982.

[73] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. Self-efficacy and Mental Models in Learning to Program. *SIGCSE Bull.*, 36(3):171–175, June 2004. ISSN 0097-8418. doi: 10.1145/1026487.1008042. URL http://doi.acm.org/10.1145/1026487.1008042.

[74] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 211–220, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4449-4. doi: 10.1145/2960310.2960329. URL http://doi.acm.org/10.1145/2960310.2960329.

[75] Manhattan-Ogden USD 383 : STEM Summer Institute, 2017. URL http://www.usd383.org/manhattan-ogden/district-office-/teaching-and-learning/stem/stem-summer-institute. [Online; accessed 13 July 2017].

[76] Andy Weir. *The Martian*. Crown, 2014.

[77] Ridley Scott (Director). The Martian. Twentieth Century Fox Film Corporation, 2015.

[78] CodeOrg. What Most Schools Don't Teach, Feb 2013. URL https://www.youtube.com/watch?v=nKIu9yen5nc. [Online; accessed 15 August 2017].

[79] Nathan H. Bean. Scratch, 2017. URL http://www.nathanhbean.com/scratch/. [Online; accessed 15 August 2017].

[80] Russell Feldhausen. CIS 115: Introduction to Computing Science, 2017. URL http://people.cs.ksu.edu/~russfeld/cis115spring2017/. [Online; accessed 24 October 2017].

[81] Gary Kacmarcik. Computer Science & Engineering for K-12, 2017. URL http://cse4k12.org/. [Online; accessed 24 October 2017].

[82] BANDAI NAMCO. Pac-Man, 2017. URL http://pacman.com/en/. [Online; accessed 25 October 2017].

[83] Chad Birch. Understanding Pac-Man Ghost Behavior, 2010. URL http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior. [Online; accessed 25 October 2017].

[84] Patrice Scott. Summer STEM Institute is an academic adventure, Jun 2012. URL https://www.k-state.edu/today/announcement.php?id=4050. [Online; accessed 13 July 2017].

[85] Doris R. Entwisle and Nan Marie Astone. Some Practical Guidelines for Measuring Youth's Race/Ethnicity and Socioeconomic Status. *Child Development*, 65(6):1521–1540, 1994. ISSN 00093920, 14678624. URL http://www.jstor.org/stable/1131278.

[86] Candace E. Currie, Rob A. Elton, Joanna Todd, and Stephen Platt. Indicators of socioeconomic status for adolescents: the who health behaviour in school-aged children survey. *Health Education Research*, 12(3):385–397, 1997. doi: 10.1093/her/12.3.385. URL http://dx.doi.org/10.1093/her/12.3.385.

[87] Robert M. Hauser. Measuring Socioeconomic Status in Studies of Child Development. *Child Development*, 65(6):1541–1545, 1994. ISSN 00093920, 14678624. URL http://www.jstor.org/stable/1131279.

[88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences - Second Edition*. Lawrence Erlbaum Associates, 1988. ISBN 0-8058-2083-5.

[89] Kansas STARBASE. Kansas STARBASE, 2017. URL http://www.kansasstarbase.org/. [Online; accessed 31 August 2017].

[90] Google Developers. Blockly, 2017. URL https://developers.google.com/blockly/. [Online; accessed 30 October 2017].

[91] Code.org. Hour of Code, 2017. URL https://hourofcode.com/us. [Online; accessed 30 October 2017].

Appendix A

Mission to Mars Curriculum

This appendix contains the full lecture notes for the *Mission to Mars* curriculum. The associated slides, activities, and other materials are available online at http://people.cs.ksu.edu/~russfeld/curriculum/.

Mission to Mars - USD 383 Summer STEM - Day 1

Summer 2017

Learning Objectives

- Students will be able to navigate and use Scratch with a basic knowledge, build scripts, and draw within the program

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/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/

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Have **Spirograph** Scratch file available on the Scratch website

Schedule

- 9:00 - Icebreaker
- 9:15 - STEM Surveys & Accounts
- 9:30 - Videos
- 9:45 - Learning Scratch
- 10:15 - Break
- 10:20 - Shapes in Scratch
- 10:50 - Spirograph
- 11:00 - Wrap-Up

Lecture Notes

1. [**Icebreaker**] Have teachers introduce and give a bit of information about themselves and what they'd like to achieve during the camp. Go around the room and have each student introduce herself/himself and say what they want to learn from the camp. You could also encourage students to ask any questions they have about the instructors or the class, just to set the stage for an open forum of ideas (provided it is PG rated, of course).

2. [**Surveys**] Before we get started, we'd like you to do an online survey about what you know so far. This helps with K-State's research about how to teach students to be computer programmers more effectively.

<<<STEM Surveys and Online Account Setup Here>>>

3. [**Code.org Video**] Computer Science is becoming an important part of many different fields, and knowledge of how computers work could be vital in the future. Code.org is a non-profit organization founded to help bring Computer Science to young people in schools and beyond. Let's take a look at their video promoting what they'd like to accomplish. **<Show Video>**.
    a. Discussion Points:
        i. What do you think about this video?
        ii. Do you think learning how computers work is important? Why/Why not?
        iii. What are some of the things you do every day that use computers? Can you do them without computers?

4. [**The Martian Video**] How can computer science be used in the real world? To help show how useful it is, this week we're going to put ourselves in the shoes of programmers at NASA working on a unique situation: someone is stranded on Mars! How many of you have seen the movie "The Martian"? <get feedback> Let's take a look at this clip from the start of the movie so we know what is going on. **<Show Video>.** This week, we'll tackle several issues presented in the movie and show how computer science can help us bring him home!

5. [**Schedule**] Here's today's Schedule: **<refer to slide>**

6. [**Introduce Scratch**] *Have students load the Scratch website, then instruct them to click on the Create button at the top. That will take them directly to the main Scratch editor. Take some time to describe the different parts of the Scratch editor:*
    a. The Stage
        i. Editing Backdrops
        ii. Editing Sounds
        iii. X & Y Coordinates (can relate back to cartesian coordinates in geometry)
        iv. X ranges from -240 to 240 and Y ranges from -180 to 180 (480 x 360 size)
    b. The Sprites
        i. How to choose a new sprite from the library
        ii. How to create a sprite from an uploaded image
        iii. How to paint a new sprite from scratch
        iv. Duplicating Sprites
        v. Deleting Sprites
        vi. Renaming Sprites (click the blue (i) on the sprite when selected)
        vii. Editing Sprite Costumes
        viii. Editing Sounds
    c. The Palette (http://wiki.scratch.mit.edu/wiki/Blocks)
        i. Motion - Blocks that move sprites around the screen
        ii. Looks - Blocks that change how sprites or backgrounds look

iii. Sound - Blocks that will play sounds and adjust volume
iv. Pen - Blocks that will draw on the screen using sprites as the pen
v. Data - Blocks for storing data into variables and lists
vi. Events - Blocks for starting and stopping the program
vii. Control - Blocks for altering the flow of the program
viii. Sensing - Blocks for learning the state of the program and getting input
ix. Operators - Blocks for mathematics, boolean, and string operations
x. More Blocks - Create your own blocks to simplify your programs

d. Block Shapes (http://wiki.scratch.mit.edu/wiki/Blocks)
   i. "Hat" blocks - they start the program based on the condition specified (ex: When green flag clicked)
   ii. "C" blocks - they have code inside them that runs at specific times based on the block (if, repeat, forever, etc.)
   iii. "Hat" blocks - Stop All, Forever, Delete this Clone
   iv. "Stack" blocks - Standard blocks
   v. "Reporter" blocks - these blocks report values such as numbers, strings, etc. (rounded edges)
   vi. "Boolean" blocks - these blocks are for representing boolean values (angled edges)
e. Menus
   i. File > Downloading to your computer
   ii. File > Upload from your computer
   iii. Edit > Undelete
f. Ask for questions / Give students some time (5-10 minutes) to play around and discover how it works on their own.

7. [**Getting Started with Scratch**]
   a. Introduce various blocks (below are some suggestions)
      i. Events - When green flag clicked
      ii. Motion - all
      iii. Looks - say / think, next costume
      iv. Sound - play sound
      v. Pen - clear, pen down, pen up
      vi. Data - (none at this time)
      vii. Control - repeat, if
      viii. Sensing - touching, ask / answer, timer / reset timer, key pressed
      ix. Operators - basic math (+ - * /)
      x. More Blocks - (none at this time)
   b. Work with students to build a simple program
      i. Start by gliding a sprite across the screen with the motion blocks. Have it bounce if it hits an edge
      ii. Have the sprite play a sound when it hits an edge
      iii. Create a second sprite, then have it say something when it touches the

main sprite
   iv. Allow students ample time to experiment and learn on their own
   v. See **Activity1_Explore.sb2** for an example
c. Students should learn:
   i. Simple motion blocks
   ii. Working with sprites and costumes
   iii. Working within the Scratch environment

8. [**Drawing in Scratch**] See the lesson plans in the "Drawing in Scratch" folder from Nathan Bean: http://www.nathanhbean.com/scratch/ScratchCurriculum/Geometry.zip
   a. Students should learn:
      i. Using the Pen blocks
      ii. Using iteration (repetition) to repeat actions
      iii. Getting user input
      iv. Simple mathematics in Scratch
   b. See the **Activity2_*.sb2** files for examples of partially completed projects

9. [**Spirographs**] Now that we can draw in Scratch, let's play around with something even more interesting: spirographs.
   a. Show the slides to help explain what a spirograph is. It includes some examples of the math that goes into making a Spirograph work, which ends with the formula for calculating the position of the pencil at any given time.
   b. Have the students open the Scratch Spirograph program: http://scratch.mit.edu/projects/21326308/
   c. Let students experiment by adjusting the value of the two variables. For best results, those values should be "relatively prime" which means that they should not share any common factors. Prime numbers are a good choice to use. Let students share good values that they find with the class.
   d. **Bonus**: Show students how to modify the variables by hand. Encourage them to try different values outside the normal range (less than 0, greater than 1, etc.)
   e. **Discussion**: Where would this be important in the real world?
      i. Artificial intelligence for games - randomly move within a set area
      ii. Rotational motion - think of the wheels of a steam locomotive
      iii. Modeling how gears interlock and move together
   f. Students should learn:
      i. Opening already created projects
      ii. Working with existing variables
      iii. Experimenting within a Scratch program

10. [**Reflections**]
   a. What did we learn today?
   b. What can we do with this new knowledge?
   c. What do we want to learn next?

d. Any other questions?

NOTE: I included a bunch of slides on programming. They are good information if you want to show how Scratch relates to real programming, as well as what real programming looks like. The notes for those slides are below:

1. [**Source Code**] Let's say, for example, that we want to write a program that will take an input from the user and print the result of the number divided by 61.

2. [**Scratch**] To do that in Scratch, we would use the following. [slide] As you can see, this program is written in a language that very closely resembles the English language that you and I use every day. That makes it really easy for a human to understand. Do you think a computer is able to easily understand this language?
    a. **Discuss. Ask them why or why not.**

3. [**Language Hierarchy**] In fact, Scratch as a programming language is very hard for a computer to understand directly. Therefore, it is called a High Level language. Let's look at some other examples of High Level languages.

4. [**C/C++**] *I usually note how the main function is like a "hat" block in Scratch*

5. [**Java**] *I usually note how the "class" in Java is like a sprite in Scratch; it can have multiple functions of "hat" blocks*

6. [**C#**]

7. [**Python**] *I used python in a "terminal" style here just to show how it can work without the framework code the other languages require.*

8. [**Other Languages**] There are many other high level languages out there. Some of these you may work with in the future, but many of them you may not need at all unless you are in a specialized field.

9. [**Language Hierarchy**] The next step in the path of most programs is to convert it to Assembly language. This is a language that is still readable by humans, but it is much closer to the language a computer actually understands.

10. [**Compiler**] To go from a high level language to assembly language, we use a program called a "compiler." Its entire purpose is to translate what we wrote in a high level language and turn it into assembly language.

11. [**Assembly Language**] This is an example of what assembly language looks like. This particular part is simply taking a number and dividing it by 61. Looks quite a bit more

complex, doesn't it?

12. [**Assembly Language**] Here's another example from the C program I showed earlier. This is the section of code that is getting input from the user.

13. [**Language Hierarchy**] Once we have the assembly language, the next step is to convert it to machine language. This is the actual "code" that computers can read and use.

14. [**Assembler**] To do that, we use another program, called an assembler, to convert our existing assembly language code into machine language code

15. [**Machine Language**] This is an example of Machine Language code, written in a way that it is somewhat approachable by humans.

16. [**Machine Language**] This is the real machine language code. It is simply a set of binary code (1s and 0s) that tell the computer exactly what to do.

Mission to Mars - USD 383 Summer STEM - Day 2

Summer 2017

Learning Objectives

- Students will explore different methods for sorting data.
- Students will see how different sorting algorithms perform differently
- Students will learn how to use a computer program to simulate a real-world scenario

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/day2.html
- Scratch Website: http://scratch.mit.edu
- Sorting Networks: http://csunplugged.org/sorting-networks/
- Potatoes on Mars Simulator: https://scratch.mit.edu/projects/112633885
- Hydrazine on Wikipedia: https://en.wikipedia.org/wiki/Hydrazine#Rocket_fuel
- Description of Chemical Reaction: https://www.quora.com/How-did-Mark-Watney-produce-water-by-burning-Hydrazine-in-a-closed-chamber-in-Mars

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Create Sorting Network on the floor using tape
- Make sure **Potatoes on Mars** is available on Scratch website.

Schedule

- 9:00 - Welcome & Icebreaker
- 9:15 - Sorting Networks & Sorting Cards
- 9:45 - Sorting in Scratch
- 10:15 - Break
- 10:20 - Martian Video & Basic Chemistry Intro
- 10:30 - Creating Water Simulator
- 11:00 - Wrap-Up Discussion

Lecture Notes

1. [**Icebreaker**] Take a minute to review what was learned yesterday. Some good questions:
    a. How do we move a sprite around the stage? *Motion blocks, Move ___ Steps, etc.*
    b. What block can we use to make the sprite turn around if it hits a wall? *If on edge, bounce*
    c. What block can we use to get the sprite to do the same thing over and over again? *Repeat or Forever*
    d. What block can we use to see if one sprite touches another sprite? *Sensing*

blocks, Touching _____

e. What block can we use to change what a sprite is doing depending on the output of another block? *If block or If - Else block*

2. [**Sorting**] Today, we are going to learn about how computers can sort data into a certain order. Let's talk about that:
    a. What are some different orderings of data you can think of? *Alphabetical, Numerical, smaller to larger, by height, weight, size, color, etc. Basically, get them thinking about how to categorize and sort data*
    b. Why would it be important for a computer to sort data? *Think about a dictionary; what if all the words were in a random order? Or a phone book?*

3. [**Sorting Network**] Let's take a look at one way that you can sort data, using a sorting network. <gesture to the sorting network on the floor> This is a special design that will guarantee any data at one end will be sorted by the time you reach the other end. Think it will work? **<get feedback>** Let's give it a try!
    a. Assign some numerical value to the students and put 6 of them at the start end of the network. Height works well, or the number of letters in their name. Try to avoid duplicates at first, but later you can deal with those as well.
    b. Have the students follow their lines. At any place where the lines intersect, the student with the higher value will go one direction, and the lower value will go the other. It helps to establish some consistent direction beforehand. I've always used "higher goes right."
    c. At the end, check to see if the students are sorted. If they aren't, see if you can start over and diagnose the problem. Some students like to go faster than others and miss the point. You can challenge the students to create groups and see which group can do it the fastest but correctly.
    d. Lots of good resources are available here: http://csunplugged.org/sorting-networks/
    e. See **SortingNetwork.mp4** in the Google Drive folder for an example.

<<<These next slides come from my CIS 115 lectures on Algorithms and can be adapted to fit the audience as needed. I usually don't get all the way through it, but I at least like to do the first two algorithms: insertion sort and bubble sort and the quick discussion that they take the same amount of time. Beyond that, it is really difficult to relate to younger students. >>>

4. [**Sorting Cards**] Now that we've explored how to sort using a sorting network, let's look at ways a computer will sort data.
    a. **<ACTIVITY>** This is very similar to the PB&J activity http://www.mathcs.emory.edu/~valerie/courses/fall13/170/resources/pbj-algorithm.pdf
        i. I need a volunteer that thinks he or she is very good at bossing people

around.

ii. I also need a volunteer that is really good at following directions

iii. Have the first person give the steps to shuffle a deck of cards without looking at what the other person is doing. Encourage the second person to be as “literal” as possible; aka - offer a knife to help “cut” the cards, etc. Hopefully that person will not be specific enough and they’ll end up with a mess.

iv. Maybe have them try again, but this time watch what each other does.

At the end, I usually relate the story of the PB&J activity so they understand what the intent was

5. [**How Shuffle**] As you can see, there is more to this than simply giving someone the steps of the process. They need to have the right ingredients, the right tools, the right skills, and the right prior knowledge before they can proceed.

6. [**al-Khwarizmi**] The origin of the word Algorithm comes from this man, Abu Abdallah Muhammad ibn Musa al-Khwarizmi (al - khwarithmi). In the 9th century AD, he wrote many important books covering the solutions to linear and quadratic equations. His solutions took the form of a series of steps, and over time, the word “algorithm,” based on his name, became the term we use to describe such a series.

7. **[al-Khwarizmi Video**] Here is a quick video from Hank Green on the Science Show about the impact of al-Khwarizmi on the world of mathematics.

   <play video; start at 0:00, stop at 0:35 when the show intro begins> *Encourage students to finish video later*

8. [**Algorithm**] In general terms, an algorithm is simply a finite list of specific instructions for carrying out a procedure or solving a problem. Let’s spend some time looking at examples of algorithms.

9. [**Euclid**] ***<<<This part can be skipped if needed; depends on the age of the students>>>*** One of the oldest algorithms still used today is called Euclid’s Algorithm. As you might know, Euclid was greek mathematician from around 300 BC. One of the things he discovered was a simple and easy way to calculate the greatest common divisor of two numbers. If you remember from algebra, this is needed to help reduce fractions.

10. [**Euclid’s Algorithm**] His algorithm is as follows

    a. Go over slide

11. [**Euclid Example**] **<Slide has 2 positions>** Let’s do an example and find the GCD of 1071 and 462.

a. Go through the example
    i. 1071, 462
    ii. 609, 462
    iii. 147, 462
    iv. 147, 315
    v. 147, 168
    vi. 147, 21
    vii. 126, 21
    viii. 105, 21
    ix. 84, 21
    x. 63, 21
    xi. 42, 21
    xii. 21, 21
    xiii. 21, 0

12. [**Euclid Example**] Here is the full process

13. [**Sorting Algorithms**] To help us really understand algorithms, let's try a couple out ourselves. We're going to look at some different sorting algorithms.

    a. <optional> First, everyone stand up and move to a different table. Try to sit at a table that doesn't have any of your other teammates. We'd like to be in groups of no more than four.
    b. Each table needs a deck of cards.
    c. First, split the deck into the 4 suits, and each person needs a single suit.
    d. Shuffle the suit so that it is random.

14. [**Sorting**] So, to begin, sort the decks of cards. As you do so, try to think of what the steps are that you follow and how you would describe that to others.

    a. Discuss: How did you do it? (Try to link back to common algorithms such as selection or insertion sort)

15. [**Insertion Sort**] First, let's implement insertion sort.

    a. Follow Slide
    b. Have students keep track of the number of times they have to ask "does this card go before this card?" starting from the beginning each time. (A crude measure of running time)
    c. Record their results on the board and average them for later
    d. As you can see, this is very similar to the way that most humans would sort something.

16. [**Bubble Sort**]

a. *Now, lets try bubble sort.*
b. *As you are doing bubble sort, have students count the number of times that they swap one card for another (not how many times you compare two cards).*
c. *Have the students record their counts on the board.*

17. [**Big O Notation**] So, now that we've played with a couple of algorithms, let's talk about how we can decide which one to use. Can we tell from our data we've recorded which one is faster? **<discuss & speculate>** Obviously we need a better way to do this. The answer lies within Big O (notation)! (No, I'm not talking about the giant robot anime from the early 2000's)

18. [**Big O Notation**] Big O notation is simply a way to express the complexity of an algorithm. It approximates the number of steps needed to complete the algorithm based on the size of the input. Finally, Big O notation assumes that the input is "worst case" for that particular algorithm.

19. [**Worst case**] So, what would you say is the worst case input for Bubble Sort?

    a. **Discuss the options**

20. [**Worst case**] As you can see, the worst case input for Bubble sort is a list that is sorted in reverse order. In the case of a suit of cards, it takes 78 swaps to get it back to the sorted order.

21. [**Graph**] If we graph the number of swaps against the number of cards, we get the following graph. What kind of a function does this look like?

    a. **Discuss. Hopefully should get x^2 or quadratic or something similar**

22. [**Sorting Algorithms**] As you can see, the complexity of Insertion Sort and Bubble Sort is in Big O of n^2. So, what about some algorithms that are faster?

23. [**Merge Sort**]

    a. *Combine the deck back together and shuffle it*
    b. *Do merge sort at the table (split into smaller and smaller piles)*
    c. *After completing, try to quickly walk them through the calculation that leads to Big O of n lg n. I usually draw a tree of piles starting with 13 at the top, then count the layers of the tree (should be lg n) and then the max number of swaps (n/2) and then they layers up again (should be lg n).*

24. [**Quicksort**]

    a. *Do the same with Quicksort*
    b. *Again, try to explain the running time of Quicksort by showing the worst-case example (already sorted) and describe how it can't be calculated absolutely but in*

practice it is very fast

25. [**Sorting Algorithms**] As you hopefully can tell, these algorithms are much quicker to implement, but they are much more complex. They run in Big O of n lg n time.

<<<Sorting Activity in Scratch>>>

26. [**Sorting in Scratch**] Now that we know how bubble sort works, let's see if we can try to write it in Scratch!
    a. Have the students create a list of numbers in Scratch. At the start of the program it should clear the list. Then, instead of assigning random numbers which would change each time, have them manually add 8 - 10 numbers in an unsorted order to the list.
    b. Discuss how bubble sort works. Hopefully they should realize that the first step is to compare the first number and the second number in the list. Show them how to access those numbers and use an If block to test for that situation.
    c. Talk about what they need to do to swap those numbers. I usually use 2 different items held in my hand. They should realize that they'll need a "3rd hand" to make it work. That leads to the idea of creating a Temp variable to store that item. Show them how to do the 3 step swap operation.
    d. Once they have that, talk about the next step. It should be to compare items 2 and 3. Show them how to duplicate the block of code you've created to handle that (tell them to watch and not follow along, since you'll undo it later), and discuss how they would do the next steps (3 and 4, 4 and 5, etc.). They should remark that it is very inefficient. Talk about how to use a repeat block to repeat steps. It should do one fewer comparison than there are number of items. Help them understand by looking at their hands. There are one fewer gaps (pairs) than there are fingers.
    e. Now that we are repeating steps, we need to change the two items it is looking at each time. Discuss how they could do that. Lead them towards using a variable to keep track of the current position, and show how to include that. They will also have to update it after each repeat.
    f. Finally, discuss how bubble sort repeats all of those steps from the beginning each time until it is sorted. Add one more forever loop and a block to reset the position to 1 each time.
    g. For more information, watch the **BubbleSort.mp4** video in the Google Drive for a demonstration from 2016.
    h. There is also a solution file included, named **BubbleSortSoln.sb2** from that demo.

27. [**Martian Video**] Sorting is just one way that computer programming can help us deal with the real world. We can also use it to simulate what would happen in a particular situation. Let's check out this video from *The Martian* to see what astronaut Mark Watney is dealing with. <**watch video**>

28. [**Chemistry**] So, Mark Watney needs to figure out how to grow food on the surface of Mars. Thankfully, in the food for the mission were actual, real potatoes. Most of the food was freeze-dried, but since they would be on Mars over Thanksgiving, they sent real potatoes so they could have a little treat. So, we have access to plants that can grow. What else do we need to grow plants? <**discuss - Water, sunlight, soil, nutrients, etc.**>.

    He could make soil by mixing the dirt on Mars with actual human waste (just like we use animal waste as natural fertilizer on farms), and there was plenty of sunlight available. The one thing he was missing was water. Does anyone know what water is made of? <**discuss - Hydrogen & Oxygen**>.

    Humans have to breathe oxygen, so he had a way to get a continuous supply of that. The only thing he was missing was hydrogen. Thankfully, hydrogen is a major component of almost all fuels, including rocket fuel like Hydrazine <**see slide**>. So, all he needs to do is break down the Hydrazine into hydrogen and nitrogen, then mix it with oxygen using fire to create useable water. It was a dangerous idea (and he about blew himself up at least once), but thankfully we can simulate it in a computer program on Scratch.

29. [**Potatoes on Mars**]
    a. Direct the students to load the Potatoes on Mars scratch project: https://scratch.mit.edu/projects/112633885
    b. The students should click the “See Inside” button to see the code. If they have a Scratch account, they can also click the “Remix” button to save it to their account.
    c. For each of the major sprites, we need to simulate what it will do. There are variables created for many different things, including the hydrogen, nitrogen, oxygen and water available, as well as the number of plants grown and the number of calories available.
        i. Hydrazine - Forever change N2H4 by 1, wait some amount of time, usually 1 second to start (can be adjusted later)
        ii. O2 Reclaimer - Forever change O by 1 if it is less than a certain value, then wait some time before checking again (can be adjusted later). 18% - 22% is the optimal range for real life, so I usually have it add more oxygen if the value is less than 20.
        iii. Ionizer - Forever if N2H4 is greater than 0, reduce it by 1 and increase N by 2 and H by 4 (using the chemical formula). It doesn’t have to wait, as it can do this anytime.
        iv. Flame - Forever if H is greater than 2 and O is greater than 18 (to keep a safe value), reduce each by the appropriate amount and release 1 water into the air.
        v. Once the water is released, the plants will start growing and multiplying.

As they grow, they release more oxygen and the simulation will run faster. However, if at anytime the amount of hydrogen in the air exceeds 100, the simulation will explode, just like it would in real life if there is too much hydrogen in the atmosphere.

vi. Students will have to adjust the wait times in the Hydrazine sprite to optimize the simulation. They can also make the O2 reclaimer work faster. Encourage them to see how fast they can make the simulation run without it blowing up.

vii. See **MarsPotatoes.mp4** in Google Drive for an example from 2016.

viii. See **PotatoesonMars.sb2** for a starter example (it will still explode and needs adjustment).

30. [**Reflections**]
    a. Why do computers want to sort data?
    b. What are some ways we can sort information?
    c. Can we use computers to simulate real world situations such as a chemical reaction?
    d. What else could we simulate with computers? What have you observed computers doing in the real world?

- 112633885

Mission to Mars - USD 383 Summer STEM - Day 3

Summer 2017

Learning Objectives

- Students will explore how a computer represents data in binary
- Students will learn how to convert simple numbers between binary and decimal formats
- Students will see how data encoding can be used to transmit data with very few data points or code words
- Students will see how a data transmission error can cause problems and will discuss ways to fix the problem.

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/day3.html
- Scratch Website: http://scratch.mit.edu
- Binary Numbers on Wikipedia: https://en.wikipedia.org/wiki/Binary_number
- Binary Numbers on Math is Fun: https://www.mathsisfun.com/binary-number-system.html
- Binary Numbers on CS Unplugged: http://csunplugged.org/binary-numbers/
- Binary Flash Cards: http://cse4k12.org/cards/number_cards.html
- Binary Worksheets: http://cse4k12.org/binary/counting_in_binary.html
- Rover Hex Code: https://scratch.mit.edu/projects/112910972/

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Print Binary Flash Cards & Binary Worksheets for students
- Make sure **Rover Hex Code** is available on Scratch website.

Schedule

- 9:00 - Welcome & Icebreaker
- 9:15 - Binary Numbers & Hexadecimal - Flash Cards & Worksheet
- 9:45 - Text, Images, etc. as Binary Data (Data Encoding in ASCII, BMP)
- 9:55 - Break
- 10:00 - Mars Message Decoder
- 11:00 - Wrap-Up Discussion

Lecture Notes

Find someone who:
1: Can draw a square using repeat blocks in Scratch
2: Can make and name a variable in Scratch
3: Can create a list in Scratch
4: Can show the block that starts the program
5: Can show you the blocks to use when computing data
6: Can show you his/her age in binary code
7: Can make a Sprite bounce on the edge.

1. [**Icebreaker**] Take a minute to review what was learned yesterday. Some good questions:
    a. Why do computers want to sort data? Why is that useful?
    b. What are some different ways we learned to sort data yesterday?
    c. What are some examples of real-world things could computers be used to simulate?

2. [**Binary Numbers**] Today we are going to learn about how computers store and manipulate data. Does anyone know how it works? <**discuss - hopefully coming to the idea of binary**>.

<<<These slides are from my CIS 115 lecture. They can be adapted to fit the audience as needed>>>

3. [**Stibitz**] The first person to really use binary in a computer was George Stibitz. In 1937, he completed his “Model K” calculator named for the “Kitchen Table” where he worked on it. It was capable of performing addition on two binary numbers.

4. [**Complex Numerical Calculator**] After receiving a full research grant from Bell labs, he was able to compete his Complex Numerical Calculator in 1940, which was able to perform calculations on complex numbers. It was also unique because it was able to perform those calculations remotely. It was attached to a phone line, and when it was demonstrated at Dartmouth College in New Hampshire he used a teletype machine to send commands to the machine while it was in New York. This was the first example of a machine ever used remotely over a telephone line. (We’ll talk more about this when we get to the history of the Internet.)

5. [**Stibitz Video**] Let’s hear the story from George Stibitz himself

    <play video; start at 1:14, stop at 4:20, right after the “Divide one by zero”

anecdote.>

6. [**Binary - Natural Numbers**] **<Slide has many stops; each bullet is a stop>** So, let's take a look at how the math works for binary numbers. As you may already know, binary numbers are simply a way to represent numbers using the powers of 2. Here is an example of an 8 bit binary number. Can anyone tell me what number this is in base 10?

7. [**Binary - Natural Numbers**] As you can see, for each space that has a one you add that power of 2 to the answer, so, in this case we have a one in the 32 position, the 8 position, and the 2 position.

8. [**Binary - Natural Numbers**] Putting it all together, we have 32 + 8 + 2 = 42.

9. [**Binary Flashcards & Worksheet**]
    a. Gather the students together on the floor or somewhere easy to work. Give each student one set of the binary flash cards from http://cse4k12.org/cards/number_cards.html. This set of activities is useful: http://csunplugged.org/binary-numbers/
    b. Have them lay down the cards in order with the most dots on the left and the least on the right. Then, to make a binary number, add up the dots on the face-up cards to find the value. Face up cards are 1, face down cards are 0. Start with small numbers first, then go up. Usually works best to first start by turning cards up and seeing if they can add the dots, then go the other way by giving them a number and letting them turn over cards to represent it.
    c. Especially focus on numbers less than powers of 2, such as 15, 31, 63. See if they can find the pattern.
    d. On the board, briefly show binary addition. Many of the same arithmetic rules apply to binary.
    e. You can also discuss place values and compare it to the decimal system.
    f. Take a minute to have students fill out the binary counting worksheet, at least to the value of 16. More is better!
    g. Once they have a grasp on binary, introduce hexadecimal. The worksheet makes it easy. Just write "A - F" next to "10 - 15" on the sheet. Explain that it is just another way to refer to the same values, but with only one letter/digit instead of many. There is also a slide to introduce the topic.

<<<More slides from CIS 115 - feel free to adapt or skip parts based on time and audience>>>

10. [**Text**] But what about other data, like words and sentences. For those items, we use a couple of different formats. The first is ASCII or "as-key", which stands for the American Standard Code for Information Interchange. This is a table of all the values in the ASCII code.

11. [**ASCII Example**] Here is a sample of information encoded into ASCII. It is really hard to understand. Let's see if we can decipher it.

    <do an example of deciphering the text a bit>

12. [**ASCII Example**] As you can see, this example says "forty two" in ASCII. There are other formats for this, most notably Unicode, which is used extensively on the internet. It has codes for a much wider variety of symbols and languages.

13. [**Images**] Another common data type that we run into everyday is images. How do you think images are encoded for computers? **<discuss>** There are really two common ways:

    a. Bitmap - each pixel in the image is assigned a numerical value representing the color of the image. This is simple to do, but doesn't work well if you want to magnify the image very far.

    b. Vector - each element of the image is defined via a series of mathematical vectors. This is much more complex (usually must be done by hand), but the images can be magnified indefinitely.

14. [**Vector**] In fact, here is an example of the code within a Vector graphics file. As you can see, it clearly defines in mathematical terms the exact shape that should be drawn to the screen, allowing it to be infinitely scaled in any direction just by applying simple mathematical operations. Most 3D graphics are defined in a very similar manner, but we'll talk about those in a later lecture.

15. [**Bitmap**] Let's look at a bitmap example. This is a 16x16 sprite that could have been used in an early video game. It consists of only four different colors. So, how do you think that would get encoded?

    a. **Discuss! See what they think**

16. [**RGB Colors**] As you may know, bitmap images are comprised of individual pixels, with each pixel giving the color of just a single dot on the picture. To store that data, a computer would simply store the color value for each pixel in an array. The color values are stored using their corresponding red, green and blue values.

17. [**Bitmap Text**] Therefore, this is an example of what a bitmap would look like stored on your computer. It looks like just a bunch of random text, but there is a method to the madness.

18. [**Bitmap Text**] Here is what that text would look like if I actually filled in each area with the corresponding color.

19. [**Bitmap Text with Key**] In many cases, instead of storing the actual color values, the

bitmap might use a color table to store the individual colors, and then use the indexes of the color table to represent the colors in the file. In this case, the file can be stored in less than 4% of the size it would normally take.

20. [**Data Compression**] **<Slide has multiple stops>** The same concept can be used for Text. Here we see a very common piece of text with repeated words. We can easily replace these words with numbers that represent those words instead, resulting in this much short version.

21. [**Mars Rover Hex Code**] Why would this be useful? Well, Mark Watney found out when he was stranded on Mars with no way to communicate with Earth. The only thing that the folks on Earth could do was rotate the camera on top of the old Mars Pathfinder rover. So, knowing that he needed a way to understand, Mark set up 16 signs, one for each hexadecimal character. Since he knew that 2 hexadecimal characters would make one text letter in ASCII, it was a really efficient way to communicate. For our activity today, we're going to build a program that decodes that information just like he did!
    a. Direct the students to load the Rover Hex Code scratch project: https://scratch.mit.edu/projects/112910972
    b. The students should click the "See Inside" button to see the code. If they have a Scratch account, they can also click the "Remix" button to save it to their account.
    c. Let the students observe the program running once to see what it does. It looks like the rover's camera moves randomly, but there is an underlying pattern. Show them how to change the stage backdrop to the circle and observe it again.
    d. Explain that this program is very much a "code and test" programming project, where we try little bits at a time and then run it to see what it does and test it. That helps us avoid little mistakes that become big mistakes later.
    e. Basic steps:
        i. Under the "When I receive Letter" block, first have the program say the direction of the camera. That helps us understand what the camera direction is saying as it rotates.
        ii. Since the camera direction is in degrees, we'll need to convert it into sixteenths. To do that, we divide it by 22.5 (which is 360/16). That will let us know which sixteenth it is pointing to. However, since the direction is negative, we'll get negative sixteenths as well. So, we'll add 8 to the result. We then store that value in the Code variable.
        iii. Next, we need to keep track of 2 consecutive code words. We can do that using the Last variable. So, we'll need a couple of If-Else blocks to check if the Last variable is blank. If it is, we'll put Code into it and get another Code variable from the next movement of the camera. If it isn't blank, we'll use the Code and Last variables to decode the letter and add it to output.
        iv. To decode the letters, refer to the ASCII table here: http://mediawiki.factotumnw.com/mediawiki/images/a/a2/ASCIITable1.jpg. The Last variable is the first hex character, and the Code variable is the

last. All of the characters are in lowercase (and uses tilde for space), so you really only need to deal with Last = 6 and Last = 7. So, you'll have a block of code looking like this:

```
If Last = 6
    If Code = 0
        Set Output to (join Output and ` )
    Else
        If Code = 1
            Set Output to (join Output and a )
    …
Else
    If Last = 7
        If Code = 0
            Set Output to (join Output and p )
    …
Set Last to (blank)
```

v. ***This part takes time***, and students have to be really careful about their code. If they get stuck, look closely at the structure to make sure they didn't get off somewhere. I usually do the "If Last = 6" part first, and show them how to duplicate the If-Else blocks 16 times, then once that is done I show them how to duplicate the whole thing for "If Last = 7" and then they just change the output.

vi. Once it is done, it should give the students the message of "move~eest~mark." Discuss how this message could be very confusing (should it be east or west). How can we fix that? Discuss ways of detecting and correcting errors in the message. If the students press "r" in Scratch, it will replay the message again, this time with the correct direction.

f. See the **MarsHexCode.mp4** video for a demonstration of how to complete this project.

g. Also check the **RoverHexCode.sb2** for a sample solution.

22. [**Reflections**]

a. How do computers represent data?

b. What is the value 42 in binary?

c. What is the value 101101 in decimal?

d. What is hexadecimal?

e. What could happen if a computer message is not correct when it is received? How can we fix that?

Mission to Mars - USD 383 Summer STEM - Day 4

Summer 2017

Learning Objectives

- Students will discuss artificial intelligence and how to determine if a computer is intelligent.
- Students will experience how a neural network is built using training data.
- Students will build a working video game AI based on PacMan.
- Students will use their knowledge of AI to write a program to avoid random obstacles.

Resources

- Slides: http://people.cs.ksu.edu/~russfeld/presentations/stem2017/day4.html
- Scratch Website: http://scratch.mit.edu
- Information on Turing Test: http://csunplugged.org/the-turing-test/
- Pacman AI Behavior: http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior
- Pacman on Scratch: https://scratch.mit.edu/projects/65404312/
- Pacman Solution on Scratch: https://scratch.mit.edu/projects/90692358/
- Mars Pathfinding on Scratch: https://scratch.mit.edu/projects/113105354/

Lesson Setup Before Class

- Log on to computers using STEM accounts
- Print Cat & Dog handouts
- Make sure **PacMan Starter** is available on Scratch website.
- Make sure **Mars Pathfinding** is available on Scratch website.

Schedule

- 9:00 - Welcome & Icebreaker
- 9:15 - Intro to AI & Decision Making
- 9:30 - Cat or Dog activity
- 9:45 - PacMan AI
- 10:10 - Break
- 10:15 - Mars Rover Pathfinding
- 10:45 -STEM Survey
- 11:00 - Wrap-Up Discussion

Lecture Notes

1. [**Icebreaker**] Take a minute to review what was learned yesterday. Some good questions:
    a. Convert some numbers back and forth between binary
    b. What other numbering system besides binary did we learn? How does it work?

    c. What could happen if a message gets jumbled up when we send it? How can we fix that?

2. [**Video**] Today we are going to talk all about artificial intelligence. Before we begin, here's a short video from PBS introducing the subject with some of the world's renowned experts in the field. (Watch through 2:05, first section)

3. [**Video 2**] Here's another video about a robot that is programmed to learn just like a human does. Do you think that would be a good way to build an artificial intelligence?

4. [**What is Artificial Intelligence**] So, what is artificial intelligence?
    a. Discuss. A good definition is "the study and design of intelligent agents."

5. [**Intelligent Behavior Diagram**] One problem with AI is that it forces the computer to mimic all human behavior, not just intelligence. Things such as slow response times, typos, and commonly held misconceptions that aren't true are all examples of human behavior that might not be considered intelligent, but it is still something a successful AI agent must possess to pass the test. Likewise, it must also act as if it cannot solve some problems that are perfectly within its abilities, simply because those problems are unsolvable by a human's intelligence.

6. [**Alan Turing**] That was the problem that Alan Turing ran into. He was deeply interested in the field, but unfortunately there was no good way to determine if a system was truly "intelligent" at the time. In 1950, he wrote a paper called "Computing Machinery and Intelligence" which he opened with the following statement: "I propose to consider the question, 'Can machines think?'" In that paper, he describes one way to test a machine's intelligence, which is now known as a Turing Test

7. [**Turing Test**] The basic idea of a turing test is as follows: you have a person in a room with a computer capable of text-based chat. In another room is either a computer or a human that responds via the text-based chat system. The computer will try to pass itself off as a human being, by responding to the prompts from the tester. The tester then must determine if he or she was conversing with a computer or a real person. So far in history, no computer has completely passed the Turing test, but many have come very close at times.
    a. So, now that we've seen the turing test, can anyone think of some problems it may have?

8. [**Chinese Room**] Another problem with the Turing Test is highlighted in the Chinese Room thought experiment, as proposed by John Searle in 1980 in his paper "Minds, Brains, and Programs." In this setup, an English speaking person is placed in a room with sufficient supplies and a set of instructions completely written in English that directs him to accept Chinese language characters as input, and output a response of Chinese

characters. On the other side of the wall is a native Chinese speaker performing a "Turing Test," and in this case that person is convinced that the person on the other side of the wall is indeed a human. However, the computer, being a human that only speaks English, is blissfully unaware of the conversation taking place in Chinese. So, is the machine "intelligent" or merely so advanced at following instructions as to appear "intelligent?"

9. [**Strong AI vs. Weak AI**] This leads to the endless debate between Strong AI and Weak AI. When you think of AI in movies, this is usually "Strong AI" which is designed to completely mimic or surpass human intelligence. Unfortunately, at this time Strong AI is not a reality, and some debate that it is even possible. Most of the AI that we deal with today is a form of Weak AI, also called Narrow AI, which is designed to perform only a subset of intelligent actions.

10. [**Marvin Minsky**] To dive a bit deeper into the topic of AI, we're going to look at a very unique tool called Neural Networks. In 1969, Marvin Minsky, one of the founders of MIT's AI lab, wrote a book called *Perceptrons* that laid the groundwork for the idea of neural networks.

11. [**Artificial Neural Networks**] The idea behind neural networks lies in the power of individual "neurons" and the connections between them. Each neuron is capable of doing a certain task, and then its output is passed on to other neurons. The strength comes in the form of the connections between the neurons. If one tends to give correct answers to a problem, other neurons will be more likely to use its output based on the strength of the connection between them. The process of strengthening good connections and weakening bad ones is how a neural network is able to "learn" how to do things.

12. [**Neural Network Activity**]
    a. Each student gets one of the half-sheet handouts. They are directed to only look at their own sheet and not share.
    b. Around the room, post numbers 0 - 10
    c. For each picture, ask the class to vote whether the overall picture is of a Cat or a Dog. I usually have them close their eyes to prevent cheating. Also, remind them that it is OK to get this wrong (some students are very self-conscious about being wrong at this age).
    d. Use the Excel document to show the full picture.
    e. At the end, any students who got it right move up to the number that represents how many they've gotten correct so far. (So, if they have 3 correct, they stand by number 3). The students at the higher numbers get "more votes" or higher weight than the lower students.
    f. At the end, let the students vote one last time with their eyes open. Hopefully they should see that most of the students who get it right are at the higher

numbers. You can also use the Excel sheet to record the number each student is at based on the X-Y coordinates on the handout. Hopefully the higher numbers should be centrally located and the lower numbers are around the outside.

g. See **StudentTeach.mp4** for a short example of how this works.

13. [**Camouflaging Tanks**] Refer to story here: http://neil.fraser.name/writing/tank/
    a. Trained a neural network with pictures of tanks hiding in trees and pictures of just trees
    b. It worked well for all the original photos, but when they brought in a new set of photos, it was totally random
    c. The reason: the original photos had all the tanks taken on sunny days, and all the trees taken on cloudy days. They had built a machine to determine if it was sunny or not!

14. [**MarI/O**] Here's another use of Neural Networks - to play a video game. This video does a great job of explaining how neural networks can be evolved to complete any task.

15. [**Pacman AI**] Now that we know a little bit about AI, let's see if we can build one in Scratch. One of the simplest and yet most interesting AIs can be found in the old game Pacman. How many of you have played Pacman? There are 4 ghosts - Inky, Blinky, Pinky, and Clyde. Let's take a look at how to write their AI:
    a. Direct the students to load the **Pacman** starter scratch project: https://scratch.mit.edu/projects/65404312
    b. The students should click the "See Inside" button to see the code. If they have a Scratch account, they can also click the "Remix" button to save it to their account.
    c. The AIs in this project use a simple perceptron model. They first perceive their surroundings, then act based on that perception. To complete this project, follow these steps:
        i. **Blinky:** Blinky always moves toward Pacman. So, in the perceive stage, it must determine which direction to move. To do this, I usually draw a circle on the board and divide it into 4 quadrants with an X. Then, each line must be labeled, with 0 being up. So, the lines will be 45, 135, -135 and -45, going clockwise from 0. Label the quadrants up, right, left and down. In the code, we must point Blinky at the direction of Pacman, then analyze our direction variable to determine what quadrant to move to. It is easiest to go from least to greatest. So, you'll have something like:

```
If direction > -135 and direction < -45
        set Direction to Pacman to left
else
        If direction > -45 and direction < 45
                set Direction to Pacman to up
…
```

Once you have the direction set, the Act phase is simply to move in that

direction. The Move block accepts directions in lower case (left, right, up, down). See **Blinky.png** in the Google Drive for an example.

ii. **Pinky**: Pinky always moves to a point a few spaces in front of Pacman, which is represented by a sprite called “PinkyPoint.” It uses much the same algorithm as Blinky. In fact, it is simple to just drag the blocks attached to “perceive” in Blinky and drop them on Pinky, then attach them to the perceive block. (DO NOT drag the Purple “Define perceive” block along with it; that will break things). Then, just change the set blocks to use the “Direction to Point” variable instead, and have Pinky move toward that point. See **Pinky.png** in the Google Drive for an example.

iii. **Clyde**: Clyde is a bit different. He will move toward Pacman most of the time, but if he gets too close he will retreat toward a point in the lower left corner of the screen, represented by a sprite called “ClydePoint.” So, his code is a combination of both Blinky and Pinky. So, just drag and drop the two perceive sections (again, DO NOT include the purple “Define perceive” blocks). In the act section, just include an If block to check if Clyde is within a set distance of PacMan (I’ve used 100). If it is too close, move toward ClydePoint, else move toward PacMan. See **Clyde.png** for an example.

iv. **Inky:** The AI for Inky is much more difficult to code in Scratch. But, if students want to experiment, all they have to do is attach blocks below the “When green flag clicked” together to active it, then they can code their own Perceive and Act phases.

d. See the **Pac Man Solution** file on Scratch for a complete example: https://scratch.mit.edu/projects/90692358

16. [**Mars Rover Pathfinder**] For the last project, we need to get Mark Watney back home. To do that, he has to traverse the Martian terrain, but there are obstacles in his way. We are going to write a program to use AI to help him find a way around the obstacles.
    a. Direct the students to load the **Mars Pathfinding** starter scratch project: https://scratch.mit.edu/projects/113105354
    b. The students should click the “See Inside” button to see the code. If they have a Scratch account, they can also click the “Remix” button to save it to their account.
    c. For this project, there are 3 randomly generated circular obstacles between the rover (lower left) and the goal (upper right). Students can press the Space key to move the obstacles, but once they start they should not change the obstacles for a while. The rover must make it to the goal in as few steps as possible. Here are the basic steps to complete the project
        i. First, have the students simply point towards the goal and put the custom Move block into a Forever block, and see what happens. By default, it will move toward the goal, but it will stop once it hits a rock.
        ii. Once it hits an obstacle, we must figure out which one using some If blocks. See **Pathfinder1.png** for an example. At the same time we only

want to move if we aren't touching an obstacle, so it needs to go in the last Else block.

iii. In the case that we've hit an obstacle, we should find the direction to the center of the obstacle, then see which side would be shorter to go around. We can do that by comparing that direction with 45 degrees. If it is less than 45 degrees, we should turn right, else we should turn left. We can decide how many degrees to turn and how many steps to make in that direction before turning back toward the goal. I usually turn 90 degrees and move 100 steps before turning back. Students can adjust those numbers to make their program more efficient. See **Pathfinder2.png** for an example.

iv. That idea can be generalized for all 3 obstacles.

v. Finally, the students should add once more If block to detect if the goal has been reached and stop the program at that point.

vi. Once that is done, encourage students to modify their solutions by adjusting the angles and movements to reduce the number of moves (reported in the upper left corner).

d. See **MarsPathfinder.mp4** for an example of how to complete this program.

<<<**There are some videos at the end of the slides for time filler if needed**>>>

17. [AI Today] Some examples of AI today:
18. [Deep Blue] Deep Blue was a computer created by IBM that plays chess. In 1996, over 20 years ago, it beat the world's reigning chess champion, Garry Kasparaov. It was once of the greatest milestones in AI, and today computers are becoming better and better at playing even more difficult games such as poker and go.
19. [Watson on Jeopardy] In 2008, another IBM computer called Watson participated in several rounds of the Jeopardy! game show against two of the greatest champions of all time. Watson won quite heavily, but not without making a few mistakes such as the one shown here in the video.
20. [2 AI Chatbots Talking] Of course, AI still have a long ways to go before it can interact just like a human would. This video shows one of the most advanced AI chatbots, Cleverbot, talking to itself. As you can see, the conversation really doesn't get very far.

21. [**Reflections**]
    a. What did we learn about computer programming this week?
    b. What was the most interesting thing we did?
    c. Do you think you could do more with computers? What other things would you like to learn?