2016

A unit design : learning to code with Finches

William V. Gruman
University of Northern Iowa

A unit design : learning to code with Finches

Abstract

The purpose of this project was to develop a coding unit for students in grades 5-8 as an introductory programming course using the programming language SNAP! with a Finch robotics platform. The robotics platform provides a means of student engagement that allows them to complete computer programming challenges that drive interest and motivation. A series of five major learning activities were created as part of a middle school technology exploratory course. The beta testers included the four course instructors and around 30 of middle school student volunteers.

The designed unit was implemented once as a pilot, and was being implemented for the second time after making some modifications. The preliminary findings show that students enjoyed learning about basic programming concepts, or coding, and would be interested in pursuing similar types of projects and teachers had a positive experience. This model could be expanded to a full-term course as more modules are completed.

A Unit Design: Learning to Code with Finches

A Graduate Project Report

Submitted to the

Division of Instructional Technology

Department of Curriculum and Instruction

In Partial Fulfillment

Of the Requirements for the Degree

Master of Arts

UNIVERSITY OF NORTHERN IOWA

by

William V. Gruman

May, 2016

This Project Report by: William Gruman
Titled: Learning to Code with Finches

has been approved as meeting the research requirement for the

Degree of Master of Arts.

6/8/16
Date Approved

Ping Gao
Graduate Faculty Reader

6-8-16
Date Approved

Leigh E. Zeitz
Graduate Faculty Reader

6-8-16
Date Approved

Jill Uhlenberg
Head, Department of Curriculum and Instruction

Abstract

The purpose of this project was to develop a coding unit for students in grades 5-8 as an introductory programming course using the programming language SNAP! with a Finch robotics platform. The robotics platform provides a means of student engagement that allows them to complete computer programming challenges that drive interest and motivation. A series of five major learning activities were created as part of a middle school technology exploratory course. The beta testers included the four course instructors and around 30 of middle school student volunteers. The designed unit was implemented once as a pilot, and was being implemented for the second time after making some modifications. The preliminary findings show that students enjoyed learning about basic programming concepts, or coding, and would be interested in pursuing similar types of projects and teachers had a positive experience. This model could be expanded to a full-term course as more modules are completed.

Keywords: design a coding unit, Finch robotics, middle school, programming course

Table of Contents

A Unit Design: Learning to Code with Finches

Learning computer programming can be an ominous task for novice students. Computing skills are an important part of solving problems and can lead to many rewarding professions. So what prompts someone to get started with programming and keep his or her interest? What will keep the novice learner motivated to learn more? Initiatives like Hour of Code, which is promoted through the Code.org website by Hadi Partovi, are reflective of efforts to expose large numbers of students to computer programming. Their work shows that coding can be learned at just about any age.

When answering the question: "What takes the learner on to the next level?" Margolis, Estrella, Goode, Holme, and Nao (2010) explained that "too many young people are tragically and unconscionably stuck in the shallow end" (p. 8). Parallels are illustrated between swimming and computer science, particularly with racial inequality. In a larger picture, instructional design of novice computer science curriculum, particularly with computer programming, often engages early learners at the shallow end of the topic. Connections must be made to allow students to venture into more substantial content, providing enough learning to become become proficient, creative, and impactful.

Personal data breaches have unfortunately become a routine part of recent news cycles. It should provide motivation to encourage new, innovative professionals towards computer science professions. The ethics and legality of personal data security has also been shown recently in phones and computers used in connection with terroristic events. The skills necessary in either case begin with novice computer programming learners.

This coding unit was developed for students in grades 5-8, all in the same middle school. It was included as one module in a trimester-long technology exploratory course. The course modules were selected based on a number of different needs identified by the teaching staff. They included keyboarding skills, digital citizenship, basic computer skills, and coding. Students were grouped in the coursework by grade level. The same general content was taught to all of the students, although older students often moved through the coursework more quickly.

The activities in this module meet several Common Core literacy standards which are listed in Appendix A. There were several decisions to be made in constructing this project. They included selecting an instructional design model and identifying and implementing effective teaching strategies that complement the instructional design model. These are both explored in the review that follows.

Literature Review

The purpose of this review was to examine the instructional design essentials in a novice computer programming course. The scope was directed towards (but not limited to) middle school students. It could be expected that results of this review provides evidence for an effective template for designing this unit. Significant goals would include providing enough substance in the coursework to prepare students for more advanced work and also to provide enough support to maintain student motivation and interest.

This review focused on research-based peer-reviewed journals for source validity. It was important to identify studies completed that were directed towards novice programmers. This included middle school, high school, and undergraduate students. Most studies found used

undergraduate subjects. Data included students in the United States and Europe, although geography was not a factor in this review.

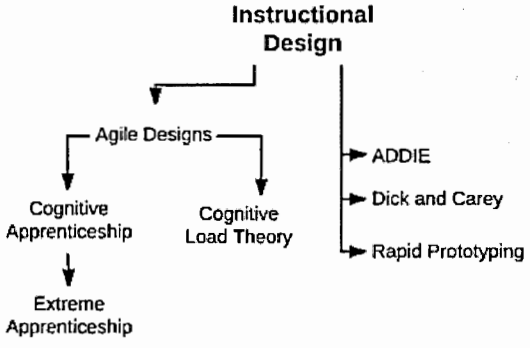
Instructional Design Forms

Various forms of instructional design are used in the development of introductory programming courses. Students build problem-solving skills and develop their interests as possible future computer scientists through this work. Additionally, these skills transfer to many other 21st Century careers, such as technology management and network administration.

Several common forms of instructional design apply to computer science coursework. The research studies in this review focused on Agile Instructional Design, Cognitive Load Theory, Cognitive Apprenticeship, and Extreme Apprenticeship. ADDIE, Dick and Carey, and Rapid Prototyping appear to be more conventional and traditional (Lembo, 2012) and were referenced in the literature, but not used. These are all considered Constructivist approaches, where students are an active part of learning. The following figure shows the relationship of the above instructional design forms.

Figure 1

Instructional Designs



Agile instructional designs. This is a category of design methods that feature students taking an active part in the learning process using higher-order thinking skills. The models can be thought of a sub-category of Constructivism design. Here, lectures are minimized and lessons include group applications, short activities, and use of feedback. Lessons are student-centered and use a problem-based learning environment. "The results include more motivated students and more satisfactory experience both for students and teachers." (Lembo, 2012, p. 9)

Lembo identifies the nine phases of Agile Design:

1) Curriculum Planning

2) Identify Learning Themes and Metaphors (Anchors)

3) Identify Learner Roles

4) Trawl for Learning Objectives, Modules and Competencies.

5) Identify Test Cases (Proof of Competencies)

6 & 7) Pair Programming (Development)

8) Unit Test (Automated Testing)

9) Release to Production (Refactor, Refactor, Refactor)

Iterate to Stage 4 within the curriculum plan. (Lembo, 2012, p. 10)

Cognitive load theory. This theory addresses the "implications of working memory limitations on instructional design" (Sweller, 1998, p. 252). In a course, students' tasks are geared more towards their long-term memory rather than short-term working memory. This can be accomplished by using open-ended projects. To reduce an overload on students, new content is delivered in small lessons to avoid overloading their working memory. Working memory is our short-term consciousness and is quite limited. "Recent instructional theories tend to focus on real-life tasks as the driving force for learning. Such tasks are typically associated with a very high cognitive load, which makes it more important than ever to take the limited human processing capacity into account" (Van Merrienboer, 2012, p. 20). Shaffer (2003, p. 4) indicated that Cognitive Load Theory "provides a model of how the mind processes information" and included two important components:

"1) Human working memory is limited

2) Two mechanisms to circumvent the limits of working memory are:

Schema acquisition, which allows us to chunk information into meaningful units, and automation of procedural knowledge"

Cognitive Load Theory provides guidelines to "prevent cognitive overload, decrease extraneous cognitive load which is not relevant to learning, and increase, within the limits of total available cognitive capacity" (Van Merrienboer, 2002, p. 12).

Cognitive apprenticeship. Here, students work with a mentor, or expert, in a subject as they develop skills. The mentor serves as a coach and provides regular feedback. Instruction is

divided into three stages: modeling, scaffolding, and fading. (Vihavainen, 2011, p. 94). In programming, examples and samples can be provided to students as models to demonstrate the thought process used. Providing engaging, collaborative activities to challenge students is a part of scaffolding. This can be completed with individual or group projects. As the student becomes successful, this scaffolding is "dismantled" in the fading stage. (Vihavainen, 2011, p.94)

In programming, **Extreme Apprenticeship** uses paired programming, where "code is written under constant reviewing" (Vihavainen, 2011, p. 94). This is a refinement of Cognitive Apprenticeship, except that the development process is emphasized over the the final product. Vihavainen (2011) describes this strategy with the following values: "Learning by doing; Continuous feedback; No compromise; an apprentice becomes the master." (Vihavainen, 2011, p.95) It is a "redefinition of the role of the teacher along with the introduction of a new collaborative way to design and manage course design." (Lembo, 2012, p.3)

Research Analysis

Several instructional design formats have been found to be effective in introductory computer programming coursework. Some form of Agile Design was used in every study in this review. In each, classroom lecture time was minimized and some type of feedback support mechanism was included. They usually included some type of collaborative work between students and additional teacher support.

Brain Research. Shaffer (2003), Sweller (1998), Van Merrienboer (2002), Van Merrienboer (2003), and Vihavainen (2011) conducted studies that used the capability of the human brain as a guide in developing instruction. They included some form of brain research of

cognitive development. "Introductory computer programming, like mathematics, requires *declarative learning* of abstract concepts and *procedural learning* acquired by practice" (Shaffer, 2003, p. 3).

Sweller (1998, p. 262) argues that "many commonly used instructional designs and procedures, because they were designed without reference to working memory limitations, are inadequate". Care must be taken then to select a design structure that does not overload students. Successes can be defined by increased student retention, passing rates, pre- and post interest survey results.

Sien (2011) identified learning challenges faced by students. "It is evident from the results of this study that students have some fundamental problems in abstracting concepts to represent the problem domain. They found it difficult to understand and work at this level of abstraction. Most importantly they seem to find it difficult to consider objects as natural representations of entities in the problem domain" (Sien, 2011, p.338).

Student Participation. None of the studies directly addressed what (curriculum) should be taught in an introductory programming course, but instead addressed how (pedagogy) it should be taught. Some form of Constructivism was used in each study explored, where students were actively engaged in the learning process.

Instructional design strategies can reduce the dropout rate of novice programmers. The passing rate of the introductory programming course studied by Vihavainen (2011) increased using 'extreme apprenticeship' methods. Also, "student feedback indicated that learning by doing was considered motivating and rewarding." (Vihavainen, 2011, p.97)

Developing an engaging course with adequate support mechanisms was a common theme in retaining novice students. These included the studies that specifically addressed paired programming, including Adams (2004), Goel (2010), Lembo (2012), Vihavainen (2011), and Williams (2002).

In a qualitative study, Thota (2010), used a constructivist approach. "A broad range of learning contexts – peer assessment, oral presentations, interactive lectures/labs, and role plays – were employed to deepen understanding and to keep students engaged and motivated" (Thota, 2010, p.112) "Helpful resources for the novice programmers were available in the form of adaptive quizzes, and lecturer and peer feedback" (Thota, 2010, p. 112) Vos (2011) also explored Constructivist Theory, developing learning connections with gaming using a focus on constructing versus just playing.

Teaching Strategies. There are several effective teaching strategies that can be matched to instructional design models for a computer programming course. These include pair programming, teacher-developed screencasts, projects developed as games, web-based instruction, creative modeling, and project-based learning. Each of these are intended to improve on using traditional teacher lectures.

Cegielski (2011) found that matching learning styles with the appropriate instructional strategy improves learning. The learning styles included Active-Reflective, Sensing-Intuitive, Visual-Verbal, and Sequential-Global. (Cegielski, 2011) It was concluded that there was evidence that "the coordination of a student's type of learning style with an instructional method

that closely reflects that learning style may enhance measurable outcomes in the educational process" (Cegielski, 2011).

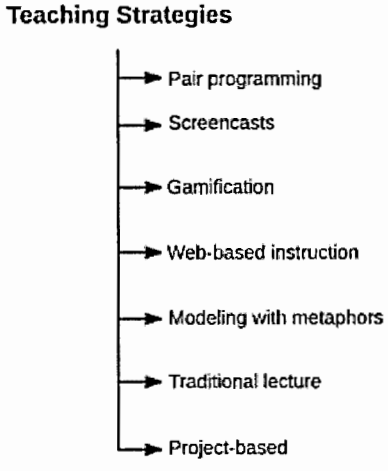
Crabtree (2013) researched placing a hands-on activity in the middle of a lecture lesson compared to the end of the class period. The results of this study did not indicate an instructional benefit for students. Koulouri (2015) changed the classroom arrangement in researching teaching problem-solving before teaching programming to determine if there was improved student performance. The study also investigated the impact of choice of programming language, problem-solving training and the use of formative assessment.

There were three distinct conclusions in the study:

1) "The choice of programming language seems to affect student learning.
2) Introducing problem-solving concepts before teaching more specific programming aspects has an impact on how students learn to program.
3) Formative feedback may not be necessarily as effective as expected unless students are ready to have a pro-active role in seeking and responding to feedback." (Koulouri, 2015, p.25)

Figure 2

Teaching Strategies



Pair Programming. "Pair programming is a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code, or test." (Williams, 2002, p. 197) They are the driver, who operates the computer, and the navigator, who is the strategic planner who oversees the driver and looks for defects (Williams, 2002). "These benefits included superior results on graded assignments, increased satisfaction/reduced frustration from the students, increased confidence from the students on their project results, and reduced workload of the teaching staff." (Williams, 2002, p. 210) According to Adams' findings (2004), when there was a difference in programming proficiency, it was better to have the less experienced partner to be the driver. When both were "experts", results were better when the partners changed roles each session. Goel (2010), Lembo (2012), and Vihavainen (2011) also included some type of peer relationship in assigning problem-solving tasks to novice students, all with positive results.

Gamification. Vos (2011) looked at student motivation and deep learning strategy use. "Constructing a game seems to be more motivating and stimulates a deep learning approach

more than playing a game." (Vos, 2011, p. 135) Theodoraki's (2014) research shows a positive correlation between a gaming environment and student attitudes of easier learning. "It is clear that the results of this study were rather positive regarding the utilization of arcade games for learning programming." (Theodoraki, 2014, p. 276) "It seems that students consider that games support them in developing an algorithmic way of thinking." (Theodoraki, 2014, p. 272) Whether student attitudes and learning are connected is not determined in this study.

Screencasts. Lee (2008) studied the use of screencasts in scaffolding instruction. It was concluded that there was no difference in learning compared with face-to-face approach. This is encouraging knowing that asynchronous instruction is effective and allows for flexible use of classroom time.

Web-based instruction. Uysal (2014) researched a web-based problem-solving instructional method compared to a traditional approach. The web tool was used for posting course materials and guided problem-solving activities (Uysal, 2014, p. 203) Here, "learners instructed by the web-supported PS method displayed higher academic performances" (Uysal, 2014, p.209)

Modeling. Davis (2007) suggests an activity to engage students by 'building' a peanut butter and jelly sandwich as a metaphor to using coding terms. Advantages of group work are also addressed.

Traditional lecture. Teachers are expected to be the content experts in the classroom. A traditional lecture format can be used to share that wise knowledge, but it is dismissed as an ineffective way to deliver a programming course. This is not a part of any of the Agile Design

instructional models since there is little student engagement.

Teacher Preparation

Adopting computer science curriculum for young learners can be a challenge in schools. Wolz (2011) addresses the lack of expertise, access to working technology, and existing school culture as deterrents to teaching computer science. "There is a critical lack of preparedness of K-12 teachers because nationally: (1) there is virtually no teacher certification in computing, (2) the pipeline crisis in computing chokes off the tap toward those entering the educational arena, and (3) the politics of professional development tends toward innovation from the top down rather than empowering teachers to be agents of change themselves. Yet, research shows that change occurs best if teachers genuinely adopt innovation." (Wolz, 2011, p.7)

Description

Background and Rationale

There is significant student access to technology in my school district. Every middle student, a population of about 520 in grades 5-8, has been issued a MacBook Pro laptop computer. The fifth grade students are allowed to take theirs home only after reaching a proficiency and speed test in typing; older students may take their devices home at the beginning of the school year.

This was the first year that a technology exploratory course was offered for all middle school students. The course evolved over a number of circumstances and needs that were identified after the start of the school year. A group of teachers and administrators served as the planning team, outlining the general scope of the course in one afternoon. This included the

curriculum director, middle school principal, three middle school teachers, two high school teachers, and myself as the district Technology Integrationist. I served as the facilitator of the committee as the outline was developed. From there, I planned the individual lessons and built a website with resources for both teachers and students.

Our team identified a number of needs for the student body. These included:

- Students had indicated in the annual Clarity Brightbytes technology survey a low level of instructional time building digital citizenship skills.
- The teaching staff has observed that students needed more support with building their keyboard skills, both in speed and accuracy, to complete their classwork.
- There was no computer science programming curriculum work included anywhere in the building. There is an emphasis this school year on improving literacy scores. Building these coding skills aligns with several Common Core literacy standards.
- Afternoon study halls were overflowing with students who had two full free periods each day.
- Students were often missing basic computer skills that would be useful in much of their coursework.

Design

The middle school technology exploratory course covers a 12-week, one-trimester time span, meeting every other day for forty-five minutes in a nine-period day. Students are in grade-alike sections. The four instructors have a diverse teaching background that includes: art, family

and consumer sciences, music, and physical education. None of the teachers had previous experience teaching a technology course.

The modules selected for the course includes Keyboarding, Google Applications, Coding, Computer Fundamentals, and Digital Citizenship. The focus of this document is the Coding module.

All activities include significant student participation, modeling an Agile Instructional Design. Students support one another working collaboratively in pairs as they gradually build their coding skills, being mindful of the cognitive load placed on them. Very little lecture is used. Instead, most instructional resources are available through web-based tools.

Programming language selection. Several programming languages were considered for the course. It was preferred to have one that was free, visual, stand-alone, and easy to install on student computers. Both Scratch and SNAP! programming languages fit the criteria very well. Python, Java, and RobotC were also considered. Scratch can be programmed in a web-based environment, but it was preferred to have a stand-alone software so that it was not essential to have Internet access outside of school. In a district technology survey earlier in the fall, less that 10% indicated that they did not have Internet access at home. However, the planning committee still considered that percentage too high. SNAP! was free and available in a stand-alone version through a download called Birdbrain Robot Server. SNAP! is made up of visual blocks of code that 'snap' together in a sequential order to build a program. The programming includes conditionals, loops, subroutines, and functions, which are all important components of advanced languages.

The SNAP! Programming language is considered a visual language and is very intuitive for early programmers to learn. This was a plus in using an Agile instructional design which includes a high student participation rate. This was important for our teachers who were mentoring the students while still learning the language themselves. The language also is relatively easy to use in a Paired Programming environment.

Robotics selection. It was preferred to use an inexpensive robotics platform that was engaging for middle school students. Robots such as the Finch, EV3, Bee Bot, Sphero, and Dash & Dot were considered. It was important that the accompanying programming language was age-appropriate for middle school students and easy to download.

A referral to a middle school in New Jersey by a local visiting computer science professor, Dr. Ursula Wolz, led us to the Finch robot. It was multi-sensory, relatively inexpensive, and interfaced well with the SNAP! programming language. The Finch robots were also selected for their durable construction, flexibility in programming, and cost-effectiveness.

The Finch robots are sold by Bird Brain Technologies and cost about $90. For a month in the fall of 2015, I checked out 25 robots through the companies' loaner program before receiving a local grant to purchase 50 of the devices. They have two wheels operated by separate motors; light, ultrasonic, and accelerometer sensors; an adjustable LED light; and a speaker that can be used play sounds at different frequencies.

This project adopted 'Extreme Apprenticeship' instructional design as identified by Vihavainen (2012). It includes three fundamental components of the instructional design: modeling, scaffolding, and fading.

As described in the research by Vihavainen (2012, p. 95), I worked to follow these practices:

Avoiding tons of preaching - limiting lecture time

Relevant examples - using tasks that all students can identify with

Start early - programming early in the course

Help available - provide scaffolding guidance

Small goals - skills progression is made in incremental steps

Exercises are mandatory - there are 5 assessed assignments in this module

Train the routine - there are repeated components in each project

Clean guidelines - the tasks are clearly written

Encourage to look for information - there are opportunities to students to use advanced skills

Programming activities. There are five programming projects within this module. Students must use critical thinking skills to collaborate and solve the open-ended tasks. Each task is intended to take about 3 class periods to complete. Table 1, Table 2, Table 3, Table 4, and Table 5 each outline the programming lessons. The first was introductory and did not need a computer. The remaining activities include programming concepts that are important for any novice programmer to learn.

Table 1

Coding activity 1

1. **Peanut Butter and Jelly** – This is a non-computer task.
    a. Build a flowchart that provides the directions to making a peanut butter and jelly sandwich.
    b. Have a partner follow your directions to complete the task. They can do only what is asked, no more and no less.
2. Use a digital flowcharting tool, such as Creately or LucidChart for the final product.

Modeling - This problem models the procedural tasks of programming.

Scaffolding - This is an entry-level task intended to emphasis the importance of strong communication skills.

Fading - Support will be provided only as students develop their communication skills.

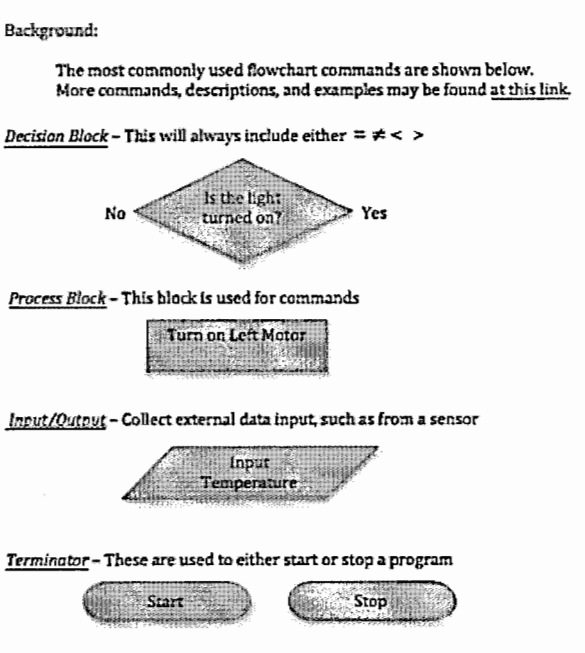
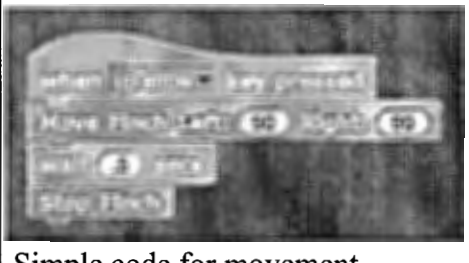


Table 2

Coding activity 2

2. **Finch Soccer** – Using a hacky-sack and two shoeboxes on their sides, build a program that will use the computer keyboard as a robot controller to play a game of soccer. Take on one of the other programming pairs in class for a best of 3 game. Control of the left and right motors will be assigned to keys on the computer keyboard.

Modeling - This problem models the procedural tasks of programming. This includes motor controls and the wait command. **Scaffolding** - Students will program only a single motor control first. The other 3 directions will be completed in a similar fashion. **Fading** - Support will be provided only as students develop their communication skills.	 Coding with SNAP!  Simple code for movement

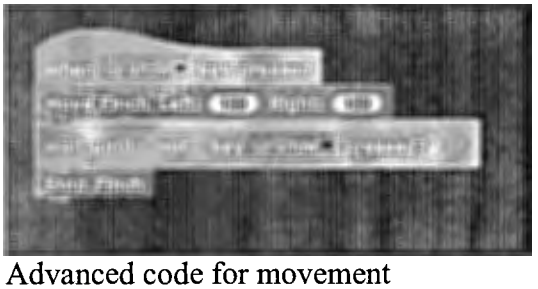
	 Advanced code for movement

Table 3

Coding activity 3

3. **Finch Food** – Construct a road path on a sheet of large paper that includes at least 3 turns with a starting point on one end and a finish on the other. Place a hacky sack at the finish to simulate bird food our robot is hungry for. This challenge has two parts. a. Part One: Use your Finch Soccer program, navigate your way through the course. Tape a marker to the tail and trace the traveled path. Each partner will take a turn as the Driver. b. Part Two: Create a new program to follow through your route with the tap of a single key. Again, use a marker taped to the tail to show your route. See how well you can stay between the lines.

Modeling - This problem models the procedural tasks of programming. This includes motor controls and the wait command. **Scaffolding** - This is a small step up in difficulty from the previous project. **Fading** - Support will be provided only as students develop their communication skills.	 Note the previous trials on the paper

Table 4

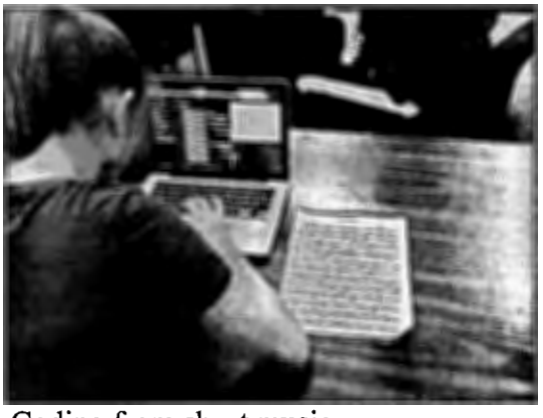
Coding activity 4

4. **Musical Finches** – Make the bird sing. Our feathered friend will play tones at different frequencies. Convert them to actual musical notes to play a familiar song of your choice that will repeat three times. Coordinate the LED light in the beak to play a coordinated light show of colors for each note played. Use loops with the LED light, sound, and motors in your coding.

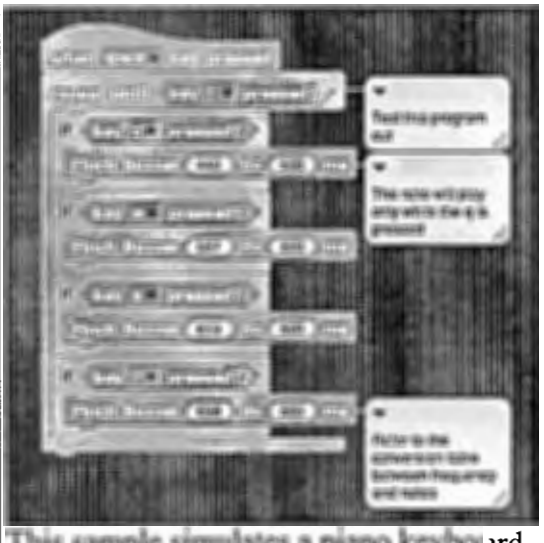
Modeling - This problem may be solved in many different ways. Students may select the model that they identify with most.

Scaffolding - There are several stages available with this project for students which depend on their skill level.

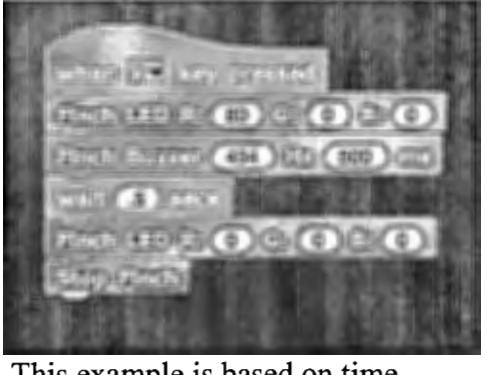
Fading - Support will be provided only as students develop their communication skills.



Coding from sheet music



This sample simulates a piano keyboard



This example is based on time.

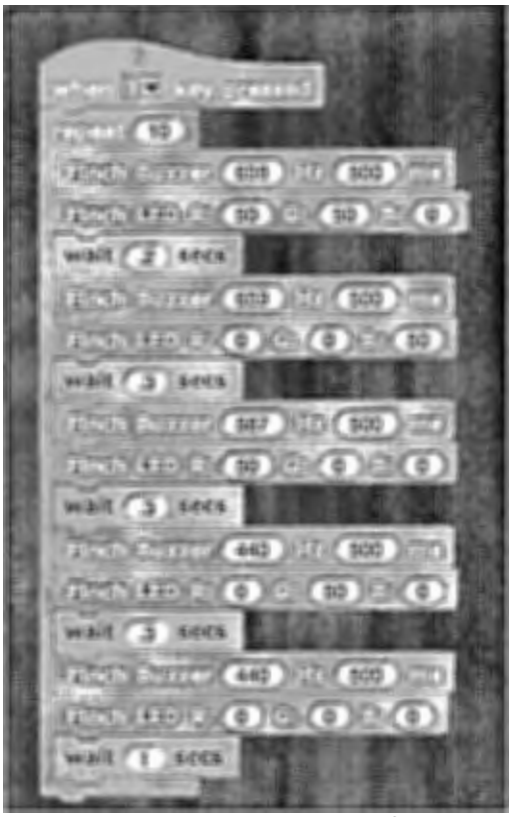
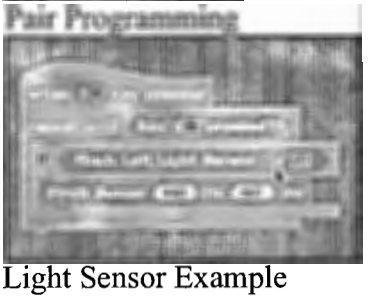
	 This example demonstrates a loop

Table 5

Coding activity 5

5. **Bird of Prey** – Prevent our little fowl from being the prey of a circling hawk. When the Finch 'sees' the shadow of a passing hawk, it will take evasive action with noises, lights, and erratic movement that will scare the hawk away. Loops and conditionals will be used with the light sensor, LED light, sound, and motors in.

Modeling - Use examples that model using the built-in light sensor. Demonstrate a conditional statement. **Scaffolding** - Students will add movement, sound, and lights as a part of this project. Students will use a conditional statement. **Fading** - Support will be provided only as students develop their communication skills.	  Light Sensor Example

Students built their programming skills in a collaborative way using what is called Pair Programming. They worked together programming on a single computer. The partners took turns assuming the roles of either Driver or Navigator.

The Driver controls the robot through the computer and writes down a design for the code. The Navigator observes the driver's work, looking for defects, and is responsible for the robot and its cords. The Finch robot is tethered to the computer by a USB cable and must be held out of the way for it to move around and complete tasks. Both driver and navigator work together to brainstorm ideas as they go.

Development

The planned implementation date was at the start of the second trimester which was in late November. It only allowed for about 4 weeks to plan a new course from scratch. I built a

Google Site with identical lesson outlines for the fifth-, sixth-, seventh-, and eight-grade students. It also included a Teacher Resources and Teacher Blog page that were not publicly viewable.

The **lesson outlines** provide links and basic information needed by the students. They were sorted by the five categories mentioned earlier. The coding components included brief descriptions of the programming challenges that students would be completing, programming examples, and multimedia resource links.

The **Resources** page included at lesson timeline which spiraled through each of the modules. This allowed for flexibility in the lesson timing, time to provide feedback to students, and pacing to maintain student interest. Student interest and motivation was a concern since the course was to be taken as pass/fail instead of receiving traditional letter grades.

The **Teacher Blog** page was built for the teachers to provide feedback and reactions after the lessons. The feedback was to be used to aid in updating the coursework for the third trimester and beyond. This page was only viewable by those classroom teachers. I created this page since the teachers were not in the same curricular area and would often not see one another to collaborate.

I had some good fortune with the robots. The manufacturer of the Finch robots had a loaner program and sent 20 devices to use at no charge by the first part of November. By mid-November I received a local grant to purchase 50 robots. Since we already had 20, the company only had to send 30 more. It was quite valuable to have some devices early for the beta testing.

Beta Testing

The beta testing was completed with two groups respectively: the classroom teachers, who were not experienced instructors of computer science and samplings of students, who were volunteers from the afternoon study halls that walked across the hall to the library with an invitation to 'play' with robots.

The teachers were each provided a robot to take home to try out the programming exercises. I installed the programming software on their computers ahead of time and met individually with them at least twice in November. We also met twice as a large group to go over the course material and address any concerns. There was considerable concern about teaching programming since none of the teachers had a programming background. Teachers shared their anxiety towards coding and reluctance to teach the course.

After seeing and working through several demonstrations and examples, the teachers replied that it was much easier than they thought it would be. There was significant relief, too, since I had planned being in their classrooms for the first several classes that included programming. There was extra reassurance knowing that there were some experienced students that they could rely on. Based on teacher feedback, I created several short tutorial videos and screenshots of example solutions for each of the challenges. Also, the music teachers offered valuable suggestions of how to provide creative support for the song creation.

We had plenty of robots for the students to use, but no containers to put them in. I talked a businessman into sponsoring plastic totes that would each hold a classroom set. For the Robot Soccer and Finch Food lessons, we were going to use ping pong balls, but found that they rolled

too easily. I happened upon an alternative "ball" by accident. The local DNR officer was handing out hacky-sacks at the middle school Health Fair, let me have the 50 or so he had left.

Students' reaction. In November, students volunteered their study hall time to learn about the programming software with some 'play' time in the library. There were 8-12 students in each session. In testing out the robots, we learned how quickly they would pick up the programming language. The interest level among the students was very high among all students.

In the pilot, I was impressed at how engaged they were and how well they utilized the robot. I found that it took very little explanation to get students started with the coding work. Initially, they enjoyed using the computer to create what they saw as a remote control for the robot. As some students figured out more complex programming, their peers gathered around computer screens to pick up new ideas. They spread out across the library floor to test out their skills and often broke out into a 'sumo-bot' competition with another team. Students were found at tables, on the carpeted floor, and spread out in the hallways, experimenting with the efficiencies of different surfaces. They learned quickly how important it was to have one student operating the computer and another acting as a tender to the robot, managing its USB tether. It was very common for students to teach one another. Afterwards, the students provided useful feedback as to how quickly they learned the programming software and also how projects could be modeled to keep them engaged.

Pilot Project

After the beta-test, the first run-through pilot included 8 sections of fifth-through-eighth grade students. Four sections met every other day for a total of 45 minutes. The other four

sections met at the same time on the opposite days. I assisted the classroom teachers in their rooms at least one full period for every module. I provided a support role, offering suggestions and fielding student questions as they worked on their projects.

It did take some time to install the programming software on all of the student computers. The technology coordinator was able to complete a remote installation for most student laptops. I was able to install the software then on any computers that were missed. Students had a few issues with the programming language not opening or saving correctly. We found that the teachers needed to emphasize a few procedural routines first before turning them loose on the projects. I have added more examples and screenshots and also modified some of the wording in the instructions.

Emily, one of the course instructors, shared that she had no idea what coding was. She was concerned about not being a classroom programming expert. She needed reassurance that her teaching role would be that of a facilitator. It helped that she has a very strong music background and was able to tie in her experience with music composition and the scripting process in a programming language. Adam, another music teacher, was inexperienced, but enthusiastic with his 5th grade students. It showed in the students' strong programming creativity.

I was impressed at how engaged student were in the projects and their willingness to explore possible solutions. Students who had some programming experience with the robots were tasked to help mentor those who were inexperienced. They preferred to be independent of one another in being creative and only wanted just enough help to get started. A common design

error was to add large numbers to the waiting time for each directional control. However, they quickly learned that less time meant more control. That is, a robot that is programmed to move forward for 3 seconds has far less control than the one that moves only a half second at a time. Colors from the LED light and sounds of 'battle cries' were often added once the motions were fine-tuned.

An important component of the lesson that was easy to overlook was letting students reflect and share their solutions with their peers. In general, students worked together very well in their paired groups. It varied with the teachers whether or not students would be allowed to select their partners.

Implementation

This project has been tested with students, teachers, and a full trimester pilot. The District is currently in its third trimester and in the second run-through with the Technology Exploratory Course. The classroom teachers are completing the lesson work with less support from me for instruction or planning. The lesson plans and support materials have been adequate, although there is still room for improvement in providing more examples. This run-through is arranged the same as the second semester, with the every-other-day format in grade-level sections.

The classroom teachers have become much more independent in this third trimester. I have received fewer requests for instructional support. They have elected to follow the lesson plans more loosely, modifying their pacing according the students' needs. There is a noticeable improvement in their confidence in teaching the course. The feedback I have received generally includes wanting more screenshots and short videos of model examples.

The teachers have editing rights on a private Google Site with lesson plans, classroom resources, and a blog. They have been very cautious to make lesson changes, but have added student-generated ideas. I have encouraged them to add entries to the private blog as feedback about the course design. Josh, an art teacher, already had significant experience with student-centered activities. He was very influential with his colleagues in adapting to the role.

In my classroom observations, students remain engaged in the classroom activities and show resilience in taking on the activity challenges. There are some students who would get off-task, but usually with additional programming. For example, when the activity called for students to use mostly motion commands, some students would spend extra time on lights, sounds, and sensors.

Reflection

There were several important results from this project. An unintended consequence was the growth in confidence of the teachers in teaching computer programming. I was impressed with how well they embraced the coursework and were willing to step out of their comfort zone. For consistency, it would have been easier to have a single classroom teacher who was already proficient at coding. It was rewarding to see the growth in so many teachers.

There is a consensus among the teachers and I that the programming activities were age and developmentally appropriate for the students. More planning will be necessary for a novice teacher to teach this course without having coaching help. Students appeared to have a positive attitude towards the programming activities, both before and after the coding projects.

Through this process, I learned how the instructional design provides significant guidance and relevance towards creating a unit of study. There are unique characteristics of a programming course intended for novices. There is a need to change teaching in many classrooms to effectively reach students. Study results indicated that an Agile design was important to use, especially an apprenticeship model. Paired Programming, web-based instruction, and modeling with metaphors helped to maintain student engagement and make them a fun, active part of completing the unit.

Future Direction

I see the next extension of these lessons working with Scratch programming language as students program a sprite, or marker, on their computer screen. There are significant resources and programming ideas available for Scratch, making it relatively easy to build more curriculum. At the high school level, I wrote and earned a Scale-Up grant for the school district to offer a one-credit computer programming course. This unit would help prepare students for that curriculum and develop interest in taking more computer science courses.

This model would work well to build more programming modules to use next school year. This Exploratory Course was the same for all students in grades 5-8. For next school year, new coursework will be needed for students in grades 6-8, and so on, until there are 4 different courses. There is significant opportunity to expand this project.

This work provides significant experience for me as I work with teachers to improve technology integration in their coursework and continue to improve the great teaching that already occurs in the district.

References

Adams, A., Lubega, J., Walmsley, S., & Williams, S. (2004). The effectiveness of assessment learning objects produced using pair programming. *Electronic Journal of E-learning*, *2*(2), 247-256.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini-languages: a way to learn programming principles. *Education and Information Technologies*, *2*(1), 65-83.

Cegielski, C. G., Hazen, B. T., & Rainer, R. K. (2011). Teach them how they learn: Learning styles and information systems education. *Journal of Information Systems Education*, *22*(2), 135.

Crabtree, J. D., Nickelson, D. W., & Parris, J. B. (2013). Clearing the Hurdles to Success in Teaching Computer Programming: Applying the Scientific Method to Improve Student Outcomes. *Academy of Business Research Journal*, *45-56*.

Davis, J., & Rebelsky, S. A. (2007, March). Food-first computer science: starting the first course right with PB&J. In *ACM SIGCSE Bulletin* (Vol. 39, No. 1, pp. 372-376). ACM.

English Language Arts Standards. (n.d.). Retrieved April 20, 2016, from http://www.corestandards.org/ELA-Literacy/

Goodyear, P. (2005). Educational Design and Networked Learning: Patterns, Pattern Languages and Design Practice. *Australasian Journal of Educational Technology*, *21*(1), 82-101.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education*, *9*, 183-196.

Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: a quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 26.

Lee, M. J., Pradhan, S., & Dalgamo, B. (2008). The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers. *Journal of Information Technology Education*, *7*(1), 61-80.

Lembo, D., & Vacca, M. (2012). Project Based Learning+ Agile Instructional Design= EXtreme Programming based Instructional Design Methodology for Collaborative Teaching. *Department of Computer and System Sciences Antonio Ruberti Technical Reports*, *4*(8).

Margolis, J., Estrella, R., Goode, J., Holme, J. J., & Nao, K. (2010). *Stuck in the shallow end: Education, race, and computing*. Cambridge, Mass.:MIT Press.

Shaffer, D., Doube, W., & Tuovinen, J. (2003). Applying cognitive load theory to computer science education. *Workshop of the Psychology of Programming Interest Group,* 333-346.

Sien, V. Y. (2011). An investigation of difficulties experienced by students developing unified modelling language (UML) class and sequence diagrams. *Computer Science Education*, *21*(4), 317-342.

Sweller, J., Van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational psychology review*, *10*(3), 251-296.

Theodoraki, A., & Xinogalos, S. (2014). Studying Students' Attitudes on Using Examples of

Game Source Code for Learning Programming. *Informatics in Education*, *13*(2), 265.

Thota, N., & Whitfield, R. (2010). Holistic approach to learning and teaching introductory object-oriented programming. *Computer Science Education*, *20*(2), 103-127.

Uysal, M. P. (2014). Improving first computer programming experiences: The case of adapting a web-supported and well-structured problem-solving method to a traditional course. *Contemporary Educational Technology*, *5*(3), 198-217.

Van Merriënboer, J. J. G., Schuurman, J. G., De Croock, M. B. M., & Paas, F. G. W. C. (2002). Redirecting learners' attention during training: Effects on cognitive load, transfer test performance and training efficiency. *Learning and Instruction*, *12*(1), 11-37.

Van Merriënboer, J. J., Kirschner, P. A., & Kester, L. (2003). Taking the load off a learner's mind: Instructional design for complex learning. *Educational psychologist*, *38*(1), 5-13.

Vihavainen, A., Paksula, M., & Luukkainen, M. (2011, March). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 93-98). ACM.

Vihavainen, A., Luukkainen, M., & Kurhila, J. (2012, October). Multi-faceted support for MOOC in programming. In *Proceedings of the 13th annual conference on Information technology education* (pp. 171-176). ACM.

Vos, N., Van Der Meijden, H., & Denessen, E. (2011). Effects of constructing versus playing an educational game on student motivation and deep learning strategy use. *Computers & Education*, *56*(1), 127-137.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair

programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197-212.

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational Thinking and Expository Writing in the Middle School. *ACM Transactions on Computing Education*, *11*(2), 1-22.

Appendix A

The following Common Core English Language Arts, or Literacy, Standards (2016) as published at www.corestandards.org may be met in this project.

CCSS.ELA-LITERACY.CCRA.W.6

Use technology, including the Internet, to produce and publish writing and to interact and collaborate with others.

CCSS.ELA-LITERACY.CCRA.SL.5

Make strategic use of digital media and visual displays of data to express information and enhance understanding of presentations.

RI.3: *K-5

Have students describe what would happen if the blocks in a program went in a specific order. Identify cause-and-effect relationships by using "if this, then. . ." blocks.

RI.5: *2-4

Locate answers to a question using keywords, sidebars, and glossaries. (Programming tools use menus and categories to organize blocks.)

RI.5.7

Draw on information from multiple print or digital sources, demonstrating the ability to locate an answer to a question quickly or to solve a problem efficiently.

RI.8.7

Evaluate the advantages and disadvantages of using different mediums (e.g., print or digital text, video, multimedia) to present a particular topic or idea.

CCSS.ELA-Literacy.W.6.8

Gather relevant information from multiple print and digital sources; assess the credibility of each source; and quote or paraphrase the data and conclusions of others while avoiding plagiarism and providing basic bibliographic information for sources.

CCSS.ELA-LITERACY.WHST.6-8.6

Use technology, including the Internet, to produce and publish writing and present the relationships between information and ideas clearly and efficiently.

MP.1: *K-8

In programming activities, students must persevere in problem solving.

NBT.1: *2-5

Use wait blocks and movement blocks in programs like Scratch and Tynker to differentiate between .01, .1, 1, and 10 seconds.

4.OA.5

Have students create drawings in programs that repeat a pattern. This can be done with the "repeat" (a.k.a. "loop") block. Students can demonstrate their understanding of multiplicative procedures and patterns that follow a specific rule.

6.NS.5, 6.NS.6, and 6.NS.7

Have students build programs where actors (or sprites) move to specific points on a coordinate plane, based on an action (a conditional).

Appendix B

The following tables indicate the relevance of each source with this project.

Table A1

Research-Based Qualitative Studies

Authors	Year	Sample	Instructional Design
Adams	2004	14 undergrad students	Pair Programming
Goel	2010	178 students	Social Learning Theory / Pair Programming
Lembo	2012		Extreme Programming,
Van Merriënboer	2003		Cognitive Load Theory
Shaffer	2003		Cognitive Load Theory
Sweller	1998		Cognitive Load Theory
Thota	2010	26 students	Constructivism
Van Merriënboer	2002	26, 69, and 87 students	Cognitive Load Theory
Van Merriënboer	2003		Cognitive Load Theory
Vihavainen	2011	20 students	Cognitive Apprenticeship
Vihavainen	2012	67 and 44 students	Extreme Apprenticeship
Vos	2011	235 students in 9 classes	Constructivist
Williams	2002	44 paired students	Pair Programming
Wolz	2011	45 students	Cultural Diffusion

Table A2

Relevant Article Resources

Authors	Year	Design Relevance
Brusilovsky	1997	Using mini-languages to introduce programming
Cegielski	2011	Matching learning styles with instructional strategy
Chang	2011	Delivery of Instruction – Dual Screen
Crabtree	2013	Include hands-on activities in instruction
Davis	2007	Engaging hook in instruction
Goodyear	2005	Cognitive Apprenticeship / Networked Learning
Koulouri	2015	Build problem-solving skills before programming skills
Lee	2008	Screencasts in instruction delivery
Sien	2011	Creating graphical representations to use in instruction
Theodoraki	2014	Game-based instruction
Uysal	2014	Web-supported problem-solving instructional method