

THE EFFECTS OF PROJECT-BASED GAME DEVELOPMENT ON STUDENT
LEARNING AND ATTITUDES: ACTION RESEARCH IN AN 8TH GRADE
INTRODUCTORY COMPUTER SCIENCE COURSE

by

Theodore G. Jenks

Bachelor of Science

The University of North Carolina at Chapel Hill, 2000

Master of Science

Southern Methodist University, 2009

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

2022

Accepted by:

Ismahan Arslan-Ari, Major Professor

Michael Grant, Committee Member

Yingxiao Qian, Committee Member

Anna C. Clifford, Committee Member

Cheryl L. Addy, Interim Vice Provost and Dean of the Graduate School

© Copyright by Theodore G. Jenks, 2022
All Rights Reserved

DEDICATION

This dissertation is dedicated to everyone who supported me through this process. Thank you to the students who participated in this study. Without your willingness to participate, this study would not have been possible. Without your perseverance and hard work, this work would not have been successful. Thank you to my administration, which allowed me to grow a computer science program and perform research at their institution.

Thank you to my mother, who provided babysitting and proofreading services. You started law school when I was two, and I can now empathize with that experience.

Most of all, this is dedicated to Claire, Gavin, and Valerie. You allowed me the space to work and were remarkably understanding for a long time. We are near the end – love you muches!

ACKNOWLEDGMENTS

I acknowledge several people who assisted with this dissertation and my journey through this program. First, my major professor, Dr. Ismahan Arslan-Ari, patiently guided me through the process of transforming raw ideas and data into a written form that others could understand. Without her support, my qualitative and quantitative analyses would have been an incoherent mess. My other committee members, Dr. Michael Grant, Dr. Yingxiao Qian, and Dr. Anna C. Clifford, provided welcome critiques to improve this work. I may not have another opportunity for a group of highly credentialed educators to review my work in such depth. I suspect that, as with many significant experiences, I may not fully appreciate this process until much later. I am very grateful to the committee for their time and efforts.

In addition to my major professor and committee members, other professors in the Learning Design and Technologies concentration were incredibly generous with their time. They demonstrated an unwavering commitment to students' success. Those professors from which I had the pleasure of learning and who had a special impact on me were: Dr. Fatih Ari, Dr. Allison Moore, Dr. William Morris, Dr. Stephen Rodriguez, and Dr. Lucas Vasconcelos. Dr. Angie Starrett provided help with statistical analysis long after I had left her statistics class.

Finally, thank you to cohort RNR for their support and camaraderie during this process.

ABSTRACT

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha. The following research questions were explored: (1) How does the game development project impact participants' ability to analyze and develop algorithms? (2) What is the effect of the game development project on participants' attitudes toward computer science? and (3) What is the relationship between participants' attitudes toward computer science and their performance?

There were 28 participants composed of students in a science, technology, engineering, and mathematics magnet program. A convergent parallel mixed-methods approach was used to answer the research questions. A content knowledge assessment pretest and posttest were administered to measure performance before and after the intervention. Content knowledge assessment scores after the intervention were significantly larger than the content knowledge assessment scores before the intervention. A survey measuring attitudes toward computer science was administered to participants before and after the intervention. The survey consisted of five subscales: (a) self-concept, (b) learning at school, (c) learning outside of school, (d) future participation, and (e) importance. For all five subscales, the subscale measure after the intervention was significantly larger than the subscale measure before the intervention. The linear correlation between participants' attitudes toward computer science and their

performance was measured at the end of the intervention. Findings suggested that as participants' scores on the post-survey for attitudes toward computer science increased, so did participants' scores on the post-content knowledge assessment.

Qualitative data was collected in the form of field notes from classroom observations and participant interviews. Inductive and deductive analysis was performed on the qualitative data to help answer the research questions. Findings showed that (a) participants' performance and attitudes improved after the intervention, (b) participants experienced barriers to success, and (c) attitudes and performance were related and appeared to influence each other. Implications and limitations of the research were discussed.

TABLE OF CONTENTS

Dedication	iii
Acknowledgments.....	iv
Abstract	v
List of Tables	ix
List of Figures	xi
List of Abbreviation.....	xiii
Chapter 1 : Introduction	1
Chapter 2 : Literature Review.....	15
Chapter 3 : Method	36
Chapter 4 : Analysis and Findings	82
Chapter 5 : Discussion, Implications, and Limitations	141
References	176
Appendix A: Checkpoints.....	199
Appendix B: Participant Information.....	201
Appendix C: Game Development Project Directions	204
Appendix D: Content Knowledge Assessment.....	226
Appendix E: Survey	240
Appendix F: Interview Protocol.....	242
Appendix G: IRB Approval	245
Appendix H: District Study Approval	247

Appendix I: Original and Current Pseudonyms.....	248
--	-----

LIST OF TABLES

Table 3.1 Interviewed Participant Demographics.....	40
Table 3.2 SC READY Math and ELA Performance	41
Table 3.3 Interviewed Participant Academic History	41
Table 3.4 Project-Based Design Element Implementations	51
Table 3.5 Research Question and Data Sources Alignment	56
Table 3.6 Content Knowledge Assessed by Question	57
Table 3.7 Course Standard Assessed by Content Knowledge Assessment	59
Table 3.8 Alignment of Interview Questions to Research Questions	64
Table 3.9 Research Question, Data Sources, and Data Analysis Alignment.....	65
Table 3.10 Timeline of Phases and Tasks.....	68
Table 4.1 Interpretation of Effect Size.....	83
Table 4.2 Performance Task Percent Agreement	84
Table 4.3 Correct Responses Per Item on Multiple-Choice Assessment.....	85
Table 4.4 Descriptive Statistics of the Pretest and Posttest	86
Table 4.5 Paired Samples t-Test Results for Content Knowledge Assessment Scores	87
Table 4.6 Survey Subscales	87
Table 4.7 Internal Consistency Measure of Survey (Researcher).....	88
Table 4.8 Descriptive Statistics of Pre and Post-CS Attitude Survey Subscales.....	88
Table 4.9 Survey Subscale Assumption Checks.....	89
Table 4.10 Parametric Inferential Results of Pre and Post-Survey Subscales.....	90

Table 4.11 Nonparametric Inferential Results of Pre and Post-Survey Subscales	91
Table 4.12 Interpreting the Size of a Correlation Coefficient	93
Table 4.13 Summary of Qualitative Data Sources.....	95
Table 4.14 Examples of First Cycle Coding Methods.....	104
Table 4.15 Category Progression.....	110
Table 4.16 Hierarchical Structure of Themes	114
Table 4.17 Description of Themes.....	116
Table B.1 Participant Demographics	201
Table B.2 Participant Academic History	202
Table H.1 Original and Current Pseudonyms	248

LIST OF FIGURES

Figure 3.1 Zulama Pinball Rule Set.....	46
Figure 3.2 Zulama Pinball Drop Button Create Event.....	46
Figure 3.3 Zulama Scat Score Hand Function	47
Figure 3.4 Abridged Game Design Document from Aleesha.....	73
Figure 3.5 Status Report from Julia	75
Figure 3.6 Abridged Playtest Document from Bree	76
Figure 4.1 Composite Post-Survey Scores Vs. Posttest Scores	93
Figure 4.2 Field Notes Composition Notebook	96
Figure 4.3 Handwritten Field Notes.....	97
Figure 4.4 Word Document Field Notes	98
Figure 4.5 Otter.ai Needs Cleaning Folder	99
Figure 4.6 Otter.ai Cleaned Interviews Folder.....	100
Figure 4.7 Review Interview Transcript Email.....	101
Figure 4.8 Delve Codes Export to Excel	105
Figure 4.9 Second Coding Cycle in Excel	105
Figure 4.10 Second Cycle Transition Notes One in Excel	106
Figure 4.11 Second Cycle Transition Notes Two in Excel.....	108
Figure 4.12 Grouped Codes in Excel at the End of Cycle Three.....	109
Figure 4.13 Performance Improvements in Delve	111
Figure 4.14 Barriers to Success in Delve.....	111

Figure 4.15 Positive Attitudes in Delve	112
Figure 4.16 Attitude Performance Feedback Loop in Delve	113
Figure 4.17 Compiler Error Symbol in GameMaker	134
Figure 4.18 Compiler Error Window in GameMaker.....	135
Figure 4.19 Runtime Error Window in GameMaker	135

LIST OF ABBREVIATION

AP	Advanced Placement
AP CSA.....	Advanced Placement Computer Science A
AP CSP	Advanced Placement Computer Science Principles
API	application programming interface
ASD.....	Agile software development
CP.....	college prep
CS.....	computer science
DGBL.....	digital game-based learning
GBL.....	game-based learning
GDBL.....	game development-based learning
IDE	integrated development environment
IRB	Institutional Review Board
PBL	problem-based learning
PjBL	project-based learning
STEM.....	science, technology, engineering, and mathematics

CHAPTER 1: INTRODUCTION

National Context

Software development is one of the fastest-growing occupations for 2021-2031, with a 25% projected growth rate and 500,000 jobs currently unfilled in the United States (U.S. Bureau of Labor Statistics, 2022). The projected growth rate does not include jobs in the computational sciences and engineering, where computer science (CS) is a necessary tool for conducting research or designing applications. Of all new jobs in science, technology, engineering, and mathematics (STEM), 67% are in computing (U.S. Bureau of Labor Statistics, 2022). Computer science concepts and competencies are becoming necessary for an increasing number of sciences and industries (Culic et al., 2019; Repenning et al., 2015; Zendler & Klautdt, 2012). Companies are unable to fill software development jobs due to a lack of individuals with adequate computer science skills (Southern Regional Education Board, 2016).

The U.S. Department of Education explicitly focused on computer science within its STEM initiatives. The U.S. Department of Education emphasized computer science for the following reasons: (a) CS education improves critical thinking; (b) CS education improves problem-solving abilities; and (c) CS job openings exceed the number of qualified candidates (DoED Secretary's Final Supplemental Priorities and Definitions for Discretionary Grant Programs, 2018). The shortage of qualified CS professionals has prompted United States technology companies to petition the federal government to increase the number of H1-B visas for technologically skilled workers (Repenning et al.,

2015). China and India each produced over three times more CS university graduates than the United States (Loyalka et al., 2019).

Computing will impact the lives of today's students. They may work in fields affected by computing, and nearly all students will be affected culturally by computing. Today's average student will need to understand computer science principles to function in society (Tucker et al., 2003). College should not be the first opportunity for students to study computer science; students should be introduced to computing concepts in K-12 (Barr & Stephenson, 2011).

In this study, *algorithm* will be defined as “a set of rules for how to take some input or starting state and produce a corresponding output or end state” (Wilkerson-Jerde, 2014). Algorithm analysis and development will be defined as understanding what existing algorithms do and developing algorithms to solve problems (McGregor & Sykes, 2001). In computer science, *algorithm analysis* refers to measuring the storage and time complexity of algorithms; that is not how the term will be used in this study. Computer programming, a subset of computational thinking, involves writing and analyzing algorithms. Programming is difficult for students to learn because of the many skills that must be mastered (Alturki, 2016; Cheah, 2020; Javidi & Sheybani, 2014; Végh & Stoffová, 2019). Students must identify problems, design solutions, translate solutions into a form that complies with the rules of syntax and semantics for a particular programming language, and test the solutions (João et al., 2019; Végh & Stoffová, 2019). Students often have difficulty writing and analyzing algorithms due to low ability in mathematical thinking and logical reasoning (Cetin & Andrews-Larson, 2016; João et al., 2019). Algorithmic solutions must be translated into a programming language, and

programming languages take time to master. Another impediment to student learning is low interest in CS and an unwillingness to devote sufficient time to mastering programming skills (Culic et al., 2019; João et al., 2019).

In the past, few students in the United States elected to take CS due to its perceived difficulty. Students who pursued CS were typically from affluent families that could afford expensive computing equipment and provide parental expertise (Goode & Margolis, 2011; Repenning et al., 2015). These students arrived in CS courses with some exposure to computing and a willingness to devote significant time to the subject. Until about 2011, College Board's Advanced Placement Computer Science A (AP CSA) course was the only computer science offering for high school students (CollegeBoard, 2020a). Its steep learning curve and use of Java console programs failed to reach students who did not already possess an affinity for computing (Goode & Margolis, 2011). Methods for teaching CS should be developed for students who do not receive exposure to computing at home.

Several alternatives to console programming have been implemented to make CS more accessible to students. In 2016, College Board introduced Advanced Placement Computer Science Principles (AP CSP) as a more accessible first introduction to CS than AP CSA (CollegeBoard, 2020b). In addition to programming and algorithm development, AP CSP focused on creativity, abstraction, data and information, the Internet, and global impact (CollegeBoard, 2020b). Block programming languages such as Scratch allow students to program visually and relieve some of the syntax overhead involved with text-based languages (Cucinelli et al., 2018). Arduino includes a development board and integrated development environment (IDE) that enable students

to build interactive devices (Perenc et al., 2019). Lego and Vex offer robotics kits that allow students to make and program robots.

Many children today spend a significant amount of time playing computer games, which suggests that gaming is engaging for young people (Anderson & Jiang, 2018). Leveraging the engagement produced by computer games could be a valuable learning tool. Constructivist theory states that learning is more efficient when constructed by students than when communicated by an instructor; students learn best when constructing “concrete and meaningful artifacts that can be shared with others” (An, 2016, p. 556). Project-based learning and game development have been found to improve learner attitudes and performance (Erümit et al., 2020; Theodoraki & Xinogalos, 2014; Topalli & Cagiltay, 2018; Végh & Stoffová, 2019; Wu & Wang, 2012). A project-based game development unit combined with a game development tool has several potential benefits if implemented carefully: (a) novice programmers should be able to produce visually appealing artifacts relatively early, (b) complex projects are available to students, (c) a wide range of projects are available to students based on their preferences (Erümit et al., 2020; Robertson, 2013; Végh & Stoffová, 2019).

Local Context

South Carolina School District Alpha has 1,401 seventh and eighth-grade students and 2,371 high school students in grades 9 through 12 (South Carolina Department of Education, 2019c). The student population is 43% Black or African-American, 14% Hispanic or Latino, 3% two or more races, 39% White, and 1% or fewer other races; 74% of the district’s students are in poverty (South Carolina Department of Education, 2019a). The percentage of students who scored met or exceeding on the SC Ready exam was

35.6% in English Language Arts and 35.9% in Mathematics (South Carolina Department of Education, 2019b). The percentage of students scoring C or higher on the End-of-Course Assessment was 43.3% in English 1 and 44.5% in Algebra 1 (South Carolina Department of Education, 2019b).

Before 2013, South Carolina School District Alpha offered few CS courses. Students were able to take courses such as business applications, image editing, and digital desktop publishing for their CS credit. In 2013, AP CSA was added to the course offerings. AP CSA was a challenging course for most students. While students performed well on the Advanced Placement (AP) exam, most students struggled initially. In 2014, networking was added to the course offerings. In 2016, AP CSP was added to the course offerings. AP CSP was a much less demanding introduction to computer science than AP CSA, but students struggled with the programming portion of AP CSP. In 2018, an advanced Java course was added to the course offerings as a dual-credit course through a neighboring technical college. The rigorous CS courses were elective, and enrollment consisted of students who were high-achieving or highly interested in computing. South Carolina School District Alpha has two high schools. All advanced CS courses (AP and dual-credit) are offered at Delta High School. Students from the other high school travel to Delta High School to take advanced CS.

In the 2018-2019 school year, 1.1% of the high school student body in South Carolina School District Alpha (26 out of approximately 2373 students) elected to take CS, but the South Carolina Department of Education will require all students to take computer science in the 2019-2020 school year (Malone, 2019). The district was granted a waiver; students who entered grade nine before 2021, and who had earned credit for

one of the previously qualifying courses, would not have to take the required computer science course for graduation. Implementing a successful CS curriculum for all students will be challenging. Abstraction and algorithm analysis and development are challenging topics in computer science for students with no programming background.

Mathematical ability is positively correlated with performance in traditional programming courses (Balmes, 2017; Southern Regional Education Board, 2016). Prerequisites for computer science courses in South Carolina School District Alpha have previously included honors mathematics. Prat et al. (2020) question the evidence for the relevance of mathematical skills to programming, claiming that other measures are more important for learning programming languages. There is an important distinction between learning a programming language and algorithm analysis and development. While learning the syntax of a programming language may involve linguistic skills more than mathematical ones, algorithm analysis and development is inherently mathematical (Cetin & Andrews-Larson, 2016; CollegeBoard, 2020a; Lewis & Papadimitriou, 1998).

In 2017-2018, 35.5% of South Carolina School District Alpha students performed at grade level in mathematics; 51.4% scored a C or higher on the SC end-of-course assessment for algebra 1 (State of South Carolina Department of Education, 2018). Because of their low mathematical proficiency, at least half of the student body is likely to struggle with a traditional programming course. South Carolina has digital literacy standards for grades K – 8, which include computational thinking, algorithms, and programming (South Carolina Department of Education, 2017). As the district gradually implements the K – 8 digital literacy standards, students may enter high school with a stronger background in computational thinking.

A teaching strategy should be implemented that motivates students and makes programming more accessible. In South Carolina School District Alpha, 72.3% of students are in poverty, so most students are unlikely to have access to expensive technology at home (State of South Carolina Department of Education, 2018). It follows that students are also unlikely to enter school with a desire to explore computer science topics. Some students in the district exhibit a high degree of apathy. Many are unmotivated by grades, even the possibility of failing grades. Cell phones are a constant distraction for students in school and lead to many off-task behaviors. Teaching programming through game development may be more successful than through a traditional programming course (Ernst & Clark, 2012; Martins et al., 2018; Thomas et al., 2011; Topalli & Cagiltay, 2018; Wu & Wang, 2012). An intervention that takes advantage of students' affinity for gameplay and technology has the possibility of capturing the interest of previously apathetic students (Boyle et al., 2016; Qian & Clark, 2016).

Statement of the Problem

The South Carolina Department of Education will require all students to take computer science starting in the 2019-2020 school year (Malone, 2019). In the past, few students requested computer science as an elective. Because a small percentage of students have elected to take computer science, South Carolina School District Alpha does not have a tested method of teaching computer science to all students. Even for students who demonstrated an interest in computer science, introductory computer science was a challenging course. Challenges with teaching all students the programming component of computer science are highly probable. In summary,

computer science is a difficult subject even for students who demonstrate an interest in it by electing to take computer science courses. We should expect greater challenges in computer science courses when all students are required to take it.

Purpose Statement

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha.

Research Questions

1. How does the game development project impact participants' ability to analyze and develop algorithms?
2. What is the effect of the game development project on participants' attitudes toward computer science?
3. What is the relationship between participants' attitudes toward computer science and their performance?

Researcher Subjectivities and Positionality

My first career was not in education. I obtained a Bachelor of Science in mathematical sciences, a Master of Science in software engineering, and worked for 10 years as a software engineer before entering education. Marketing departments where I worked dictated that applications should not require people to read or think. They wanted applications that captured and held people's attention and created a sense of urgency to buy. Most of the applications that I developed were designed to sell people things that they did not need. I developed a strong skepticism of technology and the motivations of people selling it.

I always wanted to teach, so in 2008 I entered the South Carolina Program of Alternative Certification for Educators (PACE). Since 2008, I have been teaching mathematics, computer science, and networking in grades 8-12 and at a technical college. Sales tactics in educational technology were similar to those in software development. The customers in education tended to be much less discerning regarding technology purchases. It was common for technology packages to be purchased with little demand for evidence of efficacy. I wanted to develop an ability to read existing research in educational technology and perform research, so I decided to pursue a doctorate in educational technology. I am interested in researching and ultimately improving the learning experiences of students in introductory computer science courses.

My positionality was that of an insider because I conducted research on my own teaching practice (Herr & Anderson, 2005). The focus of my research will be on the effect of a project-based game development unit on the attitudes and performance of students. I will be the instructor and interact with the participants daily. I will directly influence participants' experience of the game development unit and their learning experiences.

The pragmatic paradigm is how I will approach my research. I am fundamentally concerned with solving a problem. Pragmatism is not concerned with methods; it is concerned with understanding and solving a problem using any approach (Creswell & Creswell, 2018). Pragmatism also complements action research, where the immediate goal is to improve conditions for the participants (Zeni, 1998). Computer science has been an elective course until this year. Even students who have an interest in computer science struggle with some aspects of the course, such as algorithm analysis and

development. We face the challenge of teaching these concepts to students who have no interest in the subject and could lack the prerequisite skills necessary to comprehend some topics adequately. The pragmatist paradigm will allow me to select approaches best suited to teaching computer science concepts and measuring student learning.

I want my students, who are the participants, to have a positive learning experience. I also want to maximize their skill development. I will be careful not to underreport the failures of the intervention. If the intervention is not producing desired results, necessary modifications will be made. Assessing the intervention will begin by avoiding leading research questions (Agee, 2009). I enjoy computer science immensely, but I need to remember that many students will be taking computer science because it is required, so they may not share my enthusiasm for the subject. I should remain patient with these students and not communicate frustration.

Definition of Terms

Algorithm: “a set of rules for how to take some input or starting state and produce a corresponding output or end state” (Wilkerson-Jerde, 2014).

Algorithm analysis and development: understanding what existing algorithms do and developing algorithms to solve problems (McGregor & Sykes, 2001).

Application programming interface: a documented interface of available data and functionality.

Attitudes: beliefs, evaluations, or emotional responses toward ourselves, an object, an idea, or a person (Giannakos, 2014; Saldaña, 2021; Simonson, 1979).

Black box testing: test cases are executed using the software's specification without regard to the software's implementation; also referred to as functional testing or specification-based testing (McGregor & Sykes, 2001).

Compiler error: a defect that prevents a high-level language from translating instructions into machine language. A compiler error will also be referred to as a *syntax error*.

Computer science: "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (Tucker et al., 2003). Programming is a subset of computer science.

Content knowledge assessment: an assessment of participants' knowledge and performance. The assessment consisted of a multiple-choice portion and a performance task. When a content knowledge assessment score is referenced without qualification, it refers to the summed scores of the multiple-choice assessment and performance task.

Convention: a practice of writing code that is not enforced by the compiler or a standards body (Fowler, 2004).

Defect: a problem with software (Tian, 2005). In this research, defects will manifest in one of three ways: 1) as a failure of the game to launch; 2) the game ends suddenly due to a fault; 3) the game exhibits unexpected behavior during play. A defect will also be referred to as a *bug*.

Defect detection and removal: the process of identifying and removing defects (Tian, 2005). In this study defect detection will typically be implemented as a test with the goal of inducing a software product to perform incorrectly, thereby exposing a defect

(Sommerville, 2001). Defect removal will be implemented by correcting syntax, runtime, and logic errors in program code. Defect detection and removal will also be referred to as *debugging* or *troubleshooting*.

Design: a conceptual solution for a set of requirements that does not include implementation (Larman, 2002).

Development: the creation of one or more software artifacts (Bass et al., 2006; Larman, 2002; Sommerville, 2001).

Feedback loop: a system that returns a portion of the output signal of the system to the input of the system (Spencer, 1994)

Game design: “the process of creating the rules and content of a game, beginning with a general idea of a game, and ending with a detailed documentation describing all the elements that make up the game: conceptual, functional, artistic, and others” (Swacha et al., 2010, p. 249).

Game development: a software process that includes art, possibly audio, and interactive gameplay. The art will be experienced audiovisually by the player of the game. Art may be created by the student, or it may be an existing artifact that is used with or without modification by the student.

Integrated development environment: software for developing applications that combines several tools into a graphical user interface.

Intrinsic motivation: “engaging in learning opportunities because they are seen as enjoyable, interesting, or relevant to meeting one’s core psychological needs” (Froiland et al., 2012).

Logic error: a defect resulting in incorrect program behavior.

Modding: the process of editing or extending an existing codebase to change the functionality or add new functionality.

Performance: a measure of students’ ability to demonstrate achievement of educational goals. Performance measures are the learning outcomes examined in this study.

Playtesting: black box testing that also tests design. It is possible for a playtest to fail if it functions according to specification, but the client perceives a design flaw.

Posttest: the content knowledge assessment administered after the intervention.

Pretest: the content knowledge assessment administered before the intervention.

Programming: the process of designing and implementing computer instructions to accomplish a goal. Algorithm analysis and development are a subset of programming.

Runtime error: a defect that occurs while the program is running, which will typically terminate the execution of the program unless handled.

Scope creep: project work extends beyond what was originally intended, increasing the time required to complete the project.

Software process: “a set of activities and associated results which produce a software product. [The] activities are: software specification[,] software development[, and] software evolution” (Sommerville, 2001, p. 8). The primary

software process activities in this study will be: (a) requirements gathering, (b) game design, (c) software development, and (d) testing.

YYZ: an archive file used by GameMaker that includes the project file and all assets associated with the game.

CHAPTER 2: LITERATURE REVIEW

Introduction

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha. This study was guided by three research questions: (1) How does the game development project impact participants' ability to analyze and develop algorithms? (2) What is the effect of the game development project on participants' attitudes toward computer science? and (3) What is the relationship between participants' attitudes toward computer science and their performance?

Methodology for the Literature Review

Based on the research questions, six main variables guided the literature search: (a) game development, (b) computer science, (c) learning outcome, (d) attitude, (e) high school, and (f) game development-based learning. The databases used to search for literature were (a) Academic Search Complete, (b) Applied Science & Technology Source, (c) Computer Source, (d) Education Source, (e) ERIC, (f) ProQuest, and (g) Google Scholar. Alternate terms were used to widen the search: (a) "game design" for "game development;" (b) "programming" for "computer science;" (c) "middle school," "K-12," and "K12" for "high school;" (d) "game design-based learning" and "game-based learning" for "game development-based learning;" and (e) "performance" and "achievement" for "learning outcome." An example of a search phrase was ("game

development” OR “game design”) AND (programming OR “computer science”) AND (“high school” OR “middle school” OR “K-12” OR “K12”). A separate set of search terms was used in the literature search related to learning and instructional theories: (a) constructivism, (b) inductive learning, (c) active learning, (d) inquiry-based learning, and (e) project-based learning. Recent information was prioritized by initially searching within the last five years. When necessary, the search was expanded to include the previous ten years. References were mined from the recent literature. Some of the mined literature was older than ten years if it was cited frequently.

This literature review chapter contains three main sections: (a) challenges in teaching introductory programming, (b) theoretical foundation for game development-based learning, and (c) game development-based learning.

Challenges in Teaching Introductory Programming

Computer science is difficult for novices, and introductory computer science courses have high failure rates (Topalli & Cagiltay, 2018; Végh & Stoffová, 2019). Programming, specifically algorithm analysis and development, is the most challenging component of CS (Erol, 2020; Topalli & Cagiltay, 2018). In the United States, CS is rarely a required course (Code.org et al., 2022). In 2018, South Carolina was the first state to require CS for high school graduation. However, due to optional waivers, the requirement did not take effect in practice until the 2020-2021 school year. Four other states have since added CS as a graduation requirement: (a) Arkansas, (b) Nebraska, (c) Nevada, and (d) Tennessee. Mandating CS for all students is a new practice in other countries as well. For example, the United Kingdom and Portugal made computing mandatory for all students in 2014 and 2018, respectively (Gurer et al., 2019; João et al.,

2019). Because compulsory CS is a new phenomenon, little information is available regarding the performance and attitudes of students in the United States who were compelled to take CS.

In most studies reviewed, participants elected to take computer science; therefore, they probably had some interest in the subject. Some schools and majors have added computer science requirements; they have found that teaching computer science to non-technical students is difficult (Culic et al., 2019). Since computer science success is correlated with mathematical ability, students with weak mathematical abilities are likely to struggle in a required computer science course (Balmes, 2017). Students with low natural language aptitude are also expected to struggle with programming (Prat et al., 2020). This section will address two main topics: (a) definitions and metrics associated with assessing the success of a computer science course and (b) failure rates in computer science courses.

Metrics Related to Challenges

Performance is a measure of students' ability to demonstrate achievement of educational goals. Maximizing performance is a primary goal in computer science courses because skill development is a critical aspect of the course. In CS, performance is typically measured using written exams and coding exercises (Alturki, 2016). Written exams assess students' knowledge of programming concepts and ability to analyze algorithms (Alturki, 2016; CollegeBoard, 2020b). Coding exercises and projects assess students' ability to design, implement, and test algorithms to solve problems or accomplish tasks (Alturki, 2016; CollegeBoard, 2020b).

Attitudes are students' beliefs regarding the intended learning content (Giannakos, 2014). Simonson (1979) provides a persuasive argument for why and how attitudes should be measured: a reasonable assumption would be that students will devote more time and remember more when they have positive attitudes toward instruction and content. Some common forms of attitude measurement are (a) questionnaires or surveys, (b) interviews, and (c) observations. While a causal relationship between attitudes and achievement is difficult to demonstrate, a relationship between attitudes and achievement has been established by some researchers (Alvarez et al., 2019; Gurer et al., 2019; Tsai et al., 2019). In a university computer programming course ($N = 242$), Alvarez et al. found significant correlations between (a) performance scores and perceived value and (b) performance scores and perceived self-efficacy. In a university computer science course ($N = 119$), Gurer et al. found a correlation between achievement and attitude with $r = .473$ and $p < .01$. Not all studies have confirmed a relationship between performance and attitude. In a university programming course ($N = 58$), Cetin and Andrews-Larson (2016) implemented an intervention that produced significant performance gains but no change in students' attitudes toward computer programming.

Failure Rates

Computer science majors often fail their first computer programming course (Cheah, 2020; Végh & Stoffová, 2019). Over 30% of computer science students in the world failed or dropped an introductory programming course; some institutions have failure and drop rates of up to 65% (Alturki, 2016). Programming is a learned skill that can be improved with practice (Végh & Stoffová, 2019). Learners must spend a lot of

time programming to learn the skill. Listening to an instructor absent other instructional methods is not generally effective (Al-Makhzoomy, 2018; Gao & Hargis, 2010).

Programming has a steep learning curve and is highly stressful for students (Al-Makhzoomy, 2018; Javidi & Sheybani, 2014). Végh and Stoffová (2019) summarized the work required by students in computer science courses to develop proficient programming skills. Students needed to learn several concepts: (a) data types, (b) data structures, (c) control structures, and (d) programming language syntax. Logical and algorithmic thinking had to be mastered, which can take years of practice that novice CS students do not have. CS students were expected to solve a large number of programming problems with a wide range of difficulties.

Novice introductory computer science students must simultaneously absorb several computing themes: (a) the concept of programming; (b) an abstraction of a computer; (c) the syntax and semantics of programming; (d) standard programming problems and their solutions; and (e) a simple software process consisting of requirements gathering, design, implementation, testing, and quality control (Gurer et al., 2019). Keeping students motivated and eager to continue learning is challenging (Javidi & Sheybani, 2014).

Computer science courses and careers require high levels of math achievement. There is a strong connection between mathematics and computer science (Southern Regional Education Board, 2016). Difficulties in learning to program are related to “difficulties in problem solving activities with logical reasoning and mathematical thinking; they [students] use inadequate studying methods and do not work hard enough to develop programming competences” (João et al., 2019, p. 4). Many computing tasks,

such as implementing and analyzing sorting algorithms, are inherently mathematical (Cetin & Andrews-Larson, 2016; CollegeBoard, 2020a). Review of mathematical concepts such as geometry, algebraic equations, and function must sometimes be included in computer science curricula (Javidi & Sheybani, 2014). Prat (2020) suggests that mathematical ability is overemphasized as a predictor of programming ability. Prat examined students' ability to learn programming languages. However, the syntax of programming is a small part of creating programs; students must understand much more than mere syntax to solve complex computing problems (Cheah, 2020).

Several factors are positively correlated with programming performance: (a) self-efficacy, (b) amount of time programming, (c) attitude, and (d) perceived learning (Erol, 2020; Gurer et al., 2019; Tsai et al., 2019). A positive attitude may be a prerequisite to success in programming. Because programming requires a considerable investment of time and determination to fix numerous errors, students must be persistent to succeed (Cheah, 2020). Confounding findings regarding the correlation between attitude and performance exist. Cetin and Andrews-Larson (2016) conducted a study in which achievement was significantly raised, but attitudes were not significantly raised.

The following section will describe the theoretical foundation for game development-based learning (GDBL). The theory supports GDBL as a possible intervention for addressing the challenges of an introductory computer science course.

Theoretical Foundation for Game Development-Based Learning

GDBL is based primarily on inductive learning. Inductive learning is an essential part of inquiry learning, problem-based learning, project-based learning, case-based teaching, discovery learning, and just-in-time teaching, which are all constructivist

methods (M. J. Prince & Felder, 2006). Constructivism states that knowledge is constructed from experiences, perceptions, interpretations, and interactions with others (Harasim, 2012). Inductive methodologies begin with real, practical information and tasks, which allow the learner to generalize concepts (Gavriel, 2015). Active learning and inquiry-based learning emphasize student ownership of learning and are effective methods of enhancing student learning (Zhu, 2020). Project-based learning (PjBL) has students build a product and has demonstrated positive effects on problem-solving skills and attitudes toward learning (Harris et al., 2015; M. Prince & Felder, 2007).

The following section will describe the learning and instructional theories that support GDBL: (a) constructivism and (b) inductive learning. The GDBL intervention will use aspects of these theories.

Constructivism

Constructivism is both an epistemology and a theory of learning. Constructivism posits that learners construct knowledge from available information, prior knowledge, and new meaningful experiences (Dewey, 1916; Jumaat et al., 2017; Minner et al., 2010). Students learn by actively constructing knowledge, not by passive acquisition. Piaget believed that learners used cognitive structures to understand the environment and assimilate information into existing schemas (M. J. Prince & Felder, 2006; Scholnik et al., 2006). If new information contradicts existing schemas or cannot be integrated into existing schemas, the new information can be memorized but not learned.

Constructivism relies on several principles for effective instruction (M. J. Prince & Felder, 2006). Before new knowledge is presented, topics should be introduced through experiences and contexts familiar to students to incorporate the new knowledge

into existing schemas. Because abstract concepts are more difficult to relate to existing knowledge structures than concrete concepts, real-world applications should be the focus. Information should not be presented that requires drastic changes to existing schemas. Allowing students to improve and augment their conceptual models gradually improves the probability that new knowledge will be incorporated into their schemas. Students should be required to extrapolate or research material provided by the instructor. Dependence on the instructor should be decreased so that students can become independent learners. Students should be encouraged to work together during instruction to support collaborative learning. Perceptions are shaped through interactions with others (Harasim, 2012).

Inductive Learning

Inductive learning starts with real, practical concepts and leads to conclusions about abstract and generalized concepts (Gavriel, 2015; M. Prince & Felder, 2007). Real, practical concepts can take the form of analyzing real data, a case study, or a real-world problem, allowing students to recognize the need for content knowledge and skills immediately. Traditional deductive approaches tend to start with theory and then provide real, practical examples that support the theory. Failure to connect instruction to the real world has contributed to students leaving the sciences (M. Prince & Felder, 2007). When students perceive a benefit or need for learning, their motivation increases (M. J. Prince & Felder, 2006). Inductive learning takes advantage of prior knowledge and learners' desire and ability to recognize patterns. Inductive learning promotes critical thinking skills and self-directed learning. Inductive learning can be implemented in several ways,

three of which will be addressed: (a) active learning, (b) inquiry-based learning, and (c) project-based learning.

Active Learning

With active learning, students apply material to real life, reflect on what they are learning, and internalize what they are learning (Gao & Hargis, 2010). Teamwork in small groups is typical. Active learning students achieve higher conceptual understanding than students in traditional learning approaches (Zhu, 2020). A meta-analysis by Freeman et al. (2014) compared active learning to traditional methods in STEM courses. Freeman et al. found that student performance on assessments increased by about six percent with active learning and that students in traditional environments were 1.5 times more likely to fail. In active learning environments, student satisfaction is higher, and student retention in STEM education increases (Pundak et al., 2010; Zhu, 2020). Instructional methods include collaborative/cooperative, project/problem-based learning, role play, and debates (Gao & Hargis, 2010).

Inquiry-based Learning

Inquiry-based learning is student-centered learning where students design questions and direct inquiry to address challenges for which knowledge has not been provided (M. Prince & Felder, 2007; Silm et al., 2017). Part of the student-centered aspect is student responsibility for learning (Minner et al., 2010). Students are expected to make decisions regarding how and what they learn, identify weaknesses in pursuing knowledge, and request help when necessary. Inquiry-based learning can be used to target higher-order thinking skills (Veletsianos et al., 2016). Inquiry-based learning should be used to encourage technology exploration and increase the effectiveness of

STEM teaching. The 6E instructional model includes: engaging, exploring, explaining, engineering (elaborating), enriching, and evaluating (Lai, 2018). Inquiry-based approaches are time-consuming and should be used judiciously. Teacher-centered methods should be used for lower-order thinking and skills to increase efficiency (Veletsianos et al., 2016).

Project-Based Learning

Project-based learning reflects the theory of constructivism (Jumaat et al., 2017). A problem or question motivates learning by constructing an artifact or project in an authentic context (Helle et al., 2006). Open-ended problems are best (Papanikolaou & Boubouka, 2011). Learners collect, analyze, and synthesize information (Papanikolaou & Boubouka, 2011). Learners acquire knowledge and skills through an extended inquiry process involving complex, authentic questions and developing products (English & Kitsantas, 2013). The teacher's role is to structure activities and facilitate learning through scaffolding and feedback (English & Kitsantas, 2013). After the activity, students produce a product, and their performance is evaluated (Helle et al., 2006; Jumaat et al., 2017). Reflection and revision of the work product are essential; therefore, project submission, evaluation, and revision may go through several iterations (Papanikolaou & Boubouka, 2011).

PjBL learning can be implemented in several different ways and roughly categorized by three models: (a) project exercise, (b) project component, and (c) project orientation (Helle et al., 2006). In a project exercise, students apply knowledge and skills already acquired to complete a project defined by the instructor. The project is confined to a single subject and may take the form of a capstone event for a unit or the entire

course. In a project component, the project solves a real-world problem and may be interdisciplinary. Objectives include the development of problem-solving and time management skills. Project orientation refers to an entire program of study that is project-based. The project requirements determine the instruction and subject material. Students often have input into the projects that they complete. Larmer et al. (2015) identify seven essential project design elements: (a) challenging problem or question, (b) sustained inquiry, (c) authenticity, (d) student voice and choice, (e) reflection, (f) critique and revision, and (g) public project.

PjBL approaches have demonstrated the potential to increase student performance. Saavedra et al. (2021) employed PjBL in high school AP courses. The PjBL curriculum produced a four percent increase in qualifying scores among all students and an eight percent increase in qualifying scores among exam-takers. Some studies of PjBL implementations found improved attitudes, conceptual understanding, and problem-solving, but only comparable results for performance on content knowledge assessments (M. Prince & Felder, 2007). The Hewlett Foundation has a group of schools committed to schoolwide PjBL (Larmer et al., 2015). These schools averaged higher mathematics, reading, and science scores on the OECD PISA-based Test for Schools than comparative schools. Students in the Hewlett Foundation schools achieved higher scores on state tests in English and mathematics. They also saw improvements in attitude, self-efficacy, collaboration, and engagement.

Inductive learning can increase student motivation and performance. Students understand the utility of the content from the beginning. Inductive learning exploits students' innate desire to seek patterns and construct meaning. The following section

will describe GDBL. GDBL is based on the principles outlined in the learning theories and instructional methodologies discussed previously.

Game Development-based Learning

GDBL attempts to leverage many young people's affinity for digital games (Anderson & Jiang, 2018). Gamification and game-based learning attempt to educate students through gameplay or incorporating gaming elements (Kingsley & Grabner-Hagen, 2015). GDBL has students learn through developing games instead of primarily by playing games. GDBL is a specific type of project-based learning where the artifact is a playable game. GDBL has been shown to have positive effects on learning and attitudes (Stoffová, 2019; Topalli & Cagiltay, 2018).

The following section will describe three main topics: (a) gamification and game-based learning, (b) GDBL, and (c) project-based learning alternatives to GDBL. Evidence will be provided that supports GDBL with an emphasis on PjBL as an intervention for an introductory computer science course. The characteristics of GDBL that make it an effective intervention will be examined.

Gamification and Game-Based Learning (GBL)

Young learners show a high degree of interest in gaming (Anderson & Jiang, 2018). Anderson and Jiang (2018) of the Pew Research Center reported the following data:

- Eighty-four percent of teens (75% female and 92% male) have access to a game console at home.
- Ninety percent (83% female and 97% male) play video games.

- Eighty-five percent of teens from households earning less than \$30,000 per year have a game console at home.

Gamification

Gamification is the use of game elements in non-game contexts. The intent of gamification is to combine intrinsic and extrinsic motivation to increase motivation, engagement, and active participation. Symbols of learning progression, like badges, are an example of gamification (Kyewski & Krämer, 2018). Gamification has the potential to make schoolwork feel like activities enjoyed outside of school. Kingsley and Grabner-Hagen (2015) reported the following student perceptions of gamification:

- 95.8% of students reported a preference for days when games were used for learning.
- 87.2% of students reported that technology made learning easier.
- 93.6% of students enjoyed earning gaming badges.

However, the competitive elements (badges and leaderboards) of a gamified classroom may produce negative educational outcomes. Students in the gamified classroom showed decreases in motivation, satisfaction, and empowerment. Students in the gamified classroom also had lower final exam scores. Lower exam scores were attributed to a reduction in intrinsic motivation. Giving rewards for tasks that are already interesting decreases intrinsic motivation (Hanus & Fox, 2015).

GBL

All et al. (2015) proposed three categories of desired learning outcomes: learning, motivation, and efficiency. Learning outcomes have the following subcomponents: (a) increased interest in the subject matter, (b) increased objective performance, and (c)

learner ability to transfer knowledge and skills acquired during digital game-based learning (DGBL) to real-world contexts. Motivation outcomes have the following subcomponents: (a) enjoyment and (b) increased motivation. Efficiency outcomes have the following subcomponents: (a) time management and (b) cost-effectiveness.

Seven randomized control trials (RCTs) measuring knowledge acquisition reported that DGBL performed better than the control condition (Boyle et al., 2016). Ten randomized controlled trials measuring skill acquisition reported that DGBL performed better than control groups. Elements of uncertainty enhanced learning. Variable priority training is superior to full emphasis training. Design-based games were more effective than educational or entertainment games (Qian & Clark, 2016).

Computer games can improve engagement and motivation. Games are integral to cognitive and social development (Hwang & Wu, 2012). Engagement is related to cognitive and emotional involvement (Abdul Jabbar & Felicia, 2015). Cooperative games led to higher motivation than competitive games. Enjoyment of games was improved with rewards such as earning points and finding rare items (Boyle et al., 2016). “Multirole-play or collaborative role-play works effectively when coupled with learning tools and interactive elements and materials to motivate and help learning” (Abdul Jabbar & Felicia, 2015, p. 768). Challenges and conflicts must be matched to the abilities of the students. The following types of games and game elements had benefits: (a) role-playing games for immersion, (b) massively multiplayer online role-playing games for an engaging experience, (c) competitive play for active GBL, (d) collaborative play, (e) playing an intelligent fictional hero, (f) puzzle-based and simple gaming mechanics for engagement and learning, (g) virtual reality and multimedia elements for playful learning

and discoveries, (h) challenges and conflicts for motivation, and (i) control and choices for attention and interests (Abdul Jabbar & Felicia, 2015).

GDBL

In GDBL, learners modify or develop games as an integral part of a computer science course using a game development framework. (Wu & Wang, 2012). GDBL is closely related to game design-based learning; some researchers define game design-based learning in the same way GDBL is defined. Swacha et al. (2010, p. 249) define game design as “the process of creating the rules and content of a game, beginning with a general idea of a game, and ending with a detailed documentation describing all the elements that make up the game: conceptual, functional, artistic, and others.” Swacha et al. expressly exclude game programming from game design.

In the software engineering discipline, development is the term used to describe the creation of any software artifact such as (a) requirements documents, (b) architecture documents, (c) design documents, (d) program code, and (e) quality control documents (Bass et al., 2006; Larman, 2002; Sommerville, 2001). The software process includes all activities leading to a final software product (Sommerville, 2001). Design has a specific meaning in software engineering: a conceptual solution for a set of requirements that does not include implementation (Larman, 2002). In this study, development refers to the entire software process; therefore, design is a subset of development.

Effect on Performance

GDBL generally produces positive effects on performance (Johnson, 2017; Kynigos & Grizioti, 2020). Topalli and Cagiltay (2018) conducted a study using game development to teach introductory programming. Three hundred twenty-two students

took a Senior-project course. These students had completed an introduction to computer programming course offered in two versions: (a) 48 of them took an enriched version, and (b) 274 took the traditional course. The traditional course used the C programming language, theoretical lectures, and laboratory sessions where students wrote console programs. The enriched course added 15 minutes of Scratch instruction to the laboratory sessions of the traditional course. Students also created a game in Scratch for the enriched course. The course grades of the enriched programming course participants were significantly better than the students' grades in the traditional course. In the Senior-project course, students who took the enriched introductory programming course performed better on their senior projects.

Végh and Stoffová (2019) conducted an experiment to determine game development's effect on performance in an object-oriented programming (OOP) course. The average test score from the OOP class without gaming was 59.57%, while the average test score from the gaming class was 64.55%. Results for subtopics were mixed. The gaming students had more fun, were more engaged, and had more ideas on improving their programs. Knowledge gains by students, student surveys, and teacher surveys support game creation for improving computer science competency (Ernst & Clark, 2012; Javidi & Sheybani, 2014).

Effect on Attitudes

GDBL generally produces positive effects on attitude (Erümit et al., 2020; Hughes-Roberts et al., 2020; Johnson, 2017). Theodoraki and Xinogalos (2014) showed that game development could significantly increase students' motivation and enjoyment

of programming. Games can be played within lectures to improve participation (Wu & Wang, 2012).

Years of experiences and the researches [*sic*] show that both, beginners and advanced programmers consider computer game programming to be interesting and entertaining, so they can playfully acquire not only new knowledge but also other experiences and skills from the creation and implementation of software applications. (Stoffová, 2019, p. 40)

The motivation of students can be achieved by making lessons playful and competitive (Stoffová, 2019). By providing students with a more rewarding and creative environment, GDBL strongly incentivizes students to practice programming (Theodoraki & Xinogalos, 2014). Allowing students to design games in groups can increase enjoyment and motivation (Swacha et al., 2010). Javidi and Sheybani (2014) showed that game development could increase enrollment in advanced STEM courses.

Game Development Frameworks

Scratch. Cucinelli et al. (2018) used Scratch in their study as an accessible entry point to programming. Participants ($N = 30$) ranged in age from seven to 75 years old. The intervention was a five-hour workshop involving game development using Scratch. The following aspects were measured: (a) storytelling, (b) problem-solving, (c) collaborating, (d) creativity, (e) understanding game rules, and (f) programming. Before the intervention, participants rated themselves the lowest in the programming aspect; many identified their programming skill with a zero. After the intervention, the programming aspect showed the largest progression.

Topalli and Cagiltay (2018) used Scratch and game creation in their study to enrich a traditional programming course. Because Scratch was a block-based programming language, syntax errors were reduced. Scratch allowed for an algorithm-first approach because students did not have to struggle with learning programming language syntax and troubleshooting syntax errors.

Stagecast Creator. Denner et al. (2012) conducted a study in which Stagecast Creator was used in an after-school class for middle school girls to measure what the participants learned when programming a game. The participants were 59 girls who volunteered for an after-school program focused on computer game programming. Participants completed one to five games each. Each game was completed in four to six weeks with one to two hours per week of development time. Participants were not required to demonstrate specific programming skills. The study was designed to measure what programming tasks participants would undertake independently. Participants engaged in moderate levels of complex programming. Participants did not persist in the face of challenges and abandoned features requiring complex programming constructs. Participants rarely made more than one attempt at debugging their programs when they did not work as expected.

GameMaker. GameMaker can positively affect students' attitudes regarding technology and computer science and their perceptions of the class and instructor (Doman et al., 2015). GameMaker has advantages for teaching an introductory programming course because of the IDE and GameMaker programming language, which is similar to other C-based languages (Doman et al., 2015). The use of gaming and GameMaker may increase computer science performance. Ernst and Clark (2012)

reported a mild increase in the number of students intending to pursue a career related to computer science. A case study suggested several modifications to GDBL using GameMaker (Johnson, 2017). Students tended to begin implementation before adequate planning; therefore, game design assignments should be included. A significant amount of direct instruction was required to teach basic programming concepts and avoid frustrating programming experiences. Topics that required direct instruction included (a) problem decomposition, (b) planning, (c) testing, and (d) debugging.

Project-based Learning Alternatives to GDBL

Viable project-based alternatives to GDBL were reviewed. Two examples of such alternatives are physical devices and applications, which are discussed below.

Physical Devices

Console programs reduce engagement due to the absence of graphics and limited ability to interact with the programs. Because students born in 2000 and later have been constantly exposed to modern technologies, teaching and learning methods should mimic that media. Knowledge acquisition is improved when students understand practical applications and are engaged (Perenc et al., 2019).

Robotics has been utilized to teach programming using a physical, instead of a purely virtual, medium. Robotics helps students visualize their programs' output and makes the programming process more understandable (Erol, 2020; Pullan, 2013). Lego Mindstorms NXT is an example of a robotics kit that has been widely used to teach programming. Arduino boards are also popular methods for teaching programming. Combining technology with active learning improved students' understanding of the subject matter, attitude, and self-efficacy.

Applications

Malik et al. (2019) studied the PROBSOL application's effectiveness in improving novice programmers' problem-solving skills. Participants consisted of 65 university students. The use of PROBSOL was found to support students' cognitive gains and engagement. Programming understanding and problem-solving skills improved. Students' attitudes toward completing exercise questions also improved. Student achievement was improved, and attrition was reduced. The failure rate of students using PROBSOL was 6% compared with 9% using a traditional approach. The dropout rate of students using PROBSOL was 3% compared with 7% using the traditional approach.

Summary

Computer science is a difficult course for novices. High failure rates are common for students who elect to take computer science courses (Alturki, 2016). A reasonable assumption would be that students forced to take CS as a required course would find CS even more difficult. Inductive learning approaches are effective for improving performance and attitudes (M. J. Prince & Felder, 2006). Guided instruction, including appropriate scaffolding and just-in-time instruction, should be included (Kirschner et al., 2006; Novak, 2011; Sweller et al., 2007). PjBL produces significantly better results than traditional methods in (a) assessments of conceptual understanding, (b) ability to solve problems, and (c) attitudes to learning (Çelik et al., 2018; M. J. Prince & Felder, 2006).

Young learners spend a significant amount of their free time playing games (Anderson & Jiang, 2018). Gamification and GBL attempt to leverage students' enjoyment of games to make education more engaging. Similarly, GDBL engages

students by having them develop a playable game using a PjBL approach (Wu & Wang, 2012). GDBL is an effective methodology for improving student performance and attitudes (Ernst & Clark, 2012; Stoffová, 2019; Topalli & Cagiltay, 2018; Végh & Stoffová, 2019). The next chapter will describe the proposed methodology for implementing a GDBL intervention.

CHAPTER 3: METHOD

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha. This study was guided by three research questions: (1) How does the game development project impact participants' ability to analyze and develop algorithms? (2) What is the effect of the game development project on participants' attitudes toward computer science? and (3) What is the relationship between participants' attitudes toward computer science and their performance?

Research Design

Action research was utilized to address the purpose of this study. The South Carolina Department of Education requires that all students pass a rigorous high school computer science course to graduate (Malone, 2019). South Carolina School District Alpha must implement a solution to this new requirement and analyze the effect of the solution. Action research was appropriate for this study because immediate action and evaluation were needed to address this local-level problem of practice (Mertler, 2019). Additional indicators for action research in this study were (a) the primary motivation for this study was practical, not theoretical; and (b) the research was performed by a practicing professional instead of professional researchers (Willis & Edwards, 2014). The researcher worked with other stakeholders to understand how students performed in a rigorous computer science course. We implemented changes to the traditional

computer science curriculum in an attempt to improve the experiences of students and improve their chances of completing the computer science course.

Action research is defined as a form of systematic investigation in which the researcher(s) and other stakeholders attempt to address problems in the setting in which they work (Willis & Edwards, 2014). Action research is distinct from other forms of research in that it is participative since the researcher is more than an objective observer; it “allows teachers to study their own classrooms” (Mertler, 2019, p. 6). In action research, a problem is specified, a new solution is developed, and the effectiveness of the solution is evaluated. Mertler defines action research as a “cyclical process of planning, acting, developing, and reflecting” (2019, p. 18). The stages do not necessarily occur linearly or in the same order for every cycle. The flexibility to modify the solution during the study is another distinguishing feature of action research, which was critical to this study.

This study used a convergent parallel mixed-methods design, which included qualitative and quantitative elements (Mertler, 2019). This design is also called concurrent triangulation mixed-methods (Edmonds & Kennedy, 2017). Qualitative and quantitative data were collected concurrently and emphasized equally (Mertler, 2019). The two sources of data were compared to determine if the findings confirm each other (Creswell & Creswell, 2018). The qualitative data added context and aided with the interpretation of the quantitative data (Tracy, 2020). Qualitative research has three core concepts: self-reflexivity, context, and thick description (Tracy, 2020). Self-reflexivity describes the researcher’s consideration of how their beliefs and roles influence their interaction and interpretation of the study (Tracy, 2020). Context can be understood by

contrast with a quantitative study in a laboratory. Qualitative studies typically take place in the natural environment and consider how the setting affects the topic of study (Tracy, 2020). Thick description describes recording fine detail about the context to derive meaning (Tracy, 2020). Qualitative research is useful for uncovering unanticipated issues, which will be of particular benefit to this study because of the novelty of the situation.

Quantitative research typically involves measuring variables using instruments and performing statistical analysis on the data (Creswell & Creswell, 2018). This study measured the effect of a game development project on participants' performance and attitudes. The relationship between performance and attitude was also measured. A true experimental design was not possible because there was no control group, nor could participants be assigned to groups randomly (Creswell & Creswell, 2018). A one-group pretest-posttest design was used to measure the change in performance after the intervention (Creswell & Creswell, 2018). A survey design is ideal for producing a quantitative description of student attitudes and how they change after the intervention (Creswell & Creswell, 2018). The survey was longitudinal with two iterations of data collection, one before the intervention and one after the intervention. A correlational design was used to measure the relationship between performance and attitude after the intervention (Mertler, 2019).

I approached this study with a pragmatic worldview. I was fundamentally concerned with improving computer science learning experiences for my students. The pragmatic worldview gives researchers the freedom to choose methods that best fit the research purpose (Creswell & Creswell, 2018).

Setting

The intervention occurred in Beta Middle School (BMS) as part of a year-long game design and development course. BMS was located in South Carolina, and was part of South Carolina School District Alpha. BMS was home to the STEM magnet program serving all South Carolina School District Alpha students. The intervention at BMS was available to all eighth-grade students in the district, which had three middle schools. The intervention took place in a computer lab, which had a desktop for each participant. Each participant was also issued a Chromebook as part of a one-to-one technology initiative in the district. GameMaker Studio 2, the game development software, was licensed per installation. Participants with Windows or Mac computers at home were able to install a free version of the software if they wanted to use it at home.

Participants

Study participants ($N = 28$) were a purposive sample of eighth-grade students in the Science, Technology, Engineering, and Mathematics (STEM) magnet program who were assigned to the game design and development course by South Carolina School District Alpha. Purposive sampling was used to select information-rich cases that would maximize understanding of the intervention effects (Bloomberg & Volpe, 2015; Creswell & Creswell, 2018). These participants had a history of academic success. Students in the high school game design and development courses had a wide range of academic performance and behavior histories. The researcher wanted to test the intervention on participants who were academically motivated and were unlikely to present classroom management challenges. If the intervention failed to improve the attitudes and

performance of the STEM participants, it would likely fail with students with lower academic achievement.

Participants applied to the STEM program and passed a performance-based engineering assessment. There were 28 participants in the course. The age of the eighth-grade participants was 13 and 14. There were 16 females and 12 males. Two participants were American Indian or Alaska Native; one was Asian; six were Black or African American; one was Black or African American & White; 18 were White. See Table 3.1 for the demographics of participants who were interviewed.

Table 3.1 Interviewed Participant Demographics

Pseudonym	Age	Sex	Race
Bree	14	Female	White
Abegail	14	Female	White
Qianna	14	Female	Black or African American
Jonie	13	Female	White
Marlena	14	Female	American Indian or Alaska Native
Aleesha	14	Female	White
Julia	14	Female	White
Monster Fan	14	Male	White
Annabelle	14	Female	White
Teddie	13	Male	White
Pibb	14	Male	White
Oakley	14	Male	White

Note. $N = 28$. Age as of 4/29/22.

The STEM participants were high-achieving and had demonstrated consistent academic success. The STEM participants took algebra 1 honors and English 1 honors in the eighth grade. 20 participants were recognized as gifted and talented. See Table 3.2 for the Math and ELA performance on the 2021 South Carolina College-and Career-

Ready Assessments (SC READY). Table 3.3 shows the academic information for each participant who was interviewed.

Table 3.2 SC READY Math and ELA Performance

Performance	2021 SC READY Performance Level	
	ELA	Math
Exceeds Expectations	21	18
Meets Expectations	6	6
Approaches Expectations	1	4
Does not Meet Expectations	0	0

Note. $N = 28$.

Table 3.3 Interviewed Participant Academic History

Pseudonym	Gifted and Talented	2021 SC READY Performance Level	
		ELA	Math
Bree	Yes	Exceeds Expectations	Exceeds Expectations
Abegail	No	Meets Expectations	Meets Expectations
Qianna	Yes	Exceeds Expectations	Meets Expectations
Jonie	Yes	Meets Expectations	Approaches Expectations
Marlena	Yes	Meets Expectations	Exceeds Expectations
Aleesha	No	Exceeds Expectations	Approaches Expectations
Julia	Yes	Exceeds Expectations	Exceeds Expectations
Monster Fan	Yes	Exceeds Expectations	Exceeds Expectations
Annabelle	No	Meets Expectations	Exceeds Expectations
Teddie	No	Exceeds Expectations	Meets Expectations
Pibb	No	Exceeds Expectations	Exceeds Expectations
Oakley	Yes	Exceeds Expectations	Exceeds Expectations

Note. $N = 28$.

STEM students were divided into two groups by the district. Group one was higher performing and took algebra one honors during semester one, geometry honors during semester two, and Project Lead the Way's Introduction to Engineering Design throughout the year. The study participants were in group two. They took algebra one honors and Game Design and Development honors throughout the year. South Carolina released computer science and digital literacy standards in 2017 for Kindergarten through grade eight (South Carolina Department of Education, 2017). Study participants should have been exposed to the grade seven standards, which included a set of standards on algorithms and programming:

1. Design, evaluate, and modify simple algorithms (e.g., steps to make a sandwich; steps to a popular dance; steps for sending an email).
2. Use and compare simple coding control structures (e.g., if-then, loops).
3. Decompose problems into subproblems and write code to solve the subproblems (i.e., break down a problem into smaller parts).
4. Design and code programs to solve problems.
5. Identify variables and compare the types of data stored as variables.

The participants demonstrated no evidence of having been exposed to these standards, much less having mastered them. When asked directly about their exposure to algorithms and programming, they gave no indication of learning these standards in grade seven.

One participant claimed some programming experience in the form of modding for the Half-Life series. Modding is the practice of modifying or adding to an existing codebase.

Participants developed four games and a cutscene before the intervention, which progressed in difficulty and student autonomy. Early games were scripted and

straightforward, providing explicit detail on nearly every implementation step. Subsequent games provided less implementation detail and allowed for more participant design decisions. Participants developed the following games: (a) Pinball, (b) Ball Bouncer, (c) Matching, (d) 31 / Scat, and (e) Sky is Falling cutscene. Participants were exposed to several general programming concepts: (a) variables and data types, (b) number calculations, (c) booleans and selection statements, (d) loops, (e) functions, and (f) arrays. Participants learned how to use GameMaker Studio 2 and GameMaker Language to create games (YoYo Games, 2021). The Scat game had a significant jump in difficulty. Participants were required to design loops, functions, and arrays, which were new concepts for them. Most participants demonstrated a lack of mastery and needed significant assistance to complete the game.

The researcher's role was to develop the intervention, perform the research, and act as the instructor for the course. The researcher's responsibility to the student participants was to maximize their learning, giving them the best opportunity to pass the course and fulfill their computer science graduation requirement. The researcher strove to report the findings of the study objectively. If participants provided inaccurate information to please the researcher, the results might have been skewed. The researcher encouraged participants to disregard any concerns over how the researcher perceived their input. All participants ($N = 28$) were invited for interviews after the intervention. Fourteen participants returned the assent forms, and thirteen registered for an interview time. One participant had several scheduling conflicts and could not be interviewed. In the end, 12 participants were interviewed. See Tables 3.1 or 3.3 for a list of the participants who were interviewed. Of the participants who were interviewed, (a) eight

were female, and four were male; (b) 10 were White, one was Black or African American, and one was American Indian or Alaska Native; (c) seven were Gifted and Talented; (d) on the 2021 SC Ready ELA assessment, eight exceeded and four met expectations; and (e) on the 2021 SC Ready math assessment, seven exceeded, three met, and two approached expectations.

Intervention

The intervention in my action research was a PjBL unit in a game development course. Participants designed, implemented, and tested a computer game of their choice for their project. Traditional programming courses had high failure rates due to students with low motivation, low math ability, and low abstract thinking ability (Balmes, 2017; Culic et al., 2019; Martins et al., 2018; Végh & Stoffová, 2019). Engaging participants in the creation of a complex gaming system had the potential to improve attitudes and problem-solving abilities (Akcaoglu, 2014; Ernst & Clark, 2012; Javidi & Sheybani, 2014; Theodoraki & Xinogalos, 2014; Wu & Wang, 2012). Guided instruction was provided just in time to ensure that participants had the skills to complete the unit successfully (M. Prince & Felder, 2007).

Background

The game development course was piloted in the 2019-2020 school year at Gamma Middle School in South Carolina School District Alpha. The researcher was the instructor for the pilot course. The students were in grade eight and part of the Advancement Via Individual Determination program. Participants partially completed a curriculum created by Zulama called Introduction to Computer Science through Game Design (Carnegie Learning, 2021). The Zulama curriculum was utilized without

modification. The curriculum consisted of directions for developing six predesigned games and one game of student choice. The rules and visual assets were created for the students in the six predesigned games. Figure 3.1 shows the rule set for the first game, Zulama Pinball. Students were responsible for implementing game logic in GameMaker Studio 2 using the GameMaker Language. In the first game, the instructions detailed nearly every action and line of code that the students required to make the game work. Students could copy the directions verbatim and make the game work. Figure 3.2 shows the code provided to students in Zulama Pinball for the drop_button Create event. Subsequent games removed explicit directions for behavior that had already been implemented. Instead of providing students with code to type, the students were given partial code segments that needed to be completed, or students were given a rule that to be implemented without any starter code. Figure 3.3 shows a code template for scoring a hand in Zulama Scat. Participants were responsible for implementing the behavior described in the comments (text following double forward slash).

Rule Set: (Here's how we play!)	
Rule	Context Sensitive Rules
The player can exit the game at any time via clicking on the quit button.	
Play begins when the player clicks the drop button.	<ul style="list-style-type: none"> ❖ The ball falls downward from a random spot along the top of the playing area with gravity. ❖ As the ball bounces off of playing pieces the player's score is increased and the piece is destroyed. ❖ The ball will bounce off the walls and user paddle. ❖ When the ball collides with the bottom blocks a life is lost. ❖ When all lives are lost or when all pieces are destroyed, play is over.
Bonus elements are created when score reaches 100 and 200 points.	<ul style="list-style-type: none"> ❖ A gold star randomly appears in playing area when score reaches 100 and 200 points. ❖ A 25 point score bonus is awarded when ball collides with first gold star. The paddle length also increases. ❖ The score is doubled when the ball collides with the second gold star. ❖ The stars are destroyed when hit by ball.
The player controls the user paddle with the left and right arrow keys.	<ul style="list-style-type: none"> ❖ The user paddle will move left and right based on key pressed. ❖ The user paddle will not move off the screen.

Figure 3.1 Zulama Pinball Rule Set

```

4 drop_button: Create
*Create
1 //keep track if there is a ball on screen
2 global.ballOnScreen = false;
3 //initialize the score to 0
4 score = 0;
5 //initialize player lives to 3
6 lives = 3;
7 //initialize the number of playing pieces on the level
8 global.pieces left = PIECES ON LEVEL;
9 //initialize first bonus
10 global.first bonus = true;
11 //keep track of how many gold bonus objects have been created
12 global.bonus counter = 0;
13 //initialize score for first bonus
14 global.bonus score = 100;
15
15/15 Col:1 Ch:1

```

Figure 3.2 Zulama Pinball Drop Button Create Event

```

function func_score_hand(card1, card2, card3){
    // Return the score for the supplied three cards
    // Save the three arguments in a card array, sc_card
    // Set up an array, suits, for the four suits
    // Initialize the final_score to zero.
    // This will be the value returned by the function.
    // Begin the outer for loop to score one suit at a
    // time using index i
    for (i = 0; i < 4; i += 1)
    {
        // Initialize the temporary hand total for current
        // suit to zero.
        // Begin the inner loop to look at each card using
        // index j
        for (j = 0; j < 3; j += 1)
        {
            // Compare sc_card[j].suit with suits[i].
            // If they match, add the card's value to the
            // temporary total.
        }
        // Once the inner loop completes
        // compare the temporary total with the final_score
        // to see if this suit's total should become the new
        // final_score.
        // Tip: either use if statement to find which is
        // greater
        // or look up max function in GameMaker documentation.
    }
    return final_score;
}

```

Figure 3.3 Zulama Scat Score Hand Function

The students did not have a positive learning experience in 2019-2020. The instructor's training for the course consisted of completing the Zulama curriculum. The instructor was relatively new to GameMaker and was unprepared for the numerous mistakes that students made with GameMaker. Students did not follow the directions in the curriculum carefully, and they were careless about changing settings in GameMaker.

The instructor was unable to correct those mistakes expediently. Therefore, students spent most of their time struggling with minor syntax errors and environment settings instead of meeting the course objectives. Deadlines for project deliverables were two weeks or more. Students exhibited excessive off-task behavior because they did not feel a sense of urgency to complete tasks. The year ended prematurely due to COVID while students were working on the fourth predesigned game.

In the 2020-2021 school year, the game development course was moved to Beta Middle School. The students were in grade eight and part of the STEM program. The researcher was the instructor for this course. This year realized significant gains over the previous year in terms of learning objectives met and student productivity. The instructor was able to efficiently resolve issues with GameMaker. Guided instruction was added to supplement the Zulama curriculum. Deliverables were shortened to notify the instructor and students earlier if the students were not maintaining the desired pace. Projects still took twice as long as desired. Problems noticed in the previous year persisted but were much less severe:

- Students had difficulty reading the instructions provided in Zulama and implementing them in GameMaker.
- Any deadline longer than about two days resulted in off-task behavior.
- The instructor was reacting to trivial problems and solving them quickly, but too much time was lost to GameMaker settings and minor syntax errors.
- Students completed tasks in the curriculum but were not learning the desired skills and concepts.

- Students were unable to effectively interpret error reports in GameMaker and resolve errors without assistance.
- Students were unable to use the GameMaker application programming interface (API) as a programming language reference.

In the 2021-2022 school year, the researcher was again the game design and development course instructor. In the first semester of the 2021-2022 school year, the instructor corrected most of the severe problems experienced in the previous two years. For the first project, Pinball, the instructor completed the entire project with the class. The instructor demonstrated how the curriculum instructions should be interpreted and implemented in GameMaker. The instructor identified and discussed common errors in advance, just before students were likely to encounter them. Every student implemented the game on their computer to maximize their exposure to the game development software, GameMaker. The instructor checked students' progress every two days to keep students on task and correct errors early.

After Pinball, students worked in pairs and completed three more projects and one cutscene using the Zulama curriculum: Ball Bouncer, Matching, Scat, and Sky is Falling cutscene. Checkpoints were established to keep students on task and allow the instructor to provide scaffolding to struggling students. See Appendix A for the checkpoints.

Guided instruction was added just before students needed to apply new skills. Participants intermittently completed part of a module on Khan Academy, Intro to JavaScript (JS): Drawing & Animation, where they learned basic programming techniques and syntax (Northway et al., n.d.). Programming techniques were delivered just in time for use in game development (M. Prince & Felder, 2007).

Students experienced significant difficulty with the Scat project. Iteration and arrays were introduced and were challenging for students. Students performed poorly on assessments related to iteration and arrays. Most students struggled to complete the programming tasks associated with the Scat game.

PjBL Aspects

Effort was made to improve participants' attitudes and performance by designing PjBL with the following elements: (a) challenge, (b) sustained inquiry, (c) authenticity, (d) choice, (e) reflection, (f) critique and revision, (g) a public product, and (h) collaboration (Blumenfeld et al., 1991; Helle et al., 2006; Jumaat et al., 2017; Larmer et al., 2015; M. Prince & Felder, 2007). Before the intervention, participants complained that they did not enjoy the games. Some participants were content to submit incomplete games and games with defects. When participants chose and designed their games, they enjoyed working on their games. Presenting their games to clients encouraged participants to take pride in their work and correct problems.

An anticipated risk was that participants might resist engaging in cognitively demanding tasks; however, they responded favorably to the project because they found the project interesting and valuable, and they perceived that they could complete the project (Blumenfeld et al., 1991). The researcher provided appropriate scaffolding, so that the participants found the challenge manageable. Having participants develop games for clients improved authenticity (Papanikolaou & Boubouka, 2011). Two Participants developed a serious game for a teacher; other participants developed entertainment games for their peers. Participants found the project challenging. The project was their first experience in game design. They saw five game design documents and completed the

implementations, but they had not undertaken game design. They had to apply their coding skills without specific prompts for the first time. Participants had to research the GameMaker API to implement some aspects of their games. Participants had a great deal of choice in the project. They negotiated the game requirements with their clients, made design and implementation decisions, had very few restrictions on the games' aesthetics, and chose additional clients after the first. Participants experienced cooperation by pair programming with a peer and working with a client. Table 3.4 shows a summary of how PjBL design elements were implemented.

Table 3.4 Project-Based Design Element Implementations

Design Element	Implementation
Challenge	This was the participants' first design experience. They had to enumerate goals and rules constrained by their current abilities. Their game had to include alarms, step events, and loops, all of which were poorly understood by most participants at the start of the intervention. Many participants designed overly challenging games that had to be scaled back. For instance, some participants wanted to create multiplayer games requiring networking.
Sustained inquiry	Participants were required to create games with features that had not been implemented in the past. This required students to research methods for implementing novel game behavior. Several participants implemented platforming mechanics, which had not been covered before the intervention.
Authenticity	Several elements of a software process were included in the project: (a) a requirements and design document, (b) feedback from the researcher and designated client, (c) collaboration with a partner to iteratively develop a digital game, (d) weekly progress reports, and (e) difficult decisions to remove features that threatened a hard project deadline.
Participant choice	Participants had a high degree of freedom to select the theme of their game and include features of their choice. Aesthetic elements, including graphics and sounds, were under participants' control. Participants were constrained

Design Element	Implementation
	by a hard project deadline and required programming elements.
Reflection	Participants were required to reflect on their project progress in weekly progress reports. When implementing features, participants reviewed prior work for applicable solutions. After playtesting during iterative development, participants had to analyze algorithms that were not producing desired behavior.
Critique and revision	Participants received feedback from the researcher and client during the design phase. The researcher provided feedback on features that were likely to cause project failure and features that were unlikely to satisfy required programming elements. Clients provided feedback on playability. Participants revised their work based on feedback.
Public project	Participants developed a game to be played by their peers. One group developed a serious game for their music class to train students on transposing pitch based on the instrument.
Collaboration	Participants worked with a partner to develop their games. They also had to negotiate gameplay with one or more clients.

Participants perceived that they were capable of completing the project (Blumenfeld et al., 1991). The instructor carefully reviewed participants' design choices to ensure that participants were creating a project that they could complete in the allowed time frame. This PjBL unit was intended for participants to apply existing skills when coding their games. Because knowledge and skills in the implementation phase were familiar from previous work, the likelihood of student resistance was reduced (M. Prince & Felder, 2007). However, participants had known weaknesses with some required programming elements, such as alarms, step events, and loops. Participants also designed game features that required them to learn new content and skills. Proper scaffolding and supervision were applied to help participants complete their projects. Participants

submitted weekly status reports and had several opportunities to revise their work based on instructor and client feedback (Helle et al., 2006).

Guided Instruction

Participants demonstrated difficulty with researching solutions to novel problems and applying skills in unfamiliar contexts. Whole group guided instruction and direct just-in-time instruction were provided when required, which was particularly beneficial for younger learners of nonuniform skills (Hmelo-Silver, 2004). Guided instruction was an efficient method for improving student knowledge quickly (Winarno et al., 2018). Guided instruction was delivered to the entire class based on deficits identified in the content knowledge assessment pretest and problems from previous games. Instruction on selection and iteration statements was provided in 10-minute lessons during the first week of the intervention. When the researcher recognized that several participants were struggling with features related to platforming, a short lesson on implementing platforming elements was delivered. Scaffolding was provided as needed to participants who were struggling. This normally involved directing participants to (a) similar game behavior they had previously implemented or (b) documentation related to the desired functionality. Sometimes the researcher delivered a short personalized lesson or conducted a code trace to explain a defect.

Intervention Phases

The intervention was implemented in three phases:

1. Game design
2. Game implementation
3. Quality engineering

Game Design Phase

The game design phase lasted one week. Requirements for the game project were reviewed. Participants designed a game of their choosing that demonstrated specific programming skills as detailed in the project description in Appendix C. Participants worked with a client to gain the experience of developing a product for another person. Clients were other participants or teachers. The instructor approved the design before participants began detailed documentation and implementation. Participants produced a game design document, which was assessed according to the rubric in Appendix C. The game design document included game rules, room information, asset information, object behavior, and a timeline for deliverables.

Game Implementation Phase

The game implementation phase lasted three weeks. Participants utilized aspects of Agile software development (ASD) to implement their games, but they were not constrained by a formal development process. ASD is characterized by iterative development, regular client collaboration, fast development cycles, and adapting to changing requirements (Beck et al., 2001; Oyong & Ekong, 2019). Participants found that some of their design decisions needed to change. They needed to renegotiate requirements with clients. Participants were required to submit a working version of their game every week, along with a status report. Working versions were free of compiler errors, and working features were free of runtime errors. Features in each version matched the deliverables timeline from the game design document, or participants provided an explanation for discrepancies. Participants found that some game features needed to be removed to meet deadlines. ASD does not focus on documentation, but

participants were required to update their documentation when changes were made. ASD was well-suited to team programming environments (Sakulvirikitkul et al., 2020).

Guided instruction and scaffolding were provided as needed to individual participants and the whole class. Guided instruction was provided to the entire class during week one of this phase, targeted at deficient skills identified by the content knowledge assessment pretest and previous games. Short lessons, 10 minutes maximum, were provided two or three times per week to (a) address algorithm analysis and development problems common to several participants and (b) prepare participants for the post-content knowledge assessment.

Quality Engineering Phase

The quality engineering phase lasted two weeks. Participants documented defects and playability suggestions from their clients on the playtest document (see Appendix C). Participants then revised their games based on client feedback. Participants were required to correct defects or document their existence if they could not be corrected in the remaining time. Participants chose to implement client playability suggestions or not and documented their decisions. At the end of the first week, participants submitted a status report. The first week was a hybrid phase for many participants who were still completing the game implementation phase. At the end of the quality engineering phase, participants submitted the following artifacts as part of their completed project:

- Game Design Document
- GameMaker YYZ file (archive file containing all game logic and assets)
- Playtest Document

Data Collection Methods

Four methods of data collection were employed in an attempt to achieve triangulation, which improved the validity of results and increased understanding of the phenomena under study (Bloomberg & Volpe, 2015; Creswell & Creswell, 2018). The following sources of data were collected in this study: (a) pretest and posttest content knowledge assessments, (b) pre- and post-intervention attitudes toward computer science surveys, (c) classroom observations in the form of field notes, and (d) interviews. Table 3.5 summarizes the alignment between the research questions and data sources.

Table 3.5 Research Question and Data Sources Alignment

Research Question	Data Sources
RQ1: How does the game development project impact participants' ability to analyze and develop algorithms?	<ul style="list-style-type: none">• Content knowledge assessments• Classroom observations• Participant interviews
RQ2: What is the effect of the game development project on participants' attitudes toward computer science?	<ul style="list-style-type: none">• Participant surveys• Classroom observations• Participant interviews
RQ3: What is the relationship between participants' attitudes toward computer science and their performance?	<ul style="list-style-type: none">• Participant surveys• Content knowledge assessments• Participant interviews• Classroom observations

Content Knowledge Assessments

The content knowledge assessment measured the knowledge and skills that participants demonstrated at the time of the assessment. Evidence must be provided to the South Carolina Department of Education that students have learned computer science standards (*Exploring Computer Science*, 2019). When this study was proposed, the content knowledge assessment was considered the most important metric for judging the value of the game development unit. At the conclusion of the study, the researcher was

ambivalent about the relative importance of participants' content knowledge and their attitudes. The prioritization of improving content knowledge or attitudes will be elaborated in the discussion.

A one-group pretest-posttest design was used, in which participants took a pretest at the beginning of the intervention and a posttest at the end of the intervention (Mertler, 2019). The content knowledge assessment was administered in two parts to measure participants' ability to analyze and develop algorithms. Part one of the content knowledge assessment consisted of nine multiple-choice questions. The multiple-choice assessment was administered as a Google Forms quiz, and the results were downloaded in Microsoft Excel. Three aspects of algorithms were measured: (a) sequencing, (b) selection, (c) and iteration. The multiple-choice questions were adapted from practice questions provided by AP Classroom for AP Computer Science Principles (CollegeBoard, 2020b). CollegeBoard validates the ability of their exams to correctly place students into higher-level college courses (Patterson & Ewing, 2013). The programming language was changed from pseudocode used on the AP Computer Science Principles exam to GameMaker Language. Table 3.6 summarizes the knowledge assessed by each question. The content knowledge assessment was reviewed by a colleague in the technology department who teaches Fundamentals of Computing to ensure content validity.

Table 3.6 Content Knowledge Assessed by Question

Question Numbers	Content Knowledge
1 – 3	Sequencing
4 – 6	Selection
7 – 9	Iteration

Part two of the content knowledge assessment was a performance task that measured participants' ability to implement algorithms given a set of game behavior requirements. The programming language used in the assessment was GameMaker Language. Participants used GameMaker Studio 2 in the performance task. Participants were using GameMaker Studio 2 and GameMaker Language for over a semester, so they were familiar with the syntax and IDE. Appendix D contains the content knowledge assessment and rubric for the performance task.

The performance tasks were scored by playtesting participants' submissions. Scores for the rubric items were recorded in Microsoft Excel. A colleague independently scored half of the performance tasks that were randomly selected. Inter-rater reliability was calculated to ensure validity.

The content knowledge assessment was administered as a pre and posttest. The pretest was administered prior to the intervention, and the posttest was administered at the conclusion of the intervention. Participants had one hour to complete the multiple-choice portion and one hour to complete the performance task. Because the class periods were one hour, the multiple-choice portion was completed in one class meeting, and the performance task was completed in the following class meeting. The multiple-choice portion was worth nine points, with each question worth one point. The performance task was worth fifteen points, with points awarded as detailed in the rubric. Participants spent most of their time in the course and the intervention developing games and writing algorithms to satisfy requirements. Therefore, the performance task was weighted more

than the multiple-choice because it directly matched the skills participants had been practicing.

The course standards assessed were detailed in the Fundamentals of Computing course standards document (*Computer Science Discoveries ('19- '20)*, 2019). The following standards from section H, *problem solving and computational thinking*, were assessed:

1. Solve a problem by applying appropriate problem solving techniques (understand the problem, plan the solution, carry out the plan, review and discuss).
2. Demonstrate an understanding of algorithms and their practical applications.
3. Create, evaluate, and adjust algorithms to solve a variety of problems.

The following standards from section I, *fundamentals of programming*, were assessed:

1. Analyze and explain how a particular program functions.
2. Write code that uses variables, events, functions, operators (i.e. arithmetic, relational, logical), conditional control structures (e.g., if, if-else) and repetition/iteration control structures (e.g., while, for).
3. Edit, compile/run, test, and debug a program.

Table 3.7 shows where the course standard was assessed in the content knowledge assessment.

Table 3.7 Course Standard Assessed by Content Knowledge Assessment

Course Standard	Content Knowledge Assessment
Solve a problem by applying appropriate problem solving techniques (understand the problem, plan the solution, carry out the plan, review and discuss).	<ul style="list-style-type: none"> • Multiple-choice • Performance task
Demonstrate an understanding of algorithms and their practical applications.	<ul style="list-style-type: none"> • Multiple-choice • Performance task

Course Standard	Content Knowledge Assessment
Create, evaluate, and adjust algorithms to solve a variety of problems.	<ul style="list-style-type: none"> • Multiple-choice • Performance task
Analyze and explain how a particular program functions.	<ul style="list-style-type: none"> • Multiple-choice
Write code that uses variables, events, functions, operators (i.e. arithmetic, relational, logical), conditional control structures (e.g., if, if-else) and repetition/iteration control structures (e.g., while, for).	<ul style="list-style-type: none"> • Performance task
Edit, compile/run, test, and debug a program.	<ul style="list-style-type: none"> • Performance task

Student Surveys

Attitudes toward computer science surveys provided quantitative data about participants' attitudes necessary to answer research questions two and three (Shen et al., 2014). Participants completed the survey before the intervention and following the intervention. The surveys were administered on Google Forms and downloaded to Microsoft Excel. Participants expressed their degree of agreement with 26 question statements using a 5-point Likert scale. The five choices for their degree of agreement were: (a) strongly disagree, (b) disagree, (c) neutral, (d) agree, and (e) strongly agree. Attitudes toward computer science were classified into five subscales: (a) self-concept in computer science, (b) learning computer science at school, (c) learning computer science outside of school, (d) future participation in computer science, and (e) importance of computer science. To measure the internal consistency of the survey, Shen et al. (2014) calculated Cronbach's alpha coefficients for each aspect in two implementations. All subscales had good reliability, $\alpha > .80$ (George & Mallery, 2002; Taber, 2018).

Subscales

All survey items are in Appendix E.

Self-Concept in Computer Science. The self-concept in computer science subscale measured participants' perception of content mastery and enjoyment of computer science.

There were five statements in this subscale. Examples of statements were

- Computer science is fun.
- I feel at ease with computer science, and I understand concepts easily.

Learning Computer Science at School. The learning computer science at school subscale measured participants' enjoyment of their computer science course while in class. There were five statements in this subscale. Examples of statements were

- We learn interesting things in computer science lessons.
- I look forward to my computer science lessons.

Learning Computer Science Outside of School. The learning computer science at school subscale measured participants' enjoyment of their computer science course while outside of class. There were six statements in this subscale. Examples of statements were

- I would like to join a computer science club.
- I would like to do more computer science activities outside school.

Future Participation in Computer Science. The future participation in computer science subscale measured student's plans or desires to continue studying or working with computer science. There were five statements in this subscale. Examples of statements were

- I would like to study more computer science in the future.
- I would like to have a job working with computer science.

Importance of Computer Science. The importance of computer science subscale measured participants' perception of the impact of computer science on society. There were five statements in this subscale. Examples of statements were

- Computer science and technology are important for society.
- Computer science and technology make our lives easier and more comfortable.

Classroom Observations

Semistructured observations were conducted to gather information that participants were uncomfortable discussing or did not remember (Creswell & Creswell, 2018). The researcher fielded questions and assisted participants, which prohibited structured observations. Actual student behavior was recorded, which provided data that would be impossible to gather in another way (Mertler, 2019).

Field notes were used to collect observations. The researcher was looking for specific behaviors and attitudes that aligned with the research questions. Student attention was observed by documenting the frequency and duration of off-task behavior. The amount of effort that participants displayed while attempting to solve problems and meet deadlines was also observed. Some observation time was devoted to documenting everything that was seen, which allowed patterns to emerge organically (Mertler, 2019).

Field notes were divided into three columns: (a) observation number, date, and time; (b) observations; and (c) observer's comments to add interpretations of observations (Mertler, 2019). As many participants as possible were observed. Field notes were hand-written in a composition notebook and transcribed into Microsoft Word.

Participant Interviews

Participant interviews served two purposes. First, participants provided information that was missed or could not be gathered with observations; second, participants confirmed or disconfirmed observations of the researcher (Creswell & Creswell, 2018). The interviews also provided context to the student surveys and suggested information that should be included in future surveys. Interviews were an opportunity for participants to directly inject their views into the study (Tracy, 2020).

Fourteen participants returned the assent forms to be interviewed, and 13 participants scheduled interviews. Twelve participants were interviewed; one was not interviewed due to scheduling problems. Saturation was reached, which occurred when new information added little to existing findings (Tracy, 2020). One possible exception to saturation was that only one participant was highly critical of the intervention and reported a net negative experience. This will be discussed further in the limitations.

The interviews lasted approximately 30 minutes each. Interviews were conducted virtually over Google Meet on the weekend when possible. Other interviews were conducted during class time in the hallway outside of the classroom when participants were unable to meet outside of class meeting time. The interviews were conducted after the content knowledge posttest. The interview was audio-recorded and transcribed by Otter.ai. The interview transcripts were cleaned and downloaded to Microsoft Word. The interview questions were aligned to the research questions and survey.

The interview was semistructured, which allowed flexibility with questions and probes (Mertler, 2019; Tracy, 2020). The interview was not overly formal, which encouraged participants to relax and answer freely. See Appendix F for the interview

protocol. Table 3.8 summarizes the alignment between the interview questions and the research questions.

Table 3.8 Alignment of Interview Questions to Research Questions

Research Question	Interview Question
How does the game development project impact participants' ability to analyze and develop algorithms?	<ol style="list-style-type: none"> 1. Can you describe what you learned in this unit? Please include what you think you were expected to learn and what you actually learned. Did the assessment provide an accurate measure of what you know for each skill? 2. Describe how effective the game development project has been in helping you learn in our course. 3. How did the game development project help you learn to analyze and develop algorithms? Can you give me an example?
What is the effect of the game development project on participants' attitudes toward computer science?	<ol style="list-style-type: none"> 4. Describe any programming or game development skills that improved during the project. 5. Can you recall any instances when you enjoyed developing your game? 6. Describe how you generally feel when you come to class. How does that compare with your other courses? 7. Describe how your interest in computer science has changed outside of school. 8. Tell me about any plans you have to study or work with computer science in the future. 9. What is the most beneficial effect of computer science and technology on society? Why? 10. What is the most harmful effect of computer science and technology on society? Why?
What is the relationship between participants' attitudes toward computer science and their performance?	<ol style="list-style-type: none"> 11. In what ways do your attitudes toward computer science affect your performance in the course? 12. Would you please describe any attitudes or feelings that may have affected your ability to learn in the computer science course? 13. Describe your reactions to errors and setbacks in the game you developed. Include how you felt during the troubleshooting process.

Data Analysis

This mixed-methods study included qualitative and quantitative analysis. Table 3.9 summarizes the data analysis performed for each data source.

Table 3.9 Research Question, Data Sources, and Data Analysis Alignment

Research Question	Data Sources	Data Analysis
RQ1: How does the game development project impact participants' ability to analyze and develop algorithms?	<ul style="list-style-type: none"> • Content knowledge assessments • Classroom observations • Participant interviews 	<ul style="list-style-type: none"> • Descriptive statistics and paired samples <i>t</i>-test on pretest and posttest scores
RQ2: What is the effect of the game development project on participants' attitudes toward computer science?	<ul style="list-style-type: none"> • Participant surveys • Classroom observations • Participant interviews 	<ul style="list-style-type: none"> • Descriptive statistics and paired samples <i>t</i>-test or Wilcoxon signed-rank test on each survey subscale • Inductive and deductive analysis using interview and observation data
RQ3: What is the relationship between participants' attitudes toward computer science and their performance?	<ul style="list-style-type: none"> • Participant surveys • Content knowledge assessments • Participant interviews • Classroom observations 	<ul style="list-style-type: none"> • Pearson's <i>r</i> on composite post-survey and posttest scores • Inductive and deductive analysis using interview and observation data

Quantitative Analysis

The quantitative data was formatted in Microsoft Excel. JASP was used for data analysis. Descriptive statistics were reported for all quantitative data.

Content Knowledge Assessments

Data were uploaded to JASP for descriptive and inferential statistics. The bivariate normality was confirmed with the Shapiro-Wilk test. A paired samples *t*-test with an alpha value of .05 was run on the pretest and posttest scores to answer research question one (Adams & Lawrence, 2019).

Student Surveys

Data were uploaded to JASP for descriptive and inferential statistics. The bivariate normality was confirmed for each subscale with the Shapiro-Wilk test. A paired samples *t*-test (if normality was confirmed) or Wilcoxon signed-rank test (if normality was not confirmed) was run on each subscale to help answer research question two (Adams & Lawrence, 2019). An alpha value of .05 was used to determine the significance of a single test. Because five tests were performed simultaneously, a Bonferroni correction was used by dividing the alpha value by the number of tests, resulting in an adjusted alpha value of .01. The Bonferroni correction controls the family-wise error rate, which controls the probability of making at least one false discovery (Glickman et al., 2014; Perneger, 1998). Bonferroni is a very conservative correction for multiple comparisons, which increases the likelihood of type II errors (Glickman et al., 2014; Perneger, 1998).

Relationship of Attitudes and Performance

The post-content knowledge assessment data and post-survey data were added to Microsoft Excel. Data were uploaded to JASP for descriptive and inferential statistics. Composite post-survey scores were calculated for each participant using an unweighted average of the subscale scores. Pearson's *r* was calculated with the composite post-

survey and the posttest scores to help answer research question three. The following assumption checks were performed:

- The bivariate normality was confirmed with the Shapiro-Wilk test.
- Homoscedasticity was confirmed by examining a scatterplot of the data and verifying uniform variance along the line of best fit.

Qualitative Analysis

This study analyzed two qualitative data sources, including observation field notes and interview transcripts. Microsoft Word documents containing the field notes and interview transcripts were uploaded to Delve for analysis. The observation field notes were not verbatim text; they contained abbreviated descriptions of observed behavior with detailed interpretations of the researcher. The interview transcripts were verbatim text, some with rich narratives added (Bernard et al., 2017). Inductive and deductive analysis were used to winnow and organize data to identify critical patterns and themes (Creswell & Creswell, 2018; Fereday & Muir-Cochrane, 2006; Mertler, 2019). Strategies and steps were not performed linearly. Some steps occurred simultaneously, and some steps were repeated. The inductive and deductive analysis process consisted of: (a) transcribing, (b) memoing, (c) identifying noteworthy quotes, (d) mapping similar codes and emerging concepts, (e) drafting recurring aspects of the data, and (f) developing a theme by interpreting the data (Creswell, 2017; Saldaña, 2021).

Memoing took place in the observation field notes and interview transcripts. Coding categorized the raw data into groups of similar data (Saldaña & Omasta, 2017). The unit of analysis was a complete thought. For the first round of coding, initial coding was used for the observations and interviews to avoid forcing a framework on the data

analysis (Creswell, 2017; Saldaña, 2021). Codes were modified as the data were analyzed and similarities discovered. Inductive and deductive analysis was performed to iteratively group similar codes into categories and similar categories into more general categories until themes emerged (Creswell, 2017; Fereday & Muir-Cochrane, 2006; Saldaña, 2021; Saldaña & Omasta, 2017).

Findings for qualitative data included narrative text through themes, a table display with assertions, evidence, and descriptive narratives. Similarities and dissimilarities with the quantitative findings were explored. Similarities helped validate the quantitative data, while dissimilarities suggested problems such as invalid measurement or confounding factors (Randolph, 2008).

Procedures

The intervention for this study took six weeks in phases one, two, and three, as described in Table 3.10. Data collection occurred at the end of phase zero and continued through phase four. During phase zero, approximately five weeks during January and February were lost to rolling COVID quarantines. All participant artifacts were submitted to the instructor in Google Classroom. Each phase and the associated tasks are described below.

Table 3.10 Timeline of Phases and Tasks

Phase	Tasks	Duration in weeks
Phase 0: pre-intervention	<ol style="list-style-type: none"> 1. Direct instruction 2. Programming in Khan Academy 3. Four games and one cutscene 4. Study description 5. Distribute consent and assent forms 	24

Phase	Tasks	Duration in weeks
Phase 1: study introduction and game design	6. Review of content knowledge assessment by colleague	1
	7. Content knowledge pretest	
	8. Participant pre-surveys	
	1. Review requirements for game project	
	2. Assign clients	
	3. Game design document creation	
Phase 2: game implementation	4. Instructor and client review of game design document	3
	5. Revision of game design document	
	6. Field notes	
	1. Guided instruction	
	2. Iterative game development	
	3. Weekly progress reports	
Phase 3: quality engineering	4. Field notes	2
	1. Client playtesting	
	2. Playability revision	
Phase 4: post-intervention data collection	3. Field notes	2
	1. Content knowledge posttest	
	2. Participant post-surveys	
Phase 5: data analysis	3. Participant interviews	4
	1. Member checking	
	2. Inter-rater reliability for content knowledge assessment performance task	
	3. Descriptive statistics and paired samples <i>t</i> -test on pretest and posttest scores	
	4. Descriptive statistics and paired samples <i>t</i> -test on each survey subscale	

Phase	Tasks	Duration in weeks
	5. Pearson's r on composite post-survey and posttest scores	
	6. Inductive and deductive analysis using interview and observation data	
	7. Peer debriefing sessions on qualitative analysis	
	8. Audit trail	

Phase 0: Pre-Intervention

During the first semester and start of second semester, students received guided instruction on basic programming skills:

- sequencing
- selection
- iteration
- user-defined functions
- arrays

JavaScript was used for the initial guided instruction. Participants completed small programming assignments for practice in Khan Academy using JavaScript. The Khan Academy course was called Computer programming, and the unit was Intro to JS: Drawing & Animation. The following Khan Academy modules were completed in the Intro to JS Drawing & Animation unit: (a) Intro to programming, (b) Drawing basics, (c) Coloring, (d) Variables, (e) Animation basics, (f) Interactive programs, (g) Becoming a community coder, (h) Bonus: Resizing with variables, (i) Text and strings, and (j) Functions. Participants implemented four games and one cutscene using GameMaker Studio 2 and the Zulama curriculum: (a) Pinball, (b) Ball Bouncer, (c) Matching, (d) Scat,

and (e) Sky is Falling cutscene (Carnegie Learning, 2021). Once participants started using GameMaker, guided instruction was delivered in GameMaker Language. Participants requested their partners for the Matching game and worked with those partners from the start of that game.

Before beginning the intervention, permission to conduct the study was obtained from the University of South Carolina Institutional Review Board (IRB) and South Carolina School District Alpha (see Appendix G and Appendix H). Following the completion of the Sky is Falling cutscene in semester two, the study was described to participants. All participants were invited to participate. Consent and assent forms were distributed to participants and their parents. Participants were informed that those who did not consent to the study within two weeks would not have their data included in the study; they would not participate in the post-survey or interviews. All participants returned the signed consent forms. The assent forms were used for the interviews. Fourteen participants returned the signed assent forms. Participants took the content knowledge pretest over two days. The multiple-choice section was administered on the first day, followed by the performance task on the second day. The multiple-choice section was administered on a Google Form quiz, and participants had one hour to complete it. Participants completed the performance task in GameMaker and submitted a YYZ file; they had one hour to complete the performance task. The pre-survey was administered the day after the content knowledge assessment on a Google Form. The content knowledge assessment and the survey were administered in the normal classroom setting during regularly scheduled class time.

Phase 1: Study Introduction and Game Design

Phase one lasted one week. On day one, the requirements for the game project were reviewed with the participants. Participants read the project directions and rubrics. The instructor reviewed the requirements for the game design document and client interaction in detail. On day two, participants requested clients, which were approved by the instructor. Clients were other participants in the study. In one case, the client was a music teacher who wanted a serious game. Accommodations were made for participants who found other acceptable clients. Finally, participants created a game design document with their partners and clients according to the project directions and game design document rubric in Appendix C. Two days were scheduled for the process of negotiating requirements with the client and creating the game design document. On the fourth day, participants submitted their game design documents to the instructor for review. On day five, the instructor returned recommendations to the participants. Participants made revisions and resubmitted their game design documents on day five. See Figure 3.4 for an abridged version of the game design document submitted by Aleesha. Field notes were recorded to generate qualitative data as participants developed their game design documents.

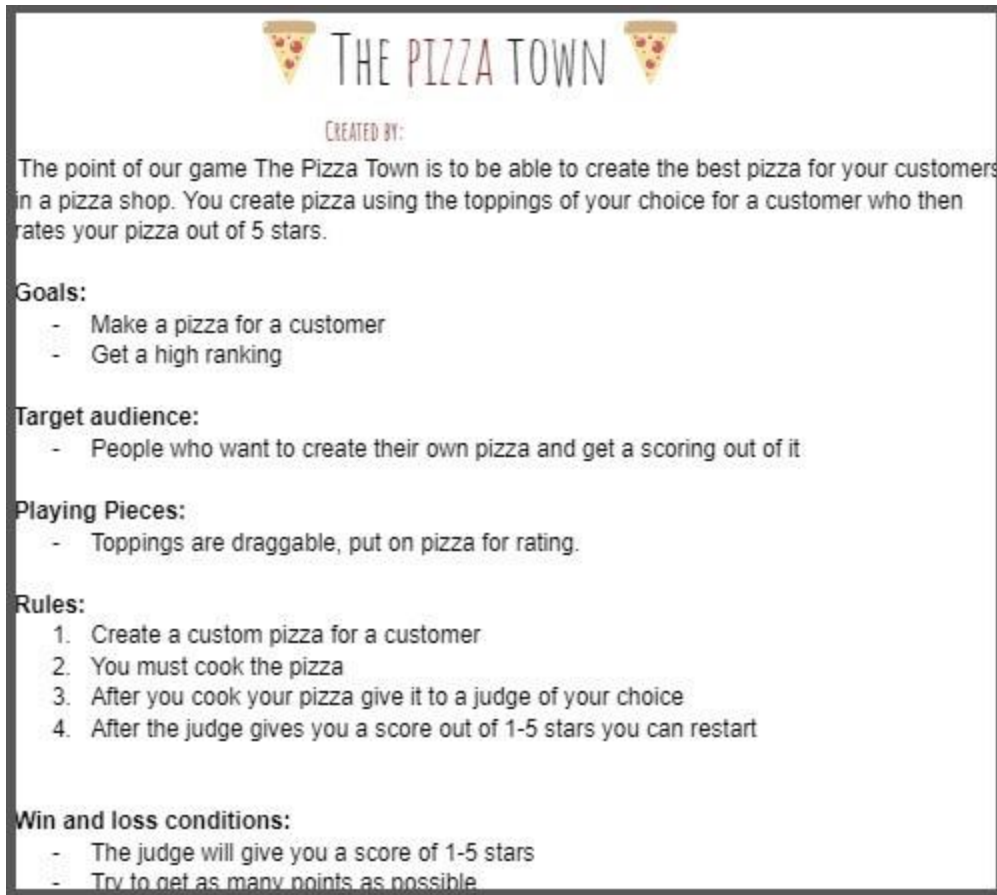


Figure 3.4 Abridged Game Design Document from Aleesha

Phase 2: Game Development

Phase two lasted three weeks. Guided instruction with a spiral approach was used to reinforce concepts that the participants needed to demonstrate in their games.

Participants were not expected to utilize any programming skills that were not covered in phase zero. However, as discussed earlier, deficits in participants' knowledge were identified by the researcher. The content knowledge assessment revealed skills that needed review. Those skills were demonstrated to the participants in week one of phase two. Twenty minutes of each hour during week one was budgeted for guided instruction, but 10 minutes was the maximum used for whole group guided instruction. Participants used Agile software development to create their games.

One participant from each group created a cloud directory on Google Drive to store project assets. Edit access to the directory was provided to the other participant in the group and the researcher. The game design document, status reports, playtest documents, and GameMaker project backups were stored in the shared project directory on Google Drive. Participants were required to save a functional (no compiler errors) version of their game daily by exporting their projects as YYZ files and saving them to Google Drive. This backup process protected participants from losing their entire project due to three causes that the instructor had witnessed in previous courses: (a) malfunctioning hardware, (b) corrupted game, or (c) the district office of computing services reimaging machines without warning if suspected malware was detected. A corrupted game usually resulted from students unintentionally adding or deleting files from the GameMaker project directory. A few participants experienced corrupted projects, but they were able to restore projects from their backups. Therefore, no participant lost more than one day of work due to a corrupted project.

Participants submitted status reports at the end of each week during phase two. The three main sections of the status report were: (a) what was accomplished, (b) what is left to do, and (c) questions and challenges. See Figure 3.5 for an example of a status report from Julia. See Appendix C for a full description of the status report. Field notes were recorded to generate qualitative data as participants developed their games.

Week 3 (4/11/22 - 4/14/22)

DONE (what did you accomplish this week?)

- We created the win and lose banner
- Added the score
- Made a score limit that took you to the end screen

TO DO (what do you plan to accomplish next week?)

- Fix win and lose banner (associate with score)

QUESTIONS/CHALLENGES

- We are still struggling with the banners

Figure 3.5 Status Report from Julia

Phase 3: Quality Engineering

Phase three lasted two weeks. The first week blended with phase two, as some participants used a few days to finish development. Participants had their clients, including at least one peer group playtest their games. Most participants had several other groups playtest their game. Five was the recommended number of playtesters. Based on playtest feedback, participants revised their games to remove previously undetected defects and improve playability. Participants submitted a status report at the end of week one. At the end of week two, participants submitted final versions of the following:

- game design document
- game as YYZ file
- playtest document

See Figure 3.6 for an abridged playtest document from Bree. Field notes were recorded to generate qualitative data as participants test their games.

	Before: Preparation	During: Observing & Recording	After: Reflecting
Playability / Fun	Theres 2 players	5	Fixing the rotten fruit makes more fruit fall.
Timing	2 minutes	2 minutes	We might add a round 2 to increase the time
Player Reactions	Excitement	Excited and entertained	Make it a little difficulter

Figure 3.6 Abridged Playtest Document from Bree

Phase 4: Post Intervention Data Collection

Phase four lasted five weeks. Participants took the content knowledge posttest over two days. The multiple-choice section was be administered on the first day, followed by the performance task on the second day. The multiple-choice section was administered on a Google Form quiz, and participants had one hour to complete it. Participants completed the performance task in GameMaker and submitted a YYZ file; they had one hour to complete the performance task. The attitudes toward computer science post-survey was administered the day after the content knowledge assessment on a Google Form. The content knowledge assessment and the survey were administered in the normal classroom setting during regularly scheduled class time.

Twelve participant interviews took place over five weeks. Each interview was scheduled for 30 minutes. When possible, the interviews were conducted virtually on Google Meet on the weekend. Other interviews took place in the hallway outside of the class during regularly scheduled class time to accommodate participants who could not meet virtually on the weekend. Automatic transcription was performed using Otter.ai. See Appendix F for the interview protocol.

Phase 5: Data Analysis

Phase five lasted about 13 weeks. Data from Google Forms was exported to Google Sheets and converted to Microsoft Excel. Data in Microsoft Excel was formatted so that the data could be imported to JASP. JASP was used for the statistical analysis, including:

- descriptive statistics of the content knowledge pretest and posttest scores
- assumption and reliability checks on the content knowledge assessment data
- paired samples *t*-tests on the content knowledge pretest and posttest scores
- descriptive statistics of the pre- and post-survey subscale scores
- reliability checks on the survey subscales
- paired samples *t*-tests and Wilcoxon signed-rank tests on the pre- and post-survey subscale scores
- assumption checks for the correlation statistics on composite post-survey and posttest scores
- linear correlation of composite post-survey and posttest scores

Delve and Microsoft Excel were used for the inductive and deductive analysis of the field notes and interview transcripts.

Rigor & Trustworthiness

Validity and reliability are measures of rigor and trustworthiness in quantitative designs; qualitative designs use other methods (Creswell & Creswell, 2018). The following methods will be used to establish rigor and trustworthiness in the qualitative data collection and analysis of this study: (a) triangulation, (b) member checking, (c) peer debriefing, (d) an audit trail, and (e) inclusion of negative or discrepant data.

Triangulation

Triangulation involves examining evidence from multiple data sources and generating findings from the convergence of the data sources (Creswell & Creswell, 2018). Individual data collection methods suffer from methodological shortcomings, but using multiple methods together compensates for the shortcomings (Shenton, 2004). This mixed-methods study employed multiple qualitative and quantitative measures: (a) observations, (b) participant interviews, (c) participant surveys, and (d) pretest and posttest content knowledge assessments. “Triangulation is an inherent component of mixed-methods research designs” (Mertler, 2019, p. 142). When several sources of data confirmed a finding, validity was increased for that finding. When sources of data did not lead to the same finding, a problem with methodology or interpretation of results may have occurred.

Member Checking

Member checking is used to confirm the accuracy of qualitative findings with participants of the study (Creswell & Creswell, 2018). In this study, interview transcripts, observations, and themes were shared with participants, which allowed participants to review how they were represented in the study. The abstract was shared

with participants so that they could review the broad findings of the study.

Trustworthiness was increased by allowing participants to audit the pertinent elements of the study (Shenton, 2004).

Peer Debriefing

Peer debriefing involves other professionals or colleagues reviewing the data collection and analysis in the study (Mertler, 2019). Meetings were held with the dissertation major professor, Dr. Arslan-Ari, to verify the rigor of the study. Several improvements to qualitative coding were made during peer-debriefing sessions with Dr. Arslan-Ari: (a) more detail was added to first cycle codes, (b) ambiguous symbols and abbreviations that may have confused reviewers were removed from codes, (c) misleading and ambiguous phrasing in code names was improved, and (d) improperly categorized codes were recategorized. Dr. Arslan-Ari, other professors, colleagues, and participants reviewed various elements of the study throughout the research process to increase rigor.

Audit Trail

The audit trail is used to record the researcher's development of interpretations as data is collected (Shenton, 2004). Notes were taken to document how codes, patterns, categories, and themes were generated from observations, surveys, and interviews. Details were recorded regarding decisions on categorizing and grouping data. Multiple exports from Delve to Microsoft Excel were produced to track the progression of codes, categories, and themes.

Negative or Discrepant Information

Negative or discrepant information includes data that does not support findings in the study (Creswell & Creswell, 2018). When possible, negative or discrepant information was used to revise findings so that the information no longer ran counter to the findings. When findings could not be modified to incorporate negative information, the negative information was reported with the findings to increase trustworthiness in the interpretations. Negative information will be reviewed in the discussion.

Plan for Sharing and Communicating Findings

The findings were shared with the following members of administration: (a) administrators of the school in which the study was conducted, (b) the assistant superintendent for secondary education, (c) the director of the technology center, and (d) other district personnel who request access. I plan to share the findings with other computer science teachers in the district during a department meeting on an inservice day. My findings will be shared with my dissertation committee. Finally, the findings will be shared with the participants, who will be able to review data and conclusions before other stakeholders. The feedback obtained from stakeholders will be used to improve instructional strategies used in teaching programming. Participants were able to withdraw their data from the study at any time in the research process (Banister, 2007). The participants understood their influence on future iterations of the action research. I applied to present my research at the Future of Education Technology Conference (FetC) in January 2023. My application was accepted, and I plan to present my finding pending the approval of my district.

Participants' confidentiality was respected and protected. Data was anonymized; pseudonyms were used for student names, the district, and schools. Participants had the option of selecting their pseudonyms. Monster Fan and Pibb chose their pseudonyms. A random name generator was used to assign the remaining pseudonyms (Campbell, 2021). Pseudonyms were originally selected from a list of Egyptian and Greek gods, but those names were judged to be distracting. Appendix I displays the map of original pseudonyms to current pseudonyms. Raw data was stored in a separate location from student identification, and the data will be destroyed after five years in order to protect participants' identity and confidentiality. Participants' identification was stored in a password-protected spreadsheet.

CHAPTER 4: ANALYSIS AND FINDINGS

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha. This study was expected to provide insight into the impact of PjBL and GDBL in an introductory CS course. The collection of data for this study was guided by three research questions: (1) How does the game development project impact participants' ability to analyze and develop algorithms? (2) What is the effect of the game development project on participants' attitudes toward computer science? and (3) What is the relationship between participants' attitudes toward computer science and their performance? This chapter presents the analysis and findings of the collected data, including pre and post-content knowledge assessments, pre and post-computer science attitude surveys, field notes from classroom observations, and participant interviews. This chapter will include quantitative and qualitative analysis.

Quantitative Findings

The purpose of this quantitative analysis was to measure the effects of a digital game development project on participants' performance and attitudes toward CS. Two quantitative data sources were used to answer the research questions: (a) a content knowledge assessment and (b) an attitudes toward computer science survey. A one-group pretest-posttest design was used (Creswell & Creswell, 2018). A content knowledge assessment was administered as a Google Forms quiz before and after the

intervention to answer research question one. For research question two, an attitudes toward computer science survey was administered on Google Forms before and after the intervention (Shen et al., 2014). For research question three, the correlation between participants' attitudes toward computer science and their performance was calculated using the post-intervention content knowledge assessment and the post-intervention attitudes toward computer science survey. Google Forms data was downloaded in Microsoft Excel. Data was organized in Microsoft Excel and exported to JASP to calculate descriptive and inferential statistics. No outliers were identified for removal from the final analysis. There was no missing data. Effect sizes were interpreted based on benchmarks proposed by Cohen (1988). See Table 4.1 for the interpretation of effect sizes. Cohen's d and rank-biserial correlation were used as measures of effect size for parametric and non-parametric data, respectively (Cohen, 1988; King et al., 2018; Lakens, 2013; Sawilowsky, 2009).

Table 4.1 Interpretation of Effect Size

Interpretation	Minimum d or r_b
small	0.2
medium	0.5
large	0.8

Content Knowledge Pre and Post-Assessments Results

The content knowledge assessment consisted of a multiple-choice test worth nine points and a performance task worth 15 points. The multiple-choice test consisted of nine questions; three questions each assessed: (a) sequencing, (b) selection, and (c) iteration. The performance task required participants to implement a set of behaviors in GameMaker using a set of requirements and a grading rubric. Participants' multiple-

choice scores and performance task scores were summed to create a total content knowledge assessment score. Strategies used to ensure reliability and content validity are discussed below.

Interrater Reliability

Interrater reliability is the extent to which multiple assessors agree on the evaluation of the same target (Adams & Lawrence, 2019). Interrater reliability was calculated to improve the reliability of the data analysis of the performance task. The performance task was scored according to the rubric (Appendix C). To establish interrater reliability, the chair of the business education and CS department, the interrater, scored 14 of the 28 performance tasks. The interrater was given all 28 performance tasks and randomly selected 14 of them to score. As shown in Table 4.2, the percent agreement between the researcher and the interrater was calculated for each of the eight rubric items, also referred to as rows. The total score for the performance task was calculated by summing the eight row scores. Finally, Cohen's weighted kappa was calculated for the total score. There was almost perfect agreement between the two graders' total scores, $\kappa = .95$, 95% CI [.79, 1], $n = 14$ (McHugh, 2012).

Table 4.2 Performance Task Percent Agreement

Rubric Row	Exact Match	Percent Agreement
1	13	93
2	13	93
3	13	93
4	12	86
5	11	79
6	13	93
7	13	93

Rubric Row	Exact Match	Percent Agreement
8	12	86

Note. $n = 14$ for each rubric row.

Internal Consistency

Kuder-Richardson Formula 20 (*KR20*) was calculated for the multiple-choice portion of the content knowledge pretest and posttest. *KR20* for the pretest was .53, and *KR20* for the posttest was .18. The quantitative claims made in this study regarding research questions one and two should be interpreted in the context of the low *KR20* scores; note that some research considers *KR20* of .50 acceptable, but most considers .70 or above as the minimum acceptable *KR20* (Anselmi et al., 2019; Ebel, 1967; Mitchell et al., 2018; Osadebe, 2015). Low variance and high item difficulty may have affected *KR20*. Nine points were available on the multiple-choice assessments, one point for each test item. The mean multiple-choice pretest score was 3.75 ($\sigma^2 = 3.33$), and the mean multiple-choice posttest score was 6.14 ($\sigma^2 = 2.19$). Table 4.3 shows the number of correct answers for each test item. *KR20* will be discussed further in the discussion of limitations.

Table 4.3 Correct Responses Per Item on Multiple-Choice Assessment

Test Item	Multiple-Choice	
	Pretest	Posttest
1	20	21
2	10	19
3	17	24
4	9	19
5	10	18
6	22	23
7	3	16

Test Item	Multiple-Choice	
	Pretest	Posttest
8	6	18
9	8	14

Note. $N = 28$. The Multiple-Choice columns represent how many participants answered each question correctly.

Descriptive Statistics

Table 4.4 summarizes the mean and standard deviation of the pretest and posttest. The mean pretest score was 7.93 ($SD = 3.43$), and the mean posttest score was 16.14 ($SD = 5.25$). None of the 28 participants scored lower on the posttest than on the pretest. Qianna's pretest score was the same as the posttest score, and she was the only participant to exhibit zero improvement on the assessment.

Table 4.4 Descriptive Statistics of the Pretest and Posttest

Assessment	M	SD
Pretest	7.93	3.43
Posttest	16.14	5.25

Note. $N = 28$.

Inferential Statistics

To test the assumption of normality, the Shapiro-Wilk test was conducted to determine whether the differences in pretest and posttest could have been produced by a normal distribution (Razali & Wah, 2011). The results of the test were not significant, $W(27) = .95$, $p = .209$, indicating that the assumption of normality was met.

To address the first research question, a paired samples t -test was conducted to compare the total content knowledge assessment scores before and after the intervention. An alpha level of .05 was used. When calculating the test statistics, the alternative hypotheses specified that the posttest mean was greater than the pretest mean. The

results of the paired samples *t*-test indicated that the difference between the posttest scores ($M = 16.14$, $SD = 5.25$) and pretest scores ($M = 7.93$, $SD = 3.43$) was significant, $t(27) = 9.12$, $p < .001$. The mean difference was 8.21, and the effect size was large, $d = 1.72$. See table 4.5 for the results of the paired samples *t*-test.

Table 4.5 Paired Samples t-Test Results for Content Knowledge Assessment Scores

Pretest		Posttest		<i>t</i>	<i>p</i>	<i>d</i>
<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>			
16.14	5.25	7.93	3.43	9.12	<.001	1.72

Note. $N = 28$.

Attitudes Survey Results

The survey contained 26 questions divided into five subscales; each subscale had five or six questions. Questions used a 5-point Likert scale ranging from 1 (*strongly disagree*) to 5 (*strongly agree*). The survey subscales and associated items are shown in Table 4.6.

Table 4.6 Survey Subscales

Subscale Description	Items
Self-concept in computer science	1-5
Learning computer science at school	6-10
Learning computer science outside of school	11-16
Future participation in computer science	17-21
Importance of computer science	22-26

Reliability. A Cronbach alpha coefficient was calculated for each subscale. The Cronbach's alpha coefficient was evaluated using the guidelines suggested by George and Mallery (2002), where $> .9$ was excellent, $> .8$ was good, $> .7$ was acceptable, $> .6$ was questionable, $> .5$ was poor, and $\leq .5$ was unacceptable.. As shown in Table 4.7,

Cronbach's alpha was calculated for each subscale. All subscales had acceptable reliability, $\alpha > .70$ (George & Mallery, 2002; Taber, 2018).

Table 4.7 Internal Consistency Measure of Survey (Researcher)

Subscale Description	Cronbach's α	
	Pre	Post
Self-concept in computer science	.92	.91
Learning computer science at school	.87	.93
Learning computer science outside of school	.86	.93
Future participation in computer science	.87	.92
Importance of computer science	.72	.85
All Items	.95	.97

Note. $N = 28$. Pre = survey before intervention; Post = survey after intervention.

Descriptive Statistics

Table 4.8 summarizes the descriptive statistics for the pre- and post-computer CS attitude surveys. The mean of every subscale increased from pre- to post-survey. The importance of CS subscale was high in the pre-survey relative to the other subscales. The subscales with the largest mean differences were self-concept in CS and learning CS at school.

Table 4.8 Descriptive Statistics of Pre and Post-CS Attitude Survey Subscales

Subscale	Pre		Post	
	<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>
Self-concept in CS	1.90	0.92	3.01	1.07
Learning CS at school	1.86	0.88	2.79	1.15
Learning CS outside of school	1.55	0.72	2.21	1.11
Future participation in CS	1.64	0.86	2.34	1.22

Subscale	Pre		Post	
	<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>
Importance of CS	3.25	0.85	3.70	0.98

Note. $N = 28$.

Inferential Statistics

To test the assumption of normality, the Shapiro-Wilk test was conducted on each subscale to determine whether the differences in pre and post-survey subscales could have been produced by a normal distribution (Razali & Wah, 2011). For self-concept in CS, the results of the test were not significant, $W(27) = .95, p = .244$, indicating that the assumption of normality was met. For learning CS at school, the results of the test were not significant, $W(27) = .96, p = .425$, indicating that the assumption of normality was met. For learning CS outside of school, the results of the test were significant, $W(27) = .88, p = .005$, indicating that the assumption of normality was not met. For future participation in CS, the results of the test were significant, $W(27) = .82, p < .001$, indicating that the assumption of normality was not met. For importance of CS, the results of the test were not significant, $W(27) = .95, p = .229$, indicating that the assumption of normality was met. See Table 4.9 for a summary of the subscale assumption checks.

Table 4.9 Survey Subscale Assumption Checks

Subscale	<i>W</i>	<i>p</i>
Self-concept in CS	.95	.244
Learning CS at school	.96	.425
Learning CS outside of school	.88	.005
Future participation in CS	.82	< .001
Importance of CS	.95	.229

Note. W is the Shapiro-Wilk test statistic.

To address the second research question, a paired samples t -tests was conducted on each subscale mean difference where the assumption of normality was met to compare the subscale means of 28 participants before the intervention and after the intervention (see Table 4.10 for each parametric test result). A Wilcoxon signed-rank tests was conducted on each subscale median difference where the assumption of normality was not met to compare the subscale medians of 28 participants before the intervention and after the intervention See Table 4.11 for each nonparametric test result. When calculating the test statistics, the alternative hypotheses specified that the post subscale mean was greater than the pre subscale mean. The conservative Bonferroni adjustment was applied to account for the increased chance of Type I errors when running five tests, which changed the alpha level from .05 to .01 (Perneger, 1998).

Table 4.10 Parametric Inferential Results of Pre and Post-Survey Subscales

Subscale	Pre		Post		M Diff	t	p	d
	M	SD	M	SD				
Self-concept in CS	1.90	0.92	3.01	1.07	1.11	7.31	< .001	1.38
Learning CS at school	1.86	0.88	2.79	1.15	0.94	5.16	< .001	0.97
Importance of CS	3.25	0.85	3.70	0.98	0.45	2.90	.004	0.55

Note. $N = 28$. Pre = survey before the intervention; Post = survey after the intervention; M Diff = mean difference; d = Cohen's d

Table 4.11 Nonparametric Inferential Results of Pre and Post-Survey Subscales

Subscale	Pre <i>Mdn</i>	Post <i>Mdn</i>	Hodges-Lehmann Estimate	<i>z</i>	<i>p</i>	<i>r_b</i>
Learning CS outside of school	1.33	1.83	0.75	3.26	< .001	0.79
Future participation in CS	1.30	2.20	0.90	3.36	< .001	0.88

Note. *N* = 28. Pre = survey before the intervention; Post = survey after the intervention; *r_b* = rank-biserial correlation.

For the first subscale, the results of the paired samples *t*-test indicated that the difference between the post-self-concept in CS mean ($M = 3.01$, $SD = 1.07$) and pre-self-concept in CS mean ($M = 1.90$, $SD = 0.92$) was significant, $t(27) = 7.31$, $p < .001$. The mean difference was 1.11, and the effect size was large, $d = 1.38$.

For the second subscale, the results of the paired samples *t*-test indicated that the difference between the post-learning CS at school mean ($M = 2.79$, $SD = 1.15$) and pre-learning CS at school mean ($M = 1.86$, $SD = 0.88$) was significant, $t(27) = 5.16$, $p < .001$. The mean difference was 0.94, and the effect size was large, $d = 0.97$.

For the third subscale, on average, learning CS outside of school was higher after ($Mdn = 1.83$) the intervention than before ($Mdn = 1.33$). A Wilcoxon signed-rank test indicated that this difference was statistically significant, $z = 3.26$, $p < .001$. The Hodges-Lehmann estimate was 0.75, and the effect size was medium, $r_b = 0.79$.

For the fourth subscale, on average, future participation in CS was higher after ($Mdn = 2.20$) the intervention than before ($Mdn = 1.30$). A Wilcoxon signed-rank test indicated that this difference was statistically significant, $z = 3.36$, $p < .001$. The Hodges-Lehmann estimate was 0.90, and the effect size was large, $r_b = 0.88$.

For the fifth subscale, the results of the paired samples t -test indicated that the difference between the post-importance of CS mean ($M = 3.70$, $SD = 0.98$) and pre-importance of CS mean ($M = 3.25$, $SD = 0.85$) was significant, $t(27) = 2.90$, $p = .004$. The mean difference was 0.45, and the effect size was medium, $d = 0.55$.

Relationship of Attitudes and Performance

Mean survey composite scores were calculated for each participant by taking an unweighted average of the five subscales on the post-survey. To address research question three, Pearson's r was calculated for the mean post-survey composite scores ($M = 2.81$, $SD = 0.97$) and the post-content knowledge assessment scores ($M = 16.14$, $SD = 5.25$) to measure the linear correlation between participants' attitudes toward computer science and their performance. When calculating the test statistics, the alternative hypothesis specified that survey and assessment scores correlated positively.

Assumption Check. The bivariate normality of scores was assessed using the Shapiro-Wilk test. The results of the test were not significant, $W(27) = .99$, $p = .963$, indicating that assumption of normality was met. Homoscedasticity was assessed by examining the variances along the line of best fit, as shown in Figure 4.1. No extreme dissimilarities in variances were observed; therefore, homoscedasticity was assumed.

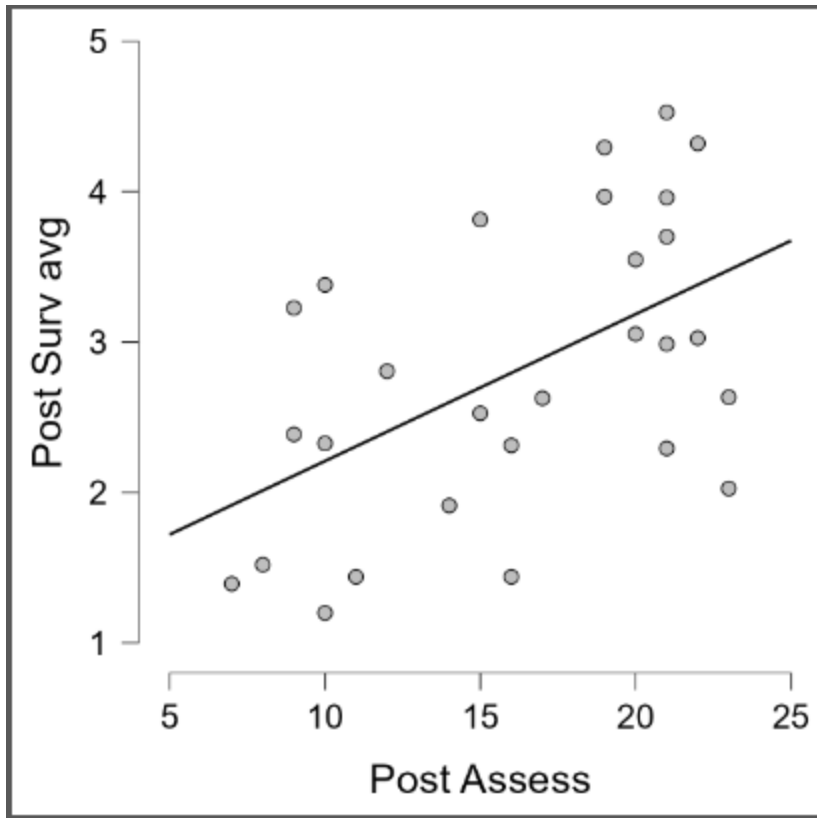


Figure 4.1 Composite Post-Survey Scores Vs. Posttest Scores

Correlation sizes were interpreted based on benchmarks proposed by Hinkle et al (1979). See Table 4.12 for the interpretation of correlation sizes. Composite post-survey scores and posttest scores were found to be moderately positively correlated, $r(26) = .53$, $p = .002$. This suggests that as participants' scores on the post-survey for attitudes toward computer science increase, so do participants' scores on the posttest.

Table 4.12 Interpreting the Size of a Correlation Coefficient

Interpretation	Size of Correlation
Very high positive (negative) correlation	.90 to 1.00 (-.90 to -1.00)
High positive (negative) correlation	.70 to .90 (-.70 to -.90)
Moderate positive (negative) correlation	.50 to .70 (-.50 to -.70)
Low positive (negative) correlation	.30 to .50 (-.30 to -.50)
Little if any correlation	.00 to .30 (.00 to -.30)

Quantitative Results Summary

The purpose of the quantitative analysis was to determine if there were (a) significant changes in participants' performance before and after the intervention, (b) significant changes in participants' attitudes before and after the intervention, and (c) a correlation between performance and attitudes after the intervention. To answer research question one, a paired samples *t*-test was conducted on the pre-content knowledge assessment scores and the post-content knowledge assessment scores. The results of the test indicated that there was a significant difference between pretest and posttest, indicating that the posttest scores were significantly higher than the pretest scores. To answer research question two, a paired samples *t*-test or a Wilcoxon signed-rank test was conducted between each of the five survey subscales before and after the intervention. All five tests were significant, indicating that for each subscale, the scores for post-survey were significantly greater than those for pre-survey. Finally, to address research question three, a correlational analysis was conducted between the post-survey composite scores and post content knowledge assessment scores. The correlation indicated that there was a moderately positive significant relationship between the variables, such that as attitudes toward computer science increased, so did the scores on the content knowledge assessment. The next chapter will discuss the implications of these results.

Qualitative Findings & Interpretations

For the qualitative portion of this study, semi-structured interviews were conducted with 12 participants after the intervention. The researcher observed participants during the intervention and compiled field notes. Qualitative data were

recorded and transcribed to prepare for analysis. The qualitative data sources and analysis are discussed below.

Qualitative Data Sources

This study used two methods for collecting qualitative data. A total of 31 field notes were collected during the intervention, and transcripts of participant interviews were analyzed using a process of inductive and deductive analysis (Creswell & Creswell, 2018; Fereday & Muir-Cochrane, 2006; Mertler, 2019; Tracy, 2020). Table 4.13 summarizes the number of codes applied to the field notes and transcripts during the first coding cycle.

Table 4.13 Summary of Qualitative Data Sources

Data Source	Number	Codes	References
Field notes	31 ^a	91	147
Participant Interviews	12	393	473
Total	43	484	620

^a Number of days that field notes were recorded.

Field Notes

Field notes consisting of researcher observations were recorded in a composition notebook during the intervention, as shown in Figure 4.2. Participant behaviors were recorded, including work on the custom game, conversations, and off-task behaviors. Researcher inferences of participants' attitudes and emotional reactions were also recorded. Most behaviors were recorded as they were observed. Researcher interactions with participants were recorded following the interactions.

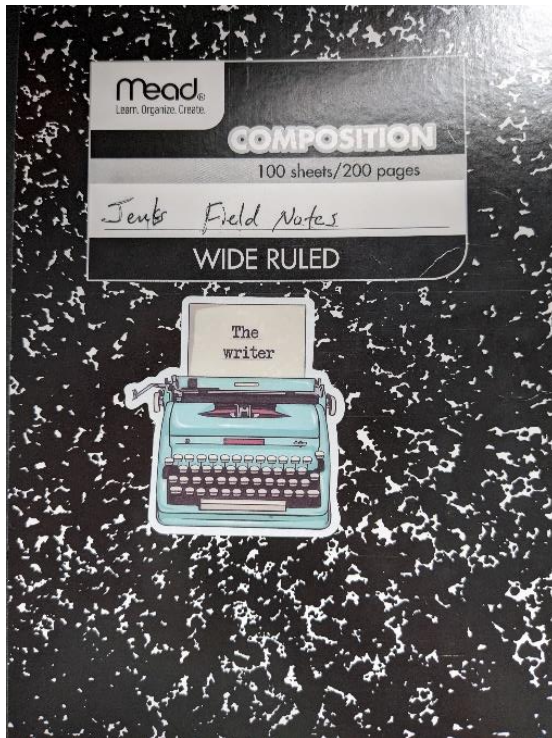


Figure 4.2 Field Notes Composition Notebook

Participant Interviews

Twelve participants completed semi-structured interviews after the intervention. Individual interviews were scheduled for 30 minutes. None of the interviews were affected by the time restriction, even when they took longer than the scheduled 30 minutes. The shortest interview took 9:54 minutes, and the longest interview took 39:51 minutes. Most interviews took place virtually in Google Meet on weekends. A few interviews were conducted during class for participants who had difficulty meeting on the weekend. The interviews were automatically transcribed in Otter.ai.

Analysis of Qualitative Data

Field notes were copied from the composition notebook into a Word Document. The Word Document consisted of three columns: (a) date, (b) observations, and (c) memo. Handwritten field notes, as shown in Figure 4.3, were copied to the observations

column in the Word Document, as shown in Figure 4.4. Analytic memos were added to the memo column of the Word Document. Once analytic memoing was completed, the field notes Word Document was uploaded to Delve for coding.

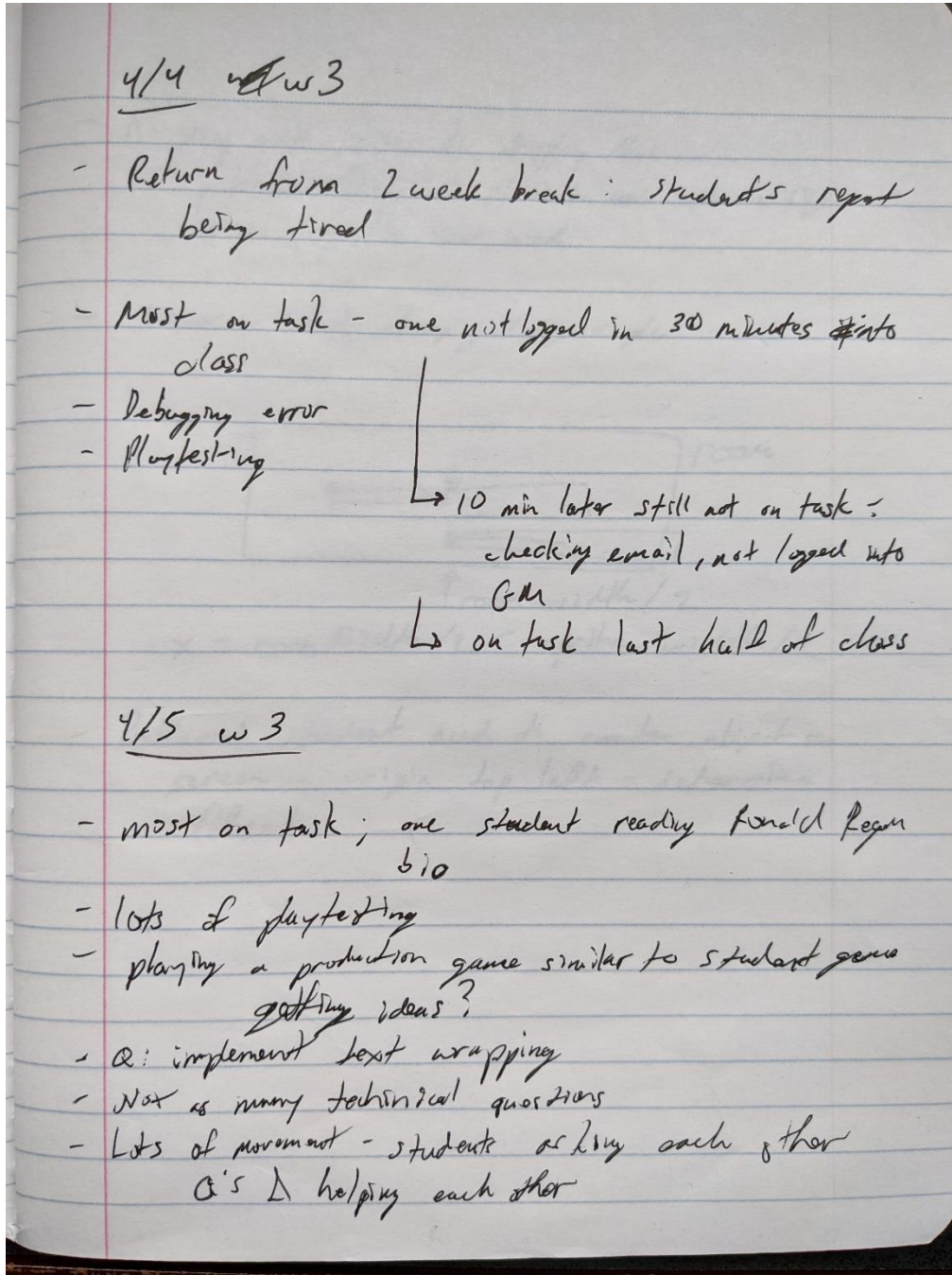


Figure 4.3 Handwritten Field Notes

W3 4/4/22	Return from <u>2 week</u> break; students report being tired	Students' sleep schedules <u>are probably</u> set to late waking hours. This class meets at 8 in the morning, and the students appear <u>very</u> tired.
	Most on task; one not logged in 30 minutes into class; 10 minutes later still not on <u>task</u> – checking email, not logged into GM; on <u>task</u> last half of class	
	Debugging error	
	Playtesting	
4/5/22	Most on task; one student reading Ronald <u>Reagan</u> biography	Another disadvantage of being <u>the</u> first class is that students sometimes need to complete work for <u>classes</u> later in the day. Students will sacrifice work time in this class to satisfy immediate deadlines in <u>other classes</u> .
	Lots of playtesting	
	<u>Playing</u> a production game similar to <u>student game</u> – getting ideas?	
	Q: implement text wrapping	
	Not as <u>many</u> technical questions	
	Lots of movement – students asking each other questions & helping each other	Students are now physically moving around the classroom. Some are seeking ideas or help from other students. Some are <u>simply</u> curious about other students' <u>games</u> and want to try <u>them</u> .
	Q: keep multiple scores & display them; problem: find all places in	<u>This</u> should be easy with the find & replace feature in GM, but students still don't utilize <u>this</u> feature.

Figure 4.4 Word Document Field Notes

The interviews were recorded and automatically transcribed in Otter.ai. The initial audio files and transcriptions were placed in a Needs Cleaning folder, as shown in Figure 4.5. The recordings were compared to the transcriptions to ensure accuracy. An example of a sentence that Otter.ai transcribed from Jonie was, “Like you said the current learning was hard to read.” After comparing this sentence to the audio, the sentence transcription was revised to, “Like you said, the Carnegie Learning was hard to read.” The cleaned audio files and transcriptions were placed in a Cleaned Interviews folder, as shown in Figure 4.6. Note that Figures 4.5 and 4.6 used the original pseudonyms (see Appendix I). The cleaned interview transcripts were then downloaded as Word Documents. The interview transcript Word Documents were uploaded to Delve for

coding. Cleaned interview transcripts were emailed to the appropriate participant for member checking. The email template is shown in Figure 4.7. None of the participants responded to the email.

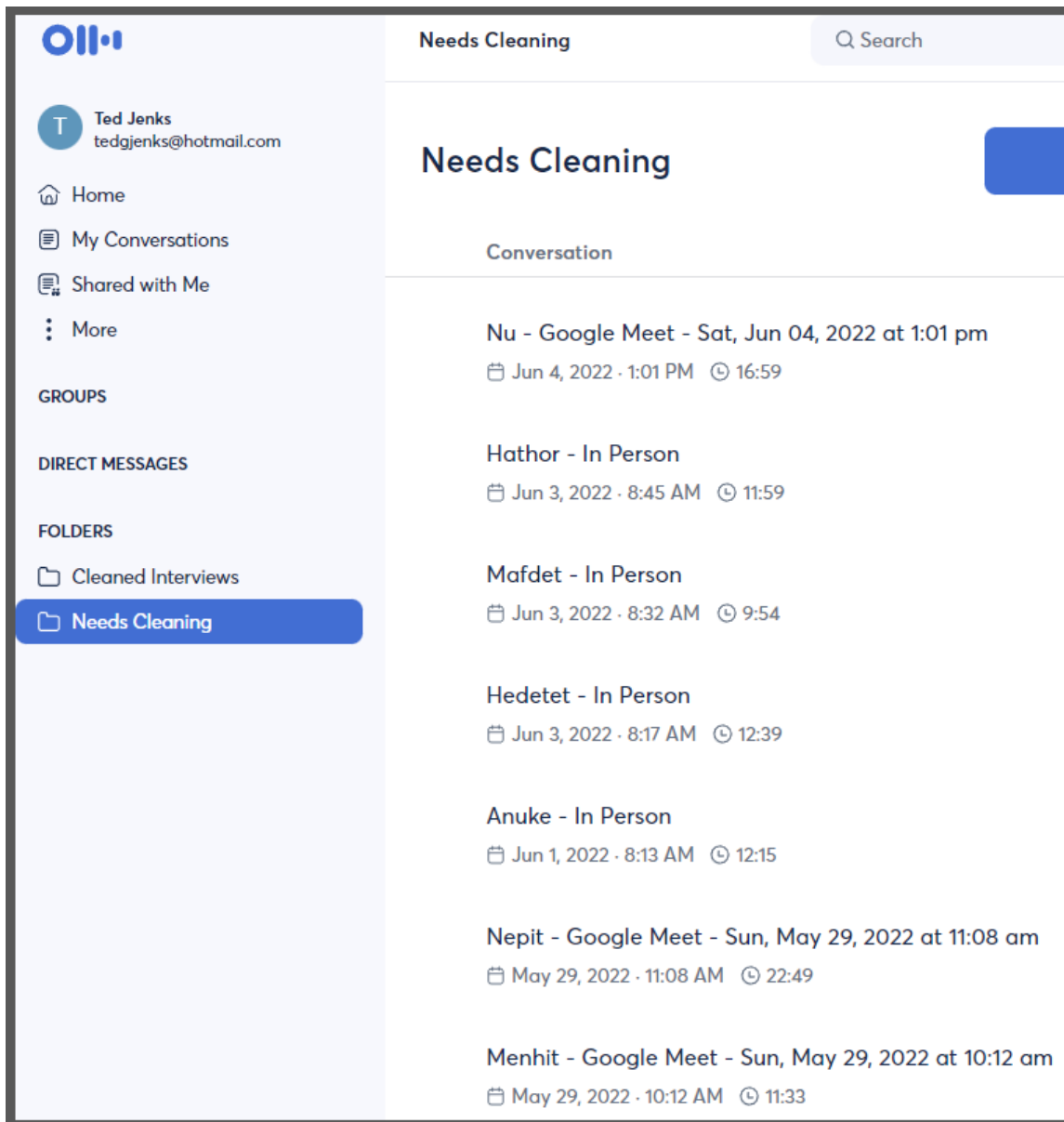


Figure 4.5 Otter.ai Needs Cleaning Folder

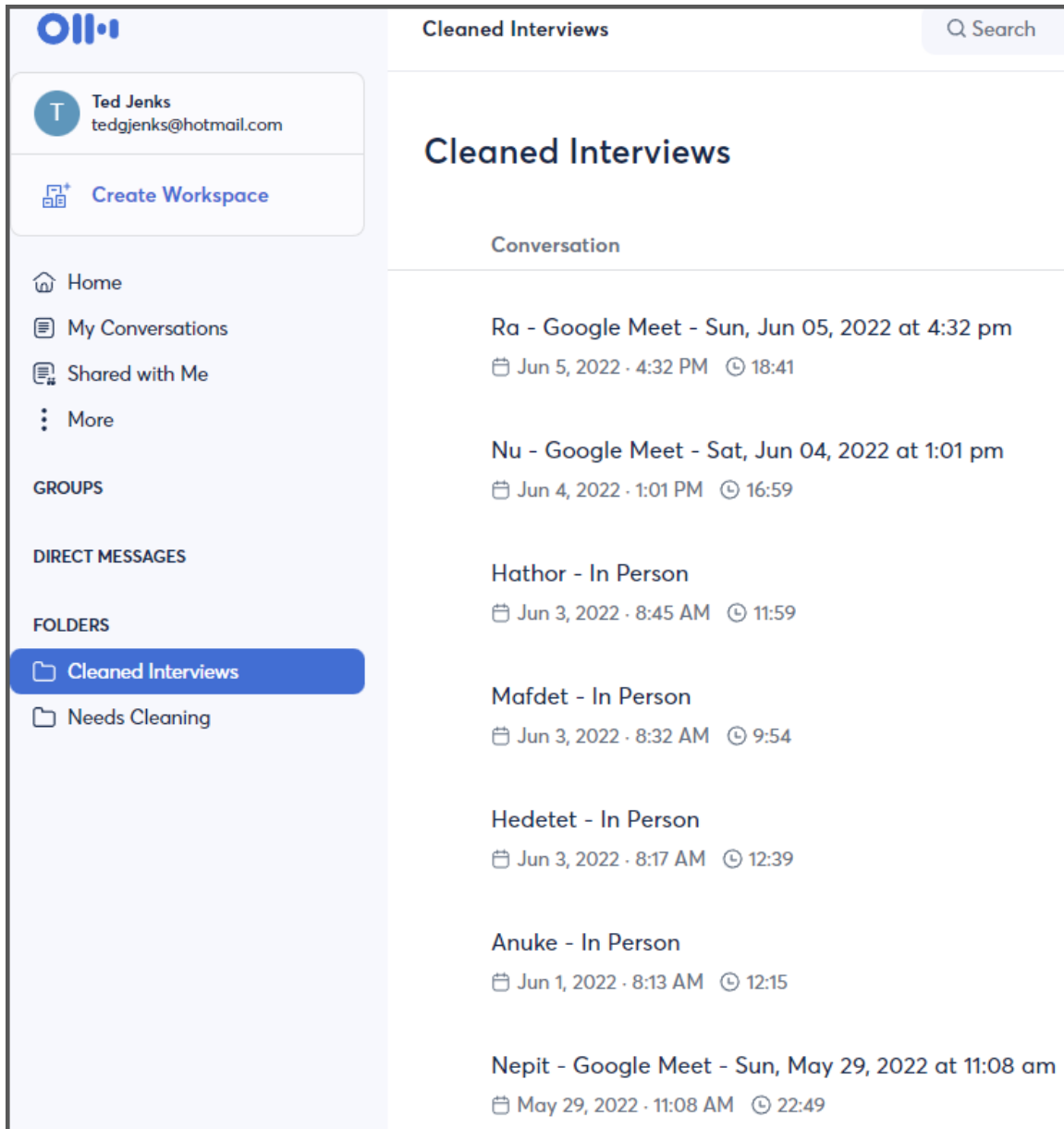
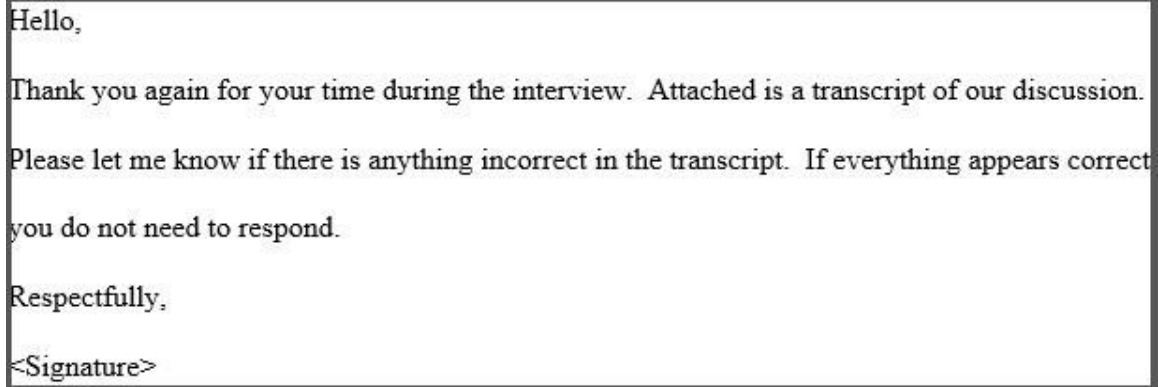


Figure 4.6 Otter.ai Cleaned Interviews Folder

The image shows a screenshot of an email template. The text is as follows:

Hello,

Thank you again for your time during the interview. Attached is a transcript of our discussion.

Please let me know if there is anything incorrect in the transcript. If everything appears correct you do not need to respond.

Respectfully,

<Signature>

Figure 4.7 Review Interview Transcript Email

Three cycles of coding were performed on the qualitative data in Delve as part of the inductive and deductive analysis (Creswell & Creswell, 2018; Fereday & Muir-Cochrane, 2006; Mertler, 2019; Saldaña, 2021; Tracy, 2020). Inductive coding was used primarily in the first coding cycle to remain open to new discoveries (Saldaña, 2021). Deductive coding was used later to make connections to the research questions (Mertler, 2019). In the first cycle of coding, the researcher read the transcripts line by line to become familiar with the data. Excerpts of the transcripts were grouped into individual codes. In the second cycle of coding, the researcher identified connections between codes and further refined them. Related codes were placed in categories that reflected their shared meaning. In the third cycle of coding, the researcher named themes and patterns based on the codes and categories that had been refined in previous steps.

Following each coding cycle, the researcher reviewed the generated codes and compared them to existing findings to ensure that the codes were reflective of the data. Coding cycles were used to allow for an iterative data analysis process. Within each cycle, multiple rounds of coding were conducted. Codes were compared to each other to identify instances where codes could be merged, revised, or removed. This process ensured that the codes accurately reflected the data.

Data analysis was guided by a hybrid approach of inductive and deductive coding (Fereday & Muir-Cochrane, 2006; Saldaña, 2021). Inductive coding was used in the first coding cycle. During rounds of coding in the second cycle, connections to the research questions were made using structural and pattern coding (Mertler, 2019; Saldaña, 2021). Categories were refined so that they were relevant to the research questions. The themes that emerged in the third coding cycle functioned as answers to the research questions.

Participants regularly devoted several sentences of their verbal responses to the same concept without adding new information. For example, participants would repeat themselves. Therefore, data were coded on the level of a complete thought when appropriate. For instance, the following statement from Monster Fan was treated as a complete thought and thus assigned one distinct code, *disliked missing class*:

I could not know that, you know, I'd like you know, like, on like, some days, like, you know, because like, I had been like a while ago, there would have been like three days I was out, you know, and so I couldn't, you know, work on the project. And I like been planning on like big stuff to do or like not real big, but you know, like planning on stuff to do. And because you know, that like me being behind by a little bit a few days, you know, I probably have like gone and as far as like I possibly could have, you know, iron out a couple of bugs, but you know, like when I've found bugs, you know, I could at some time figure out how to get rid of them. I like honestly, like didn't like being out those days because like I said I was planning on doing stuff for the project – couldn't do that.

First Coding Cycle

For the first cycle of coding, the following methods were used: (a) descriptive, (b) in vivo, (c) process, (d) values, and (e) causation (Saldaña, 2021). Additional coding methods were used when convenient; for instance, simultaneous coding was used sparingly to apply multiple codes to a datum (Saldaña, 2021). For example, the codes *coding is hard* and *no future interest* were applied to Annabelle's statement, "Um, coding is hard, and [I] don't want to pursue any sort of career that involves computer science." Descriptive coding was used to identify topics. In vivo codes were used whenever possible to prioritize participants' voices. Process coding was appropriate for identifying the behaviors of participants while they attempted to reach a goal or solve a problem. Values coding was used to capture participants' attitudes, which addressed two of the three research questions. Causation coding was used to infer the effects that the intervention had on participants' performance and attitudes. Causation coding was also used to explore relationships between attitude and performance. Examples of codes for each of the main first cycle coding methods are listed in Table 4.14. After a peer debriefing session with Dr. Arslan-Ari, an effort was made to make first cycle codes more specific. For example, codes for demonstration of learned skills (LS) included information about which skills were learned. *Ls: coordinates* coded participants who used the coordinate system to position visual output. *Ls: movement* coded participants who implemented moving instances. Four hundred eighty-seven codes emerged after the first coding cycle. The right arrow symbol (→) is used in causation coding to indicate that the cause on the left produced the outcome on the right (Saldaña, 2021).

Table 4.14 Examples of First Cycle Coding Methods

Method	Example Code	Code Explanation
descriptive	Learned skill: arrays	Participants reported learning how to use arrays.
in vivo	“Now I don’t see it boring at all”	The participant thought coding was boring prior to the intervention but did not view it that way after the intervention.
process	Enjoying aesthetic design	Participants reported or were observed enjoying aesthetic work in their games.
values	Not feeling confused	The participant reported feelings of confusion prior to the intervention but did not feel confused about the content after the intervention.
causation	Choice → fun	The participant reported that developing the game was fun because “I [the participant] got to do what I [the participant] wanted to do.”

Second Coding Cycle

Pattern and structural coding were used for subsequent rounds of coding to group codes into condensed categories and connect categories to the research questions (Saldaña, 2021). In pattern coding, summarized segments of data are grouped into condensed categories and themes (Saldaña, 2021). In structural coding, conceptual phrases relate data to specific research questions (Saldaña, 2021). Microsoft Excel was used in the second coding cycle to organize codes into categories. First cycle codes were exported from Delve into Excel as shown in Figure 4.8. During the second coding cycle, codes were organized into categories; a partial view is shown in Figure 4.9. Codes that were determined to have similar meaning or represent a group of codes were coalesced. Fifteen categories were generated by codes and categories of similar meaning. The 15 categories included: *using resources*, *hurting performance*, *impacts performance*

measurement, real-life lessons, enthusiasm, soliciting feedback, engagement, pride in work, dev problems, ambitious, beyond scope, experimenting, skill evidence, perception of assessment, and attitude/perception.

1	Order of C	Nested Le	Code Nam	Number o	Code Des	Code URL
2	1 >		NOT TESTI	1		https://app
3	2 >		UNFAIR AS	1		https://app
4	3 >		IGNORING	1		https://app
5	4 >		IGNORING	1		https://app
6	5 >		"ONE LAST	2		https://app
7	6 >		"WHAT AF	1		https://app
8	7 >		"WANNA	1		https://app
9	8 >		"I FOUND	1		https://app
10	9 >		"THAT'S TI	1		https://app
11	10 >		SMILING	1		https://app
12	11 >		LAUGHING	1		https://app
13	12 >		OFF TASK:	1		https://app
14	13 >		"CAN I PLA	1		https://app
15	14 >		INTENSE F	1		https://app
16	15 >		"IT'S ENTE	1		https://app
17	16 >		"GAME IS	1		https://app
18	17 >		PROBLEM:	1		https://app
19	18 >		"SO SICK C	1		https://app

Figure 4.8 Delve Codes Export to Excel

1	USING RESOURCES	HURTING PERFORMANCE	IMPACTS PERFORMANCE MEASUREMENT	REAL LIFE LESSONS	ENTHUSIASM	SOLICITING FEEDBACK	ENGAGEMENT
2	REFERENCING GM IN	NOT TESTING ENOUGH	UNFAIR ASSESSMENT	IGNORING GAME IN	"ONE LAST TIN"	"WHAT ARE YOUR COM	INTENSE FOCUS
3	DOCUMENTATION F	IGNORING PERFORMANCE	INSTRUCTOR: LATE PROBLEM RECOGNITIO	"I FOUND A WAY TC	SMILING	"WANNA PLAY?"	PLAYTESTING
4	SCAFFOLD WITHOUT	OFF TASK: TEXTING	VAGUE LO	"THAT'S THE ONE BI	LAUGHING		MAINTAIN DOCU
5	ABILITY TO DIG DEAI	PROBLEM: CODE MAINTENANCE		"GAME IS SO BROKE"	"CAN I PLAYTEST OTHER GAMES?"		RAPID PROTOTO
6	SIMILAR MATH CON	"SO SICK OF THIS GAME - OVERCODED!"		AMBIGUOUS GAME	"IT'S ENTERTAINING"		AESTHETIC WOR
7	ACTIVATING PRIOR	OFF TASK: PHONE			EXCITED ABOUT PROGRESS		ON TASK
8	LS SEARCH / REPLAC	FORGETTING WHAT YOUR OWN CODE DOES			WANT TO PLAY		ATTENTION TO D
9		DEBUGGING INEFICIENTLY			PHYSICAL MOVEMENT		ATTENTION TO D
10		UNDERUTILIZED TOOL					
11		REFACTORING WOES					
12		OFF TASK: OTHER COURSEWORK					
13		OFF TASK: EMAIL					
14		AVOIDING CODING					
15		NOT USING DOCS					

Figure 4.9 Second Coding Cycle in Excel

In subsequent rounds of coding, categories were created in Excel with the research questions in mind. Excel was also used to track which categories were

subsumed by new categories, as shown in Figure 4.10. *Impacts performance* and *beyond scope* included codes, such as *unfair assessment*, that interfered with participants' ability to demonstrate skills or knowledge; therefore, they were subsumed by *hurting performance*. *Dev problems* contained instances of participants who struggled initially but later demonstrated skills or knowledge following appropriate scaffolding. Thus *dev problems* was renamed *scaffolding*. *Drop* was created to hold codes that did not seem relevant such as *expected multiplayer*.

1	Final	Source 1	Source 2	Source 3	Source 4	Source 5
2	HURTING PERFORMANCE	HURTING PERFC	IMPACTS PERFORM	BEYOND SCOPE		
3	SCAFFOLDING	DEV PROBLEMS				
4	SKILL EVIDENCE	SKILL EVIDENCE				
5	DROP?					
6	Self-concept	ATTITUDE/PERC	USING RESOURCES	AMBITIOUS	EXPERIMEN	PERCEPTIO
7	Learning at School	ATTITUDE/PERC	ENTHUSIASM	SOLICITING FEED	ENGAGEMENT	PRIDE IN W
8	Learning Outside School	ATTITUDE/PERCEPTION				
9	Future Participation	ATTITUDE/PERCEPTION				
10	Importance	ATTITUDE/PERC	REAL LIFE LESSONS			
11	Attitude Performance Rel	ATTITUDE/PERCEPTION				

Figure 4.10 Second Cycle Transition Notes One in Excel

Self-concept, *learning at school*, *learning outside school*, *future participation*, and *importance* related directly to the survey subscales. The hybrid process of inductive and deductive coding allowed themes to emerge from the data through inductive coding and the analysis to be guided by the research questions serving as a priori templates (Fereday & Muir-Cochrane, 2006). Codes from *attitude/perception* that dealt with a change in attitude during the intervention were moved to the categories corresponding to the survey subscales. *Using resources*, *ambitious*, and *experimenting* included codes, such as *ability to dig deeper*, that demonstrated confidence in the participants' knowledge and skills; they were subsumed by *self-concept*. *Enthusiasm*, *soliciting feedback*, *engagement*, and *pride in work* contained codes, such as *excited about progress*, that demonstrated positive

attitudes during the intervention at school; they were subsumed by *learning at school*. *Real-life lessons* contained codes, such as *ignoring game instructions*, demonstrating participants' understanding of how computer science impacted individuals or society outside of school; *real-life lessons* was subsumed by *importance*. Codes from *attitude/perception*, such as *good attitude* → *try to fix problems*, that implied a relationship between attitude and performance were moved to *attitude performance relationship*.

In the next round of coding, codes were categorized in Excel. The coding structure in Excel was then applied to the codes in Delve. Category changes were tracked in Excel, as shown in Figure 4.11. The codes that reflected a positive change in self-concept were moved from *self-concept* to *pos self-concept*, while the codes that reflected a negative change in self-concept were moved from *self-concept* to *neg self-concept*. For example, *good at coding* was categorized as *pos self-concept*, and *cs is complicated* was categorized as *neg self-concept*. The codes that reflected a positive attitude change in *learning at school* were moved to *pos learning at school*, while the codes that reflected a negative attitude change in *learning at school* were moved to *neg learning at school*. For example, *doing is fun* was categorized as *pos learning at school*, and *don't enjoy class* was categorized as *neg learning at school*. *Learning outside school* and *future participation* indicated changes in attitude beyond the intervention and were subsumed by *sparked interest*. Some codes, such as *time with error correlated with level of struggle*, that revealed a relationship between attitude and performance were moved from *self-concept* to *attitude performance relationship*.

1	Final	Source 1	Source 2
2	POS SELF-CONCEPT	Self-concept	
3	NEG SELF-CONCEPT	Self-concept	
4	POS LEARNING AT SCHOOL	Learning at School	
5	NEG LEARNING AT SCHOOL	Learning at School	
6	SPARKED INTEREST	Learning Outside School	Future Participation
7	IMPORTANCE	Importance	
8	ATTITUDE PERFORMANCE RELATIONSHIP	Attitude Performance Relationship	Self-concept
9	HURTING PERFORMANCE	HURTING PERFORMANCE	
10	SCAFFOLDING	SCAFFOLDING	
11	SKILL EVIDENCE	SKILL EVIDENCE	
12	DROP?	DROP?	

Figure 4.11 Second Cycle Transition Notes Two in Excel

After a peer-debriefing session with Dr. Arslan-Ari, another round of coding examined the category *attitude performance relationship*. Subcategories were added to provide additional information regarding the relationship of the 65 codes in the *attitude performance relationship* category. Codes in *attitude performance relationship* were categorized as (a) *rolling with the punches*, (b) *negative att*, (c) *no relationship*, (d) *general att*, (e) *intervention positive*, (f) *baggage*, (g) *success multiplier*, and (h) *frustrating errors*.

Third Coding Cycle

Delve was used to categorize codes and develop themes in the third coding cycle. Codes were exported from Delve to Excel to obtain code counts for each category. In Excel, codes were grouped and collapsed for a summative view of themes and categories, as shown in Figure 4.12. After a peer debriefing session with Dr. Arslan Ari, several changes were made to the categories. As shown in Figure 4.13, the theme *performance improvements* was created with the following categories: (a) *transfer*, (b) *productivity*, (c) *algorithm skills*, (d) *tool skills*, and (e) *+collaboration / learning from*. *Scaffolding* and *skill evidence* were removed as categories, and most of their codes were moved to

algorithm skills. To improve clarity, the label “problem,” which appeared in several of the scaffolding codes, was changed to “scaffold.” *Problem: object destruction* was changed to *scaffold: object destruction* and moved from *scaffolding* to *algorithm skills*. Several codes from *pos self-concept*, such as *performance task better*, did not mention attitudes and were moved to *performance improvements*. Table 4.15 contains a detailed depiction of the progression of categories through coding cycles two and three and the themes that emerged at the end of cycle three.

1	2	A	B	C	D	E
	1	Order of C	Nested	Code Name	Cat Count	Number o
	2	1 >		PERFORMANCE IMPROVEMENTS	185	0
	3	2 >>		ALGORITHM SKILLS	125	0
+	85	84 >>		+COLLABORATION LEARNING FI	20	11
+	94	93 >>		PRODUCTIVITY	4	0
+	97	96 >>		TOOL SKILLS	35	0
+	128	127 >>		TRANSFERENCE	1	0
+	130	129 >		BARRIERS TO SUCCESS	73	0
	131	130 >>		ALGORITHM TROUBLE	6	0
+	137	136 >>		FACTORS DEC PERFORMANCE	67	1
+	174	173 >		POSITIVE ATTITUDES	290	0
	175	174 >>		+ATT AS	128	0
+	262	261 >>		-ATT AS	23	0
+	281	280 >>		IMPORTANCE	66	0
+	338	337 >>		+SELF-CONCEPT	34	0
+	367	366 >>		-SELF-CONCEPT	3	0
+	371	370 >>		SPARKED INTEREST	30	0
+	397	396 >>		NONPOSITIVE INTEREST	6	0
+	401	400 >		ATTITUDE PERFORMANCE FEEDB.	69	1
	402	401 >>		BAGGAGE	3	0
+	406	405 >>		FRUSTRATING ERRORS	19	0
+	423	422 >>		GENERAL ATT	9	0
+	432	431 >>		INTERVENTION POSITIVE	9	1
+	441	440 >>		NEGATIVE ATT	2	0
+	444	443 >>		NO RELATIONSHIP	1	0
+	446	445 >>		ROLLING WITH THE PUNCHES	2	0
+	449	448 >>		SUCCESS MULTIPLIER	23	0

Figure 4.12 Grouped Codes in Excel at the End of Cycle Three

Table 4.15 Category Progression

Coding Cycle	Categories
2	Using resources, Hurting performance, Impacts performance measurement, Real life lessons, Enthusiasm, Soliciting feedback, Engagement, Pride in work, Dev problems, Ambitious, Beyond scope, Experimenting, Skill evidence, Perception of assessment, Attitude/perception
2	Hurting performance, Scaffolding, Skill evidence, Drop?, Self-concept, Learning at school, Learning outside school, Future participation, Importance, Attitude performance relationship
2	Hurting performance, Scaffolding, Skill evidence, Drop?, Positive self-concept, Negative self-concept, Positive learning at school, Negative learning at school, Sparked interest, Importance, Attitude performance relationship
3	Algorithm skills, Positive collaboration and learning from others, Productivity, Tool skills, Transfer, Algorithm trouble, Factors decreasing performance, Positive attitude at school, Negative attitude at school, Importance, Positive self-concept, Negative self-concept, Sparked interest, Nonpositive interest, Baggage, Frustrating errors, General attitude, Intervention positive, Negative attitude, No relationship, Rolling with the punches, Success multiplier
3	Themes: Performance improvements, Barriers to success, Positive attitudes, Attitude performance feedback loop

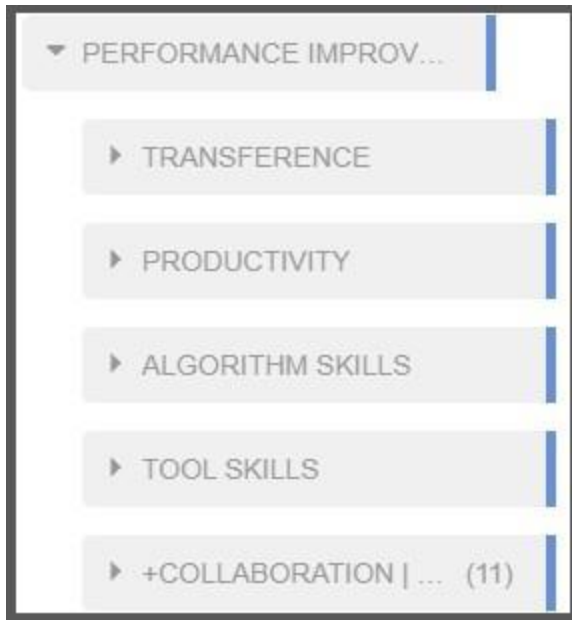


Figure 4.13 Performance Improvements in Delve

Hurting performance was dissolved into two categories: *algorithm trouble* and *factors dec performance*. *Algorithm trouble* codes documented participants' difficulties with algorithms. Codes in *factors dec performance* documented factors that interfered with participants' ability to learn or perform. As shown in Figure 4.14, *algorithm trouble* and *factors dec performance* were then added to the theme *barriers to success*.

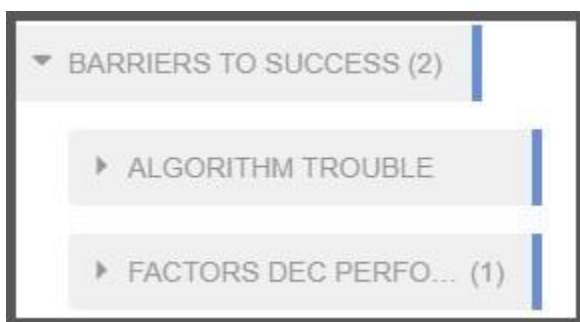


Figure 4.14 Barriers to Success in Delve

The theme *changes in attitude* was changed to the theme *positive attitudes*. Many of the codes and participant statements did not specifically mention change. Some categories represent negative attitudes, but the net attitudes reported by participants were

overwhelmingly positive. As shown in Figure 4.15, the theme *positive attitudes* contained the following categories: (a) *positive attitude at school*, (b) *negative attitude at school*, (c) *importance*, (d) *positive self-concept*, (e) *negative self-concept*, (f) *sparked interest*, and (g) *nonpositive interest*. After a peer-debriefing session with Dr. Arslan-Ari, “Learning” was judged to be a misleading name for a category with codes addressing attitudes, but not learning. Therefore, *Pos learning at school* and *neg learning at school* were changed to *positive attitude at school* and *negative attitude at school*, respectively. *Sparked interest* contained codes for participants who expressed no interest in computer science, such as *no future interest*. Those codes were placed in a new category, *nonpositive interest*.



Figure 4.15 Positive Attitudes in Delve

The category *attitude performance relationship* was developed into a theme and renamed *attitude performance feedback loop*, as shown in Figure 4.16.



Figure 4.16 Attitude Performance Feedback Loop in Delve

Several criteria were used for creating categories and determining which codes had significance: (a) relevance to the research questions; (b) high frequency of related codes; (c) counterevidence to the claims made in the study; and (d) insightful participant comments, especially if they were unsolicited by the interview questions. Most categories with fewer than 10 codes were either counterevidence or insightful comments. Table 4.16 presents the thematic structure of analysis.

Table 4.16 Hierarchical Structure of Themes

Themes	Categories	Sample Codes
1. Performance improvements	Algorithm skills	Applying knowledge→learning, playtesting feedback→improvement, initial
	Positive collaboration and learning from others	teamwork improved, pair programming, compromise, express all ideas, share ideas,
	Productivity	Code formatting for management, improved time management
	Tool skills	Good information on errors→easy to fix, data type mismatch→unpredictable behavior
	Transfer	Math
2. Barriers to success	Algorithm trouble	MC test hard, concepts not sticking, no algorithm example, coding is hard
	Factors decreasing Performance	GDD missing the point, not using prior work and not using resources and not using docs and API useless,
3. Positive attitudes	Positive attitude at school	Enjoying CS, enjoying playing, playing as reward, dread→custom game→eager
	Negative attitude at school	Morning fatigue, random boredom, bugs were a bummer, arguing, fighting like siblings
	Importance	CS helps people, AI accident, tech in psychiatry, benefit: fix bugs, many benefits
	Positive self-concept	Going through the motions, not feeling confused, self-reliance, scripted harder
	Negative self-concept	Troubleshooting stressful, “I’m not a coding type person”
	Sparked interest	More CS classes, future interest animation, future CS courses, Possible jackpot
	Nonpositive interest	Grasping future interest, no future interest and not changing interest

Themes	Categories	Sample Codes
4. Attitude performance feedback loop	Baggage	People talking → mad → decreased performance, unmotivated → unproductive, tired → unproductive
	Frustrating errors	Persistent error → confusion, frustration level depends on time to fix, errors → anger
	General attitude	Bad attitude → decreased learning, no interest → decreased learning, good attitude → better results
	Intervention positive	Creative freedom → brain flow → success, positive attitude → productive, early low motivation → low attention
	Negative attitude	Bad attitude → don't care → accept low scores, dislike coding → decreased motivation
	No relationship	No attitudes affected learning
	Rolling with the punches	Quick fix errors → no frustration, initial errors → neutral reaction
	Success multiplier	Good attitude → try to fix problems, want to learn CS → positive learning, focus → learning

Presentation of Findings

Qualitative data were recorded as field notes from observations and participant interview transcripts. Twenty-eight participants were observed, and 12 of those participants were interviewed. Four themes emerged from the qualitative analysis of the data, as shown in Table 4.17: (a) performance improvements, (b) barriers to success, (c) changes in attitude, and (d) attitude performance feedback loop. Quotations were verbatim from participant interview transcripts or field notes that recorded participants' conversations. Participants' names have been replaced with pseudonyms or redacted to ensure confidentiality.

Table 4.17 Description of Themes

Themes	Assertions
1. Performance improvements	Participants demonstrated performance improvements during and after this intervention by developing a digital game.
2. Barriers to success	Many factors that were not directly related to the intervention reduced the effectiveness of the intervention.
3. Changes in attitude	Participants' attitudinal changes were overwhelmingly positive throughout the intervention.
4. Attitude performance feedback loop	Participants' attitudes and performance were related and magnified each other.

Performance Improvements

Most participants exhibited and reported performance improvements after this intervention, as indicated by codes for *performance improvements* appearing 185 times throughout the field notes and interview transcripts. All participants successfully constructed a digital game and satisfied the minimum project requirements. All 12 participants who were interviewed reported performance improvements. For this study, performance is defined as a demonstration of learning or application of a skill. Previous research indicated that PjBL and GDBL were effective strategies for introducing programming concepts (Erümit et al., 2020; Johnson, 2017; Theodoraki & Xinogalos, 2014). Participants were observed learning and applying computer science concepts during the development of a digital game. After the intervention, participants reported an increase in their computer science knowledge and ability to construct a digital game. This theme is supported by five categories: (a) *algorithm skills*, (b) *positive collaboration and learning from others*, (c) *productivity*, (d) *tool skills*, and (e) *transfer*.

Algorithm Skills. The category *algorithm skills* represented participants' increased ability to analyze and develop algorithms following the intervention. Participants ($n = 12$) demonstrated or reported an increased ability to analyze and develop algorithms after the intervention. These skills have some independence from the GameMaker IDE and could be applied to other programming languages and IDEs. Qianna said, "If I read like a paragraph on like, what to do, then I had to like, analyze it, and like, break down the steps in order to do it." The ability to evaluate a large problem and separate it into manageable tasks is a fundamental skill in algorithm development (Bowden, 2019; Committee on STEM Education of the National Science and Technology Council, 2018). Julia reported learning iteration, "I learned how to make for loops, that was a big one." Jonie indicated success with analyzing sequencing, "We had to ... rethink about the question because ... the numbers had to be in a specific order." Sequencing and iteration are fundamental constructs in algorithm development (CollegeBoard, 2020b; Moreno, 2012). Annabelle stated that before the intervention, "I copied the exact code word for word, maybe changed a couple of numbers. But that was it. And now I can actually code some if I'm just following instruction [*sic*]." Pibb said, "It's [the intervention] helped me to ... program my own code personally, like not having to write off the template." Annabelle's description indicated little perceived learning before the intervention. After the intervention, Annabelle and Pibb described the ability to write software from requirements without starter code.

Collaboration. The category *positive collaboration and learning from others* provided evidence that collaboration skills improved. Jumaat et al. (2017) considered the opportunity to work collaboratively as an essential PjBL element. Participants reported

improvements in their ability to collaborate effectively. Julia said, “We would switch off each day while the other person would ... put the code into the game, and their partner read the code and tell the partner how to do it.” Jonie reported utilizing other participants for help: “If I had like a bug in my game, and before asking you I would turn to somebody beside me [to] see if they can help.” Pair programming and peer review have the potential to improve code quality and speed when used correctly. Julia and Jonie were able to improve their programming efficiency through collaboration.

Other participants reported improved teamwork and communication skills. Annabelle said, “Teamwork was one thing I developed because I didn’t ... like working ... with people.” Marlena said, “I guess taking this course was helping me become a little bit more social towards other people in my class.” Marlena also said, “The computer science course ... involves to listen [*sic*] to other people’s opinions.” After discussing sharing ideas between partners, Aleesha explained that it was important to share ideas “so you can make games you both want to make. So it’s not just one-sided.” These comments demonstrate that participants developed an appreciation for others’ opinions and a willingness to compromise.

Productivity. The category *productivity* provided evidence of productivity increases that were not directly related to algorithm analysis and development or GameMaker. In this study, productivity refers to the ratio of correctly implemented game behaviors to development time. Participants were not explicitly asked about productivity, but Pibb and Julia mentioned productivity improvements when asked about what skills participants were expected to learn. Pibb said, “Indenting, when you get to coding a lot ... you can have like huge pages worth of code, and it can get really hard to manage, but

indenting helps me see ... what affects the other thing.” Indentation is not required by the GameMaker compiler, but it is a useful coding convention for indicating that code blocks belong to control structures. As Pibb indicated, when the codebase grows, indentation improves the readability of the code. Pibb said that the intervention helped with understanding “how to manage the time I have with programming because we really didn’t have ... too much time to work on it.” Julia said, “It [the intervention] also improved my time management because I knew I had to get it done.” Participants were given a strict development schedule to guarantee that they had a working game by the end of the intervention. When Pibb and Julia claimed that the intervention helped with their time management skills, they were probably referring to the deliverable schedule provided with the project directions.

Tool Skills. The category *tool skills* documented skills related to the GameMaker language and IDE. Unlike *algorithm skills* and *productivity*, *tool skills* were limited to GameMaker and would not necessarily transfer to another development environment or programming language. Bree said, “I feel like I’m a lot faster. ... I know where everything is on GameMaker.” Adding specificity, Bree said, “Debugging and using the search and replace is very helpful for us.” Bree reported increased development efficiency due to increased familiarity with GameMaker. Qianna made a similar comment, “With GameMaker, it’s like you can do like so much more with it ... it seems like you can do anything with it.” Like Bree, Qianna expressed increased familiarity with GameMaker and an increased ability to produce desired results.

Transfer. The *transfer* category characterized the way in which learning in the intervention enhanced learning in areas outside of computer science. One participant

commented on transfer. Participants were not asked about the perceived benefits that the intervention had in other subject areas. When asked about game development or programming skills acquired during the intervention, Abegail stated, “It also helped me with math a little too.” Programming is expected to have a positive transfer effect on mathematical skills (Scherer et al., 2019). Abegail could not provide significant detail regarding mathematical skills that benefitted from the intervention. When prompted, Abegail responded, “X and y values? And sometimes you have to multiply, divide.” All participants were enrolled in an algebra one mathematics course during the intervention. Some topics that appeared in the intervention and the algebra one course were (a) the Cartesian coordinate system, (b) rates of change, and (c) linear relationships.

Barriers to Success

Participants experienced some difficulty with mastering concepts and implementing their games during the intervention, as indicated by codes for *barriers to success* appearing 73 times throughout the field notes and interview transcripts. Nine participants reported barriers to success in the interviews. Barriers to success were expected; previous research stated that programming is challenging for students (Alturki, 2016; Cheah, 2020; Javidi & Sheybani, 2014; Végh & Stoffová, 2019). A small number of statements indicated problems with algorithm analysis and development, which the intervention directly targeted. The vast majority of barriers to participants’ success involved factors that were not directly related to the concepts or activities the intervention intended to address. This theme is supported by two categories: (a) *algorithm trouble* and (b) *factors decreasing performance*.

Algorithm Trouble. The category *algorithm trouble* provided evidence of participants ($n = 4$) who struggled with algorithm analysis and development. Teddie said,

The multiple choice one - I think it was a little hard for me. ... I did better on the performance test. I understood the code more, and I knew I understood what I was supposed to do better than when I first did it.

Teddie's declaration that the multiple-choice test was hard indicates difficulty with analyzing algorithms. Her relative success with the performance test shows that she was able to develop algorithms from requirements written in natural language. However, Teddie was unable to provide an example of an algorithm from the project: "I don't think I'd be able to give you an example. What do you mean by like algorithm, because I think that was one of the requirements right to make like an algorithm?" The researcher supplied Teddie with a definition of algorithm, but Teddie was still unable to produce an example. Teddie implemented several algorithms in the custom project. Teddie may have been anxious during the interview and had probably never integrated a useful algorithm definition. Some participants initially appeared confused when asked for an algorithm example, but almost all immediately provided an example when given a description of an algorithm.

Annabelle expressed an inability to retain concepts, "It's not sticking completely ... how to do it isn't going to stick for like the next 20 years. ... [concepts and skills] from the pinball game didn't stick with me when I was working on the custom project." An expectation that programming concepts will stick for 20 years without reinforcement is probably unreasonable. However, Annabelle's following observation is essential. The pinball game was completed before the intervention, and the custom game was part of the

intervention. Participants were provided a codebase and detailed instructions for modifications to achieve desired game behavior. Annabelle expressed an inability to apply the techniques in the pinball game to the custom project. Annabelle said, “Coding is hard, and [I] don’t want to pursue any sort of career that involves computer science.” Abegail echoed Annabelle’s statement about the difficulty of coding, “I didn’t know how hard it was to actually code a game.” Abegail also said, “It’s [the intervention] been pretty effective, I would say because I went into the course knowing absolutely nothing, but now I know a lot more than I did at the beginning.” The contrast between Annabelle’s and Abegail’s statements was striking. Both found coding difficult. Annabelle’s reaction was adverse, and Annabelle decided to avoid computer science. Abegail’s reaction was one of appreciation for the difficult work of programming a game.

Factors Decreasing Performance. The category *factors decreasing performance* provided evidence of behaviors and misunderstandings that decreased participants’ ($n = 7$) performance. Many factors decreased participants’ performance in the intervention. Instances of participants’ failing to utilize previous work and examples were documented in the field notes. Participants were instructed to keep a coding “cookbook” with common development and debugging procedures. Participants got stuck on problems that were nearly identical to previously solved problems. The researcher observed that few participants were utilizing their cookbooks when developing. The researcher would ask for their cookbooks when they asked for assistance on previously solved problems. Participants were rarely able to locate their cookbooks, indicating that the cookbooks were not utilized.

Wasted time due to off-task behavior was infrequent but observed during the intervention. Typical off-task behaviors were playing or messaging on smart devices and completing work for other courses. The intervention took place during the participants' first period; if they failed to complete homework the previous day, they would attempt to complete it during the first period. Development errors precipitated some of the off-task behavior. The following participant statements are examples of this problem.

- Aleesha: "When we got a bunch of errors, it kind of made us want to stop."
- Marlena: "If I couldn't do it, I'd give up."
- Jonie: "Like five to 10 minutes [before I give up because of an error]."
- Monster Fan: "If you like touched, you know, supposed to disappear just wasn't disappearing. ... I just forgot to put in the line of code. ... And so you goof off a little bit."

Participants were given heuristics for identifying and resolving errors. Unfortunately, several participants were not using the heuristics effectively.

Some participants exhibited frustration with the researcher's attempts at scaffolding because the researcher would not solve problems for participants. Instead, the researcher tried to lead participants to the information that would help them.

Annabelle expressed this frustration:

When there were some errors that we couldn't figure out, we tried to get you to help, but ... you're talking to us like we're college students on like, almost senior level in college, and we're like down all the way down here. And ... you didn't give us enough information on how to do what you were saying. And so when we ran into an error that we couldn't figure out, and we kept asking you for help, and

you kept giving us answers, and we tried to implement what you said. But we couldn't because ... we didn't understand how to do it completely. Because neither of us are ... people who want to code. ... It affected learning and doing code. ... Maybe instead of explaining what we need to do, try explaining how we need to do it.

Annabelle stopped caring about understanding the content; she just wanted the errors fixed. Annabelle's comment about "talking to us like we're college students" indicated a vocabulary deficit or a lack of prerequisite knowledge from previous units. The researcher's explanations may have demanded excessive cognitive load from this participant.

Positive Attitudes

For this study, attitudes were defined as beliefs, evaluations, or emotional responses toward ourselves, an object, an idea, or a person (Giannakos, 2014; Saldaña, 2021; Simonson, 1979). Of particular interest were attitudes toward computer science specified by the subscales in the associated survey: (a) self-concept, (b) learning at school, (c) learning outside of school, (d) future participation, and (e) importance. Previous research suggested that participants' attitudes would improve following a game development-based learning intervention (Çelik et al., 2018; Doman et al., 2015; Theodoraki & Xinogalos, 2014). The *positive attitudes* theme is supported by seven categories: (a) *positive attitude at school*, (b) *negative attitude at school*, (c) *importance*, (d) *positive self-concept*, (e) *negative self-concept*, (f) *sparked interest*, and (g) *nonpositive interest*. These categories represented attitudes that were not directly connected to performance, although relationships to performance could have been

inferred. Codes representing a direct relationship between attitudes and performance were represented in the *attitude performance feedback loop* theme.

Positive attitudes were coded in the categories (a) *positive attitude at school*, (b) *positive self-concept*, and (c) *sparked interest*. Participants reported net positive attitudes after the intervention, as indicated by codes appearing 192 times throughout the interview transcripts and field notes in the positive categories. All participants reported positive attitude changes.

Negative attitudes were coded in the categories (a) *negative attitude at school*, (b) *negative self-concept*, and (c) *nonpositive interest*. Negative attitudes appeared 32 times throughout the interview transcripts and field notes in the negative categories. Seven participants reported negative attitudes. Annabelle was the only interview participant whose attitude was consistently negative. Annabelle's interview transcript contained 18 of the 32 negative attitude codes.

Importance was considered a neutral category. Codes relating to the importance of computer science appeared 66 times in the interview transcripts and field notes. All participants felt that computer science was an important subject.

Positive Attitude at School. The category *positive attitude at school* captured participants' expressions of positive attitudes during the intervention while in the computer science class. Participants' attitude toward computer science at school during and after the intervention was remarkably positive. Before the intervention, participants completed games with predefined rules and gameplay. Qianna said, "In the beginning ... I didn't want to do it [program digital games] because it wasn't like [I didn't like] the games that we were doing. I wasn't interested." Participants found the games before the

intervention boring because they had no input into the game design. During the intervention, observations were recorded in the field notes of participants enjoying the creation of their games. In the interview, participants were asked if they enjoyed any aspects of the intervention:

- Oakley: We had the freedom to do whatever you want.
- Aleesha: Being able to do whatever we wanted without restrictions.
- Jonie: When we were trying to figure out what like what aspects to add to the game. ... So I feel like that part was the fun part. We just kind of messed around with a bunch of stuff.
- Bree: I do like the custom game cuz [*sic*] I feel like you can do anything with the game like that.
- Qianna I did enjoy developing my game because I thought it was fun because I got to do what I wanted to do.

Participants viewed the creative freedom of the custom game very favorably. They also enjoyed experimenting with features and gameplay.

Participants were eager to start class during the intervention. Teddie said, “First period is definitely the class I look forward to the most compared to other classes.”

Abegail said, “Well, it's the first class. I'm tired, but ... the thought of going to the class ... makes the beginning of my day a little bit better.” Pibb stated,

At the end of class every day, I'd suddenly get a new idea. And it'd be terrible because I had to leave. But the next day, I'd have that idea fresh in my mind. So I was excited to put it in and try and get it to work.

Participants were noticeably more enthused about class during the intervention than they were before the intervention.

Participants became especially animated when they were able to playtest each others' games. The room was filled with laughter and smiles during the playtesting phase. Following are remarks recorded in the field notes while participants were playtesting near the end of the intervention:

- “Can I playtest other games?”
- “What are your comments? We need player comments!”
- “Wanna play?”
- “One last time!”

Participants found the gameplay fun, but they also took pride in their work and were eager to show their games to their peers. Pibb recalled, “See that really come through and finally beat [name redacted], I was really, really exciting [*sic*].” Pibb is referring to his artificial intelligence system beating a human player. In another instance, Pibb described adjusting the playability to make the game easier, “At first we didn’t really give you any chance ... I ended up giving three extra rockets. ... It’s just so much fun seeing that come through in the end.” Most participants cared deeply about others enjoying their games. They found that playability was subjective and challenging; too easy or repetitive was boring but too hard or unpredictable was frustrating.

Negative Attitude at School. The category *negative attitude at school* captured participants’ expressions of negative attitudes during the intervention while in the computer science class. Participants expressed negative attitudes toward computer science after the intervention. The intervention occurred during the first period of the

day, just after 8:00. Some participants reported fatigue. Monster Fan said, “You know, with it being in the morning, I feel kind of like ... groggy.” Abegail said, “Well, it’s the first class. I’m tired, but ... the thought of going to the class makes ... the beginning of my day a little bit better.” Tired participants found focusing more challenging and did not perform at their best.

About half of the codes recorded in this category came from the interview with Annabelle. Annabelle said, “I hated coding – all of it. ... If you’re not one of those people who’s just like, instantly clicks. ... I don’t understand how other people would choose to do it in life.” Annabelle disliked everything about the whole class, not just the intervention. Annabelle did not enjoy the content, and held the misconception that programming was a native ability. Annabelle was aggravated by the environment as well, from the “freezing” room to the “really annoying kids.” Annabelle described working with a partner “like working with one of my sisters on literally anything argumentative.” Annabelle’s attitude and entire experience were very negative.

Importance. The *importance* category captured participants’ attitudes toward the importance of computer science. All participants felt that computer science was an important field with significant personal or societal impacts. Automation was seen as both potentially beneficial and harmful. Bree said, “It [self-driving cars] can be helpful to certain people that like can’t drive.” Jonie said, “For surgeries and stuff, if a robot is better, then computer science would need to be in that department.” These participants predicted beneficial effects of computer science. Abegail worried that “it [robots] could put people out of jobs.” Marlena agreed: “Many people could be losing their jobs

because of technology.” These participants worried that technology could create high unemployment.

Positive Self-Concept. The category *positive self-concept* provided evidence of participants’ positive self-concept after the intervention. Participants reported a net positive self-concept toward computer science after the intervention. For this study, self-concept was defined as participants’ beliefs regarding their competence in mastering computer science (Shen et al., 2014). Participants expressed confidence in their computer science knowledge and ability to program. Pibb said, “I enjoy doing it [computer science]. And it seems like I’m pretty good at it.” Teddie said, “In general, I was able to improve not only the code but my opinion on programming.” Bree, who struggled before the intervention, said, “I don’t feel like I’m very confused on anything anymore.” Participants were aware of how much they had learned.

Participants became more self-reliant after the intervention. Qianna said, “Instead of just sitting there not like doing anything, ... I like look it up online so that I can learn how to do it myself.” Marlena compared previous projects, which had detailed development steps, to the custom game where “I had no type of steps. ... I had to figure out that by myself.” Pibb said, “It [the intervention] really, really helped me understand ... how to do things completely on my own with no instructions.” Before the intervention, participants would passively wait for the researcher to solve problems for them. After the intervention, participants gained the confidence to problem solve on their own or collaboratively with other participants.

Negative Self-Concept. The category *negative self-concept* provided evidence of participants’ negative self-concept after the intervention. Some participants expressed a

negative self-concept following the intervention. Annabelle said, “I’m not a coding type person. ... My brain doesn’t work that way.” Oakley said, “The troubleshooting part was stressful, but in the end, you know, we got through it.” Annabelle had a low perceived computer science competency. The reason provided by Annabelle was a lack of native ability. Oakley described troubleshooting as stressful but acknowledged success in the end. Like Oakley, other participants reported negative emotions when dealing with errors. Reports of negative emotions related to performance decreases were categorized as *frustrating errors*.

Sparked Interest. The category *sparked interest* provided evidence that participants had an increased interest in computer science after the intervention. Participants expressed interest in computer science outside the classroom or in the future. Marlena described working on the custom game:

There was this one time where I had ... a really major error in our soccer game that I was talking about, and it got to the point where I came home one day and actually searched it up and took like, like some 20 minutes I think to like, search up how to solve a problem and then yeah, something that ... I used to not do.

Pibb compared his mathematics and computer science experiences:

I don't know how I'm good at programming but so bad at math. ... I don't really work on the stuff [mathematics] at home. ... Game design is one of those few classes [I] will like want to work on things at home.

Marlena and Pibb described working on their custom projects outside of class. Pibb attributed his relatively poor performance in mathematics and good performance in computer science to the extra time Pibb spent programming at home.

Participants discussed future interests in computer science. Qianna said, “I feel like because of this class, I might take one like a computer science class in high school, because I thought it was interesting, but I would think it'll help with like the job I want to do.” Jonie detailed a change in interest:

Well, when I walked into this class, I had no interest in computer science at all, but this class has made me think that maybe I might want to do something when I go to college with computer science just because it was fun.

Teddie’s future interest was “kind of like a hobby that I will just do in my free time.”

Participants declared interest in taking future computer science classes in high school and college as well as interests as hobbyists.

Nonpositive Interest. The category *nonpositive interest* provided evidence that participants showed no increased interest in computer science after the intervention. Two participants showed no interest in computer science outside of the classroom. Bree described how her interest in computer science changed as “not really much. ... I don’t think I’m really gonna stay in that.” When describing her aversion to computer science, Annabelle said,

Coding is hard, and [I] don’t want to pursue any sort of career that involves computer science. ... I’m learning something [computer science] that I’m never going to do. ... I have no plans to follow any computer science in my life.

While Bree’s response might indicate no change in interest, Annabelle’s interest declined severely. When asked about future interest in computer science, Abegail said, “I don’t know if this applies for being a psychiatrist in the medical field. You have to use a lot of technology in order to be able to do that.” Abegail’s response acknowledged that

technology would be hard to avoid, but Abegail did not reveal an increase in interest in computer science.

Attitude Performance Feedback Loop

Participants reported a strong relationship between attitude and performance, as indicated by codes for *attitude performance feedback loop* appearing 69 times throughout the field notes and interview transcripts. All 12 participants who were interviewed reported a relationship between attitude and performance. A feedback loop between attitude and performance was inferred from participants' interview responses. For this study, feedback loop was defined as a system that returns a portion of the output signal of the system to the input of the system (Spencer, 1994). According to participants, their performance influenced their attitudes, and their attitudes influenced their performance. Previous research suggested that attitude and performance were correlated (Alvarez et al., 2019; Çelik et al., 2018; Gurer et al., 2019). The theme *attitude performance feedback loop* is supported by eight categories: (a) *baggage*, (b) *frustrating errors*, (c) *general attitude*, (d) *intervention positive*, (e) *negative attitude*, (f) *no relationship*, (g) *rolling with the punches*, and (h) *success multiplier*.

Baggage. The category *baggage* provided evidence that participants arrived with negative attitudes unrelated to the class or intervention. Participants arrived with baggage before class began. Likewise, the cause of baggage was outside of the class. Participants reported being tired, irritated by other students, or generally unmotivated when they entered class. These attitudes negatively affected their performance. Abegail described a typical early morning experience:

I get aggravated really easily, so when people like in the hallways are trying to talk to me, I'm just not having it. Then ... I'll go into the classroom mad, so it's gonna affect how I perform for that first part of class.

Bree described being tired and unmotivated: "If I've come in like tired, or like I just don't want to do it, it's just kind of hard to get through it." The negative attitudes that participants brought to class hurt their performance and could not be controlled by the intervention.

Frustrating Errors. The category *frustrating errors* provided evidence that project defects caused participants ($n = 9$) to feel anger, confusion, and frustration. Pibb recounted troubleshooting logic errors,

I'll start with the feeling of desperation. ... I didn't know what was going on. So I just be sitting there. So so frustrated that I didn't really have the motivation to work anymore. ... The most frustrating type of error is when it's like I guess I call it a soft error [logic error]. ... Those are the most frustrating to work on. ... When you've got like one of those soft errors [logic errors] where it's not breaking the game, but something's wrong, it can get really frustrating, and ... it can get ... rid of a lot of the motivation to work on it. ... When it comes out to like your brutal errors ... that take days to fix, it's really demoralizing.

Julia recalled frustration with errors that could not be fixed quickly, "If I couldn't fix it, it was kind of frustrating." Marlana made a similar comment, "It was like really frustrating because we had no idea how to like solve the problem." Qianna described frustration with runtime errors,

I feel like it would tell me where the problem was. But it wouldn't tell me like what the problem was. ... I knew where it was in the error when the box popped up. It told me what lines, and it had like the actual code that was there, but when I went to it, it just had like the error, and I didn't know how to fix it, so that was frustrating.

Most of the scaffolding the researcher provided during the intervention was an effort to address defects that participants were experiencing with their games. Most participants quit working when the researcher assisted them and were visibly frustrated. Logic errors induced the highest level of frustration in participants because GameMaker provided no information on the error. Most participants were better able to deal with compiler errors and runtime errors because GameMaker identified the problem code. See Figures 4.17, 4.18, and 4.19 for examples of compiler and runtime error messages from GameMaker. The negative feelings that participants experienced caused them to shut down, which negatively affected their performance.

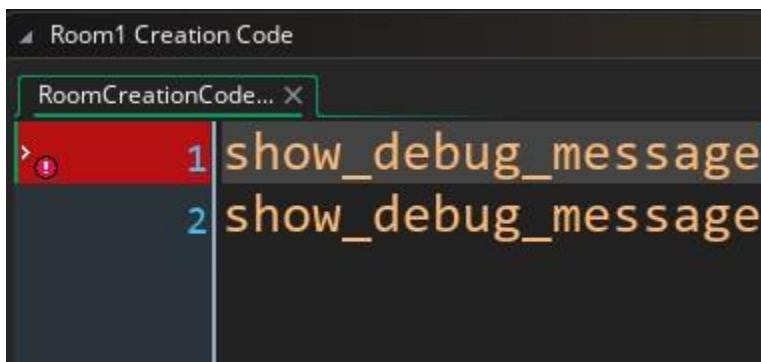


Figure 4.17 Compiler Error Symbol in GameMaker

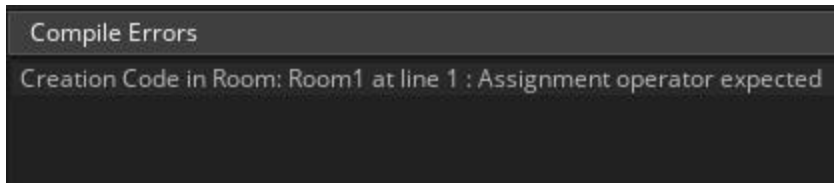


Figure 4.18 Compiler Error Window in GameMaker

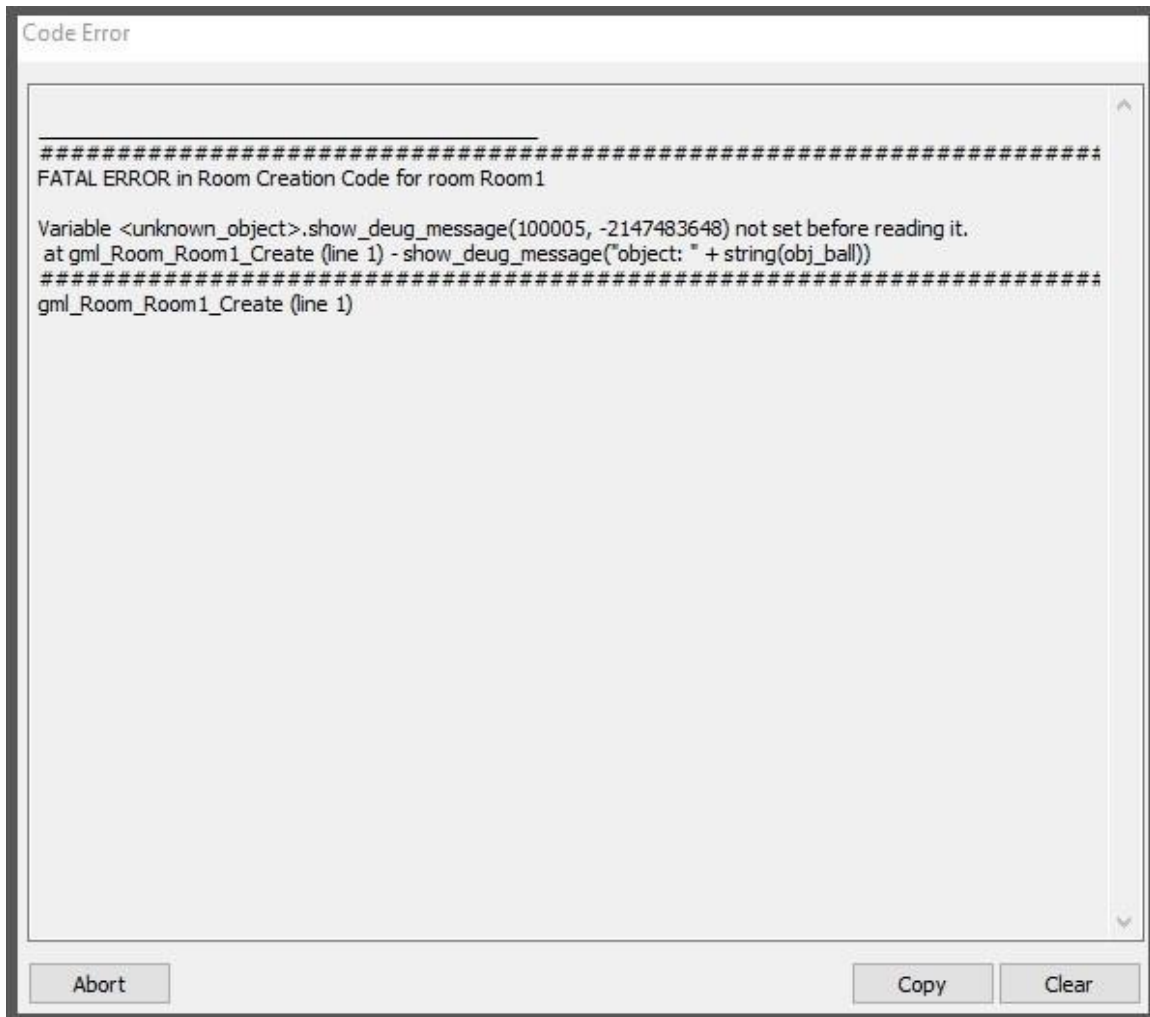


Figure 4.19 Runtime Error Window in GameMaker

General Attitude. The category *general attitude* contained general views that participants offered on how attitudes affected learning and performance. Oakley observed,

I definitely think if you really want to learn about computer science, you will learn something, and you will enjoy it. But I think if you come [to] the class and think, oh, I don't want to do this, you're not gonna learn anything. ... I think if you had a bad attitude, you wouldn't really want to learn anything.

Oakley implied that students decide in advance, though perhaps not consciously, to learn and enjoy material.

Annabelle's view on attitude affecting learning was,

With not wanting to code ... it's something you're not going to want to learn, and that affects your ability to remember it and share it because it's happened [in] multiple different classes. It's not just coding. When you don't want to do it, it's not going to stick, and it's not something you want to ... stick.

Annabelle believed that students would have a reduced ability to remember material without a desire to learn. Annabelle took this a step further by stating that students would not want to remember things that do not interest them. Abigail said, "If I go into the course with a good attitude, then I'm more likely to ... leave the class with better results." Abigail believed that a good attitude would improve performance.

Intervention Positive. The category *intervention positive* provided evidence that the intervention improved participants' attitudes, which was related to performance improvements. When asked if any attitudes toward computer science affected her performance, Aleesha said, "When we got the part [the custom game] that I did enjoy, I worked harder towards what I want to do. Aleesha claimed that because she enjoyed the intervention, she worked harder. Because of the question Aleesha answered, her response implied that her performance improved. Aleesha also said, "When we first

started [before the intervention], it wasn't as fun, but now, when we were working on our custom games that we were more like independent and knew how to do stuff, it was fun."

Aleesha described the intervention as fun in relation to her improved performance.

Aleesha asserted that her performance and attitude were related. Bree linked her attitude to productivity:

When I look forward to it, ... it goes by smoother, like I can actually code and get stuff done. ... If I've come in like tired or like I just don't want to do it, it's just kind of hard to get through it.

Bree reported an improvement in performance when her attitude was positive. When her attitude was negative, she reported difficulty making progress.

Negative Attitude. The category *negative attitude* provided evidence that negative attitudes reduced performance. Oakley noted, "Games like [the] 31 game, I had a bad attitude. So when things like the algorithm was messed up, I just didn't care, and I was like, I'll be fine. I'll just let my score go down." The 31 card game, also known as Scat, was implemented before the intervention. Oakley recognized errors in his algorithms but ignored them because of his negative attitude. Oakley knowingly accepted a reduced grade on the assignment because he didn't care about the assignment.

No Relationship. The category *no relationship* documented one case of a participant who was unable to relate an attitude to performance. When asked if any attitudes affected her ability to learn in the computer science course, Aleesha responded, "Not really." Aleesha did not believe that her attitude was related to her learning. However, as reported above, Aleesha cited specific instances of her attitude changing with her performance.

Rolling with the Punches. The category *rolling with the punches* provided evidence that participants were able to experience errors without a decline in attitude for short periods. Participants were able to tolerate errors for five to 10 minutes before they experienced frustration and other negative emotions. When asked about troubleshooting, Jonie said, “I would get very frustrated if I couldn’t find where the error was. ... I get frustrated, and ... my brain would shut off.” When asked how long it would take for her to feel frustrated, Jonie responded, “Like five to 10 minutes.” Pibb described the troubleshooting process,

So like when I first see an error, I just kind of [feel] neutral about it. ... When it comes to like areas where it actually shows you where it is and what’s going on, it becomes a lot easier to fix because ... you see where it is and you see like the error code, so it tells you what’s going on. ... It really depends on like, how long it takes to fix. ... [If] it only takes... like 10 minutes to fix, it’s not that bad. ... I ... start out pretty fine when it comes to those errors. Like I don’t really care if it takes like 10 minutes.

Jonie and Pibb estimated a 10-minute time frame for troubleshooting defects before they experienced frustration, suggesting that negative attitudes did not accompany simple errors. The ability to quickly locate errors was another critical factor in their troubleshooting experience. GameMaker provided detailed information on the location of syntax and runtime errors, which allowed participants to identify problematic code. Most of the frustration was related to troubleshooting logic errors, which GameMaker was unable to recognize.

Success Multiplier. The category *success multiplier* provided evidence of participants' ($n = 8$) who experienced a cycle of success, followed by an attitude improvement and further success. For example, when Pibb described his experience after fixing "brutal errors," he said, "I'm just really relieved. ... You're like praising the Lord." Similarly, Abegail said, "When I figured out the problem, I feel relieved." Pibb also described success producing feelings of "happiness" and satisfaction. Pibb went on to describe a specific error with implementing step events, "When I was having an issue with step events, ... [that] helped me get more of an understanding of step events. ... I understand how they work now and ... fixing those errors ... helps me understand more." Julia also commented on successful troubleshooting, "Once I fix the error, it was felt rewarding." Pibb and Abegail felt relief after solving a challenging problem. Pibb also gained a better understanding of the concept causing the problem. This feeling of relief and improved understanding plausibly contributed to subsequent successes. Julia reported a rewarding feeling when errors were resolved. Julia added, "Near the end [of the intervention], I started to like understand it more. So my attitude got a little better, and it was a lot more enjoyable." Once again, Julia related success and improved attitude.

After the intervention, Teddie commented on attitudes that influenced his performance and learning,

I enjoy computer science very much, which I think allows me to think about what I need to do better because if I didn't enjoy the course then my creativity for what to do next would be much more limited. ... I think very highly of computer science. I think ... it's really great. I like it a lot, which allows me to ... pay more

attention to what I'm doing and to notice small things that I wouldn't normally be able to notice if I didn't enjoy it.

Monster Fan made a similar comment, “I reckon because of my own like, you know, enjoyment of code, and it’s probably making me more productive.” Teddie credited his enjoyment of computer science for improved thinking, creativity, and focus, which led to learning and performance improvements. Like Teddie, Monster Fan credited his enjoyment of code with increased productivity.

Qualitative Results Summary

Participants reported performance improvements after the intervention. The field notes from observations during the intervention recorded evidence of acquired skills and successful implementation. Participants experienced barriers to success, some of which the intervention could not control. Participants reported positive and negative attitudes after the intervention. There was a large net positive change in participants’ attitudes. Participants reported a feedback loop between attitude and performance. The feedback loop was positive in some cases and negative in others. Failures in performance tended to accompany negative attitudes. Successful performance tended to accompany positive attitudes. Participants reported a positive correlation between attitude and performance. The next chapter will discuss the implications of these results.

CHAPTER 5: DISCUSSION, IMPLICATIONS, AND LIMITATIONS

The purpose of this action research was to implement a digital game development project and describe its effects on the performance and attitudes of eighth-grade students in a required computer science course at South Carolina School District Alpha. This chapter discusses the findings in relation to the research questions and literature on introductory CS courses. Participants' attitudes and content knowledge assessment scores rose significantly after the intervention. A positive correlation was found between participants' post-intervention attitudes and content knowledge assessment scores. Four themes emerged from the data analysis (see Table 4.17). Effects of the intervention on participants were reflected by (a) performance improvements, (b) barriers to success, (c) positive attitudes, and (d) attitude performance feedback loop. Quantitative and qualitative methods were utilized for data collection and analysis. Following is a discussion of the research questions, implications of the research, limitations of the research, and conclusion.

Discussion

The quantitative and qualitative findings were evaluated together to answer this study's research questions. Previous research on PjBL, GDBL, CS instruction, attitudes, and performance helped with understanding the findings. Each research question was discussed in detail below to examine the findings: (1) How does the game development project impact participants' ability to analyze and develop algorithms? (2) What is the effect of the game development project on participants' attitudes toward computer

science? and (3) What is the relationship between participants' attitudes toward computer science and their performance?

Research Question 1: How Does the Game Development Project Impact

Participants' Ability to Analyze and Develop Algorithms?

South Carolina School District Alpha needed to develop instructional methods for students who are required to take an introductory CS course. Algorithm analysis and development, a subset of programming, was the most difficult aspect of CS for students to master (Culic et al., 2019; Erol, 2020; Véghe & Stoffová, 2019). The findings of this study aligned with previous research suggesting that two sets of instructional strategies would improve performance in CS: (a) PjBL and (b) GDBL (Blumenfeld et al., 1991; Çelik et al., 2018; Erümit et al., 2020; Javidi & Sheybani, 2014; Wu & Wang, 2012). Research question one investigated the impact of the intervention on participants' ability to analyze and develop algorithms. Performance improvements related to research question one included (a) algorithm skills and (b) productivity and tool skills.

Algorithm Skills

A content knowledge assessment was used to quantitatively measure the change in participants' ability to analyze and develop algorithms. The content knowledge assessment consisted of a multiple-choice test worth nine points and a performance task worth fifteen points. A pretest was administered before the intervention, and a posttest was administered after the intervention. Twenty-seven of the 28 participants improved their content knowledge assessment scores after the intervention. One participant received the same pretest and posttest scores. There was significant evidence suggesting that the intervention increased assessment scores with a large effect size. The findings

aligned with previous research suggesting that GDBL with GameMaker involving students working in pairs was successful in teaching introductory programming (Javidi & Sheybani, 2014; Jenson & Droumeva, 2016).

Jenson and Droumeva (2016) conducted a study using GDBL with GameMaker in Ontario, Canada. Participants were 67 students in grade six. The intervention was conducted over six days for a total of 15 hours. A 16-item test was administered before and after the intervention. The average score improved from 6.7 before the intervention to 9.3 after the intervention out of 16 total available points. This study confirmed the findings of Jenson and Droumeva that GDBL improves performance.

Amaya Chávez et al. (2020) conducted a quasi-experimental study comparing problem-based learning (PBL) to lecture-based instruction. Participants were first-year undergraduate CS and engineering students. The control group contained 40 participants, and the experimental PBL group contained 39 participants. The percentage of passing scores after the intervention was 60% and 79.49% in the control and experimental groups, respectively. Amaya Chávez et al. found that the difference in passing scores was significant. PBL elements in the Amaya Chávez et al. were similar to the PjBL elements in this study. In both studies, participants (a) analyzed a problem to be solved and identified relevant material from the course, (b) researched learning content required to solve the problem, (c) reported potential solutions and ongoing issues with the problem, and (d) reflected on the solution to the problem. This study confirmed the findings of the Amaya Chávez et al. study: PBL and PjBL improve participant performance.

Qualitative findings confirmed the quantitative findings. Participants successfully decomposed large problems into smaller solvable problems. Evidence of participants decomposing problems was recorded in the field notes and interview transcripts. Evaluating a large problem and separating it into manageable tasks is a fundamental skill in algorithm development (Bowden, 2019; Committee on STEM Education of the National Science and Technology Council, 2018). Seven participants described the intervention as effective in helping them learn to program. Eleven of the 12 participants interviewed were able to describe improvements in their ability to write algorithms to solve problems. Participants described both general and specific algorithm development skills: (a) solving problems independently, (b) handling collisions, (c) making instances move, (d) programming a delay for instance creation, and (e) randomizing which instances were created.

Before the intervention, participants were given predesigned games to modify or implement. Participants were provided with code when algorithms were required for novel game behavior. When presented with a requirement for game behavior identical to that of a previous game, participants were expected to write the algorithm without the aid of solution code. Participants were able to review code from previous games, so they were not forced to develop original algorithms. The results of the content knowledge pretest indicated that the instruction before the intervention was not effective in helping participants learn to analyze and develop algorithms. Participant interviews revealed that many participants felt lost and confused at the beginning of the intervention. Participants reported that the templates provided in games before the intervention did not help with learning to code. They struggled to reproduce algorithmic solutions that had previously

been provided to them. After the intervention, participants gained the ability to develop algorithms without the aid of a template.

Johnson (2017) suggested several instructional strategies to support students during game development: (a) demonstrate coding solutions for common game mechanics and associated programming constructs, (b) assign small programming tasks that require students to extend functionality or correct errors, and (c) train students to read error messages and resolve errors. Deficiencies in participants' knowledge and skills were identified; guided instruction was developed to address the deficiencies. The multiple-choice assessment identified problems with analyzing selection statements and iteration. The performance task revealed that participants could create sprites, objects, and instances in GameMaker; however, participants were unable to implement required behavior: (a) user-controlled instance movement, (c) dynamic instance creation, (d) collision handling, (e) losing lives, (f) scoring, and (g) handling the end of the game. During the intervention, guided instruction was delivered as participants needed it to implement game behavior. When the researcher recognized that several groups would benefit from a lesson, guided instruction was provided to the entire class on the relevant concept. Otherwise, guided instruction was provided to individual groups when they asked the researcher for help or when the researcher recognized that the group required scaffolding.

Scaffolding allows learners to engage in learning and activities that would otherwise be prohibitively challenging (Chen & Law, 2016; Rum & Ismail, 2017). Scaffolding never included writing code for participants. Instead, scaffolding generally included: (a) an explanation of programming concepts, (b) algorithm development

strategies, (c) error message interpretation, (d) error location, (e) testing strategies, and (f) an explanation of why a given implementation was not working as intended. Participants recalled cases where scaffolding was provided. The researcher explained why implementations were not producing desired game behavior and helped participants devise strategies that would work as intended. Participants were responsible for developing algorithms to implement the strategy.

Johnson (2017) conducted a study using GDBL with GameMaker in South East England. Participants were 22 students who were 13 to 14 years old. Johnson found that the constructionist approach of the intervention was not conducive to learning programming concepts for all participants. Johnson found a need for significant scaffolding activities and direct instruction to support constructionist game development. The findings of this study confirm those of the Johnson study. In this study, guided instruction and targeted scaffolding were utilized to support participants' understanding of algorithm design and analysis.

Productivity and Tool Skills

In this study, productivity refers to the ratio of correctly implemented game behaviors to development time. Tool skills refer to skills specific to the GameMaker language, IDE, or API. Because tool skills are related to GameMaker, they would not necessarily improve algorithm analysis and development in another language. Tool skills may be considered a subset of productivity because they help participants develop game behavior efficiently.

Productivity. Code formatting conventions improved productivity. A convention is a practice of writing code that is not enforced by the compiler or a

standards body (Fowler, 2004). Conventions generally improve readability and maintainability for the person or persons working on a software product. Indentation of controlled code was an example of a convention that participants used. Participants indented code that was controlled or belonged to other lines of code. For example, code was indented that was controlled by *if* statements. When looking at large amounts of code, indentation helped participants identify at a glance which code was controlled by other structures.

Participants were given a deliverable schedule consisting of deadlines and required artifacts. Participants created and submitted a design document in the first week of the intervention. The design documents were reviewed to ensure that participants targeted proper functionality for their expected skill set and knowledge. At the end of each week, participants submitted a status report. The researcher used the status reports to help participants deliver a working game. If participants were behind schedule, one of two decisions was made: (a) scaffolding was provided, or (b) functionality was tabled. Scaffolding was provided if participants were struggling with a minor technical detail or conceptual misunderstanding. Functionality was tabled if it jeopardized the project deadline. The researcher watched carefully for scope creep and functionality that was likely beyond the capability of the participants to complete in the given time. Jenson and Droumeva (2016) found that student expectations often exceeded their abilities. Participants were regularly reminded that a working product had to be submitted by the deadline, even if it lacked functionality from the original design specification. Creating a complete software architecture, including formal estimates of code volume and time requirements, was far beyond the scope of the course and intervention (Bass et al., 2006).

It was expected that some design elements would be unreasonable to implement in the given time; participants were advised to remove those design elements. Participants were told that commercial software often shipped without planned features or defect corrections to meet release deadlines.

Tool Skills. Participants acquired numerous tool skills, which helped them implement game behavior. Participants' productivity increased due to their increased familiarity with GameMaker. Baytak et al. (2011) examined eight games developed using GameMaker in their study. Two games utilized modding of existing game templates, and the remaining six games were developed from scratch. Baytak et al. found that students learned to use GameMaker quickly and without much difficulty. This study confirmed that participants were able to use GameMaker quickly. However, when participants used GameMaker incorrectly, they initially had difficulty resolving problems.

Participants learned to recognize common errors that could be made with GameMaker. The misuse of assets was documented in the field notes. An example of this was dynamic instance creation. GameMaker creates instances from objects. Assets, including objects and sprites, are numerically coded, which is invisible to the developer unless explicitly queried. Participants frequently supplied a sprite asset instead of an object asset when creating an instance. If the numerical encoding for the sprite matched the numerical encoding for an object, an instance of the object matching the numerical encoding was created. To the participant, it appeared as if an instance of a random object was being created. Once this problem was explained to participants, they quickly learned to check the asset provided to the instance creation function. Another example of asset

confusion involved sprites and sub-images of sprites. Participants conflated the variables *sprite_index*, which referred to a sprite, and *image_index*, which referred to a sub-image of a sprite. Participants learned to recognize this error and how to correct it quickly.

Participants used GameMaker features to detect and remove defects. Pibb commented that defects were easier to find and correct when GameMaker identified the error, provided the location, and provided information about the nature of the defect. GameMaker identifies compiler errors; the user can click on the error, and the user is taken directly to the code containing the compiler error. GameMaker also provides detailed information on runtime errors, although the user is responsible for navigating to the code containing the error. Participants learned to interpret runtime errors and correct them. The search feature in GameMaker was another useful tool that participants learned to use. Bree remarked that debugging with the search and replace was “very helpful.” The code referenced in GameMaker’s runtime error message could be entered in the search feature, which would take the participant to the relevant code.

Algorithm Trouble and Factors Decreasing Performance

Relatively few participants reported insurmountable problems with developing reasonable algorithms. Expectations were managed during the first week, and scaffolding was provided as needed to assist participants. Participants indicated some surprise at how difficult programming was. Jenson and Droumeva (2016) encountered similar reactions from their study participants.

Multiple factors decreased participants’ performance that were not directly related to the challenge of analyzing and developing algorithms. Participants spent a lot of time on game features that were not relevant to their grading rubric. For example, creating

aesthetic artifacts often received more focus than the algorithm development, which was a problem that Baytak et al. (2011) documented in their study. Participants clearly enjoyed working on images and sound effects, so the work positively affected their attitudes toward the intervention. The tradeoff between the development of technical skills and enjoyment will be discussed further in the implications section.

Eleven incidents of off-task behavior were observed during the intervention and documented in the field notes. Most of the off-task behavior involved smart devices like smartphones. Texting, playing games, and social media activity were the smartphone behaviors documented. Participants attempted to complete other coursework during the intervention. The intervention took place during the first period. Participants who failed to complete homework for a course at a later period would try to complete the homework during the intervention.

Failure to utilize prior work and resources cost many participants valuable time. Ten cases of this were documented. Participants were instructed to keep a “cookbook” or “how-to” document. When the researcher encountered participants struggling with functionality that they had implemented before the intervention, the researcher asked to see their cookbooks. Most participants could not produce them. The researcher would help them locate a previous project where the functionality had been implemented. Some participants never learned to use the GameMaker API. The API should have been a valuable reference for the variables and functionality that GameMaker provided. Proper use of the API should have prevented participants from reproducing functionality that GameMaker had already implemented.

Participants did not test frequently enough, causing them to waste time on the custom game and the performance task portion of the content knowledge assessment. Participants were encouraged to test after every testable element was added; therefore, participants should have been running a test every five to 10 minutes. When participants did not test frequently, they had difficulty locating errors, especially logic errors. Logic errors occurred when unexpected behavior was observed. GameMaker did not provide any information about logic errors. Participants who failed to test regularly had to examine many code segments for the source of logic errors.

A defect detection technique for syntax and runtime errors was demonstrated to participants. The technique involved commenting blocks of code until the defect disappeared. GameMaker ignored commented code. The code was gradually uncommented until the defect was observed again. If used properly, this technique allowed participants to pinpoint the exact line of code causing the defect. When the researcher assisted participants who could not identify the code causing a defect, this technique had not been utilized.

The pressure of a looming deadline negatively affected some participants. Annabelle complained that the researcher was trying to explain concepts, but Annabelle just wanted her errors fixed. Annabelle described a problem that the researcher frequently encountered with students. In the researcher's experience, when students are short on time and have a problem, they stop caring about understanding the problem and just want the problem fixed.

Research Question 2: What Is the Effect of the Game Development Project on Participants' Attitudes Toward Computer Science?

Attitudes should be measured, even if they do not directly impact performance (Simonson, 1979). While acquiring knowledge and skills may be paramount, positive attitudes are desirable. If participants have positive attitudes toward a subject, they are more likely to pursue that subject beyond their current course (Doman et al., 2015; Giannakos, 2014). This study found that PjBL produced positive attitudes, which confirmed previous studies (Blumenfeld et al., 1991; Jumaat et al., 2017; Laakso et al., 2021). Participants demonstrated positive attitudes using GDBL, which confirmed previous studies (Doman et al., 2015; Johnson, 2017; Wu & Wang, 2012). Research question two examined the impact of the intervention on participants' attitudes toward CS. Four categories of attitudes toward CS were examined: (a) learning at school, (b) importance, (c) self-concept, and (d) interest.

A survey measuring attitudes toward computer science was administered to 28 participants before and after the intervention (Shen et al., 2014). The pre- and post-surveys compared the attitudes of participants toward computer science before and after the intervention. The survey contained 26 questions divided into five subscales; each subscale had five or six questions. Questions used a 5-point Likert scale ranging from (1) *strongly disagree* to (5) *strongly agree*. The survey subscales and associated items are shown in Table 4.6.

Doman et al. (2015) utilized a GDBL intervention with GameMaker in the experimental group for their study. Participants were 395 undergraduate students in seven sections of the control class and eight sections of the experimental class. Two of

the data collection instruments were a CS attitude survey and qualitative student perceptions of GameMaker. The GameMaker students reported positive attitudes toward CS for the following: (a) self-concept, (b) interest, and (c) useful. This study confirmed those findings. However, the GameMaker students in the Doman et al. study reported no difference from the control group for (a) relevance of CS to their careers and (b) future plans to use CS. This study did not confirm those findings. This study found improvements in attitudes toward CS importance and future interest following the GDBL intervention.

Çelik et al. (2018) conducted a PjBL intervention study in a vocational school of higher education. Participants were 13 freshman students enrolled in a programming course. This study confirmed the findings of Çelik et al. Participants' attitudes toward learning CS at school improved using PjBL. PjBL sparked participants' interest in CS. Participants in the Çelik et al. study stated that they experienced difficulty with developing computer applications. However, they also claimed that their ability to develop computer applications improved following the intervention. The findings of this study were similar: participants found the game development project challenging, but their self-concept in CS improved.

Attitude Toward Learning Computer Science at School

Participants' attitudes toward learning CS at school improved following the intervention. The learning CS at school subscale of the CS attitude survey was used to quantitatively measure the change in participants' attitudes toward CS at school. Four participants' attitudes toward learning CS at school decreased following the intervention; two had no change; 22 increased. There was significant evidence suggesting that the

intervention improved participants' attitudes toward learning CS at school with a large effect size. The findings of this study confirm those of Çelik et al. (2018) who found that participants had more fun and found instruction less boring with PjBL. The findings of this study align with previous research suggesting that students were likely to appreciate GDBL (R. Reynolds & Caperton, 2011; Swacha et al., 2010; Theodoraki & Xinogalos, 2014). Swacha et al. found that 97% of survey responders would like to participate in game design classes. Theodoraki and Xinogalos found that 95.7% of participants in their study believed that GDBL made programming more interesting.

Qualitative findings confirmed the quantitative findings. Participants found the pre-designed games before the intervention boring. Qianna said that she was not interested in the games before the intervention and did not want to work on them. This was a common sentiment before the intervention. Marlena agreed with Qianna and said that she was bored and disinterested in the games before the intervention. Previous research suggested that modding games produced positive results (Grizioti & Kynigos, 2020; O'Grady-Jones, 2020). Modding is the practice of modifying or adding to an existing codebase. Baytak et al. (2011) chose GameMaker as a development tool in their study because it was conducive to modding. They cited the following advantages of modding compared with developing games from scratch: (a) modding begins with proven game concepts, (b) games are more likely to be engaging, and (c) working prototypes can be developed rapidly.

It was surprising to discover how much the participants in this study disliked modding pre-designed games. Oakley particularly disliked the 31 game. He said that his negative attitude toward the game reduced his motivation and made him apathetic about

errors. He felt like fixing errors would be useless because he did not care about the game. When modding is employed as an instructional tool, the choice of game can have a significant impact.

Participants reported a dramatic increase in enjoyment and excitement during the intervention. Creative freedom was the primary reason for improving attitudes toward CS in school. Participants had a choice in nearly all aspects of their custom game during the intervention, which they appreciated and enjoyed. Participants appeared to attach a sense of ownership to their custom games and took pride in the results. Participants' pride in their games was particularly evident during the playtesting phase of the intervention. They were animated and excited to show their peers the features of their games. These findings confirmed those of Doman et al. (2015). Doman et al. prompted participants to describe something interesting about their experience with GDBL and GameMaker. Participants reported enjoying the creative aspect of designing and developing a game. They also enjoyed playing their games.

Importance of Computer Science

Participants' attitudes toward the importance of CS improved following the intervention. Alvarez et al. (2019) found a correlation between perceived value and performance. If perceived value and performance are related, importance of CS may offer insights into CS performance. The importance of CS subscale of the CS attitude survey was used to quantitatively measure the change in participants' attitudes toward the importance of CS. Six participants' attitudes toward the importance of CS decreased following the intervention; one had no change; 21 increased. There was significant

evidence suggesting that the intervention improved participants' attitudes toward importance of CS with a medium effect size.

Qualitative findings confirmed the quantitative findings. By the end of the intervention, participants felt that computer science was an important field with significant personal or societal impacts. Short weekly discussions were held during the intervention regarding current events in computing and the global impact of computing. Computational science was discussed to stress the synergy of computing with other disciplines. Participants were captivated by the possible impact of automation on the job market. Participants felt that CS had the potential to improve job performance and improve people's lives. They also felt that automation could cause mass unemployment. Beneficial and harmful effects of computing were cited, but all participants felt that CS was an important field.

Self-Concept in Computer Science

Participants' attitudes toward self-concept in CS improved following the intervention. For this study, self-concept was defined as participants' beliefs regarding their competence in mastering computer science (Shen et al., 2014). Previous research has found a significant correlation between self-concept and performance (Alvarez et al., 2019; Gurer et al., 2019). The self-concept in CS subscale of the CS attitude survey was used to quantitatively measure the change in participants' self-concept in CS. One participant's attitude toward self-concept in CS decreased following the intervention; two had no change; 25 increased. There was significant evidence suggesting that the intervention improved participants' attitudes toward self-concept in CS with a large effect size. The findings of this study aligned with previous research suggesting that GDBL

enhanced the self-efficacy of participants (Jenson & Droumeva, 2016; Laakso et al., 2021; Theodoraki & Xinogalos, 2014).

Qualitative findings confirmed the quantitative findings. Participants made positive declarations of their self-concept in CS. They expressed enjoyment and understanding of CS. Pibb had some prior programming experience with modding games and expressed a high self-concept in CS. Teddie and Bree had no prior programming experience and were struggling with the course before the intervention. Teddie and Bree said that they no longer felt confused after the intervention. Successfully completing a custom game without any starter code provided a strong boost to participants' self-concept. While the researcher provided scaffolding to participants, the researcher never wrote code for them. As a result, they were responsible for every working part of their custom games, which increased their self-concept in CS.

Sparked Interest in Computer Science

Participants' interest in CS increased following the intervention. The findings of this study aligned with previous research suggesting that GDBL increased future participation in CS (Javidi & Sheybani, 2014; Laakso et al., 2021; Theodoraki & Xinogalos, 2014). Baytak et al. (2011) found that GDBL with GameMaker resulted in students continuing to create games after the end of the course. Two subscales of the CS attitude survey were used to quantitatively measure the change in participants' interest in CS: (a) learning CS outside of school and (b) future participation in CS.

Participants' attitudes toward learning CS outside of school improved following the intervention. The learning CS outside of school subscale of the CS attitude survey was used to quantitatively measure the change in participants' attitudes toward CS

outside of school. Five participants' attitudes toward learning CS outside of school decreased following the intervention; six had no change; 17 increased. There was significant evidence suggesting that the intervention improved participants' attitudes toward learning CS outside of school with a medium effect size.

Participants' attitudes toward future participation in CS improved following the intervention. The future participation in CS subscale of the CS attitude survey was used to quantitatively measure the change in participants' attitudes toward future participation in CS. Two participants' attitudes toward future participation in CS decreased following the intervention; nine had no change; 17 increased. There was significant evidence suggesting that the intervention improved participants' attitudes toward future participation in CS with a large effect size.

Qualitative findings confirmed the quantitative findings. Marlena described an error with her custom soccer game. She researched a solution to the problem at home and implemented the solution. She added that researching programming solutions at home was something that she had not done before the intervention. Pibb also described working on his custom game at home. Pibb speculated that he was "bad at math" because he did not work on it at home but "good at programming" because he liked to program games at home. In both cases, participants described working on their custom games outside of school. No participant related a single instance of working on a game at home before the intervention.

Participants expressed future interests in computer science. Qianna expressed interest in taking CS in high school because she found the game development class interesting, and she thought it would help with her future career. Jonie stated that she had

no interest in CS before the class. After the intervention, she said that she might want to study CS in college because “it was fun.” Teddie noted that he planned to work on CS “kind of like a hobby that I will just do in my free time.” Participants declared interest in future computer science classes in high school and college and interests as hobbyists. Successfully generating a software product made participants more likely to pursue computer science in the future.

Research Question 3: What is the Relationship Between Participants’ Attitudes Toward Computer Science and Their Performance?

Research question three examined the relationship between participants’ attitudes and their performance. Attitudes toward computer science and performance in computer science were positively correlated following the intervention. Negative attitudes were associated with errors, and positive attitudes were associated with successes.

A composite survey score was calculated for each participant by taking an unweighted average of the five subscale means on the post-survey. Pearson’s r was calculated for the composite survey scores ($M = 2.81$, $SD = 0.97$) and the posttest scores ($M = 16.14$, $SD = 5.25$) to measure the linear correlation between participants’ attitudes toward computer science and their performance. Significant evidence showed that post-survey and posttest scores were moderately positively correlated.

Alvarez et al. (2019) studied the relationship of three indicators to academic performance in an introductory programming course: (a) implicit theories of intelligence, (b) error orientation, and (c) student attitudes toward computer programming. Participants were 242 freshman students. Student attitudes toward computer programming had the strongest correlation to performance, specifically perceived self-

efficacy and perceived value. This study confirmed the findings of Alvarez et al. However, the population of this study was too small to examine the correlations of the attitudes toward CS survey subscales with performance. Perceived self-efficacy in the Alvarez et al. study mapped to self-concept in this study. Perceived value in the Alvarez et al. study mapped to future participation and importance in this study. In another confirming study, Jenson and Droumeva (2016) found that confidence in problem-solving was the attitude with the strongest relationship to performance in computer programming.

Gurer et al. (2019) studied factors related to pre-service CS teachers' attitudes toward computer programming. Participants were 119 university students. Their study found that attitudes toward computer programming were significantly correlated with (a) performance mean in computer programming courses, $r(119) = .47$; (b) self-concept in CS, $r(119) = .74$; and (c) perceived learning, $r(119) = .71$. This study confirmed the findings of Gurer et al. that attitudes toward CS and performance are positively correlated. This study found a moderate positive correlation, while Gurer et al. found a low positive correlation. However, this study treated self-concept as a subscale of attitudes, whereas Gurer et al. treated self-concept as a separate factor, which had a high positive correlation with attitudes. Gurer et al. measured the following correlations with attitudes toward CS, which were not measured in this study: (a) perceived learning, (b) grade level, (c) high school type, and (d) gender. Only perceived learning was found to be significantly correlated with attitudes toward CS.

Frustration was a common reaction to errors during the intervention. Participants expressed the greatest feelings of frustration and consternation with logic errors. Logic

errors demoralized participants and reduced participants' motivation. Logic errors manifested as undesired game behavior during playtesting, but GameMaker did not assist participants with detecting or resolving the error. The researcher quickly assisted participants with compiler and runtime errors because they were conspicuous. Compiler and runtime errors were visible on participants' screens. Participants often had to notify the researcher of logic errors before the researcher addressed them. Participants sometimes notified the researcher explicitly of logic errors with a raised hand or verbal call for assistance. Other times, the researcher recognized that participants were struggling with logic errors because they were noticeably frustrated or off-task.

Participants were able to persist with troubleshooting a given error for about 10 minutes. After 10 minutes of troubleshooting, participants typically stopped working. Some participants reported frustration with compiler and runtime errors. They complained that GameMaker provided information on the location of the error but not sufficient information on what was causing the error or how to fix it. Most of the scaffolding that the researcher provided during the intervention was an effort to address defects that participants were experiencing with their games. The majority of participants had quit working and were visibly frustrated when the researcher assisted them with errors. The negative feelings that participants experienced were associated with a decrease in performance.

Some level of frustration and failure was desirable. Psychology and neuroscience research suggested that failure and emotional response could enhance learning; failures of mental models to correctly map to reality prime the brain for adaptation (Bjork & Bjork, 2011; Franklin & Grossberg, 2017; Richland et al., 2009; Tyng et al., 2017). Participants

who expressed frustration but continued working tended to experience more success than those who quit quickly. Pibb handled frustration well and was not observed shutting down. At worst, he would table a feature and work on another part of his game while waiting for assistance. Frustration caused other participants to quit. Qianna was one of several participants who would quit troubleshooting after about 10 minutes. When frustration contributed to participants quitting, their performance suffered. Qianna showed no improvement in the content knowledge assessment score from pretest to posttest. Marlena was a participant who would persist in troubleshooting errors or work on other features while awaiting assistance. She reported feelings of elation after solving difficult problems. Marlena increased her content knowledge assessment score by 12 points from pretest to posttest, which was higher than the mean difference of 8.21.

Participants described successes leading to positive attitudes. After successful troubleshooting, participants reported feelings of relief, happiness, satisfaction, and reward. They also reported improved understanding of concepts. These positive feelings and improved understanding plausibly contributed to subsequent successes. Julia reported a rewarding feeling when errors were resolved. Julia observed that near the end of the intervention, her understanding increased, which improved her attitude and enjoyment. Julia and other participants associated success with improved attitudes.

Other participants implied that their attitudes influenced their performance. Teddie said that his enjoyment of computer science increased his thinking, creativity, and attention. Monster Fan claimed that his enjoyment of computer science made him more productive. Once again, participants associated success with positive attitudes.

The findings of this study align with the claim of Blumenfeld et al. (Blumenfeld et al., 1991) that for projects to promote learning, students should have a high self-concept in the knowledge and skills required by the project. Participants reported a feedback loop between attitude and performance. For this study, feedback loop was defined as a system that returns a portion of the output signal of the system to the input of the system (Spencer, 1994). The feedback loop was positive in some cases and negative in others. Failures in performance were associated with negative attitudes unless a successful correction followed the failure. Successful performance was associated with positive attitudes. Participants reported a positive correlation between attitude and performance. An attempt was made to provide guided instruction and scaffolding to participants as they needed it. The intent was to produce as many successes as possible and increase self-concept. For participants like Annabelle and Qianna, the scaffolding was unsuccessful or too late.

Implications

This research holds implications for me as an instructor, personnel charged with computer science instruction, and other researchers investigating methods of computer science instruction. In the following section, three implications are discussed: (a) personal implications, (b) recommendations for computer science curriculum in South Carolina School District Alpha, and (c) implications for future research.

Personal Implications

This study revealed several implications for my role as a computer science teacher: (a) PjBL works, (b) keep project groups small, (c) build a strong foundation for

open design, and (d) continue to integrate guided instruction with discovery-based learning.

PjBL Works

The intervention was much more successful in improving learners' performance and attitudes than were previous units in the class. The focus on PjBL was the primary distinguishing feature of the intervention. The following are essential project design elements in what the Buck Institute for Education calls gold standard PjBL: (a) challenging problem or question, (b) sustained inquiry, (c) authenticity, (d) student voice & choice, (e) reflection, (f) critique & revision, and (g) public product (Larmer et al., 2015). Previous research also includes the opportunity to work collaboratively as an essential PjBL element (Jumaat et al., 2017). Elements of PjBL should be introduced as early as possible using an understanding of learners' prior knowledge (Jumaat et al., 2017).

Choice was the most influential PjBL design element on participants' attitudes; evidence for this claim was derived from participants' responses in interviews. Introducing choice caused participants to support most of the other PjBL design elements. Participants proposed projects that were generally too challenging. The researcher had to scale back some of the proposed functionality. Because of their interest in their custom games, participants eagerly engaged in sustained inquiry, reflection, and critique and revision. Participants created games that they wanted to play, so the games had personal authenticity. The custom games became public products because the games were distributed to participants' peers. Sometimes student enjoyment comes at the cost of academic rigor. PjBL may be one method of increasing both.

Keep Project Groups Small

Improvements in teamwork skills were cited by participants, which was suggested by previous research (Amaya Chávez et al., 2020). Groups of two worked well. Large groups may not work well because they involve learners assuming different team roles. Learners assuming roles that reduced their time developing algorithms would be undesirable. Previous research found that the level of learning in technical competencies was related to group roles (Laakso et al., 2021; R. B. Reynolds, 2016).

Build a Strong Foundation for Open Design

Creating an open design experience can be overwhelming if introduced before a significant skill set is developed (Jenson & Droumeva, 2016). The intervention would probably not be successful as the first exposure to programming. Game modding has been successfully implemented in introductory programming courses (Kynigos & Grizioti, 2020), but participants in this study disliked it. Identifying predesigned games that interest students will be a high priority for future class iterations. Allowing students to choose among several games to modify may improve enjoyment. Playtesting others' games was one of the most enjoyable aspects of the intervention for participants. Playtesting could be introduced into modding activities if students were modding different games.

Forcing students to practice behaviors that will benefit them in the future may be necessary. With one instructor and 28 students, the students must have the ability to research solutions. Many participants in this study did not utilize the suggested coding cookbook to track solutions and procedures. As a result, the researcher spent significant

time helping participants resolve problems that they had solved before. Keeping a detailed coding cookbook should be a graded activity.

The API is an invaluable tool for researching the functionality and behavior of the IDE. The API should be used as a reference for syntax, variable meaning, and function behavior. More guided instruction should be provided to train students on API use. Optimizing the use of the API would reduce students' dependency on the instructor.

Because successful performance was associated with positive attitudes in this study, successes should come frequently and as early as possible. Topalli and Cagiltay (2018) found that using a block-based programming language instead of a text-based programming language allowed for an algorithm-first approach. Removing syntax errors as a barrier to programming allowed students to focus on algorithm development much earlier than with a text-based language. In addition to the text-based GameMaker Language used in this study, GameMaker has a block-based language called Drag and Drop. Students should start programming with a block-based language so that frustration with syntax errors is not their first experience with programming, which should produce earlier success with algorithm development.

Integrate Guided Instruction and Discovery-Based Learning

Guided instruction was a successful instructional strategy (Clark et al., 2012; Johnson, 2017; Kirschner et al., 2006). Continued experimentation with the proper mix of guided instruction and discovery-based learning will continue. The findings of this study aligned with previous research suggesting that frustration may be inversely related to expertise during discovery learning (Clark et al., 2012; Kirschner et al., 2006; Kirschner & De Bruyckere, 2017; R. Reynolds & Caperton, 2011). In this study,

participants with less expertise required greater monitoring and scaffolding than those with more expertise. Participants identified two areas for improvement in the interviews. First, participants should master technical vocabulary before encountering it in scaffolding or project instructions. Second, better instruction on the use of the API should be provided. The API was an invaluable tool for participants in the problem-solving process. Without it, they were overly reliant on instructor assistance for developing algorithms.

Recommendations for Computer Science Curriculum in South Carolina School District Alpha

The South Carolina Department of Education has CS and digital literacy standards for Kindergarten through grade eight that all students are supposed to learn (South Carolina Department of Education, 2017). In South Carolina School District Alpha, students take an introductory computer science course in grade seven. The participants in my study were in the STEM magnet program. Participants showed no indication of retaining an ability to analyze and develop algorithms when they arrived in my class.

South Carolina School District Alpha should be prepared for the South Carolina Department of Education to create an assessment process for ensuring that CS standards are being taught. The South Carolina Department of Education may choose to create a CS end-of-course examination or add CS as a subject in the South Carolina Palmetto Assessment of State Standards. South Carolina School District Alpha should develop an evaluation system to measure the degree to which the CS and digital literacy standards are integrated into the curriculum. Professional development should be offered to assist

teachers with including CS and digital literacy standards in their instruction. Considering the challenges STEM magnet participants experienced with CS, significant challenges should be expected with teaching CS to the entire student body.

Finally, South Carolina School District Alpha should carefully consider the level of rigor required in the introductory computer science course. Most students in the district currently receive one-half CS credit in grade seven and one-half CS credit in grade eight. Gifted and Talented students take algebra one in grade eight, and other students take algebra one in grade nine. As stated earlier, programming ability is related to mathematical ability (Cetin & Andrews-Larson, 2016; João et al., 2019). College Board lists algebra one as a prerequisite to AP CSP, an introductory computer science course (CollegeBoard, 2020b). Students may not have the necessary mathematical sophistication to complete an introductory computer science course before completing algebra one.

Implications for Future Research

Findings in this study suggest implications for future research in CS instruction: (a) refine PjBL goals and implementation for CS, (b) test nontraditional CS instructional methods against each other, and (c) explore the long-term impacts of CS attitude changes.

Refine PjBL Goals and Implementation for CS

PjBL has many components and goals, which makes the impacts of the individual components challenging to measure (Helle et al., 2006). Computer science activities aligned with PjBL goals should be well-defined. Methods for measuring students' potential for handling PjBL elements should also be defined. As an instructor, finding the appropriate level of challenge for a problem should be more algorithmic and less art.

The amount of structure that is optimal for students is also difficult to determine. Essential project design elements for CS should be tested individually in controlled experimental tests (Larmer et al., 2015; Sweller et al., 2007). This would help instructors new to PjBL prioritize the addition of PjBL design elements to their instruction.

Test Nontraditional CS Instructional Methods Against Each Other

GDBL should be tested against other nontraditional instructional strategies. The relative contributions to the effect size of GDBL and PjBL were not measured in this study. Projects other than digital games should be tested. For example, Erol (2020) found that robotic programming improved students' attitudes toward programming. The computer science intervention studies reviewed in the literature for this study used traditional computer science instruction as a control. A reasonable assumption would be that students with different backgrounds and interests would respond differently to GDBL and a robotics programming curriculum. Variation exists within the realm of GDBL. Modding predesigned games was implemented successfully in other studies, but participants in this study did not respond well to modding. Multiple development environments and programming languages can also be used for GDBL.

Explore the Long-Term Impacts of CS Attitude Changes

Doman et al. (2015) found that short-term attitudes toward CS improved, but long-term attitudes did not. One of the goals of a successful computer science course should be to encourage students to enroll in future computer science courses. An understudied question is whether attitude changes last long enough for students to continue studying CS. Are there any future behavioral changes attributable to attitude changes in a CS course?

Tradeoffs between learning activities that align with course standards and activities that improve attitudes can exist. For example, participants in this study immensely enjoyed developing aesthetic elements of their games. However, graphics design and sound effects had no relevance to course standards. Goals in a given course may be to (a) maximize learning and skills related to standards, (b) prepare students for related coursework and professional work, and (c) encourage students to continue in a field of study. Considering the positive correlation between programming experience and self-efficacy, encouraging additional CS coursework is desirable (Tsai et al., 2019). Braga et al. (2014) found that their measure of instructor effectiveness was negatively correlated with students' evaluations of instructors. Braga et al. compared future outcomes of students in related coursework to measure instructor effectiveness. This raises a question regarding the utility of using students' attitudes as a metric for judging the quality of a course and may help explain findings suggesting no relationship between attitudes and performance (Cetin & Andrews-Larson, 2016; Gurer et al., 2019). Further research should explore the relative long-term impacts of knowledge acquisition and attitude changes in a given course.

Limitations

This study had limitations that could be addressed by future research. The following limitations are discussed below: (a) study design and methodology, (b) participants, and (c) researcher.

Study Design and Methodology

This was an action research study. As such, the results may not be generalizable. The quantitative data collection utilized a one-group pretest-posttest design (Creswell &

Creswell, 2018). The identical assessment was administered for the pretest and posttest with six weeks between administrations. Test familiarity is a threat to internal validity when using identical assessments for pretest and posttest. The pretest was not recorded in participants' grades for the course, but the posttest was. It was common practice for unit pretests to be formative (ungraded) and unit posttests to be summative (graded). Participants may have been more motivated on the posttest than the pretest.

There was an alignment problem with the survey subscale and interview questions regarding the importance of computer science. The survey subscale emphasized the benefits of CS (see Appendix E). The interview questions asked about the beneficial and harmful effects of CS without prioritizing either (see Appendix F). The quantitative data from the survey subscale and the qualitative data from the interviews regarding the importance of CS measured slightly different attitudes. Based on the interview responses, participants' attitudes toward the importance of CS were heavily influenced by researcher-led discussions of the beneficial and harmful effects of CS. Participants should have been directed to research the global impacts of CS for themselves.

The Kuder-Richardson Formula 20 score for the multiple-choice portion of the content knowledge assessment was low. *KR20* for the pretest was 0.53, and *KR20* for the posttest was 0.18. The quantitative claims made in this study involving the content knowledge assessment should be interpreted in the context of the low *KR20* scores. *KR20* was not viewed as a disqualifying reliability metric for the multiple-choice portion of the content knowledge assessment in this study. This study was interested in the gain scores from pretest to posttest. *KR20* is important for tests designed to discriminate between test takers when an assessment is administered once (Anselmi et al., 2019; Ebel,

1967; Mertler, 2019; Osadebe, 2015). Low variance and high test item difficulty may have contributed to the low *KR20* score. Because a small number of test items will likely have a low variance, *KR20* is expected to be low for assessments with a low number of test items. The Spearman-Brown prophecy formula is .85 when the number of test items is 45; the Spearman-Brown prophecy formula is used to predict reliability when test items of similar difficulty are added or removed (de Vet et al., 2017). Therefore, the reliability of the multiple-choice assessment could probably be improved by adding additional test items of similar difficulty.

Participants

The participants were a group of eighth-grade students in the STEM magnet program. This was a purposive sample, not a random sample. The participants were unlikely to be representative of the general student body. The participants had to apply to the STEM magnet program and were selected for previous academic achievements. They were likely motivated by factors such as grades and parental pressure that were not measured by the study.

While not formally surveyed, most participants indicated they were not particularly interested in STEM. The participants stated that their motivating factors for applying to STEM were to be in classes with other high-achieving students and to avoid the behavior problems common to classes outside the magnet program. For example, Julia stated that enrollment in the magnet program, which separated her from the general student population, was the only reason she attended South Carolina School District Alpha instead of a private school. When the STEM magnet program ended after grade eight, Julia transferred to a private school.

The participants did not exhibit disruptive behaviors that would probably have manifested with students in a college prep (CP) course. CP courses at South Carolina School District Alpha are less challenging than honors and AP courses. CP courses tend to have more incidents of disruptive and noncompliant behavior. When the participants in the study became frustrated with assignments or were waiting for assistance from the instructor, they exhibited off-task behavior. The off-task behavior was usually not disruptive to other participants. Therefore, the negative consequences of frustrating tasks were confined to the participants who experienced the frustration. In contrast, the disruptive behavior resulting from frustrating tasks in a CP class would probably affect multiple students.

Annabelle was the only participant interviewed who expressed strong negative criticisms and attitudes regarding the intervention. Therefore, this study did not obtain saturation concerning participants' perceptions of deficiencies in the intervention. Ideally, more participants would have been interviewed. Unfortunately, more interviews could not be conducted due to unreturned assent forms and scheduling problems. Roughly half of the negative attitude codes were attributed to Annabelle. Negative attitudes could have been overrepresented in this study due to the equal inclusion of codes from Annabelle's interview. Annabelle was an outlier relative to the other participants who were interviewed. However, Annabelle's criticisms may have been shared by some of the participants who were not interviewed. Voicing criticisms to an authority figure is intimidating. Therefore, other participants who held negative opinions of the intervention may have avoided the interview.

Member checking was performed, but no participants responded to email. South Carolina School District Alpha provided Gmail accounts for their students. Participants' district email addresses were used for member checking. Cleaned interview transcripts were emailed to the matching interviewee to check for accuracy (see Figure 4.7). The abstract and description of themes (see Table 4.17) were emailed to all participants for comment. It is not unusual for South Carolina School Alpha students to ignore their district email. Member checking may have been improved by utilizing participants' personal email addresses. However, the researcher has a policy of not using personal accounts to communicate with students. After the intervention and game development course ended, Pibb enrolled in AP CSP where the researcher was the instructor. The major findings of the study were discussed verbally with Pibb. Pibb expressed no disagreements with the major findings.

Researcher

The researcher's biases influenced the intervention, data collection, and data analysis. The content knowledge assessment was constructed by adapting practice test items from CollegeBoard's AP Computer Science Principles exam. The researcher provided instruction to help participants perform well on the assessment. Results on a test to which the researcher did not have access may have been very different. The researcher used personal judgment regarding when and how much scaffolding was provided to participants. When reading interview transcripts, the researcher recognized many missed opportunities to ask follow-up and clarifying questions. As a result, the information collected in the interviews could have been much more detailed. The interview questions may have been leading. The diction used in the questions prompted

participants to speculate on causal relationships between attitude and performance. The participants may have disregarded instructions to be completely honest in an effort to please the researcher, who was also the instructor. The quantitative results were known to the researcher when the qualitative analysis was performed. Therefore the quantitative results probably influenced the qualitative analysis.

Conclusion

François Chollet, the Keras project lead, recently predicted, “Within 10-20 years, nearly every branch of science will be, for all intents and purposes, a branch of computer science. Computational physics, comp chemistry, comp biology, comp medicine... Even comp archeology. Realistic simulations, big data analysis, and ML everywhere.” Knowledge of CS will be required by an increasing number of fields. The degree to which choice in game design affected participants cannot be overstated. The change in mood was palpable when participants began creating their custom games with complete freedom to build any game of their choosing. Participants were much more receptive to guided instruction when it directly applied to a problem that they cared about solving. Project-based learning with an emphasis on student voice and choice is an instructional practice with enormous potential to improve attitudes and performance in CS.

REFERENCES

- Abdul Jabbar, A. I., & Felicia, P. (2015). Gameplay engagement and learning in game-based learning: A systematic review. *Review of Educational Research*, 85(4), 740–779. <https://doi.org/10.3102/0034654315577210>
- Adams, K. A., & Lawrence, E. K. (2019). *Research methods, statistics, and applications* (2nd ed.). Sage Publications, Inc.
- Agee, J. (2009). Developing qualitative research questions: A reflective process. *International Journal of Qualitative Studies in Education*, 22(4), 431–447. <https://doi.org/10.1080/09518390902736512>
- Akcaoglu, M. (2014). Learning problem-solving through making games at the game design and learning summer program. *Educational Technology Research and Development*, 62(5), 583–600. <https://doi.org/10.1007/s11423-014-9347-4>
- Al-Makhzoomy, A. K. (2018). *Effect of game development-based learning on the ability of information technology undergraduates to learn computer and object-oriented programming* (Publication No. 10973970) [Doctoral Dissertation, Wayne State University]. ProQuest Dissertations & Theses Global.
- All, A., Nunez Castellar, E. P., & Van Looy, J. (2015). Towards a conceptual framework for assessing the effectiveness of digital game-based learning. *Computers & Education*, 88, 29–37. <https://doi.org/10.1016/j.compedu.2015.04.012>
- Alturki, R. A. (2016). Measuring and improving student performance in an introductory programming course. *Informatics in Education*, 15(2), 183–204.

<https://doi.org/10.15388/infedu.2016.10>

- Alvarez, C., Wise, A., Altermatt, S., & Aranguiz, I. (2019). Predicting academic results in a modular computer programming course [Paper presentation]. In E. Scheihing, J. Guerra, V. Henriquez, C. Olivares, & P. J. Munoz-Merino (Eds.), *LALA 2019 - Proceedings of the 2nd Latin American Conference on Learning Analytics* (Vol. 2425, pp. 21–30). <http://ceur-ws.org/Vol-2425/paper22.pdf>
- Amaya Chávez, D., Gámiz-Sánchez, V.-M., & Cañas Vargas, A. (2020). Problem-based learning: Effects on academic performance and perceptions of engineering students in computer sciences. *Journal of Technology and Science Education*, 10(2), 306–328. <https://doi.org/10.3926/jotse.969>
- An, Y. J. (2016). A case study of educational computer game design by middle school students. *Educational Technology Research and Development*, 64(4), 555–571. <https://doi.org/10.1007/s11423-016-9428-7>
- Anderson, M., & Jiang, J. (2018). Teens, social media & technology 2018. In *Pew Research Center* (Issue May). <https://www.pewresearch.org/internet/2018/05/31/teens-social-media-technology-2018/>
- Anselmi, P., Colledani, D., & Robusto, E. (2019). A comparison of classical and modern measures of internal consistency. *Frontiers in Psychology*, 10(2714), 1–12. <https://doi.org/10.3389/fpsyg.2019.02714>
- Balmes, I. (2017). Correlation of mathematical ability and programming ability of the computer science students. *Asia Pacific Journal of Education, Arts and Sciences*, 4(3), 85–88. <https://doi.org/10.13140/RG.2.2.22763.23849>

- Banister, S. (2007). Ethical issues and qualitative methods in the 21st century: How can digital technologies be embraced in the research community? *Journal for Ethnographic and Qualitative Research, 1*, 1–10.
- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48. <https://doi.org/10.1145/1929887.1929905>
- Bass, L., Clements, P., & Kazman, R. (2006). *Software architecture in practice* (2nd ed.). Addison-Wesley.
- Baytak, A., Land, S. M., & Smith, B. K. (2011). Children as educational computer game designers: An exploratory study. *Turkish Online Journal of Educational Technology, 10*(4), 84–92.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile software development*. <http://agilemanifesto.org/>
- Bernard, H. R., Wutich, A., & Ryan, G. W. (2017). Finding themes. In *Analyzing qualitative data: Systematic approaches* (pp. 101–123). Sage Publications.
- Bjork, E. L., & Bjork, R. A. (2011). Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. In M. A. Gernsbacher, R. W. Pew, L. M. Hough, & J. R. Pomerantz (Eds.), *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society* (pp. 55–64). Worth Publishers.
- Bloomberg, L. D., & Volpe, M. (2015). *Completing your qualitative dissertation: A*

roadmap from beginning to end (3rd ed.). SAGE Publications, Inc.

<https://doi.org/10.4135/9781452226613.n3>

Blumenfeld, P. C., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M., & Palincsar, A.

(1991). Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist*, 26(3–4), 369–398.

<https://doi.org/10.1080/00461520.1991.9653139>

Bowden, H. M. (2019). Problem-solving in collaborative game design practices :

Epistemic stance , affect , and engagement. *Learning, Media and Technology*, 44(2), 124–143. <https://doi.org/10.1080/17439884.2018.1563106>

Boyle, E. A., Hainey, T., Connolly, T. M., Gray, G., Earp, J., Ott, M., Lim, T., Ninaus,

M., Ribeiro, C., & Pereira, J. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. *Computers & Education*, 94, 178–192.

<https://doi.org/10.1016/j.compedu.2015.11.003>

Braga, M., Paccagnella, M., & Pellizzari, M. (2014). Evaluating students' evaluations of professors. *Economics of Education Review*, 41, 71–88.

<https://doi.org/10.1016/j.econedurev.2014.04.002>

Campbell, M. (2021). *Random name generator*. Behind the Name.

<https://www.behindthename.com/random/>

Carnegie Learning. (2021). *Zulama: Computer science education for every classroom*.

<https://www.carnegielearning.com/solutions/applied-sciences/computer-science/>

Çelik, H. C., Ertas, H., & İlhan, A. (2018). The impact of project-based learning on achievement and student views: The case of AutoCAD programming course.

Journal of Education and Learning, 7(6), 67–80.

<https://doi.org/10.5539/jel.v7n6p67>

Cetin, I., & Andrews-Larson, C. (2016). Learning sorting algorithms through visualization construction. *Computer Science Education*, 26(1), 27–43.

<https://doi.org/10.1080/08993408.2016.1160664>

Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, 12(2), 1–14. <https://doi.org/10.30935/cedtech/8247>

Chen, C. H., & Law, V. (2016). Scaffolding individual and collaborative game-based learning in learning performance and intrinsic motivation. *Computers in Human Behavior*, 55, 1201–1212. <https://doi.org/10.1016/j.chb.2015.03.010>

Clark, R. E., Kirschner, P. a, & Sweller, J. (2012). Putting students on the path to learning: The case for fully guided instruction. *American Educator*, 36(1), 6–11.

Code.org, CSTA, & ECEP Alliance. (2022). *2022 state of computer science education: Understanding our national imperative*. <https://advocacy.code.org/stateofcs>

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Routledge Academic.

CollegeBoard. (2020a). *AP computer science A: Course and exam description*. AP Central. <https://apcentral.collegeboard.org/pdf/ap-computer-science-a-course-and-exam-description.pdf>

CollegeBoard. (2020b). *AP computer science principles: Course and exam description*. AP Central. <https://apcentral.collegeboard.org/pdf/ap-computer-science-principles-course-and-exam-description.pdf>

- Committee on STEM Education of the National Science and Technology Council.
- (2018). *Charting a course for success: America's strategy for STEM education*. White House Office of Science and Technology Policy.
- <https://trumpwhitehouse.archives.gov/wp-content/uploads/2018/12/STEM-Education-Strategic-Plan-2018.pdf>
- Computer science discoveries ('19- '20)*. (2019). <https://studio.code.org/courses/csd-2019>
- Creswell, J. W. (2017). *Qualitative inquiry and research design: Choosing among the five traditions*. SAGE Publications, Inc.
- Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed method approaches* (5th ed.). SAGE.
- Cucinelli, G., Davidson, A., Romero, M., & Matheson, T. (2018). Intergenerational learning through a participatory video game design workshop. *Journal of Intergenerational Relationships*, 16(1–2), 146–165.
- <https://doi.org/10.1080/15350770.2018.1404855>
- Culic, I., Radovici, A., & Vaduva, J. A. (2019). Teaching computer engineering concepts to non-technical students. *ELearning & Software for Education*, 1, 249–254.
- <https://doi.org/10.12753/2066-026X-19-034>
- de Vet, H. C. W., Mokkink, L. B., Mosmuller, D. G., & Terwee, C. B. (2017). Spearman–Brown prophecy formula and Cronbach's alpha: Different faces of reliability and opportunities for new applications. *Journal of Clinical Epidemiology*, 85, 45–49. <https://doi.org/10.1016/j.jclinepi.2017.01.013>
- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?

Computers & Education, 58(1), 240–249.

<https://doi.org/10.1016/j.compedu.2011.08.006>

Dewey, J. (1916). *Democracy and education: An introduction to the philosophy of education*. (n.p.).

Diaz, L., Kick, R. A., & Kuemmel, A. (2016). *AP Computer Science Principles Performance Task*. 720–720. <https://doi.org/10.1145/2839509.2844711>

DoED secretary's final supplemental priorities and definitions for discretionary grant programs, 83 Fed. Reg. 9096, (2018).

Doman, M., Sleight, M., & Garrison, C. (2015). Effect of GameMaker on student attitudes and perceptions of instructors. *International Journal of Modern Education and Computer Science*, 7(9), 1–13. <https://doi.org/10.5815/ijmecs.2015.09.01>

Ebel, R. L. (1967). The relation of item discrimination to test reliability. *Journal of Educational Measurement*, 4(3), 125–128.

Edmonds, W. A., & Kennedy, T. D. (2017). *An applied guide to research designs: Quantitative, qualitative, and mixed methods*. SAGE Publications, Inc.
<https://doi.org/10.4135/9781071802779>

English, M. C., & Kitsantas, A. (2013). Supporting student self-regulated learning in problem-and project-based learning. *Interdisciplinary Journal of Problem-Based Learning*, 7(2). <https://doi.org/10.7771/1541-5015.1339>

Ernst, J. V, & Clark, A. C. (2012). Fundamental computer science conceptual understandings for high school students using original computer game design. *Journal of STEM Education: Innovations & Research*, 13(5), 40–45.

Erol, O. (2020). How do students' attitudes towards programming and self-efficacy in

- programming change in the robotic programming process? *International Journal of Progressive Education*, 16(4), 13–26. <https://doi.org/10.29329/ijpe.2020.268.2>
- Erümit, A. K., Öngöz, S., & Aksoy, D. A. (2020). Designing a computer programming environment for gifted students: A case study. *Malaysian Online Journal of Educational Technology*, 8(3), 41–58. <https://doi.org/10.17220/mojet.2020.03.003>
- Exploring computer science*. (2019). <https://ed.sc.gov/instruction/career-and-technical-education/programs-and-courses/career-clusters/information-technology/fundamentals-of-computing-standards/>
- Fereday, J., & Muir-Cochrane, E. (2006). Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International Journal of Qualitative Methods*, 5(1), 80–92. <https://doi.org/10.1177/160940690600500107>
- Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley.
- Franklin, D. J., & Grossberg, S. (2017). A neural model of normal and abnormal learning and memory consolidation: Adaptively timed conditioning, hippocampus, amnesia, neurotrophins, and consciousness. *Cognitive, Affective and Behavioral Neuroscience*, 17, 24–76. <https://doi.org/10.3758/s13415-016-0463-y>
- Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences of the United States of America*, 111(23), 8410–8415. <https://doi.org/10.1073/pnas.1319030111>

- Froiland, J. M., Oros, E., Smith, L., & Hirschert, T. (2012). Intrinsic motivation to learn: The nexus between psychological health and academic success. *Contemporary School Psychology, 16*, 91–100.
- Gao, J., & Hargis, J. (2010). Promoting technology-assisted active learning in computer science education. *Journal of Effective Teaching, 10*(2), 81–93.
- Gavriel, J. (2015). Tips on inductive learning and building resilience. *Education for Primary Care, 26*(5), 332–334. <https://doi.org/10.1080/14739879.2015.1079080>
- George, D., & Mallery, P. (2002). *SPSS for Windows step by step: A simple guide and reference, 11.0 update* (4th ed.). Allyn & Bacon.
- Giannakos, M. N. (2014). Exploring students intentions to study computer science and identifying the differences among ICT and programming based courses. *Turkish Online Journal of Educational Technology, 13*(4), 36–46.
- Glickman, M. E., Rao, S. R., & Schultz, M. R. (2014). False discovery rate control is a recommended alternative to Bonferroni-type adjustments in health studies. *Journal of Clinical Epidemiology, 67*(8), 850–857.
<https://doi.org/10.1016/j.jclinepi.2014.03.012>
- Goode, J., & Margolis, J. (2011). Exploring computer science: A case study of school reform. *ACM Transactions on Computing Education, 11*(2), 1–16.
<https://doi.org/10.1145/1993069.1993076>
- Grizioti, M., & Kynigos, C. (2020). Computer-based learning, computational thinking, and constructionist approaches. In A. Tatnall (Ed.), *Encyclopedia of Education and Information Technologies*. Springer, Cham. https://doi.org/10.1007/978-3-319-60013-0_75-2

- Gurer, M. D., Cetin, I., & Top, E. (2019). Factors affecting students' attitudes toward computer programming. *Informatics in Education*, 18(2), 281–296.
<https://doi.org/10.15388/infedu.2019.13>
- Hanus, M. D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Computers & Education*, 80, 152–161.
<https://doi.org/10.1016/j.compedu.2014.08.019>
- Harasim, L. (2012). *Learning theory and online technologies*. Routledge.
- Harris, C. J., Penuel, W. R., D'Angelo, C. M., DeBarger, A. H., Gallagher, L. P., Kennedy, C. A., Cheng, B. H., & Krajcik, J. S. (2015). Impact of project-based curriculum materials on student learning in science: Results of a randomized controlled trial. *Journal of Research in Science Teaching*, 52(10), 1362–1385.
<https://doi.org/10.1002/tea.21263>
- Helle, L., Tynjälä, P., & Olkinuora, E. (2006). Project-based learning in post-secondary education - Theory, practice and rubber sling shots. *Higher Education*, 51(2), 287–314. <https://doi.org/10.1007/s10734-004-6386-5>
- Herr, K., & Anderson, G. (2005). *The action research dissertation: A guide for students and faculty*. Sage Publications, Inc.
- Hinkle, D. E., Wiersma, W., & Jurs, S. G. (1979). *Applied statistics for the behavior sciences*. Rand McNally College Publishing.
- Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review*, 16(3), 235–266.
<https://doi.org/10.1023/B:EDPR.0000034022.16470.f3>

- Hughes-Roberts, T., Brown, D., Boulton, H., Burton, A., Shopland, N., & Martinovs, D. (2020). Examining the potential impact of digital game making in curricula based teaching: Initial observations. *Computers and Education*, 158, 1–15.
<https://doi.org/10.1016/j.compedu.2020.103988>
- Hwang, G., & Wu, P. (2012). Advancements and trends in digital game-based learning research: A review of publications in selected journals from 2001 to 2010. *British Journal Of Educational Technology*, 43(1), E6–E10. <https://doi.org/10.1111/j.1467-8535.2011.01242.x>
- Javidi, G., & Sheybani, E. (2014). Teaching computer programming through game design: A game-first approach. *GSTF Journal on Computing*, 4(1), 17–22.
https://doi.org/10.5176/2251-3043_4.1.303
- Jenson, J., & Droumeva, M. (2016). Exploring media literacy and computational thinking: A Game Maker curriculum study. *Electronic Journal of E-Learning*, 14(2), 111–121.
- João, P., Nuno, D., Fábio, S. F., & Ana, P. (2019). A cross-analysis of block-based and visual programming apps with computer science student-teachers. *Education Sciences*, 9(3), 181. <https://doi.org/10.3390/educsci9030181>
- Johnson, C. (2017). Learning to program with Game Maker. *International Journal of Computer Science Education in Schools*, 1(2), 1–20.
<https://doi.org/10.21585/ijcses.v1i2.5>
- Jumaat, N. F., Tasir, Z., Abd halim, N. D., & Mohamad Ashari, Z. (2017). Project-based learning from constructivism point of view. *Advanced Science Letters*, 23(8), 7904–7906. <https://doi.org/10.1166/asl.2017.9605>

- King, B. M., Rosopa, P. J., & Minium, E. W. (2018). *Statistical reasoning in the behavioral sciences* (7th ed.). Wiley.
- Kingsley, T. L., & Grabner-Hagen, M. M. (2015). Gamification: Questing to integrate content knowledge, literacy, and 21st-century learning. *Journal of Adolescent & Adult Literacy*, 59(1), 51–61. <https://doi.org/10.1002/jaal.426>
- Kirschner, P. A., & De Bruyckere, P. (2017). The myths of the digital native and the multitasker. *Teaching and Teacher Education*, 67, 135–142. <https://doi.org/10.1016/j.tate.2017.06.001>
- Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2), 75–86. https://doi.org/10.1207/s15326985ep4102_1
- Kyewski, E., & Krämer, N. C. (2018). To gamify or not to gamify? An experimental field study of the influence of badges on motivation, activity, and performance in an online learning course. *Computers and Education*, 118(April 2017), 25–37. <https://doi.org/10.1016/j.compedu.2017.11.006>
- Kynigos, C., & Grizioti, M. (2020). Modifying games with ChoiCo: Integrated affordances and engineered bugs for computational thinking. *British Journal of Educational Technology*, 51(6), 2252–2267. <https://doi.org/10.1111/bjet.12898>
- Laakso, N. L., Korhonen, T. S., & Hakkarainen, K. P. J. (2021). Developing students' digital competences through collaborative game design. *Computers and Education*, 174(104308), 1–15. <https://doi.org/10.1016/j.compedu.2021.104308>
- Lai, C. (2018). Using inquiry-based strategies for enhancing students' STEM education

- learning. *Journal of Education in Science, Environment and Health*, 4(1), 110–117.
<https://doi.org/10.21891/jeseh.389740>
- Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: A practical primer for t-tests and ANOVAs. *Frontiers in Psychology*, 4, 1–12.
<https://doi.org/10.3389/fpsyg.2013.00863>
- Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process* (2nd ed.). Prentice Hall PTR.
- Larmer, J., Mergendoller, J., & Boss, S. (2015). *Setting the standard for project based learning: A proven approach to rigorous classroom instruction*. ASCD.
- Lewis, H. R., & Papadimitriou, C. H. (1998). *Elements of the theory of computation* (2nd ed.). Prentice-Hall.
- Loyalka, P., Liu, O. L., Li, G., Chirikov, I., Kardanova, E., Gu, L., Ling, G., Yu, N., Guo, F., Ma, L., Hu, S., Johnson, A. S., Bhuradia, A., Khanna, S., Froumin, I., Shi, J., Choudhury, P. K., Beteille, T., Marmolejo, F., & Tognatta, N. (2019). Computer science skills across China, India, Russia, and the United States. *Proceedings of the National Academy of Sciences*, 116(14), 6732–6736.
<https://doi.org/10.1073/pnas.1814646116>
- Malik, S. I., Mathew, R., Al-nuaimi, R., Al-sideiri, A., & Coldwell-Neilson, J. (2019). Learning problem solving skills: Comparison of E-learning and M-learning in an introductory programming course. *Education & Information Technologies*, 24(5), 2779–2796. <https://doi.org/10.1007/s10639-019-09896-1>
- Malone, A. H. (2019). Computer science high school graduation credit. In *State of South Carolina Department of Education*. <https://www.ed.sc.gov/newsroom/school->

district-memoranda-archive/computer-science-high-school-graduation-credit-memo/computer-science-high-school-graduation-credit-memo/

Martins, V. F., Concilio, I. A. S., & Guimarães, M. P. (2018). Problem based learning associated to the development of games for programming teaching. *Computer Applications in Engineering Education*, 26(5), 1577–1589.

<https://doi.org/10.1002/cae.21968>

McGregor, J. D., & Sykes, D. A. (2001). *A practical guide to testing object-oriented software*. Addison-Wesley.

McHugh, M. L. (2012). Interrater reliability: The kappa statistic. *Biochemia Medica*, 22(3), 276–282.

Mertler, C. A. (2019). *Action research: Improving schools and empowering educators* (6th ed.). SAGE.

Minner, D. D., Levy, A. J., & Century, J. (2010). Inquiry-based science instruction-what is it and does it matter? Results from a research synthesis years 1984 to 2002. *Journal of Research in Science Teaching*, 47(4), 474–496.

<https://doi.org/10.1002/tea.20347>

Mitchell, J. D., Amir, R., Montealegre-Gallegos, M., Mahmood, F., Shnider, M., Mashari, A., Yeh, L., Bose, R., Wong, V., Hess, P., Amador, Y., Jeganathan, J., Jones, S. B., & Matyal, R. (2018). Summative objective structured clinical examination assessment at the end of anesthesia residency for perioperative ultrasound. *Anesthesia and Analgesia*, 126(6), 2065–2068.

<https://doi.org/10.1213/ANE.0000000000002826>

Moreno, J. (2012). Digital competition game to improve programming skills. *Journal of*

Educational Technology & Society, 15(3), 288–297.

<http://search.ebscohost.com.pallas2.tcl.sc.edu/login.aspx?direct=true&db=eue&AN=79816983&site=ehost-live>

Northway, S., Dauphin, Y., Jin, B., Heinan, T., Quispe, A., Mills, B., Cai, C., Guo, P., Haynes, L., Ojeda, M., & Lubimir, A. (n.d.). *Computing: Computer programming*. Retrieved July 27, 2019, from <https://www.khanacademy.org/computing/computer-programming>

Novak, G. M. (2011). Just-in-time teaching. *New Directions for Teaching & Learning*, 2011(128), 63–73. <https://doi.org/10.1002/tl.469>

O’Grady-Jones, M. K. (2020). *Ready coder one: Action research exploring the effects of collaborative game design-based learning on gifted fourth graders’ 21st century skills and science content knowledge*. (Publication No. 27744700) [Doctoral Dissertation, University of South Carolina]. ProQuest Dissertations & Theses @ University of South Carolina.

Osadebe, P. U. (2015). Construction of valid and reliable test for assessment of students. *Journal of Education and Practice*, 6(1), 51–56.

Oyong, S. B., & Ekong, V. E. (2019). An explorative survey of formal and agile software development methods. *Global Journal of Pure and Applied Sciences*, 25, 71–79. <https://doi.org/10.4314/gjpas.v25i1.10>

Papanikolaou, K., & Boubouka, M. (2011). Promoting collaboration in a project-based E-learning context. *Journal of Research on Technology in Education*, 43(2), 135–155.

Patterson, B. F., & Ewing, M. (2013). *Validating the use of AP exam scores for college course placement*. <https://files.eric.ed.gov/fulltext/ED558108.pdf>

- Perenc, I., Jaworski, T., & Duch, P. (2019). Teaching programming using dedicated Arduino Educational Board. *Computer Applications in Engineering Education*, 27(4), 943–954. <https://doi.org/10.1002/cae.22134>
- Perneger, T. V. (1998). What’s wrong with Bonferroni adjustments. *BMJ (Clinical Research Ed.)*, 316(7139), 1236–1238. <https://doi.org/10.1136/bmj.316.7139.1236>
- Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C.-H. (2020). Relating natural language aptitude to individual differences in learning programming languages. *Scientific Reports*, 10(1), 3817–3826. <https://doi.org/10.1038/s41598-020-60661-8>
- Prince, M., & Felder, R. (2007). The many faces of inductive teaching and learning. *Journal of College Science Teaching*, 36(5), 14–20.
- Prince, M. J., & Felder, R. M. (2006). Inductive teaching and learning methods: Definitions, comparisons, and research bases. *Journal of Engineering Education*, 95(2), 123–138. <https://doi.org/10.1002/j.2168-9830.2006.tb00884.x>
- Pullan, M. (2013). Using robotics to improve retention and increase comprehension in introductory programming courses. *Journal of Educational Technology Systems*, 42(2), 141–149. <https://doi.org/10.2190/ET.42.2.f>
- Pundak, D., Herscovitz, O., & Shacham, M. (2010). Attitudes of face-to-face and E-learning instructors toward “active learning.” *European Journal of Open, Distance and E-Learning*, 2, 1–12.
- Qian, M., & Clark, K. R. (2016). Game-based learning and 21st century skills: A review of recent research. *Computers in Human Behavior*, 63, 50–58. <https://doi.org/10.1016/j.chb.2016.05.023>
- Randolph, J. J. (2008). *Multidisciplinary methods in educational technology research and*

development. JULKAISIJA. <http://justus.randolph.name/methods>

- Razali, N. M., & Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1), 21–33.
- Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., Horses, I. H. M., Basawapatna, A., Gluck, F., Grover, R., Gutierrez, K., & Repenning, N. (2015). Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education*, 15(2), 1–31.
<https://doi.org/10.1145/2700517>
- Reynolds, R. B. (2016). Relationships among tasks, collaborative inquiry processes, inquiry resolutions, and knowledge outcomes in adolescents during guided discovery-based game design in school. *Journal of Information Science*, 42(1), 35–58. <https://doi.org/10.1177/0165551515614537>
- Reynolds, R., & Caperton, I. H. (2011). Contrasts in student engagement, meaning-making, dislikes, and challenges in a discovery-based program of game design learning. *Educational Technology Research and Development*, 59, 267–289.
<https://doi.org/10.1007/s11423-011-9191-8>
- Richland, L. E., Kornell, N., & Kao, L. S. (2009). The pretesting effect: Do unsuccessful retrieval attempts enhance learning? *Journal of Experimental Psychology: Applied*, 15(3), 243–257. <https://doi.org/10.1037/a0016496>
- Robertson, J. (2013). The influence of a game-making project on male and female learners' attitudes to computing. *Computer Science Education*, 23(1), 58–83.

<https://doi.org/10.1080/08993408.2013.774155>

Rum, S. N. M., & Ismail, M. A. (2017). Metocognitive support accelerates computer assisted learning for novice programmers. *Journal of Educational Technology & Society*, 20(3), 170–181.

<http://search.ebscohost.com.pallas2.tcl.sc.edu/login.aspx?direct=true&db=eue&AN=123966665&site=ehost-live>

Saavedra, A. R., Liu, Y., Haderlein, S. K., Rapaport, A., Garland, M., Hoepfner, D., Morgan, K. L., & Hu, A. (2021). *Knowledge in action efficacy study over two years*. Center for Economic and Social Research, USC Dornsife.

[https://cesr.usc.edu/sites/default/files/Knowledge in Action Efficacy Study_18feb2021_final.pdf](https://cesr.usc.edu/sites/default/files/Knowledge%20in%20Action%20Efficacy%20Study_18feb2021_final.pdf)

Sakulvirikitkul, P., Sintanakul, K., & Srisomphan, J. (2020). The design of a learning process for promoting teamwork using project-based learning and the concept of agile software development. *International Journal of Emerging Technologies in Learning*, 15(3), 207–222. <https://doi.org/10.3991/ijet.v15i03.10480>

Saldaña, J. (2021). *The coding manual for qualitative researchers* (4th ed.). Sage.

Saldaña, J., & Omasta, M. (2017). Analyzing documents, artifacts, and visual materials.

In *Qualitative resarch: Analyzing life* (pp. 63–88). Sage Publications.

Sawilowsky, S. S. (2009). New effect size rules of thumb. *Journal of Modern Applied Statistical Methods*, 8(2), 597–599. <https://doi.org/10.22237/jmasm/1257035100>

Scholnik, M., Kol, S., & Abarbanel, J. (2006). Constructivism in theory and in practice. *English Teaching Forum*, 44(4), 12–20.

Scherer, R., Siddiq, F., & Viveros, B. S. (2019). The cognitive benefits of learning

- computer programming : A meta-analysis of transfer effects. *Journal of Educational Psychology*, 111(5), 764–792. <https://doi.org/10.1037/edu0000314>
- Shen, C. W., Wu, Y.-C. J., & Lee, T.-C. (2014). Developing a NFC-equipped smart classroom: Effects on attitudes toward computer science. *Computers in Human Behavior*, 30, 731–738. <https://doi.org/10.1016/j.chb.2013.09.002>
- Shenton, A. K. (2004). Strategies for ensuring trustworthiness in qualitative research projects. *Education for Information*, 22(2), 63–75. <https://doi.org/10.3233/EFI-2004-22201>
- Silm, G., Tiitsaar, K., Pedaste, M., Zacharia, Z. C., & Papaevripidou, M. (2017). Teachers’ readiness to use inquiry-based learning: An investigation of teachers’ sense of efficacy and attitudes toward inquiry-based learning. *Science Education International*, 28(4), 315–325.
- Simonson, M. (1979). Attitude measurement : Why and how. *Educational Technology*, 19(9), 34–38.
- Sommerville, I. (2001). *Software engineering* (6th ed.). Addison-Wesley.
- South Carolina Department of Education. (2017). *South Carolina computer science and digital literacy standards*.
- South Carolina Department of Education. (2019a). *District headcount by gender, ethnicity and pupils in poverty 2019-20*. <https://ed.sc.gov/data/other/student-counts/active-student-headcounts/2019-20-active-student-head-counts/45-day-district-headcount-by-gender-ethnicity-and-pupils-in-poverty-2019-20/>
- South Carolina Department of Education. (2019b). *SC school report card: Greenwood school district 50 / 2018-2019*.

<https://www.screportcards.com/overview/academics/academic-achievement/?q=eT0yMDE5JnQ9RCZzaWQ9MjQ1MDAwMA>

South Carolina Department of Education. (2019c). *School headcount by grade 2019-20*.

<https://ed.sc.gov/data/other/student-counts/active-student-headcounts/2019-20-active-student-head-counts/45-day-school-headcount-by-grade-2019-20/>

Southern Regional Education Board. (2016). *Bridging the computer science education gap: Five actions states can take*. https://www.sreb.org/sites/main/files/file-attachments/cs_commission_2016_0.pdf

Spencer, D. (Ed.). (1994). *Webster's new world dictionary of computer terms* (5th ed.). Macmillan.

State of South Carolina Department of Education. (2018). *SC school report card:*

Greenwood school district 50 | 2017-2018.

<https://www.screportcards.com/overview/print/?q=eT0yMDE4JnQ9RCZzaWQ9MjQ1MDAwMA>

Stoffová, V. (2019). Educational computer games in programming teaching and learning.

ELearning & Software for Education, 1, 39–45. <https://doi.org/10.12753/2066-026X-19-004>

Swacha, J., Skrzyszewski, A., & Sysło, W. A. (2010). Computer game design classes:

The students' and professionals' perspectives. *Informatics in Education, 9*(2), 249–260.

Sweller, J., Kirschner, P. A., & Clark, R. E. (2007). Why minimally guided teaching

techniques do not work: A reply to commentaries. *Educational Psychologist, 42*(2), 115–121. <https://doi.org/10.1080/00461520701263426>

- Taber, K. S. (2018). The use of Cronbach's alpha when developing and reporting research instruments in science education. *Research in Science Education*, 48, 1273–1296. <https://doi.org/10.1007/s11165-016-9602-2>
- Theodoraki, A., & Xinogalos, S. (2014). Studying students' attitudes on using examples of game source code for learning programming. *Informatics in Education*, 13(2), 265–277.
- Thomas, M. K., Ge, X., & Greene, B. A. (2011). Fostering 21st century skill development by engaging students in authentic game design projects in a high school computer programming class. *Journal of Educational Computing Research*, 44(4), 391–408. <https://doi.org/10.2190/EC.44.4.b>
- Tian, J. (2005). *Software quality engineering: Testing, quality assurance, and quantifiable improvement*. John Wiley & Sons, Inc.
- Topalli, D., & Cagiltay, E. N. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers & Education*, 120, 64–74. <https://doi.org/10.1016/j.compedu.2018.01.011>
- Tracy, S. J. (2020). *Qualitative research methods: Collecting evidence, crafting analysis, communicating impact* (2nd ed.). Wiley-Blackwell.
- Tsai, M., Wang, C., & Hsu, P. (2019). Developing the computer programming self-efficacy scale for computer literacy education. *Journal of Educational Computing Research*, 56(8), 1345–1360. <https://doi.org/10.1177/0735633117746747>
- Tucker, A., Deek, F., Jones, J., Mccowan, D., Stephenson, C., & Verno, A. (2003). *A model curriculum for K-12 computer science: Final report of the ACM K-12 task force curriculum committee* (No. 104043). ACM.

- Tyng, C. M., Amin, H. U., Saad, M. N. M., & Malik, A. S. (2017). The influences of emotion on learning and memory. *Frontiers in Psychology*, 8(1454), 1–22.
<https://doi.org/10.3389/fpsyg.2017.01454>
- U.S. Bureau of Labor Statistics. (2022, September 8). *Employment projections*.
<https://www.bls.gov/emp/tables.htm>
- Végh, L., & Stoffová, V. (2019). Learning object-oriented programming by creating games. *ELearning & Software for Education*, 1, 20–29.
<https://doi.org/10.12753/2066-026X-19-002>
- Veletsianos, G., Beth, B., Lin, C., & Russell, G. (2016). Design principles for “thriving in our digital world”: A high school computer science course. *Journal of Educational Computing Research*, 54(4), 443–461. <https://doi.org/10.1177/0735633115625247>
- Wilkerson-Jerde, M. H. (2014). Construction, categorization, and consensus: Student generated computational artifacts as a context for disciplinary reflection. *Educational Technology Research and Development*, 62(1), 99–121.
<https://doi.org/10.1007/s11423-013-9327-0>
- Willis, J., & Edwards, C. L. (Eds.). (2014). *Action research: Models, methods, and examples*. Information Age Publishing, Inc.
- Winarno, S., Muthu, K. S., & Ling, L. S. (2018). Direct problem-based learning (DPBL): A framework for integrating direct instruction and problem-based learning approach. *International Education Studies*, 11(1), 119–126.
<https://doi.org/10.5539/ies.v11n1p119>
- Wu, B., & Wang, A. I. (2012). A guideline for game development-based learning: A literature review. *International Journal of Computer Games Technology*, 2012, 1–

20. <https://doi.org/10.1155/2012/103710>

YoYo Games. (2021). *GameMaker studio 2*. <https://www.yoyogames.com/gamemaker>

Zendler, A., & Klaudt, D. (2012). Central computer science concepts to research-based teacher training in computer science: An experimental study. *Journal of Educational Computing Research*, 46(2), 153–172. <https://doi.org/10.2190/EC.46.2.c>

Zeni, J. (1998). A guide to ethical issues and action research. *Educational Action Research*, 6(1), 9–19. <https://doi.org/10.1080/09650799800200053>

Zhu, M. (2020). Effective pedagogical strategies for STEM education from instructors' perspective: OER for educators. *Open Praxis*, 12(2), 257–270. <https://doi.org/10.5944/openpraxis.12.2.1074>

APPENDIX A: CHECKPOINTS

Project	Module	Time	Description	Date
Pinball	1-5			
Ball Bouncer	6	2	Lesson 21-23: wall blocks, ball, goal, ball-goal collision, ball-walls collision	9/14
		1	Lesson 24: playing pieces (diamond, octagon, hour_glass, peak, paddle) added to bin area	9/16
	7	2	Lesson 25a: playing piece drop - drags, legal drop, dropped piece does not drag, reappears in bin	9/21
		1	Lesson 25b: ball collision with playing piece - increase score, bounce, destroy diamond	10/11
		1	Lesson 27: ball drops out of cannon; alignment & direction	10/12
		1	Lesson 28a: goal top, win/loss end screen	10/18
		1	Lesson 28b: max quantity playing pieces	10/19
		1	Lesson 29a: power up	10/20
		1	Lesson 29b: rotating paddle, no playing piece drag after ball drop	10/21
Matching	8	1	Lesson 31: play and quit buttons work	10/26
		1	Lesson 32: card sprite with 7 frames; card turns over on left released	10/28
		2	Lesson 33: game timer; 12 cards dealt dynamically	11/2
	9	2	Lesson 34: remove match; end screen on 6 matches	11/5
		1	Lesson 35: adjust timer on win (increase or stop); delay before end screen	11/7
31/Scat	10	1	Lesson 39: sprite for each suit	11/11
		2	Lesson 40-42: deck shuffle in console output	11/15
		2 1	Lesson 43 (function lesson): deal card to player card	11/22
		1	Lesson 44: All cards in room	11/23
	11	4	Lesson 45: draw deck; discard; draw discard; knock	12/1
		3	Lesson 46: computer turn; arrow points to current turn	12/6

		3	Lesson 47: end hand; remove tokens; start new hand	12/13
Sky Falling CS	12	1	Lesson 49: start screen to cut scene	2/25

APPENDIX B: PARTICIPANT INFORMATION

Table B.1 Participant Demographics

Pseudonym	Age	Sex	Race
Christal	14	Female	White
Krystina	13	Female	Black or African American
Bree ^a	14	Female	White
Maris	14	Female	White
Jerrod	14	Male	White
Denzel	14	Male	Asian
Shelly	13	Female	White
Abegail ^a	14	Female	White
Mary Jo	14	Female	Black or African American
Damion	13	Male	Black or African American & White
Dakota	14	Male	White
Qianna ^a	14	Female	Black or African American
Jonie ^a	13	Female	White
Lucius	14	Male	Black or African American
Indie	14	Female	White
Milford	13	Male	White
Hailey	14	Female	Black or African American
Marlena ^a	14	Female	American Indian or Alaska Native
Aleesha ^a	14	Female	White
Julia ^a	14	Female	White
Monster Fan ^a	14	Male	White
Shaylyn	14	Female	American Indian or Alaska Native
Annabelle ^a	14	Female	White

Pseudonym	Age	Sex	Race
Teddie ^a	13	Male	White
Pibb ^a	14	Male	White
Sanford	14	Male	Black or African American
Oakley ^a	14	Male	White
Reuben	14	Male	White

Note. $N = 28$. Age as of 4/29/22.

^a Completed the interview.

Table B.2 Participant Academic History

Pseudonym	Gifted and Talented	2021 SC READY Performance Level	
		ELA	Math
Christal	No	Exceeds Expectations	Approaches Expectations
Krystina	Yes	Exceeds Expectations	Exceeds Expectations
Bree ^a	Yes	Exceeds Expectations	Exceeds Expectations
Maris	No	Exceeds Expectations	Meets Expectations
Jerrold	Yes	Exceeds Expectations	Exceeds Expectations
Denzel	No	Exceeds Expectations	Exceeds Expectations
Shelly	Yes	Exceeds Expectations	Exceeds Expectations
Abegail ^a	No	Meets Expectations	Meets Expectations
Mary Jo	Yes	Exceeds Expectations	Meets Expectations
Damion	Yes	Exceeds Expectations	Exceeds Expectations
Dakota	Yes	Approaches Expectations	Approaches Expectations
Qianna ^a	Yes	Exceeds Expectations	Meets Expectations
Jonie ^a	Yes	Meets Expectations	Approaches Expectations
Lucius	Yes	Exceeds Expectations	Exceeds Expectations
Indie	Yes	Exceeds Expectations	Meets Expectations
Milford	Yes	Exceeds Expectations	Exceeds Expectations
Hailey	Yes	Meets Expectations	Exceeds Expectations
Marlena ^a	Yes	Meets Expectations	Exceeds Expectations

Pseudonym	Gifted and Talented	2021 SC READY Performance Level	
		ELA	Math
Aleesha ^a	No	Exceeds Expectations	Approaches Expectations
Julia ^a	Yes	Exceeds Expectations	Exceeds Expectations
Monster Fan ^a	Yes	Exceeds Expectations	Exceeds Expectations
Shaylyn	Yes	Exceeds Expectations	Exceeds Expectations
Annabelle ^a	No	Meets Expectations	Exceeds Expectations
Teddie ^a	No	Exceeds Expectations	Meets Expectations
Pibb ^a	No	Exceeds Expectations	Exceeds Expectations
Sanford	Yes	Meets Expectations	Exceeds Expectations
Oakley ^a	Yes	Exceeds Expectations	Exceeds Expectations
Reuben	Yes	Exceeds Expectations	Exceeds Expectations

Note. $N = 28$.

^a Completed the interview.

APPENDIX C: GAME DEVELOPMENT PROJECT DIRECTIONS

Project Directions

Project directions adapted from Zulama (Carnegie Learning, 2021).

You and a partner will be designing and coding an original game. Here are the parameters for your game project.

1) You can pick any game theme you think can be turned into an engaging and fun game. However, the theme must be within guidelines set by your teacher. Discuss your game theme with your teacher before moving forward with it. You might want to talk to a science or social studies teacher about your project and build a game for another class. Perhaps you feel strongly about a social issue and would like to build a serious game and pose solutions to such global problems as world hunger. Maybe you'd like to address a civic issue such as the election process and voting rights.

Whatever you decide after brainstorming and deciding on a game theme, keep your game simple. World hunger is a huge topic with a lot of problems to solve. Though it may seem like a great idea for a game, perhaps it is too much for your first game. Think simple. Focus on a game idea that poses a problem that is interesting and solvable.

2) You are required to demonstrate specific game design principles and coding skills in this game. You will find a complete list of these in the upcoming directions. They include:

- Minimum resource requirements
- Specific requirements for the rooms (for example, main game play level)
- Coding requirements
- Events requirements

The emphasis of your game design should be on the game mechanics and program logic. See the project rubric for how your game will be scored.

3) You will need to have at least two (try for five) people playtest your game. Start thinking about who that group will be and when that will happen.

4) Your first step is to write the Game Design Document, or GDD. Read on to learn more about this important document.

The Importance of Game Design Documents

A game design documents, also called a GDD, provides focus for your game's design. Game design documents used in the games industry are very detailed and include not only the game's design but also analyze the marketability of the proposed game . It is very involved and very lengthy.

The competition in the games industry is strong . It takes many months, and sometimes years, for a game to move from idea to market . Game design companies invest in games that they believe will earn a healthy profit . They also must decide if an initial loss on a new game will eventually turn the corner and become quite profitable . This makes the game design document a critical starting point for individuals wishing to pitch their idea .

A professional game design document includes:

- Theme
- Setting
- Genre
- Core game mechanics and platform
- Monetization model (how the game is sold and can make a profit)
- Project scope (team roles, time frame, cost to produce)
- Elevator Pitch (60 seconds)
- Project description
- What sets the project apart from other similar games
- Game's story
- Gameplay

- Assets needed (characters, sound, animation)
- Pseudocode and animation
- Schedule

Can you estimate the number of pages that make up a professional game design document? Yes, a lot. The document is highly detailed and can run beyond 50 pages. It is a big part of the initial time investment made by game designers into an idea they believe will be the makings of a very popular and profitable game.

Your Game Design Document

Your game design document will be much shorter! However, it needs to define the type of game you plan to make and how that will happen. Read through the Game Design Document Rubric to see what's expected. Be sure to include the following elements in your game design document:

Vision Statement—This is your elevator pitch. It should be a short statement that captures interest and should be no longer than 60 seconds when read aloud.

Target Audience—Define your audience. Age level? Gender? Beginning players? Experienced player? Who will play your game?

Platform—You are designing a computer game. Explain that here.

Genre—Identify the type of game you are designing. Is it a race game, a puzzle game, an adventure or role playing game? Explain that here.

Gameplay—Define the goals, game mechanics, the game's components, and user experience, including levels, here.

Game art—Describe what is planned for art. What will the style be? Is there a time period? Is there fantasy art or will the art be more realistic. Define that here.

Detailed User Interface—Include mock ups of the main game levels that show the objects that will be included on each level along with text to explain the object's mechanics. This is the section that is often referred to as a one-page design.

Story—Write a short summary of the game's story. Identify characters. Make this short. Just like your vision statement, this explanation should quickly capture interest.

Share your plans for your GDD with your teacher. Be sure you understand your teacher's requirements for the GDD.

Game Design Document Pitfalls

While your game design document is essential to your project, it also is dynamic. One of the pitfalls of game design documents is to think of them as absolute. They are not. Game design documents are edited and changed as the game's development progresses. Remember to keep track of new versions by renaming them. So, your first version is v1.0. Your second version could be numbered v1.1, and so on. Your GDD is simply an editable document that should serve as a guide. It is not a plan etched in granite, so feel comfortable with making changes to it.

Another pitfall of game design documents is to think it is perfectly fine if a GDD is not regularly updated. Be careful not to fall into this trap. Trying to list all past design changes to the game will be very difficult if days pass between the change and logging the change in your GDD. Keep your game design document current at all times. Visit it often!

You may think that it is okay to shortcut what you write in your GDD because it is more important to start designing and coding than to start writing a document. This is a third pitfall. Your GDD is central to the success of your game project. Give it the time it needs so that your design process will be smooth and planned.

Keep in mind the concept of Fail Forward. Your GDD should recognize that some of what you initially plan simply will not work. Every failure is a stepping stone to success and essential to the process of learning from doing. Your GDD should reflect the many iterations you make to your game. Be proud of them. They show you care about creating a really good player experience.

Find a group of friends and begin brainstorming ideas for your game. Remember that brainstorming means tossing every idea on the table (or against the wall!), using a post a note for each idea. Number them so when the group goes back to them they are easy to locate. "Let's revisit idea number 3."

While you do not have to work with the game idea the group settles on as the most viable idea, you should seriously consider what they have to say.

Work with an Idea

Do you have your game idea? Have you cleared it with your teacher?

If so, then it is time to write your game design document and upload it in the **Game Design Document** assignment for your teacher to read.

Then read the next set of directions, where you will spend some time thinking about the player experience.

Is It Fun?

Ask people why they play games and you will often hear that playing games is "fun." How is fun defined ? Think about a game you enjoy—what makes *it* fun?

Games as an Imaginary Experience

Playing games is a highly personal experience. It's also highly imaginary. The game is the vehicle for the imaginary experience. The player controls the pace and, through choices made during game play, how the game evolves. It is the player who crafts the experience, but it is the designer who provides the tools for the experience through the game's mechanics, art, and story.

While some may enjoy games that continually challenge and require strategy and goal setting, others may only want to play games they are sure to always win. Still others may like to throw their experience to chance. Take for example, individuals who play Solitaire over and over again until they win a game. Players know that chance directs the game, but use of strategy in when and where to move cards balances the game somewhat and helps to tip a win in favor of the player.

In some games the player becomes his or her own storyteller. The player experience is defined by the player's choice of characters and whether the story is scripted or if the game's play is altered by decisions the player makes.

How will you use story in your game's design ?

Mastering Levels

Consider what happens during your game play when you have mastered a level. How does it affect your player experience ? Do you lose interest in the game ? Do you play the game again just because you know you can win?

Consider these questions as you design your game:

- How long should it take the player to win a game?

- How much risk do you anticipate the player will be willing to take before either frustration or boredom results and the game is placed to the side?
- What is a good balance of risk and reward?

Adding levels to your game is an effective way to present both challenge and balance to your game. The beginning levels work for players with limited skills and provide the opportunity for players to gain new skills. This builds game play interest and confidence. More advanced players quickly work through the beginning levels, validate their skills, and then move to more challenging game play.

When designing balance and levels, consider both the skills and the interests of the player. Let's see what this means in games that may be familiar to you.

Take time to analyze how balance is used in games you play. Then look at games that are new to you. What role does balance have in the games?

Game Design Principles

You are required to demonstrate specific game design principles and coding skills in this game.

The following are the minimum resource requirements:

- 4 backgrounds
- 4 rooms
- 10 sprites, at least one of these must include multiple images
- 10 objects
- 1 font
- 1 sound (extra credit)
- 1 user-defined function

The rooms must meet the following requirements:

- Start Screen included

- Instructions or Cut scene—Although not required, consider using a timeline to progress through the game introduction. When players leave this room, they should be familiar with how to play the game and how the player controls game moves (key presses and/or mouse clicks.)
- Main game play level—Depending on the game genre and the length of this level, your game may include additional rooms and levels.
- End Screen which includes feedback on whether the player won or lost. Each room must include a background.
- Navigation to advance from the start screen, replay, and quit the game must be based on mouse clicks.

Coding Elements

Keep the following coding requirements in mind as you brainstorm ideas for your game:

Your game must have:

- A scoring system displayed on the main game level
- At least 2 collision events
- A random function so that the game has replayability
- At least 1 object type which is created dynamically when the game is running
- At least 1 object that moves
- At least 2 Boolean variables
- Clear comments with your code

The following events are also required at some point in the game:

- Create
- Alarm
- Draw
- Step

The following coding concepts must be used at some point in your game:

- User-defined variables
- Conditional statements
- Arrays (extra credit)
- Loops

Keep in Mind

The emphasis of your game design should be on the game mechanics and program logic. Resist the temptation to devote too much time to perfecting the game art. Plan that as one of your iterations. It is important to have a playable prototype quickly so that you can see if your design is working and if the project scope is manageable. To help with game art, use the Internet to search for copyright-free game art. The site OpenGameArt.org is a starting point.

You will need to have at least 2 (try for 5) people playtest your game. Start thinking about who that group will be and when that will happen. What type of feedback will you be hoping to receive from your playtesters ? If you said, "Lots of ideas of how to improve the game," you are exactly correct!

Develop a plan for building your game. Write your plan as steps you will take to build and code your game. What will you do first ? Next ? After that ? What is your last step?

Remember to refer to your checklist often to make sure you have included all required game design and coding elements.

Now it's time to build your game! Have fun with it.

Status Reports

Status Reports are weekly updates you give to your teacher on your progress. You are now ready to build your prototype, which means you also are ready to write your first status report.

As pair programmers you will submit a status report from both of you.

Your **status report** has three main parts. Bulleted items are added to each section.

DONE—What's been accomplished from a start date to the current date. Here is an example of what might be on a DONE list at this point. Make sure you list all that you've accomplished during the time frame. What else could be on this list ?

- Completed design document
- Uploaded design document for review
- Organized sprites
- Created five new sprites

TO DO—This is a list of what you plan to do during the next time frame (such as the next week of class). The list should be as detailed as possible. Here is an example of the start of a TO DO list.

- Build backgrounds and rooms
- Create events
- Code mouse input
- Add goal
- Add the User Interface

QUESTIONS/CHALLENGES—This part of the status report is a list of questions and challenges you anticipate addressing at some point. This list serves as the basis for conversations you may have with your classmates and teacher. Asking for feedback and sharing ideas with others is a good practice to follow, and one that indicates that you are invested in making the best game possible.

Carefully think through your questions and state them in a clear manner so others understand what you are trying to achieve in your game.

- Possible question for my teacher
- Possible question for my peers
- Questions I need to address in my game design and / or coding
- Challenges that I may encounter as I work through my To Do list

- Challenges I anticipate with the game as a whole.

Several status reports should be submitted from the time you have completed your design document to the time when you are ready to upload your prototype for review by your teacher. One status report is due after submitting your Game Design Document. One per week are due during prototyping. Check with your teacher on the deadlines for your Status Reports.

It is Ready!

Upload your game to the **Prototype** assignment in Google Classroom when you have finished building your prototype and are ready to have it reviewed by your teacher.

You Did It!

- You wrote a design document.
- You built an original game to specs provided for you.
- You completed several status reports.
- You uploaded an original and playable game for review.

That's a lot!

What comes next ? That's right. Playtesting. You are at a very important point in game development. You have tested your game over and over again as you were building it. Chances are you have played your game all the way through several times, as well. Are you ready to share it with others?

Sure!

Go for it! But, what's the best approach to playtesting your game ? How will you get the most out of playtesting?

Keep the iterative cycle in mind. A big part of your playtesting is to figure out what can be done to move your game at least one notch higher on a 1–10 scale of playability! Remember: A game is never really finished!

Now it's time to put together a plan for your playtest.

1. Write a playtest plan. Use the Playtest Document as a guide for your playtest plan.
2. Choose your playtesters.
3. Determine where and when your game will be playtested

Iterative Game Design



Take a look at the Iterative Cycle.

- What does it tell you about game design?
- Where is your game on the iterative game cycle?
- What is meant by "A game is never finished?"

How was your playtest ? Was your game received as you thought it would be ? Did players finish the game ? Did they seem engaged ? Was it too complex? Consider the balance of depth and complexity.

Now take a look at the results of your playtest

- Make a list of suggested improvements based on the comments made by your playtesters.
- Make a list of planned improvements (iterations).

- Then set up a time line to complete the iterations.

Second Playtest

A second playtest is important to ensure that the improvements you made to your game actually make it more playable and engaging. Follow these steps to set up your second playtest.

- Revise your playtest plan. What did you learn from the organization of your first playtest?
 - Is there anything you would change ? Perhaps the space used or the time set aside for the playtest ? Change up your plan as needed to ensure a valued playtest experience.
- Choose your playtesters.
 - Should you use the same playtesters or new ones ? Which will give you the best results ? This decision is up to you, but be prepared to justify your decision.

Collect the results of the playtest. Decide if your game needs further iterations or is ready to share. Take the time you need to build a game that you like to play, and well, like! While it may be difficult to be completely unbiased about your game, you know it best. If you like playing it, chances are others will, as well. However, be your worst critic. Remember, a game is never finished, so even if you are ready to share it, you still can go back and improve it as new ideas for game enhancements come to mind.

Project Rubric

All events and code should be labeled *Rubric: <description>*

I should be able to search your code for *Rubric* and locate all of the code needed for grading. You do not have to label the Game Design Document or resources/assets.

Reporting Category	Scoring Criteria	Decision Rules
Row 1 Game Design Document (0-4 points)	<ul style="list-style-type: none"> See the Game Design Document Rubric 	There are 100 points available in the Game Design Document Rubric. Divide the points earned by 25 to obtain a range from 0 to 4.
Row 3 Resource / Asset Requirements (0-2 points; 1 extra credit point)	<p>The submitted game project includes the following resources:</p> <ul style="list-style-type: none"> 4 backgrounds 4 rooms 10 sprites (one with multiple frames) 1 font 1 sound (optional) 	<p>Award 2 points if all required resources are included.</p> <p>Award 1 point if one or two required resources are missing.</p> <p>Award 1 extra point if a sound is included.</p>
Row 4 Coding Elements (0-4 points)	<p>Comments identify elements for scoring.</p> <p>Mechanics:</p> <ul style="list-style-type: none"> Score displayed on main level 2 collisions Use of random values Dynamic instance creation 1 moving instance 2 Boolean variables <p>Events:</p> <ul style="list-style-type: none"> Create Alarm Draw Step 	Award 4 points if all required coding elements are present and labeled with comments. Deduct 1 point for each missing element.
Row 2 Algorithm Implementation (0-4 points; 1 extra point)	The submitted source code includes a program code segment of a student-	Consider ONLY the section of code identified as <i>submitted algorithm</i>

	<p>developed algorithm that includes:</p> <ul style="list-style-type: none"> • user-defined variable • sequencing • selection (conditional statement) • iteration • arrays (optional) 	<p>through comments in the submitted source code.</p> <p>Award one point each for user-defined variable, sequencing, selection, and iteration. These points may be earned if the algorithm is not encapsulated in a procedure.</p> <p>If this code segment calls other student-developed procedures, the procedures called from within the identified procedure can be considered.</p> <p>The use of selection and iteration cannot be trivial and must affect the outcome of the game.</p> <p>Award an extra point for the use of an array.</p>
<p>Row 3 Procedural Abstraction (0-2 points)</p>	<p>The submitted source code includes:</p> <ul style="list-style-type: none"> • a student-developed function with at least one parameter that has an effect on the functionality of the function. • at least two calls to the function with different parameter values. 	<p>Consider <i>ONLY</i> the section of code identified as <i>submitted function definition</i> through comments in the submitted source code and calls to the submitted function.</p> <p>Award one point for the function definition.</p> <p>Award one point for at least two calls to the function with different parameter values. The parameter values chosen must demonstrate the possibility of different behavior in the selection or iteration.</p>

<p>Row 4 Managing Complexity (0-1 points)</p>	<p>The submitted source code includes a comment above the student-developed procedure that explains how the procedure manages complexity in the program.</p>	<p>Consider ONLY the section of code identified as <i>submitted procedure definition</i> through comments in the submitted source code.</p> <p>Award one point for an explanation of how the procedure manages complexity in comments above the procedure definition.</p>
---	--	---

Game Design Document Rubric

Rubric created by Zulama (Carnegie Learning, 2021).

Skills Assessed	Levels of Achievement (<i>circle the level achieved for each assessed skill</i>)			
Criteria	Unsatisfactory	Basic	Proficient	Advanced
Use of Evidence	Design Document lacks linkage between concept explained in the lesson and the design solution. 0	Design Document contains a weak link between the concept explained in the lesson and the design solution. 6	Design Document contains a satisfactory link between the concept explained in the lesson and the design solution. 8	Design Document contains a strong, well-articulated link between the concept explained in the lesson to the design solution. 10
Accuracy of Information	Design Document contains design changes that lack knowledge of design goal. 0	Design Document notes design changes that will produce limited results related to the design goal. 6	Design Document contains a plan for the construction of the pieces and parts to be changed, but the details may be vague, or it is unclear if the plan will achieve the desired results. 8	Design Document contains a detailed plan for the construction of the pieces and parts to be changed in order to meet design goal. 10
Development of a Clear Argument	Design Document lacks explanation of lesson concept and how it will be demonstrated in the final deliverable.	Design Document contains a weak explanation of both the lesson concept and how it will be demonstrated in the final deliverable.	Design Document contains some explanation of the lesson concept and how it will be demonstrated in the final deliverable, but	Design Document contains detailed explanation of the lesson concept and how it will be demonstrated in the final

			details are missing.	deliverable.
	0	6	8	10
Logical and Effective Reasoning and Attention to Writing Conventions	Design Document lacks detail. The scope and intent of the project is ambiguous. 0	Design Document misses a few major design details, leading to misinterpretation of the plan. 6	Design Document addresses major design details, but does lack explanation of every part of the plan. 8	Design Document is clear and complete, addressing all design details. 10
Design Layout	Unsatisfactory	Basic	Proficient	Advanced
Attention to Writing / Artwork Conventions	Layout lacks changes to pieces and parts called for in the Design Document; lacks connection between the written plan and the images. The images are incomplete. 0	Layout lacks many of the changes to pieces and parts called for in the Design Document; lacks one-to-one correspondence between the written plan and the images. The images are ambiguous. 6	Layout includes most changes to pieces and parts called for in the Design Document; there is a tenuous correspondence between the written plan and the images. The images are generally clear but lack detail. 8	Layout includes all changes to pieces and parts called for in the Design Document; there is a one-to-one correspondence between the written plan and the images, displayed in an easy-to-understand manner. 10

Depth of Study	Layout lacks visual demonstration of lesson concepts. 0	Layout provides vague visual demonstration of lesson concepts. 6	Layout visually demonstrates the lesson concepts, but some details are missing or the game play is ambiguous. 8	Layout visually demonstrates the lesson concepts with clear, fleshed-out detail. 10
Effective Problem-Solving Strategies	The amount of work proposed is far beyond the scope of the assignment. 0	The amount of work proposed is difficult to achieve within the assignment timeframe and the design must be radically altered. 6	The amount of work proposed is achievable only with a slight reduction in scope. Features are cut before work begins. 8	The amount of work proposed is achievable within the assignment timeframe. 10
Deliverable Project	Unsatisfactory	Basic	Proficient	Advanced
Necessary Knowledge Acquisition	Project barely runs and reveals lack of knowledge acquisition and concerted effort. 0	Project runs with many bugs and design underwent radical changes that were capricious and lacked ties to design iteration. 6	Project runs as designed with minor bugs OR Project runs without bugs but slight unsubstantiated design changes were made. 8	Project runs as designed without bugs. 10

Use of Evidence	Project lacks features that were detailed in the Design Document and demonstrated in the Design Layout. <
-----------------	--

Student point total = the total points of “Skills Assessed.”

Playtest Document

Adapted from Zulama (Carnegie Learning, 2021).

	Before: Preparation	During: Observing & Recording	After: Reflecting
Playability / Fun	What do you expect your player will enjoy about the game? (Be specific.)	On a scale of 1 to 5, with 1 being boring and 5 being fun, how would you rate how much fun the player had with your game? Explain your rating.	What iterations could you make in order to make the game more fun?
Timing	How long do you think your game takes to play?	How much time does the player spend on your game?	Will you need to adjust the length of play? If so, how will you accomplish that?
Player Reactions	Which type of reaction do you expect your player to show?	What type of reaction did the player show?	Are there any changes you could make in order to elicit more positive reactions from your player?
Body Language	What will you look for when observing body language? Describe what you anticipate observing.	Describe any body language that you see the player use.	What does this body language reveal about any iterations you need to make?
Strategic	What kinds of	List the strategies	List ideas you have

Approach	strategies do you anticipate the player will use to overcome obstacles in your game?	you observe the player using to overcome obstacles.	to improve the player's use of strategy.
Bumps in the Road	What challenges do you anticipate the player may have? (These are design aspects of the game that may need to be improved, but you want to see how the player responds before changes are made.)	Is there a moment where the player wants to quit or give up? If so, when? What comments are made that indicate frustration or boredom with the game?	What are some possible changes you could make to improve the game's playability?
Player Comments	What types of comments will be especially helpful to you as a game designer?	Write down any other comments, questions, or concerns you hear the player express.	Based on the playtest, what can you list in the two columns about your game?

APPENDIX D: CONTENT KNOWLEDGE ASSESSMENT

Multiple-Choice

1. A certain game keeps track of the maximum and minimum scores obtained so far. If num represents the most recent score obtained, which of the following algorithms correctly updates the values of the maximum and the minimum?
 - a. If num is greater than the minimum, set the minimum equal to num. Otherwise, if num is greater than the maximum, set the maximum equal to num.
 - b. If num is less than the minimum, set the minimum equal to num. Otherwise, if num is greater than the maximum, set the maximum equal to num.
 - c. If num is less than the minimum, set the minimum equal to num. Otherwise, if num is less than the maximum, set the maximum equal to num.
 - d. If num is greater than the minimum, set the minimum equal to num. Otherwise, if num is less than the maximum, set the maximum equal to num.
2. A programmer is creating an algorithm that will be used to turn on the motor to open the gate in a parking garage. The specifications for the algorithm are as follows.
 - The gate should not open when the time is outside of business hours.
 - The motor should not turn on unless the gate sensor is activated.
 - The motor should not turn on if the gate is already open.Which of the following algorithms can be used to open the gate under the appropriate conditions?
 - a. Check if the time is outside of business hours. If it is, check if the gate sensor is activated. If it is, check if the gate is closed. If it is, turn on the motor.
 - b. Check if the time is during business hours. If it is, check if the gate sensor is activated. If it is, check if the gate is open. If it is, turn on the motor.
 - c. Check if the time is during business hours. If it is, check if the gate sensor is activated. If it is not, check if the gate is open. If it is not, turn on the motor.
 - d. Check if the time is during business hours. If it is, check if the gate sensor is activated. If it is, check if the gate is open. If it is not, turn on the motor.

3. Three different numbers need to be placed in order from least to greatest. For example, if the numbers are ordered 9, 16, 4, they should be reordered as 4, 9, 16. Which of the following algorithms can be used to place any three numbers in the correct order?
- a. If the first number is greater than the last number, swap them. Then, if the first number is greater than the middle number, swap them.
 - b. If the first number is greater than the middle number, swap them. Then, if the middle number is greater than the last number, swap them.
 - c. If the first number is greater than the middle number, swap them. Then, if the middle number is greater than the last number, swap them. Then, if the first number is greater than the last number, swap them.
 - d. If the first number is greater than the middle number, swap them. Then, if the middle number is greater than the last number, swap them. Then, if the first number is greater than the middle number, swap them.
4. In a certain game, the integer variable `bonus` is assigned a value based on the value of the integer variable `score`.

-If `score` is greater than 100, `bonus` is assigned a value that is 10 times `score`.

-If `score` is between 50 and 100 inclusive, `bonus` is assigned the value of `score`.

-If `score` is less than 50, `bonus` is assigned a value of 0.

Which of the following code segments assigns `bonus` correctly for all possible integer values of `score`?

Select **two** answers.

- a.

```
if score > 100 {  
    bonus = score * 10  
} else {  
    if score >= 50 {  
        bonus = score  
    } else {  
        bonus = 0  
    }  
}
```
- b.

```
if score >= 50 {  
    if score > 100 {  
        bonus = score * 10  
    } else {  
        bonus = 0  
    }  
} else {
```



```

        bonus = 0
    }

c. if score < 50 {
    bonus = 0
} else {
    if score >= 50 {
        bonus = score
    } else {
        bonus = score * 10
    }
}

d. if score < 50 {
    bonus = 0
} else {
    if score > 100 {
        bonus = score * 10
    } else {
        bonus = score
    }
}

```

5. The cost of a customer's electricity bill is based on the number of units of electricity the customer uses.

- For the first 25 units of electricity, the cost is \$5 per unit.
- For units of electricity after the first 25, the cost is \$7 per unit.

Which of the following code segments correctly sets the value of the variable `cost` to the cost, in dollars, of using `numUnits` units of electricity?

```

a. if numUnits <= 25 {
    cost = numUnits * 5
} else {
    cost = numUnits * 7
}

b. if numUnits <= 25 {
    cost = numUnits * 5
} else {
    cost = (numUnits - 25) * 7
}

c. if numUnits <= 25 {
    cost = numUnits * 5
} else {
    cost = 25 * 5 + (numUnits - 25) * 7
}

```

```
    }  
d. if numUnits <= 25 {  
    cost = numUnits * 5  
} else {  
    cost = 25 * 7 + (numUnits - 25) * 5  
}
```

6. The ticket prices at a movie theater are given below.

Type of Ticket	Price (in dollars)
Regular	12
Child (ages 12 and below)	9
Senior (ages 60 and above)	9

Additional \$5 fee for 3-D movies

A programmer is creating an algorithm to set the value of `ticketPrice` based on the information in the table. The programmer uses the integer variable `age` for the age of the moviegoer. The Boolean variable `is3D` is `true` when the movie is 3-D and `false` otherwise.

Which of the following code segments correctly sets the value of `ticketPrice`?

- a.

```
ticketPrice = 12
if age <= 12 or age >= 60 {
    ticketPrice = 9
}
if is3D {
    ticketPrice = 17
}
```
- b.

```
ticketPrice = 12
if age <= 12 or age >= 60 {
    ticketPrice = 9
}
else {
    ticketPrice = 17
}
```
- c.

```
ticketPrice = 12
if age <= 12 or age >= 60 {
    ticketPrice = 9
}
if is3D {
    ticketPrice += 5
}
```
- d.

```
ticketPrice = 12
if age <= 12 or age >= 60 {
    ticketPrice = 9
}
else {
    ticketPrice += 5
}
```

7. A biologist wrote a program to simulate the population of a sample of bacteria. The program uses the following procedures.

Procedure Call	Explanation
<code>InitialPopulation()</code>	Returns the number of bacteria at the start of the simulation.
<code>NextPopulation(currPop)</code>	Based on the current value of <code>currPop</code> , returns the number of bacteria after one hour

Code for the simulation is shown below.

```
hours = 0
startPop = InitialPopulation()
currentPop = startPop
while hours < 24 and currentPop > 0 {
    currentPop = NextPopulation(currentPop)
    hours += 1
}
show_debug_message(currentPop - startPop)
```

Which of the following are true statements about the simulation?

- I. The simulation continues until either 24 hours pass or the population reaches 0.
 - II. The simulation displays the average change in population per hour over the course of the simulation.
 - III. The simulation displays the total population at the end of the simulation.
- a. I only
 - b. II only
 - c. III only
 - d. I and II

8. The code segment below is intended to display all multiples of 5 between the values `start` and `end`, inclusive. For example, if `start` has the value 35 and `end` has the value 50, the code segment should display the values 35, 40, 45, and 50. Assume that `start` and `end` are multiples of 5 and that `start` is less than `end`.

```
i = start
for(count = 0; count < <MISSING EXPRESSION>;
count++) {
    show_debug_message(i)
    i += 5
}
```

Which of the following could replace **<MISSING EXPRESSION>** so that the code segment works as intended?

- a. `end - start + 1`
- b. `end - start + 6`
- c. `((end - start) / 5) + 1`
- d. `5 * (end - start) + 1`

9. An algorithm is intended to display the following output:

red red blue red red blue red red blue

Which of the following code segments can be used to display the intended output?

- a.

```
for(i = 0; i < 2; i++) {  
    for(j = 0; j < 3; j++) {  
        show_debug_message("red")  
    }  
    show_debug_message("blue")  
}
```
- b.

```
for(i = 0; i < 2; i++) {  
    for(j = 0; j < 3; j++) {  
        show_debug_message("blue")  
    }  
    show_debug_message("red")  
}
```
- c.

```
for(i = 0; i < 3; i++) {  
    for(j = 0; j < 2; j++) {  
        show_debug_message("red")  
    }  
    show_debug_message("blue")  
}
```
- d.

```
for(i = 0; i < 3; i++) {  
    for(j = 0; j < 2; j++) {  
        show_debug_message("blue")  
    }  
    show_debug_message("red")  
}
```

Multiple-Choice Answer Key

1. b
2. d
3. d
4. a, d
5. c
6. c
7. a
8. c
9. c

Performance Task

- Read all task instructions before beginning.
- Width and height numbers are in pixels.
- You may use any of the images provided to you during the course. You may also create your own images. You will not be assessed on aesthetics, but your objects should contrast sufficiently with your background so that functionality can be assessed. All of the images you need can be found in the Pinball resources.
- Create a rectangular room with width 1024 and height 768. You should have exactly one room.
- Create walls to bound the edges of the rectangular room. The walls should be 32 pixels thick. Your bottom wall will function differently than your other walls.
- Create a rectangular paddle just above the bottom floor with width 160 and height 64.
- The paddle can move right and left by the player pressing the right and left arrow keys, respectively. The paddle should continue to move when the right or left arrow key is held down. The paddle should stop moving when it hits the walls.
- The game immediately begins when run. Therefore, you should not add a start screen or other functionality to start the game.
- Every two seconds, add an instance of a ball to the room with width 32 and height 32. For each ball:
 - speed is 4
 - gravity is 0.1
 - gravity direction is down

- direction is a random angle
 - y is 100
 - x is a random number between 100 and 900
- If a ball hits any wall other than the bottom, the ball should bounce.
- The ball should bounce off of the paddle.
- If a ball hits the bottom wall:
 - The ball is destroyed
 - A life is lost.
- When lives reach zero:
 - Destroy the balls
 - Stop spawning new balls
 - Display “Game Over” in the middle of the room.
 - Display a “Play Again” button that will restart the room when pressed.

Restarting the room should:

 - Set score to 0
 - Set lives to 3
 - Set paddle to starting length
- Start the game with three lives. Display the lives in the upper right corner of the room.
- Add 10 points to the score every time a ball hits the paddle. Display the score in the upper left corner of the room.
- When the score reaches 50, change the paddle width to 224.

Performance Task Rubric

Reporting Category	Scoring Criteria	Decision Rules
Row 1 Sprite Dimensions (0-2 points)	<p>The dimension of the following sprites should match the specifications:</p> <ul style="list-style-type: none"> • room • walls • balls • initial paddle • long paddle 	<p>2 points: all sprites have the appropriate dimensions.</p> <p>1 point: one sprite is significantly different.</p> <p>0 points: more than one sprite is significantly different than required.</p>
Row 2 Initial Instances Present (0-1 points)	<p>The following instances should be in the room when the game begins:</p> <ul style="list-style-type: none"> • room • walls • initial paddle 	<p>1 point: all initial instances are present when the game begins. The walls should be on the edges of the room, and the paddle should be just above the bottom wall.</p> <p>0 points: any incorrect or missing instances</p>
Row 3 Paddle Movement (0-2 points)	<p>The paddle</p> <ul style="list-style-type: none"> • should move left and right when the left and right arrow keys are pressed. • should not move left when the left wall is reached and should not move right when the right wall is reached 	<p>2 points: the paddle moves left and right when the left and right arrow keys are pressed, and the paddle will not move through the walls.</p> <p>1 point: the paddle moves in both directions but moves through a wall.</p> <p>0 points: the paddle does not move or moves in only one direction OR the paddle does not continue to move when the left or right arrow keys are held down.</p>
Row 4 Ball Spawn (0-2 points)	<p>A ball</p> <ul style="list-style-type: none"> • spawns every second • has the correct properties when it spawns 	<p>2 points: a new ball instance is added to the room every 1 to 3 seconds and has properties that satisfy the requirements.</p>

		<p>1 point: a ball does not spawn every 1 to 3 seconds or has incorrect properties</p> <p>0 points: a ball does not spawn every 1 to 3 seconds and has incorrect properties OR a ball is in the room, but no balls were created dynamically</p>
Row 5 Ball Bounce (0-2 points)	<p>The ball:</p> <ul style="list-style-type: none"> • bounces on collision with top and side walls • bounces on collision with paddle • if this is difficult to playtest, check code for correct collision handling 	<p>2 points: all bounces work properly.</p> <p>1 point: one bounce works properly.</p> <p>0 points: more than one bounce does not work properly</p>
Row 6 Lives (0-2 points)	<ul style="list-style-type: none"> • Lives are displayed in the upper right corner of the room. • The game begins with three lives. • A life is lost when the ball hits the bottom wall. 	<p>2 points: all scoring criteria are met.</p> <p>1 point: all but one criterion is met.</p> <p>0 points: lives are not displayed OR more than one criterion is not met</p>
Row 7 Score (0-2 points)	<ul style="list-style-type: none"> • The score is displayed in the upper left corner of the room. • The score increases by 10 for each ball that hits the paddle. • A score of 50 (cutoff may be as high as 100) causes the paddle to increase in width. 	<p>2 points: all scoring criteria are met.</p> <p>1 point: all but one criterion is met.</p> <p>0 points: score is not displayed OR more than one criterion is not met</p>
Row 8 Game Over (0-2 points)	<p>When lives reach zero:</p> <ul style="list-style-type: none"> • Destroy the balls • Stop spawning new balls • Display “Game Over” in the middle of the room. 	<p>2 points: all scoring criteria are met.</p> <p>1 point: “Game Over” or “Play Again” appears when lives reach zero.</p>

	<ul style="list-style-type: none"> • Display a “Play Again” button that will restart the room when pressed. Restarting the room should: <ul style="list-style-type: none"> ○ Set score to 0 ○ Set lives to 3 ○ Set paddle to starting length 	0 points: score and lives are not displayed OR more than one criterion is not met
--	---	--

APPENDIX E: SURVEY

Directions

This series of statements will be used to gauge your attitudes regarding computer science.

There are no right or wrong answers, so please answer honestly. Rate your level of agreement with the following statements using the following scale: 1 = strongly disagree, 2 = disagree, 3 = neutral, 4 = agree, 5 = strongly agree.

These survey statements and aspects were developed by Shen et al. (2014).

Aspect (a): Self-concept in Computer Science

1. Computer science is fun
2. I feel at ease with computer science, and I understand concepts easily
3. Computer science is one of my best subjects
4. The feeling that I have toward computer science is positive
5. Computer science is a topic, which I enjoy studying

Aspect (b): Learning Computer Science at School

6. We learn interesting things in computer science lessons
7. I look forward to my computer science lessons
8. Computer science lessons are exciting
9. I would like to do more computer science at school
10. I like computer science better than most other subjects at school

Aspect (c): Learning Computer Science Outside of School

11. I would like to join a computer science club

- 12. I like watching computer science programs on TV
- 13. I like to visit computer science museums
- 14. I would like to do more computer science activities outside school
- 15. I like reading computer science magazines and books
- 16. It is exciting to learn about new things happening in computer science

Aspect (d): Future Participation in Computer Science

- 17. I would like to study more computer science in the future
- 18. I would like to have a job working with computer science
- 19. I would like to become a computer science teacher
- 20. I would like to become a computer scientist
- 21. Computer science knowledge is necessary for my future career

Aspect (e): Importance of Computer Science

- 22. Computer science and technology is important for society
- 23. Computer science and technology makes our lives easier and more comfortable
- 24. The benefits of computer science are greater than the harmful effects
- 25. Computer science and technology are helping the poor
- 26. There are many exciting things happening in computer science and technology

APPENDIX F: INTERVIEW PROTOCOL

Date/Time:

Interviewee:

Informed Consent

Good morning. I would like to focus on your responses without being distracted by note-taking. You and your parent or guardian have already signed a consent form, but I want to review a few important points. I would like to use a tool to record and transcribe our conversation. Only I will have access to the recording, and your identity will be kept confidential. Your participation is voluntary, and you may choose to stop at any time. Your participation will not affect your grade in the course. This interview is scheduled for 30 minutes. I may need to interrupt you occasionally or move to new questions to stay within the time restriction. Thank you for participating.

Introduction

You have been selected to speak with me because you offer a valuable perspective on the game development project that you just completed. My research focuses on improving curriculum and instruction for students learning to program. Please be completely honest and do not worry about hurting my feelings or anticipating what I want to hear. Are you ready to begin?

Questions

1. Can you describe what you learned in this unit? Please include what you think you were expected to learn and what you actually learned. Did the assessment provide an accurate measure of what you know for each skill?
2. Describe how effective the game development project has been in helping you learn in our course.
3. How did the game development project help you learn to analyze and develop algorithms? Can you give me an example?
4. Describe any programming or game development skills that improved during the project.
5. Can you recall any instances when you enjoyed developing your game?
6. Describe how you generally feel when you come to class. How does that compare with your other courses?
7. Describe how your interest in computer science has changed outside of school.
8. Tell me about any plans you have to study or work with computer science in the future.
9. What is the most beneficial effect of computer science and technology on society? Why?
10. What is the most harmful effect of computer science and technology on society? Why?
11. In what ways do your attitudes toward computer science affect your performance in the course?

12. Would you please describe any attitudes or feelings that may have affected your ability to learn in the computer science course?

13. Describe your reactions to errors and setbacks in the game you developed.

Include how you felt during the troubleshooting process.

Conclusion

That concludes the questions that I have prepared. Is there anything that I should know but failed to ask? Thank you again for your time. When I report my results, I plan to use fake names for the participants. Should I make one up, or is there a name that you want me to use for you?

APPENDIX G: IRB APPROVAL



OFFICE OF RESEARCH COMPLIANCE

INSTITUTIONAL REVIEW BOARD FOR HUMAN RESEARCH DECLARATION of NOT RESEARCH

Theodore Jenks
Wardlaw College
820 Main Street
Columbia, SC 29208

Re: **Pro00118427**

Dear Theodore Jenks:

This is to certify that research study entitled ***THE EFFECTS OF PROJECT-BASED GAME DEVELOPMENT ON STUDENT LEARNING AND ATTITUDES: ACTION RESEARCH IN AN 8TH GRADE INTRODUCTORY COMPUTER SCIENCE COURSE*** was reviewed on **1/21/2022** by the Office of Research Compliance, which is an administrative office that supports the University of South Carolina Institutional Review Board (USC IRB). The Office of Research Compliance, on behalf of the Institutional Review Board, has determined that the referenced research study is not subject to the Protection of Human Subject Regulations in accordance with the Code of Federal Regulations 45 CFR 46 et. seq.

No further oversight by the USC IRB is required. However, the investigator should inform the Office of Research Compliance prior to making any substantive changes in the research methods, as this may alter the status of the project and require another review.

If you have questions, contact Lisa M. Johnson at lisaj@mailbox.sc.edu or (803) 777-6670.

Sincerely,

A handwritten signature in blue ink, appearing to read 'Lisa M. Johnson', with a stylized, flowing script.

Lisa M. Johnson
ORC Assistant Director and IRB Manager

APPENDIX H: DISTRICT STUDY APPROVAL

Theodore Jenks

Tue, Sep 7, 2021 at 12:09 PM

To: "[name redacted]"

Cc: [name redacted], [name redacted]

Hi [name redacted],

I will be conducting the data collection phase for my doctoral program in the spring. USC requires me to obtain approval from my building supervisor (principal: [name redacted] at [redacted]) and district. Whom should I contact for approval at the district?

Very respectfully,

--

Theodore Jenks
Technology Department

Theodore Jenks [email redacted]

Wed, Sep 8, 2021 at 12:21 PM

To: [name redacted], [name redacted], [name redacted]

Request for study approval - please let me know if you need more detail.

The purpose of this action research will be to implement a digital game development curriculum and describe its effects on the learning outcomes and attitudes of eighth-grade students in a required computer science course at [redacted].

Very respectfully,

Theodore Jenks
Technology Department

[name redacted] Thu, Sep 9, 2021 at 9:19 AM To: "Jenks, Theodore"

You have been approved for your doctoral study.

Thanks

[name redacted]--

[name redacted]

Assistant Superintendent for Instruction

APPENDIX I: ORIGINAL AND CURRENT PSEUDONYMS

Table H.1 Original and Current Pseudonyms

Original	Current
Achlys	Christal
Ananke	Krystina
Anuke	Bree
Aphrodite	Maris
Apollo	Jerrold
Ares	Denzel
Athena	Shelly
Bastet	Abegail
Demeter	Mary Jo
Dionysus	Damion
Hades	Dakota
Hathor	Qianna
Hedetet	Jonie
Hephaestus	Lucius
Hera	Indie
Hermes	Milford
Hestia	Hailey
Isis	Marlena
Mafdet	Aleesha
Menhit	Julia
Monster Fan	Monster Fan
Nemesis	Shaylyn
Nepit	Annabelle
Nu	Teddie

Original	Current
Pibb	Pibb
Poseidon	Sanford
Ra	Oakley
Zeus	Reuben

ProQuest Number: 29398345

INFORMATION TO ALL USERS

The quality and completeness of this reproduction is dependent on the quality and completeness of the copy made available to ProQuest.



Distributed by ProQuest LLC (2023).

Copyright of the Dissertation is held by the Author unless otherwise noted.

This work may be used in accordance with the terms of the Creative Commons license or other rights statement, as indicated in the copyright statement or in the metadata associated with this work. Unless otherwise specified in the copyright statement or the metadata, all rights are reserved by the copyright holder.

This work is protected against unauthorized copying under Title 17,
United States Code and other applicable copyright laws.

Microform Edition where available © ProQuest LLC. No reproduction or digitization of the Microform Edition is authorized without permission of ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346 USA