A Study of Computational Thinking Skills and Attitudes Towards Computer Science with Middle School Students

Lorien W. Cafarella

Follow this and additional works at: https://scholarcommons.sc.edu/etd

Part of the Curriculum and Instruction Commons

Recommended Citation

This Open Access Dissertation is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.
A STUDY OF COMPUTATIONAL THINKING SKILLS AND ATTITUDES TOWARDS COMPUTER SCIENCE WITH MIDDLE SCHOOL STUDENTS

By

Lorien W. Cafarella

Bachelor of Science
University of Mary-Hardin Baylor, 1995

Master of Education
Western Governors University, 2017

Submitted in Partial Fulfillment of the Requirements
For the Degree of Doctor of Education in
Curriculum and Instruction
College of Education
University of South Carolina
2023

Accepted by:
Lucas Lima de Vasconcelos, Major Professor
Michael Grant, Committee Member
Hengtao Tang, Committee Member
Anna Clifford, Committee Member
Cheryl L. Addy, Interim Vice Provost and Dean of the Graduate School
DEDICATION

This dissertation is dedicated to my family and friends. To my husband for supporting me through the good days and the bad. To my friends who always encouraged me when self-doubt took my energy. To my family, near and far, who proudly announced that I would be a Doctor someday and knew I could do this. You are my tribe, thank you.
ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my professor, Dr. Lucas Lima de Vasconcelos. Your encouragement, support, and patience were felt throughout every step of the process and I will be forever grateful. Next, I would like to thank Dr. William Morris and the University of South Carolina for paving the way for this dissertation. A big thank you to the entire Cohort QWERTY, your support was instrumental to my success. Last, but definitely not least, my husband for understanding the hours of reading and writing were necessary and allowed me the time to work on my assignments, and for providing a hug or motivation whenever I needed it. Without the support from everyone, this farfetched dream would not be a reality.
ABSTRACT

Many middle school students come to Computer Science (CS) classes without any previous CS instruction or any Computational Thinking (CT) skills that are needed to be successful. To overcome this, many curriculums and programming environments have been used to engage students and to instill a love of CS. Which curriculum should be used to develop CT skills that will also increase positive attitudes towards CS so students will continue in this field of study in higher grades? The purpose of this mixed method action research study was to evaluate how Code.org’s block-based programming curriculum affects middle school CS students’ CT skills, and their attitudes towards CT and CS. This study explored the following two research questions: (1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills? (2) How and to what degree were the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?

This study implemented a game design curriculum in Code.org’s block-based programming environment over the course of 11 weeks. Participants included 16 eighth grade students at a middle school in South Carolina. The qualitative data collected were student interviews and the quantitative data were pre- and post-tests, surveys, and student artifacts. Qualitative data analysis included inductive thematic analysis. Quantitative data analysis included descriptive statistics and paired samples t-tests. Students’ artifacts showed that students learned how to code in Code.org’s block-based programming environment. There was a statistically significant increase in
participants’ algorithmic thinking, debugging, and pattern recognition skills. The increase in participants’ abstraction skill was not statistically significantly different. Qualitative themes revealed that participants enjoyed block-based programming as evidenced by their references to the elements of game design during the interviews, but the quantitative findings revealed that students’ attitudes towards CT and CS did not have any statistically significant increase. This may be due to the students’ performance levels and that not all students perceive coding as easy. Future directions for research and practices are discussed.
TABLE OF CONTENTS

Dedication................................................................................................................iii
Acknowledgements........................................................................................................iv
Abstract .........................................................................................................................v
List of Tables .................................................................................................................x
List of Figures ...............................................................................................................xi
Chapter 1 Introduction ...............................................................................................1
    National Context ........................................................................................................1
    Local Context ............................................................................................................4
    Problem Statement .................................................................................................5
    Purpose of Study .......................................................................................................6
    Research Questions .................................................................................................6
    Statement of Research Subjectivities and Positionality ........................................6
    Definition of Terms .................................................................................................8
Chapter 2 Literature Review .......................................................................................11
    Theoretical framework of Constructionism ............................................................12
    CS and STEM Education in Middle School ............................................................13
    CT and Assessing Middle School Students .............................................................15
    Students’ Attitudes Towards Coding and CS ........................................................21
    Instructional Strategies for CT in Middle School .....................................................25
    Chapter Summary .................................................................................................29
Chapter 3 Method .......................................................................................................30
Purpose..................................................................................................................30
Research Design..................................................................................................30
Setting and Participants......................................................................................31
Intervention..........................................................................................................33
Data Collection ..................................................................................................47
Data Analysis ......................................................................................................54
Procedures and Timeline....................................................................................56
Rigor and Trustworthiness..................................................................................58
Plans for Sharing and Communicating Findings .............................................60
Chapter 4 Analysis and Findings .....................................................................65
  Quantitative Analysis and Findings .................................................................66
  Qualitative Analysis, Findings, and Interpretations........................................78
  Qualitative Themes ..........................................................................................83
  Chapter Summary ............................................................................................89
Chapter 5 Discussions, Implications, and Limitations......................................90
  Discussion .........................................................................................................90
  Implications ....................................................................................................99
  Limitations .....................................................................................................101
  Conclusion ......................................................................................................102
References .........................................................................................................104
Appendix A: CS Altitudinal Survey .................................................................118
Appendix B: Pre- Post-test .................................................................................123
Appendix C: Interview Protocol ........................................................................136
Appendix D: Game Creation Rubric .................................................................138
Appendix E: Student Assent Form .................................................................................. 140
Appendix F: Parent Consent Form ................................................................................ 142
Appendix G: Student 164894 Artifact ......................................................................... 144
Appendix H: Student 192880 Artifact ......................................................................... 149
Appendix I: Table of Round One Coding Cycle List ................................................ 153
LIST OF TABLES

Table 3.1 Intervention Schedule .................................................................35
Table 3.2 Data Sources ..............................................................................48
Table 3.3 Pre/Post-Test Questions and Origins ........................................49
Table 3.4 Reliability Analysis Results Considering the Whole
Scale and its Factors..................................................................................50
Table 3.5 Survey Questions Composite Scales ..........................................51
Table 3.6 Interview Questions Alignment to Research Questions ..............52
Table 3.7 Research Questions, Data Sources and Methods of Analysis ........55
Table 3.8 Procedures and Timeline .............................................................57
Table 4.1 Reliability Analysis of Pre- and Post-test Overall,
Subscales, and Test Questions Breakdown................................................67
Table 4.2 Descriptive Statistics Pre- and Post-Test Skills Scores ...............67
Table 4.3 Shapiro-Wilk Normality Test .......................................................69
Table 4.4 Paired Samples T-Tests ...............................................................70
Table 4.5 Survey Questions Focus and Composite Scales .........................71
Table 4.6 Descriptive Statistics-Pre- and Post-Survey Composites ..........72
Table 4.7 Shapiro-Wilk Normality Test Survey ..........................................74
Table 4.8 Paired Samples T-Test Survey and Composites .......................74
Table 4.9 Rubric Points and CT Focus .......................................................75
Table 4.10 Descriptive Statistics-Artifact Rubric Scores ..........................76
Table 4.11 Interviewee’s Demographic Information.................................79
Table 4.12 Codes, Categories and Themes

83
LIST OF FIGURES

Figures 3.1 Teacher Dashboard .................................................................34
Figures 3.2 Teacher Feedback and Grading .............................................34
Figures 3.3 Lesson 4 Instructional Example.............................................37
Figures 3.4 Lesson 9 Instructional Example.............................................38
Figures 3.5 Mini Project Lesson 10 Participant Example............................39
Figures 3.6 Lesson 15 Instructional Example.............................................39
Figures 3.7 Mini Project Lesson 17 Participant Example............................41
Figures 3.8 Lesson 18 Instructional Example.............................................42
Figures 3.9 Mini Project Lesson 20 Scroller Game....................................43
Figures 3.10 Lesson 22 Instructional Example...........................................44
Figures 3.11 Mini Project Lesson 23 Flyer Game.......................................45
Figures 3.12 Final Project Lesson 27 Participant Example..........................47
Figures 4.1 Post-test Score Distribution ...................................................68
Figures 4.2 Algorithmic Thinking Skills Score Distribution.........................69
Figures 4.3 Participant Attitudes- Pre- and Post-Survey ............................73
Figures 4.4 Computational Thinking Beliefs-Pre- and Post-Survey ...............73
Figures 4.5 Artifact- Overall Scores Lowest to Highest............................77
Figures 4.6 Artifact- Algorithmic Thinking Scores Distribution..................77
Figures 4.7 Artifact- Abstraction Scores Distribution.................................77
Figures 4.8 Artifact- Debugging Scores Distribution.................................78
Figures 4.9 Artifact- Pattern Recognition Scores Distribution.....................78
Figures 4.10 Complete List of Categories and Codes included..........................81
Figures 4.11 Categories with Descriptions ..........................................................81
CHAPTER 1:
INTRODUCTION

National Context

Students have had limited exposure to Computer Science (CS) and in Computational Thinking (CT) skills in K-12 schools (Brown et al., 2014), hampering their ability to create and understand CT. While programming is a skill not synonymous with CT skills, programming is a benefit of one thinking computationally (Shute et al., 2017). Since the 1960’s, CT has been discussed, but it is currently a “focus of educational innovation, as a set of problem-solving skills that must be acquired by the new generations of students to thrive in a digital world full of objects driven by software” (Román-González et al., 2017, p. 1). The revival in teaching CS is attributed to three concerns: The first is an economic drive for preparing the future workforce, students should not only be consumers, but also producers of innovation, and knowledge and skills obtained from learning CS and CT are necessary for students (Popat & Starkey, 2019).

With the current revival of CS in education, there is also a revelation that are not enough qualified teachers to teach it. In the past couple of decades, schools were teaching students how to use computers and not to create (Kafai, 2016; Runciman, 2011). The Computer Science Teachers Association (2019) made several recommendations to address the lack of teachers in the State of Computer Science Education Equity and Diversity executive summary. Such suggestions include creating pathways for teachers to
become CS endorsed, funding for professional development, and offering CS courses to all preservice teachers (Computer Science Teachers Association, 2019).

Computational Thinking (CT) is a combination of skills comprising of decomposition, iteration, algorithms (algorithmic thinking), abstraction, debugging, problem solving or generalization (Shute et al., 2017). Decomposition is the ability to break down big problems into smaller, manageable problems. Iteration is the ability to analyze and look for repeating sequences. Algorithmic thinking is the ability to create step-by-step directions on how to do something. Abstraction is the skill to remove unnecessary parts of a problem and make one solution work for multiple problems. Debugging is to detect and correct errors when a program does not work. Problem solving or generalization is the ability to recognize a problem, plan for a solution, test and reflect on success and failures to decide if the problem is solved or another problem is found (Shute, Sun, & Asbell-Clarke, 2017).

Wing (2006) stated that CT skills are fundamental skills that all students need to be successful (Wing, 2006). CT is an essential part of learning for all ages and should be incorporated into the school curricula (Runciman, 2011). Policymakers and companies have supported this idea and have created many programs and curricula to teach CT skills, emphasized this with coding or programming. Such program-curricula consist of Code.org’s CS Discoveries, Microsoft’s MicroBits, MIT’s App Creator, CS First by Google, Codesters an online Python platform, Code Combat, Foundations for Advancing Computational Thinking (FACT) curriculum, and others. Furthermore, there are also many unplugged sites that offer lessons and activities to teach CT without using a computer, such as csunplugged.org, Code.org, digitalschoolhouse.org/uk and kodable.com.
Since the launch of Code.org in 2013, there have been over 900 million hours of code completed by students and over one million teachers using the site to teach CS and CT skills around the world (Dixon, 2020). In Code.org’s annual report titled *The State of K-12 Computer Science*, all fifty United States adopted the new policies to incorporate CS in K-12 standards in 2020 (Dixon, 2020). The site also supports sixty-seven languages, this report shows the growth of CS materials and the desire to teach CS and CT to the world’s youth.

CT is at the heart of all STEM disciplines and is essential to create rather than consume technology. Ketenci, Calandra, Margulieux and Cohen (2019) reported that the relationship between learner characteristics and learning outcomes; where prior experience, self-efficacy, and interest in the content were valid measures for students’ growth in CT (2019). Students need to be interested in CS and CT to be successful. To achieve this, the curricula must include projects and activities that are interesting to them. Additionally, there is also a need to assess these CT skills to create valuable and innovative curricula. Without an assessment to validate learning, CT skills cannot be successfully or effectively taught (Román-González et al., 2017). Deeper learning is increasingly seen as an imperative for helping students develop robust, transferable knowledge for 21st century CT skills (Grover et al., 2015).

The use of block-based programming environments (BBPE) is considered the best way to teach introductory programming and CT to younger and middle grades. Weintrop and Wilensky (2019) determined that these environments are more prevalent in K-12 education. Also, that educators believe the concepts learned and developed using these tools will prepare learners for future computer science learning in conventional text-based programming languages (Weintrop & Wilensky, 2019).
Local Context

Over the past few years, there has been a change in what courses are considered when teaching CS. With this increase in curricula options this has increased the availability for students in middle and high schools across the country. In 2018, the South Carolina Department of Education created a new set of state-wide CS standards, including an Introduction to CS course as a requirement for high school graduation. As well as created a new set of state CS standards that did not exist before 2018. The state also included a set of updated CS standards for K-8, including CS in all content areas (South Carolina Department of Education, 2018). Before this, the schools decided what, if any, CS was to be taught in their schools. This decision was based on teacher availability and student interest.

The current plan in Charleston County School District is to offer middle school students an introductory CS course to fulfill the high school requirement, CS Discoveries course offered by Code.org was chosen. This course encourages students to first learn what problem solving is and to create solutions using programming in web development, games and animation, and the design process with app creation. Java Script using a block-based programming environment eases student into the creation of real-world solutions to problems they feel are important to them to solve (Weintrop & Wilensky, 2019). The process of using problem-solving techniques with technology to solve a personal problem is expected to have a long-lasting effect on students’ CT skills (Korkmaz, Çakir, & Özden, 2017). The Massachusetts Institute of Technology App Creators and the PLTW App Creation course also use Java Script BBPE to create working apps for Android mobile devices that students can share with others. The examples above show that middle students can create real-world, shareable products.
without being bogged down in learning syntax. Creation is a powerful tool to encourage problem-solving and to learn CT skills. (Korkmaz et al., 2017)

In 2018, there were 200 middle school students enrolled in the CS Discoveries class offered at CE Williams. All students completed a pre- and post-test to assess their learning of CS and CT concepts and how they felt about those concepts. The percentage of correct responses for the concepts of understanding variables, identifying conditionals, and loops increased by more than 20%. Other concepts such as sprites, sprite placement, and how a computer works had smaller increases between 5-10%. Likert style questions were included, and two responses stood out with the largest increase. The answers ‘I have the ability to do CS’ and ‘I’m good at CS’ increased from 13% to 20% on both responses. These data show that we are beginning to help students learn and gain confidence about CS and CT skills.

Problem Statement

Students have limited skills in Computational Thinking (CT). This hampers their abilities in creating and understanding computer science (CS) and programming. As a middle school CS teacher, I have seen first-hand students lacking basic CT skills needed to complete simple programming tasks. Shute et al., (2017) stated that “the rapid onset of computers in the 20th century is forcing an analogous revolution where digital literacy is now an essential skill to succeed in our complex, digital 21st century world.” (p. 143) As well as lack of CT skills, student have negative attitudes towards CT and CS, perceiving it as hard and not fun. With inadequate access to CT and CS in younger grades, this can be the reason for negative attitudes and limited skills. The need to teach CS to students using a block-based programming environment that will integrate positive learning opportunities, impact on student attitudes, and increase their CT skills is imperative.
Purpose of This Study

The purpose of this action research study was to evaluate how a block-based programming curriculum in game design affects middle school CS students CT skills, and their attitudes towards CT and CS at CE Williams Middle School.

Research Questions

(1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills?

(2) How and to what degree are the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?

Statement of Research Subjectivities and Positionality

I am a Caucasian, middle-aged, female, middle school technology/computer science teacher. I have been teaching these subjects for 23 years in four U.S. states. I have taught many CT concepts in these classes, such as coding, robotics, video creation, audio editing, graphic design, computer art, Microsoft Office Suite, and all things Google. Over these many years, my coworkers have asked me for help in all areas of technology, and I am happy to oblige. I have presented at my schools, districts, and state-level technology conferences on several technology topics. In 2017, I completed my Master of Education in Technology and Learning, where my Capstone was on teacher technology training for a group of coworkers as participants. They were quite surprised at how much technology there was to add to their curricula, and many increased their incorporation and knowledge of technology for their subject areas.

Having taught in four different states, I have seen how the state standards differ from one another, and I have needed to adjust my curriculum to fit them. I have also seen how the students are different. In Florida, I taught at a Title 1 school where 95% of the
students were on free or reduced lunch. These students had nothing and loved my class because I let them use technology. They never had such opportunities at home or in other schools. However, I have taught at schools in South Carolina with 5% of the students free and reduced lunch. These students could be difficult, because they had better technology at home and did not care what was taught unless it was fun. Teaching in these opposite communities is daunting, and I believe these experiences have made me a better teacher in my subject area.

My worldview is pragmatic, and that focuses on what works rather than what might be considered true or real (Frey, 2018). Being a computer teacher, what works and logic are how I think, how computers work, and how we program and debug to make them work. These are important aspects of CT and CS in general. A pragmatic study is focused on evaluating and changing aspects of real-world educational experiences for a rich understanding of the research (Frey, 2018).

My positionality as a teacher/researcher is that I oversee my students learning and environment. It is my responsibility to encourage and push them to new ideas and goals. I may not effectively do this with all my students; some will listen, do the work, and learn while others will push back and refuse to try. This can be due to how they see me or how they feel about CS. As a teacher/researcher, I am the leader of the instruction and the researcher who will observe and collect data about how the instruction either succeeded or failed. I have assessed how well the students learned the concepts or not. I am evaluating myself as well as the students.

I have negotiated my positionality as an insider/outsider to increase trust and rapport with the students by being open and honest with expectations and processes throughout the research study. As an insider, all the participants know me, my teaching
style, and the subject I teach. This has given me easy access to all students included in the study. And as an outsider, I am an authority figure in the room and school. This can create a power struggle in interviews and in gaining knowledge. I removed the barriers to knowledge acquisition. I accomplished this making everyone feel welcome and respected, showed that their thoughts and ideas have value, and increased their confidence in learning a concept they perceive as hard.

The research might be biased given that I am the researcher and the teacher and have been teaching the participants in the previous semester. I may interpret what students mean but did not say, in the interviews or on the survey. Ways to address the bias was to ask probing questions, clarify answers and member checking. My values as an educator are that I want all students to be successful in my class, and as a researcher, is to get to the truth of what they learned in the process.

**Definition of Terms**

Abstraction- The collecting relevant data and discarding irrelevant data from programs to generate patterns and find commonalities with other programs to simplify the problem (Cetin & Dubinsky, 2017; Román-González et al., 2017; Shute et al., 2017; Wing, 2006).

Algorithm- is a set of instructions designed to perform a specific task(Aho, 2012; German, 2019; Lockwood et al., 2016).

Computational thinking (CT)- is the process of approaching a problem in a systematic manner and creating and expressing a solution such that it can be carried out by a computer (Aho, 2012; Román-González et al., 2017; Shute et al., 2017; Wing, 2006).

Coding environments or IDE-An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers.

Decomposition- is a way of thinking about problems, algorithms, artefacts, processes, and systems in terms of their parts, break down into easier to understand parts (Lockwood et al., 2016; Román-González et al., 2017; Shute et al., 2017; Wing, 2006).

Pattern recognition- Part of problem solving, to identify patterns and rules in the data, the patterns are similarities or characteristics that some of the problems share (Angeli & Giannakos, 2020; Korkmaz & Bai, 2019; Shute et al., 2017).

Scratch- is a free programming web site and online community where you can create your own interactive stories, games, and animations using block-based code.

Sprite- a type of "stand-alone" computer graphic, a two-dimensional image or animated image that plays a specific role, often independently manipulated, within a larger game environment. (What Is a Sprite?, n.d.)

Syntax- Syntax refers to the spelling and grammar of a programming language. Computers are inflexible machines that understand what you type only if you type it in the exact form that the computer expects. The expected form is called the syntax (Cetin & Dubinsky, 2017; Weintrop, 2019; Weintrop & Wilensky, 2017, 2018, 2019).

Problem solving- part of computational thinking, the process of finding solutions to difficult or complex problems (Korkmaz & Bai, 2019; Lockwood et al., 2016; Román-González et al., 2017; Shute et al., 2017).
Block-based programming environment is the de facto way to teach kids introductory programming in the US. Instead of traditional, that involves dragging “blocks” of instructions (Weintrop, 2019; Weintrop & Wilensky, 2017, 2018, 2019).

Text-based programming environment—coding is done through typing various characters from a syntax, or list of codes readable by a particular language (Weintrop, 2019; Weintrop & Wilensky, 2017, 2018, 2019).

Coding or Programming— is the process of creating a set of instructions that tell a computer how to perform a task (Chalmers, 2018; Kafai, 2016; Merriam-Webster, n.d.).
CHAPTER 2:
REVIEW OF LITERATURE

The purpose of this action research study was to evaluate how BBPE influence students’ CT skills and students’ attitudes towards CS at CE Williams Middle School.

This study focused on the two research questions: (1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills? (2) How and to what degree were the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?

Method

The resources that were collected for this literature review used electronic databases, which include ERIC, Google Scholar, USC Libraries, and Science Direct. The keywords used to search these databases were: middle school, computer science, STEM, computational thinking, computational thinking skills, participant attitudes towards computer science, coding, programming, computer science instructional strategies, game design, secondary education, robotics, problem solving and algorithmic thinking. Many times, these keywords were used together to narrow the results. Many articles were found using the mining technique of reading other articles’ bibliographies for articles that were included in their studies.

Based on the research questions, this review contains five sections: (1) theoretical framework of pragmatic constructionism used in this research; (2) the history of Computer Science and STEM education in middle schools; (3) background and skills
behind CT and how to assess these skills in middle school participants; (4) participants’ attitudes toward coding and CS; (5) instructional strategies to teach CT in middle schools.

Theoretical Framework of Constructionism

The start of this literature review investigated the theoretical framework that drives this study. (1) Constructionism is a framework where people construct their actions based on the reality of their environment. (2) Constructionist where students form knowledge and meaning based upon their experiences.

Constructionism in Teaching and Learning

Papert’s (1993) idea of constructionism is that children will do better by building for themselves the specific knowledge needed to complete tasks. As well, when they are actively engaged, they will mentally construct knowledge from the input they receive from teachers and activities (Papavlasopoulou et al., 2019; Papert, 1993). “Constructionism’s basic idea is that the most effective learning experiences are those that include active creation, socially meaningful artifacts, interaction with others, and the use of elements that support one's own learning and thinking” (Papavlasopoulou et al., 2019, p. 416). Constructionism comes from Piaget’s theory of constructivism that individuals construct knowledge, “which associates experiential learning via active engagement, that existing knowledge and meaningful experience will determine how one deal with new experience and environment and hence develop conceptual understanding.” (Chuechote et al., 2020, p. 2) The primary focus of Computer Science (CS) is to build knowledge through repeated experience in coding and programming. Students need to experience their learning. Therefore, CS courses utilize hands-on programming so that students build long-lasting knowledge.
**Constructionist Approach to CS**

Dolgopolovas, Dagienė, Jasutė, and Jevsikova (2019) stated that the constructionist approach to learning focused on making physical or mental ‘things’ during instruction and student interactions with them. Additionally, Atmatzidou and Demetriadis (2016) said that “learning is successful when students are experiencing and discovering concepts and activities for themselves” (p. 2). When students learn for themselves, they create or construct their own knowledge. Writing programs or code is a hands-on creative process, students make instructions that a computer follow, but programs do not always work. Students need to find what is not working and fix it, defined as debugging. Programming is learning what works and what does not work. Constructivist problem-solving learning is a way to connect coding opportunities into a simulated environment that is used in an interactive program (Pellas & Vosinakis, 2018). To clarify, when students create games or other programming projects, they are constructing their CS knowledge.

**CS and STEM Education in Middle Schools**

This section looks back at the history of CS and STEM education in the middle school setting by (1) defining STEM, (2) defining CS, and (3) explaining the history of CS in schools.

**Definition of STEM**

STEM is the combination of Science, Technology, Engineering and Mathematics as a discipline to create cocurricular activities to instruct students in many fields at once (Stohlmann et al., 2012). STEM a relatively new teaching philosophy is the integration of four core disciplines into a single, cross-disciplinary program that the instruction focuses on real-world problems and solutions (Dalton, 2019). The benefits of using STEM
education are that students become better problem-solvers, innovators, inventors, critical thinkers, and more adept in technology (Stohlmann et al., 2012). STEM is an ever-growing field in education where we see students in all grades solving real-world problems in the classroom. Brown et al. (2014) stated that the STEM discipline had more interest and value with the inclusion of CT to students then the previous ICT classes without it. The increase in the demand to teach STEM in K-12 education has also led to an increased demand for CS and CT, where “CT being viewed as at the core of all STEM disciplines” (Grover & Pea, 2013, p. 38).

**Definition of CS**

CS is defined as “the study of computation—what can be computed and how to compute it” (Wing, 2006, p. 6). Merriam-Webster defines CS as a branch of science that deals with the theory of computation or the design of computers (Merriam-Webster, n.d.). For this study I used the definition from Grover et al. (2015), which defines CS as a “problem-solving discipline that is used in the real world, and computer scientists are creative problem solvers who identify problems and create solutions to make our lives better and easier” (p. 209). This definition embodies every aspect of CS; computer scientists are creative problem solvers.

**History of CS in K12 Schools**

In the past, traditional teaching methods for CS were more focused on language learning than on CT concepts. Today’s student needs more access to coding instruction and not just at school (Kert et al., 2019). The need for online, self-graded, and self-directed environments, where students can learn to program and be engaged in the concepts, have increased.
Students now have a broad selection of courses to take including languages, computer systems, computer repair and others. Teaching programming can be traced back to *Logo* programming language written in 1968 (Kalelioglu, 2015). Logo is an educational programming language developed by Wally Feurzeig, Seymour Papert, and Cynthia Solomon, and it is known for its use of the turtle graphic. Logo was created to offer a conceptual foundation for teaching logical ways of thinking with programming ideas and activities (Kalelioglu, 2015). Later, to help teach children CS concepts and to program, *Scratch* was developed in 2003. It was one of the first BBPE and was created by Mitchel Resnick, Yasmin Kafai, and John Maeda at the Massachusetts Institute of Technology (MIT) Media Labs. The use of programs created to teach CS and CT skills has a short history; most systems simply used a textbook and had students create programs. Today, we have many more programs and curriculums to teach CS.

The inclusion of CS and programming for high school graduation started its current uptrend in 2013. This happened when the non-profit group Code.org began its global movement to teach CS to every child with *Hour of Code* events (Dixon, 2020). At the same time the need to have students learn to program and to be creators of technology became a focus for CS education. When Wing (2014) stated that “we need to continue to promote with passion and commitment the importance of teaching CS to K-12 students” (p. 6), the boom in CS began. Taken together, CS has continued to grow, and it is expected that more and more K-12 students will learn how to code and appreciate CS in our society.

**Computational Thinking and Assessing Middle School Students**

CT was the focus of this research study and is the majority of this literature review. This section expands on the following topics: (1) definition of CT, (2) definition
of the skills that are included in CT, (3) student assessments and evaluations to determine CT skill development and attitudes, and (4) the research and methods that have been conducted done prior.

**Computational Thinking (CT)**

The term *computational thinking* (CT) has received varied definitions over the years. Wing (2006) explained that CT “involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to CS” (p. 33). CT was further defined by Dagli and Sancar Tokmak (2021) as a problem-solving process aimed to understand how computers work. A CT framework was proposed by Brennan and Resnick (2012) that included three dimensions: computational and programming concepts; computational practices, and computational perspectives. The concepts included in each dimension are: computational and programming concepts focused on sequences, loops, events, conditionals, operators, and data; computational practices included incremental and iterative programming, testing and debugging, reusing and remixing, and abstracting and modularizing; and last computational perspectives which are formed about the world and themselves as they program (Brennan & Resnick, 2012). Aho (2012) described CT as “the thought process involved in formulating problems so their solutions can be represented as computational steps and algorithms” (p. 835). For this study, I chose Shute, Sun, and Asbell-Clark’s (2017) definition of CT, because it brought the concepts of problem solving and the creation of solutions together. Their view of CT is defined “as the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts” (Shute et al., 2017, p. 1).
CT Skills

CT skills are needed for secondary students to be successful in CS. Different authors include different skills they determined were associated with CT. Wing (2006) described the skills as problem-solving, recursive thinking, using abstraction and decomposition, preparing for prevention or protection, and of heuristic reasoning. Shute et al. (2017) believed that the skills included in CT were decomposition, abstraction, algorithms, debugging, iteration, and generalization. Dagli et al. (2021) defined CT skills as “interpreting and understanding the digital data, algorithmic thinking, critical thinking, and decision making” (p. 3). From the previous definitions many of the skills are similar. The CT skills this study focused on are algorithmic thinking, abstraction, debugging, and pattern recognition. In my opinion, these four skills are the basis for understanding programming and being able to transfer knowledge from one programming environment or computer language to another.

Algorithmic Thinking

The term algorithm is defined by Peel and Friedrichsen (2018) as a sequence of steps. Algorithmic thinking is described as a “logical, organized way of thinking used to break down a complicated goal into a series of (ordered) steps using available tools.” (Lockwood et al., 2016, p. 1591). There are many ways to teach algorithmic thinking, those including programming with BBPE, robotics and unplugged activities. Unplugged activities could change the student’s idea of CS and make it more interesting for them, and ensure they understand key concepts of CT (Tonbuloğlu & Tonbuloğlu, 2019). This study used BBPE to teach CT and the definition of algorithmic thinking as the ability to think in steps or sequential rules to solve a problem, from Chuechote, Nokkaew, Phongsasithorn and Laosinchai (2020).
Abstraction

Abstraction is defined by Peel and Friedrichsen (2018) as simplifying information or conveying only the information that is needed. Shute et al. (2017) explained abstraction as the ability to gather important information, discard unrelated data from large systems to develop patterns, and discover commonalities. I used the programming version of the term abstraction, defined as the ability to handle complexity by hiding unnecessary details which is most often seen in the use of functions (Cetin & Dubinsky, 2017).

Debugging

This study’s definition for debugging is the identifying and correcting of errors or “bugs” in a program when the solutions do not work as imagined (Shute et al., 2017). Others have defined it similarly. For example, Dagli and Sancar Tokmak (2021), for example, defined it as finding and fixing errors. Brennan and Resnick (2012) explained that programs rarely work just as we imagined or on the first try; it is crucial for students to develop strategies for dealing with problems, and debugging is one strategy. Debugging a program is a form of problem solving in that one must read each line of code to determine the error and fix it; it is a systematic process for correcting problems (Peel & Friedrichsen, 2018).

Pattern Recognition

Pattern recognition is defined by Shute et al. (2017), “identify patterns/rules underlying the data/information structure” (p. 153). These patterns are also conceptual programming concepts linked to interactions that students use to develop games or programs (Shute et al., 2017). To assess pattern recognition, one must verify if students can recognize patterns or sequences in previously completed programs and apply them to
current programs (Chalmers, 2018). So, developing the ability to identify patterns in a program and reuse or remix them is what this study used to define pattern recognition in CT (Prextová et al., 2018).

**Evaluations of Student CT Skill Development**

There are two ways to evaluate student CT skills and attitudes: assessments and surveys.

**Assessments and Scales Used to Evaluate CT Skills**

The use of scales and/or assessments of students’ learning is necessary to validate the relatively new field of CS and the skills of CT (Román-González et al., 2017). Cutumisu, Adams and Lu (2019) stated that assessments help students improve their skills and increase their understanding. However, a single assessment cannot measure growth. Therefore, the need for pre- and post-tests to evaluate student growth and the effectiveness of the curriculum is necessary (Román-González et al., 2017). Yagci (2019) “created the *Computational Thinking Scale* (CTS) that was validated and contained 4 parts of CT: problem solving, creative thinking, algorithmic thinking and critical thinking” (p. 19). The CTS effectively evaluate a student’s development in CT. The Likert style items asked a variety of questions as in “It is fun to try to solve the complex problems” and “I like the people who are sure of most of their decisions” (Korkmaz et al., 2017, p. 565). Rachmatullah et al. (2020) developed Middle Grades Computer Science Concept Inventory (MG-CSCI) Assessment on the development of CS concepts for middle school students with 27 multiple-choice questions focused on the skills associated with variables, loops, conditionals, and algorithms. Examples of questions “ability to describe what a given loop is doing” and “ability to create variables, assign values and update variables” (Rachmatullah et al., 2020, p. 12).
Denner, Werner and Ortiz (2012) developed a strategy for coding student programs and summarized the results of how well students learned CT skills. There are 26 programming categories used to “evaluate the quality of the solution and analyze whether the programming constructs are used correctly” (p. 245). Some of the categories are (1) clear instructions on how to play, (2) clear goals on how to win or lose the game, and (3) a game that is functional or free of bugs (Denner et al., 2012).

**Surveys to Assess Student Attitudes Toward CT**

To assess student’s self-efficacy, scales were used to identify how students feel about CS. The most frequently used scale is CTS developed by Yagci (2019), a Likert scale questionnaire that contains 29 items and can be used by participants from different educational levels and age groups. Korkmaz et al. (2017) adapted a CTS as well, with the belief that “critical thinking has been seen as skill of the individual for determining the assumptions, hidden belief, values and attitudes” (p. 561). It was divided into five sections and 29 Likert style questions such as “it is fun to try and solve complex problems” and “I am willing to try challenging things” (Korkmaz et al., 2017, p. 565). Mouza et al. (2016) added attitudinal questions to their instrument to document changes in student attitudes towards CS and CT. Used simple questions like “I think I am a computer scientist” and “I like coding” can be a big help to understand students’ attitudes. Positive student attitudes determine positive development in CT.

**Prior Research and Methods Conducted on CT**

CT research has increased over the past few years and the methods assessing those skills have varied. The most often used methods have been questionnaires, interviews, and reviewed empirical literature. Questionnaires were used as a primary method to measure student growth in CT in several studies (e.g., Atmatzidou et al., 2016;
Brennan & Resnick, 2012; Chalmers, 2018; Fronza et al., 2017; Kert et al., 2019; Mouza et al., 2020; Román-González et al., 2017, 2018; Tonbuloğlu & Tonbuloğlu, 2019; Turchi et al., 2019; Weintrop et al., 2017, 2018, 2019; Yağcı, 2019; Yıldız Durak, 2020; Zhao et al., 2019). The questionnaires were in the form of pre- and post-tests. These tests measured content knowledge and CT skills students learned. Interviews were used to understand students’ perceptions of CT, although there are not as many studies with them (e.g., Brennan et al., 2012; Chalmers, 2018; Dagli et al., 2021; Lee et al., 2014; Lockwood et al., 2016). Other studies reviewed empirical literature to compare the findings against each other (Cutumisu et al., 2019; García-Peñalvo & Mendes, 2018; Hsu et al., 2018; Lye & Koh, 2014; Prextová et al., 2018; Shute et al., 2017; Tang et al., 2020). Therefore, what that means is that the use of tests and interviews can assess the extent to which students learned CT, and these methods should render some insightful results for my own study.

**Students’ Attitudes Toward Coding and CS**

Part of understanding CS and CT is understanding students’ attitudes toward them. To do this, this section investigated the following topics: (1) student success in programming in CS, (2) understanding students’ attitudes towards CS, coding, and gaming, (3) and previous research conducted on students’ attitudes about CS.

Student attitudes can determine if students will continue in the field and if they will enjoy learning CT. Many students have improper ideas and negative attitudes about CS, and they perceive it to be boring and difficult, requiring many hours of tedious work in front of the computer (Carter, 2006).
Student Attitudes and Success in Programming and CS

Cetin and Ozden (2015) believed that there is an association between students’ attitudes and their achievement. When students hold negative attitudes towards CS this can hinder them from developing the CT skills needed to be successful or limit their motivation (Zhao & Shute, 2019). Increasing student understanding to affect their perceptions and attitudes toward CT could help develop their CT skills and change their attitudes (Grover et al., 2015). One study showed that there are positive effects of video games and BBPE toward CS, including confidence in programming and self-identification as a programmer (Zhao & Shute, 2019). Thus, if students feel confident in programming, they will increase their positive attitudes towards CT and continue their CS education.

Understanding Students’ Attitudes Towards CS, Coding, and Gaming

The variables included in determining student attitudes towards CS are student misconceptions, low self-efficacy, performance, and type of instruction. Students typically misinterpret or have negative attitudes that CS is just coding, it is only for smart people, it is boring, or requires a tremendous amount of work (Carter, 2006; Lewis et al., 2010; Taub et al., 2012). Yildiz Durak (2020) found that “the more the programming environment becomes complicated, the more possibly students may leave the programming” (p. 191). They also have misconceptions about what types of careers are included, such as students may see it as a programming job when CS is about problem solving (Grover et al., 2015; Taub et al., 2012).

Low self-efficacy, or how students see themselves in CS, may be a factor in success and future involvement in CS (Gunbatar & Karalar, 2018). Studies have shown that student attitudes and beliefs are linked to performance (Lewis et al., 2010). Also, the
cause may be previous programming experiences, where some students like the idea of making games where others typically do not (Gunbatar & Karalar, 2018; Yildiz Durak, 2020). And yet a third reason may be that student attitudes might not be related to the students at all but to the programming environments themselves (Gunbatar & Karalar, 2018). Text-based environments are difficult, time consuming, and are not perceived as fun (Carter, 2006; Weintrop & Wilensky, 2018), while block-based environments are perceived as fun and more interactive (Weintrop & Wilensky, 2017; Weintrop, 2019). BBPE are effective in deleting obstacles to programming instruction and overcoming negative attitudes (Gunbatar & Karalar, 2018).

Grover, Pea, and Copper’s (2015) study showed an increase in positive student attitudes toward CS where computers were seen as a tool to help solve problems and sparked curiosity to learn more. Video games used to teach CT have shown an improvement in cognitive and attitudinal development with students (Zhao & Shute, 2019). Also, when student interest and personal choice are considered when placing students in classes, increased academic performance and retention were observed (Ketenci et al., 2019). Furthermore, the creation of CS courses using BBPE showed a positive impact on interest and motivation to learn from middle school students (Stohlmann et al., 2012). Using any or all of these strategies has the potential to improve students’ attitudes toward CS and CT.

**Researching Student Attitudes Toward CS**

Research studies on student attitudes toward CS include instruments with Likert style responses to understand students’ attitudes and to track the changes in these attitudes from the beginning to the end of the study. Cetin and Ozden (2015) developed a scale to determine students’ attitudes toward programming and CS called *Computer*
Programming Attitude Scale, CPAS. This scale was researched and tested in two phases. It contained 18 items to assess three dimensions: affection, cognition, and behavior. The test was created for university students but could be used for other age groups. Some examples of items include “I find programming frustrating”, “Program-writing is boring” and “Programming improves your problem-solving skills.” (Cetin & Ozden, 2015, p. 672) The authors determined that CPAS was a “promising tool for both instruction and research in CS” (p. 670). Lewis, Jackson, and Waite (2010) developed a 32-item attitudinal and beliefs questionnaire for their faculty and university students to guide curriculum improvements. Their results provided insight and surprising attitudes towards CS. Some items asked were “You can only truly understand something if you figure it out on your own”, “If you can do something you don’t need to understand it” and “I cannot learn computer science without a teacher to explain it”(Lewis et al., 2010, p. 80).

Gunbatar and Karalar (2018) conducted a pre/post design quasi-experimental study to explain changes in students’ self-perceptions and attitudes towards programming. That study, used MBlock, an online BBPE to learn programming. The results showed a significant increase in students’ attitudes towards programming. As well, Taub, Armoni, and Ben-Ari (2012) investigated middle school students’ ideas and attitudes about CS in a mixed-method study, utilizing questionnaires and interviews after a series of unplugged CS activities. Their results did not increase students’ desires to study CS in high school but did improve students’ views of CS as a career. Therefore, what this means is that there are quite a few questions that can be asked and evaluated to ascertain students’ attitudes towards CS.
Instructional Strategies for CT in Middle Schools

There are many instructional strategies used to teach CS and CT. In this section, I will discuss five of them, which are: (1) BBPE, (2) simulations, (3) physical and hands-on coding, (4) game design, and (5) the prior research conducted in CS and coding instruction.

BBPE Used to Teach CT

Block-based programming environments (BBPE) are now considered an important part of CS as they are used to teach younger students how to program without learning the syntax of the programming languages (Weintrop, 2019). Syntax is the wording and punctuation specific to each programming language (e.g., Python, Java, QBasic, and HTML). The absence of syntax reduces programming errors, permits students to focus on the CT concepts, and allows them to transition to text-based programming languages in higher grades easily (Weintrop et al., 2019). Weintrop and Wilensky (2018) defined a BBPE as a platform that offers blocks of code allowing for a puzzle-piece view of a program as well as visual cues that teach the user how the commands can be assembled. These environments give students opportunities to be introduced to CT concepts in fun and useful ways as students can create games, animations, simulations, and more. Research has found that students become more motivated and develop positive attitudes towards coding after working with BBPE (Papavlasopoulou et al., 2019). The current trend in CS education is the increasing use of BBPE in all K-12 classrooms and using text-based as the students become more accustomed to coding. 

CT and its various components can be developed through multiple instructional tools. Programming with block-based and text-based environments help students to
develop CT skills, however block-based are more accessible and promote higher engagement in younger audiences (Yildiz Durak, 2020). CT includes other skills such as problem solving, debugging, algorithmic thinking, control, and cooperation, which can all be acquired through programming and should be taught to all secondary students (Tsai et al., 2019). Studies have found that using BBPE led to middle school students being more engaged and learn CT without memorizing the syntax, which can limit understanding (Celepkolu et al., 2020; Mladenović et al., 2018; Shapiro & Ahrens, 2016; Weintrop, 2019; Weintrop & Wilensky, 2017, 2018, 2019).

In 2013, Code.org launched their site that allowed students to learn programming with Blockly (Dixon, 2020), a visual drag and drop format, for free (Kalelioglu, 2015) with many similarities to Scratch. This revolution in teaching programming opened the door for many others to follow and offer block-based style online programming environments for students of all ages to learn. Now there are many environments, free and paid for, where students can learn to program and have fun with game-style interactions. Students, as young as five are learning basic CT concepts with robots and online programming environments.

**Simulations for CT Instruction**

Students can learn CT skills by designing and programming games and simulations. (Repenning et al., 2015). Merriam-Websters (n.d.) defined simulation as “the imitative representation of the functioning of one system or process by means of the functioning of another”. Simulations are more than simple animations; they are “dynamic computer models that allow students to test different conditions and investigate different outcomes” (Celepkolu et al., 2020, p. 672). Simulations help students form a better understanding of CT concepts (Celepkolu et al., 2020). Simulations challenge students to
create solutions to real-world problems. The use of these simulated situations requires the connection of a problem with programming skills to improve their CT skills (Pellas & Vosinakis, 2018). A game simulation called Zoombinis created by Rowe (2021), where players interact in a variety of tasks and played multiple rounds of CT strategy development. Simulations allowed for experience with a problem and the creation of a possible solution.

**Physical and Hands-On Coding Used to Teach CT**

Robotics and hands-on computing inspire students to learn CT skills and CS (Merkouris et al., 2017). These curriculums utilize project-based interventions and those that involve collaboration and physical objects to program tasks can be enjoyable and effective at teaching CS (Scherer et al., 2020). Atmatzidou and Demetriadia (2016) stated that “educational robotics is being introduced in many schools as an innovative learning environment, enhancing and building higher-order thinking skills and abilities, and helping students solve complex problems” (p. 661). For instance, the use of Arduinos with the combination of BBPE as hands-on problem-solving activities is ever increasing. Micro:Bits is another one of these hands-on simulations that allows students to create input/output interactions with the computer and environment. The CT skills learned with these simulations include debugging, abstraction, algorithmic thinking, and pattern recognition.

**Game Design Used to Teach CT**

Game design is an ideal environment for learning CT, as student designers needed to create a series of unique user interfaces but ensured that these interfaces work (Kafai & Burke, 2015). Ouabain et al. (2015) explained that “the creation of simple games by students using a suitable programming environment for novices can improve the
motivation” (p. 1480). This, in turn, can help in learning CT. When students who
designed games, they used more variables, loops, and if-statements in their programs,
debugged, revised, and tested their games repeatedly (Kafai & Burke, 2015). Game
design helps in reinforcing CT skills with students because of the repetitive nature of the
process. Akcaoglu and Green (2019) explained that “games are complex systems,
composed of various elements that interact with one another in many different and
complex ways” (p. 5). With the use of scalable game design, curriculums start with
encouraging students first in game design and then moving them to designing STEM
simulations by applying their CT skills (Repenning et al., 2015). Game-based learning
and design is an area of CS where learners can discover CT and increase performance in
programming courses (Topalli & Cagiltay, 2018). Thus, when students are having fun
developing a game, they are learning CT.

Prior Research Conducted in CS and Programming Instruction

Most research into CT and programming was focused on BBPE such as Code.org,
Scratch, MBlock, Alice, and App Creator (Gunbatar & Karalar, 2018; Ouahbi et al.,
2015; Topalli & Cagiltay, 2018). These environments allowed students to program
without having to memorize syntax, and in turn students were more successful and show
growth in CS and CT skills (Weintrop & Wilensky, 2019). Connections have been
identified between designing digital games and increased cognition and motivation as
well as an elevated ability to problem solve and CT concepts (Akcaoglu & Green, 2019).
Robotics and hands-on computing have been purported to be an inspiring curriculum for
influencing students to participate in the field of CS. (Merkouris et al., 2017).
Furthermore, physical and hands-on programming such as robotics and Arduino-style
computers also use block-based systems and have seen similar growth in skills (Sáez-López et al., 2016; Weintrop & Wilensky, 2017, 2018, 2019).

**Chapter Summary**

This literature review was conducted to investigate the previous research on CT with middle school students, student attitudes and the methods of research used. This mixed-method study used the theoretical framework of constructivism, which is a study rooted in the focus on real-world situations and where the students form knowledge and meaning based upon their experiences. The definitions of CS and STEM, of how CS education has changed over the years, what its future holds, and all the new innovations used to teach CS and coding will help in the development of a sound research study into CT. CT is the ability to think like a computer and the skills of understanding algorithms, abstraction, debugging, and problem solving are central to CS and becoming a successful in CS. The assessments and surveys found in previous studies will help determine the growth of CT skills and attitudinal levels of students towards CS and coding. Research in understanding student success in programming is connected to students’ like or dislike of CS pinpoint what instructional strategies should be used. With an ever-increasing need for students in the CS field, students’ attitudes towards coding were a determining factor if they continue in the field. There are many instructional strategies available to teach CS and CT. Most are internet-based and include: BBPE, simulations, physical and hands-on coding, and game design. Previous research and methods of research have shown us what works and what doesn’t.
CHAPTER 3: 
METHOD 

Purpose Statement 

The purpose of this action research study was to evaluate how a block-based programming curriculum in game design affects middle school CS participants’ CT skills and their attitudes towards CT and CS at CE Williams Middle School. 

Research Questions 

(1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills? 

(2) How and to what degree are the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming? 

Research Design 

Action research is “any systematic inquiry conducted by educators for the purpose of gathering information about how their schools operate, how they teach, and how students learn” (Mertler, 2020, p. 18). It centers on the improvement of one's practice (Anderson et al., 2001; Johnson, 2008). It is iterative and is used to address problems in practice (Anderson et al., 2001). To educators, action research, is the process that improves education by incorporating change. It is open-minded, and it is collaborative in that educators talk and work together to improve relationships (Anderson et al., 2001; Mertler, 2020). Action research is appropriate for my study to create an in-depth level of knowledge of participants’ current CT skill levels and attitudes towards CT, and how they develop and change over time while enrolled in CS classes.
Action research is different from other types of research, because it is collaborative, is focused on practical and small population settings, not a large population and includes the students with whom educators worked with. Zeni (1998) says it involves educators “studying their own professional practice and outlining their own questions. Their research has the immediate goal of assessing, developing or improving their practice” (p. 6). Action research is not conclusive; there are no right or wrong answers, just tentative solutions (Mertler, 2020).

Creswell and Creswell (2018) explained that the combining of qualitative and quantitative research is mixed methods, where the integration of both types of data produce insights beyond the information provided by one type of data alone. Qualitative data is data in the forms of words from observations, interviews, and cases whereas quantitative data are numbers or scores from tests and surveys (Creswell & Creswell, 2018). This action research will be a triangulation mixed methods study where the quantitative and qualitative data are collected around the same time. This allows for the data to be analyzed equally, so that the results are used simultaneously to understand the research problem (Mertler, 2020).

**Setting and Participants**

The setting for this research study was a middle school in Charleston County School District in South Carolina. CE Williams Middle School was an urban school with 850 students where 68% receive free or reduced lunch. This school was considered a Title 1 school and received additional funding for special projects to enhance the learning environment. Title 1 enabled the school to provide enrichment, promote schoolwide
reform, upgrade the quality of education, and help parents participate in their students learning *(South Carolina Department of Education Title 1, n.d.)*. The demographics of this school were 37.06% White, 39.50% Black, .09% Hispanic or Latino, .06% two or more races, and .02% Asian or Pacific Islander. Approximately 18% of students received special education or 504 services, and 20% were gifted. The Rehabilitation Act of 1973 and the ADA Americans with Disabilities Act of 2008 provide protection to students in elementary and secondary school, “Section 504 requires recipients to provide to students with disabilities appropriate educational services designed to meet the individual needs of such students to the same extent as the needs of students without disabilities are met” *(Protecting Students with Disabilities, 2010, paragraph 2)*. The school’s make up of genders is 53.4% were male and 46.6% were female.

This school offered two different courses of CS for students to choose from, Discovering CS and Project Lead the Way (PLTW) CS for Innovators and Makers. Students could choose either or both courses as an elective, but neither are required. The Discovering CS course was highly encouraged in that it is one of the courses students can take to earn high school credit and was the only one offered in middle school. Beginning in 2019, the state of South Carolina required all students to earn one credit in CS to graduate. The DCS course curriculum includes web design, interactive animations and games, app creation as well as data and society. The design process and physical computing to expose students to all types of programming and basic computer concepts. Students participated in the course for a full year and met every day for 45 minutes.
The criteria to select students for this study were all who are enrolled in the one CS course taught by the teacher/researcher. Administration randomly chose students to participate in the course based on test scores and availability in the students’ schedule. There were 25 students from the 8th grade enrolled in the DCS course in the spring of 2022. The demographic make-up for the course was 12 aged 14 and 4 aged 13; 9 females and 7 males; 1 English as a Second Language; 8 race was black, 4 race was white, 1 race was Asian and 3 who were two or more races. There will be students considered gifted in the course, since the curriculum is high school credit, therefore no accommodations will need to be made. There were no students with 504’s.

**Intervention**

This section describes the intervention being implemented in this 11-week study. It begins with (1) detailing the curriculum developed by Code.org, (2) then discuss the activities participants participate in each week, and (3) describe the projects, and (4) finished with the project’s participants will complete.

**Code.org**

Since 2015, I have taught using Code.org’s CS Discoveries and have found that the curriculum to be engaging, challenging, and appropriate for middle school students. Problem solving and CT are central to the lesson plans, and it relies on guided discovery to promote “a level of freedom for learners so that they explore the problem, identify patterns, and discover the underlying principles on the problem” (Kale & Yuan, 2021, p. 639). Reasons for choosing Code.org was it includes instructive materials focused on programming and CT, it is well-organized, has activities with increasing level of
challenge from simple to complex. Teacher lesson plans have detailed notes about what is being taught, minor scripting to start class discussions, and included answer keys to all programming steps in the lessons. It also allowed teachers to monitor student progress in the teacher dashboard (see Figure 3.1) and give feedback to students and grade student created programs (see Figure 3.2) (Kalelioglu, 2015). This feedback is given to students through the site, so students can go back and make corrections. It gives the teacher an opportunity to see which errors students make and what is needed support them with. Furthermore, the platform has a “keep working” tag (see Figure 3.2) that showed students what needs to be addressed quickly in terms of tasks and errors.

![Figure 3.1 Teacher Dashboard](image)
Code.org is a block-based programming environment with lessons that participants learned how to design and create games, discovered CS and CT concepts without the burden of memorizing the syntax, and developed higher order thinking skills (Kalelioglu, 2015). A list of the concepts taught in Code.org’s curriculum from Kalelioglu (2015), included the skills this study is focused on which are debugging, abstraction, algorithmic thinking, and pattern recognition.

**Weekly Schedule**

This intervention is nine weeks long. This class met every day face-to-face, Monday through Friday for 45 minutes for the entire school year. All assignments were completed individually on their assigned school owned Chromebooks. Participants were allowed to collaborate with each other, to give insight into other participants’ programs, help find errors, and encourage others to create fun and interactive games. The game
design unit included two chapters. Chapter one contained lessons 1 through 17, and chapter two included lessons 18 through 27. An overview of how each lesson was addressed in the intervention along the nine weeks is provided in Table 3.1. This table details the focus on each of the lessons and when mini-projects and projects occur.

**Table 3.1 Intervention Schedule**

<table>
<thead>
<tr>
<th>Week</th>
<th>Lessons</th>
<th>Topics for Lessons</th>
<th>Minutes Per Lesson</th>
<th>Intervention Deliverables &amp; Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chapter 1 Lessons 1-4</td>
<td>Plotting shapes and parameters of shapes</td>
<td>55 minutes</td>
<td>rect(x,y,w,h) ellipsis(x,y,w,h) fill(color) background(color)</td>
</tr>
<tr>
<td>2</td>
<td>Lessons 5-9</td>
<td>Variables, random numbers, sprites, sprites properties and text</td>
<td>45 minutes</td>
<td>var randomNumber(0,10) createSprite setAnimation text(Str), x,y</td>
</tr>
<tr>
<td>3</td>
<td>Lesson 10-12</td>
<td>Draw loop and sprite movement</td>
<td>L10: 45 minutes 11: 45 minutes L12: 135 minutes</td>
<td>drawFunction Counter Pattern Mini Project: Create a background</td>
</tr>
<tr>
<td>4</td>
<td>Lesson 13-15</td>
<td>Conditionals and Keyboard input</td>
<td>75 minutes</td>
<td>If then keyDown(code) Mini Project- Sprite movement</td>
</tr>
<tr>
<td>5</td>
<td>Lesson 16-17</td>
<td>Mouse input</td>
<td>L16: 45 minutes L17: 180</td>
<td>world.mouseX world.mouseY Mini-Project: Interactive greeting card</td>
</tr>
<tr>
<td></td>
<td>Chapter 2 Lessons 18-20</td>
<td>Velocity Collision detection</td>
<td>L18: 45 minutes 19: 45 minutes 20: 135 minutes</td>
<td>velocityX velocityY bounce bounceOff collide displace Mini Project: Scroller game</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>Lessons 21-23</td>
<td>Complex sprite movement and collisions</td>
<td>L21 L22: 45 minutes L23: 135 minutes</td>
<td>Velocity with the counter pattern isTouching Mini Project: Flyer game</td>
</tr>
<tr>
<td>7</td>
<td>Lesson 24-26</td>
<td>Functions, game design process and using the game design process</td>
<td>75 minutes</td>
<td>Create Function Call Function</td>
</tr>
<tr>
<td>8</td>
<td>Lesson 27</td>
<td></td>
<td>225 minutes</td>
<td>Final Project: Design and Create original game No new Code Blocks</td>
</tr>
</tbody>
</table>

Week one included lessons 1 through 4. These lessons focused on how to create and place shapes on the screen as well as how to change their parameters and colors. The parameters used here are, x and y position on the screen, height, and width for size, and fill and outline colors. In each lesson different code blocks are introduced; in these lessons, participants learned to use the rectangle and ellipsis to create a background. Figure 3.3 shows the participant interface. The far left is the preview screen, where participants see their program in action. The middle section is the toolbox where all code blocks are located, and the far right is the workspace where participants drag the code blocks to create their programs. In this example program the participants were asked to correct the position and size of the shapes to recreate this background.
In week two, participants worked through lessons 5 through 9 and are introduced to variables, random number, text, sprites, and sprite properties. In the variables lesson, participants created and used variables with the parameters of the shapes. Participants used random numbers to make changes to the shapes where the parameters change each time the code is run. Participants are introduced to what a sprite is, as the characters used in games and animations. Participants also learned how to change their appearance and location with parameters of size, x, and y position. The last lesson for this week was adding text to the background. Lesson 9 also covered the flow of the program. The flow of how items (sprites, shapes, and text) are shown on the screen. Items that are programmed first are placed on the screen first, and any additional are layered as the program is read from the top to the bottom of the code blocks. This lesson focused on this, and participants must rearrange the code blocks to see the snake behind bars and the text in front of the bars, snake, and background, see Figure 3.4.

**Figure 3.3 Lesson 4 Instructional Example**
Lessons 10 through 12 were completed in week three. The first mini project is in lesson 10 where participants created a background using colors, shapes, and text, and include multiple sprites with changes to their properties (i.e., x and y position and size). The participant example for lesson 10 shows the creation of a background with many shapes (planets), a sprite and text, see Figure 3.5. The participant also used comments to help organize their code. In lessons 11, participants learned how to animate their screens with the draw loop using the draw function and drawSprites block, and to make the sprite appear to move from side to side by changing the x or y with random number. Next, in lesson 12, sprite movement was introduced with the counter pattern. This concept will help participants to understand and move their sprites around on the screen. For example, the counter pattern is making a new x position by taking the sprites current x position and adding 1 to it. The function looks like sprite.x=sprite.x+1. The sprite moves 1 position every time the draw loop is run.
Week 4 included lessons 13-15. Lesson 13 featured the next mini project. Participants used lesson 10’s completed program and added a draw loop as well as sprite movement with the counter pattern. In Figure 3.5 the rocket ship moved to the right inside the draw loop. In lessons 14 and 15, participants used conditionals with keyboard input to move sprites. We call this user interaction. Figure 3.6 shows a instructional program using a sprite that moved around the screen using the counter pattern inside four conditionals where the if statement used keyboard input to move the sprite left, right, up, or down when keys are pressed inside the draw loop.
In week 5 participants worked through lesson 16 on mouse inputs, and on lesson 17 on the midway major project. The project entailed creating an interactive greeting card with a background that includes shapes, colors, text, sprites, sprite movement, and conditionals with user input. This will conclude chapter one. Participants planned their project by drawing a background that includes shapes and colors, decide what sprite(s) to used, what movement the sprites performed, and what input to used. Text directions must be on the screen to tell the user how to interact. A variable to count interactions should cause either a change in the background or sprites. For example, in Figure 3.7 after 5 fork movements “Happy Birthday” showed on the screen with multi-colored balloons. Once planning is complete, participants began to code, starting with the background, adding sprite(s), using the counter pattern inside the draw loop with conditionals to allow for user input to move the sprite(s), and code what happened when the user followed the directions.
Chapter two started in week 6 with lessons 18 through 20. Participants learned about velocity, collision detection, and started their first game. The velocity lesson included activities that participants added 1 or 2 blocks of code to complete the task. Figure 3.8 is a preset lesson; where participants added the block inside the if statements to make the fish turn around then continue moving. Lesson 20 is a mini project in that the
participants created a game called the scroller game. They created a game that scrolls, and the sprites move with user input. During this lesson, each step included directions of what should be completed and what the code does and added comments. As seen in Figure 3.9, the preset code directed participants where to place the needed code. Participants add all code blocks starting with adding sprites, backgrounds, if statements, and counter patterns. Each step brought the participants closer to completion and a working game.

Figure 3.8 Lesson 18 Instructional Example
Figure 3.9 Mini Project Lesson 20 Scroller Game Instructional Example

Week 7 include lessons 21 through 23. Lessons 21 and 22 included complex sprite movements and collisions. There are 4 types of collisions and sprite reactions that
are seen in Figure 3.10. This is a preset activity where participants add each of the reactions and chose the correct one based on the image below. The next mini project is implemented in lesson 23, participants recreated the flyer game. This game has a sprite that moves in all 4 directions to collect (touch) the coin to earn points, see Figure 3.11. Step by step directions for participants to follow were included with preset code and comments, and this was the first game they have created with all the concepts taught in the unit.

```
function draw() {...
  background("lightblue");
  snake.bounceOff(pig);
  hippo.displace(elephant);
  rabbit.collide(monkey);
  giraffe.bounce(parrot);
  drawSprites();
}
```

Figure 3.10 Lesson 22 Instructional Example
In week 8, lessons 24 through 26 included functions that teach participants how to create a reusable code, one that is created and is called on more than once. Additionally, participants learned the game design process, where students used problem solving to plan, create, debug, and improve the game until it worked and followed all the
requirements. Participants took what they have learned from lesson 25 to plan, design, create, debug, and improve a game of their own. In the participant example of final project, we see all CT concepts implemented, see Figure 3.12.

The final project for the unit and chapter two is implemented in week 9, in lesson 27. For this final project, participants designed and programmed an interactive game. This game implemented a project guide where participants develop their plan for the sprites, backgrounds, and outcomes for their game. They next created pseudocode for the actions that take place in the program, such as what conditionals, variables, functions, user input, and interactions take place.

The CT skills developed throughout the intervention should be visible in the participant programs. Algorithmic thinking is seen in the planning of their program by using the project guide and in the creation of original outcomes in their program. Debugging is about finding and fixing errors, where the platform helps participants identify errors and use strategies to find the errors and fix them. Abstraction is demonstrated in the creation of functions for reusable sections of code. And pattern recognition is seen in the use of previously seen programs, or the reuse of code participants created in earlier lessons.
Data Collection

There are four different data sources collected to analyze how well participants have developed CT skills and if participant’s attitudes towards CS have changed at CE.
Williams Middle School. To understand the depth of understanding, these data sources each focused on different aspects of learning and the research questions (see Table 3.2). The data sources are (a) pre-test, (b) post-test, (c) survey, (d) participant interviews, and (e) participant artifacts.

**Table 3.2: Data Sources**

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Data Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills?</td>
<td>• Pre-test/Post-test</td>
</tr>
<tr>
<td></td>
<td>• Survey</td>
</tr>
<tr>
<td></td>
<td>• Participant Artifacts</td>
</tr>
<tr>
<td>RQ2: How and to what degree are the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?</td>
<td>• Participant Interviews</td>
</tr>
</tbody>
</table>

**Pre-test and Post-test**

Participants took the pre-test assessment to establish their entry level CT skills before any intervention took place. MG-CSCI included 27 multiple choice questions to assess students’ CS learning gains, three questions were included in this study’s test (Rachmatullah et al., 2020). Other questions contained in this studies assessment are fifteen teacher created questions, and ten from Code.org (see Table 3.3). All questions from other sources were not altered, 3 images of programs were recreated in Blockly to ensure students understood the representation. There were 22 multiple-choice, two true false, and six short answer content questions included. Tang et al. (2020) suggest that
teachers and researchers find a reliable and valid CT assessment, which can be adapted and used to demonstrate mastery of CS concepts. See Appendix C to see the entire pre-/post-test. Many questions on this assessment were piloted by the researcher with previous participants enrolled in the CS Discoveries course for the past two years. Participants were given a pre- and post-unit test to analyze their growth of knowledge.

The questions focused on CT skills, vocabulary, and CS concepts.

**Table 3.3 Pre-/Post-Test Questions and Origin**

<table>
<thead>
<tr>
<th>Question Origin</th>
<th>Question Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher Created</td>
<td>27, 29, 32, 35, 37, 39, 40, 41, 43 - 50</td>
</tr>
<tr>
<td>Code.org</td>
<td>22 - 26, 28, 30, 31, 33, 34, 35</td>
</tr>
<tr>
<td>MG-CSCI</td>
<td>36, 38, 42</td>
</tr>
</tbody>
</table>

All participants enrolled in the CS course and parents, or guardians were invited to a zoom meeting where the intervention, research parameters, and student involvement was discussed, and questions were answered. Afterward the sixteen who returned parental consent and student assent were included in this study. The assessment was completed in 30 minutes. When the intervention is completed, participants took this assessment again and the researcher reviewed the changes in depth of knowledge gained or lost.

**Survey**

To understand how participants’ attitudes are before the intervention began, there was an attitudinal survey with 20 questions embedded into the pre-test. Eight questions were taken from Korkmaz et al., (2017) Computational Thinking Scale or CTS. It was
created in 2017, where it was used with college level Turkish participants, and was later translated into Chinese to use with high school participants there. It was established that it could be used to determine participants' computational thinking skills and attitudes (Korkmaz et al., 2017). Questions were confirmed to be reliable and valid. When questions were included with this survey small adjustments were made for different languages and for use with lower grade participants. In Korkmaz et al. (2017) and in Korkmaz et al. (2019) the survey was used with college and high school students, there is no data using it with middle school and there is no data if using part of the scale will diminish the reliability analysis. The questions were confirmed to be reliable and valid, even with the change to a different language and when used with lower grade students. As seen in Table 3.4, Cronbach’s Alpha falls above .60 is considered to be high reliability and an acceptable index (Cresswell, 2018).

Table 3.4 Reliability Analysis Results Considering the Whole Scale and its Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Number of items</th>
<th>Cronbach’s Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creativity</td>
<td>8</td>
<td>.84</td>
</tr>
<tr>
<td>Algorithmic Thinking</td>
<td>6</td>
<td>.87</td>
</tr>
<tr>
<td>Critical Thinking</td>
<td>5</td>
<td>.79</td>
</tr>
<tr>
<td>Total</td>
<td>29</td>
<td>.82</td>
</tr>
</tbody>
</table>

The types of questions asked were “I like Computer Science” and “Do you consider yourself a computer scientist”, these are similar to the interview protocol questions, see Appendix C. The purpose is to see if their attitude changes after the intervention and if it does, why. Understanding how participants feel towards CS helped
the researcher determine if the intervention played a part in any changes and why. The breakdown of the questions and the focus are as follows: 1-6 are demographic questions, 7 - 9, 11, and 18-20 are attitudinal questions, 12, 13 and 15 are on algorithmic thinking, 10, 14, 16, and 17 are creative thinking, and 10 and 16 are creativity. All of the questions but numbers 1-6 and 21, were grouped into two composites of CT beliefs and participant attitudes (See Table 3.5). These composites groupings were created to better analyze participant attitudes and beliefs they hold about CT and CS.

### Table 3.5 Survey Questions Composite Scales

<table>
<thead>
<tr>
<th>Demographic</th>
<th>1-6, 21</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attitudinal</td>
<td>7-9, 11, 18-20</td>
<td>Participant Attitudes</td>
</tr>
<tr>
<td>Algorithmic Thinking</td>
<td>12, 13, 15</td>
<td>CT Beliefs</td>
</tr>
<tr>
<td>Creative Thinking</td>
<td>10, 14, 16, 17</td>
<td>CT Beliefs</td>
</tr>
</tbody>
</table>

### Participant Interviews

The interviews gave the researcher a more in-depth understanding of the participants’ skill levels in CT and of their attitudes towards CS. The use of these semi-structured interviews allowed flexibility for the interviewer to explore participants’ responses and to encourage participants to reflect on and discuss their perceptions and
skills (Chalmers, 2018). To fill this gap, the use of interviews to “investigate participants’ emergent in-depth thinking processes” (Tang et al., 2020, p. 4).

The interviews took place after the intervention was completed. A random sampling of participants was chosen, all pseudonyms were placed in a box and 4 were chosen to sit for 20-minute one-on-one interviews about their experiences during the intervention. The interviews included questions about content such as “Explain how loops work?”, and “When you encountered a problem in your code without an obvious answer what steps did you take to solve it?” where participants had the opportunity to explain their knowledge of their CT skills. Also included were attitudinal questions such as “What did you enjoy most in computer science class?” and “Do you see yourself as a computer scientist?” to understand their feeling about CS and the class. All questions were open-ended to encourage open discussion, answers should not be “yes” or “no”, and follow-ups were included to explore their understanding further or probe for more information. Questions about their artifacts were also included, these questions focus on the code used by the participants to create their games and how they understood them to work. Interviews included questions to “probe students’ strategy on designing video games using block-based programming” (Tang et al., 2020, p. 4).

During the interviews the researcher took notes and record the entire conversation using Apple Garage Band. After all interviews were completed, the researcher listened to and transcribe all the interviews using www.temi.com. Temi allow the user to upload the audio and export a transcript, while listening and reading the transcripts to ensure all answers were heard correctly and understood. Using the transcriptions to “read through
all the data and create a coding structure using a combination of topics from the interview protocol as codes and developing codes based on what came up in the interviews” (Koch & Gorges, 2016, p. 6).

**Interview Protocol**

The interview protocol was created to align with the research questions, see Table 3.6. This protocol is more of a guide to ensure all responses needed to understand the participants’ skills and attitudes after the intervention were asked. The questions were designed to ask the participants’ understanding of concepts and attitudes toward CS and BBPE used during the intervention. This tool was used to further explain the participant’s understanding of CT skills as compared to their posttests and artifacts.

**Table 3.6 Interview Questions Alignment to Research Questions**

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Interview Questions</th>
</tr>
</thead>
</table>
| RQ1: How and to what degree did BBPE affected CT skill levels (algorithms, abstraction, debugging, and problem solving) for middle school CS participants? | 1a. Loops are used in many areas of Game Design; will you explain how loops work?  
  b. Tell me about a loop you used in your code.  
  c. What is a function of a loop?  
  2a. Tell me how a conditional works?  
  b. Where have you used them during this class?  
  c. Did they work?  
  d. What happened when one worked?  
  e. What happened when one didn’t work?  
  3a. Name ways you can include variables in Game Design?  
  b. How do they work?  
  c. How did it help complete your game?  
  4a. The Counter Pattern is used in many tasks in Game Design, tell me how it works?  
  b. Where did you use it most?  
  c. Was it an easy concept to learn?  
  d. Why is that? |
5a. When you encountered a problem in your code without an obvious answer what steps did you take to solve it?
   
   b. Did you collaborate with other participants to solve it?
   c. Was that helpful?
   d. Did you read others code to better understand how to solve it?
   e. Was that helpful?

6a. Looking at your artifact (program your game) tell me how this program worked.
   
   b. Why did you choose (select a line of code) this?
   c. Explain the process of how this line works?

RQ2: How and what degree were the differences in middle school participant attitudes towards CT and CS after a unit in block-based programming?

1a. Which aspect of programming did you enjoy most in Computer Science class?
   
   b. Why did you enjoy that aspect so much?
   c. Would you recommend this class to other participants?

2a. What did you like least about CS class?
   
   b. Why did you like that so little?

3a. What aspects of the block-based programming did you like most?
   
   b. Please explain.
   c. Least?
   d. Please explain.

4a. Based on your experience in CS class, do you think CS is interesting?
   
   b. Do you see yourself as a Computer Scientist?
   c. Why do you feel that way?

5a. Did you enjoy dragging blocks in Java Script?
   
   b. Why is that?
   c. Please explain.

Participants Artifacts
Participants were creating games in a BBPE during the intervention. Collecting and analyzing the student created games gave the researcher the ability to see how well participants understood the CT skills taught, and to ascertain if participants can use code blocks correctly. A digital copy of each project, including the code, was saved from Code.org and labeled with a non-identifiable number.

Artifacts, or any type of performance assessment or student projects, are used to systematically evaluate the attainment of the learning targets (McMillan, 2013). Like a performance assessment, the researcher used a grading rubric with different levels of achievement for each dimension of CT skill to indicate whether a certain criterion was met. For example, Bers, Flannery, Kazakoff, & Sullivan, (2014) evaluated participants’ knowledge of debugging, correspondence, sequencing, and control flow by assigning the appropriate level of achievement based on a scoring rubric to each participant’s robotics project. Artifact “assessment could capture a holistic view of what skills students have obtained through projects” (Tang et al., 2020, p. 8).

**Data Analysis**

This study included quantitative and qualitative data analysis methods to develop a better understanding of the data (Creswell & Creswell, 2018) as seen in Table 3.7 below. These methods consisted of descriptive statistics, paired samples t-tests, thematic analysis, inductive and deductive analysis using the data from (a) pre-test, (b) post-test, (c) survey, (d) participant interviews, and (e) participant artifacts. The rubric (see Appendix D) helped participants identify the requirements of their game and aid the teacher in grading of the final project.
Table 3.7 Research Questions, Data Sources and Methods of Analysis

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Data Sources</th>
<th>Methods of Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1: How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school participants’ CT skills?</td>
<td>• Pre-test</td>
<td>• Descriptive statistics</td>
</tr>
<tr>
<td></td>
<td>• Post-test</td>
<td>• Paired samples t-tests</td>
</tr>
<tr>
<td></td>
<td>• Survey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Participant Artifacts</td>
<td>• Inductive analysis</td>
</tr>
<tr>
<td>RQ2: How and to what degree are the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming</td>
<td>• Participant Interviews</td>
<td>• Inductive and thematic analysis</td>
</tr>
</tbody>
</table>

Quantitative Data Analysis

The quantitative data analyzed came from the pre- and post-tests and surveys completed by participants at the beginning and end of the intervention. JASP, is a computer statistics software, was used to investigate the hypothesis of an increase in CT skills and the changes in attitude towards CT after the intervention. Descriptive statistics was used. Specifically, measures of central tendency of each data set to help summarize the large amounts of data that was collected. Measures of dispersion and variation were also used to better understand to what degree the data varies from the calculated means of each data set (Hanneman et al., 2012; Mertler, 2020). Dispersion indicated how much variability is in the group’s responses. Standard deviation, the average distance away from the mean, helped to identify outliers.
The Shapiro-Wilk test for normality was conducted on the pre- and post-test data to determine if the data had normal distribution and if the paired samples t-test is appropriate or if a non-parametric test was more appropriate. Paired samples t-tests (Hanneman et al., 2012) was used to determine if there were changes in the dependent variable of participant CT skill levels. The pre- and post-test scores were compared to determine the extent to which the means differ. The investigated level of significance for this hypothesis, an $\alpha \leq .05$ was considered statistically significant.

**Qualitative Data Analysis**

The qualitative data analysis on participant interviews was add to the quantitative data results. Inductive analysis was used to identify and organize data into codes and categories to construct a framework in which to present findings (Mertler, 2020). Open and in vivo coding was used identify any patterns, concepts, or similarities, coding was an iterative process done multiple times using Delve tool coding feature allowed the researcher to read all data collected multiple times. The codes were then analyzed to develop categories, all codes and categories were used to develop themes that illustrated the patterns within the data (Braun & Clarke, 2006; Clarke & Braun, 2017). This thematic analysis allowed the researcher to evaluate the qualitative data and develop themes that emerged from the data (Braun & Clarke, 2006 & 2017). These themes were then checked against excerpts of the coded data to explore the accuracy of the themes and determine if further refining of the data was needed. Findings were communicated as themes using thick rich descriptions with examples from the qualitative data collected.

**Procedures and Timeline**

58
This was an 11-week long action research study. The recruitment process started by inviting parents and guardians to a zoom meeting where the intervention, research parameters, student involvement, and confidentiality were explained, and questions were answered. Parent consent forms were sent home to be signed and returned. Once consent forms were received, participants signed assent forms. Participants had one week to sign and return all forms.

Following this, all participants completed the pre-test and survey. The questions included in the pre-test focused on the skills of programming a game that include variables, sprites, sprite movements and properties. The survey included Likert style questions about how participants felt about CS. Next, participants attended a 9-week intervention. The intervention included a total of 27 lessons divided into two chapters in this unit using BBPE. Participants had one day to complete each lesson. The intervention also featured four mini projects and two major projects, which participants completed over the course of several days. The mini projects were 1-to-2-day activities where participants used the skills taught in the previous lessons. After the intervention, in the last week, participants completed a post-test, survey, and attend individual interviews. An overview of the study’s procedures and timeline is provided in Table 3.8.

**Table 3.8 Procedures and Timeline**

<table>
<thead>
<tr>
<th>Week</th>
<th>Study Procedures, Lessons, and Topics</th>
<th>Data Collection</th>
<th>Data Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Recruitment: consent and assent forms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th>Chapter 1</th>
<th>Lessons 1-4, Plotting shapes and parameters of shapes</th>
<th>Pre-test and survey</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Lessons 5-9, Variables, random numbers, sprites, sprites properties and text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Lesson 10-12, Draw loop and sprite movement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lesson 13-15, Conditionals and Keyboard input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Lesson 16-17, Mouse input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Chapter 2</td>
<td>Lessons 18-20, Velocity and Collision detection</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Lessons 21-23, Complex sprite movement and collisions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Lesson 24-26, Functions, game design process and using the game design process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Lesson 27, Final Project, Game Creation</td>
<td>Participant Artifacts</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Post-tests</td>
<td>Participant interviews</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Convert Audio to text, look for codes in text</td>
<td>Develop Categories and themes</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Upload data to JASP</td>
<td>Look for means in questions</td>
<td></td>
</tr>
</tbody>
</table>
Begin looking for central tendencies and variation in questions, paired t-tests

Rigor and Trustworthiness

Validity and Reliability

Validity is an indispensable aspect of research, and it asks the question “did we actually measure what we intended to measure?” (Mertler, 2020, p. 155). Reliability is another important aspect to ensure rigor and is related to the consistency of the collected data (Mertler, 2020). Using Google forms as the testing platform allowed for a more consistent testing environment that was reproduced for all participants in all classrooms. “Internal consistency is the most basic psychometric consideration for neural measures—and refers to the homogeneity of items on the test.” (Hajcak et al., 2017, p. 826). To determine the internal consistency using KR-20, Kuder-Richardson formula 20 to calculate the reliability coefficient which ranged from 0.00 to 1.00, where the consistency degree increases if the coefficient is closer to 1.00 and decreases if closer to 0.00 (Cho, 2016 & Mertler, 2020).

Qualitative

*Thick rich descriptions*

Detailed descriptions of the setting, participants, and process were used to paint a more complete picture of the research study. This included direct quotes from participant
interviews to illustrate what perceptions participants have of CT and CS from the beginning to the end of the intervention (Mertler, 2020).

**Triangulation**

There were five data sources and three different methods of data used for triangulation purposes: (a) pre-test, (b) post-test, (c) survey, (d) participant interviews, and (e) participant artifacts. Methodological triangulation is a robust approach for enriching the quality of research, mainly the credibility of findings (Krefting, 1991). It is built on the view of convergence of multiple types of data sources and methods to confirm that all attributes of a phenomenon have been explored (Morse, 1990). Triangulation was used to develop an understanding of the research questions, where measures of central tendencies of each data set to help summarize the large amounts of data that was collected. Having multiple types of data helped to provide depth to the findings. Having multiple sources and methods of data collection enhanced and supported the findings and conclusions (Mertler, 2020).

**Peer Debriefing**

To help verify the findings of my research, I met with the dissertation chairperson and Doctoral cohort members to review the data, my interpretations, and any problems in my research (Krefting, 1991). The process in determining the codes, categories and themes was also debated and reviewed by peers and participants in my cohort. Multiple meetings took place to certify rigor throughout the data analysis and interpretation phases. According to Mertler (2020), peer debriefing is helpful for improving my
research as others will be critiquing my methods of data collection, analysis, and interpretation.

**Audit Trail**

An audit trail is the process of keeping a journal or memos of the research process. I kept a research journal to document the process of creating codes, winnowing the categories, and developing the themes for the qualitative data. After each interview, I wrote notes about what stood out and what changes needed to be made for future interviews. This reflection helped to integrate my preliminary thoughts with my observation notes (Mertler, 2020). Guba (1981) noted a researcher should include documentation for any claims or interpretations to make certain that the data helps with the analysis and interpretation of the findings.

**Plan for Sharing and Communicating Findings**

The plan for sharing the research findings with all stakeholders is to present to each group. Participants received a summary of the data collected from them, such as any pre- and post-tests, surveys, and interviews. The data is organized in an aggregate manner and explained to them with charts and tables. After this presentation to study participants, any questions or feedback is used to better explain the findings to other stakeholders. The school administration received a report that described the study methodology and all findings. Moreover, a professional presentation was shared with the dissertation committee of the University of South Carolina where more feedback was gathered, and suggestions made for improvement. The sharing of findings will take place at the South Carolina Educational Technology Conference (SC EdTech) and similar conferences in
the field. In an effort to protect all participants’ identities, the researcher assigned them random numbers and pseudonyms, and their personal information was deleted at the conclusion of the study.
CHAPTER 4:

ANALYSIS AND FINDINGS

The purpose of this action research study was to evaluate how a block-based programming curriculum in game design affects CS participants’ CT skills, and their attitudes towards CT and CS at a South Carolina middle school. The data collection was aligned to the following research questions:

(1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills?

(2) How and to what degree were the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?

This chapter provides the quantitative and qualitative data analysis and findings that explain any changes in CT skills with middle school participants. The 16 participants completed all intervention lessons, pre- and post-tests, pre- and post-surveys, and created block-based programming artifacts. Four participants were randomly selected for one-on-one interviews.

This chapter is divided into three sections. The first section presents the quantitative data analysis and findings from the pre- and post-tests, surveys, and artifacts. The second section presents the qualitative data analysis, findings, and interpretations from the participants’ interviews. The final section presents a summary of quantitative and qualitative findings.
Quantitative Analysis and Findings

Pre- and Post-tests

The pre-test was administered to 16 participants before the 11-week intervention. The post-test was administered after the completion of all lessons, both tests contained the same 29 questions. This assessment included 17 teacher-created questions, nine questions from Code.org’s CS Discoveries curriculum, and three questions from MG-CSCI Middle Grades Computer Science Concept Inventory (Rachmatullah et al., 2020). The types of questions included were: two true-false, six short answer, and 21 multiple-choice questions with 4-5 choices. Ten of the multiple-choice questions included images of a program for participants to use in answering the questions as well as all six short answer questions. The CT skills tested in this assessment included algorithmic thinking (14 questions), abstraction (five questions), pattern recognition (five questions), and debugging (five questions) (see Table 4.1). This assessment was delivered using Google Forms where the participants viewed one question at a time and were given the opportunity to review their responses before submitting the completed assessment.

Face validity and reliability for this assessment was established with the teacher-created questions over a two-year period with previous participants enrolled in CS Discoveries classes. Code.org used a third-party evaluator to evaluate participant questions in the curriculum to assess validity for their assessments. The KR-20 method, or Kuder-Richardson Formula 20 was used to determine the tests reliability, the closer to 1 the more reliable the questions are. This is a method of calculating reliability for quantitative instruments and was used on the subscales of abstraction ($p = .46$), algorithmic thinking ($p = .86$), debugging ($p = .32$), pattern recognition ($p = .53$) the overall test ($p = .78$). According to DeVellis (2016), reliability coefficients of .70 or
above have acceptable reliability. Table 4.1 presents the breakdown for each of the skills and which questions are associated with them.

**Table 4.1 Reliability Analysis of Pre- and Post-test Overall, Subscales, and Test Questions Breakdown**

<table>
<thead>
<tr>
<th>Subscales</th>
<th>Number of items</th>
<th>Question Numbers</th>
<th>KR-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction</td>
<td>5</td>
<td>24, 28, 30, 33, 34</td>
<td>.46</td>
</tr>
<tr>
<td>Algorithmic Thinking</td>
<td>14</td>
<td>22, 25, 27, 29, 32, 36, 37, 38, 42, 43, 44, 45, *49, *50</td>
<td>.86</td>
</tr>
<tr>
<td>Debugging</td>
<td>5</td>
<td>23, 31, 35, *39, *46</td>
<td>.32</td>
</tr>
<tr>
<td>Pattern Recognition</td>
<td>5</td>
<td>26, 40, 41, *47, *48</td>
<td>.53</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>29</td>
<td></td>
<td>.78</td>
</tr>
</tbody>
</table>

*Note: * Short Answer; **Bold** include picture of a program or image

**Descriptive statistics**

The pre- and post-tests were first analyzed with Excel to ascertain the number of correct answers for all participants. The assessment tested four skills: algorithmic thinking, debugging, abstraction, and pattern recognition. JASP was used to analyze the descriptive statistics for the overall pre- and post-test scores. There was a noticeable increase in the participant’s overall scores between the pre-test scores (M = 11.38, SD = 3.32) and the post-test scores (M = 18.69, SD = 4.81) The greatest improvement was in algorithmic thinking followed by pattern recognition. The skill with the smallest improvement was abstraction (see Table 4.2).

**Table 4.2 Descriptive Statistics Pre- and Post-Test Skills Scores**

<table>
<thead>
<tr>
<th>Skills</th>
<th>Pre-Test M</th>
<th>SD</th>
<th>Post-Test M</th>
<th>SD</th>
<th>Increase in M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic Thinking Skills</td>
<td>6.20</td>
<td>2.65</td>
<td>9.25</td>
<td>2.38</td>
<td>3.05</td>
</tr>
<tr>
<td>Debugging Skills</td>
<td>1.80</td>
<td>.56</td>
<td>3.06</td>
<td>.77</td>
<td>1.26</td>
</tr>
<tr>
<td>Abstraction Skills</td>
<td>2.87</td>
<td>1.06</td>
<td>3.38</td>
<td>.96</td>
<td>.51</td>
</tr>
<tr>
<td>Pattern Recognition Skills</td>
<td>1.40</td>
<td>0.74</td>
<td>3.31</td>
<td>1.40</td>
<td>1.91</td>
</tr>
<tr>
<td><strong>Overall Scores</strong></td>
<td>11.38</td>
<td>3.32</td>
<td>18.69</td>
<td>4.81</td>
<td>7.31</td>
</tr>
</tbody>
</table>
Note: Maximum score for Algorithmic Thinking=14, Debugging=5, Abstraction=5, and Pattern Recognition=5; Maximum score for Pre- and Post-Test=29.

There was a wide variation in the overall post-test score given the large standard deviation ($SD = 4.81$). The standard deviation in the post-test was high. The distribution of scores (Figure 4.1) showed that participants’ scores ranged from 12-27 points, although it was noticeable that several participants scored 12-16 points, which is a relatively low score. Only three participants had high test scores between 25 and 27.

![Figure 4.1 Post-test Score Distribution](image)

The graph below (Figure 4.2) shows the distribution of scores for the pre- and post-test scores for algorithmic thinking, the blue lines are the pre-test scores, and red line are the post-test scores. This distribution shows improvement in this skill with five participants scoring at or above an 8 with the pre-test to 13 participants with the post-test. Figure 4.1 also shows how the distributions of scores increase from pre- 1 to 11 and the post 5 to 13. Fourteen participants scored higher than 7, the midline, on the post-test in algorithmic thinking nine scored below 7 on the pre-test.
Shapiro-Wilk Normality Test

The Shapiro-Wilk test for normality was administered on all the subskill and the post-test overall scores. These tests were completed using JASP to determine if the data are normally distributed (Gibbons & Chakraborti, 2021). The results of this test for the subscales were pattern recognition ($p = .97$), debugging ($p = .93$), abstraction ($p = .84$), and algorithmic thinking ($p = .94$) (See table 4.3). All subscales were normally distributed because the $p$-values are greater than the threshold of significance of $p \leq .05$. Since the Shapiro-Wilk normality test has been found to have normal distributions, it was determined that paired-samples t-test would be the most appropriate method of analyzing the data inferentially (Gibbons & Chakraborti, 2021).

Table 4.3 Shapiro-Wilk Normality Test

<table>
<thead>
<tr>
<th></th>
<th>Algorithmic Thinking Skills</th>
<th>Debugging Skills</th>
<th>Abstraction Skills</th>
<th>Pattern Recognition Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapiro-Wilk</td>
<td>.94</td>
<td>.93</td>
<td>.84</td>
<td>.97</td>
</tr>
</tbody>
</table>
Paired samples t-tests

The data from the pre-test and post-test were analyzed using the paired-samples t-test (See Table 4.4). To account for any type 1 errors, the Bonferroni Correction was used to lower the threshold of the p-value, hence \( p < .0125 \) was the new threshold. Results revealed that differences in three of the subscales were statistically significant in that all \( p \)-values are less than .0125. Specifically, participants’ algorithmic thinking (\( M = 6.20, SD = 2.65, t (14) = -3.11, p = 0.004 \)), debugging (\( M = 3.06, SD = 0.77, t (14) = -4.22, p < .001 \)), and pattern recognition skills (\( M = 1.40, SD = 0.74, t (14) = -4.50, p < .001 \)) had a statistically significant improvement. Conversely, differences in participants’ abstraction were not statistically significant (\( M = 3.38, SD = 0.96, t (14) = -1.20, p = .035 \)).

According to Cohen (1988), a \( d \) value of an 0.8 indicates a large effect size. A medium effect size was found for abstraction skills (\( d = 0.51 \)), and a large effect size was found for algorithmic thinking (\( d = .80 \)), debugging (\( d = 1.09 \)), pattern recognition (\( d = 1.16 \)), and overall test (\( d = 1.20 \)).

Table 4.4 Paired Samples T-Tests

<table>
<thead>
<tr>
<th></th>
<th>Pre-Test</th>
<th></th>
<th>Post-Test</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( M )</td>
<td>( SD )</td>
<td>( M )</td>
<td>( SD )</td>
<td>( t )</td>
<td>( df )</td>
<td>( p )</td>
<td>Cohen’s ( d )</td>
<td></td>
</tr>
<tr>
<td>Abstraction Skills</td>
<td>2.87</td>
<td>1.06</td>
<td>3.38</td>
<td>.96</td>
<td>-1.20</td>
<td>14</td>
<td>0.035</td>
<td>.51</td>
<td></td>
</tr>
<tr>
<td>Algorithmic Thinking Skills</td>
<td>6.20</td>
<td>2.65</td>
<td>9.25</td>
<td>2.38</td>
<td>-3.11</td>
<td>14</td>
<td>0.004</td>
<td>.80</td>
<td></td>
</tr>
<tr>
<td>Debugging Skills</td>
<td>1.80</td>
<td>.56</td>
<td>3.06</td>
<td>.77</td>
<td>-4.22</td>
<td>14</td>
<td>&lt;.001</td>
<td>1.09</td>
<td></td>
</tr>
<tr>
<td>Pattern Recognition Skills</td>
<td>1.40</td>
<td>.74</td>
<td>3.31</td>
<td>1.40</td>
<td>-4.50</td>
<td>14</td>
<td>&lt;.001</td>
<td>1.16</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td>11.38</td>
<td>3.32</td>
<td>18.69</td>
<td>4.81</td>
<td>-7.99</td>
<td>15</td>
<td>&lt;.001</td>
<td>1.20</td>
<td></td>
</tr>
</tbody>
</table>

Note. \( n=16 \)
Pre- and Post-Survey of Participant Attitudes

The pre- and post-survey was administered to 16 participants before and after the 11-week intervention to measure changes in participant attitudes towards CS and CT. The post-survey was administered after the completion of all lessons, and both surveys contained the same 21 questions. Fourteen questions were on a Likert scale ranging from strongly disagreed (1) to strongly agree (5). For example, there were items such as “I like computer science” and “I believe I can solve most of the problems I face if I have sufficient time and if I show effort.” The questions covered different topics and attitudes (See Table 4.5) and two composite scales were created to better analyze participant attitudes and beliefs they hold about CT and CS. Seven questions were about participants’ demographic information. All the reliability coefficients of each of the composites within both administrations fall within the range of .74 to .86. According to DeVellis (2016), reliability coefficients of .70 and above have acceptable reliability.

Table 4.5 Survey Questions Focus and Composite Scales

<table>
<thead>
<tr>
<th>Question Numbers</th>
<th>Cronbach’s Alpha</th>
<th>Composite Scales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographic</td>
<td>1-6, 21</td>
<td>N/A</td>
</tr>
<tr>
<td>Attitudinal</td>
<td>7-9, 11, 18-20</td>
<td>.86</td>
</tr>
<tr>
<td>Algorithmic</td>
<td>12, 13, 15</td>
<td>.74</td>
</tr>
<tr>
<td>Thinking</td>
<td>Creative Thinking</td>
<td>10, 14, 16, 17</td>
</tr>
</tbody>
</table>

Eight survey questions were taken from Korkmaz et al.’s (2017) Computational Thinking Scale. It was created in 2017 to be used with college-level Turkish students, and it was later translated into Chinese to be used with high school students. It was determined that it can be used to determine students' computational thinking skills (Korkmaz et al., 2017). The three subscales in the survey used for this study were algorithmic thinking, creativity, and critical thinking. These questions were more closely
aligned to the focus of this study. There is no data using this with middle school participants, and there is no data about whether using part of the scale will affect the results. All the reliability coefficients of each of the subscales within both administrations fall within the range of .79 to .87. According to DeVellis (2016), reliability coefficients of .70 and above have acceptable reliability.

**Descriptive statistics**

The pre- and post-survey data was first analyzed with Microsoft Excel, then JASP was used to create the pre- and post- survey composite scores for participant attitudes and CT beliefs. There was a marginal increase in participant attitudes towards CS and CT comparing the mean scores for pre- \((M = 25.31, SD = 4.90)\) and post-attitudes \((M = 25.75, SD = 5.46)\) (see Table 4.6). There was an increase in participants’ CT beliefs from the pre-survey \((M = 24.25, SD = 3.98)\) to the post-survey \((M = 25.95, SD = 6.01)\). The overall survey composite scores showed a similar increase in mean scores from the pre-survey \((M = 49.56, SD = 8.51)\) to the post-survey \((M = 51.69, SD = 11.23)\).

<table>
<thead>
<tr>
<th>Table 4.6 Descriptive Statistics-Pre- and Post-Survey Composites</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre-Survey</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Attitudes Composites</td>
</tr>
<tr>
<td>CT Beliefs Composite</td>
</tr>
<tr>
<td>Overall</td>
</tr>
</tbody>
</table>

The results of the participant attitudes subscale from the pre- and post-survey showed an increase in the scores. Figure 4.3 shows the pre- and post-scores for each participant. Eight participants increased their score between two to seven points, and a decrease with seven participants between one to seven, and one that remained the same (see Figure 4.3). The computational thinking beliefs subscale revealed similar changes. Nine participants’ scores increased between one to thirteen, six decreased between one to five, and one remained the same (see Figure 4.4).
The Shapiro-Wilk test for normality was used on all the composite subscales and the overall surveys. These tests were completed using JASP to determine if the data are normally distributed. The calculated $p$-value for post-attitude composite ($p = .96$), post-CT beliefs composite ($p = .95$), and post-overall survey ($p = .97$) (See Table 4.7). Both composites were normally distributed because the $p$-values were greater than the threshold of significance of $p \leq .05$. To account for any type 1 errors the Bonferroni
Correction was performed on the \( p \)-value, the new value (\( p < .025 \)) is the new threshold. Both subscales are normally distributed because the \( p \)-values are greater than the threshold of significance of \( p < .025 \). Since the Shapiro-Wilk normality test has been found to have normal distributions, it was determined that paired-samples t-test would be the most appropriate method of analyzing the data inferentially (Gibbons & Chakraborti, 2021).

**Table 4.7 Shapiro-Wilk Normality Test Survey**

<table>
<thead>
<tr>
<th>Shapiro-Wilk</th>
<th>Post-Attitudes</th>
<th>Post-CT Beliefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>.96</td>
<td>.95</td>
<td></td>
</tr>
</tbody>
</table>

**Paired Samples T-Tests**

The data from each composite score for pre-attitudes composite (\( M = 25.31, SD = 4.9 \)), post attitudes composite (\( M = 25.75, SD = 5.46, t (15) = -.43, p = .34 \)), and pre-CT beliefs composite (\( M = 24.25, SD = 3.98 \)), post-CT beliefs composite (\( M = 25.94, SD = 6.01, t(15) = -1.41, p = .09 \)), along with the pre-overall (\( M = 49.56, SD = 8.51 \)), and post-overall composite (\( M = 51.69, SD = 11.23, t(15) = -1.05, p = .16 \)) scores were analyzed using the paired-samples t-test (see Table 4.8). Results revealed that the differences between the pre- and the post-surveys were not statistically significant for any of the subscales. Cohen (1988) determines that, if the \( d \) value is between 0.2 and 0.3, there is a small effect size. The results for attitudes (\( d = .11 \)), CT beliefs (\( d = .35 \)), and overall survey (\( d = .26 \)) suggest that the difference represents a small effect.

**Table 4.8 Paired Samples T-Test Survey and Composites**

<table>
<thead>
<tr>
<th></th>
<th>Pre-Survey</th>
<th>Post-Survey</th>
<th>( t )</th>
<th>( df )</th>
<th>( p )</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attitudes Composite</td>
<td>25.31</td>
<td>25.75</td>
<td>-0.43</td>
<td>15</td>
<td>.34</td>
<td>.11</td>
</tr>
</tbody>
</table>
Participant Artifacts

At the end of the intervention, participants created games in a block-based programming environment at the completion of all lessons. Collecting and analyzing results with a rubric gave the researcher the ability to see how well participants can use the CT skills that were taught, and scores indicate the different levels of achievement for each of those skills (algorithmic thinking, debugging, abstraction, and pattern recognition) for a maximum score of 100 points (see Table 4.9). This rubric was created by Code.org (see Appendix D), with seven areas to evaluate each participant’s programs.

The researcher was the only scorer for the artifacts.

**Table 4.9 Rubric Points and CT Focus**

<table>
<thead>
<tr>
<th>Criteria</th>
<th>CT Focus</th>
<th>Maximum Points Available</th>
<th>Levels of Achievement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Extensive Evidence</td>
</tr>
<tr>
<td>Program Development</td>
<td>Algorithmic Thinking</td>
<td>10</td>
<td>7 - 10</td>
</tr>
<tr>
<td>Program Readability</td>
<td>Debugging</td>
<td>10</td>
<td>7 – 10</td>
</tr>
<tr>
<td>Use of Functions</td>
<td>Abstraction</td>
<td>10</td>
<td>7 – 10</td>
</tr>
<tr>
<td>Backgrounds and Variables</td>
<td>Algorithmic Thinking</td>
<td>10</td>
<td>7 – 10</td>
</tr>
<tr>
<td>Interactions and Controls</td>
<td>Algorithmic Thinking, Pattern</td>
<td>25</td>
<td>19- 25</td>
</tr>
<tr>
<td></td>
<td>Recognition Debugging</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Position and Movement Pattern Recognition Abstraction
Variables Algorithmic Thinking, Abstraction

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic Thinking</td>
<td>24.88</td>
<td>8.84</td>
</tr>
<tr>
<td>Debugging</td>
<td>14.75</td>
<td>3.43</td>
</tr>
<tr>
<td>Abstraction</td>
<td>20.31</td>
<td>6.48</td>
</tr>
<tr>
<td>Pattern Recognition</td>
<td>15.38</td>
<td>4.78</td>
</tr>
<tr>
<td>Overall Scores</td>
<td>75.06</td>
<td>22.63</td>
</tr>
</tbody>
</table>

Note: Maximum Score: AT=34, Db=18, Ab=27, PR=21, Overall=100

Descriptive statistics

The rubric scores were entered into an Excel spreadsheet and then analyzed with JASP. The results showed that most of the participants created games that were original and entertaining. The overall artifact scores were \(M = 75.06, SD = 22.63\), and the subscale score results for algorithmic thinking \(M = 24.88, SD = 8.84\), debugging \(M = 14.75, SD = 3.43\), abstraction \(M = 20.31, SD = 6.48\) and pattern recognition \(M = 15.38, SD = 4.78\) (see table 4.10).

Table 4.10 Descriptive Statistics-Artifact Rubric Scores

Of the 16 study participants, 14 received an overall passing score of 60 or higher, of those 9 participants earned a score of 80 or above (see Figure 4.5). The distribution of scores for the subscales shows most of the participants scored in the upper 50%.

Regarding the algorithmic thinking subscale, 12 participants scored 20 or more points out of the 34 possible (Figure 4.6). In the abstraction subscale, 11 participants scored 20 or more points out of a possible 27 (Figure 4.7). In the debugging subscale, 10 participants scored 15 or more points out of a possible 18 (Figure 4.8). And in the pattern recognition subscale, 11 participants scored 16 or more points out of a possible 21 (Figure 4.9).
Fourteen of the participants created games that showed their CT understanding. Two participants did not complete their games due to the time restrictions.

**Figure 4.5** Artifact- Overall Scores Lowest to Highest

**Figure 4.6** Artifact- Algorithmic Thinking Scores Distribution Note: Maximum Score 34

**Figure 4.7** Artifact- Abstraction Scores Distribution Note: Maximum Score 27
Qualitative Analysis, Findings, and Interpretations

After the intervention was complete, qualitative data were collected for this study through participant interviews. The interviews included a random sampling of four participants who agreed to join 20-minute one-on-one interviews about their experiences during the intervention. Questions allowed the researcher to delve deeper into what participants understood about CT and CS, as well as ascertain their attitudes towards CS.

Participant Interviews

Participants’ numbers on slips of paper were placed into a hat, and four were drawn to have one-on-one interviews with the researcher. These interviews took place face-to-face in the classroom, and they were recorded using Garage Band. A Macintosh desktop and a microphone were used to record better quality audio. All files were saved and transferred to a flash drive. Sample interview questions included “Explain how loops
work?”, and “When you encountered a problem in your code without an obvious answer, what steps did you take to solve it?” where participants expressed their CT skills.

Attitudinal questions were included as well, such as “What did you enjoy most in the computer science class?” and “Do you see yourself as a computer scientist?” to understand their feelings about CS and the class overall. All questions were open-ended, where the answers should not have been “yes” or “no”. If so, follow-up questions were asked to further explore their understanding and perspectives. Questions about their final projects/artifacts were asked, and these questions focused on the code used to create their projects and to assess participants’ understanding of the CT and CS concepts.

The interviewees consisted of four participants, 2 females and 2 males. The average age of the interviewees was 13.75 years old. Twenty-five percent identified as White/Caucasian by their parents, 25% identified as Asian, 25% identified as African American/Black and 25% identified and more than one ethnicity. All participants were in the 8th grade. Three participants reported that they previously participated in a CS course before in the 6th grade, which was also taught by the researcher (see Table 4.11).

<table>
<thead>
<tr>
<th>Pseudonym</th>
<th>Age</th>
<th>Gender</th>
<th>Grade</th>
<th>Previous CS Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>14</td>
<td>M</td>
<td>8TH</td>
<td>Yes</td>
</tr>
<tr>
<td>Mary</td>
<td>13</td>
<td>F</td>
<td>8TH</td>
<td>No</td>
</tr>
<tr>
<td>John</td>
<td>14</td>
<td>M</td>
<td>8TH</td>
<td>Yes</td>
</tr>
<tr>
<td>May</td>
<td>14</td>
<td>F</td>
<td>8TH</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.11 Interviewee’s Demographic Information

Round One of Coding

When all interviews were completed, they were transcribed with Temi, an audio to text transcription website. After carefully listening to and reading the transcripts multiple times to ensure accuracy, the transcripts were then uploaded to Delve, which is a qualitative data analysis tool. The first round of coding started using open coding, when
responses are assigned codes to label them. Open coding entails data that are assigned codes to label them without a pre-existing coding scheme (Williams & Moser, 2019). During this round, participant responses were also coded using in-vivo coding using participants’ direct quotes to create codes and simultaneous coding when more than one code is applied to a quote multiple time (Saldaña, 2015). For example, when asked “What is a loop?” a participant’s response was “I used them for resetting sprites positions.” Position and sprite were used to code that sentence. Value coding was used as well during this round by identifying participants’ feelings, experiences, beliefs, and perspectives to create codes (Saldaña, 2015). An example of this is when a participant responded to the question “what aspects of coding did you like the most?” by saying “creating your own games and websites, which is very fun. I've always wanted to learn how to create those.” The codes games, fun, and like coding were added.

While continuing the first-round, structural coding was incorporated by aligning codes with the interview questions. Structural coding allows for responses that answer direct research questions to be grouped together (Saldaña, 2015). This led to computer scientist, recommend, disliked about code, loops, conditional, variable, counter pattern, and debugging to be added as new codes. A complete list of rounds one codes is provided in Appendix I.

**Round Two of Coding**

This round of coding involved reviewing the current codes and looking for patterns, similarities, and connections to the research questions (Saldaña, 2015). The idea was to group similar responses together to narrow down the codes into one category, called pattern coding (Saldaña, 2015). For example, debugging, solution, and collaboration were all grouped under the category solving coding problems. See Figure
4.10 for the complete list of categories and the codes under each that was completed in Delve. To help understand the new categories descriptions were created along with each category in Delve as seen in Figure 4.11.

Figure 4.10 Complete List of Categories and Codes included

Figure 4.11 Categories with Descriptions
Peer Debriefing

A meeting with the dissertation chair was held to use the categories and codes to identify themes. We looked for categories that could be combined to create themes in sentence form. Starting with the two categories *programming skills* and *enjoy/like*, we looked at the codes with the highest number of snippets within them. They were *block-based coding enjoyed, score, position, and game*. Thinking back to when these skills were taught, I recalled that instruction used video games and the gaming concepts to teach the participants how to increase the score when a sprite touched another sprite using positioning. This, in combination with conceptual understanding, helped to create the first theme, *elements of game-based learning enable participants to use programming skills to create their own game*. 

Further review of the categories and instructional elements helped to create the next theme. From the codes of *fun, sprite, coding/program*, and the category of *dislike/not enjoyed* we developed an idea about why the participants enjoyed programming. I explained that to help participants focus on the skills being taught in each lesson, they were asked not to create new sprites but encouraged to use what is available in Code.org. And when they were creating their mini and final projects, they could create sprites however they wanted, thus allowing for freedom to create and program. *Balancing creative freedom and structure led to more enjoyment of the coding process* (see Table 4.x below).

For the third theme we wanted to look at the category of *solving coding problems and disliked/not enjoyed* (see Table 4.x below). With this category we discussed how participants developed the skill of debugging and how that skill was diminished due to the heavy use of block-based coding. Participants struggled to see “bugs” in their code as
well as believing that code could be incorrect if it was a block, they “brought into” the program. The use of collaboration worked to overcome this deficiency. During class time, participants were allowed to look at each other’s code to help find bugs and work together to find solutions. Then the third theme emerged: Collaboration promoted debugging of participants own programs (see Table 4.12 below).

Table 4.12 Codes, Categories and Themes

<table>
<thead>
<tr>
<th>Theme</th>
<th>Category</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements of game-based learning enabled participants to use programming skills to create their own game</td>
<td>Programming Skills</td>
<td>Conditional, Score, Counter Pattern, Game, Variable, If then, Position, Sprite, Loops, Coding/Program</td>
</tr>
<tr>
<td>Balancing creative freedom and structure led to more enjoyment of the coding process.</td>
<td>Enjoy/Like</td>
<td>Like Code, Block-Based Coding, Enjoyed Fun</td>
</tr>
<tr>
<td>Collaboration promoted debugging of participants own programs.</td>
<td>Solving Coding Problems</td>
<td>Collaboration, Debugging, Solution, Block-Based Coding, Least Liked Dislike Code</td>
</tr>
</tbody>
</table>

Qualitative Themes

The themes emerged from the codes and categories to explain what impact the instructional unit had on the participants CT skills and attitudes towards CS. The three overarching themes were developed following the qualitative analysis. The themes were created from responses from the participants during the one-on-one interviews. Qualitative themes are presented below.
Theme 1: Elements of game-based learning enabled participants to use programming skills to create their own game.

This theme was developed from programming skills due the overarching theme of the intervention that students would create a game when all skills were learned. Participants used their experiences playing video games to related to what they were learning in the intervention. The skills taught during the intervention also used this connection to aid in instruction and retention. The concepts included how to earn points, how to move the character around the scene, how other items interact with the character, and what needs to happen to win or lose the game. Game-based learning is an educational strategy where students learn either by playing games or creating them. Torchi et. al. (2019) stated that by designing games, participants are exposed to basic programming concepts and this supported learning programming and CT skills.

Two participants that were interviewed used the concept of ‘game’ to explain programming concepts. For example, Joe said that “I wanted the game to end because if somebody has something like negative one and if they continue playing, then the game will never stop”. When participants say game, they mean video game which is a concept they feel comfortable with and the instructor used when teaching CT skills. Video games where people play on a device either cell phone or PC where there are characters and rules to play as well as an end result. Similarly, May stated “the loop I used is that I wanted when the player touches the sprites, the sprites had to go in a different place.”

The following categories are subsumed by this qualitative theme.

**Programming skills.** This category includes many of the skills participants learned during the intervention. The first is score, which is used to determine a winner or loser. Adding to or taking from the current score as well as knowing the current value is
determined using variables, the counter pattern and sprite interactions. When asked about variables in the interviews all participants answered using score as their example in how to add and subtract using variables as well as how the winner or losers were determined. The participants were asked the question “what is a variable and how does it work?”, to which Mary stated a “variable is a number that is subject to change. So, um, in scoring, you could say every time your sprite touches an object, you will gain a point”. And May said, “if the player touches the enemy, the health would go down, but if you touch like a candy, your scoring will go up.”

The next skill is the if then. This skill was included in the programming skills theme as well because of how it directly relates to game design and creation. The use of if then statements or conditionals gives the creator the ability to check for a variety of situations that arise when creating games, examples of this are, “if the score is higher than 10”, or “if the right arrow is pressed.” The participants understood the concept of if then statements. When Mary answered the question “what is a conditional?”, the response was “a conditional statement asks if a certain, uh, aspect is true or false and based off of whether that aspect is true or false, it will perform a certain action”. Most participants in the interviews had similar answers.

Another skill included is coding/program. Coding emerged from the data collected from the heavy use of the word when discussing programs throughout the intervention with the participants. The participants understood the word meant the lines of instructions they created to make their game work. Popat & Starkey, (2019) stated that students’ “see coding as a way for people to express and share their ideas”, so coding is how students can show us what they have learned. For instance, Mary stated, “I like to be able to create code and then watch it actually work.”
Sprites are also important in developing a game. The word sprite is what programmers call a two-dimensional image which is what we use as characters and background items in a game. Participants liked the word, thought it was funny and used it to describe all images adopted in their programs. Participants learned how to move them with the arrow keys using if statements, the counter pattern, and loops. John stated, “the loop will allow the fish sprite to reset back to the left of the screen”. This means that his sprite, the fish, would start over on the right side of the screen once the if then statement tracking it was found to be true, or fish’s x position was greater than 400.

**Theme 2: Balancing creative freedom and structure led to more enjoyment of the coding process.**

To be able to have the freedom to create a game that you want is powerful. The process to get there has structure which includes; planning, project requirements, and CT skills. Planning means to decide what the game will look like, how it will work and how to play it. Project requirements include what types of coding structures does the planning need to work, what concepts does the project require to receive a good grade and how the code should be organized. CT skills are programming elements learned during the intervention and where apply them to the project. Once participants learned the skills during the intervention, they recreated two different games step-by-step. This is where they learned where to start in the game design process and what the steps were to create a game. An important question asked in the interviews, “What did you enjoy the most?”. Mary stated, “I like to be able to create code and then watch it actually work”, and “I enjoy creating games once I know how to make them.” The freedom to choose their own sprites, game design and code added were important for participants. The categories that formed this theme are presented as follows.
Enjoy/Like. Having fun doing an activity will motivate and increase positive attitudes for coding (Papavlasopoulou et al., 2019). The participants seemed to enjoy the process and worked very hard on their games. May stated that “I'd just say creating your own games and websites is very fun. I've always wanted to learn how to create those.” The question “Do you like CS?”, Joe said “I just like coding in general. It helps me understand how like video games and stuff like that work”. And May stated, “I thought it wasn't gonna be, but now it's very, I'm having a lot of fun doing it.”

Dislike/Not Enjoyed. Some participants did have some difficulties. And some disliked coding. Some didn’t like how the internet could slow them down when working, and others were overwhelmed by the number of blocks available. For example, May stated “I messed something up then it deletes, then you to like redo all the coding” addressing the frustration with website and internet issues. As well as Mary also stated “when we had a lot of different blocks to work with it is it takes more time to find the blocks than the beginning, which makes it take longer to code” the dislike of too many blocks available and slowing down the process of coding. On the other hand, Joe said that he disliked other classmates being disruptive by arguing that “we try focus on your assignment, it'll be like loud and stuff.”

**Theme 3: Collaboration promoted debugging of participants’ own programs.**

Collaboration is working with a team to complete, correct, and plan a program. During this intervention participants were encouraged to ask their neighbors if their code was not working or if they were stuck and did not know what to do or look on with another participant and analyze how that programmed worked before asking the teacher for help. Collaboration was encouraged with the participants every day. This step allowed for participants to take ownership of their work and to understand that collaboration is not
cheating. Seeing others’ code helps participants to see what they missed and to see their own errors easier, working with others in the development of coding helps students develop their CT skills. When asked about working with a peer during the interview, May stated that “I asked one of my friends, ‘cause she's very helpful for me. She helped me a bit for the velocity stuff. And then the rest I was able to figure out on my own.” Mary stated that “it was very helpful”, collaborating with a classmate. These statements show that the use of collaboration in class helped participants feel at ease and understood the CS concepts better. Qualitative categories combined to create this theme are presented next.

**Solving coding problems.** The concept of debugging is a difficult skill to develop. Throughout the intervention, participants were given code that had bugs and were told to fix the code or identify the bug. This process helped the participants develop the problem-solving skill and their ability to find and fix bugs in their own programs. For example, Joe stated that “I try to look back at the code and make sure that there's no spelling errors.” This is a great first step in the debugging process and to find the solution to unexpected errors.

The use of collaboration enhanced the debugging process, where other participants looked at their code and helped identify bugs. Along those lines, looking at another’s code can give the students clues to what their bug might be. An example of this is when May said, “what she did is she showed me a bit of hers and that showed me how to deal with mine”. This helped her find the solution to her coding bugs, so the game worked as expected.
Chapter Summary

This chapter presented the data analysis methods and presented the quantitative and qualitative findings and themes from the data that was collected. Quantitative data from the pre- and post-tests and the artifacts used descriptive statistics, Shapiro-Wilk normality test, and paired samples t-tests to address RQ1. The data from the pre- and posts-tests’ attitudinal surveys were analyzed using descriptive statistics, Shapiro-Wilk normality test, and paired samples t-tests to address RQ2. The findings related to RQ1 revealed that there were improvements in all skills tested which included algorithmic thinking, debugging, abstraction, and pattern recognition. The greatest improvement was in algorithmic thinking and pattern recognition while abstraction had the lowest amount of improvement. The findings related to RQ2 found that there was a marginal increase in participant attitudes towards CS and their CT beliefs.

Qualitative data from the participant interviews were analyzed using thematic analysis. The qualitative findings revealed the three themes: (1) Elements of game-based learning enable participants to use programming skills to create their own game; (2) Balancing creative freedom and structure led to more enjoyment of the coding process; and (3) Collaboration promoted debugging of participants’ own programs. Theme one identified that the participants learned the concepts of CT with their increase in the use and understanding of programming skills, while theme two identified that participants enjoyed the freedom to create their own games, and theme three identified that participants used collaboration to help solve their coding problems and debug their programs.
CHAPTER 5:
DISCUSSION, IMPLICATIONS, AND LIMITATIONS

The purpose of this action research study was to evaluate how a block-based programming curriculum in game design affects middle school CS participants’ CT skills, and their attitudes towards CT and CS at CE Williams Middle School. The study focused on the following research questions; (1) How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills? (2) How and to what degree were the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming? This chapter discusses the implications and limitations of this study and how the findings relate to the degree that block-based programming affect students CT skills and attitudes toward CS and CT. This chapter shares the (a) discussion, (b) implications, and (c) limitations of this study in the following sections.

Discussion

Quantitative and qualitative data were triangulated in an attempt to understand the impact the BBPE had on participants’ CT skills and attitudes towards CS and CT. The quantitative data findings from this study suggest that a BBPE can enhance students’ CT skills, in that three of the four CT skills saw improvement from the pre- to the post-test. However, students’ attitudes towards CS and CT did not show improvement. The qualitative data supports that participants attitudes showed no noticeable change towards CS and CT. The following sections discuss the findings for RQ1 and RQ2 considering related literature.
RQ1: How and to what degree did Code.org’s block-based programming curriculum in game design affect middle school students’ CT skills?

Wing (2019) stated that CT is a fundamental skill and knowing CT will give that person an advantage in understanding and using that skill over those who do not. Students have had limited exposure to CS, and this has diminished their opportunities to develop CT skills (Brown et al., 2014). Education in CS made the leap from teaching students to consume technology to teaching students to be creators within the past 10 years (Kafai, 2016; Runciman, 2011). Despite the development of a plethora of new programming initiatives and curricula, students are still experiencing the same issues; it is difficult to learn, to understand, and to master (Mladenović et al., 2018). One way to develop middle school students’ CT skills was to use Code.org’s CS Discoveries game design curriculum that utilizes BBPE in the classroom. The CT concepts taught in Code.org’s curriculum include debugging, abstraction, algorithmic thinking, and pattern recognition (Kalelioglu, 2015). Data analysis revealed that before and after the intervention three of the subscales were statistically significant. Specifically, participants’ algorithmic thinking, debugging, and pattern recognition skills had a statistically significant improvement. Conversely, abstraction was not statistically significant. As well 14 out of the 16 participants experienced an improvement in their overall test scores. This research question is addressed in five sections (a) Block-based Programming, (b) Algorithmic Thinking, (c) Debugging, (d) Pattern Recognition, and (e) Abstraction.

**Block-based Programming**

Kale & Yuan, (2021) stated that “programming is an effective approach providing learners with such an opportunity to explicitly practice CT skills” (p. 623). Code.org CS Discoveries is a block-based programming environment that includes lessons for students
to learn how to design and create games, learn CT skills without the burden of memorizing the syntax, and develop higher order thinking skills (Kalelioglu, 2015). Stated “that block-based programming, in conjunction with effective pedagogy can serve as an effective way to develop important computational thinking skills” (p. 3). Further, block-based programming has a direct impact on students’ experiences with programming and learning CT skills (Weintrop & Wilensky, 2017). BBPE prevent syntax errors in that not all blocks will fit together because commands can only be used in specific places. This study used an 11-week intervention to teach CT and is in line with other studies where skill development requires an adequate amount of training time and practice (Atmatzidou & Demetriadis, 2016; Bers et al., 2014b; Clark et al., 2016). The use of games in learning to programming has shown to be more likely to develop CT knowledge efficiently (Quahbi et al., 2015). Schererrer et al., (2020) stated as well that “game-based interventions were indeed most beneficial to higher-grade students’ learning of computer programming” (p. 4). My results were similar to the studies listed above, Code.org’s BBPE provided students with an environment of game like activities focused on CT skills and participants’ had a significant increase in those skills from the pre-test to the post-test.

Along with the pre- and post-tests, student created their own games made in Code.org’s block-based Game Lab that were used as artifacts. Participants’ scores showed a significant development of their CT skills, over 87% of the participants scored a 60 or better with their games. With 75% of the participants scoring “extensive evidence”, the highest rating available on the rubric, as well as in the subskills of algorithmic thinking and pattern recognition. Moreover, 68.75% of the participants scored the rating of “extensive evidence” in abstraction and 62.5% of the participants in
debugging. These scores showed that students can program and understand computationally.

**Algorithmic Thinking**

Algorithmic thinking is the ability to think in steps or sequentially to solve problems (Chuechote et al., 2020). Previous research studies that taught using code.org curriculums students showed an improvement in their algorithmic thinking skills (Chuechote et al., 2020; Lockwood et al., 2016; Oluk & Çakir, 2021; Peel & Friedrichsen, 2018; Tonbuloglu & Tonbuloglu, 2019). Yildiz Durak (2020) confirmed that block-based programming will help students develop CT skills by providing opportunities for creativity, critical thinking, and develop algorithmic thinking behaviors by creating solutions to complex problems. Harangus & Kátai (2018) noticed in their study that students who mastered algorithmic thinking “were able to solve tasks demanding thinking, not only using the rules and algorithms they had learned” (p. 7). BBPE serve as an effective tool to develop algorithmic thinking and prepare students for future CS courses (Weintrop, 2019). These environments help students focus on the process of learning rather than the syntax of the programming language. Likewise, Weintrop's (2019) study explained that using a BBPE like Code.org allow for students to focus on building the plan, or to think algorithmically. With their emphasis on this skill “Code.org lessons have the potential to help students develop such key CT skills” (Kale et al., 2022, p. 12). The studies above had similar results with the development of algorithmic thinking skills and the participants in this study had statistically significant increase in algorithmic thinking using Code.org and the BBPE.
Debugging

Debugging is essential in that finding and fixing errors in one’s programs or their classmates’ programs is part of the development of CT. Kim et al.’s (2018) reported that “learning to debug while learning to program improved programming concept understanding and programming performance” (p. 768). The ability to find and fix their own bugs is an important skill, but if they get stuck, they should seek help from their classmates. The teacher encouraged the use of collaboration when participants realized there were bugs in their programs that they struggled to fix. Bugs often are highlighted on the screen in either red or yellow to show where the error occurs. The errors that occurred most were spelling errors which can be fixed quickly. But other errors may be more difficult to identify, and this is where having a fellow classmate to look over the code to see if they can identify the bugs is helpful to both participants. John stated in the interview that when he had a program that did not work, he would “try to look back at the code and make sure that there's no spelling errors”. If the bug went unfound, then the instructor would look over the code and provided assistance. This process helped participants in learning coding and was reported by Papavlasopoulou et al.’s (2019) study where students that collaborated or worked together “had a higher level of shared understanding and could communicate better during the coding activity” (p. 421). Kim et al.'s (2022) study also reported that when students are allowed meaningful struggle or given an opportunity to find and fix errors alone or with student collaborators, they were more successful at programming. Collaboration with debugging resulted in students being more persistent when coding individually. This helped to solve coding problems as well as contributed to increases in programming skills (Margulieux et al., 2020; Tonbuloğlu & Tonbuloğlu, 2019; Turchi et al., 2019). This study showed similar results
to the previous studies, in that the pre-/post-test revealed a statistically significant increase in debugging and improved students’ ability to program.

**Pattern Recognition**

In pattern recognition, students see similarities between coding elements that can be matched with previous tasks (Qian & Choi, 2022). Pattern recognition is part of CT and is a necessary competency to solve a problem that involves locating similarities or patterns from one problem to the next (Barrón-Estrada et al., 2022). Yasin & Nusantara's (2023) study discussed how pattern recognitions characteristics for problem solving could be identified by “understanding the problem by looking for information on the problem context” and “extracting the problem's components” (p.1). When encountering a new problem, participants can think back to previous problems to look for similarities and develop a plan. During the interventions curriculum repeatedly referred back to programs and parts of programs that students previously completed. This was to show students that programs with similar activities will use the same code. An example of this is when students first learned to add to the score to show progress in a game, this piece of code was used in every lesson after as the activities became more challenging. Barrón-Estrada et al.’s (2022) study, which used a mobile application to develop pattern recognition skills in students. This study saw similar results that pattern recognition scores improved after the intervention and that it was statistically significant.

**Abstraction**

The idea of abstraction or the use of functions is to hide parts of the program that were reused and were called when needed. For example, adding points to the score, we do not need to see the actual code for that, we can call that function when it is needed and keep the code out of eyesight. Usually, that part of a program is what can be used
repeatedly from other programs, and participants either did not understand that the code is the same and could be copied and reused or they did not realize they could copy it from their previous work. Based on previous research, “older students were found to do better on the abstraction task than students in the youngest age group” (Rijke et al., 2018, p. 86). The participants in this study may not have been mature enough to develop this skill. Jean Piaget’s theory of cognitive development stated that until the age of 12 children are still forming schemas, which make abstract reasoning difficult (Cherry, 2022). So, it is possible that the low scores for my study were due to lack of cognitive development in participants. The similarity to Rijke et al.’s (2018) study where younger students did not show development of their abstraction skills is consistent. Female participants did do better than the male participants on the abstraction skills (M=0.62), and this is supported by Rijke et al.’ (2018) study as well. The authors reported that female students did achieve better scores on abstraction because “after the age of 9.5 years old, female students begin to outperform their male peers on the abstraction task” (p. 87). However, there were no statistically significant differences in participants’ abstraction skills, though there was minimal improvement of 0.51, which supports the findings reported in Kale’s (2021) study where abstraction and decomposition improvements were significantly lower at the end of the study. This is due to the skill of abstraction being not a primary focus in the code.org curriculum (Kale & Yuan, 2021).

RQ2: How and to what degree were the differences in middle school students’ attitudes towards CT and CS after participating in a unit in block-based programming?

Students’ attitudes are attributed to student understanding of the concepts taught and successes, as well as their beliefs in their CT abilities. Lewis et al.’s (2010) study
reported that students’ attitudes as well as their beliefs in their abilities are tied to performance. “There is a relation between children’s attitudes and their cognitive processes while coding. Highly motivated children with positive attitudes have the ability to handle cognitive load and better manage the construction of their artifacts” (Papavlasopoulou et al., 2019, p. 421). This research question is addressed in two sections below: (a) student attitudes towards CT, and (b) student attitudes towards CS.

**Student Attitudes Toward CT**

The participants of this study liked coding in BBPE. This is similar, to Kalelioglu’s (2015) study, in which students reported that they liked coding in the Code.org site and desired to learn more about programming. Gunbatar & Karalar's (2018) study stated, “that visual programming environments can increase students' self-efficacy perceptions and attitudes toward programming” (p. 931). Along these lines, Weintrop & Wilensky (2017) found that when students who learned coding in BBPE reported more of an interest in future programming courses. Hromkovic & Staub's (2019) study reported that making games proved to be compelling task for learning computational concepts and expanded the participants’ perspectives and interest in coding. Usually, if students like something they are more apt to learn and have success.

Of the many concepts taught, conditionals are considered a major aspect of CT. In Brennan & Resnick’s (2012) study not all students could correctly describe how a conditional worked and why they were used. This can be contributed to lack of belief in their CT abilities. In this study, one interviewee had difficulty answering the concept questions but knew how to complete the task. This may be attributed to students not necessarily knowing the term, but they understood where to use the skill in which it is useful but not why (Agnihotri, 2021). Lambić et al.'s (2021) study concluded that a
limited ability to use algorithmic thinking and problem-solving skills in some students can be the reason for a less positive attitude towards programming and CT develops. Perhaps the lack of statistically significant differences in this study is due to the many concepts taught during the intervention and that created some confusion among students. As well, the lack of a significant increase in students’ positive attitudes towards CT even though students stated that they enjoyed using BBPE and had success in creating games may still be connected to preexisting ideas about CT and the careers attached to it.

**Student Attitudes Towards CS**

For students to have a positive attitude towards coding and CS, students need to have fun and enjoy it. Lewis et al.’s (2010) study revealed that student attitudes and beliefs are related to performance, so when students are successful in something, they are more likely to have a positive attitude towards it. Gunbatar et. al. (2018) study showed that programming via a BBPE may close the gap between attitudes and perceptions towards programming. Along these lines, Bastug & Kircaburun’s (2017) study showed that Code.org’s block-based programming had a positive effect on students’ attitudes toward CS. Additionally, Kelelioglu’s (2015) study also investigated the effects of teaching with Code.org and stated that students developed positive attitudes towards coding and CS. However, Lambić et al.’s (2021) study found no significant difference in attitudes after an intervention using Code.org’s curriculum. Similarly, in Taub et al.’s (2012) study reported no change in students’ attitudes in the desired direction using an unplugged CT program. This study aligns with the later studies results, there was no statistically significant increase in students’ attitudes towards CS from the pre- and post-survey. This can most likely be that not all students felt that programming was easy, so their attitudes did not change or that they did not feel successful in creating their games.
Implications

This research has implications for me, as well as researchers. There are two types of implications to consider: (a) personal implications and (b) implications for future research.

Personal Implications

As a result of this study, I have learned some personal lessons that I will use as a CS teacher in practice. Those include (a) using differentiation with lessons and (b) the power of collaboration.

Using differentiation with lessons.

Providing for differentiated assistance to all students is based in the individuals needs or difficulties with the content. Those who need more time or assistance can be provided with additional examples, as well as encourage students that work faster to stay motivated by providing advanced or extended assignments (Fonseca et al., 2018). The Code.org curriculum lessons are broken up into steps. These steps include skill building, practice, assessments, and challenges. In order to help students, complete the lessons, considering their ability might affect the outcomes of what students learn and their attitudes. It can become tedious for some to complete everything. Others can be bored because the lesson is too easy. To alleviate this, using differentiation to adjust the number of steps required to be completed would decrease time needed to complete and boredom in the classroom. Slower students should complete the skill building and assessments steps only and those who are flying through the lessons would be required to complete all steps.
The power of collaboration.

Pair programming is a great idea but, but it becomes problematic if some students take a passive stance to watch their peers complete all the work. I encourage collaboration instead. Project-based activities and interventions were determined to be “enjoy-able and effective” if they implemented collaboration with certain programming tasks (Scherer et. al., 2020). This means that everyone must complete the lessons, but they can ask a neighbor first and then the teacher if they get stuck or have an error. This helps the students learn from each other and collaboration is an important skill to learn. “Studies focusing on metacognition instruction were more effective if they engaged students in collaboration rather than having students work on programming tasks individually” (Scherer et. al., 2020, p. 14). When students are solving problems or programming, collaboration is considered essential skill for the modern workforce.

Implications for Future Research

The implications for future research are: (a) CT definition needs to be standardized and (b) develop competencies for CT and a standardized assessment.

CT definition needs to be standardized.

The confusion plaguing CT and the varied perspectives and ever-changing definitions in the research needs to be set. “There are too many definitions of CT across the many research studies that have been done over the years” (Grover & Pea, 2013, p. 38). One example of this is Angeli & Giannakos (2020) definition and explanation:

“CT is a term applied to describe the increasing attention on students’ knowledge development about designing computational solutions to problems, algorithmic thinking, and coding. It focuses on skills children develop from practicing programming and algorithms and enables the
development of qualities such as abstract thinking, problem solving, pattern recognition, and logical reasoning” (Angeli & Giannakos, 2020, p.1).

In order to set the standard, either a consensus needs to be reached within the CS community or an empirical research study on any previous research that is focused on CT should be conducted to formalize a definition for CT. By not having a widely adopted definition for CT there can be no rubric or scale to measure if students are developing this skill fully.

**Develop Competencies for CT and a standardized assessment.**

In order to adopt CT, further research needs to be done to address the issues; defining CT and the competencies for each school grade level, and which pedagogical strategies will work best in the instruction of CT (Angeli & Giannakos, 2020). Along the lines of formalizing the definition of CT, competencies need to be developed specific to CT and align with K-12 abilities. This could alleviate students who are not old enough working on CT skills, which would minimize frustration and prevent negative attitudes towards CS. Further, there are many different CT assessments found in the research literature as well. There should be a standard assessment created and used for all CT evaluations. This can also be based on grade levels to be able to assess students CT skills when they should have acquired them. And lastly, the national investment with STEM in K–12 education has had billions of dollars in funding, although research specifically in computing education is underfunded (Grover & Pea, 2013).

**Limitations**

One of the limitations of this study was the sample size. It would have been more beneficial if there were more participants or more classes in this study to have a more
robust quantitative data collection and analysis. Another limitation is that the intervention was quite long. If the instructional unit was five to six weeks, perhaps student attitudes may have been more positive towards CS. There were 12 days during the intervention where the lesson plan was not followed because of many different reasons, those being: field trips, state testing, over half of the class being absent, students needing more instruction on a lesson, or extra days to work on projects. A third limitation was in participant interviews. Participants were not as descriptive in their answers. More questions or different prompts should have been asked to help participants externalize more clearly their attitudes towards CT and CS. Fourth, only the researcher analyzed the artifacts. In the future asking another CS teacher to score them would increase the validity of the data. Finally, students did not submit attitudinal surveys during the intervention. This could have added more time points to the data collection, which would have allowed me to connect attitudes to specific topics or tasks and analyze changes in attitudes over time.

Conclusion

The need for positive learning opportunities that develop students’ CT skills and have an impact on their attitudes is still at the forefront of CS education. Prior to this study, many of the students had limited access to CS and CT. However, these students developed three of the four subskills of CT during the intervention. Algorithmic thinking, pattern recognition and debugging all had a significant increase due to the Code.org curriculum that utilizes BBPE. Abstraction had some improvement, but it was not significant. This change is due to the hands-on programming and increased use of collaboration.
Student attitudes towards CT and CS did not improve as much as we would have liked. The intervention gave them access to a programming environment where they created interactive games. And while many of the students stated that they enjoyed making their game, their attitudes did not improve to a significant level. Student attitudes are tied to how successful they feel and if they perceive the classwork as easy or fun to do. Challenging students is an important aspect of education so finding a balance is imperative.
REFERENCES


https://doi.org/10.1080/08993408.2022.2145549


https://doi.org/10.15388/infedu.2019.06

https://doi.org/10.1080/15391523.2018.1553024


https://doi.org/10.1007/s11251-018-9453-5


Psychology of Mathematics Education Tucson, AZ: The University of Arizona., 1588–1596.


Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing Loops Misconceptions in Block-Based and Text-Based Programming Languages at the K-12 Level. Education and Information Technologies, 23(4), 1483–1500. https://doi.org/10.1007/s10639-017-9673-3


What Does Sprite Mean? within a larger image environment.


https://doi.org/10.1145/1999747.1999811


https://pearsonaccelerated.com/blog/stem


https://doi.org/10.1007/s10639-018-9801-8


https://doi.org/10.1016/j.compedu.2019.103633
APPENDIX A

CS ATTITUDBINAL SURVEY

1. Gender (choose 1)
   a. Male
   b. Female
   c. Choose not to answer

2. Age
   a. 11
   b. 12
   c. 13
   d. 14
   e. 15

3. Which of the following best represents your racial/ethnic identity?
   a. American Indian or Alaskan Native
   b. Asian
   c. Black or African American
   d. Hispanic or LatinX
   e. Native Hawaiian or Pacific Islander
   f. White
   g. Two or more Races

4. English as a second language
   a. Yes
b. No

5. Have you completed a Computer Science course before this school year?
   a. Yes
   b. No

6. If the previous question is Yes, when and where did you complete the course?
   (Short Answer)

7. I have the ability to learn Computer Science
   (Mark only one oval.
   [Diagram with options: 1 Strongly Disagree, 2, 3, 4, 5 Strongly Agree])

8. I think I will be better at Computer Science than most other students at my school.
   (Mark only one oval.
   [Diagram with options: 1 Strongly Disagree, 2, 3, 4, 5 Strongly Agree])

9. I am willing to learn challenging things. (Korkmaz et al., 2017)
   (Mark only one oval.
   [Diagram with options: 1 Strongly Disagree, 2, 3, 4, 5 Strongly Agree])

10. I believe that I can solve most problems I face if I have sufficient time and if I show effort. (Korkmaz et al., 2017)
11. I think I will be very good at Computer Science.

12. I think that I have a special interest in the computational process. (Korkmaz et al., 2017)

13. I think that I learn better with instructions that include symbols and examples (Korkmaz et al., 2017)

14. It is fun to try and solve complex computer problems. (Korkmaz et al., 2017)

15. I believe that I can catch the relationship between pictures and computer programs. (Korkmaz et al., 2017)
16. I believe that I can solve problems when I encounter new situations in Computer Science (Korkmaz et al., 2017)

17. I think I will be able to figure out how to solve the most difficult issues in my computer science class if I try. (Korkmaz et al., 2017)

18. I like computer science

19. I think I could become a computer scientist one day
20. I will continue to take computer science classes in the future.

*Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strongly Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B

CS PRE-POST-TEST

21. Have you ever participated in any computer science/programming programs before this class? Select where you previously participated.
   a. No
   b. After School Program
   c. Summer Camp
   d. Online
   e. Other

22. What does an algorithm have to do with processing? Code.org

   ☐ algorithms are a series of steps
   ☐ algorithms are steps that would be converted into a program that a computer could use to process information, changing it from output to input
   ☐ algorithms have nothing to do with processing
   ☐ algorithms are steps that would be converted into a program that a computer could use to process information, changing it from input to output

23. Finding and fixing problems in your algorithm. Code.org
   a. Property
   b. Debugging
   c. Parameter
   d. Sprite

24. Attributes that describe an objects characteristic. Code.org
   a. Sprite
25. A placeholder for a piece of information that can change. Code.org
   a. Variable
   b. Property
   c. Parameter
   d. Code

26. A series of images that creates the illusion motion by being shown rapidly one after the other. Code.org
   a. If-Statement
   b. Animation
   c. Boolean
   d. Frame Rate

27. Which of the following is the correct code for initializing a variable?
   a. var = 0
   b. var score = 0
   c. score = 0
   d. variable(score) = 0

28. Why do programmers use functions? (Choose 2) Code.org
   A. They keep the code organized and easy to understand.
   B. They are a good way to store user input to use later in the program.
   C. They make the program run faster.
   D. They are necessary for any program to run.
   E. They let you use the same code in different places in the program without having to rewrite it.
29. What is the apples X position at the end of this program?

```
apple.x = 20;
banana.x = 60;
apple.x = banana.x + 30 ;
banana.x = 80;
```

a. 20

b. 90

c. 110

d. There will be 2 apples in 2 different positions

e. None, the program has an error

30. Which of the following is not best to decide before beginning to write code for a program? Code.org

A. The general purpose and design of the program.
B. What kinds of variables and functions you will need to make it work.
C. How the user will interact with the program.
D. How many lines of code you want to write.
E. What types of media, such as pictures or sounds, you will need.

31. A team is making a platform jumper game. The background and the platforms are working, but the main character won’t appear. What might be the problem? Code.org

1. They forgot to call `drawSprites` in the draw loop.
2. The character sprite is drawn behind the background sprite.
3. The character sprite is drawn off the screen.

a. 3 only

b. 1 and 2

c. 1 and 3
d. 2 and 3

e. All of them

32. Where is the conditional statement in the code below?

```
var player = createSprite(200, 200);
player.setAnimation("alien");
var bubbleY = 400;
var bubbleX = 99;
// comment
stroke("white");

function draw() {
  background("darkblue");

  if (keyWentDown("space")) {
    // comment
    player.velocityY = 5;
  }

  // comment
  player.velocityY = player.velocityY + 0.2;

  drawSprites();

  // comment
  bubbleY = bubbleY - 5;

  // comment
  bubbleX = bubbleX + randomNumber(-3, 3);

  ellipse(bubbleX, bubbleY, 10);
}
```

a. Line 15
b. Line 10
c. Line 20
d. Line 16

33. Dot notation is handy to access variable properties of sprites. Ex sprite.scale

Code.org

a. True
b. False

34. A simplified representation of something more complex, ________ allows you to hide details to help you manage complexity, focus on the relevant concepts and reason about the problems at a higher level. Code.org
a. Abstraction
b. Function
c. Boolean
d. Parameters

35. Choose the correct code to rotate the blue gear.

![Image of rotating gears]

a. blueGear.rotation+=1
b. blueGear=blueGear+1
c. blueGear.rotation=blueGear.rotation+1
d. rotation=blueGear+=1

36. How many times will the word “here” be said when the code is run? (Rachmatullah et al., 2020)

```javascript
var x = 1;
function draw() {
  if (x <= 10) {
    text("here", x+50, 15);
    console.log("here");
    x = x-1;
  }
}
```

a. 0
b. 1
c. 5
d. 10

37. Which code moves the bike up and to the right?

a. bike.y=bike.x +2 bike.x=bike.y-2
b. bike.x=bike.x+2 bike.y=bike.y+2
c. bike.y=bike.rotation+2
d. bike.x=bike.x+2 bike.y=bike.y-2

38. What needs to be changed in this code for the value of x to equal 15?

(Rachmatullah et al., 2020)

a. Changes line 3 to read x=x+x
b. Change line 2 to read var y=y+x
c. Change line 2 to read var y = 5

d. nothing

39. What is the bug in this program?

40. Which lines uses the counter pattern? Choose all that apply.
41. Look at the following code, what comment should be placed at line 11?

```javascript
var player = createSprite(200, 200); //
player.setAnimation("alien");
var bubble = createSprite(100, 100);
var bubbleX = 50;
// comment
sprite("white");
function draw() {
background("darkblue");
if (keyDown("space")) {
// comment
player.velocityY = 0;
}
// comment
player.velocityY = player.velocityY + 2;
// comment
drawSprites();
// comment
bubbleX = bubbleX - 1;
// comment
bubbles = bubbles - 1;
// comment
bubbleX = bubbleX + randomNumber(0, 5);
ellipse(bubbles, bubbleX, 10, 10);

```

a. Give the bubble an outline
b. Shake back and forth
c. Float up
d. Jump up
e. Fall down

42. (Rachmatullah et al., 2020)
A robot is going to deliver a package to an owner. Below are the steps the robot needs to take to deliver the package.

1. Locate the owner of the package
2. Follow the fastest path from the robot location to the owner’s location
3. Calculate the fastest path from the robot location to the owner’s location
4. Drop the package

However, there might be small mistake in the order of the steps. Can you find the mistake?

a. The order is correct
b. Step 2 should be after step 3
c. Step 2 should be after step 4
d. Step 1 should be after steps 2 and 3

43. What line of code is asking a Boolean expression?

```javascript
function draw() {
  // Draw Background
  background(\"white\");
  // Update
  raceCar.x = raceCar.x - 2;
  if (raceCar.x < finishLine.x) {
    text("WINNER!!", 200, 200);
  }
  // Draw Animations
drawSprites();
}
```

a. Line 12
b. Line 13
c. Line 14

d. Line 17

44. A Boolean expression will only execute the code if the statement is…

a. True

b. False

45. Which line of code is executing a loop?

a. Line 2

b. Line 8

c. Line 15
46. What is the bug in this program?

```javascript
var blender = createSprite(200, 200);
blender.setAnimation("blender");
background("lightyellow");
drawSprites();
function draw() {
  blender.x = randomNumber(195, 205);
  blender.y = randomNumber(195, 205);
}
```

47. What is the counter pattern doing in this code? (short answer)

```javascript
function collectItems() {
  if(player.isTouched(star1)) {
    star1.y = 0;
    star1.x = randomNumber(0, 400);
    score = score + 1;
  }

  if(player.isTouched(star2)) {
    star2.y = 0;
    star2.x = randomNumber(0, 400);
    score = score + 1;
  }
}
```

48. What is the function loopItems doing in this program? (short answer)
49. What is the x position of star1? (short answer)

```javascript
    function loopItems() {
        if (star1.y > 400) {
            star1.y = 0;
            star1.x = randomNumber(0, 400);
        }
        if (star2.y > 400) {
            star2.y = 0;
            star2.x = randomNumber(0, 400);
        }
    }

    function playerFall() {
        player.velocityY = player.velocityY + 0.75;
        if (player.y > 400) {
            player.y = 0;
        }
    }
```

50. There are 2 background functions, explain how and when each will be displayed.

    (short answer)
function draw() {
    // draw the background
    // update the sprites
    if (score < 10) {
        background1();
    } else {
        background2();
    }
    if (score < 10) {
        showScore();
    }
    loopPlatforms();
    loopSprites();
    playerFall();
    controlPlayer();
    playerLands();
    collectItems();
    drawSprites();
}

// Functions
function background1() {
    background("darkBlue");
## APPENDIX C

### INTERVIEW PROTOCOL

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Interview Questions</th>
</tr>
</thead>
</table>
| RQ1: How and to what degree did BBPE affect CT skill levels (algorithms, abstraction, debugging, and problem solving) for middle school CS participants? | 1a. Loops are used in many areas of Game Design; will you explain how loops work?  
 b. Tell me about a loop you used in your code.  
 c. What is a function of a loop?  
 2a. Tell me how a conditional works?  
 b. Where have you used them during this class?  
 c. Did they work?  
 d. What happened when one worked?  
 e. What happened when one didn’t work?  
 3a. Name ways you can include variables in Game Design?  
 b. How do they work?  
 c. How did it help complete your game?  
 4a. The Counter Pattern is used in many tasks in Game Design, tell me how it works?  
 b. Where did you use it most?  
 c. Was it an easy concept to learn?  
 d. Why is that?  
 5a. When you encountered a problem in your code without an obvious answer what steps did you take to solve it?  
 b. Did you collaborate with other participants to solve it?  
 c. Was that helpful?  
 d. Did you read others code to better understand how to solve it?  
 e. Was that helpful?  
 6a. Looking at your artifact (program your game) tell me how this program worked.  
 b. Why did you choose (select a line of code) this?  
 c. Explain the process of how this line works? |
<p>| RQ2: How and what degree were the differences in middle school | 1a. Which aspect of programming did you enjoy most in Computer Science class? |</p>
<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
</table>
| participant attitudes towards CT and CS after a unit in block-based programming? | b. Why did you enjoy that aspect so much?  
c. Would you recommend this class to other participants? |
| 2a. What did you like least about CS class?                             | b. Why did you like that so little? |
| 3a. What aspects of the block-based programming did you like most?       | b. Please explain.  
c. Least?  
d. Please explain. |
| 4a. Based on your experience in CS class, do you think CS is interesting? | b. Do you see yourself as a Computer Scientist?  
c. Why do you feel that way? |
| 5a. Did you enjoy dragging blocks in Java Script?                       | b. Why is that?  
c. Please explain. |
## APPENDIX D

### GAME CREATION PROJECT RUBRIC

<table>
<thead>
<tr>
<th>Key Concept</th>
<th>Extensive Evidence</th>
<th>Convincing Evidence</th>
<th>Limited Evidence</th>
<th>No Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Development (Algorithmic Thinking)</td>
<td>Your project guide is complete and reflects the project as submitted.</td>
<td>Your project guide is mostly complete and is generally reflective of the submitted project.</td>
<td>Your project guide is filled out but is not complete or does not reflect the submitted project.</td>
<td>Your project guide is incomplete or missing.</td>
</tr>
<tr>
<td>Program Readability (Debugging)</td>
<td>Your program code effectively uses whitespace, good naming conventions, indentation, and comments to make the code easily readable.</td>
<td>Your program code makes use of whitespace, indentation, and comments.</td>
<td>Your program code has few comments and does not consistently use formatting such as whitespace and indentation.</td>
<td>Your program code does not contain comments and is difficult to read.</td>
</tr>
<tr>
<td>Use of Functions (Abstraction)</td>
<td>At least three functions are used to organize your code into logical segments. At least one of these functions is called multiple times in your program.</td>
<td>At least two functions are used in your program to organize your code into logical segments.</td>
<td>At least one function is used in your program.</td>
<td>There are no functions in your program.</td>
</tr>
<tr>
<td>Backgrounds and Variables</td>
<td>Your game has at least three backgrounds that are displayed</td>
<td>Your game has multiple backgrounds that are displayed</td>
<td>Your game has multiple backgrounds.</td>
<td>Your game does not have</td>
</tr>
</tbody>
</table>

138
<table>
<thead>
<tr>
<th>(Algorithmic Thinking)</th>
<th>during run time, and at least one change is triggered automatically through a variable (e.g. score).</th>
<th>during run time (e.g. main background and “end game” screen)</th>
<th>multiple backgrounds.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactions and Controls (Algorithmic Thinking, Pattern recognition, Debugging)</td>
<td>Your game includes multiple different interactions between sprites, and it responds to multiple types of user input (e.g. different arrow keys).</td>
<td>Your game includes at least one type of sprite interaction, and it responds to user input.</td>
<td>Your game includes no conditionals.</td>
</tr>
<tr>
<td>Position and Movement (Pattern Recognition, Abstraction)</td>
<td>Complex movement such as acceleration, moving in a curve, or jumping is included in multiple places in your program.</td>
<td>Your program includes some complex movement, such as jumping, acceleration, or moving in a curve.</td>
<td>Your program includes simple independent movement, such as a straight line or rotation.</td>
</tr>
<tr>
<td>Variables (Algorithmic Thinking, Abstraction)</td>
<td>Your game includes multiple variables that are updated during the game and affect how the game is played.</td>
<td>Your game includes at least one variable that is updated during the game and affects the way the game is played.</td>
<td>There is at least one variable used in your program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>There are no variables, or they are not updated.</td>
</tr>
</tbody>
</table>
APPENDIX E

STUDENT ASSENT FORM
UNIVERSITY OF SOUTH CAROLINA
ASSENT FORM FOR RESEARCH PARTICIPANTS
A STUDY OF COMPUTATIONAL THINKING SKILLS AND ATTITUDES WITH MIDDLE SCHOOL STUDENTS
COLLEGE OF EDUCATION, CURRICULUM, AND INSTRUCTION

Researcher: LORIEN CAFARELLA

We are doing a research study. A research study is a special way to find out about something. The purpose of this action research study will be to evaluate how a block-based programming environment affects middle school CS students’ Computation Thinking skills, and their attitudes towards CT and CS at CE Williams Middle School.

You can be in this study if you want to. If you want to be in this study, you will be asked to participate in completing a pre and posttest, interviews and to collect your game you created in the Game Design Unit.

If you decide to be in this study, some good things might happen to you. There is no direct benefit to you for being in the study. But we don’t know for sure that these things will happen. We might also find out things that will help other students’ someday.

When I am done with the study, I will write a report about what I found out. I won’t use your name in the report. All the information you provide will be kept private. No one except the me will know that you are in the study unless you and your parents decide to tell them.

Even if your parent/guardian agrees to your participation in this study, it is still your decision whether or not to be in the study. You do not have to be in this study if you don’t want to. You can say “no” and nothing bad will happen. If you say “yes” now, but you want to stop later, that’s okay too. If something about the study bothers you, you can stop being in the study. All you have to do is tell the researcher you want to stop. If there is anything you don’t like about being in the study, you should tell us and if we can, we will try to change it for you.

If you have any questions about the study, you can ask the me. I will try to explain everything that is being done and why. Please ask me about anything you want to know.

If you want to be in this study, please sign and print your name.
I, ______________________________, want to be in this research study.

(Write your name here)

____________________________________   __________________
Sign your name here                     (Date)

____________________________________   __________________
Researcher signature                    (Date)
APPENDIX F
PARENT CONSENT FORM

UNIVERSITY OF SOUTH CAROLINA
PARENT PERMISSION FORM
A STUDY OF COMPUTATIONAL THINKING SKILLS AND ATTITUDES WITH MIDDLE SCHOOL STUDENTS
LORIEN CAFARELLA
COLLEGE OF EDUCATION, CURRICULUM, AND INSTRUCTION

Your child has been invited to participate in this research study. Before you agree to allow your child to participate, it is important that you read and understand the following information. Participation is completely voluntary. Please ask questions about anything you do not understand before deciding whether or not to give permission for your child to participate.

PURPOSE: The purpose of this action research study will be to evaluate how a block-based programming environment affects middle school CS students’ Computational Thinking skills, and their attitudes towards CT and CS at CE Williams Middle School. Your child will be one of approximately 25 participants in this research study.

PROCEDURES: There will be six different data sources collected to analyze how well students have developed Computational Thinking (CT) skills and if student’s attitudes towards Computer Science (CS) have changed. To understand the depth of understanding these data sources will include: (a) pre-test, (b) post-test, (c) survey, (d) student interviews, and (e) student artifacts. Your child will be audio taped during the interview portion of the study to ensure accuracy. The tapes will later be transcribed and destroyed after 2 years beyond the completion of the study. For confidentiality purposes, your child’s name will not be recorded.

DURATION: Your child’s participation will consist of 9 weeks, during each 9th period class every day for 45 minutes.

RISKS: The risks associated with participation in this study are no more than the child would encounter in school every day.

BENEFITS: The benefits associated with participation in this study include an understanding of Middle School students’ abilities in Computer Science and if the content used will increase student skills and attitudes. Students may acquire the ability to program and the need to continue in the field.
CONFIDENTIALITY: All information your child reveals in this study will be kept confidential. All your child’s data will be assigned an arbitrary code number rather than using your child’s name or other information that could identify your child as an individual. When the results of the study are published, your child will not be identified by name. The data will be destroyed by shredding paper documents and deleting electronic files 2 years after the completion of the study. Your child’s research records may be inspected by the University of South Carolina Institutional Review Board or its designees, and (as allowable by law) state and federal agencies.

VOLUNTARY NATURE OF PARTICIPATION: Your child’s participation in this study is completely voluntary and your child may withdraw from the study and stop participating at any time without penalty or loss of benefits to which your child is otherwise entitled. The procedure to withdraw, a parent letter that includes the date of withdrawal and be signed by the parent.

CONTACT INFORMATION: If you have any questions about this research project, you can contact loriencafarella@charleston.k12.sc.us or 843-367-1529. If you have questions or concerns about your child’s rights as a research participant, you can contact University of South Carolina College of Education at

I HAVE HAD THE OPPORTUNITY TO READ THIS PARENT PERMISSION FORM, ASK QUESTIONS ABOUT THE RESEARCH PROJECT AND AM PREPARED TO GIVE MY PERMISSION FOR MY CHILD TO PARTICIPATE IN THIS PROJECT.

____________________________________________
Parent’s Signature(s) Date

____________________________________________
Parent’s Name(s)

____________________________________________
Researcher’s Signature Date
Appendix G

Student 164894 Artifact
bubble2.velocityY = -3;

var bubble3 = createSprite(randomNumber(10, 390), 390, 2);
bubble3.animation("bubble");
bubble3.scale = 0.1;
bubble3.velocityY = -3;

function draw() {
    // draw background
    fill("blue");
    rect(0, 0, 400, 600);
    // update sprites
    playerMovement();
    Interactions();
    Looping();
    GameEnding();
    fill("black");
    textSize(20);
    text("Score: " + score, 10, 10, 100, 100);
    drawSprites();
}

function Interactions() {
    if (fish.isTouching(shark)) {
        score = score + 2;
        shark.x = randomNumber(0, 50);
        shark.y = randomNumber(0, 400);
    }
    if (fish.isTouching(turtle)) {
        score = score + 1;
        turtle.x = randomNumber(0, 50);
        turtle.y = randomNumber(0, 400);
    }
    if (fish.isTouching(ffish)) {
        score = score + 1;
    }
}
Pfish.x = randomNumber(0, 50);
Pfish.y = randomNumber(0, 400);
}
+
if (fish.isTouched(nemo)) {
    score = score + 1;
nemo.x = randomNumber(0, 50);
nemo.y = randomNumber(0, 400);
}
+
if (fish.isTouched(fishb)) {
    score = score + 1;
    fishb.x = randomNumber(0, 50);
    fishb.y = randomNumber(0, 400);
}
+
if (score > 10) {
    BackGround2(); =
}
+
// Create your functions here
function playerMovement() {
    if (keyDown(\"up\")) {
        fish.setAnimation(\"fishmovingup\");
fish.y = fish.y - 3;
    }
+
    if (keyDown(\"left\")) {
        fish.setAnimation(\"fish\");
fish.x = fish.x - 3;
    }
+
    if (keyDown(\"right\")) {
        fish.setAnimation(\"fishmovingright\");
fish.x = fish.x + 3;
    }
+
    if (keyDown(\"down\")) {
        fish.setAnimation(\"fishmovingdown\");
fish.y = fish.y + 3;
    }
}
function Looping() {
  if (shark.x > 425) {
    shark.y = randomNumber(0, 400);
    shark.x = -10;
  }
  if (Pfish.x > 425) {
    PFish.y = randomNumber(0, 400);
    PFish.x = -10;
  }
  if (Cturtle.x > 425) {
    curtle.y = randomNumber(0, 400);
    curtle.x = -10;
  }
  if (fish3.x > 425) {
    fishb.y = randomNumber(0, 400);
    fishb.x = -10;
  }
  if (bubble1.y < -20) {
    bubble1.y = 380;
    bubble1.x = randomNumber(10, 390);
  }
  if (bubble2.y < -20) {
    bubble2.y = 390;
    bubble2.x = randomNumber(10, 390);
  }
  if (bubble3.y < -20) {
    bubble3.y = 400;
    bubble3.x = randomNumber(10, 390);
  }
}

function Background2() {
  background("*darkblue");
}
```javascript
function GameEnding() {
    if (score < 0) {
        background("black");
        textSize(50);
        fill("green");
        text("Game Over!", 50, 200);
        bubble1.visible = false;
        bubble2.visible = false;
        bubble3.visible = false;
        fish.visible = false;
        fishb.visible = false;
        Pfish.visible = false;
        shark.visible = false;
        turtle.visible = false;
        nemo.visible = false;
    }
}
```
Appendix H

Student 192880 Artifact
function draw() {
    // draw background
    background("#black");

    // update sprites
    shipMovement(); //
    loopItems();    //
    ShipLazer();    //
    lazerRock();    //
    shipScore();    //
    rockShip();     //
    shipHealth();   //
    healthShip();   //

    drawSprites();
}

// Create your functions here

function shipMovement() {
    if (keyDown("right")) {
        ship.x = ship.x + 4;
    }
    else {
        ship.x = ship.x - 4;
    }
}

function loopItems() {
    if (rock.y > 415) {
        rock.x = randomNumber(40, 350);
        rock.y = randomNumber(-30, -60);
    }
    if (rock2.y > 415) {
        rock2.x = randomNumber(40, 350);
        rock2.y = randomNumber(-30, -60);
    }
    if (rock3.y > 415) {
        rock3.x = randomNumber(40, 350);
        rock3.y = randomNumber(-30, -60);
    }
}
function ShipLazer() {
    if (keydown(\"up\") ) {
        lazer.visible = true;
        lazer.velocityY = -4;
        lazer.x = ship.x;
        lazer.y = 350;
    }
}

function lazerRock() {
    if (lazer.isToucHing(rock)) {
        score = score + 100;
        rock.x = randomNumber(40, 350);
        rock.y = randomNumber(-30, -60);
        lazer.y = 450;
    }
    if (lazer.isToucHing(rock2)) {
        rock2.x = randomNumber(40, 350);
        rock2.y = randomNumber(-30, -60);
        score = score + 100;
        lazer.y = 450;
    }
    if (lazer.isToucHing(rock3)) {
        rock3.x = randomNumber(40, 350);
        rock3.y = randomNumber(-30, -60);
        score = score + 100;
        lazer.y = 450;
    }
}

function rockShip() {
    if (rock.isToucHing(ship)) {
        health = health - 10;
    }
    if (rock.isToucHing(ship)) {
        rock.x = randomNumber(40, 350);
        rock.y = randomNumber(-30, -60);
    }
}
if (rock2.isTouching(ship)) {
    health = health - 10;
}
if (rock2.isTouching(ship)) {
    rock2.x = randomNumber(40, 350);
    rock2.y = randomNumber(-30, -60);
}
if (rock3.isTouching(ship)) {
    health = health - 10;
}
if (rock3.isTouching(ship)) {
    rock3.x = randomNumber(40, 350);
    rock3.y = randomNumber(-30, -60);
}
}

function shipScore() {
    fill(▼"white");
    textSize(20);
    text("score", 0, 14);
    text(score, 59, 15);
}

function shipHealth() {
    fill(▼"white");
    textSize(20);
    text("health", 290, 14);
    text(health, 350, 15);
}

function healthShip() {
    if (health < 0) {
        background(▼"black");
        fill(▼"green");
        textSize(50);
        text("Game Over!", 40, 200);
        rock.visible = false;
        rock2.visible = false;
        rock3.visible = false;
        ship.visible = false;
    }
}
### Table of Round One Coding Cycle List

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Number of Snippets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Scientist</td>
<td>7</td>
</tr>
<tr>
<td>Fun</td>
<td>5</td>
</tr>
<tr>
<td>Recommend</td>
<td>3</td>
</tr>
<tr>
<td>Disliked About Code</td>
<td>5</td>
</tr>
<tr>
<td>Block Based Coding Liked Least</td>
<td>1</td>
</tr>
<tr>
<td>Block Based Coding Enjoyed Most</td>
<td>11</td>
</tr>
<tr>
<td>Like About Code</td>
<td>11</td>
</tr>
<tr>
<td>Coding/Programs</td>
<td>11</td>
</tr>
<tr>
<td>Loops</td>
<td>9</td>
</tr>
<tr>
<td>Sprite</td>
<td>18</td>
</tr>
<tr>
<td>Position</td>
<td>15</td>
</tr>
<tr>
<td>If Then</td>
<td>8</td>
</tr>
<tr>
<td>Conditional</td>
<td>5</td>
</tr>
<tr>
<td>Variable</td>
<td>8</td>
</tr>
<tr>
<td>Score</td>
<td>20</td>
</tr>
<tr>
<td>Counter Pattern</td>
<td>6</td>
</tr>
<tr>
<td>Collaboration</td>
<td>5</td>
</tr>
<tr>
<td>Solution</td>
<td>3</td>
</tr>
<tr>
<td>Debugging</td>
<td>7</td>
</tr>
<tr>
<td>Game</td>
<td>13</td>
</tr>
</tbody>
</table>