Technical Report CS12-01

Examining Tools for Teaching Children Computer Programming Concepts

Eric A. Hildebrand

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Robert S. Roos
Second Reader: Dr. Gregory M. Kapfhammer

Allegheny College
2013

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

Eric A. Hildebrand

ERIC A. HILDEBRAND. Examining Tools for Teaching Children Computer Programming Concepts.
(Under the direction of Dr. Robert S. Roos.)

ABSTRACT

This project investigates the goals of seven tools used for introducing children of ages 8-16 to computer science through computer programming: Computer Science Unplugged, Logo, StarLogo TNG, Scratch, Alice, Storytelling Alice, and the Kodu Game Lab. These tools are examined in the light how well they achieve their goals in practice and how well-suited they are to conveying introductory computer science concepts based upon a careful examination of existing research and fresh exploration and analysis. The results serve as a resource to educators, the developers of these tools, and future developers of tools designed to address existing shortfalls.

Acknowledgments

Firstly, I want to thank my parents for all of their support and encouragement over the years and during this project.

I also want to thank Dr. Andrew Thall and Dr. Gregory Kapfhammer for meeting with me while I searched for the right undergraduate college. Your passion for the field and devotion to your students made my school choice for me, and I wouldn't have had it any other way.

Dave Roncolato, the Office of Community Service, and the Bonner Foundation deserve many thanks as well, providing me with an opportunity to both afford tuition and find my passion in service. My time working at William Gill Village was one of the most rewarding experiences of my time at Allegheny, and frequently was responsible for providing the energy boost to make it through the hardest of weeks.

Thanks also go out to my advisors during this process. Without the guidance and patience of Dr. Robert S. Roos and the late Dr. Robert D. Cupper this project would not have happened. Thank you, professors, for always having time for questions and going out of your way to help every at every step along the way.

Contents

List of Figures

Glossary

Throughout this project, seven tools are examined and seven basic computer programming concepts are defined. This glossary is provided as a convenient reference for the reader and includes links to where the tools can be found.

Computer Science Unplugged (*CS Unplugged*) A tool being developed at Canterbury University of New Zealand that provides a framework for teaching computer science and programming concepts through computerless activities.
`http://csunplugged.org/`

Logo An adaptation of the Lisp programming language developed by Seymour Papert. The user commands a turtle which can be used to trace patterns and move around on a two dimensional canvas. No one version of Logo is standard but the core of the tool remains unchanged among most variations.
`http://www.eecs.berkeley.edu/~bh/logo.html`

StarLogo The Next Generation (*StarLogo TNG, SLTNG*) The latest version of the Logo offshoot StarLogo, being developed at MIT's Scheller Teacher Education Program. StarLogo TNG provides a drag-and-drop block based system for programming and manipulates a 3-D turtle model in a virtual world. Available at no cost for Windows, Mac, and Linux.
`http://education.mit.edu/projects/starlogo-tng`

Scratch A tool under development at MIT's Media Lab, Scratch provides an environment for programming using puzzle-piece-like blocks that can be dragged

onto the canvas from a palette. These blocks manipulate 2-D sprites (some come standard, others can be drawn from within the tool) on a 2-D stage. Available at no cost for Windows, Mac, and Linux.

`http://scratch.mit.edu/`

Alice A tool being developed at Carnegie Mellon University that has a similar drag-and-drop interface to StarLogo TNG and Scratch. The user manipulates 3-D models in virtual world. This tool is built using Java and can bidirectionally port code between the standard interface and Java. Available at no cost for Windows and Mac, with a free beta version for Linux.

`http://www.alice.org/`

Storytelling Alice (*ST Alice*) An offshoot of Alice, ST Alice builds upon the number of standard models and animations. Additionally, it includes more methods for object interaction and comes with a tutorial that uses the storytelling method to introduce a user to programming in Alice. Available at no cost for Windows only.

`http://www.alice.org/kelleher/storytelling/index.html`

Kodu Game Lab (*Kodu*) Being developed at Microsoft's Research FUSE Labs, this is a tool that presents a unique interface for programming through a video game controller. Kodu provides a high number of standard methods for interaction and world creation, and uses pages of scripts and circular trees to bring relevant code blocks to the attention of users quickly. Available at no cost for Windows and $5 for Xbox 360.

`http://fuse.microsoft.com/page/kodu`

Interaction The handling of received input from the user to affect execution and the displaying of output from a program or script.

Repetition Constructs for several styles of iteration or looping, such as *infinite*, *conditional*, and *for* loops.

Conditional Logic A combination of branching structures and boolean expressions and logic, such as the `if-then-else` construct.

Variables The symbolic representation of data that can be referenced by name and store changing values.

Functions The construct for creating encapsulated blocks of code that can be called from elsewhere in a program with or without parameters.

Randomization The generation of arbitrary data or choices during execution not based upon input or other code.

Parallelism The apparent simultaneous execution of multiple separate blocks of code, and the communication and synchronization between them.

Chapter 1

Overview

There can be little doubt that computers are increasingly becoming part of daily life. Since the advent of the personal computer in the mid seventies, computers have become tools for any number of activities and realms of study. Some examples of this are the fields of statistics, biology, economics, mathematics, aerospace, and even art [22]. With these fundamental changes to the system it has become important to practice computational thinking. Jeannette Wing defines the term as "taking an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing [22]." Computer science is being pushed to become part of early schooling by organizations like the Computer Science Teachers Association (CSTA). The CSTA K-12 curriculum breaks computer science into five related threads: computational thinking; collaboration; computing practice and programming; computers and communications devices; and community, global, and ethical impacts [11].

Computational thinking acts as a methodology for problem-solving and links computer science with other applicable fields. The second thread, collaboration, emphasizes the importance of working in a group. As written in the CSTA K-12, "significant progress is rarely made in computer science by one person working alone [11]." Computing practice and programming, likely the most related to the lay-person's idea of

computer science, focuses on the use of tools and interfaces to explore the concepts of programming in order to solve computational problems. The next thread is computer and communications devices, and it centers around understanding the components of computers, communication devices, and computer networks. Community, global, and ethical impacts is the final thread, which studies ethical concepts such as privacy, security, and intellectual property.

1.1 Computer science through programming

For many students, the first real association with the computer science field is through computer programming. Indeed, the first computer science course at Allegheny College is listed as "An introduction to the principles of computer science with an emphasis on algorithmic problem solving and the realization of algorithms using a modern object-oriented programming language." However, this course is of a college level and the language used may not be appropriate for all ages. Multiple languages and tools have been developed to provide a more suitable environment for middle school and high school students to learn about computer science and computer programming concepts. This project examines seven of these tools, all of which are undergoing continuing development in one form or another: Computer Science Unplugged, Logo, StarLogoTNG, Scratch, Alice, Storytelling Alice, and the Kodu Game Lab.

1.2 Investigated tools

The first project covered will be Computer Science Unplugged. This project, created at Canterbury University of New Zealand, focuses on providing methods to teach children basic computer science concepts without using computers themselves. The activities are predominantly kinesthetic and encourage teamwork within the group of

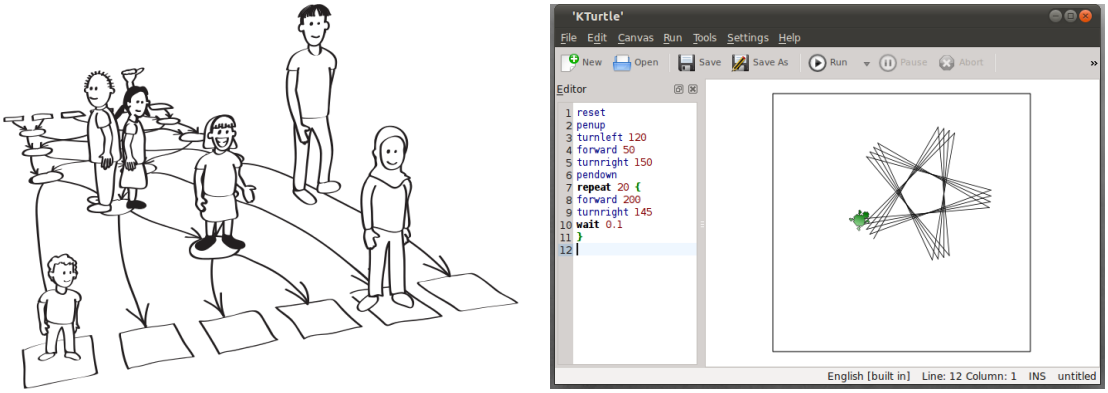

Figure 1.1: A cartoon depicting a CS Unplugged teaching data sorting (left) and a small sample program in KTurtle, a Logo programming environment for Linux (right).

learners, but can also be competitive between groups. After an activity is completed, the teacher explains the concept of computer science or computer programming that was being emulated. This may be useful in dispelling myths about the "boring" nature of computer science and may be useful in outreach programs, due to its computerless nature. However, there may be many drawbacks as well, such as needing a minimum number of students to be able to complete activities. Additionally the entirety of the material is being presented by the instructor, which may not allow for independent learning by the students themselves. As explained in the third section of this document, CS Unplugged is very different from the remaining six tools, but remains a valuable consideration for comparison [2][15].

Logo is a computer programming language that began as an adaptation of Lisp by Seymour Papert. While there is no standard dialect of Logo, the majority of implementations share the same basic core. The most commonly recognized aspect is the turtle, as seen in figure 1.1. This is an on-screen cursor that can be moved around the screen to draw lines and patterns. Logo is an interpreted language, where users can direct the turtle to proceed forward or backward, or turn left or right, by an arbitrary amount. Users also control whether the turtle's motions leave behind a trail by PENUP and PENDOWN commands. Additionally, Logo supports looping,

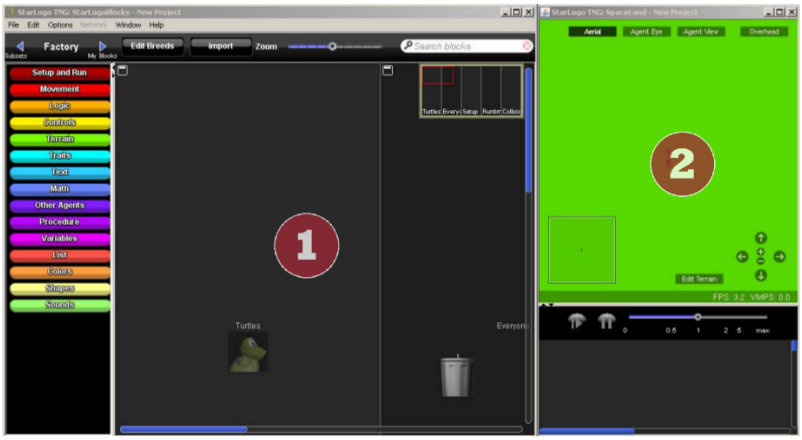

Figure 1.2: The standard interface of StarLogo TNG, with a palette of code blocks on the left edge, the programming canvas in the center, and the viewport on the right.

functions, recursion, and other features inherited from Lisp. A more advanced form of Logo, titled StarLogo TNG, is under development at the Scheller Teacher Education Program at MIT. This enhancement removes the focus from the keyboard and allows users to connect blocks of code on a canvas through a multiple-pane drag-and-drop interface (see Figure 1.2). This code then manipulates a turtle model (or any of the other included models) in a three-dimensional world. [12][10][9].

Another tool used in educating children about computer science concepts is the Scratch programming language. Being developed at the MIT Media Lab, Scratch is a very visual language where the user creates scripts and programs by dragging and dropping code blocks onto a programming surface to control sprites on a 2-dimensional "stage." The interface layout presents a window for selecting new code on the left, a canvas for programming in the center, and the output stage on the right as seen in Figure 1.3. This interface is very similar to that of StarLogo TNG, which may be related to the fact that both projects are currently under development at the same institution. This layout can streamline the learning process by removing the possibility for minor syntax errors of other languages (such as semicolons and parenthesis/bracket matching). Through arranging blocks into stacks, users can create

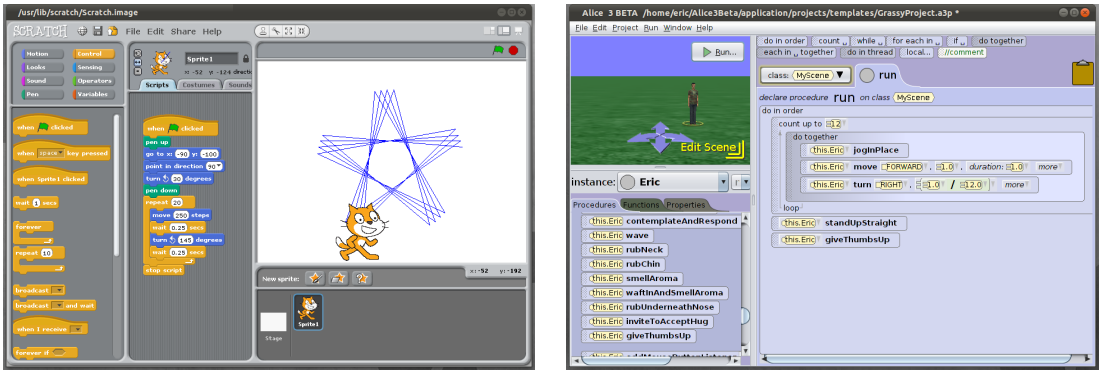

Figure 1.3: The Scratch interface (left) with its palette of code blocks, programming canvas, and stage; and the interface of Alice (right) with viewport, code block palette, and large programming surface.

projects ranging from simple 2D animations to complex interactive games. Scratch also includes a very thorough help system where the user can right click on any block to find a plain-text description of its function as well as an example of common usage. This tool has the basic drawbacks of all machine-centric approaches, requiring a computer for each learner or pair of students in a pair-programming situation [6][19].

Alice is a drag-and-drop programming language, similar to Scratch, being developed at Carnegie Mellon University. Alice differs in that it focuses on a 3D environment for the resultant animations and interactive games. The interface is laid out with a viewport in the upper left, the palette of available code blocks in the lower left, and a large programming area on the right side as seen in Figure 1.3. Alice is object-oriented, with objects such as people, animals, and vehicles being placed in a 3D world. Included within the Alice tool is a standard tutorial system that walks users through the basics of the interface and some common code blocks. Based on Java, Alice allows for two way porting of code between the standard visual interface and a Java programming environment. This may be useful in a sense that the tool can grow with the user more than others. There exists a branch of this tool called Storytelling Alice with a similar interface, which differs in that it offers many more

Figure 1.4: Kodu Game Lab interface, showing script pages (left) and the circular tree system used for selecting code blocks (right).

high-level object interactions, more standard 3D objects, and a unique tutorial that teaches the user about the interface and mechanics of ST Alice through a follow-along narrative [19][4].

The last tool this project will investigate is the Kodu Game Lab environment being developed by Microsoft's FUSE Labs. Released for the Xbox 360 gaming console in 2009, Kodu is now also available as a free download for Microsoft Windows machines. Kodu presents users with a unique programming interface: the user programs solely via an Xbox controller, instead of a keyboard and mouse. Using this input device, objects can be added to the world and their behavior can be controlled through script pages that users populate through a circular tree of options as demonstrated in Figure 1.4. By detecting which fields the user is trying to place a new block, the tool automatically narrows the shown options down to remove blocks that are not situationally applicable. Writing these scripts, programmers can control the interactions between objects, characters, and the environment in a three-dimensional world similar to that of Alice and StarLogo TNG. The project aims to make game programming more accessible to children through a simplified grammar and an interface that removes the possibility of syntax errors [8][7].

1.3 Thesis statement, claims and contributions

This project investigates the intended goals of the seven chosen tools, and examines studies of children using the tools to judge their effectiveness in representing and teaching a set of basic computer programming concepts comprised of: interaction, repetition, conditional logic, variables, functions, randomization, and parallelism. To reduce the amount of missing information from the existing studies, each tool-concept pairing is also analyzed by the author when no other information is available.

The existing tools for introducing children of ages 8-16 to computer science through computer programming, while useful, are imperfect and can be improved in a number of ways. Research and analysis according to a strict criteria set is useful to determine the best and worst features of each tool in order to present feedback to the tools' current developers, provide a resource to potential users or teachers, and serve as a reference for developers of future tools.

1.4 Thesis outline

In the next chapter, the goals of the chosen tools are analyzed and criteria in the form of computer programming concepts are chose and explained. In chapter 3, other considerations are accounted for and Computer Science Unplugged is investigated. In chapter 4, the investigation turns to Logo and its descendant StarLogo TNG. Chapter 5 studies Scratch, Alice, and its offshoot Storytelling Alice. The Kodu Game Lab is examined in chapter 6. Effects of gender on the experience of learners are described in chapter 7. Finally, the results from chapters 3 through 7 are compiled and discussed in chapter 8. This chapter discusses the significance of the findings and sheds light on the usefulness of each tool. This leads into a reflection on the key points that educators and current developers can take away from this study, as well as a reflection on how the next generation of tool can be improved to address the existing shortcomings.

Chapter 2

Methods

Before discussing a method of approach, there are two ideas which first need to be determined: the goals of the tools and the criteria upon which they are to be judged. The selected tools share a number of goals, with the exception of Computer Science Unplugged as will be discussed in Chapter 3.

2.1 Tool goals

One of the common goals of the chosen tools is to lower the barrier of entry for computer programming through graphical output or a graphical interface [5][8]. Logo, in its original form, presents users with a two-dimensional view of code execution by way of a turtle icon and a tracing of the path the turtle has taken. More advanced versions of Logo, such as StarLogo TNG, steer away from the traditional typed-code input and present users with a graphical means for selecting code blocks and connecting them together [9]. Scratch incorporates an interface that is similar to StarLogo TNG, with a palette of code blocks displayed on the left that can be dragged into stacks on the programming surface. Very little typing is necessary to program with Scratch, and the same is true about Alice. Alice simplifies the programming interface in a similar manner to both StarLogo TNG and Scratch, providing a two-

dimensional drag-and-drop interface for programming. The Kodu Game Lab takes the interface simplification one step further, removing the keyboard and mouse entirely and displaying the available programming options through circular trees that are navigated via a dual joystick game controller [8]. All of these manners of simplification do two things: reduce or remove the possibilities of minor errors in syntax, and display instant feedback about the executing code through visual animations. These animations sometimes highlight the exact line of code being executed, such as in Logo, or the block of code being run such as in Scratch.

Another shared goal of these tools is to present an interface that encourages the learning of computational ideas and computer programming concepts [10]. Standard Logo adheres to this by simplifying the syntax for common concepts such as looping, but does not inform the user of the complete list of commands available. StarLogo TNG, Scratch, and Alice all present the full array of code blocks in a tabbed palette. This reduces the need for other resources or supervised learning, as users can locate code with which they are unfamiliar and then insert it into a project and experiment with its behavior. Kodu also presents a full listing of the available pieces of code, and presents it through a tree that contextually filters out everything except for the blocks that are applicable to the location they are being inserted. The tools present all of the appropriate computer programming concepts in one manner or another, usually renamed to be more readily understood by beginners. Most of these tools include a number of starter objects, allowing users to begin experimenting with programming right away instead of needing to create characters or draw scenes.

A third and final common goal is to make programming more appealing to youth. Appealing is used in a general sense, as what specifically appeals to one subset of the demographic is unlikely to be as enjoyable to others. To accomplish this, these tools build upon the previous two goals with colorful and clean interfaces. Additionally, the

standard objects or characters and scenery include cartoon turtles and cats and also three-dimensional models of many animals, locations, people, and robots. Having this wide range of objects built-in can help appeal to the unique interests of the potential learners.

2.2 Examination criteria

Now that the goals of the tools have been examined, the set of criteria upon which they will be judged must be decided. This project expands upon the basic concepts defined in the CSTA K-12 Curriculum [11] to study the concepts of interaction, repetition, conditional logic, variables, functions, randomization, and parallelism.

Interaction is defined as the concept of taking input from the standard channel, such as the keyboard and mouse in most tools or the controller in Kodu, and processing it to affect the execution of a program. Additionally this concept includes information flow of the opposite direction, presenting data or objects from the execution via a standard output channel. This occurs by default in the given tools, as a result of the graphical nature of the languages.

Repetition encompasses all manners of iteration over blocks of code. This includes infinite looping, conditional looping such as while-loops, and counter based iteration such as for-looping. Control flow concepts such as breaking out of a loop are decided to be of a higher level and are not part of this project.

Conditional logic combines two interrelated concepts: branching and boolean logic. Branching can be thought of through if-then-else or case structures, allowing different segments of code to be run in different circumstances, based upon a provided condition. These conditions are frequently represented by a boolean expression. Boolean logic covers the notions of comparison and also the combination of small expressions into larger expressions through the operations of `and`, `or`, and `not`.

Variables are the constructs of a symbolic name being used to represent information. This data can be known beforehand or be an unknown result of programmed code, and can be either changing or static. This concept is directly related to the majority of other concepts in practice, serving as a location to store input or output, a counter for iteration, or values used in conditions or comparisons.

Functions, also known as subroutines or methods, are the concept of encapsulating a block of code so that it can be referred to by name from other areas of the program. These constructs can include the concepts of parameters and return values, but these are not always necessary. Indeed, a beginning programmer may instead create functions without these subconcepts by way of global or shared variables or objects.

Randomization is a concept not listed in the CSTA curriculum, but is important to include due to the nature of the given tools. This concept pertains to the generation of arbitrary data during execution, without interaction from the user or other sources on input. This concept can be important to simulation and game design, which many of these tools list as a secondary focus.

Parallelism is defined to be the apparent simultaneous execution of multiple calculations or code segments at once. This concept refers to the idea of multithreading, but it also considers the interactions between threads in the form of communication and synchronization.

2.3 Method of Approach

This project follows a two-pronged approach. The first method of research examines existing studies of the use of these tools in learning environments with children between ages 8 and 16. These studies are examined to discover the benefits and drawbacks of each tool in classroom settings, but also focus on how well each tool

appears to convey each of the computer programming concepts listed in the previous section. Due to the varying nature of each study, not all results are directly useful; this impact is noted in the the respective analyses. Furthermore, the research is far from comprehensive and there are gaps in what can be learned from studying existing materials. To combat this, this project also incorporates exploratory analysis by the author into each of the respective tools. Each concept is implemented by the author in a trial and error fashion in each tool, with minor prior experience with Scratch, and no previous experience in the remaining tools. This data is useful for comparison between tools as well as expanding the amount of information that is available for each concept.

Chapter 3

Other Considerations and CS Unplugged

As mentioned earlier, Computer Science Unplugged is different from the remaining six tools, and will be examined in the second half of this chapter. Before that, however, there are several other considerations to be discussed.

3.1 Special Considerations

Two important precursors to teaching children computer programming concepts are the presence of computers in the learning environment, and the level of basic computer literacy possessed by the students. Six of the seven tools require a machine for use and thus learning. All of the studies that this project examines take for granted a basic knowledge of the input devices such as the mouse, keyboard, and video game controllers. As mentioned in the CSTA K-12 curriculum, computer programming is just one strand of the broader field of computer science and therefore the students should also be taught about other aspects if they are not already familiar with them [11].

Beyond basic computer skills, several of these tools also expect a basic level of

mathematical fluency for full use. If a user wants to move a turtle forward and then turn to the right, it is helpful for them to understand the concept of *angles*. However, some tools may be able to avoid this issue by allowing a run-and-see type of experimentation where users can enter numbers and quickly see the results on the screen. For more advanced object motion it may also be important for learners to grasp the idea of a plane, with x- and y-coordinates, or even the third dimension.

Another important consideration is the level of familiarity that the instructors have with the tools being used. In many of the studies examined the instructors possessed little to no prior experience with the utilized tools. This has the potential to affect the results of each study, and this project makes note of this factor for all studies that provide this information. This combines with the fact that some of the studies are instruction-based, with teachers guiding the learning, while others are exploration-based. In this latter case, the above factor may play less of a role in learning.

Many of the studies looked at use games or art as a motivation for learning. A study from the Osaka University of the Arts examined the role of these motivators in a learning environment and presented important information. When a curriculum was based on game design, the interest levels of students were higher throughout the duration of the study. Art and animation based curricula also had positive effects on the interest and motivation levels of learners, and though not as strong the effect reached comparable levels by the end of the study [16].

3.2 Computer Science Unplugged

Computer Science Unplugged is unique among the researched tools in multiple ways. The most apparent distinction is that CS Unplugged is the only tool that is not dependent upon computers themselves. The learning activities can be printed out

from the tool's website or purchased in book form from several sources. The activities themselves take the form of magic tricks, games, and competitions using paper, writing utensils, chalkboards, and other analogue media [2]. One example is the Sorting Algorithms activity, where students explore algorithms such as Selection, Quick, Insertion, Bubble, and Merge-sorting. Items of various weights are compared two-at-a-time and moved into order from lightest to heaviest. CS Unplugged is also unique in its focus on computer science as a whole, not just computer programming. In fact, one of the goals of the tool is to differentiate between computer science and programming; the activities cover a wide range of material from binary numbers to encryption and beyond.

Presentation of concepts

Since CS Unplugged is not programming-centric, the programming concepts being examined by this project are more difficult to detect. In most cases, the concepts are present, but may not be noticed by learners unless special note of them is made by the activity leader.

Interaction is present in every activity. In the sorting algorithms activity, input is given as a series of objects with varying weight, and the output is the same set of objects, but sorted from lightest to heaviest. The public key encryption activity utilizes inputs of a text message and outputs of a secured, scrambled message that can only be understood by the intended recipient.

Repetition is present in most of the activities as well. Activities frequently have a set procedure that is repeated again and again, for example in the sorting algorithms activity. Here, the repeated tasks structure is: choose two objects to compare, compare the objects, and replace the objects in an ordered manner. This repetition also exists in the text compression activity, where patterns are repeatedly found and

replaced in several paragraphs.

CS Unplugged continually uses conditional logic throughout its activities. In the above example of the sorting algorithms, conditional logic is expressed as two different outcomes of a weight comparison. If the object on the left weighs less than the object on the right, no switching would occur. However if the opposite is true, the student would execute a different task and rearrange the compared weights.

Variables are presented in several activities. One example of this is the sorting network activity. Squares are placed on the floor representing variables, and the students represent the values stored through cards with numbers written on them. The initial values are set by assigning students to starting locations, and the variables are changed by having students physically move to other locations.

Functions, randomization, and parallelism are not presented like the above concepts, but can be found in some activities and highlighted by instructors wishing to do so. Functions can be noticed in the encryption activity, where one student could write a message and pass it to another student. This second student encrypts the message and returns the encrypted form to the first student. Randomization can be seen in the sorting network activity if students can pick randomly from a set of cards to determine everyone's values. This activity also can be used to demonstrate parallelism, as a large number of students can be comparing their values with each other independently at the same time during each step of the network.

Research Findings

Research into the effectiveness of CS Unplugged has typically focused on its effects on the interest levels of young potential computer scientists, with data about the effectiveness of conveying the chosen concepts being limited. One of these studies, conducted by the authors of the tool, examined the impact of CS Unplugged on

students between the ages of 10 and 12 years. Interviews after participation in the program showed increases in the level of interest in computer science, and the authors conclude that the tool is valuable in outreach as well as for education about computer science and programming concepts without delving into actual computer programming [15].

Another study gathered results of CS Unplugged activities on a group of 12-14 year-old students. Led by a computer science graduate, 18 activities were used and data was gathered from both the participants and similar students not participating in CS Unplugged. While the results did show a small increase in interest in computer science among participants, the level of understanding of computer science as a field remained unchanged. The authors conclude that CS Unplugged may be useful in limited situations such as a first-level introduction to the field of computer science, but is not effective for real education about the specifics of computer science and programming concepts. The study also concludes that CS Unplugged fails to encourage the study of computer science in later education such as high school and college, and suggests that a more detailed view of what computer scientists do in the real world would be beneficial [14].

Personal Exploration and Summary

This tool has a several of advantages to the others since it requires no access to computers or gaming consoles. This would certainly be beneficial in outreach programs and allows for great flexibility of deployment. Additionally the beginning CS Unplugged activities are suitable to a range of ages that begins earlier than the remaining tools while the later activities cover much more complicated computer science ideas in a digestible manner. A drawback to CS Unplugged is the necessity of an instructor and multiple users. While handouts are readily available for printing and

distribution, the activities are always require a leader to direct the group's learning. The tool focuses on computer science concepts and not programming concepts by virtue of its goals, and programming concepts used during the activities may not be apparent to learners unless the leader brings them to the group's attention.

While Computer Science Unplugged achieves some of its goals, it does not meet the goals specified by this research project very well. This tool would be most suitable for younger students unfamiliar with computers, outreach and service programs, and potentially as brief overview of computer science before transitioning to another of the examined tools.

Chapter 4

Logo and StarLogo TNG

In this chapter the examined tools are Logo and StarLogo The Next Generation. There are many different branches of the Logo programming language, so the two specific variants studied are UCB Logo and KTurtle. UCB Logo, named for the location of its development (University of California: Berkeley) is well-maintained, with the last major release being within the last four years. This branch is also regarded as staying true to the original goals of Seymour Papert. The second variant examined is KTurtle, a Logo programming environment developed as part of the KDE Education project. This is a tool designed to provide more assistance to the novice user through simpler error presentation (syntax is highlighted and errors are explained in commonsense terms) and is available in at least 6 languages including English. KTurtle changes several of the basic commands, but the structure of the language is true to the original implementation.

StarLogo TNG is based upon the ideas of Logo but, as discussed, presents the user with a wide array of code blocks that can be arranged on a surface to manipulate events in a 3-dimensional world. Each of the different categories of code blocks has both a unique shape and unique color. The shape of the blocks allows users to instantly judge which blocks can be placed into a given area and are meant to be useful for diminishing time wasted on impossible arrangements of code. The coloring

system functions in a similar manner, but allows users to see the patterns in the arrangement of blocks by category on the coding surface [9].

4.1 Logo

Presentation of concepts

The first concept examined is interaction. Being the original turtle-based language system, Logo and its variants focus on user input through typing lines of code either into a terminal or a word-processor like environment. While more advanced input mechanisms are available, such as reading from the filesystem and other devices, they fall outside of the beginner level construct range being studied. UCB Logo and KTurtle are interpreted languages that step sequentially through user written code that commands the movement of the turtle, drawing paths and patterns on the 2-dimensional canvas. To move the turtle, the user would write a line of the format MOVEMENTCOMMAND AMOUNT. For example,

RIGHT 90

FORWARD 100

commands the turtle to turn 90 degrees to its right and move forward in this new direction by 100 units (typically pixels). The basic set of commands for UCB Logo is: FORWARD, BACK, LEFT, and RIGHT. Each of these commands have respective abbreviations: FD, BK, LT, and RT. KTurtle differs slightly with the commands being FORWARD, BACKWARD, TURNLEFT, and TURNRIGHT. These commands are more descriptive than those of UCB Logo, particularly with the commands expressing that the turtle is *turning* in the appropriate direction, not *moving* that way. The abbreviations are FW, BW, TL, and TR. As the turtles move within the plane of the canvas, their paths are traced by penstrokes. The user can raise and lower the pen

to disable or enable this tracing via the commands PENUP and PENDOWN, or PU and PD for short. KTurtle allows for messagebox input prompts using ASK <MESSAGE>. As the above commands demonstrate, the primary output channel of Logo is the drawing canvas. Additionally, KTurtle can display messageboxes by MESSAGE <TEXT>. More extensive output options are available for both variants examined, but fall beyond the scope of this project.

Repetition of code blocks in these tools is present in all expected styles. Simple repetition for a set number of iterations is through

REPEAT <COUNT> { <CODE> } .

KTurtle varies only in that square brackets are used in place of curly braces. For loops are accessible in UCB Logo via

FOR <COUNTER> {<BEGIN> <END> <STEP>} {<CODE>}

and similarly,

FOR <VARIABLE> = <BEGIN> TO <END> STEP <STEP> { <CODE> }

in KTurtle. In both instances, the step component can be left out, resulting in a default step of 1. Iteration based upon a more general testable condition is done by

WHILE <EXPRESSION> { <CODE }

in both tools. UCB Logo includes two additional iteration commands: the UNTIL loop that is structured in the same manner as WHILE and intuitively loops while the condition expression is not true, and the FOREVER { <CODE> } loop which continues indefinitely. KTurtle can mimic this functionality through

WHILE TRUE { <CODE> }.

Conditional logic and branching is presented in several useful forms in Logo. Both tools have IF structures of the forms

IF <CONDITION> { <CODE-IF-TRUE> }

IF <CONDITION> { <CODE-IF-TRUE> } { <CODE-IF-FALSE> }

that can be used to selectively execute code segments based upon any desired condition (doing nothing in the first line when the condition is not true and executing the <CODE-IF-FALSE> code block in the second line. Case structures, which can compare an input to many different conditions at once, are available in both tools as well. However, the implementations of these commands are more complicated than the above and may require more extensive programming experience than this study assumes.

UCB Logo and KTurtle handle the concept of variables in two different ways. The first of the two creates a new variable (or replaces an existing variable's value) by MAKE <VARNAME> <VALUE>, in which the variable name must be a word and is not case-sensitive. To get the value stored in a variable, the user would type :<VARNAME>. For example, a loop that only iterates while a variable X is greater than 5 could be written as:

WHILE :X > 5 [...].

This colon tells the interpreter that <VARNAME> is a variable, it would assume that it is a function otherwise. KTurtle simplifies the above by referring to variables with a $ symbol. Thus to set a variable X to be 10 and then execute the same loop as before would be:

$X = 10

WHILE $X > 5 { ... }.

Both tools allow users to create and use functions, albeit in slightly different syntax. UCB Logo lets users define a function using the syntax

TO <FUNCTION> [:INPUT1 ...]

<FUNCTIONCODE>

END

where <FUNCTION> is a word, case-insensitive, and can have any number of input parameters. Then, to call this defined function, the user can simply type the name of the function followed by a list of parameters (if necessary). KTurtle instead uses the LEARN command and curly braces, such as:

```
learn <function> $input1, $input2 ... {
<functioncode>
}
```

In both cases functions are able to recurse, though this feature may not be realized by or suited for users learning the introductory concepts.

Users can generate random numbers in several ways in these two tools. UCB Logo allows for generation of random integers given a maximum number and, optionally, a minimum number:

random <minimum> <maximum>

It is important to note that when given both parameters the function returns an integer in the range, inclusive. When only given the maximum value, the function returns a random non-negative integer up to, but not including, the defined limit. KTurtle uses the same command name, but requires both the minimum and maximum value (separated by a comma). The generated number is chosen from the integers between the limits, inclusively. UCB Logo also can choose a random element from a list. One of only two examined tools to be able to do this in a unique command, the syntax would be:

pick <list> .

While some more advanced implementations of Logo allow for parallelism, UCB Logo and KTurtle do not.

Research findings

Even though the first implementation of the Logo language was finished in 1967 with the main goal of use in an educational setting, research into how well children learning the language grasp the core concepts of programming is limited. One study of fourth grade students learning Logo reported promising information about the chosen criteria. Conducted over the course of ten weeks in 2006, it examines the effect of weekly 45-minute sessions on 58 students' opinions and understanding of computer science and programming concepts [3]. In order to not require software installation on the computers used, the authors chose to use a web-based tool called VVLogo. During the study, the curriculum covered general computer science principles and several of the defined concepts of this project. Feedback collected before, during, and after the teaching modules reported little difficulty in the understanding of the concepts of interaction, repetition, conditional logic, and functions. In the post-test, the majority of students were able to successfully solve a problem incorporating these ideas. The concepts of variables, randomization, and parallelism were not stated to be included in the lessons. Beyond general feedback about computer programming concepts, data was also gathered that shows that Logo is achieving its goals in terms of lowering the barrier of entry and encouraging active learning. When asked to describe their opinion of computers and computer science, students responded with positive words 106 times, compared to unrelated/neutral words 43 times and negative words 11 times. Anecdotal evidence and server logs show that as early as the first session, students were seeking out and experimenting with commands not yet covered. Furthermore, records indicate that students continued to access the VVLogo server

for many months after the end of the course.

Other studies, while not reporting concrete feedback about the understanding of the introductory computer programming concepts, have shown positive indication that students readily grasp the turtle concept and begin identifying with the commands relative to its perspective. A main goal of Logo and its creator is lowering the barrier to entry of programming for younger learners, and from his writings the level of his passion for the idea is evident [10]. Research shows that this concept is achieved and can be traced partially to the instant feedback presented by the turtle's path as the code is executed. This feature is shown to help students recognize bugs in their code [12], which is always a good skill for potential computer scientists.

Individual exploration and summary

Since research about Logo and it's ability to convey introductory programming concepts remains mostly nonexistent or at the least not very accessible, a detailed personal exploration of the concepts presentation in the language is a crucial source of information. Through unguided attempts to write code and repeated searches of software manuals and syntax indexes, it was seen that most of the concepts are present and are called upon in a logical manner. The exceptions to this are interaction and parallelism. Interaction with the code during execution is very limited, and only one of the two tools presented a simple procedure for prompting input from the user. Output options are also limited, but this is expected due to the focus being on the canvas alone for display. Repetition and conditional logic are flexible and described in straightforward command names. Variables, while not being intuitive to discover without examples, are presented in a consistent fashion that became habit after three or four uses. Functions and randomization also required searches of the manual for example code but were memorable after discovery.

While this tool is the oldest among the seven studied, this tool provides an uncomplicated entry-point for budding computer scientists. The lines-of-text style of coding leaves risk for syntax errors, but this is simultaneously counteracted by the simplicity of the basic commands and the instant visual feedback traced by the moving turtle. One negative aspect is that the number of variants of the original language can have very different commands, but this may not be important if learners use just one and don't attempt to port code between different implementations. The dated interface isn't as polished as those of the remainder of the tools, but some versions allow more detailed turtle sprites and colors. Parallelism is lacking in entirety in the investigated tools, but may exist in other unexplored options.

4.2 StarLogo TNG

Presentation of concepts

StarLogo TNG addresses the computer programming concept of interaction more fully than its ancestor Logo. This tool retains the commands to move the turtles forward, back, left, and right in the Movements section of the code palette, but also includes further input options. Users can retrieve input from both the keyboard and joysticks during execution by dragging blocks such as KEYBOARD _?, JOY X-AXIS, or BUTTON _? from the Controls tab of the palette. Additionally, the user can access input from other controls like SLIDER (found in the Setup and Run tab) to create on-screen input areas. Output in StarLogo is presented through the viewport, where the results of the program can be seen during execution by the movements and behaviors of the three-dimensional characters. These characters can display text to the screen using the SAY command found under the Text tab. More detailed outputs are available in the Setup and Run tab, where blocks can be found to display clocks, monitors,

graphs, and tables of data values as the program runs.

Repetition is available in two different locations of the palette in StarLogo TNG. Simple repetition of code for a set number of cycles is accessed via REPEAT which is located on the Logic tab. Users can drag a number into the repeat count slot to control the number of cycles executed. For loops and conditional loops are not unique block structures, and are able to be simulated through infinite looping and conditional statements that will be described shortly. To create an infinite loop, users can utilize the FOREVER control block located in the Setup and Run tab of the palette.

StarLogo TNG includes a variety of conditional logic options by default. IF, IF-ELSE, and SWITCH blocks are found within the Logic tab. Also located in this tab are the blocks necessary to create comparison expressions, such as AND, OR, and NOT which can be combined into complex arrangements.

Variables are intuitively located on the Variables page of StarLogo TNG's palette. From here, users have the choice to create global or local variables of any of the supported types: boolean, number, text, and lists. Lists can hold an arbitrary number of elements of a single type. Global variables are denoted by SHARED while local variables are labeled AGENT. Once created, variables can be accessed and modified by switching the palette to the second page, titled My Blocks, where the existing variables are broken into different categories based upon ownership. To access the value, users drag the block with the appropriate label name, while setting of values and modification of value are available through SET and INCREMENT blocks at the same location.

Functions are able to be defined from the Procedure tab. Here, users can drag the PROCEDURE block onto the programming surface and give it a unique name. These procedures are able to accept parameters, which can be arranged by dragging the desired parameter type from the Procedure tab into the slot beside the function's

name. Functions can be terminated with EXIT PROC which returns no information, or with OUTPUT if data should be returned to the code in which the function was called. Once the function has been created, it can be referenced by navigating to the My Blocks page of the palette and dragging the chosen function name into place.

Randomization is possible through the creation of random numbers. This functionality is located under the Math tab of the palette, in the RANDOM code block. This can be inserted into any slot which would accept a number, and chooses a random number between 1 and the given maximum value. StarLogo TNG also natively allows for random selection of items in a list, which can be found as PICK LIST ITEM within the List tab.

StarLogo TNG provides easily accessible constructs for building parallelism into project code. Under the Setup and Run tab, any number of blocks can be dragged onto the programming surface and executed independently. Some of these code wrappers are: SETUP which executes once before the start of a project, or FOREVER and RUN which can be used to create parallel execution once the program begins. These threads can be synchronized by waiting for a shared conditional expression, but do not support communication except through shared (or global) variables.

Research findings

StarLogo TNG would seem to be used less frequently than the remainder of the tools from the availability of research into its effectiveness. However, the few published studies do shine some light on the tool and its applications with young learners. In one study, StarLogo TNG was compared with Logo to judge its effectiveness in reducing the barriers to entry for young programmers. The researches observed that the graphical environment increased the students' willingness to explore concepts on their own. The research does not show the effectiveness of the tool in conveying the

individual programming concepts, but it does emphasize the all-inclusive improvement of both interest and learning among students using the graphical user interface of StarLogo. From their conclusions, it is clear that StarLogo builds upon and is more effective than Logo for children in the range being examined.[17]

Individual exploration and summary

When exploring the StarLogo TNG tool, it is immediately apparent that the interface encourages exploration without guidance. All of the possibilities of the language are able to be quickly examined via the palette of blocks. This approach eliminated the need for external guides and materials, as were needed during the exploration of Logo. Interaction has a wide array of useful blocks that felt natural and intuitive. Repetition is readily accessible via the Repeat code block, and clearly shows the area of code to be run multiple times. The Logic tab of the palette grants quick access to simple conditional logic, but making compound logic expressions with And and Or was at times messy, and required several attempts. Variables were able to be created and used much more quickly than during the exploration of Logo, and the assortment of blocks involving them was easy to begin using once found. Using the Procedure tab to define functions required several failed attempts, but once they were defined, it was very easy to call them by name. Randomization of both numbers and choice of items of lists were straightforward and intuitively located in the Math tab. The ability to choose a random element of a list directly (and not involving indices) was a surprising feature, and would allow many uses of randomization in projects to be less complex. Parallelism is a very natural part of the environment, and since it is done by creating multiple different stacks of code blocks, it was already in use before that specific concept was targeted for study. Users may find themselves using parallelism without consciously deciding upon it.

StarLogo TNG is a clear improvement upon Logo in a number of ways. Firstly, the graphical environment and defined palette of code blocks leads to more interest and exploration of the tool. Furthermore, StarLogo presents all seven programming concepts unlike Logo. This tool is visually appealing with a wide variety of colorful blocks and characters, but the colors of the blocks seem arbitrary at times. The three-dimensional models and environments they exist in are more up to date with the graphics of videogames, and may inspire creativity.

Chapter 5

Scratch and Alice

Though developed by different teams for different goals, Scratch and Alice share a number of similarities with each other and also with StarLogo TNG. This chapter investigates the effectiveness of Scratch and two versions of Alice with regard to teaching the basic programming concepts and the impact upon the outlook of the youth learning to program. Scratch presents a cartoon-like combination of interface and existing sprites with all programming blocks being colorful and having shapes depicting the proper placement locations in relation to the other code within the scripts. Alice and Storytelling Alice, based in Java, showcase an object-oriented programming style more visibly than the other tools examined. While the programming interface is presented without much color, the available character models are colorful, often comedic, and designed with children in mind.

5.1 Scratch

Presentation of concepts

Scratch addresses the programming concept of interaction in a manner similar to StarLogo TNG. Input during execution can be through several channels: the keyboard, mouse, and even connected sensors. To access this functionality, the user drags the

appropriate block from the Control or Sensing tab of the block palette. Mouse clicking and keyboard presses are located in Control, and have dropdown boxes that can be changed to the desired key or button of the device. Mouse location detection and sensor input blocks are located in Sensing, and can retrieve information about the x,y coordinates of the mouse cursor along with a variety of sensor data such as light level, sound, tilt, and resistance if the necessary hardware is available. Output is primarily through the stage, where the sprites are animated during execution. The majority of animation commands are located in the Movement tab of the palette, including blocks that make sprites take steps in the direction they are facing, move toward a location on the stage, or rotate in place. More commands are located in the Looks, Sound, and Pen tabs, which include code to change sprite properties, display text, play sounds from the computer speakers, and control the tracing of sprite movement. Like Logo, Scratch also supports output to robotic devices through motors, though this again requires additional specialized hardware.

Repetition is handled by blocks in Control. Simple iteration for a set number of cycles is available via the REPEAT block. Code blocks that are to be iterated over are then dragged into the slot in the middle of this piece. Conditional looping is available in the form of either FOREVER IF or REPEAT UNTIL blocks. These conditions are assigned by dragging the desired expressions from the Operator tab in the palette, and iterated code is placed into the iteration block's slot in the same manner as REPEAT. For loops are not available as a pre-made block, but the functionality can be achieved with the existing iterators. Additionally, infinite looping is available as the FOREVER control block.

Conditional branching is found in the Control tab. There, the user can find both IF and IF-ELSE code blocks. As with conditional loops, the user defines the checked condition by dragging the desired expression blocks from the Operator tab. In the

IF-ELSE block, code placed into the upper slot is executed when the condition is true, and the code in the lower slot is executed when the condition is false. Case structures are not present in Scratch.

Variables are intuitively found in the Variables tab of the code palette. At the start of a project, the user is presented with only the options to make a variable and make a list. When clicked, a prompt window allows the user to define the variable or list name and whether it belongs only to the currently selected sprite or is available to all sprites. Once a variable is created, the name and current value will be displayed in the corner of the Stage. Additionally blocks will be available from the Variables tab that set or change the value, and toggle visibility of the variable on the Stage. Upon creation of a list, blocks become available to add, delete, insert, replace, and access items in the list.

Functions are available with limited features in Scratch. Multiple stacks of connected blocks can be placed on the programming surface at once. Through Control blocks, one stack can cause a second stack to execute by broadcasting a message (BROADCAST AND WAIT block) for which the second stack is waiting (WHEN I RECEIVE block). This allows some of the benefits of functions, but may not be intuitive and parameter passing is not supported directly. Workarounds for parameters such as global variables are possible, as is recursion, but the block arrangements may be too complicated for novice users.

Randomization in Scratch is found as a single block in the Operators tab. Using the PICK RANDOM block, a random integer can be chosen between user-entered minimum and maximum values, which are inclusive.

Parallelism is present in Scratch through placing multiple stacks of blocks. Any number of stacks can be executing at the same time, provided the Control block that the stack depends on is activated. Synchronization can be demonstrated as simply

having two stacks waiting upon the same condition. Communication between stacks is available through broadcasting and receiving messages, as discussed above regarding functions.

Research findings

Though Scratch was created only six years ago, there exists a wealth of information about its effectiveness in classroom and teaching situations. One important study was conducted at an urban after school program called the Computer Clubhouse and in Los Angeles [5]. This program primarily serves African American and Latino youth of a low-income area of the city, and offers activities such as playing video games, making music, photo editing, and 3D modeling. During the course of the 18-month study, a gender balanced group of 80 youth between the ages of 8 and 18 participated in the Clubhouse's first computer programming activity. The activity was structured to be completely open-ended, with the focus being on individual exploration rather than group teaching. Mentors (Graduate and Undergraduate students) were available, but had little or no computer programming experience. In the end, 536 unique projects were collected from the youth to be analyzed along with notes taken by observers present during the activity and interviews with the youth about their experiences. The collected projects were searched for the presence of introductory programming concepts very similar to those of this paper, and these concepts were ranked by the percentage of projects in which they were utilized. 111 projects contained no code, and were removed from the data pool, leaving 425 of value to be studied. Many of these projects gravitated towards video games. Of the examined projects, approximately 54% implemented user interaction of some kind and this figure remained consistent over time. Loops were a part of 52% of all projects, with later projects using loops significantly more than earlier ones. One in four projects

contained conditional branching, while one in ten projects used boolean logic both of these intertwined concepts increased in popularity over time. 9.6% of projects used variables to store values, with nearly ten projects using variables in the second half of the study, for every two projects in the first half. Random numbers were used very infrequently, and were represented in a mere 4.7% of the projects. A remarkable 88% of the analyzed scripts contained multiple stacks of instructions, demonstrating either use of functions or parallelism.

Another published study provides similarly hopeful results, building Scratch into a middle-school classroom setting [6]. The classes were not able to be chosen by students, thus giving a wide range of computer interest among the selected. A curriculum was designed for the study, focusing on different computer science concepts per module and arranged with the goal to present the most difficult topics last. Instead of analyzing projects for concept use, students were given tests at three stages of the study: before the course began, during the middle of the course, and upon completion. The post-test results demonstrate a correct understanding of conditional loops and bounded loops by 75% and 57.5% of students, respectively. The participants understood the concept of message passing 62.5% of the time, with the comprehension of variable initialization standing at a lower 17.5%. Variables and concurrency were the least understood concepts tested, with students expressing the correct meaning 7.5% of the time for each. The results for concurrency are surprising, when compared to the number of projects that used the concept.

Individual exploration and summary

Though significant studies have published information about the effectiveness of Scratch, a personal exploration was conducted thoroughly, as well. Interaction, repetition, and conditional logic were found to be immediately understandable, with the

organization of block locations and shapes feeling very natural. Parallelism felt intrinsic to the language, often being incorporated without realization. Randomization was missed upon first examination, and would have benefited from more separation from other blocks to show its difference. However, once located, using random integers became part of most continued scripts. Variables seem out of place in their current method of presentation, but that did not limit their usefulness. Functions were the least intuitive concept during investigation, with frequent revisions to code arrangement in order to produce desired results. Overall, the interface itself conveys a friendly atmosphere, and with the exceptions listed above exemplifies a polished final product. One discovered feature that proved to have the biggest importance was the help system. Right-clicking on any block allows the user to view a description of its functionality, as well as a brief example of correct usage.

Scratch is well-suited to understanding programming concepts via both individual learning and group-learning. In the studies described, none of the instructors were familiar with Scratch, and yet significant learning was evident. Combining the natural appeal of Scratch with better preparation, would almost certainly increase understanding of the introductory concepts. The ability to incorporate custom-made sprites into any project can distract users from the programming itself, but also gives each project a chance to be a more personal experience. Scratch is not without flaws as a tool, but thus far is the most inviting and encouraging learning environment of the seven chosen tools.

5.2 Alice

Presentation of concepts

Alice allows user interaction during execution through mouse and keyboard controls as well as input prompts. To reach keyboard/mouse input functionality, the user would select the virtual object to be interacted with, and then drag the desired code block from the tabbed palette in the lower left corner. The Procedures tab houses both addKeyListener and addMouseButtonListener blocks, but to generate an input prompt, the user would look to the Functions tab. Output, like most of the researched tools, is primarily visual/aural. The program execution manipulates the 3-dimensional environment and characters through dozens of included animations and procedures.

Repetition can be implemented by dragging the appropriate command from the row of control-flow blocks across the upper screen edge. Simple repetition of a code block for a set number of cycles uses the count block. After placing the block, the number of iterations can be set using the block's drop-down menu. For loops, while not a unique block, can be made using the for each in construct and giving it the set of numbers to step through. Conditional looping is presented by the while block alone. The condition statement is populated by drop-down menus containing values, comparators, and more. Infinite looping is available by choosing a while block and inserting the condition true, which is the first item shown in the menu.

Alice contains only one introductory level conditional structure, which is located among the control blocks. Though labeled if, the block is placed with slots for both the if-true and else code segments. The checked condition is entered using the standard drop-down menu.

Variables can be defined by dragging the control block local... onto the pro-

gramming surface. When placed, Alice prompts for variable name, type, and initial value. To avoid confusion between variable types, examples are shown beside each option. The variable can also be defined to be an array of objects by toggling a check-box, but no explanation is shown. Once created, each existing variable will be shown at the top of the screen in the control block area. Dragging the variable to the coding surface allows users to change its value. Inserting a reference to a variable within an expression is done by selecting the variable from the bottom of the standard drop-down menu.

Creating a new function can be done by clicking on the CLASS: dropdown menu directly below the control blocks, choosing which object is desired, followed by DECLARE PROCEDURE if nothing will be returned, or DECLARE FUNCTION otherwise. Both options show prompts for naming your function, with the latter also requiring that a return type be selected. Parameters are defined after the method is made, by activating the ADD PARAMETER button adjacent to the method's name. Recursion is possible, but relatively complex.

The programming environment allows randomization through navigating the familiar drop-down menu for the desired placement. Randomization options are show depending on context, with the ability to generate random booleans, integers, and real numbers. Inclusive and exclusive ranges are available to select and enter from the resultant menu.

By default, procedures are of layout DO IN ORDER, but parallelism is available through the control blocks labeled DO TOGETHER, EACH IN _ TOGETHER, and DO IN THREAD. The DO TOGETHER actions are performed by all objects sharing the given actions. EACH IN _ TOGETHER allows users to create a custom list of objects for which the actions will be performed. The last option, DO IN THREAD, allows code to be executed as a separate process, executing the chosen process while also continuing

to the code after the code block.

Research findings

Several research studies have gathered information about the effectiveness of Alice, particularly in terms of programming concept understanding, the appeal of the language to youth learners, and comparison to other introductory programming options. Alice presents an object-oriented programming experience, and has been compared to both C++ and Java.

One study of approximately 160 tenth grade students compares the experiences and learning of two groups: one half learning through Alice, while the other half is taught C++ [18]. The authors collected data regarding motivation, experience, and achievement which presents affirming results. The study does not state detailed learning data for each of the programming concepts, but is shown to cover a minimum of interaction, repetition, conditional logic, and variables. The authors make special note of the finding that looping was noticeably easier for the participants to grasp when using Alice. Upon completion of the program, students learning with Alice showed a significantly greater mean test score (approximately 7% higher). The authors attribute this difference to the reduction in time spent learning and debugging syntax, since Alice presents a much simpler interface to users. The motivation and experience surveys did not show a significant difference between groups, but interviews with the Alice group provide a positive reaction to the simplicity of the programming interface and the visual appeal of the tool as compared with C++, and a higher tendency for unprompted individual exploration of the programming environment. The participants also commonly pointed out that while the interface simplified many tasks, the pull-down menu system was a source of frustration.

A second study compares the experiences of students learning introductory pro-

gramming in Alice and Java [1]. While the participants were above the age being examined in this research project, the findings are relevant as a comparison to Alice's base language. Approximately 30 undergraduate students attending a community college participated, one class teaching Alice for the first 6 weeks of the course and then switching to Java, while the other class utilized only Java throughout the course. While little information was reported about the individual programming concepts, the author found that Alice resulted in much higher confidence levels in programming ability as a whole; users of Alice gained more than twice as much confidence as compared with the gains of the pure Java group. When asked about the transition to programming in Java, several participants remarked that their experiences with Alice made the switch easier. Additionally, Alice was found to encourage enrollment in further computer science courses.

Werner, Campe, and Denner present an additional study of the experiences of middle-school students learning programming through Alice [20]. Over one semester, 325 students participated in a game-programming course where data was collected about many introductory programming concepts. Upon analysis of 231 completed games, users showed a strong understanding of interaction, variables, conditional logic, and function use. Nearly 100% of students executed some style of user input in a bug-free fashion, and of the over two-thirds of the participants who created their own methods, more than 80% were error-free. Variables and conditional logic were used by approximately one third of the group, but showed promising success rates of nearly 90% when attempted. Parallel synchronization was present in almost two-thirds of games, with a 94% success rate. Results were less promising when the focus was shifted to repetition, where usage was below 20%. The authors present significant evidence that programming in Alice can result in a firm grasp of several introductory programming concepts.

Individual exploration and summary

Upon the start of exploration, Alice immediately felt more powerful than the other tools. This feeling was quickly tempered by the impression that there was little time spent polishing the interface and making navigation intuitive. Though targeting a similar age group as Scratch, the programming surface is very different. Noticeably absent are color-cues and different shapes; all programming blocks are rectangular and a light shade of blue/gray. A central feature is the context-sensitive drop-down box for writing expressions, which benefits from paring down the menu to only options relevant to the current placement and variable types. This is helpful, yet the remaining menu structure remains cluttered and confusing. Crucially, a tutorial system is present to guide new users through both the interface and many of the most basic programming concepts. Interaction and repetition are presented in a straightforward and descriptive style. Conditional logic is accessible, but suffers from the state of the pull-down menu system when entering the conditional expressions. Variables are not presented intuitively, with no clue about the function of the Local... control block being visible until dragged to the surface. Once accustomed to the quirks of the tool, the large number of standard animations for each object allows for complex scenes to be designed quickly and at a high level.

Alice shows signs of living up to its goals, eliminating syntactical errors in much the same manner as the already discussed visual programming tools. The tool presents the object-oriented style of programming in a manner which research is showing to be beneficial, especially when compared with other high-level language options. Though the majority of results are promising, the flaws of Alice also become evident. Variables and functions necessitate detailed explanation by an instructor familiar with the tool, and the interface may benefit from a redesign in which more tooltips and visual cues convey the meaning of the programming blocks.

5.3 Storytelling Alice

Presentation of concepts

Storytelling Alice, as a branch of Alice, is very similar in its presentation of the introductory programming concepts. The above details for Alice remain true, with the following exceptions.

Repetition in ST Alice receives a unique option for creating infinite loops, allowing the user to choose INFINITY to set how many times the standard loop will iterate. Declaring methods is easier to locate, having a dedicated button near the top of the screen. Communication during parallel execution benefits from custom events that can be run when a chosen variable's value changes. This can be useful for passing data between objects automatically, but may be too complicated to be valuable to beginning programmers.

Research findings

Being a variant of Alice, the most relevant research presented is a comparison between Storytelling Alice and regular Alice. A study at Carnegie Mellon University presented 88 middle-school girls with an opportunity to learn introductory programming through both Alice and ST Alice [4]. One group first followed a tutorial using Alice and wrote their own program, followed by a programming quiz and a survey, then were given 30 minutes to try Storytelling Alice. The second group followed a similar structure, with Storytelling Alice being the focus of the longer session and Alice following later. At the end of the day, all participants were allowed to choose a project to present from either version of Alice. The programming quiz taken between modules covered interaction, repetition, conditional logic, functions, and parallelism. The quiz results showed no significant difference between groups in terms of under-

standing. However, the study showed that when programming in Storytelling Alice, users spent 42% more time staying on-task programming instead of placing and customizing characters in the scene. Users of ST Alice were also three times as likely to use intermission time between lessons to sneak in extra work on their projects. When given a choice between the two versions at the end of the study, the participants overwhelmingly chose to take home the Storytelling version of Alice for future use.

Another study examines the use of Storytelling Alice by middle-school students during two weeks of summer courses [21]. This course focused on game programming in pairs, and the final products were collected to be analyzed. All collected programs successfully implemented interactive events. Only 17% of projects included repetition concepts, and one quarter of the games included conditional logic and branching structures. A greater number of programs included variables, which were present in approximately 40%, and nearly half contained user-written methods. Parallelism was built into just over 50% of projects as well. Beyond data about the individual concepts, the study also collected information about the confidence and experiences of participants wherein 90% of students enjoyed the experience, and nearly two-thirds found the creation of their game to be easy. Students also remarked that the drag and drop interface was very useful, and the focus on game creation helped make the material interesting.

Individual exploration and summary

This tool is an improvement on Alice in a number of ways. One of the most important differences is the high quality and extensiveness of the included tutorial system. Complicated actions are shown step-by-step, walking users through the process with a combination of explanatory text, arrows, and masks that hide areas of the interface that are irrelevant for the current task. The presentation of the basic concepts is

nearly identical to the standard version of Alice, but the presence of this detailed guide eases the learning process significantly. Another improvement is seen in the number of standard objects, and this is compounded by several unique animations for each one. These improvements come at the cost of platform-independence, limiting this tool to only machines that run Microsoft Windows.

This tool shows substantial improvements over standard Alice among middle-school girls, but similar data for boys is unknown. However, since the modifications to Alice are almost entirely additive, and do not change much of the basic structure of the tool, it is unlikely that standard Alice would be an improvement to Storytelling Alice for the male population. Overall, Storytelling Alice shows promising results of reducing the barriers to entry for young programmers, and bolstering the visual appeal and motivation to learn.

Chapter 6

Kodu Game Lab

6.1 Kodu

This chapter examines the final remaining tool, the Kodu Game Lab. Alone in the capability to create programs on a computer without requiring a keyboard and mouse, Kodu differentiates itself from the other tools in a number of ways. Allowing users to design programs using a video-game controller greatly reduces the number of input options, with two joysticks and ten buttons comprising the full control scheme. Kodu is also entirely event driven, with scripts relying solely on sets of conditions and matched actions to produce the desired result. As objects and characters are added to the environment, each gets up to 12 pages of script space, with six rules fitting on each page [8].

Presentation of concepts

Interaction in the Kodu Game Lab is through the joypad, but can also be controlled through a mouse and keyboard when using a Windows machine. Users can receive input by first adding the desired device to a blank line of a script, then selecting the exact input button from the available options. Once this is completed, the second half of the line can be created to give the input purpose such as moving a character

on the screen. An example of this could be expressed as When Left Joystick, Move Robot, which infers that the robot should move in the direction in which the joystick is being pressed. Output is entirely visual/aural, with the program displaying the movement, behavior, and sounds of characters in the 3-dimensional environment.

Repetition is performed by default in Kodu as the system continually monitors each event condition. This mostly removes the ability of the user to implement looping structures, but limited functionality remains in the form of script pages. Users can create cycles by programming conditions that switch between different pages of code. For example, one page could be focused on directing a character towards an apple, and once found, switch to another page of scripts that handles what the character should do with the found apple. This second page can include a line to switch back to the first page after some event, allowing the process to repeat.

Conditional logic is central to Kodu's rule-based programming. Every line of a script is a rule, and each rule has a condition and an action. This can be seen in a script which serves as Kodu's Hello World, When See Apple, Move Toward. If-else structures are possible as well, for example:

When See Apple, Move Toward
When See Apple Not, Wander Quickly.

Kodu allows for both global and local variables. Global variables are available as predefined scores, which store integer values. Scores can be incremented, decremented, and set to completely new values by selecting the variable in the action half of a rule and then choosing the desired procedure from its circular display of options.

While functions are not truly available in Kodu, they can be simulated using script pages. For example, a program could have a main script page which watches for an event, switches to a different page to execute several commands, update a variable, then return to the main page upon completion. However, parameters are not able to

be passed between pages, and would require use of specified global variables.

Randomization is available at any location where the user can enter a number. To generate a random number, the user selects Random and can then choose one or two numbers. In the event only one number is chosen, the tool will generate a random integer between it and 1; otherwise, an integer will be chosen from the selected range inclusively.

Parallelism exists by default within Kodu. Each character or item on the screen maintains its own scripts, which all are continually checked for the event conditions. Synchronization between different characters is possible through shared events and script page switching. Communication between characters can utilize the many built-in events such as SEE, HEAR, and BUMP, or through variables and variable-checking conditions.

Research findings

One of the newest tools researched, limited information is accessible regarding the effectiveness of the Kodu Game Lab for teaching computer programming concepts to youth. One published study examined over one month of usage data from the PC version of Kodu, and collected 346 user-created programs from the Xbox version [13]. No demographic data was collected from users, but anecdotal evidence shows that many users fall within the desired age range. Analyzing the collected data, users spent only 15% of their total time writing scripts for their projects, significantly less than the 23% of time spent playing their creations. Input and output were present in nearly all of the examined projects, and repetition created by users was a part of almost 50% of games. Conditional logic (not counting the standard event-action structure of the language) was included in only 8.4% of projects. Variables were very common within the analyzed games with between 60% and 80% of submissions having global and local

variables, respectively. Data for function usage was not collected, but may be related in some way to the 58.4% of projects that implemented multiple states of characters using multiple script pages. Randomization using random number generation was present in slightly more than one-third of projects. No direct information about parallelism was gathered, but usage can be assumed to be common by noticing that the average number of characters was greater than five per game, and nearly three-fourths of all characters present contained some programming evidence.

Individual exploration and summary

Kodu is the only tool studied available on a current gaming console, and the impact this environment has on the the interface is immediately noticeable. No typing is necessary for programming, the characters and world are created through joystick or mouse input. The resultant code selection wheel is kept manageable through context-sensitive options, showing only the code blocks appropriate for the position at which they are being inserted into the program. Organization of the circular selection tree is polished and organized intuitively; creating multiple characters with unique scripts is a quick process. However, the event-based nature of Kodu's programming language can make user-defined repetition, functions, and parallelism overly complicated.

Kodu is a very polished and user-friendly tool for programming and game design, and the research conducted thus far suggests that it can be useful in the process of learning about the chosen computer programming concepts. However, the event-based nature of the tool may automate repetition and conditional logic to the extent that users may not even realize they are being implemented. This tool is the most game-centric of the seven, both in terms of resultant projects and the feel of the interface, but would appear to be less effective than many of the other tools which allow for more user-level management of each programming concept.

Chapter 7

Effects of Gender

Several of the studies examined in this project included in their data information about the effects of the tools and curricula on each gender of the learning groups. While many tools effected learning from both genders equally, either expressed directly or by the omission of any differences when discussing gender, others contained aspects that were of more interest to a specific gender. This information may be important to consider when deciding which tool to pick for teaching the core programming concepts.

7.1 Similarities

The research examined shows that both girls and boys prefer, and learn the concepts of computer programming more readily in, the more graphical tools such as Scratch and Alice. Both genders were more likely to explore the capabilities of these tools, and several of the studies make light of the students' desire to continue using the tool after the studies ended [19]. Students using the more visual tools were less likely to associate their creations with the act of computer programming, which is concluded to help prevent students from being affected by any negative perceptions of computer programming. This may also be useful in disconnecting the learning process from the

idea that computer programming is something for boys only.

7.2 Differences

The most obvious difference between the learning experiences of the different genders is not directly related to the exploration and understanding of the core programming concepts, but in the direction that the students take when designing scripts and programs. In one study of Alice and Storytelling Alice, the researchers point out that the girls involved tend to create visual stories or scenes [4]. Boys, however, are shown to more often put more emphasis on user interaction and, in many cases, chose to design interactive games. As such, Storytelling Alice was more interesting to the female students, while the male students showed no preference. This difference is apparent even in studies that do not directly focus on the gender differences. In the examples of programs created at the Computer Clubhouse, the program written by a female student was an animation inspired by a music video [5]. The example program written by a male student was an interactive side-scrolling game. These examples alone are not enough to denote a trend, but in each investigated study where student programs are explained or shown, this division is evident. The underlying cause of this difference may be related to societal gender roles and may change over time, but it is important to note that even though the types of creations are often different for each gender, the creations display an understanding of the core programming concepts.

Chapter 8

Conclusion

8.1 Results

While there is no one best tool for every situation or use, many of the tools show unique features that they execute well or poorly. Computer Science Unplugged is a worthwhile tool for teaching basic concepts of computer science, but does not convey programming concepts with similar ease. CS Unplugged fails to convey the concepts of functions, randomness, and parallelism without special attention from the leader of the activities. This tool has potential for outreach programs that do not have access to computers, but requires an informed instructor and a group of learners.

Logo stands alone as the oldest tool in the set, and the only tool that focuses on abstract programming instead of visual programming. As such, more time must be spent educating students on the available commands that can be entered, as well as their proper syntax. Nonetheless, Logo shows signs of achieving its goals of reducing complexity and barriers to entry, and many students begin associating with the turtle-programming idea quickly. Logo presents all seven of the basic programming concepts, except parallelism, in a straightforward manner that more closely matches the common languages such as C++ and Java that would be the next step. Logo may be the best tool for a low-overhead computing experience, as long as the students are

familiar with using the computer keyboard as a primary means of input, and many different implementations are available for many different systems.

StarLogo TNG presents computer programming in a visual manner, with multi-colored blocks linking together in specific patterns and removing the risk for syntax errors altogether. The arrangement of blocks onto the programming surface allows for quick recognition of missing/incorrect code and the layout of the tool is convenient after an adjustment period. Interaction, conditional logic, variables, random numbers, and parallelism are all presented in a manner that studies are showing to be effective, however, repetition options are lacking and are confusingly located in multiple locations. The stand-out feature of StarLogo TNG is its presentation of functions. Of the seven, this tool allows for the easiest creation of procedures that accept input parameters and return values.

The third tool investigated, Scratch, stands out as one of the best. This tool has been the focus of a number of studies, all of which show a high effectiveness at conveying the basic computer programming concepts to youth learners. The interface is organized into four simple sections, with uniquely shaped code blocks giving visual feedback during programming. Scratch presents each concept clearly except for the concept of functions. This tool does not allow for parameters or return values without a round-about solution. The key features of the tool are the human readable titles of each code block as well as the built-in help system. Once discovered, context-sensitive help menus provide the best experience for independent exploratory learners, removing the necessity of instructors familiar with the tool.

Alice and Storytelling Alice are very similar and seem to share the same results pertaining to effectiveness. These two tools present a well-rounded array of code blocks that are given titles descriptive enough to be understood at a glance. These tools are highly object-oriented, and are shown to reduce the difficulty in transitioning

to the more standard object-oriented programming languages such as Java and C++. Though the interface is cluttered at times, the wide array of standard characters and animations are shown to encourage creativity and motivation among students. Storytelling Alice comes packaged with a thorough tutorial system, and its larger availability of built-in procedures may appeal more to female students, but more study is necessary to be conclusive.

The final tool examined, Microsoft's Kodu Game Lab, presents the most streamlined interface, and also the only implementation on both personal computers and a gaming console. The streamlined design has negative effects on its presentation of the basic computer programming concepts, however, and several concepts such as repetition, looping, and functions are handled mostly automatically or are otherwise unavailable. Kodu may be the best suited towards simple video game design and have the most refined interface, but it falls short when the focus is on teaching the seven chosen computer programming concepts.

8.2 Future Work

The gaps in existing research leave much to be desired, and there are many opportunities for continued research into the effectiveness of these tools. One direction in which this research could be focused is to compare these tools side-by-side in the same conditions. The array of research studies used by this project are useful, but cannot be directly compared with each other due to the varying populations of learners and curriculum styles. Additionally, more work could be conducted on tools to improve upon their weaknesses. Just as Storytelling Alice aimed to fix some problems present in Alice, modifications could be made to the interfaces of existing tools and compared against the original tools. The last, and potentially most exciting direction for further research, could be to create a brand new tool combining the best features of each.

8.3 Final Remarks

As a computer scientist who began learning to program in BASIC when I was approximately the age that this study applies to, I cannot help but to be impressed with the overall quality of these teaching tools. I was one of the students who stayed with it even though it was complicated, but several of my friends found the topic too challenging and did not continue learning more about computer science and programming. When asked why they found the class too difficult, almost all cited trouble mastering the syntax of the language. Tools like the ones studied here provide a way for learners to begin programming without memorizing syntax, and hopefully the use of these tools paves the way for more students learning about computer science.

Bibliography

[1] Tebring Daly. Minimizing to Maximize: An Initial Attempt at Teaching Introductory Programming using Alice. *J. Comput. Sci. Coll.*, 26(5):23–30, May 2011.

[2] T. Bell et al. Computer Science Unplugged: School Students Doing Real Computing Without Computers. *JACIT*, 13:20–29, 2009.

[3] Daniel Frost. Fourth Grade Computer Science. *SIGCSE Bull.*, 39(1):302–306, March 2007.

[4] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 1455–1464, New York, NY, USA, 2007. ACM.

[5] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by Choice: Urban Youth Learning Programming with Scratch. *SIGCSE Bull.*, 40:367–371, March 2008.

[6] 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.

[7] Microsoft Corporation. Microsoft Research FUSE Labs - Kodu Game Lab, 2011. http://fuse.microsoft.com/page/kodu.

[8] Microsoft Corporation. Kodu - Microsoft Research, February 2012. http://research.microsoft.com/en-us/projects/kodu/.

[9] MIT Scheller Teacher Education Program. StarLogo TNG — MIT STEP, December 2011. http://education.mit.edu/projects/starlogo-tng.

[10] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1993.

[11] Deborah Seehorn, Stephen Carey, Brian Fuschetto, Irene Lee, Daniel Moix, Dianne O'Grady-Cunniff, Barbara Boucher Owens, Chris Stephenson, and Anita Verno. CSTA K-12 Computer Science Standards, 2011. http://www.csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf.

[12] Cynthia J. Solomon. Teaching Young Children to Program in a LOGO Turtle Computer Culture. *SIGCUE Outlook*, 12:20–29, July 1978.

[13] Kathryn T. Stolee and Teale Fristoe. Expressing Computer Science Concepts through Kodu Game Lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 99–104, New York, NY, USA, 2011. ACM.

[14] Rivka Taub, Michal Armoni, and Mordechai Ben-Ari. Cs Unplugged and Middle-School Students Views, Attitudes, and Intentions Regarding CS. *Trans. Comput. Educ.*, 12(2):8:1–8:29, April 2012.

[15] Rivka Taub, Mordechai Ben-Ari, and Michal Armoni. The Effect of CS Unplugged on Middle-School Students' Views of CS. *SIGCSE Bull.*, 41:99–103, July 2009.

[16] Hidekuni Tsukamoto, Hideo Nagumo, Yasuhiro Takemura, and Kenichi Matsumoto. Comparative Analysis of 2D Games and Artwork as the Motivation to Learn Programming. In *Proceedings of the 39th IEEE international conference on Frontiers in education conference*, FIE'09, pages 1613–1618, Piscataway, NJ, USA, 2009. IEEE Press.

[17] Kevin Wang, Corey McCaffrey, Daniel Wendel, and Eric Klopfer. 3D Game Design with Programming Blocks in StarLogo TNG. In *Proceedings of the 7th international conference on Learning sciences*, ICLS '06, pages 1008–1009. International Society of the Learning Sciences, 2006.

[18] Ting-Chung Wang, Wen-Hui Mei, Shu-Ling Lin, Sheng-Kuang Chiu, and Janet Mei-Chuen Lin. Teaching Programming Concepts to High School Students with Alice. In *Proceedings of the 39th IEEE international conference on Frontiers in education conference*, FIE'09, pages 955–960, Piscataway, NJ, USA, 2009. IEEE Press.

[19] Brett Ward, Daniela Marghitu, Tim Bell, and Lynn Lambert. Teaching Computer Science Concepts in Scratch and Alice. *J. Comput. Small Coll.*, 26:173–180, December 2010.

[20] 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.

[21] Linda Werner, Jill Denner, Michelle Bliesner, and Pat Rex. Can Middle-schoolers use Storytelling Alice to Make Games?: Results of a Pilot Study. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 207–214, New York, NY, USA, 2009. ACM.

[22] Jeannette M. Wing. Computational Thinking and Thinking About Computing. *Phil. Trans. R. Soc. A*, 366, October 2008.