

Learning to Code: Effects of Programming Modality
in a Game-based Learning Environment

Nirmaliz Colón-Acosta

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2019

© 2019

Nirmaliz Colón-Acosta

All rights reserved

ABSTRACT

Learning to Code: Effects of Programming Modality in a Game-based Learning Environment

Nirmaliz Colón-Acosta

As new introductory block-based coding applications for young students to learn basic computer science concepts, such as, loops and conditionals, continue to increase in popularity, it is necessary to consider the best method of teaching students these skills. Many of these products continue to exhibit programmatic misconceptions of these concepts and many students struggle with how to apply what they learn to a text-based format due to the difficulties with learning the syntactic structure not present in block-based programming languages. If the goal of teaching young students how to program is meant to develop a set of skills they may apply when learning more complex programming languages, then discerning how they are introduced to those practices is imperative. However, few studies have examined how the specific modality in which students are taught to program effects how they learn and what skills they develop. More specifically, research has yet to effectively investigate modality in the context of an educational coding game where the modality feature is controlled, and content is consistent throughout game-play. This is mainly due to the lack of available games with this feature designed into the application.

This dissertation explores whether programming modality effects how well students can learn and transfer computer science concepts and practices from an educational programming game. I proposed that by being guided from a blocks-based to text-based programming language would instill a deeper understanding of basic computer science concepts and would support learning and improve transfer and performance on new challenging tasks.

Two experimental studies facilitated game-play sessions on the developed application for this project. The first study was a 2x2 between subjects design comparing educational module (game versus basic) and programming modality (guided versus free choice). The findings from Study 1 informed the final version design for the module used in the second study where only the game module was used in order to focus the comparison between programming modality. Findings showed that students who coded using the game module performed better on a learning test. Study 2 results showed that students who are transitioned from blocks-based to text-based programming language learn basic computer science concepts with greater success than those with the free choice modality.

A comparative study was conducted using quantitative data from learning measures and qualitative video data from the interviews during the challenge task of the second study. This study examined how students at the extreme levels of performance utilized the toggle switch feature during game-play and how the absence of the feature impacted how they completed the challenge task. This analysis showed two different methods of toggle switch usage being implemented by a high and low performing student. The high performing student utilized the resources more often during the challenge tasks in lieu of leveraging the toggle switch and were still able to submit high level code. Results suggest that a free choice student who uses the feature as a tool to check their prewritten code rather than as a short cut for piecing code together as blocks and submitting the text upon the final attempt. This practice leads to a shallower understanding of the basic concepts and make it extremely difficult to expand and apply that knowledge to a more difficult task.

This dissertation includes five chapters: an introduction and theoretical framework, a game design framework and implementation description, two experimental investigations, and a

quantitative and qualitative comparative analysis. Chapter one provides the conceptual and theoretical framework for the two experimental investigations. Chapter two describes the theory and design structure for the game developed for this dissertation work. Chapter three and four will discuss the effects of programming modality on learning outcomes. Specifically, chapter 3 focuses on implications of programming modality when determining how to implement changes for the design of the game for Study 2. Chapter five discusses a comparative analysis that investigated differing work flow patterns within the free choice condition between high and low performing students. Results from these three chapters illustrate the importance of examining this component of the computer science education process in supplemental games for middle and high school students. Additionally, this work contributes in furthering the investigation of these educational games and discusses implications for design of similar applications.

Table of Contents

<u>LIST OF TABLES</u>	v
<u>LIST OF FIGURES</u>	vi
<u>ACKNOWLEDGEMENTS</u>	viii
<u>CHAPTER 1: INTRODUCTION AND THEORETICAL FRAMEWORK</u>	1
THEORETICAL FRAMEWORK	4
GAMES FOR LEARNING	4
INSTRUCTIONISM IN CONSTRUCTIVIST GAMES FOR PROGRAMMING EDUCATION.....	6
PROGRAMMING MODALITY	7
COLLABORATIVE-BASED LEARNING: PAIRED PROGRAMMING	10
COMPUTATIONAL THINKING	12
<u>CHAPTER 2: THE PEDAGOGICAL DESIGN OF MICROCITY ACADEMY</u>	15
GAME DESCRIPTION	15
STUDY 1 VERSION DESCRIPTIONS	16
GAME VERSION.....	16
BASIC VERSION.....	17
PEDAGOGICAL FRAMEWORK AND DESIGN	17
SPECIFIC IMPLEMENTATION OF PEDAGOGICAL FRAMEWORK	21
THE AVATAR AND PROBLEM CONTEXT: NAVIGATING THE GRID WITH “BOT”	21
INTEGRATING SCAFFOLDS	22

KEY FEATURES IN MCA	23
POTENTIAL IMPLICATIONS FOR PROGRAMMING MODALITY	25
CHAPTER 3: STUDY 1 DESIGN.....	30
RESEARCH QUESTIONS	30
METHODS	31
PARTICIPANTS	31
CONDITIONS	31
GAME VERSION.....	32
BASIC VERSION.....	32
MODALITY FEATURE	34
INCLUDED MODULE FEATURES	35
MEASURES	36
QUESTIONNAIRE. STUDENTS WERE GIVEN A MODIFIED VERSION OF A PRIOR EXPERIENCE SURVEY	36
PROCEDURE.....	37
RESULTS	38
EXPLORATORY ANALYSIS.....	42
STUDY 1 DISCUSSION.....	43
LIMITATIONS	43
CONCLUSIONS AND IMPLICATIONS	44
CHAPTER 4: STUDY 2 DESIGN.....	47
OVERVIEW	48
RESEARCH QUESTIONS	48
STUDY DESIGN	51

PARTICIPANTS	51
GAME MODULE VERSION 2	52
STUDY ACTIVITIES AND TIMELINE	55
STUDY 2 MEASURES	57
STUDY 2 RESULTS.....	59
POSTTEST PERFORMANCE OUTCOMES	60
GAME DATA LOG AND SURVEY OUTCOMES	62
STUDY 2 DISCUSSION.....	64
OVERALL DISCUSSION	66
LIMITATIONS	66
CONCLUSIONS AND IMPLICATIONS	67
<u>CHAPTER 5: FREE CHOICE CONDITION INVESTIGATION</u>	<u>70</u>
PROCEDURE AND MEASURES.....	70
PARTICIPANTS	71
GENERAL ANALYSIS.....	71
POSTTEST PERFORMANCE OUTCOMES	72
GAME DATA LOG OUTCOMES	73
EXPLORATORY ANALYSIS.....	76
COMPUTATIONAL THINKING BEHAVIORS INDEX	76
INTERVIEW: CHALLENGE TASKS	77
CHALLENGE TASKS	78
CT BEHAVIORS, OUTCOMES AND GAME BEHAVIORS	85
DISCUSSION.....	86
LIMITATIONS	87

CONCLUSION.....	88
CHAPTER 6: CONCLUSION	92
REFERENCES	96
APPENDIX A: STUDY 1 PRE-TEST	108
APPENDIX B: STUDY 1 POST-TEST	114
APPENDIX C: STUDY 1 ONLINE PRIOR KNOWLEDGE SURVEY.....	120
APPENDIX D: COMPUTATIONAL THINKING BEHAVIOR INDEX	125
APPENDIX E. COMPUTATIONAL THINKING BEHAVIOR INDEX RUBRIC	126
APPENDIX F. SUMMARY OF CT FACETS AND DEFINITIONS (<i>SHUTE, SUN, & ASBELL-CLARKE, 2017</i>)	127
APPENDIX G: STUDY 2 ATTITUDINAL SURVEY	128
APPENDIX H: STUDY 2 PRE/POST-TEST (ENGLISH).....	131
APPENDIX I: STUDY 2 RUBRIC EXAMPLE RESPONSE	141
APPENDIX J: STUDY 2 TRANSFER TASKS	142
APPENDIX K: STUDY 2 INTERVIEW PROTOCOL – CHALLENGE PROBLEMS	145
APPENDIX L: ANNOTATED SCREENSHOTS OF GAME VERSION 1 AND 2.....	146

LIST OF TABLES

<i>Table 1.</i> Description of Game Design Principle Components	19
<i>Table 2.</i> Terms and Constructs of MCA	27
<i>Table 3.</i> Marginal Means of Coding Knowledge Difference Scores by Condition	39
<i>Table 4.</i> ICC Between Pairs on Outcome Measures	59
<i>Table 5.</i> Means (SD) of Posttest Performance Outcomes by Condition	62
<i>Table 6.</i> Means (SD) of Survey Outcomes by Condition.	64
<i>Table 7.</i> Scores on Outcome Measures for Comparative Analysis	71
<i>Table 8.</i> Means (SD) of Posttest Outcomes within Free Choice Condition	73
<i>Table 9.</i> Kappa Values for Computational Thinking Behavior Index on Challenge Tasks	77
<i>Table 10.</i> Performance Condition, Computational Thinking Behaviors and Outcome Measures ...	90
<i>Table 11.</i> Performance Condition, Computational Thinking Behaviors and Game Behaviors	91

LIST OF FIGURES

<i>Figure 1.</i> Example of Tynker Free Choice from Blocks to Swift Python and Block/Text hybrid format in Swift Playground.....	9
<i>Figure 2.</i> Example of Chunk and Progression of students	19
<i>Figure 3.</i> Adapter GBL Design Process for MCA	26
<i>Figure 4.</i> Layout of MCA v2.0 Block-Console	28
<i>Figure 5.</i> Layout of MCA v2.0 with Text-Console	29
<i>Figure 6.</i> Game Version Example	33
<i>Figure 7.</i> Basic Version Example.....	34
<i>Figure 8.</i> Block Index Examples in Both Language Formats	35
<i>Figure 9.</i> Example of a Prompted Hint in Version 1	36
<i>Figure 10.</i> Study 1 Design by Day	38
<i>Figure 11.</i> Average Difference Scores by Module Condition	40
<i>Figure 12.</i> Average Difference Scores by Modality Condition.....	41
<i>Figure 13.</i> Frequency of Student Final Submissions using Complex Code	43
<i>Figure 14.</i> Instructional Agent “Prof. Neo” and Robot “bot” Avatar	52
<i>Figure 15.</i> Example of Prompt for Process Worksheet in Version 2	53
<i>Figure 16.</i> Block Index Usage Example.....	53
<i>Figure 17.</i> Example of Built-in Glossary	54
<i>Figure 18.</i> Example of Prompted Hint in Version 2.....	44
<i>Figure 19.</i> Study 2 Design Overview	56
<i>Figure 20.</i> Marginal Means of Game Behaviors by Condition	63
<i>Figure 21.</i> Marginal Means of Free Choice Switches per Level (Difficult Levels).....	74

Figure 22. Counts of Free Choice Switches per Level (Difficult Levels75

Figure 23. Challenge Tasks78

Figure 24. Steps of Low Performing Free Choice Student Challenge 1 Task Completion.....79

Figure 25. Glossary Selection of High Performing Student Challenge 1 Task.....80

Figure 26. Steps of High Performing Free Choice Student Challenge 1 Task Completion.....80

Figure 27. Low Performer First and Last Attempt for Challenge 281

Figure 28. Steps of High Performing Free Choice Student Challenge 2 Task Completion.....83

ACKNOWLEDGEMENTS

The task of acknowledging all the people who contributed to the completion of this dissertation would become a chapter unto itself, but here goes my attempt at including everyone.

I would like to thank my family, my parents and brothers, for the constant love and encouragement. Thank you to my aunt, Nildabel, for helping me find the site for my second study and doing the work of five research assistants during data collection, gracias tia mia Thank you to my cousin, Adrian, for helping me translate the game and all the materials. Gracias a Papa Yayo y Buelita por apoyarme en este proceso y por todo el amor, bendiciones. Huge thanks to my friends at TC, including Laura Malkiewich, Jenna Marks, Aakash Kumar, Amy Hachigan, and Lauren Young for all the guidance and support. To my awesome ladies, including Kendel Sorrell, Elyse Warren, Kayla Gallagher, Brooke Hadden and Allison Summers, thank you for all the many long chats and last-minute NYC visits. You girls kept me going and laughing. Thank you to my friend and mentor, Rachel Flynn, who first told me to pursue a doctorate degree. I hope I have made you proud, I definitely would not be here without you and your unwavering support. Your faith in me knows no bounds and I appreciate it more than I can say.

I have to thank John Black, my advisor, for constantly encouraging me to pursue my own work. He has been there for every twist and turn of the last three years. Dr. Black, thank you for the encouragement, your unwavering confidence in me and every ounce of your time. Thank you to Bryan Keller for your patience and positive chats throughout the tail end of this project. Your constant reassurance is appreciated. Thank you to Nathan Holbert, your class four years ago led to my passion for game design and your course was the catalyst for the game itself. Thank you for fueling those first initial versions and getting me excited about taking on creating my own game for this dissertation.

A special thank you and appreciation to everyone who generously donated to help supply the equipment. Huge thank you to Gabriel Colon-Acosta, Kendel Sorrell, Brooke Hadden, Luda Janda, Laney Varnadoe, Ivette Gronauer, Lorraine Lazerus, WJ Lazerus, Rich and Susan Duran, Jeff Entratter, Steve and Donna Plain, Felice Entratter, Alex, Entratter, Samuel Pietrzak, Jonathon Garcia. Karen Alba, Leslie Aaholm, Carol Bembry, Lila Nelson, Mary Gish, Susan Goldman, Rocio Santos-Carrillo, Andrea Lopez, and Grisel Polanco.

Special thanks to Adam Weiss for letting me recruit at Camp Ramapo for Children as my first site. Huge thanks to Noel and Jessica Delgado for allowing me stay in your home while I completed that first study. Thank you to the students, teachers and directors of the middle and high school sites in Salinas, Puerto Rico for allowing me to use your schools for my second study site.

Lastly, there are not enough ways I can say thank you to Daniel Plain, without whom absolutely none of this would be possible. Thank you for everything you have done for me and the magnitude in which you have contributed to this project, from all the late-night study sessions to taking on the development of the game and actively working on its' design with me. Thank you for this true labor of love, sweet man.

CHAPTER 1: INTRODUCTION AND THEORETICAL FRAMEWORK

Computer science (CS), the current buzzword in the technology and education sphere, has rapidly become less associated with genius hackers that can code and replaced with the notion that *anyone* can learn to code. More specifically, children can learn to code! With the need for graduates with computational skills to fill employment demand projected to heights well over 5 million by the year 2026 there has been a major push in education to incorporate CS into K-12 curriculum (U.S. Bureau of Labor Statistics, 2015). President Obama's 2016 initiative, Computer Science for All, demonstrates how CS has been integrated into education in recent years (Smith, 2016). This initiative seeks to expose students of all ages and backgrounds to CS and, as a result, has led to the development of multiple CS integrated learning standards including, the K-12 Computer Science Framework that is being widely adopted across the country and the Digital Readiness K-8 Computer Science Standards most recently approved by the Tennessee Department of Education (K-12 CS Framework, 2016; Digital Readiness K-8 CS Standards, 2018). Both are meant to serve as a guide for educators as they are setting goals and expectations when designing a CS integrated STEM curriculum for their students with the focus on developing computational thinking skills along with a basic understanding of CS.

Along with the increased interest in CS, there has also been a tremendous emphasis on how to foster computational thinking (CT) skills in students when teaching foundational CS concepts. CT skills, often referred to as "21st century skills", assist in efficiently solving problems in an effective manner (Shute, Sun & Asbell-Clarke, 2017). Several CT skills include; decomposition, algorithmic thinking, debugging, iteration, and pattern recognition. Although these skills are applicable across STEM or even non-STEM domains, CT is typically associated with CS, particularly using learning how to code as the task in which these skills can be fostered and

practiced. Now we are seeing an influx of learning tools, educational technologies and web or mobile applications made available to schools, educators, parents, and students with a primary focus on the introduction to CS. These tools have taken shape as educational applications target younger children by using visual block-based programming environments to motivate and engage them while they are learning CS content (Weintrop, Hansen, Harlow & Franklin, 2018). With educational initiatives like Computer Science for All and Hour of Code, block-based environments have shaped the way young children are learning and conceptualizing CS. Rather than dealing with the difficulty of learning the syntactical nuisances of programs such as JavaScript or Python as an introduction, children are presented with a programming-primitive-puzzle-piece representation for how to use and structure blocks into sequences that work best (Weintrop, Hansen, Harlow & Franklin, 2018; Bau, Gray, Kelleher, Sheldon & Turbak, 2017; Maloney, Resnick, Rusk, Silverman & Eastmond, 2010; Hansen, Iveland, Carlin, Harlow & Franklin, 2016).

Many of the available developmentally-appropriate applications for children, such as Scratch or Tynker, are teaching basic coding skills to children by introducing content through gamified educational applications that serve as supplemental learning experiences. These supplemental educational games expose children to CS concepts through an interactive and engaging problem-based framework. This is not surprising with the rise in popularity and accessibility of digital games and the average school-aged student spending approximately three hours each day split between playing video games on a console, mobile device, or spending time on the computer (0-8 Common Sense Media, 2017; Mladenovic, Boljat & Zanko, 2018). In an effort to engage students with new content in a digital application, numerous games for learning attempt to create a context for students to experience content interactively with the support of a game-like elements (Mladenovic, Boljat & Zanko, 2018).

Previous research examining games for the purposes of learning across contexts, informal and formal, report that game-play increases student motivation, affords the student an interactive experience with new content and lends to creating an enjoyable and engaging learning experience (Kafai, 2006; Vogel et al., 2006; Wouters, van Nimwegen, van Oostendorp & van der Spek, 2013). The combination of game-play and programming modality utilized in these applications offer the opportunity to ask fundamental questions involving the efficacy of these supplemental learning products. For example, what role does programming modality play in the transfer of learning from block-based to text-based programming languages? What effect do gamified supplemental learning applications have on a child's knowledge and conceptual understanding? Finally, do students perceptions of their ability to *learn* new coding concepts and skills differ dependent on the programming modality of the application?

To address these questions around the utility of programming modality in learning to code and how the application may contribute to that effect, I have included several specific areas of study pertaining to this line of investigation. Particularly discussing the current research and present theories regarding programming modality and the implications for how digital educational tools introduce new coding content to novice programmers. This is of the utmost interest as it is the ultimate goal when providing these educational experiences for children, to create opportunities of coding exposure and knowledge with the intention of preparing them to apply that knowledge when attempting to learn more complex concepts. I will then speak to the use of combining instructionist design and constructivist learning theory as the basis for many educational tools and a guiding framework for the procedure of this research. Instructionism focuses on applying educational practices that are aligned to with direct-instruction and generally lack interactivity (Jonassen,1991). Constructivism suggests that learning should be and interactive

process of exchanging ideas and fostering new understanding through those exchanges. Applicable in and out of the game, constructivism lends to supporting a method of practice which creates a collaborative and interactive space outside for learning and building meaning around new concepts and practices. The implementation of an instructionist design and constructionist learning theory framework, supported by the use an educational game, consists of incorporating explorative and collaborative practices with instructional content.

To do this between students, paired programming procedures will be included as part of the game-play experience. Both games and collaborative learning offer affordances of active feedback and repeated failure reflective of the iterative processes inherent to programming in a real-world context. This context gives way to fostering computational thinking skills as students are actively engaging in iterative problem-solving, debugging other student's code and learning to recognize patterns to increase efficacy of code. Tying these components of study together, game-based learning, and paired programming, provides the learning environment and conditions for examining the role of programming modality and the impact on student's basic understanding of computer science concepts, as well as, their perceptions regarding self-perceived competency for future learning of more complex CS concepts.

Theoretical Framework

Games for Learning

Research on the use of games for education has yielded an array of positive outcomes, such as, increasing student motivation (Papastergiou, 2009; Ertmer & Ottenbreit-Leftwich, 2013; Nadolny, Alaswad, Culber & Wang, 2017) and self-efficacy (Ritterfeld & Weber, 2005), providing timely feedback (Puzziferro & Shelton, 2008), facilitating a space for collaborative problem-based learning (Kiili, 2007) and encouraging the players to actively communicate and exchange ideas as

they actively construct a conceptualization of new knowledge from their experience (Linderoth, 2012). Games have been studied as an alternative tool for supplementing classroom-based learning and instruction (Ross & Morrison, 1989; Ross, Morrison & Lowther, 2010; Papastergiou, 2009). Originally seen as a tool for “assisted instruction”, games are being fitted to classroom learning differently. Rather than in place of a teacher they are being integrated as a supplement to traditional instruction, to make learning skills and concepts more interactive, engaging and improve a teacher’s effectiveness in the classroom (Chambers, Cheung, Gifford, Madden & Slavin, 2006; Slavin, 2009).

Ross and Lowther (2009) posit that digital supplemental tools serve a purpose for supporting learning in the classroom and at home. For instance, supplemental educational tools afford students the opportunity to practice core concepts and skills on their own and allowing the teacher time to give individual tutoring to low performing students. Additionally, these tools can be used to provide instruction for low performing students and offer engaging enrichment tasks for students who have done well with the material in class and have time to work with the content at a deeper level before continuing on to the next lesson. Finally, supplemental instruction can be offered to students outside the classroom setting who may not have direct access to teachers, during an after-school program or at home learning on their own.

Recently, the widespread saturation of games in the lives of children and adolescents has created an interest in further investigating methods for how to integrate formal learning practices into a gaming structure (Alaswad & Nadolny, 2015). Simply put, game-based learning includes the use of problem scenarios that are placed in the context of play (Tsai & Fan, 2013). Therefore, in developing a digital space where a student can be provided with new instructional content, the resources to explore the concepts further on their own and the ability to practice those skills will

ultimately guide them in building a conceptual understanding of basic coding constructs, how they interact with each other and informing their mental model of that knowledge.

Instructionism in Constructivist Games for Programming Education

With the emphasis placed on CS integration in K-12 education and teaching basic programming skills, novice-friendly block-based programming curriculum is being implemented in schools, as well as, made readily available in the home through game-based learning applications accessible on multiple devices. There is an array of block-based programming platforms derived from programs like Scratch (MIT, 2003) and text-based programming instructional tools, such as, Code.org and CodeHs, that are being specifically designed for the classroom. There is a divide in how these tools were developed, some being structured from an instructionist perspective and others from of a constructivist approach to how to teach introductory coding skills.

The theoretical framework for the pedagogical design of the game used in current study game is grounded in a combination of instructionist design and constructivist theory of learning. Instructionism is highlighted as a pedagogical approach to the design of educational technologies by teachers who are designing instructional educational tools to embed in games. Instructionism refers to the application of educational practices by teachers that are skill-based, typically non-interactive and generally prescribed based on the goals on of the instructor (Jonassen,1991). However, in blending constructivist practices in which the implementation of instructional design is rooted in student-centered approaches to produce interactive experiences and engaging exchanges of information, students have an opportunity to establish a deeper understanding of the content with which to expand their mental models and apply that knowledge in future contexts (Johnson, 2005; Honebein, 1996; Brandt, 1997).

Constructivism, as mentioned before, posits that learning is an interactive and responsive process. Particularly social-constructivism, where students are able to actively construct knowledge through an engaging interaction. A student is not only exposed to new content, but also exchanging ideas and their own understanding of the content and how to apply that understanding in novel ways with another person, that interaction is driving the exchange of information and acts as the agent of development for that students' mental model (Werhane, Hartman, Moberg, Englehart & Pritchard, 2009). Regardless, a constructivist learning environment can benefit from the structure provided by instructionism despite their extreme differences in approaches to learning. The combination of direct instruction and self-guided learning are combined several educational coding applications, however, there is not a true constructivist level of freedom in many cases when it comes to commercially available tools.

Applications that host entire curricula for novice programmers are more course-like in nature and remove the element of “game-play”. Others are strictly self-guided with very few resources to support a true novice to the computer science world. It is ideal to examine how to create an application in which students are provided with 1) instructional supports; 2) loose game-play; 3) open-ended tasks and 4) resources, as these types of games are becoming more mainstream.

Programming Modality

The continued adoption of new programming environments with different programming modalities requires that there be a better understanding of the effects of these approaches to instructing and learning CS concepts. Examples of block-based programming environments include Scratch, Alice, Blockly, Tynker, LightBot, and many more (Yaroslavski, 2014). However, these platforms do not enable the player to directly work with traditional programming languages,

such as, Java, Python, Javascript, etc. (Bau, 2015). Past research suggests that students who are introduced to block-based programming first and then moved to text-based programming report lower self-efficacy and feel overwhelmed with the syntax structure of the text programming language (Powers, Ecott, & Hirshfield, 2007). In contrast, Lewis (2010) found that students just learning to code who were introduced to text-based programming first reported higher self-efficacy in their coding abilities and program writing skills.

Recently, the effectiveness of one type of programming modality over the other has been a question of interest, however, that focus has not centered on how we can address the transitional gap between understanding basic CS concepts in a visual programming environment and successfully applying that knowledge when learning in a traditional programming environment. While previous work has found that learning to program in a blocks-based language can lead to transfer in other “real” programming languages, the concept of programming modality is understudied in both game-based and traditional learning platforms (Armoni, Meerbaum-Salant & Ben-Ari, 2015). Modality has primarily been examined in instances specific to the use of blocks or the use of text. There have been studies with hybrid conditions, block-based to text-based, however, there has not been research that examines the pedagogical approach of guiding this transition within the same digital learning environment due to the lack of applications available that would allow a researcher that level of control across content and in-app features.

Applications, such as Tynker, have incorporated a “toggle” feature where students may see their code formatted as blocks or as text format (see Figure 1). This feature is not guided, it is at the discretion of the student if he/she would want to see their code in a more “authentic” structure (Weintrop & Wilensky, 2015). Swift Playground is another example of an application with the

goal of teaching basic coding skills, this platform includes text-based programming using block-shaped text to place the code in the console (see Figure 1).

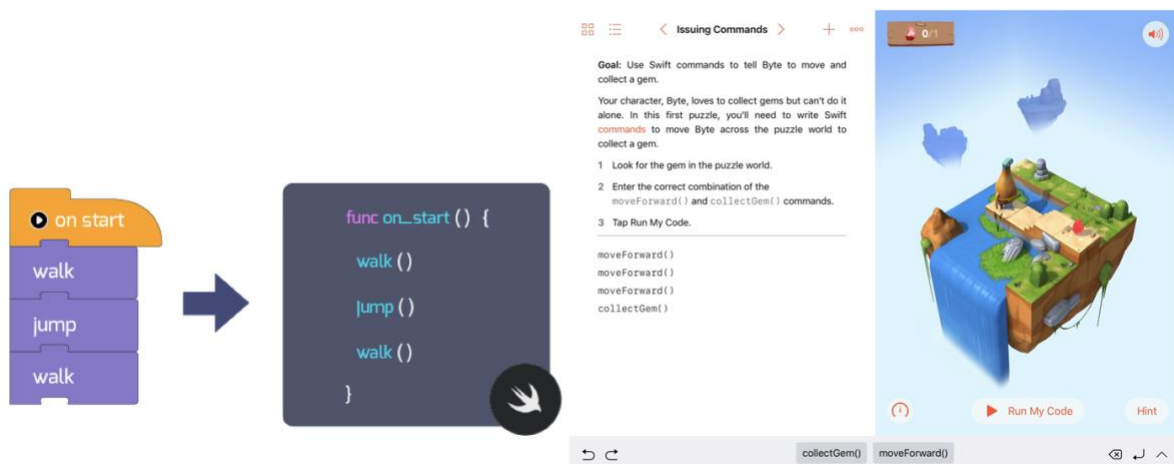


Figure 1. Example of Tynker Toggle from Blocks to Swift Python and Block/Text hybrid format in Swift Playground.

Despite all these new introductory coding applications and resources available, students continue to exhibit programmatic misconceptions of basic CS concepts, such as, loops and conditionals, and struggle with how to apply what they learn to a text-based format due to the difficulties with learning the syntactic structure not present in block-based programming languages (Grover, Basu, 2017; Mladenovic, Boljat & Zanko, 2018; Pila, Aladé, Sheehan, Lauricella & Wartella, 2019). Additionally, there are differences in conceptual understanding of program comprehension from one modality to the other. Students who learn introductory skills in block-based programming perform better on comprehension assessment items than those who learning with text-based language (Weintrop & Wilensky, 2017). Much of this may be related to whether the resources available are providing the best instructional supports or the learning context in which the resources are being applied in, informal or formal learning contexts (Mladenovic, Boljat & Zanko, 2018; Kafai & Burke, 2015). However, this outcome could also be indicative of the fact

that block-based learning simply requires less cognitive effort from the student as they are using this modality to manipulate representational command units rather than constructing those programs one piece at a time.

Therefore, several studies would suggest that the initial learning focus of digital programming environments should be on language semantics rather than on developing an understanding of syntax (Mladenovic, Boljat & Zanko, 2018). Once a concept has been introduced and practiced a student may be exposed to the text-based formatting of that same concept in a different task, allowing for the opportunity to practice skills consistently across both coding modalities.

Furthermore, developing a game wherein students are learning from a combination of instructional scaffolding and a level of autonomous discovery to navigate the transition between programming modality may support the conceptual comprehension regardless of modality (Grover et al., 2015, Dalbey & Linn, 1985; Linn & Dalbey, 1985; Winslow, 1996). Additionally, providing external instructional supports for student learning may contribute to how students conceptualize the experience of learning to code using an educational tool, such as, collaborative learning in a project-based design to facilitate the exchange of ideas and construction of novel solutions (Wouters, van Nimwegen, van Oostendorp & van der Spek, 2013).

Collaborative-based Learning: Paired Programming

Collaborative-based learning relies on activities or projects that focus on maximizing collaboration among students to enhance the activities by which they learn and the outcomes of that learning (Slavin, 1980). A collaborative game-based learning method of instruction allows for students to engage, interact and practice both basic and difficult concepts through the natural

progression of a game, this can be facilitated by implementing paired programming during game-play for the best learning outcomes (Chi and Wyle, 2014).

Paired programming is a practice defined as two programmers working together to complete a programming task, one as the “driver” who is coding the solution and one as the “navigator” who is providing the direction for how the code should be constructed (Hahn, Mentz, & Meyer, 2009; Williams & Upchurch, 2001). Previous research has found several advantages to the practice of paired programming include; fewer errors in code, better performing program, improved programming efficacy, minimized errors, decreased stress levels, and an excellent method for teaching (Tomayko, 2002; VandeGrift, 2004; Sung, Ahn & Black, 2017). A possible explanation for these positive outcomes may be related to the collaborative aspect of paired programming, wherein pairs are communicating and exchanging information, checking each other’s work and problem solving together to reach a common goal.

The notion of programming together is not an unusual one, in fact, paired programming has become a common practice in the professional field. Collaborating on one coding task between two programmers simulates the main tenets of social constructivism, where the focus lies between the social and individual student-centered processes involved during the co-construction of knowledge (Palinscar, 1998; Vygotsky et al., 1978). Students learn from modeling and practicing what they see while modifying as they develop their own understanding of new material (Palinscar, 1998; Chan et al., 1997). Paired programming facilitates both tenants, the protocol is inherently designed to support one student performing the task and the other to observe and influence those actions. This allows both participants to have the opportunity to perform and receive feedback from their partner as well as the game, additionally, the design of the practice is such that one participant cannot solve a given task without the input of their partner. Thereby requiring

participants to explain their reasoning, explain how to construct the program, test their solution, and receive the feedback and take the necessary next steps together.

The frustration or strain of working through challenging new content can be mitigated when feel the assistance of a partner in completing those difficult tasks through the exchanging on feedback and general support. Despite the activity being collaborative, performance measures, such as learning outcomes and reported self-efficacy and self-perceived competence, are assessed individually. This will assist in answering questions regarding student perception differences between programming modality conditions, and alternatively, within their own paired programming experience. Bandura (1986) defines self-efficacy as a person's judgement regarding their own capabilities to execute a task required to achieve a specific goal. A student's self-efficacy influences the amount of effort put forth in difficult of challenging tasks, motivation to persist on those tasks when presented with frequent failure and overall performance outcomes (Bandura, 1986). Student's make judgements about their own abilities based on their observations of their peers, social influences and support, and what they believe they have already achieved. It would be interesting to determine whether completing the task with a partner ultimately contributed to growth in their self-efficacy, leading to more engaged participation and high performance on the learning measures. These perceptions may, in turn, increase their self-perceive competence when asked how they feel about their ability to learn more complex coding concepts and skills in a separate instance and new material.

Computational Thinking

When learning to program students are introduced to writing code, developing algorithmic features that structure data with the expected outcome of an efficiently running program (Kafai & Burke, 2013). Students learn about the various components that make up a complete framework

of a program, such as, the syntax, conditional statements, variables and loops. Understanding these components and allowing students the opportunity to put them into practice can improve their development of computational thinking skills (Kafai & Burke, 2013).

Computational thinking (CT) has recently been identified as the conceptual foundation required to solve problems effectively and efficiently with solutions that are reusable in different contexts. CT includes: 1) decomposition; 2) abstraction; 3) algorithmic thinking; 4) debugging; 5) iteration (Wing, 2006; Shute, Sun, & Asbell-Clarke, 2017). These core concepts have been found to apply to all science, technology, engineering, and mathematics (STEM) disciplines (Henderson, Cortina, Hazzan & Wing, 2007). The task of measuring CT is still a highly debated in theory and methodology. Several assessment tools have been developed and utilized, however, they are typically aligned with the understanding on computer science concepts rather than assessed as practices (Grover et al., 2015; Korkmaz, Cakir & Ozden, 2017; Roman-Gonzalez, Perez-Gonzalez & Jimenez-Fernandez, 2016; Bers et al, 2014; Atmatzidou & Demetriadis, 2016). The same assessments are used to assess coding knowledge and I argue that examining CT relies on whether students are actively engaging and employing these problem-solving skills when constructing their solution to a given task. In a review of the CT field of research, Shute et al. (2017) presented a framework of CT facets to consider as core competencies (see Appendix E).

In distinguishing CT skills from computer science, the review reports that while CT originates from computer science, the skills are not exclusively the same as programming even though being able to program is a positive consequence of having the ability to think computationally (Shute et al., 2017; Ioannidou et al., 2011; Israel et al., 2015). However, regardless of this close relationship to programming, research examining best practices when developing CT skills in the context of programming proposed that instruction of CT should include abstracted

representation of programming languages (Lu & Fletcher, 2009). Therefore, an index based on Shute et al.'s (2017) CT framework will be developed to align with observable patterns of coding behaviors (see Appendix D). Using the collected log data from student game-play to code for the types of exhibited CT behaviors and differences in frequency by condition of modality will elucidate the question of whether there is an observable difference between programming modalities and which supports the development of CT skills more effectively. The following chapter discusses how this framework was integrated into the design of the educational game used in this work.

CHAPTER 2: THE PEDAGOGICAL DESIGN OF MICROCITY ACADEMY

Game Description

Microcity Academy (MCA) is a web-based application developed for this research meant to serve as a supplemental learning tool for teaching children basic coding skills and practices in the context of a game-based environment. The game consists of 20 coding tasks varying in difficulty and programming modality dependent on the condition randomly assigned to the student login.

Similar to Tynker, Lightbot, Scratch Jr., Blockly, or Human Resource Machine MCA's application focuses on exposing novice programmers to computer science in an interactive visual coding environment in addition to offering a guided experience when transitioning students from programming in a block-based to a text-based language. The strict guidance has since been removed from the final version of the game and students are provided with more resources to guide their own understanding of the content without the direct instruction of the instructional agent or restrictive design of the tasks themselves.

MCA is presented as a programming school where the students have been enrolled to learn how to program to become a *Microcity Programmer* and that learning these skills is important to graduating to that assigned job. Students in the game are provided a pedagogical agent who is there to serve as their instructor for new concepts and give them prompts for understanding the environment when initially navigating the console and the features of the application. The overall goal of the game is to write code that will instruct a bot to move from point A to point B at the end of the path they are shown. The application's curriculum was created to target novice programming students in middle and high school to control for the level of difficult when structuring the progression of content. Concepts incorporated into the 20 tasks vary from basic sequencing, tasks

with *for loops* and *while loops*, simple conditional statements (if/then), more complex conditional statements (if/then/else) and introducing the practices of nesting code and debugging another programmer's code.

Study 1 Version Descriptions

Study 1 used a 2x2 design to examine differences in learning performance across modality, students were randomly assigned within each module. Student were assigned to either the traditional module (basic) or a game-based module (game), and one of two programming modality conditions; free choice or guided. The description of each module version designed specifically for this study are below.

Game Version

The game version of the module is developed around the narrative that the participant is a student at Microcity Academy, a school that teaches recruits how to program so that they can take up computing jobs in Microcity (see Figure 6). For each new level the student's goal was to navigate through a grid and collect the microchips needed to boost their resource library. To do this, students are presented with tasks that are scored based on three components; 1) effectiveness of code submitted (does it solve the problem presented in the task?), 2) efficiency of code submitted (is this the best solution for the task?) and 3) the number of attempts it took for the student to solve the task. Effectiveness is measured in two ways, a) whether the program gets the bot to the exit point of the grid and b) were the microchips collected prior to completing the task? Efficiency included a count of the number of moves the bot executed, an average of total moves per task was taken from prior user data to calculate a threshold of "efficiency" for each level.

Basic Version

The basic version of the module is developed without a narrative of any kind, the user interface is stripped down and plain and no pedagogical agent even though they receive the same instructional information as students in the game version (see Figure 7). For each new task the student's goal was to navigate through a grid and collect the dots presented in similar variation to the game version, however, the student did not receive points for either of these tasks. However, all the that same information is collected in the data log.

Pedagogical Framework and Design

The theoretical framework for the pedagogical design of this game is grounded in a combination of instructionist and constructivist theories of learning. Instructionism is highlighted as a pedagogical approach to the design of educational technologies by teachers who are designing instructional educational tools to embed in games. Instructionism refers to the application of educational practices by teachers that are skill-based, typically non-interactive and generally prescribed (Jonassen,1991). However, in blending constructivist practices in which, the implementation of instructional design is rooted in student-centered approaches to produce interactive experiences and engaging exchanges of information, students have an opportunity to establish a deeper understanding of the content with which to expand their mental models and apply that knowledge in future contexts (Johnson, 2005; Honebein, 1996; Brandt, 1997). As students are actively constructing and developing their own mental representations and interpretations of the material. Those mental representations serve to link their knowledge with prior understanding and look to apply the information in different contexts.

The game was designed to address the question of whether programming modality has an effect on students learning of CS concepts. Additionally, the tasks were structured to assess how

they apply those concepts in new situations across different contexts. The game was organized to present constructs and instructions from a skills-based perspective, but with a constructivist environment to practice those skills and manipulate those constructs to develop an understanding of their utility from seeing them in practice.

In referencing previous research studies on best practices when teaching basic introductory programming, it is suggested that this methodology, situated in this combined instructionist design and the constructivist theory of learning, of organizing constructs and practices in as “chunks” of related information is ideal (Robins et al., 2003; Rogalski & Samurcay, 1990). However, students are also being transitioned from block-based to text-based programming and there is not a large scope of relevant research indicating the best strategy for that transition. This is only challenging with the students who are assigned to the guided modality, whereas, students who are assigned to the free choice modality, have the freedom to switch from blocks to text-based programming syntax at their discretion. To structure this shift for the students being automatically transitioned between block-based and text-based programming, the switch will occur at different times within each conceptual chunk in order to allow for students to practice in both modalities across all coding constructs (see Figure 2 for example).

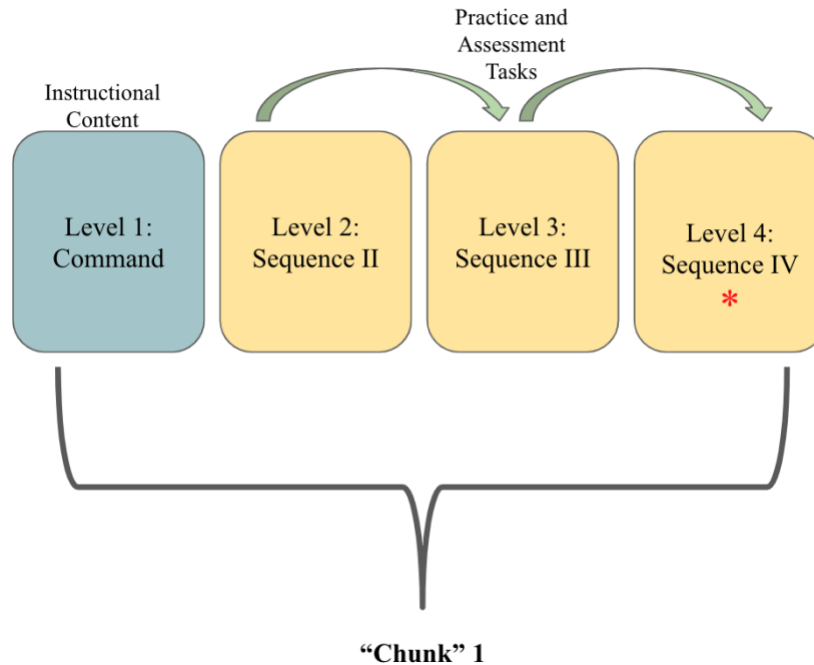


Figure 2. Example of Chunk and Progression of Instruction (Red asterisk denotes the guided switch)

The following pedagogical design was created for the game to determine the best framework for developing a digital learning environment around introducing coding constructs to novice learners while facilitating a collaborative work flow between students as they program in pairs to complete various conceptual and skill-centric tasks. These design principles were developed after a literature review of best practices for teaching and learning programming skills to children and adolescents, then narrowing the focus to practices that aligned or could be modified to comply with the fusion of the instructionist and constructivist framework.

Table 1. Description of Game Design Principle Components

Principle	Description
Collaborative Learning	Facilitate learning activities that support the exchanging of ideas through collaboration and reflective thinking process representative of real-world practices of computer programming.

	<p style="text-align: center;">Implementation</p> <p>Paired programming: MCA provides prompts to focus how pairs begin to communicate about a task when they seem stuck on a problem or after a <i>chunk</i> is complete, they are prompted to reflect on their process and workflow. Additionally, the structure of paired programming alone offers a context similar to how real programmers work on problems in unison (Thomas, Ratcliffe & Robertson, 2003; Salleh, Mendes, & Grundy, 2011)</p>
Self-guided Experiential Environment	<p style="text-align: center;">Description</p> <p>Create a context that supports the ability for self-guided exploration and practice in an engaging and progressively complex environment representative of real-world practices of computer programming.</p>
	<p style="text-align: center;">Implementation</p> <p>MCA is a self-guided learning experience for learning introductory programming skills. Instruction and problem-based learning are combined to mitigate the typically constructionist pitfalls of completely sandbox systems that lead to logical misconceptions and require more direct instruction.</p>
External Learning Tools	<p style="text-align: center;">Description</p> <p>Include external cognitive tools, exercises or assessments for students to gain more support and practice outside the context of the game.</p>
	<p style="text-align: center;">Implementation</p> <p>Corresponding/conceptually aligned practice questions in MCA workbook (Vogel et al., 2006; Wouters et al., 2013)</p>
Scaffolded Units of Instruction	<p style="text-align: center;">Description</p> <p>Provide scaffolded learning tasks to support students transitioning from one concept to the next in developing a deeper understanding and working knowledge.</p>
	<p style="text-align: center;">Implementation</p> <p>“Chunks” of instruction and practice tasks are integrated throughout the progression of the game. When a concept is introduced students have resources to reference for informing a better conceptual understanding (Robins et al., 2003; Rogalski & Samurcay, 1990).</p>
Tools for Engagement with	<p style="text-align: center;">Description</p> <p>Provide features and tools learners can use to actively engage with their construction of knowledge.</p>

Content Knowledge	Implementation
	Specific to the goals of this research, programming modality is the main feature by which the learner is engaging with this content. Additionally, the learner is given a context in which to practice and build on knowledge after the initial introduction in the game.

Specific Implementation of Pedagogical Framework

When integrating these instructional principles into the development of MCA a modified game-based learning design process from Alasward and Nadolny (2015) was used (see Figure 3). In transferring these principles into features and tools in the game there were certain factors taken into consideration for what to include. A brief overview for the decision behind choosing an avatar, using the “grid” as the problem-space context and integration of certain scaffolds in the game is provided below.

The Avatar and Problem Context: Navigating the Grid with “Bot”

The in-game Bot avatar serves as the method of interacting with the learning environment more directly (Barab et al., 2005). The use of the Bot avatar allows students to execute their code and immediately see the results in action. I would argue that this utility reflects the use of explicit imaginary embodiment. As students are decomposing the problem-space of how to navigate through all the obstacles on the grid, the Bot is able to orient them and help in managing their perception of the task.

Regarding the grid students will have to navigate their Bot through, the goal was to keep this version as open-ended and flexible as possible. The first version of MCA followed the example set by Tynker, Lightbot, CodeCombat, Swift Playground and many other coding games in creating problem-spaces with more or less specific parameters for outcomes. That is to say, there was one path and typically only one or two ways of completing the task. This was setting

students up for narrow-minded problem-solving opportunities, which is contrary to the constructivist framework of this product. Constructivism, as mentioned before, posits that learning is an active process. Particularly, social-constructivism, students not just engaging with new content, but also exchanging understanding and how to apply those concepts in novel ways while another person is what drives the development of mental model – knowledge construction (Werhane, Hartman, Moberg, Englehart & Pritchard, 2009). As students are actively constructing and developing their own mental representations and interpretations of the material. Those mental representations serve to link their knowledge with prior understanding and look to apply the information in different contexts.

Integrating Scaffolds

To assist students in learning basic concepts in-app, specific scaffolding features were employed in the context of this digital learning environment. These scaffolds are presented as 1) visual representations used as examples for how to navigate the console and construct or structure code; 2) an instructional agent that introduces new concepts and provides prompting when guiding students to reference material (such as, the glossary or hint feature); and 3) instructional prompts and in-task feedback. With these supports in place a well-structured student-centered learning experience is created (Devolder, van Braak & Tondeur, 2012; Moreno et al., 2001).

Previous research examining the categories of scaffolding and feedback support suggest that conceptual scaffolds promote a deeper understanding of content material, promote engagement and assist the learner in identifying relevant information in the context of the task they are presented (Devolder, van Braak & Tondeur, 2012; Hannafin, Hannafin & Gabbitas, 2009; Brush & Saye, 2001). As students are introduced to new concepts, skills and practices, these supports can guide students in how to decompose multi-step tasks to make them more manageable

without taking away from the complexity of the task (Vygotsky, 1987). MCA includes conceptual, strategic and interactive scaffolds throughout the student's game-play.

Conceptual and strategic scaffolds assist students in focusing on what it is the task requires, providing guidance as they are introduced to a new concept and determine the best way to approach a complicated task (Devolder, van Braak & Tondeur, 2012; Hannafin, Hannafin & Gabbitas, 2009; Sharma & Hannafin, 2007). Interactive scaffolds were not embedded in the digital environment; however, students will be completing these tasks while paired programming. Students are given the opportunity to construct their own understanding of the concepts with their partner, work through each problem actively through dialogue and maintain an iterative workflow that involves defining the problem and constructing multiple potential solutions.

Additionally, MCA employs gamification elements, with the goal of increasing the student's motivation to complete each task and keep them engaged with the content of each new task. To complete a level, students must navigate through a grid and collect the microchips, they earn points, increasing their "Method" score and receiving challenging tasks that align to their skill level. These elements were not included in the basic version of the application. A list and description of the key features incorporated in the game are provided below.

Key Features in MCA

BotSpeak and Real Code. MCA uses a combination of pseudocode (block-based) called "BotSpeak" and JavaScript (text-based) programming languages. This provides the necessary abstraction of the coding process when students are given a task to complete in block-based language. However, when transitioning to text-based programming, it was important to not only give the perception of "real programming", but to introduce a simplified language that students would be able to recognize in the future.

Instructional Agent. Professor Neo is the instructional agent that introduces new concepts, presents students with prompts and appears when students ask for help. He also prompts students when it is time for them to complete their corresponding worksheet problem or when to notify the researcher to complete their midpoint survey.

Built-in glossary. Specific descriptions and definitions are inputted into the glossary for students to reference when working through the tasks. This allows students to understand the concepts and terminology at a deeper level while they are constructing solutions.

Block Index. The index includes all the blocks presented for use in a task and the ability to switch the index from block-based to text-based view so that students can reference the syntax of a block in text-based format.

Hints and Prompts. Students can receive help along the way if they get stuck on a task. Hints are prompted if a student has attempted a task more than the set threshold for that specific problem. Hints are offered as prompts to look at the glossary, check out the block index or to see a video tutorial on how a solution to a similar problem was constructed by an “alumni of MCA”.

Leaderboard. The leaderboard serves two purposes, 1) it provides individual progress for students to monitor their performance in the game and 2) the board presents mock stats on gameplay to the student of what they are told is live game data from how other students in the MCA universe are performing on any given task.

Time Slider. A step-by-step controller of the speed (faster or slower) code is running after the student has submitted a solution affords the student the ability to detect errors or potential improvements.

Toggle Switch. This feature is only available to students in the free choice condition. It allows the student to freely switch from block-based to text-based programming as often as they would like to within a given task.

Sim Lab. The lab provides a place where students can practice constructing their solutions to a new type of task or practice how to use a new block and run the code without worrying about losing points because of high attempt counts. Any code constructed and run in the Sim Lab is not counted as a final submission for points in a given task. Affording the perceived freedom to make mistakes and try out new practices without the fear of failure.

Potential Implications for Programming Modality

Programming modality is simply the format of code the student is programming a solution with for a given task. In the context of this research, modality is integrated differently between two conditions. The first is the free choice, where students are given the ability to choose which programming modality to code with in a task (block or text). The second is the guided condition, wherein students are automatically transitioned between the two programming modalities. The features implemented in the MCA game are isomorphic regardless of modality condition. However, based on the new design of the grid and changes in user interface tutorials that were not available in version 1, there are several differences in programming behaviors that could be expected based on these developments. Technically, the free choice modality a more flexible tool and in this new environment, based on a recent study that examined programming behaviors in a hybrid programming environment (Weintrop & Wilensky, 2018), it's possible that I will see students leveraging the affordances of the simple block-based language and employing the text-based programming language more fluidly.

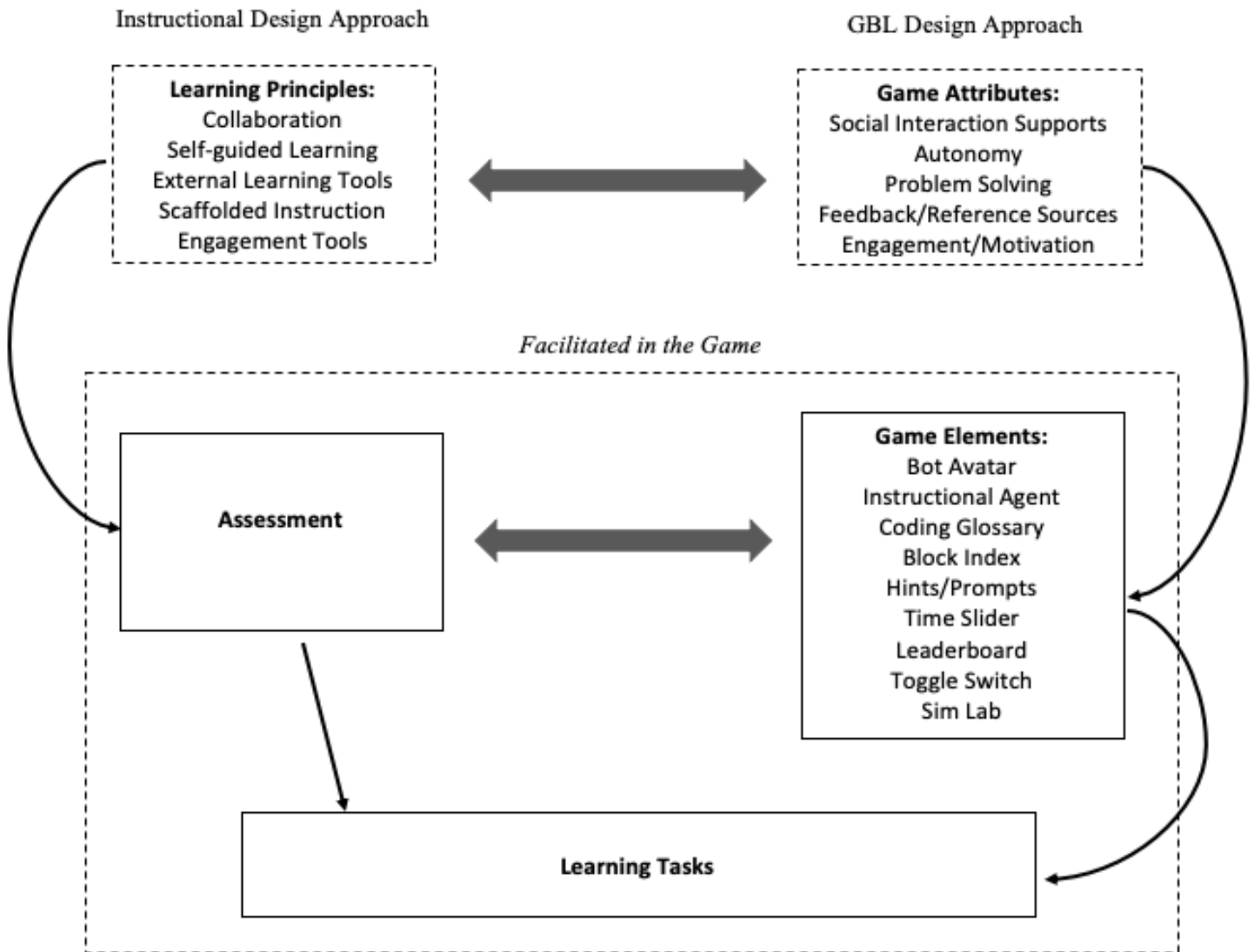


Figure 3. Adapted GBL Design Process for MCA

Table 2: *Terms and Constructs of MCA*

	Sequencing	Decomposition	Iteration	Pattern Recognition	Algorithmic Thinking
Commands	X				X
For Loops	X		X	X	
Conditional Statements	X		X		X
While Loops	X			X	
Nested Constructs	X	X	X	X	X
Debugging		X	X	X	X
Term/Construct	Description				
Sequencing	Specifying a series of steps for a task/identifying the number of steps required to complete the task.				
Algorithmic thinking	Utilizing the correct concept to apply to successfully sequence the solution for the task.				
Pattern Recognition	Denote similarities and differences in code.				
Loops	Loop commands that will continue to repeat until a condition is met OR for a set amount of iterations.				
Simple Conditional	Applying one conditional statement in a solution.				
Complex Conditional	Applying more than one conditional statement in a solution				
Loops/Conditionals	Denoting a pattern and using conditional logic with a loop				
Nesting Conditionals/Loops	Constructing code with more than one conditional within a loop.				

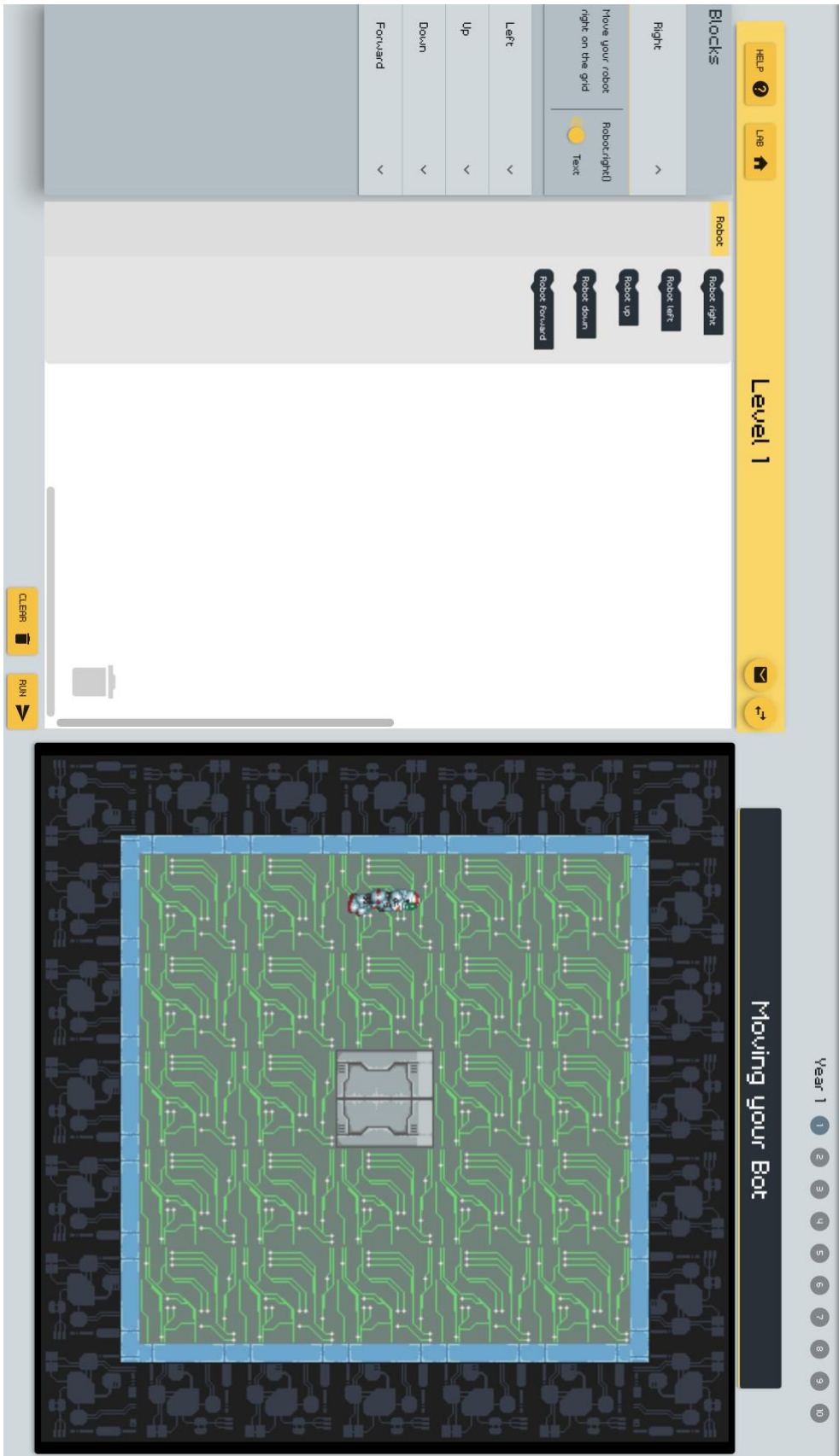


Figure 4. Layout of MCA v2.0 Block-console

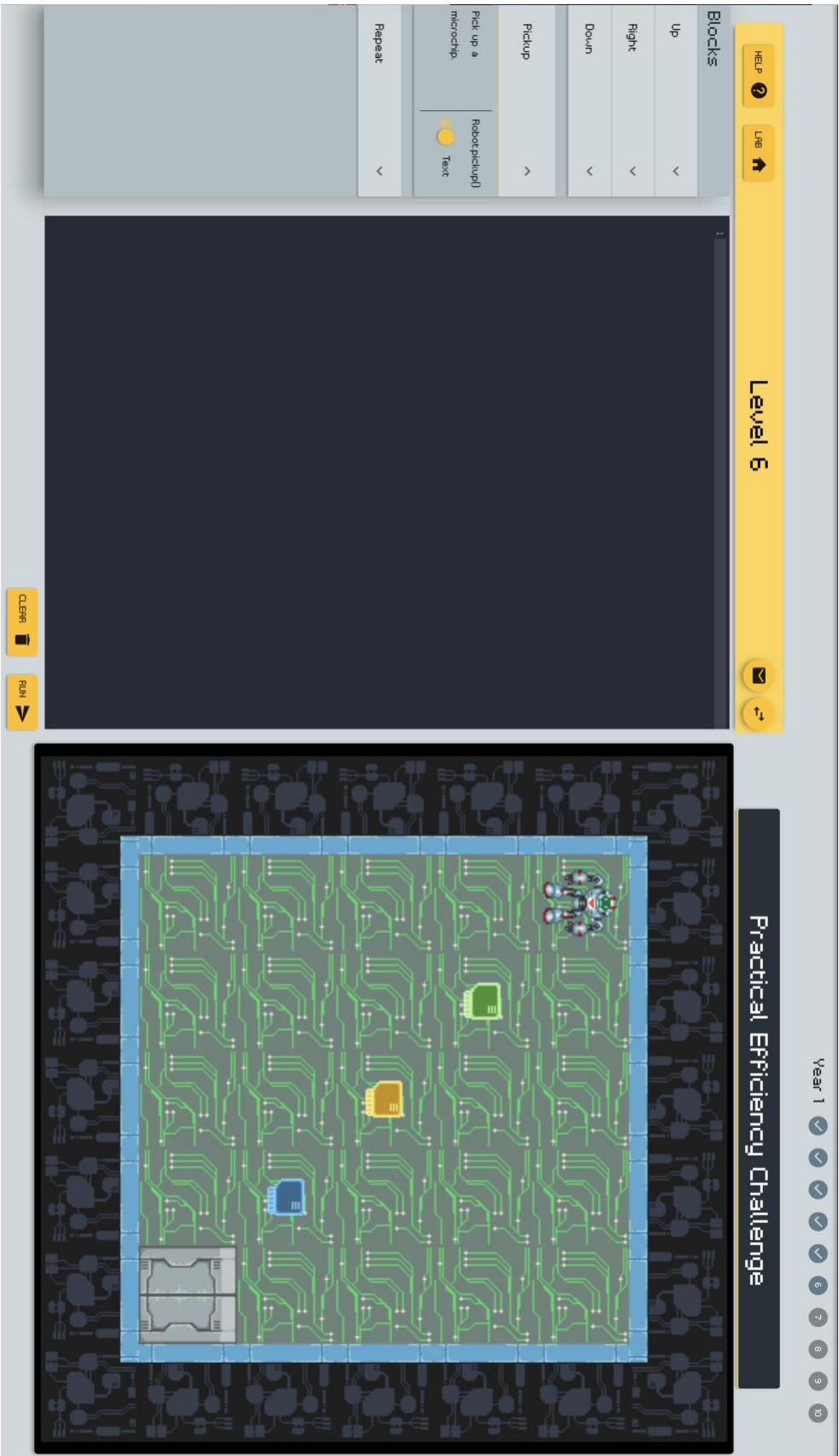


Figure 5. Layout of MCA v2.0 with Text-console

CHAPTER 3: STUDY 1 DESIGN

Research Questions

With the various applications currently available for children, parents and educators to choose from it is necessary to assess how these applications improve coding competencies, effect perceptions of ability and foster computational thinking skills while providing an engaging and interactive context from which to learn the material. The purpose of the current pilot study was to examine whether programming modality effects a student's understanding of basic computer science concepts and if those differences were also impacted by using two different versions of a coding application. Therefore, the following questions will guide this investigation:

RQ₁: Does learning to code within the context of a supplemental educational game-based module result in a greater understanding of basic computer science concepts when compared to a traditional instructional module?

RQ₂: What effect is there on a student's coding knowledge when they are transitioned from block-based to text-based programming over the course of game-play?

The hypothesis was that (1) students who are assigned to the game-based version of the application (Game module) will perform better on the coding learning measures when compared to those who are assigned to the basic version of the application (Basic module) for several reasons. The basic version lacks the engagement and simulated interaction with the instructional agent; therefore, they are less likely to respond to feedback and utilize hints without prompting (Devolder, van Braak & Tondeur, 2012; Moreno et al., 2001). If students in the basic version of the application are not actively seeking a better understanding of the material, they will not perform as well as those in the game version. The second hypothesis was that (2) students assigned to learn how to code in the fading transition from block-based to text-based programming will have a better

conceptual understanding of basic coding concepts due to the lower initial cognitive load of learning how to piece together solutions to tasks without the distraction of a different modality available (Weintrop & Wilensky, 2017). Through the initial scaffolded learning of basic concepts and guided transition to text-based language, the joint exposure and instruction would lend to a deeper understanding when compared with students in the free choice condition.

Methods

To examine differences in learning performance across modality within each module students were randomly assigned to one of two learning module conditions; traditional module (basic) or a game-based module (game), and one of two programming modality conditions; free choice or guided.

Participants

Overall, 68 students were recruited through a summer camp in Rhinebeck, New York. Due to scheduling conflicts for sessions and preferable alternative activities 31 students were withdrawn from participation. However, of the 37 remaining students, three were dropped from analysis because they were unable to complete the module sessions due to unexpected disruptive behavior in camp that resulted in a discontinuation of their final session and five students were removed due to incomplete post-tests. In the remaining sample of 30, students ranged between 10 and 18 years of age ($M_{age} = 12.21$, $SD_{age} = 2.62$), 42% female, 48% male and 10% none. Students participated in two 45-minute sessions. Teacher's College Institutional Review Board approved all procedures and materials; parents gave informed consent and participants gave informed assent.

Conditions

In a 2x2 design, students were assigned to either free choice or guided programming modality conditions and then assigned to the game-based learning module (game) or the traditional

learning module (basic). The descriptions of the two module versions were described previously in Chapter Two but are included below for reference as they pertain to this investigation.

Game Version

The game version of the module is developed around the narrative that the participant is a student at Microcity Academy, a school that teaches recruits how to program so that they can take up computing jobs in Microcity (see Figure 6). For each new level the student's goal was to navigate through a grid and collect the microchips needed to boost their resource library. To do this, students are presented with tasks that are scored based on three components; 1) effectiveness of code submitted (does it solve the problem presented in the task?), 2) efficiency of code submitted (is this the best solution for the task?) and 3) the number of attempts it took for the student to solve the task. Effectiveness is measured in two ways, a) whether the program gets the bot to an exit point of the grid, and b) were the microchips collected prior to completing the task? Efficiency included a count of the number of moves the bot executed, an average of total moves per task was taken from prior user data to calculate a threshold of "efficiency" for each level. Additionally, the pedagogical agent that carried out instructions was more present throughout game-play.

Basic Version

The basic version of the module is developed without a narrative of any kind, the user interface is stripped down and plain and no pedagogical agent even though they receive the same instructional information as students in the game version (see Figure 7). For each new task the student's goal was to navigate through a grid and collect the dots presented in similar variation to the game version, however, the student did not receive points for either of these tasks. However, all the that same information is collected in the data log.

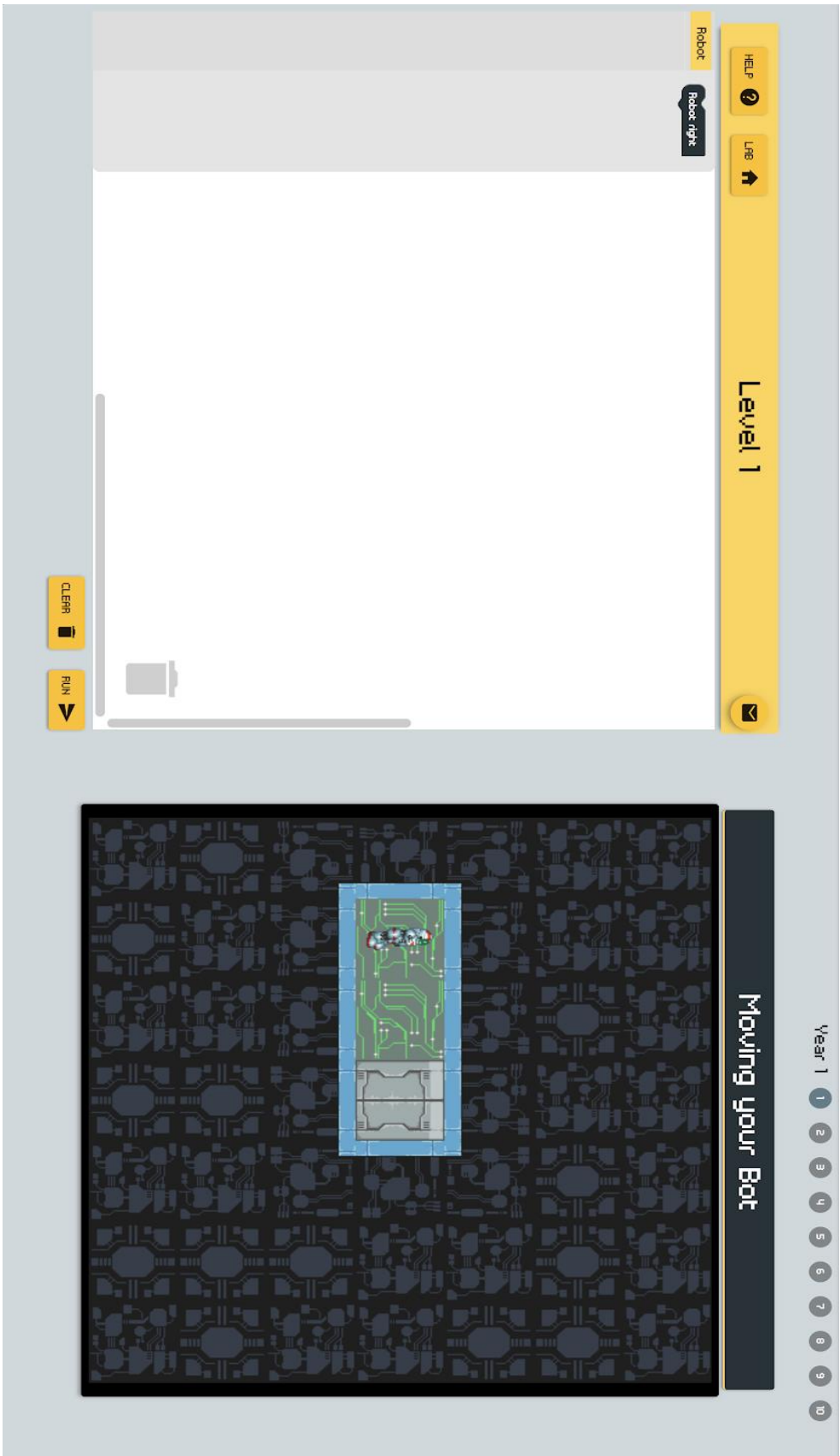


Figure 6. Game Version Example

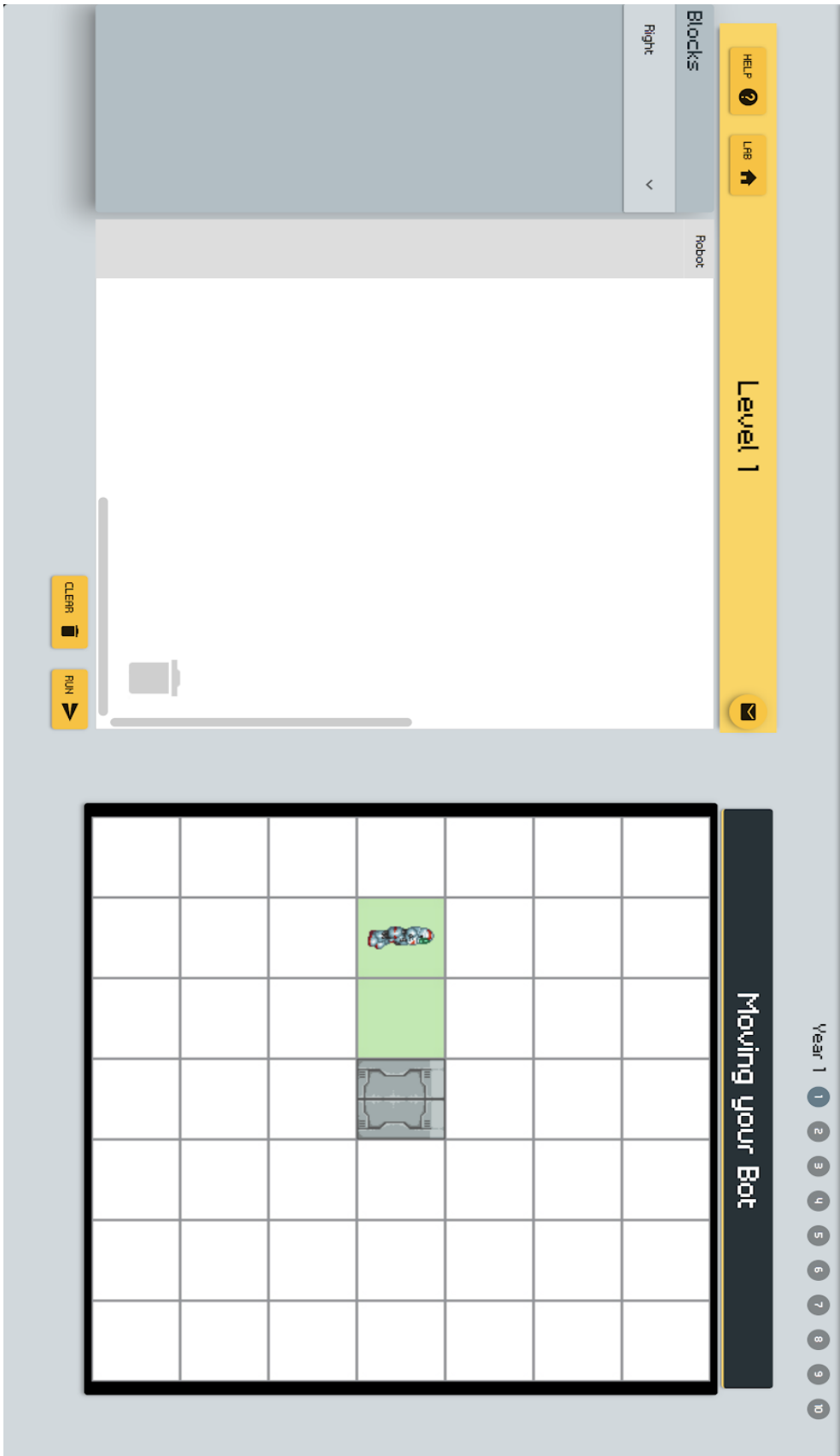


Figure 7. Basic Version Example

Modality Feature

Students were assigned to one of two modalities, guided and free choice. The guided condition guides students through the introduction of new concepts and programming them in block-based language, then transitions them to text-based language to practice those concepts. This condition forces the student to engage with the complexities of syntax and iterative practice in programming. The free choice condition presents the same content in the same sequence as the guided condition; however, the transition is not forced throughout the application. Students in this condition are frequently encouraged to try the toggle switch to see their code in text-based format, but they have that functionality available to them throughout the duration of the session. They can switch between the two language formats as often as they would like and submit their code in either format for any task presented to them.

Included Module Features

The two main resource features available in this first version of the module, across both the game and basic versions, include the block index and the hints or prompts provided throughout the course of the activity.

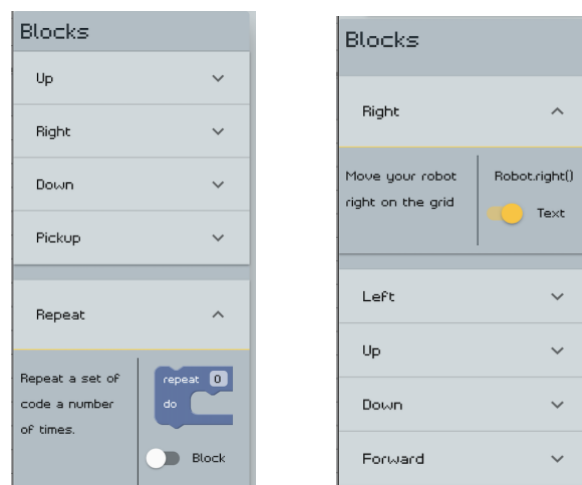


Figure 8. Block Index Examples in Both Language Formats

The block index was included with all the possible blocks available for use in the task the student was working on solving (see Figure 8). It was built to show the student all the available blocks at their disposal and provide descriptions of their usage. The index also allowed the student to switch the descriptions from the block format to the text format of the language so that they could reference the syntax and accurately write their code. Additionally, students were able to access hints related to the tasks to describe the task or to help them understand a newly introduced concept (see Figure 9). Students received or could ask for help along the way if they were stuck on a task. Hints are prompted if a student attempted a task more than the set threshold for that specific problem. Hints are offered as prompts to look at the glossary or check out the block index.

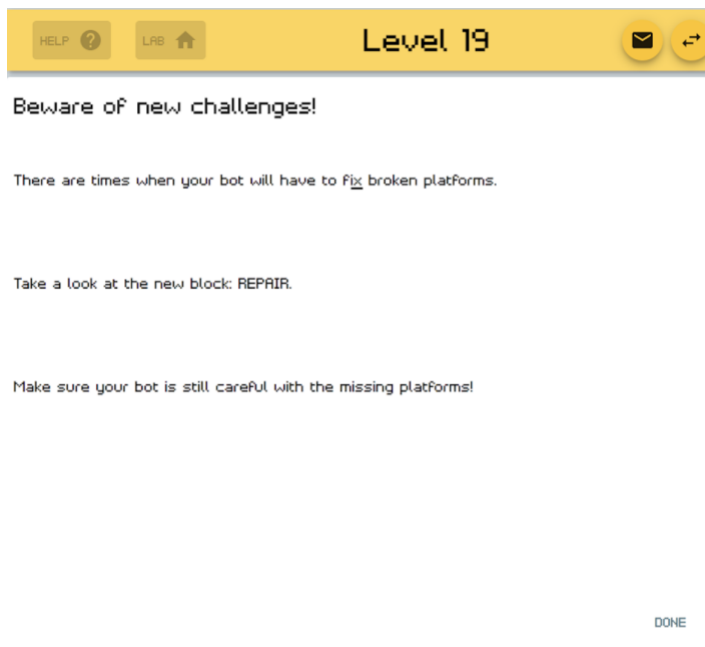


Figure 9. Example of a Prompted Hint in Version 1

Measures

Questionnaire. Students were given a modified version of a prior experience survey (Grover, 2014), asking questions regarding their previous experience with different technology

(e.g. computer/laptop, iPad/tablet, smart phone), frequency of access to those resources (rated on a scale of 0-5, “Less Often” = 0 and “Everyday” = 5) and whether they have any prior experience learning to code.

Learning Measure. Students completed a modified pre-and post-assessment to serve as a baseline for their understanding of basic coding concepts (Grover, 2014). Assessments were not tailored to a specific programming modality, instead, questions were presented for conceptual understanding in both text-based and block-based programming and the assessment was administered regardless of the condition to which the students were assigned. At the conclusion of the last module session students were given a variation of the same assessment to determine whether there were any improvements in their understanding (see Appendix A).

Procedure

Prior to the beginning of each session all laptops were logged in and set for the students arriving. On day 1, students completed an online survey regarding their prior coding experience, technology usage and a pre-test assessing their coding knowledge. On day 2 and 3 students in two 45-minute sessions working through the tasks of the module they were randomly assigned to complete. All students were encouraged to use the resources of each module the entire time to the best of their abilities and to seek help from the research assistant only when necessary. On day 4, students completed the final tasks of the module and were asked to complete a post-test of basic coding knowledge (see Figure 10).

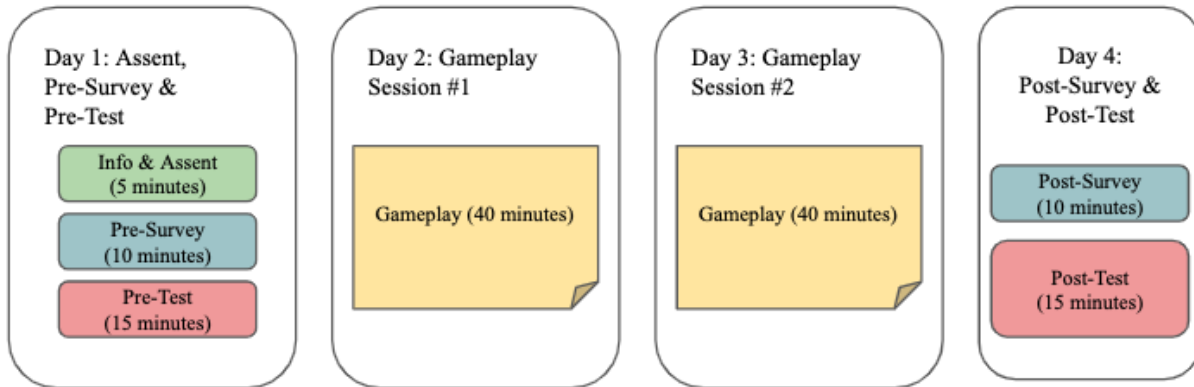


Figure 10. Study Design by Day

Results

Prior Coding and Gaming Experience. Students reported their prior experience with coding using mobile applications or web-based programs, including the development of their own projects, as well as game-play on computers, mobile devices or game consoles. There were no differences in the frequency of technological use by condition. Independent Samples *t*-tests revealed there were no significant differences in learning outcomes between frequent and infrequent exposure to games and none of the students who participated in this study reported having any prior coding experience.

Coding Knowledge Outcomes. Initially, this investigation included a question with regards to the interaction of the two factors, Modality and Module. Specifically asking whether the students in the game-guided condition would perform better on the post-test than students in all other conditions. However, due to the low sample size and the low distribution of participants in each condition, specifically the basic-free choice and game-guided, there are not enough participants in this pilot study to assess whether there is an interaction between programming modality and the learning modules. We can see that there were some observable differences across

all four conditions (see Table 3), however, we cannot reliably test for them. Therefore, all analyses will include comparisons between either the basic or game conditions and the free choice or guided conditions, focusing on the first two research questions.

Table 3. *Marginal Means of Coding Knowledge Difference Scores by Condition*

	Basic	Game
Free Choice	5.50	6.15
Guided	4.55	5.90

RQ1: Does learning to code within the context of a supplemental educational game-based module result in a greater understanding of basic computer science concepts when compared to a traditional instructional module?

To verify that there were no significant differences at baseline for coding knowledge across each condition, an independent sample *t*-test was conducted. Results confirmed there were no significant differences between modality ($M_T = 3.59, SD_T = 2.56; M_S = 3.10, SD_S = 1.56$) or module conditions ($M_B = 3.27, SD_B = 1.39; M_G = 3.42, SD_G = 2.56$) at their coding knowledge baseline, indicating that all participating students were at relatively the same novice level with regards to programming.

All analyses included responses to all assessment questions across all question types (conceptual, writing and debugging). An independent samples *t*-test was conducted to compare learning differences scores from pre- to post-test between students in the basic and game module conditions. There was a statistically significant difference ($t(29) = -2.367, p = .029$), students in the game-based version of the application ($M = 6.08, SD = 1.67$) showed more improvement than those in the basic condition ($M = 4.77, SD = 1.41$) from pre- to post-test by an average of 1.31

points (see Figure 11). This confirms H_1 that students in the game module condition would demonstrate more conceptual understanding of basic coding constructs than students in the basic module. This suggests that by presenting the same content in a game-based environment there is an underlying motivational aspect to the module that may be more engaging than the way the content is presented in the basic module.

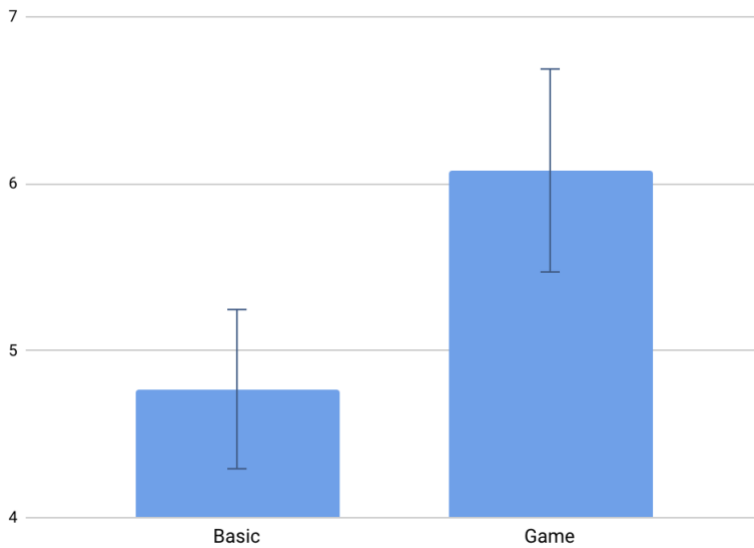


Figure 11. Average Difference Scores by Module Condition

RQ₂: What effect is there on a student’s coding knowledge when they are transitioned from block-based to text-based programming during the progression the module?

An independent samples t -test was conducted to examine learning outcome differences from pre-to post-test between students in the guided and free choice conditions. Results showed there was not a statistically significant difference between modality conditions ($t(29) = 1.756, p = .087$), however, Cohen’s effect size value was calculated ($d = .633$) suggesting there is a moderate potential for significance. Contrary to the expected outcome, students in the free choice condition ($M = 6.03, SD = 1.43$) showed more growth from pre- to post-test than those in the

guided condition ($M = 5.00$, $SD = 1.80$) by an average of 1.03 points (see Figure 12). This outcome supports the opposite of what was expected in H₂ in that students from the guided condition were expected to show more growth in their understanding. There may be several reasons that the free choice condition performed better, for instance, this may be due to the more flexible work environment of the toggle feature when compared with the more structured transition of the guided condition. Ideally, the application would be equally instructional, however, the work console in which students are programming and learning how to apply concepts practically, the guided condition can be defined as more restrictive than the free choice. This may impact perceptions of ease for learning, change student engagement, or perceptions of self-competence for students in the guided condition when tasks are more difficult, and concepts are not as obvious to apply (Wouters et al., 2013; Vogel et al., 2006). Initially, it was expected that students who were automatically transitioned from block-based to text-based would be more aware of the resources available to them when completing a task (block glossary, code index, hints, tutorials, etc.). Log data on general game behaviors will contribute to understanding these differences in Study 2.

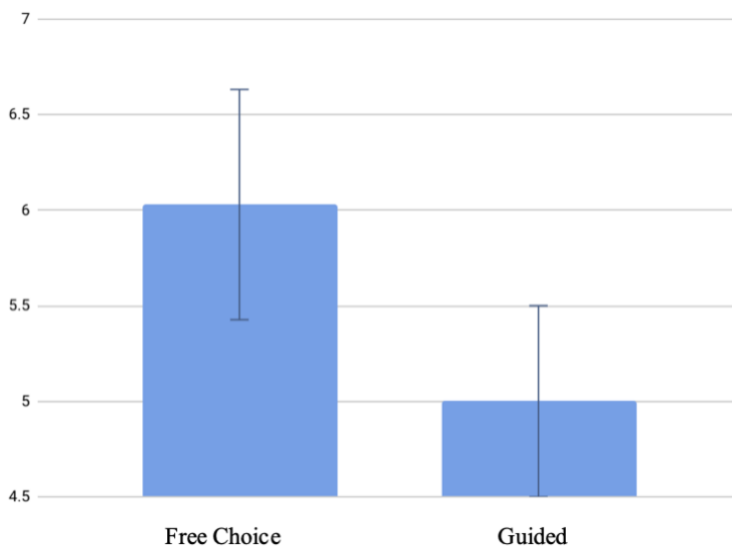


Figure 12. Average Difference Scores by Modality Condition

Exploratory Analysis

Reviewing the log data, there are observable differences in the manner in which students in the free choice condition confront a coding task. Students who utilize the toggle feature would often times switch over from block-based to text-based programming an average of five times on each task. The additional exposure to both styles of the language may have contributed to their higher growth performance on the learning measures.

There were some unexpected patterns when reviewing submissions for each task, there is an observable progression of how often students are submitting code with complex concepts of programming, such as, conditional statements or any loops and even going so far as to nest their code rather than hard-code their program (see Figure 13). The dip at level 13 marks when the concept of conditionals is introduced, and it is expected that students in the guided condition would perform better than those in the free choice condition because they are being guided through that concept in blocks. However, we can see an overlap in the frequency of complex code submissions. While this pattern is indicative that students are understanding those more difficult concepts while they continue to practice throughout the trajectory of the module, we can see that there is still approximately half of the sample that continued to use simple blocks or text to solve tasks. Moving forward into the final design, intermittent check-in surveys on student perceptions of ease for learning the content would be interesting to collect and use to align with log data of game-play.

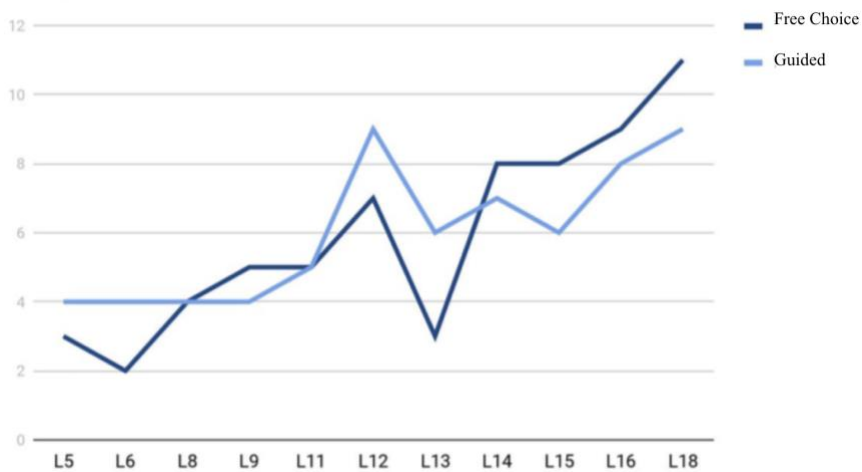


Figure 13. Frequency of Student Final Submissions using Complex Code

Study 1 Discussion

Limitations

There were several limitations to executing Study 1. In the initial 2X2 design, a third research question of this investigation asked whether students in the game-guided condition would outperform all other conditions on the learning measure. However, due to the low sample size and the low distribution of participants in each condition, specifically the basic-free choice and game-guided, there were not enough participants in this study to assess that condition interaction. Several factors contributed to a limited sample size, such as, the setting of this pilot study. With a camp environment there was a high level of attrition among participants due to the extracurricular activities and commitments involving camp festivities. This led to incomplete module sessions, missing assessment data and required students to be removed from analyses. Additionally, there were instances during game-play when behavior was a challenge when students were trying to complete their module session. These distractions led to incomplete sessions and disengaged participants.

Conclusions and Implications

This study sought to understand the relationship between programming modality and its effects on how well students learn and understand basic programming concepts using a supplemental instructional game-based tool. Previous work investigating modality as a feature of learning in any programming environment suggests that there is more to explore regarding blended or hybrid programming environments to assist in the transition from blocks-based to text-based programming languages (Tabet, Gedway, Alshikhabobakr, Razak, 2016; Weintrop & Wilensky, 2015; Weintrop & Wilensky 2018a; Weintrop & Wilensky 2018b). This pilot study focused on developing a digital environment to allow for the comparison of blocks-based to text-based programming modality to take place within consistent content. In pursuit of a basic understanding of whether student learning in block-based to text-based programming environments differs when given the choice to switch between the two modalities and when being guided instructionally between the two modalities, the following research questions were asked:

RQ₁: Does learning to code within the context of a supplemental educational game-based module result in a greater understanding of basic computer science concepts when compared to a traditional instructional module?

RQ₂: What effect is there on a student's coding knowledge when they are transitioned from block-based to text-based programming during the progression the module?

The findings of this pilot study suggest that there is a difference in how students perform on the learning measure based on the programming modality to which they are assigned and the supports of the environment. The outcome that students in the free choice condition outperformed those in guided was not expected. There could be several reasons for this difference, the guided condition is more rigid than the free choice condition in that the transition from blocks-based to

text-based programming is forced. This does not lend well to the tinkering process of coding that takes place in the free choice condition based on what the log data shows about how they utilized the feature. This might indicate a need for a design change to the levels to make the problems less structured and more open-ended to offer students a more flexible environment regardless of modality. Further exploration of student perceptions during game-play is needed to understand how the ease of the tasks and utility of the features in the tool are contributing to the student's coding process. Additional analyses examining game behavior differences between modality conditions is needed and might explain why the students who had free choice in this study seem to be outperforming those who were being guided from blocks-based to text-based programming.

With the growing interest in the importance of exposing student to computer science concepts, skills and practices while also fostering computational thinking skills, there remains a need for continued study of the process by which those skills may be fostered in education (Grover & Pea, 2013). Specifically, how does the modality of the program they study impact how they perceive their learning experience and their ability to learn more complex skills in the future. The goals of this study were two-fold, there was a need to examine whether there would be any learning differences between the programming modality conditions, would students who were being transitioned from blocks-based to text-based programming perform better on a coding learning measure? Do they have more exposure to both modalities? Do students use the toggle feature to check their work and therefore, is it a tool for reflection? Discerning how the feature is utilized may be indicative of their performance and potential differences in understanding from the students practicing with the guided feature.

Based on these findings and with these questions in mind, Study 2 only included the game version condition of the application to focus on the condition of interest, programming modality.

Specifically assessing the utility and effectiveness of programming modality used as a teaching tool in this coding application for novice programmers with little to no exposure with the content. The follow up study included attitudinal measures before, during and after game-play, post interviews to see what kinds of behaviors students employ in their coding when given a challenging level with new blocks to learn and apply based on their knowledge from the game-play session. Additional adjustments to game-play sessions included structuring the coding tasks in pairs to serve as an external feature of instructional support and the use of content aligned worksheets with questions for the pair to reflect on and submit with each game-play session. Providing students with resources and an external activity that aligns with the practices of the digital curriculum should assist in maintaining the focus of the pairs as they are working through each level of the game (Vogel et al., 2006; Wouters, van Nimwegen, van Oostendorp & van der Spek, 2013).

Overall, Study 2 compared the effect of programming modality, free choice and guided, on learning measures, student self-perceived competence regarding ability to learn new coding concepts, as well as, their level of engagement and enjoyment during game-play sessions.

In addition, paired programming and the external assignment may contribute to student's appraisal of each problem, resource-usage, reflection on the content and how they apply information to each task in the game to produce a better conceptual understanding of basic CS concepts and practices.

CHAPTER 4: STUDY 2 DESIGN

This study seeks to demonstrate the potential affordances of two models of a blended programming environment combining the engagement and support of blocks-based systems and the perceived authenticity of text-based programming tools. The goals of Study 2 include comparing the effect of programming modality, free choice and guided, on learning measures, student self-perceived competence regarding ability to learn new coding concepts, perception differences of the game, as well as, their level of engagement and enjoyment during game-play sessions. With the focus of determining whether programming modality effects students' attitudes towards computer science and programming, perceptions towards the game and of their own abilities to learn the concepts and apply them to new challenging problems.

The results of Study 1 suggest that there is a difference in learning of basic coding knowledge tied to the programming modality with which students are learning to code. However, discerning what behaviors students elicit in the game to contribute to those learning differences, whether it is feature-usage or basic interaction with the console, is relevant to measuring the effect of modality when learning introductory programming skills and concepts. The participants in Study 1 were a wide age range of young children and adolescents who were recruited from a summer camp. There was minimal structure and a high level of distraction during the sessions that led to difficulty collecting completed session game data. However, in reviewing basic attempt and submissions of code by level, there was a surprising pattern in the complex code submission from those in the free choice condition. As the level of difficulty increased so did their submission of complex solutions for those tasks. While that difference between free choice and guided participants was not significant observable coding behaviors and patterns are worth further exploration. The paired programming protocol and the external assignment may contribute to

student's appraisal of each problem, resource-usage, reflection on the content and how they apply information to each task in the game to produce a better conceptual understanding of basic CS concepts and practices.

Overview

My hypothesis is that by focusing on the comparison between the programming modalities available in the game, I can expect to see behavioral differences in coding practices between the two conditions. Additionally, the external collaborative and interactive components of paired programming and the content-aligned worksheets will help students develop a deeper conceptualization of the CS constructs and gain an external experience that will contribute to their overall in-game learning experience, therefore, grounding their formal understanding as they are developing their own mental models of these concepts. I expect that by organizing how new knowledge is presented to them in and outside the game, both by interacting with the module and with their partner, students will have a better grasp on how to apply what they've learned to more difficult tasks and feel they are capable of learning how to do so. The goal of Study 2 is to investigate whether programming modality effects how students perceive their learning experience in a blended programming environment and whether that impacts their learning of basic computer science concepts and elicits computational thinking behaviors.

Research Questions

The research questions Study 2 include:

RQ₁: Does being transitioned from blocks-based to text-based programming foster a deeper understanding of basic programming concepts in the game and therefore result in better performance on posttest, debugging, transfer and challenge tasks than participants in the free choice condition?

RQ₂: Are there differences in general game behavior between condition in their approach when solving the game levels, such as using hints or the help glossary, using more attempts per level, or using the lab to practice their solutions?

RQ₃: How do participant perceptions of the game and their own ability to learn how to code differ by modality condition?

The primary question of this research is RQ₁, whether transitioning students from blocks-based to text-based programming fosters a deeper understanding of basic programming concepts and leads to better performance on new challenging tasks? I expect that students learning to code in a game with a feature designed to transition the user from blocks-based to text-based programming will develop a deeper understanding of basic CS concepts and coding skills than students who are in a less guided design of the same content.

Prior research on programming modality and the findings of Study 1 posit that, regardless of modality condition, students learn from this type of introductory instruction. However, research examining whether a blended or hybrid programming environment is more effective in teaching novice programmers basic coding skills has been limited. Previous investigations used comparable digital platforms which students are switched between environments built for block-based language with text-based components that students are not able to directly manipulate. Therefore, developing an application wherein students can be exposed to the exact same content in the same sequence while having the ability to construct code in both modalities is an asset to this work.

Furthermore, the comparison between the free choice and guided conditions across equal content will allow me to discern whether being given the choice to practice in both block-based

and text-based programming languages freely or a guided transition is more effective in teaching novice programmers. My hypotheses for this research question are as follows:

H_{1a}: Participants in the guided condition will perform better on measures of conceptual understanding than those in the free choice condition.

H_{1b}: Participants in the free choice condition will perform better on measures requiring debugging code.

H_{1c}: Participants in the guided condition will perform better on the two challenge tasks.

H_{1d}: Participants in the guided condition will perform better on the transfer tasks.

RQ₂ asks whether there is an observable difference in game behavior, captured in log data, between the programming modality conditions when approaching a new task? General game behaviors include the number of attempts needed to solve each level, the number of hints used, the count of usage for the coding glossary, the count of usage of the Lab Sim, and full submission blocks or text from each level attempt. These resources are available to both conditions, however, the frequency and manner in which they are used will be interesting to map against the new challenge tasks post game-play. My hypothesis for this research question is as follows:

H₂: The guided condition participants will be more likely to use the resources when confronted with a difficult task due to the restriction of the modality they are in when given the task, compared to participants in the free choice condition, which are able to switch between modalities during difficult concepts and tasks.

RQ₃ asks whether participant perceptions of the game and their own ability to learn how to code differ by modality condition. Using a different modality impacts how you perceive the ease of learning a new coding concept, therefore, it should follow that the perception of how easy it is to learn to code, as well as, the level of enjoyment with the game would differ by condition

(Powers, Ecott, & Hirshfield, 2007). However, there is no prior evidence that there would be differences in perception of competence as there has yet to be a true comparison between these two types of blended programming environments. Although, modality research indicates that block-based programming is thought to be initially easier to understand and leads to higher reported feelings of self-efficacy, however, there are studies where text-based programming was seen as a more authentic method of programming as opposed to learning in the limited block-based format (Weintrop & Wilensky, 2017; Weintrop & Wilensky, 2015; Tabet et al., 2016). Regardless, I cannot be certain of an outcome for participant perceptions within this application or overall experience. Therefore, I will conduct an exploratory analysis without specifying a hypothesis.

Study Design

Participants

A total of 163 students with no prior coding experience were recruited to participate in this study from a combined middle and high school (grades 8th, 10th, 11th and 12th) in Puerto Rico. Of the 163 students recruited, eight students were removed from the study due to lack of attendance and three did not complete the game play sessions and were dropped from the final analysis. The remaining 152 students consisted of 54% male, 47% female and ranged in age from 12 to 17 years old ($M = 14.88$, $SD = 1.59$). This school is placed in the bottom 50% of all schools in Puerto Rico for overall test scores with 86% of the student population eligible for free or reduced lunch. Students participated in the study as part of their regular 45-minute science class period (either biology, chemistry or physics) where 78 students were randomly assigned to the guided condition and 74 were assigned to the free choice condition.

Procedure. This study employed a randomized design, with randomized assignment of students within classes to one of two programming modalities: The Free Choice (F) condition or

the Guided (G) condition. Both conditions played the game with isomorphic content, however participants in the free choice condition had the ability to switch between blocks-based and text-based programming, the same way as participants in Study 1. Participants in the guided condition played a version of the game that transitioned them from blocks-based programming to text-based in a sequence of content tasks automatically built into the development of the levels. In this condition there is no switching back and forth between the programming modalities.

Game Module Version 2

There were several new inclusions and updates made to the version of the game used in this study. Two new additions to the game were the instructional agent, Professor Neo, and an enhanced integration of the avatar. Updates were made to the block index and the hints or prompts and an additional glossary was included to help breakdown the syntax and usage descriptions with examples.

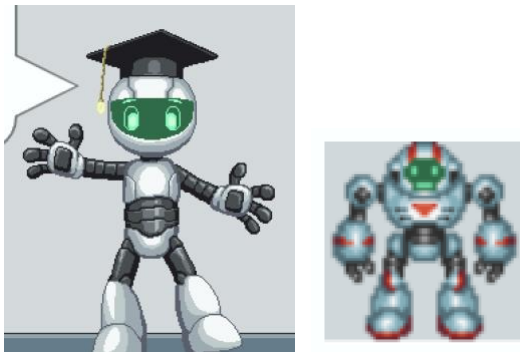


Figure 14. Instructional Agent “Prof. Neo” and Robot “bot” Avatar

Professor Neo is the instructional agent that introduces new concepts, presents students with prompts and appears when students ask for help (see Figure 14). He also prompts students when it is time for them to complete their corresponding worksheet problem or when to notify the researcher to complete their midpoint survey (see Figure 15). The avatar, referred to in the game as “bot”, was enhanced to improve the students’ navigation perspective by allowing the students

to see the grid from the bot's initial point of view then code the solution for the task. This decision was made based on concept of surrogate embodiment, wherein the manipulation of the "surrogate" presents the perspective of the learner (Black, Segal, Vitale and Fadjo, 2012).

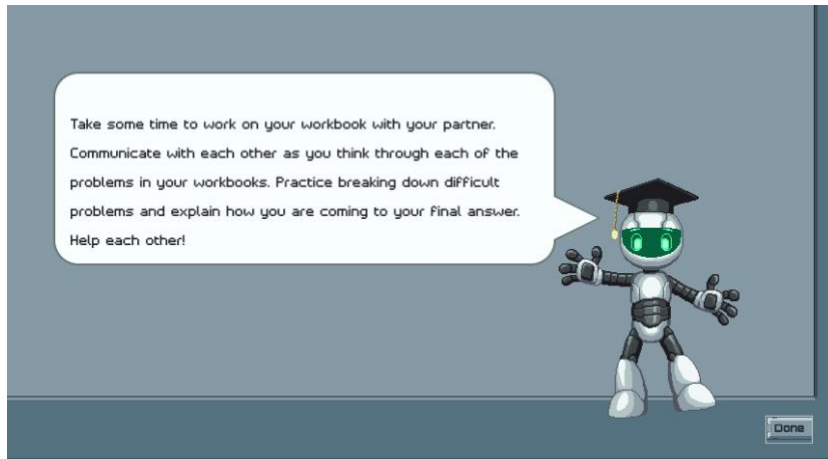


Figure 15. Example of Prompt for Process Worksheet in Version 2

The block index was included with all the possible blocks available for use in the task the student was working on solving (see figure 16). It was built to show the student all the available blocks at their disposal and provide descriptions of their usage. The index still allowed the students to switch the descriptions from the block format to the text format of the language so that they could reference the syntax and accurately write their code.

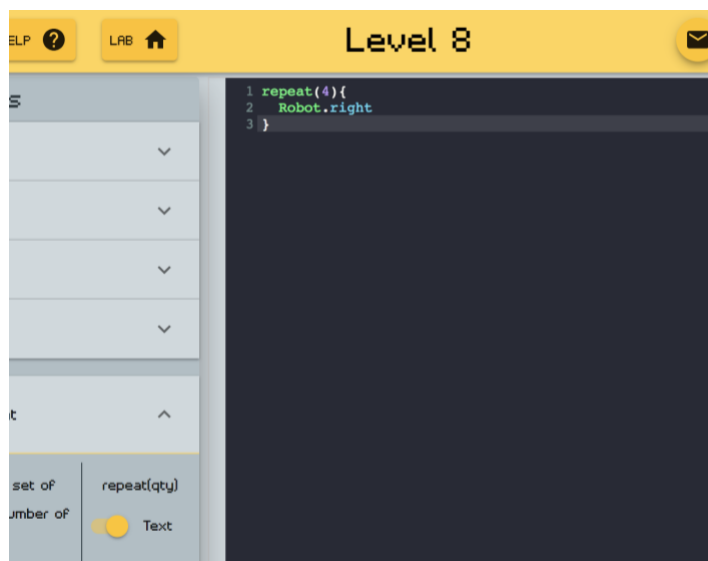


Figure 16. Block Index Usage Example

The built-in glossary was designed as an extension of the block index. The glossary is meant to provide more specific descriptions and definitions for students to reference when working through the tasks. The main intent behind this feature is to provide students with some additional assistance in developing a better understanding of the concepts and how to implement them at a deeper level when constructing solutions (see Figure 17).



Figure 17. Example of Built-in Glossary

Lastly, students were still able to access hints related to the tasks to describe the task or to help them understand a newly introduced concept. However, in addition to receiving help on a given task the hints were changed to include feedback after each attempt and automatically offer support based on the detected error. For example, a syntax hint would be prompted if a student attempted a task more than the set threshold for that specific problem with the same type of syntactical problem (see Figure 18).

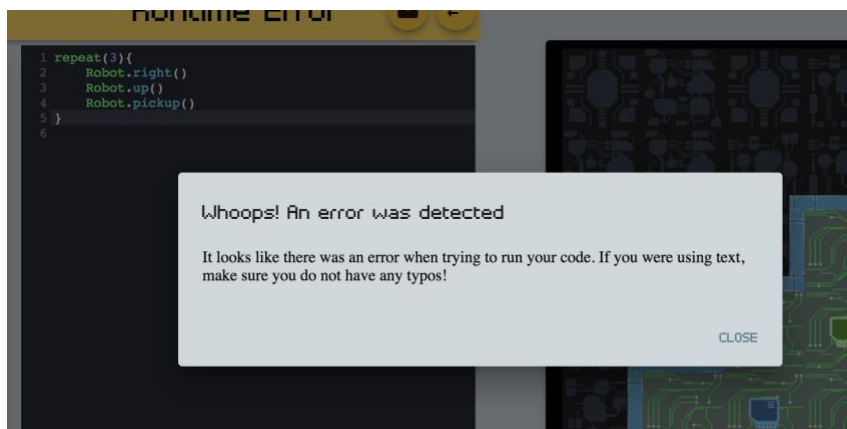


Figure 18. Example of Prompted Hint in Version 2

Study Activities and Timeline

Students participated in the study activities over the course of 4 to 5 days. Day 1 consisted of administering a 10-minute pre-survey that asked participants to report their prior coding and gaming experiences, their perceptions regarding the field of computer science and programming, confidence in their ability to learn how to program and how they expected to do during the sessions. They also completed a 15-minute pre-test before beginning the first 5 levels of Microcity Academy which would fill the last 20 minutes of the class period.

On Day 2 participants continued game-play for the full period. Game-play was completed over 2 to 3 sessions as was previously determined based on the findings of a meta-analysis on educational games indicating that more than one session of play is a more effective learning treatment (Wouters et al., 2013). Additionally, as participants were working through the game levels, they also completed a 10-minute mid-survey and the corresponding worksheet which aligned with the sequence of the concepts and tasks introduced.

Day 3 and 4 included administering the 15-minute post-survey asking isomorphic questions about their self-perceived competence in understanding the basic coding concepts and their perceptions of games' utility for learning and level of engagement in completing the activity, as well as their paired programming experience using Microcity Academy. Participants also completed a posttest and challenge task, during which some participants were randomly selected participants from each class group were asked to complete the two challenging levels during an individual 15-minute interview (see Figure 19 for an overview of the study components).

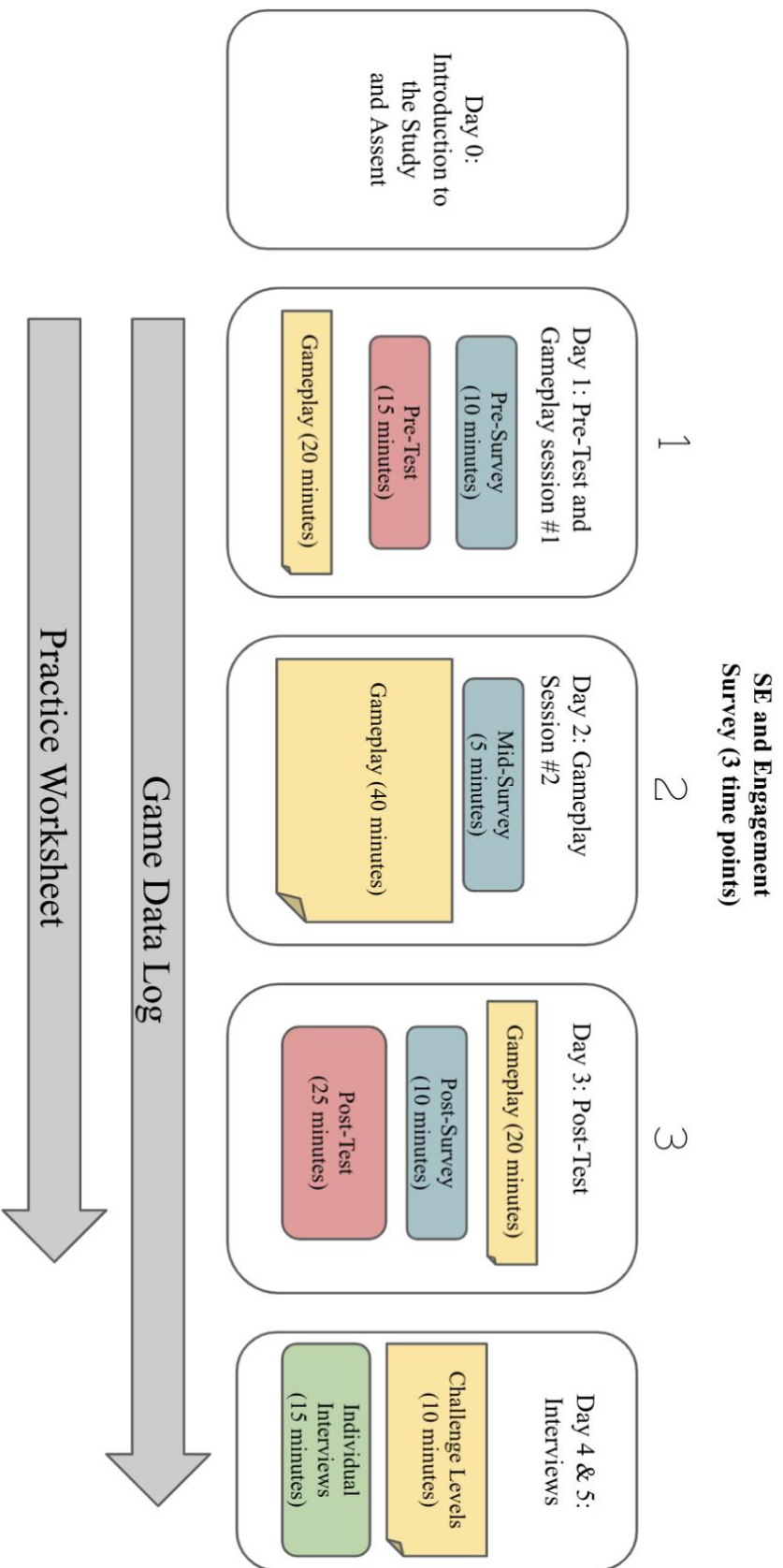


Figure 19. Study 2 Design Overview

Study 2 Measures

Learning measures. The learning measures from Study 1 were modified and used for this study, a student's coding knowledge was examined using a pre- and post-test. Participants were asked questions regarding a) general concepts, b) comprehending/writing code, and c) debugging corrupted code. Pre-test was administered before the start of game-play and post-test at the conclusion of their last game-play session (see Appendix H). Responses to multiple choice items for conceptual knowledge and program comprehension, scored correct, partially correct or incorrect. Written responses explaining code or writing the program were scored based on a rubric of acceptable responses.

MCA Corresponding Worksheet. Participants received a corresponding worksheet with questions that aligned with concepts being introduced in-app but required them to critique "corrupt" code and explain their reasoning for those suggested changes or if there should not be changes made to that code. Participants were asked to use the resources they have in-app, as well as their partner to work through the questions as they progress through the game. Prompts in the game noted when they should take the time to answer one of the problems, for instance, the instructional agent might pop up on a module and state "*Now that you know about sequencing, try to solve the first question on your worksheet to practice this new skill!*". This worksheet was not scored as a final outcome measure but serves as an additional instructional support for the pair as they navigate through the first few difficult concepts.

Transfer measure. The transfer CT task asked that students attempt to solve two difficult problems that required decomposing the question and iterative problem solving. The first problem was based in the context of a chemistry. Students were asked to imagine they were given two containers, one that could only hold 4ml of water and the other 7ml of water, both are blacked out

with tape and you can only tell if they are full or empty. Students were tasked with listing the steps they would take to get one of those containers to hold 5ml of water. The second problem was based in mathematics, students were given two words problems and asked to write out an algorithm for the solution or to indicate the pattern within the problem.

Challenge measure. The challenge task was administered at posttest and consisted of two difficult challenge levels in a more constrained grid environment that would limit the ways that students could successfully reach the goal. Both modalities were used, in the first challenge students were only able to submit their code in text-based language and in the second they were only able to utilize the blocks-based language. All resources and instructional supports were available to them throughout the course of their attempts. All students were administered the two tasks; however, some were randomly pulled to interview while completing the tasks. To perform well on either level, students had to use the tools and resources provided to them by the application as they were given this measure to complete alone without their programming partner.

Behavioral measures. Log data from the game was used to capture participants' coding strategies and behaviors, including the number of attempts needed to solve each level, the number of hints used, the usage number of the coding glossary, the usage number of the Lab Sim, and full submission blocks or text from each level attempt (this was not something available Study 1). Additionally, the log data specifically from the free choice condition was analyzed for how often participants switch between modality and how often solutions were submitted in text versus in blocks.

Attitudinal Survey. A modified attitudinal survey (Weintrop, 2016) was administered at three instances in the duration of the study (pre-game-play, mid-game-play and post-game-play). The pre-survey asked questions to determine prior exposure to coding and games on various

devices, their perceptions of computer science, programming and their own ability to a) learn to code and b) complete a coding task. The mid-survey had a combination of multiple choice and open-ended questions pertaining to their perceptions of the game and the learning tasks, working with their partner and how often they use resources in-app. A similar survey was administered at the last session without the prior experience questions, to assess any changes in their perception of coding competence, their experience with paired programming, overall enjoyment of the game, and thoughts on continuing to learn about computer science.

Study 2 Results

The models used in the analyses of this study accounted for the pairs effect on all outcome measures. Due to the fact that students were playing the game and learning in pairs, their individual performance on all outcome measures may not be independent of each other. This is not a new problem, where students have learning together, but are assessed at the individual level. To address this question of independence between measures within each pair, intra-class correlation was calculated for each outcome measure including, the learning posttest, post debugging task, the transfer task and the challenge task (see Table 4). All but one outcome measure had a significant ICC based on the cutoff point of significance of a two-tailed p -value of $< .20$ (Kenny, Kashy, Cook, & Simpson, 2006). However, regardless of this result it was determined best to account for the pair effect across all outcome measures.

Table 4. *ICC Between Pairs on Outcome Measures*

Measure	r	F	df	Sig.
Learning Posttest	0.53	3.30	75,76	$< .01^*$
Post Debug	0.18	1.44	75,76	.06*
Transfer Task	0.04	1.09	75,76	.36

Challenge Task	0.15	1.36	75,76	.09*
----------------	------	------	-------	------

Note. * $p < .20$

Due to the task structure of this study, students learning in pairs, most measures were assessed using a linear mixed-effects model with pair as the random effect. This model is a hierarchical linear model, where level 1 models fixed effects including the effects of modality condition, pretest scores and pre debugging scores on students' posttest scores or transfer, post debugging and challenge scores. To address the pairs effect, level 2 will model pairs as a random effect on the intercept for level 1.

$$\text{Level 1: } Y_{ij} = \beta_{0j} + \beta_1 \text{Condition}_{ij} + \beta_2 \text{Pretest}_{ij} + \beta_3 \text{Predebug}_{ij} + e_{ij}$$

$$\text{Level 2: } \beta_{0j} = \gamma_{00} + u_{0j}$$

Posttest Performance Outcomes

Age and Learning. Age was not included in these analyses as it was not found to have a significant effect for any learning measure. However, there was a significant inverse correlational relationship between age and the transfer task, $r(152) = -.165, p < .05$, suggesting that student who were older did not perform as well on this task as younger students. Though, when a linear mixed-effects model was run with condition and age at the first level as fixed effects and pair modeled on the second level as a random effect there was no significant age effect between conditions, $t(152) = -.108, p > .91$.

Learning. To assess learning at posttest, a linear mixed-effects model was run with condition and pretest scores at the first level as fixed effects and pair modeled on the second level as a random effect. There was a significant main effect of condition, $t(77.02) = 2.26, p < .05$ and of pretest score, $t(148.32) = 2.96, p < .01$. Therefore, modality did affect posttest learning scores,

more specifically, students in the guided condition ($M_G = 17.22$, $SD_G = 6.22$) performed better on the learning measure than students in the free choice condition ($M_F = 14.38$, $SD_F = 5.32$) (see Table 5).

Debugging. Modality condition did not affect performance on debugging tasks at post-test. To evaluate debugging task performance at posttest, a linear mixed-effects model was run with condition and pre-debugging scores at the first level as fixed effects and pair modeled on the second level as a random effect. There is no main effect of condition, $p > .90$, and there was no effect of pre-debugging score, $p > .10$. Coding modality does not seem to have an effect on how students performed on the debugging tasks (see Table 5).

Transfer. To evaluate student performance on the transfer task at post-test, a linear mixed-effects model was run with condition at the first level as a fixed effect and pair modeled on the second level as a random effect. There was a main effect of condition, $t(152) = 5.16$, $p < .001$, where students in the guided condition ($M_G = 5.10$, $SD_G = 1.80$) performed better on the transfer tasks than those in the free choice condition ($M_F = 3.66$, $SD_F = 1.66$) (see Table 5).

Challenge. Modality condition also had an effect on how students performed on the challenge task at post. To evaluate challenge performance, a linear mixed-effects model was run with condition at the first level as the fixed effect and level 2 modeling pair as the random effect. There was a main effect of condition, $t(78.43) = 2.61$, $p < .01$, where students in the guided condition ($M_G = 7.84$, $SD_G = 2.01$) performed better on the challenge tasks than those in the free choice condition ($M_F = 6.96$, $SD_F = 1.99$) (Table 5). Overall, the students in the guided condition outperformed the free choice condition and the same can be seen when the tasks are split and assessed individually.

Table 5. Means (SD) of Posttest Performance Outcomes by Condition

	Learning	Debugging	Transfer	Challenge
Condition	Posttest Score	Posttest Score	Task Score	Task Score
Guided	17.22 (6.22)	7.69 (3.90)	5.10 (1.80)	7.84 (2.01)
Free Choice	14.38 (5.32)	7.54 (3.95)	3.66 (1.66)	6.96 (1.99)

Game Data Log and Survey Outcomes

Behavioral outcomes. In order to test whether there were differences in game behaviors between conditions, a MANOVA was run with conditions as the between-subjects factor and the general types of game behaviors logged in-app as outcome variables; average attempt count, average hint usage, average lab sim usage and average glossary usage. These counts were pulled from the three shifts in the game where students were asked to submit a solution for a learned concept in a different modality before moving on to the next chunk. There were two game types with condition effects, hint usage $F(1,80) = 10.49, p < .01$ and glossary usage $F(1,80) = 8.07, p < .05$. Lab sim usage and attempt count showed no significant condition effects $p > .170$ even though it seems that the students in the free choice condition used the lab sim more often, on average, and submitted more attempts than students in the guided condition. In contrast, students in the guided condition used the reference resources more frequently across all 3 points than students who were able to toggle during the tasks (see Figure 20).

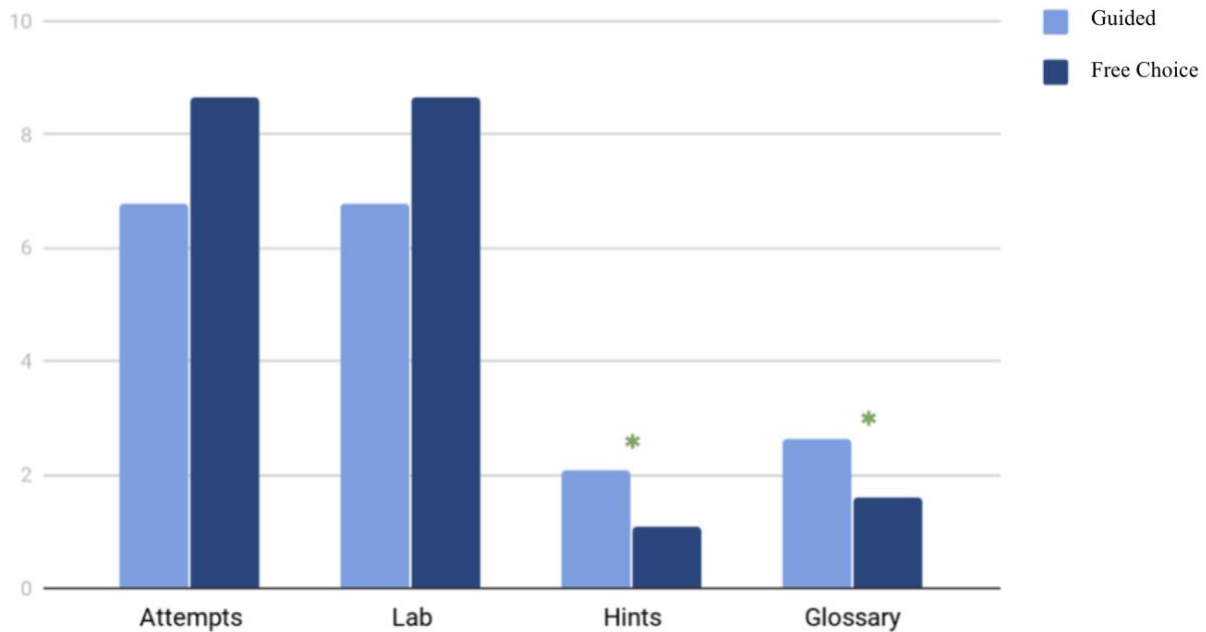


Figure 20. Marginal Means of Game Behaviors by Condition

Attitudinal Survey. A linear mixed effects model was used to test whether there was growth over the cross of game-play in a students' perceived self-competence score, with Time as a 3-level within-subjects factor and programming modality condition as a between-subjects factor at level 1 and the random effect of pair modeled at level 2. There were no significant condition effect, $p > .86.$, however, results show a significant effect of time, $t(377.88) = 11.04, p < .001$ and an interaction effect, $t(377.88) = 3.50, p < .01$. Additionally, this model was run with engagement scores across the three points the survey was administered to determine changes in student engagement across game-play. There was a significant main effect of condition, $t(415.89) = -2.01, p < .05$. Additionally, there was an interaction effect of time and condition, $t(355.09) = 23.18, p < .01$, where multiple comparisons between conditions across the three time points show that the guided students reported higher engagement from time 1 to time 2 and were consistent in their reporting at time 3 (see Table 6). However, there was a significant difference between condition

at time 3, $F(1, 136) = 11.32, p < .01$, where the students in the guided condition reported higher engagement than those in the free choice.

Table 6. *Means (SD) of Survey Outcomes by Condition*

Condition	Perceived Self-Competence			Engagement		
	Pre	Mid	Post	Pre	Mid	Post
Guided	15.59 (2.77)	22.94 (4.37)	26.15 (2.01)	11.19 (1.78)	12.12 (1.84)	12.65 (1.75)
Free Choice	14.12 (2.86)	20.08 (5.39)	21.45 (4.20)	11.47 (1.64)	11.64 (1.59)	11.58 (1.76)

Study 2 Discussion

Results for study 2 showed that being guided from blocks-based to text-based programming improved coding knowledge, transfer task performance and challenge task performance. Hypothesis 1 was confirmed, in that students who were guided performed better than those who were given free choice on the learning, transfer and challenge tasks. This finding is contrary to the results in Study 1 where students in the free choice condition showed a higher performance on the learning task than those who were guided. Several differences in Study 2 may have led to this change in outcome. The instructional supports implemented in the change of game design were meant to support both conditions equally, however, the guided students were able to utilize them more effectively when forced to program in a different modality. This gave them a forced opportunity to learn concepts in *both* modalities where they could practice key concepts and practices on multiple tasks. Thus, making them more equipped to respond to the questions on the learning measure, and apply those practices on the transfer and challenge tasks.

However, when predicting outcomes for the debugging task, Hypothesis 2 was not confirmed. Students performed relatively the same regardless of condition. There could be

several reasons for this result; one might be that while students were given opportunities to practice debugging code in the game, it was not an explicit skill they were being taught. Another reason might be that debugging as a skill is very difficult and is not something that can be trained and easily applied with this level of play dosage.

Additionally, Study 2's version of the MCA game included a variety of instructional support tools and resources that version 1 did not provide. Usage across these resources differed by condition, showing that students who were guided used the hints and glossary more frequently than the students toggling. However, students who were given free choice made frequent use of the sim lab where they could practice and submit their code without a penalty to their score. It seems that even though the free choice condition made use of that resource, unless paired with the instructional references provided by the hint prompts and the glossary index, there was not a huge gain when it came to their performance on the learning outcome measure.

Lastly, when looking at student reported perceptions of their engagement during the game across their experience, all students reported high levels of engagement across all 3 time points, however, only those in the guided condition continued to increase. Reported perceptions of student self-competence showed that the guided condition increased significantly across all three points while those in the free choice condition increased from time point 1 to time point 2 and remained fairly consistent at time point 3. This outcome supports the assumption that by providing those additional resources, students who were guided between modalities experienced an increase in their perceived ability to learn more challenging coding concepts. Similarly, students in the free choice condition showed initial growth in their perceived self-competence but reported no change after the last session of game-play.

When comparing the learning outcome results to those of Study 1 where students from the free choice condition outperformed those in the guided, it is important to note that the changes to the game structure accommodated for gaps in all of the students' ability to effectively work through each task in the game. However, the relationship between condition and feature usage with regards to how the guided students made use of their resources in comparison to how the free choice students used theirs is important to distinguish when interpreting these results and determining the implications when designing this type of learning application.

Overall Discussion

This work sought to understand whether supplemental games and applications developed for basic programming education for the 6-12 student population are effective learning tools. Additionally, this research examined whether the modality by which students are learning has an effect on their learning experience and their usage of the educational tools. In Study 1, there was a significant difference in learning performance for students in the game condition and this was a finding implemented in the design of Study 2. In Study 2 only the game was used and only the guided and free choice conditions were manipulated to better compare learning outcomes between those two learning styles in the context a supplemental educational game. However, despite the previous study to determine the best methodology for moving forward with Study 2, there were some limitations that should be addressed in future work.

Limitations

There are several limitations to this study that require a deeper examination in future research. Specifically, the debugging task had a technical difficulty that was not noticed until just before the study began. The tasks were initially provided in the application; however, the tasks were not active which resulting in students completing the tasks in paper format, which is not

conducive to iterative problem-solving for a coding task as its' best. However, students were given multiple copies of each task to complete so they could work through the task iteratively to an extent.

Additionally, the survey measures were provided prior to first game session, during the middle of the second game session and at the end of the third game session. While this was not an issue at pre- and post-test, making students stop their game-play to distribute the survey was problematic. Ideally, students would receive these questions fluidly in the game when they are not mid-task. Several issues came up regarding time in task and made mid-game time data unreliable as an on-task indicator.

Lastly, while students were able to complete the transfer tasks successfully, it may not be the best measure of transfer for computational thinking skills and would be difficult to align with practices and behaviors from one context to the other. Especially when considering the difficulty of working iteratively on a paper and pencil measure. However, students were given multiple copies to iterate their solutions as often as they needed. Regardless, future work in developing a better measure of transfer that can be coded and scored effectively based on observable CT behaviors and practices rather than simply correctness and efficiency of response may be a preferable and more sensitive method of measuring CT skills across contexts.

Conclusions and Implications

In summary, findings suggest that programming modality matters and has a significant impact on how novice programmers learn, develop CT practices and solve new challenging coding tasks. Learning to code is not only becoming more popular, it is *currently* the new must-have skill of the 21st century. However, while the vast variety games, applications and online courses are striving towards teaching coding for all, few studies are focusing on what is going to

make younger novice programmers able to learn how to code and transfer those skills to other contexts.

Programming modality is a small facet of the computer science research presently available. Discerning differences in how students learn depending on the modality in which they are learning had yet to be truly compared within the same application. This work suggests that students can learn basic computer science concepts and programming skills in a simple block-based programming modality and be able to apply those concepts successfully in a text-based programming modality without the explicit instruction of a full course. Supplemental learning for K-8 is how this field will pique student interest, not just in programming, but in the skills, they develop and practice overtime through game-play.

Additionally, a potential implication of this work relates to the age of the students in Study 2. Age had no effect on their performance outcomes and while there is little research regarding what level students across the 6-12 age range should be performing at in a supplemental coding game, the performance typically varies. However, in this case, there as only a weak correlation between the transfer task and student age. The lack of effect may indicate that there is no advantage to being older when prior experience across all ages is the same. This poses the question of how developers and educators work to level out the learning experience for students of all ages?

For instance, a student in 12th grade should have the skills and experience to be able to learn this new content and outperform a student in 8th grade. Though that is not what is reflected in the findings of this research, it is what we might developmentally expect. However, based on the current study's results, students in both middle and high school are learning at relatively the same level when accounting for baseline knowledge. This outcome would suggest that early

exposure is key to gaining these skills. Calling for alternative design and feature building for students at different ages regardless of the lack of coding background. More research on the effect of age on how students learn to code and how well they can apply that knowledge across contexts is needed to determine whether this is a result that may better inform game developers and collaborating educators when designing new games across age groups.

Finally, this research, in general, can provide a framework for how to implement novice coding curriculum and instructional resources into supplemental educational game design. This work can now begin to assist in addressing the gap between blocks-based learning and text-based programming. Students who are exposed to new concepts in blocks-based programming languages and master those concepts can apply that same knowledge when confronted with a similar, more challenging task, but in a text-based programming language without explicit instruction in a game. This design allows for students to try and fail in a constructive way and to learn meaningful practices that will serve them well in other contexts.

CHAPTER 5: FREE CHOICE CONDITION INVESTIGATION

This case study examines how students of the free choice condition are using their toggle feature to learn, build and practice their programming skills throughout the course of their gameplay. In Study 2, there was a performance difference between students who were guided and students who were given free choice. However, this case study aims to investigate whether there are differences within the free choice condition. By comparing two students from the free choice condition with extreme differences in performance on the learning measure, this study will investigate what components of game play and feature-usage lead to success or failure within the free choice condition.

Method

Procedure and Measures

This is an investigation which includes all outcome measures from Study 2 along with all log data and video data that was taken of the computer screens of students who were interviewed at the end of the last day of the study. Only interview data for the free choice condition was pulled for this case study. The interviews took place in a separate area of the science classroom and was modeled after the paired programming structure of the game-play sessions in which the researcher was the driver (initially starts to program a solution for the task) and the student was partnered with researcher as the navigator (tells the driver what to code).

Students were told the following at the start of each challenge task; “I would like you to look at this task. Try to solve it, but step-by-step, tell me what you are doing. Pretend I’m your partner and you have to tell me how to write the code. What’s the first thing you want me to do?”. The goal of each task was similar to that of the tasks presented in the game, produce a program that will navigate the robot to the door on the other side of the grid (see Figure 17)

Participants

The analysis for this study was done in two parts with subset populations of the overall sample of Study 2. The first set of analyses includes only the students assigned to the free choice condition, $N = 74$ (42 low performing and 32 high performing). For the second part, only one student was selected from each performance condition of the free choice subset. However, the interviews were aligned with log data from the students' game-play and needed to be linked with their performance outcome scores. Therefore, each pair outcome scores were averaged together and coded for high and low performance with the same score indicator, $N = 37$ (21 low performing and 16 high performing).

The first student selected for the higher performing subset is Laney and the second student from the lower performing subset is Justin (all students names used in this study description and in transcripts are pseudonyms). These students were selected because they scored on the extremes of their subset group, Laney, age 14, scored a perfect 30 while Justin, age 16, scored a 5.50 on the learning posttest.

Table 7. *Scores on Outcome Measures for Comparative Analysis*

Student	Posttest	Transfer	Challenge
Laney	30.0	2.0	10.0
Justin	5.5	5.5	6.0

General Analysis

For this portion of analyses, the total sample was restructured only to include students in the free choice condition. Within that condition, students with a score of 15 or lower on their posttest performance score were recoded as lower performing as they will have succeeded in scoring half

or less than half on the total content. Students scoring more than 15 points were recoded as high performing and this was marked as the *performance condition*.

Each performance outcome measure will be reevaluated with the new subset sample. Though this is a subset of the original sample, the students still learned in pairs and therefore, pairs will be accounted for in a linear mixed-effect model as the random effect at the second level of each model run.

Posttest Performance Outcomes

Debugging. While modality did not affect performance on debugging tasks at posttest, there is an affect when examining differences between the high and low performing students in the free choice condition. To evaluate debugging task performance at post-test, a linear mixed-effects model was run with performance condition and pre-debugging scores at the first level as fixed effects and pair modeled on the second level as a random effect. There was a main effect of performance condition, $t(74) = -3.56, p < .01$, and there was no effect of pre-debugging score, $p > .80$. Students in the higher performing category ($M_H = 9.31, SD_H = 3.27$) within the free choice modality seemed to perform better on the debugging tasks than those who performed lower ($M_L = 6.25, SD_L = 3.92$) on the posttest (see Table 8).

Transfer. To evaluate student performance on the transfer task at post-test, a linear mixed-effects model was run with performance condition at the first level a fixed effect and pair modeled on the second level as a random effect. There was no main effect of performance condition, $p > .05$ ($M_L = 3.62, SD_L = 1.70; M_H = 3.66, SD_H = 1.64$) (see Table 8).

Challenge. Although when comparing modality conditions there was an effect on how students performed on the challenge task at post-test, that is not the case when comparing the high and low performers in the free choice condition. To assess challenge performance, a linear mixed-

effects model was run with performance condition at the first level as the fixed effect and level 2 modeling pair as the random effect. There was no main effect of performance condition, $p > .90$, where both high and low performers scored relatively the same on the overall task ($M_L = 6.93$, $SD_L = 1.79$; $M_H = 6.97$, $SD_H = 2.24$) (see Table 8).

Table 8. Means (SD) of Posttest Outcomes within Free Choice Condition

	Learning	Debugging	Transfer	Challenge
Performance	Posttest Score	Posttest Score	Task Score	Task Score
Low	10.65 (3.39)	6.25 (3.92)	3.62 (1.70)	6.93 (1.79)
High	19.22 (2.76)	9.31 (3.27)	3.66 (1.64)	6.97 (2.24)

Game Data Log Outcomes

Behavioral outcomes. To address the question of whether different coding behavior patterns throughout game-play can impact learning measures in a variety of ways, taking a closer look at how students in the free choice condition differ in how they use the application may lead to uncovering interesting patterns. A MANOVA was run to examine differences in resource usage within the free choice condition between high and low performing students. Performance condition was the between-subjects factor with the general types of game behaviors logged in-app as outcome variables; average attempt count, average hint usage, average lab sim usage and average glossary usage. Since log data was used for this analysis there was no need to account for the random pair effect as the game was played using one login per pair and was logged as one outcome. Averages were computed for hints, glossary use, lab sim use, attempts and toggle switches. These averages were pulled from the three shifts in the game where students were asked to submit a solution for a previously learned concept in a different modality before moving on to the next chunk.

The is a significant effect of average toggle switches, $F(1,41) = 6.26, p < .05$ and a trend for lab sim usage, $F(1,41) = 3.69, p = .06$. Average Hint usage, attempt count, and glossary usage showed no significant effects $p > .440$. Further investigation into the how and when students are toggle switching, a repeated measures ANOVA was run to determine where those differences appear across the three most difficult shifts in the game-play. There was no significant main effect of toggle switches across these three levels found between high and low performers, there was only a trend, $p = .079$, with an interaction of $p = .05$ (see Figure 21). Though it is not significant, there is an observable interaction between performance category and toggle switches between level 13 and level 17, where higher performing students are toggling slightly less for level 17.

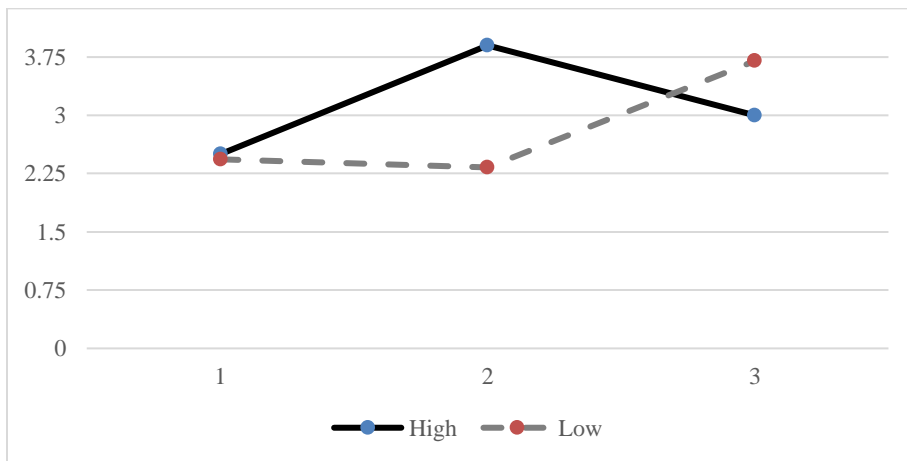


Figure 21. Marginal Means of Toggle Switches per Level (Difficult Levels)

This behavior is mirrored by both the highest and lowest performers selected from the subset (see Figure 22). Initially, both students are submitting their code and toggling at the same rate, however, at the second shift of content at level 13 (time point 2) higher performing students are toggling more than those with lower performance. Referring to Laney’s log data for level 13, she switches from text to blocks and begins to place the sequence she knows should work. However, rather than submit immediately she toggles between the two modalities three more times to switch out a repeat block. It seems she is not sure how to implement it in the program.

Regardless, she submits and is given the following feedback, “*Oops, seems you missed something. Give it another try!*”.

Laney removed the repeat block and just hardcoded the program. She runs that code and it receives an error feedback, rather than fix it she has removed the longer program and has toggled an additional three times to determine if she has successfully implemented the *repeat* block again. She successfully submits her code. Unlike this iterative behavior Laney is exhibiting, Justin immediately toggles back to blocks and hardcodes his solution using the basic blocks from the initial content chunk (forward, down, left, right, jump, and pickup) without trying to incorporate the new loop block.

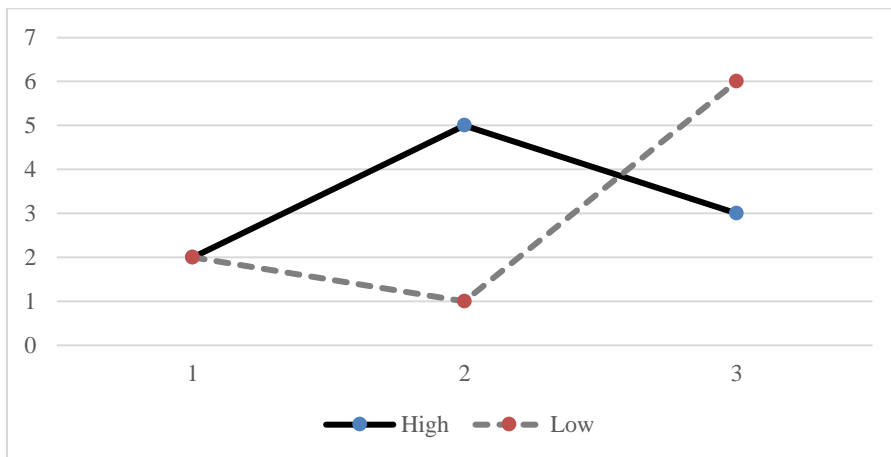


Figure 22. Counts of Toggle Switches per Level (Difficult Levels)

However, on level 17 (time point 3) we can see the opposite is true. When referring back to the data log, Justin’s attempts to submit on this level only once, but is actively using the sim lab to submit his code five times. Laney makes five attempts on this level after using the lab six times. During this task, students are prevented from using basic code blocks and syntax even when they toggle over. They are forced to try using the *repeat* or *repeat until* code to achieve success on this task. For Laney, this doesn’t seem to require much toggling for her, however, she does practice about the same amount as Justin in the sim lab. Though Justin is submitting in the

sim lab the same number of attempts, he's using his toggle feature differently. He's only toggles over to add in the *repeat* block and then switches back to type in the code he already knows.

Both of these students used this feature very differently when faced with a difficult task, in this case, new content and code to manipulate at the same time. Without having each student explain each action, it is not certain whether the toggle feature itself is what really sets them apart or if it's the function of that feature in conjunction with how the student is utilizing the other resources simultaneously.

Exploratory Analysis

Computational Thinking Behaviors Index

Interview videos were analyzed using a CT behavior index developed for this research from the CT facets and definitions framework provided by Shute, Sun, and Asbell-Clarke, (2017). The goal of this index was to code for counts of observable CT behaviors and language exhibited by the students during the challenge tasks. 43 students were interviewed and due to the novelty of this index two researchers blind coded approximately 35% of the data and achieved, an inter-rater reliability of $\kappa > .70$ except for decomposition behavior (Table 9). Items where inter-rater reliability could not be achieved, all data was coded by two raters while discussing all disagreements. For all items where inter-rater reliability was achieved, one master coder coded all the data.

While two raters coded for decomposition throughout all 43 interviews, due to the disagreements brought up in discussion, it is an inherently subjective item and cannot be reliably coded without supporting language from the student being interviewed during the challenge tasks with descriptions and explanations for their decision-making process. Therefore, it is included in these analyses, however, findings in relation to that item should be considered with caution.

Table 9. Kappa Values for Computational Thinking Behavior Index on Challenge Tasks

CT Behavior	Challenge 1	Challenge 2
Iteration	0.82	0.81
Pattern Recognition	0.70	0.90
Decomposition	0.58	0.61
Algorithmic Thinking	0.72	0.71

Interview: Challenge Tasks

Students were asked to complete two challenge tasks that were similar to the tasks they had seen in the game with a few differences. The blocks and syntax would have new commands they had not yet used, and the grid would be simpler, limiting the possible solutions. The goal of each task was similar to that of the tasks presented in the game, produce a program that will navigate the robot to the door on the other side of the grid (see Figure 23).

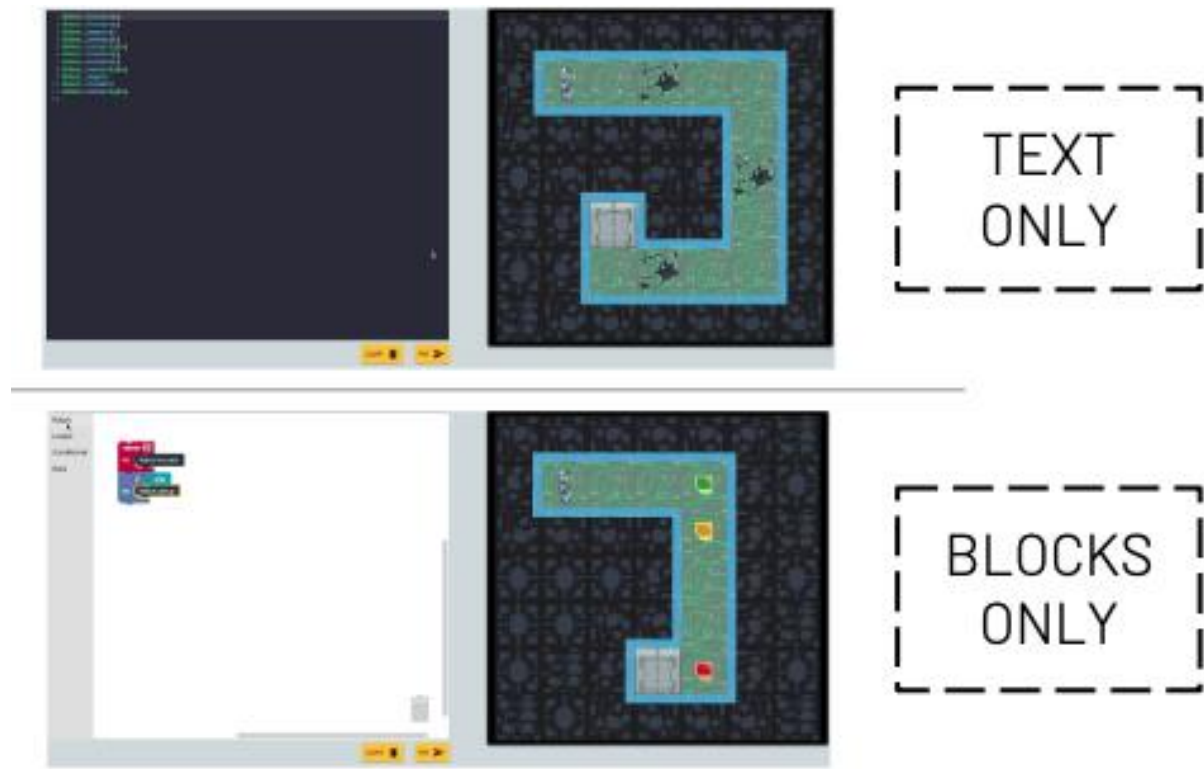
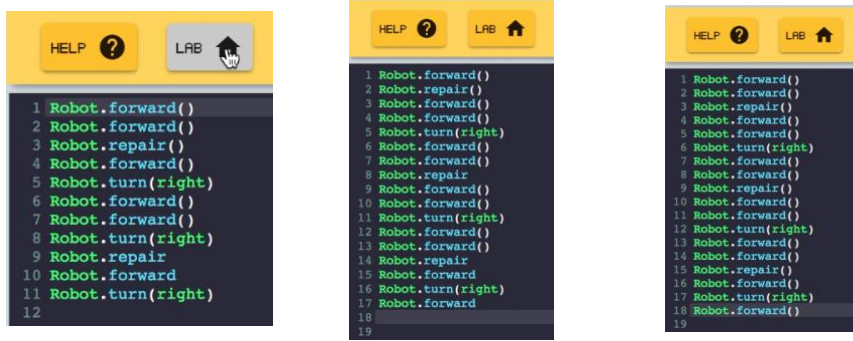


Figure 23. Challenge Tasks

Challenge Tasks

During the first challenge, students were given a text only task to complete with a platform they had not yet seen in the regular game-play sessions (see Figure 16). There are several ways in which students were able to complete this task. A student could complete the task using basic sequencing of simple commands or using the higher-level concepts, such as, conditional or repeat commands to get the bot to the door at the end of the path. There are differences to how each level of performer completed this task and how they work within the restrictions of the challenge.

The lower performing free choice student's play on the first challenge task was straight forward and quickly broken down to two steps (see figure 24). Justin initially noted that the program was too short to reach the end of the path and that at line five there was a missing command that would lead to the bot not being capable of executing the right turn. Justin went through the code line by line pasting in the correct code command. Eventually, he was able to produce a correct solution for the task. However, he did not take the time to use any of the resources provided to him in this challenge. Additionally, there is very little indication that he recognized the pattern of code in the solution that could be noticed and used to simplify the program.



Initial Code

Step 1

Step 2 (attempt 4)

Figure 24. Steps of Low Performing Free Choice Student Challenge 1 Task Completion

In contrast, the higher performing free choice student was able to identify that there were several errors and how they aligned to the grid the bot needed to navigate through. Initially, Laney was confused when the toggle switch was not available to use, but quickly adapted and clicked into the Sim Lab to practice her adjustments. Additionally, after aligning the errors with the grid and identifying the best commands to complete the task, she referenced the glossary to determine the correct syntax for the program she was writing (see Figure 25).

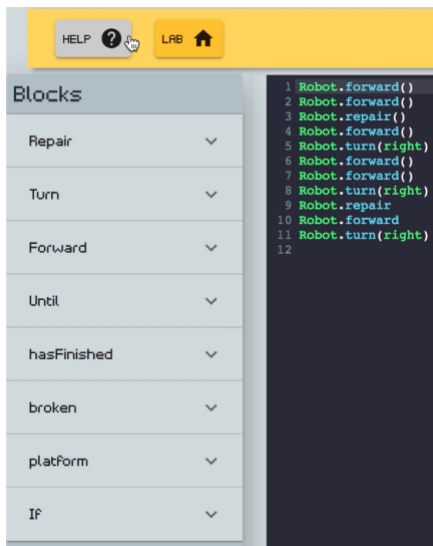


Figure 25. Glossary Selection of High Performing Free Choice Student Challenge 1 Task

Figure 26 illustrates how Laney wrote her program while referencing the provided code and the glossary tool to accurately type the syntax for the *Until* loop command and the two

nested conditionals she included. This demonstrates a deeper level of understanding of each concept when compared to Justin’s code, which was limited to basic sequencing of commands to complete the task. Though he did not take as much time on the task as Laney (Justin’s time = five minutes, 32 seconds, Laney’s time = 9 minutes, 47 seconds), she was able to discern the features of the program she recognized could be simplified into a few concise commands.



Figure 26. Steps of High Performing Free Choice Student Challenge 1 Task Completion

For the second challenge, students were given a block only task with a grid they had not yet seen in regular game-play (see Figure 23). Justin was able to complete the task, however, expressed difficulty in not being able to use what they knew would make the task faster to complete. In contrast to his attempts in the text only task, Justin understood how the code would perform, however the conditional was a problem to implement. For example, at one point, he placed only one conditional in the program asking the bot to check for a platform to the right and if it was present then turn left and pick up the microchip. Therefore, when the bot reached first turn, both conditions were met and carried out, however this wasn't true at the second turn and the bot left two microchips behind. Ultimately Justin decided that he would remove the conditional and submit a program with only repeat commands to complete this task (see Figure 27).

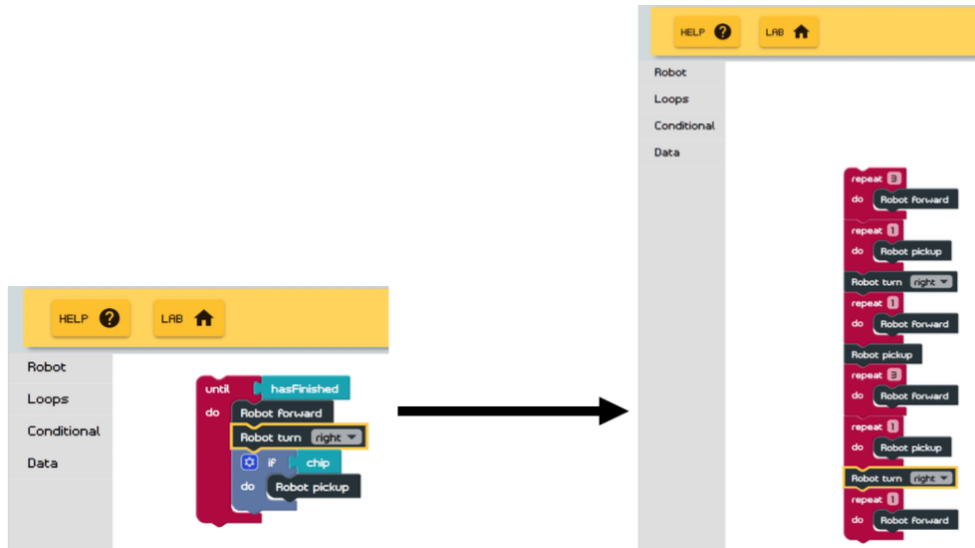


Figure 27. Low Performer First and Last Attempt for Challenge 2

Low Performing Free Choice Excerpt: Usage Example

Justin: *[Tries to Toggle]*

Researcher: “That button isn’t going to work in this task either.”

Justin: *[laughs]* Sorry, it’s a habit.”

Researcher: “What are you trying to do?”

Justin: “I wanted to type the other commands so I can paste copies faster?” (*forward, turn right*)

Researcher: “You can do that with the blocks though. Why do you need to switch?”

Justin: “It takes too long to drag blocks each time. It’s annoying. I can copy the blocks?”

Researcher: “Yes, just drag to select the blocks you want to duplicate. But, can you tell me why you don’t need to type out these blocks?” (*points to repeat and if*)

Justin: “I never get those right, if it’s in block that’s easier.”

In this example, Justin is illustrating a workflow pattern that is reliant on the toggle switch. Rather than taking the time to drag the blocks over, which he finds tedious, he attempts to use the toggle switch to type then copy and paste his code to complete the program. For Justin, the toggle switch is a tool for efficient writing, not necessarily a reference for how to correctly program his bot or as an indication that the program he is writing is correct. His confidence in his ability to complete the task was reliant on the use of a feature that he was could not implement in this solution and though he initially found it difficult, the use of the toggle switch would not necessarily have benefited him in this challenge. Based on his final submission, he seemed confused with how to nest the two conditionals needed for the solution within the repeat block.

High Performing Free Choice Excerpt: Usage Example

Researcher: “I notice you keep clicking the switch button. What are you trying to do?”

Laney: “It’s not working, I was trying to switch it back so I can just copy and paste forward three times then turn right then paste the three forwards again until it reaches the door.”

Researcher: “Do you know how to use these blocks?”

Laney: “I think so. I was going to put the forward blocks all together but that would be too big for the screen. I put in the repeat but it’s not going to work because there are two turns.”

Laney: [*Pulls up Glossary Index three times, then places Until block*]

Researcher: “You put the *Until* block over the rest of your code. Why?”

Laney: “I think I have to switch them. The help button said it should keep going until it finishes. That would work better because there isn’t a chip on each platform. And I already know how to get the bot to turn and pick up, we (*Laney and her pair*) did it a couple times in the class.”

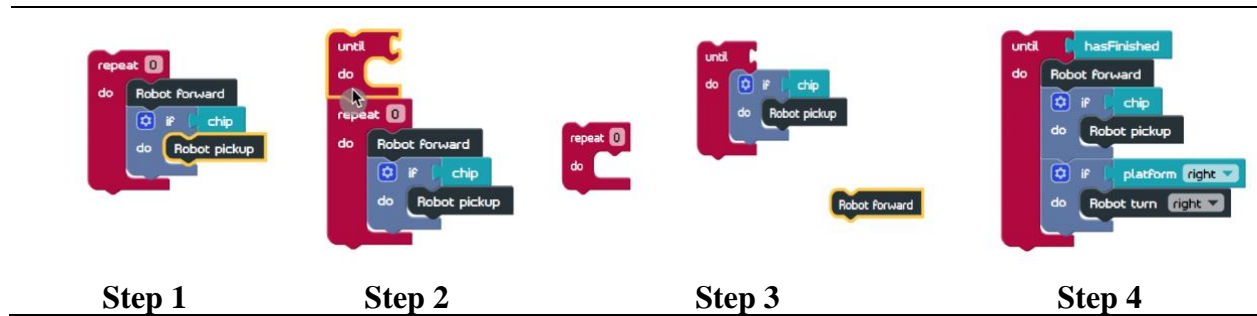


Figure 28. Steps of High Performing Free Choice Student Challenge 2 Task Completion

The example work flow above portrays the contrast between these two levels of performance. Laney is model of how a student with an established understanding of the basic and more complex CS concepts is able to decompose a problem and utilize her resources to applying that knowledge in a new task. Initially, she addresses the problem similarly to how Justin submitted a final solution, to program commands within multiple *repeat* loops. However, in starting that code she quickly realizes there is a recognizable pattern and understands the new command a block she can learn to implement quickly. Additionally, it is important to note that she indicates she is already familiar with the concept of conditionals and has an established knowledge of integrating them within a loop. Extend that knowledge to an *Until* loop was not a far transfer.

High Performing Free Choice Excerpt: Dealing with Challenges

Researcher: "What did you think of that problem?"

Laney: “It was kind of hard. I didn't know the robot could jump. And I'm not good at remembering how to write those blocks right.”

Researcher: "When you didn't know how to write or use a command you used the glossary. Was that helpful?"

Laney: "Yes, if I didn't know what to do, I used the glossary because it was easy to look up how to use the command."

Researcher: "How does that make it easier?"

Laney: "Usually I can type and then switch to check if I wrote my code right. But that problem we just did was different because it was new blocks. I had to open the glossary to make sure I knew how to use them."

Researcher: "When did you use glossary and help when you were playing?"

Laney: "Mostly when I had to type out the code. My partner was the one that knew how to write then better, so when she saw me struggling, she would remind me about the details. Like I always forget that the repeat has the parentheses with the twists (*brackets*)."

Low Performing Free Choice Excerpt: Dealing with Challenges

Researcher:

Justin: "A little bit. The pictures of code were easy to use, sometimes I didn't understand the explanations. Sometimes if I don't know how the code will look, I just switch the blocks on and move the block and switch back."

Researcher: "How does that make it easier?"

Justin: “I don’t know...sometimes I know what I need to do, but I don’t always know how to type the commands. But I *do* know how to put them together like a puzzle. So, I’ll just switch it and see what it’s supposed to look like after I put make it with the blocks. But I couldn’t do that this time, the glossary was okay, but I wanted to ask my partner because she probably knew how to do it better. I know it was too long in that last one (*Challenge 2*).”

Researcher: “When did you use glossary and help when you were playing?”

Justin: “When I needed to look up the new stuff, we didn’t have all the same commands in this problem and since I couldn’t switch, I had to use the help and the glossary. The help button just asked me questions at first then told me to try the glossary after my third try.”

A key difference between these two students was the frequency in which they relied on either the researcher or the resources provided in the console to address their questions. When Justin struggled, he typically asked the researcher how something worked or why something went wrong. Laney referenced the index and practiced the new program she was writing in the Sim lab often without much dialogue.

CT Behaviors, Outcomes and Game Behaviors

Correlational analyses were used to examine the relationships between high and low performing free choice students and their scores on all outcome measures, as well as their observed CT behaviors (see Table 10). Results indicated a positive relationship between performance condition and iterative behaviors, $r(37) = .627, p < .01$; pattern recognition behaviors, $r(37) = .579, p < .01$; and decomposition behaviors, $r(37) = .540, p < .01$. This suggests

that students in the free choice condition who performed better on the posttest exhibited more observable CT behaviors during the challenge tasks than low performing students.

There was no significant relationship between any CT behaviors and the transfer task. Although, the challenge task is positively correlated with observable iterative behaviors $r(37) = .389, p < .05$, and with performance on the learning posttest, $r(37) = .602, p < .01$. It's possible that the more students iterated the better their understanding of how a command would function in practice improved and that applied knowledge was beneficial to their performance on both the overall challenge task and learning posttest.

Additionally, a correlation analysis was used to examine the relationships among observable CT behaviors and game behavior counts pulled from log data (see Table 11). Results indicated an inverse relationship between the amount of toggle switches and the glossary usage, $r(37) = -.362, p < .05$. This is a pattern seen illustrated in the usage examples of Laney and Justin. However, there were no significant relationship differences in seen between posttest performance and any of the game behaviors; hints, glossary, attempt count, sim lab, and toggle switching. Regardless of this finding, further investigation is needed to determine what behavioral differences may have contributed to more successfully learning outcome scores.

Discussion

This investigation into the free choice condition to compare the highest and lowest performing students in the sample shows that students who use the toggle button in conjunction with the resources available to them in the game will be more successful. Generally, in Study 2, students programming in a more structured environment while effectively using the educational reference tools provided, resulted in higher performance on most learning outcome measures. However, within the specific comparison of the free choice condition there was no significant

relationship between their general game behaviors and their observable CT behaviors during the challenge tasks.

However, when using the resources, the high performing student, Laney, was able to detect the function of more complex commands and implement them into her code. For example, the high performing student utilized the tools in the console more often than the lower performing student. She used the glossary as a reference tool and the Sim Lab as a practice space for how to incorporate new coding concepts into her program. In both tasks, the lower performing student stayed within his comfort level and only referenced the glossary for syntactical support in completing basic command functions.

Limitations

While this comparative study provides a specific depiction of how different utility of the toggle switch and resource tools may affect conceptual understanding and skill-based practice, it is unable to make any generalizations regarding instructional implications. Future work should examine the effects of the toggle capability alone within when introducing coding education in order to generate a most robust understanding of how best to design a digital environment where students can thrive using this less structure methodology.

Another limitation of this study was the use of the developed CT behaviors index (see Appendix D and E). The index used to code for CT behaviors observed during the video data of the challenge tasks is new and requires a broader scope of research to more acutely define each practice. Additionally, there was disagreement among raters in regard to the subjective categorization of behaviors for decomposition. Future research is needed to establish a more descriptive CT index that involves generalizing observable behaviors and developing absolute categories of each practice within the rubric.

Lastly, while the quantitative component of this study accounted for the pairs effect on all outcome measures, that was not something that could be included in the qualitative component as students were interviewed individually. However, both students mentioned their pair at some point in the interview, whether to say they would be helpful or how they relied on them for assistance for specific skills. This is something that should be considered and included in any future work related to collaborative learning, more specifically, paired programming.

Conclusion

This comparative analysis study focuses on developing a basic level of understanding as to the knowledge and practices of high and low performing free choice students. The quantitative component contributes to the assumption that within this condition there are a subset of student who performed successfully and a subset with less than satisfactory scores. While both high and low performing students made use of the toggle feature, there were clear differences in how it was utilized and there are some effective methods for its usage in this game that led to a deep and establish knowledge of basic CS concepts. A high performing student was able to extend her conceptual knowledge more flexibly when met with a more restrictive and challenging new task. Whereas, a low performing student relied on the very basic understanding on simple sequencing to complete each task, using the resource glossary and help tool for syntactical reference rather than to learn a new concept. Generally, there are two overarching behaviors, a high performing student would use the toggle to check her work while the low performing student explained he typically used it to make the code process itself faster.

These behaviors helped each student complete the task at varying success rates. It was clear through the exhibited work flow that high performing students who used the toggle feature as another learning tool, rather than a short-cut, knew how to leverage the console and resources.

I expect that there is a larger split of behavioral practice in this condition and that simply using the resource tools is not an effective indicator of how well a student understands the material or how apt they are at discovering the new information themselves. The missing qualitative component to this study is the effect of pairs on their perspective. Interviews were conducted individually, but this work already brings up large implications for the 6-12 coding education and computational thinking fields. However, more work focused on discerning the implications for design and content structure for this particular type of learning may be worth examining as these types of games continue to be developed.

Table 10.

Performance Condition, Computational Thinking Behaviors and Outcome Measures

Variables	1	2	3	4	5	6	7	8
1. Performance Condition	—							
2. Iteration	.627**	—						
3. Pattern Recognition	.579**	.388*	—					
4. Decomposition	.540**	.310	.709**	—				
5. Algorithmic Thinking	.293	.174	.026	.046	—			
6. Posttest	.879**	.706**	.470**	.556**	.323	—		
7. Transfer Task	.194	.110	.218	.103	.035	-.059	—	
8. Challenge Task	.425**	.389*	.004	.127	.241	.602**	-.231	—

Performance condition: 1 = *Low*, 2 = *High*.* $p < .05$. ** $p < .01$

Table 11.

Performance Condition, Computational Thinking Behaviors and Game Behaviors

Variables	1	2	3	4	5	6	7	8	9	10
1. Performance Condition	—									
2. Iteration	.627**	—								
3. Pattern Recognition	.579**	.388*	—							
4. Decomposition	.540**	.310	.709**	—						
5. Algorithmic Thinking	.293	.174	.026	.046	—					
6. Attempts	-.076	.236	-.145	-.112	.117	—				
7. Hints	.019	.320	.123	.159	-.074	.118	—			
8. Glossary	-.076	.009	.034	.053	-.129	-.082	.525**	—		
9. Sim Lab	-.227	.140	-.067	-.075	-.064	.775**	.090	-.151	—	
10. Toggle Switches	.180	.061	.199	.063	.293	.014	-.149	-.362*	-.008	—

Performance condition: 1 = *Low*, 2 = *High*.* $p < .05$, ** $p < .01$

CHAPTER 6: CONCLUSION

Overall, this work suggests that there is a benefit to learning how to code using a dual programming modality that may be driving the development of concrete computer science concepts and foundational complex thinking skills. These findings have several implications for how to support learning in this context for middle and high school aged students, specifically through the use of supplemental educational games.

First, Study 1, presented in Chapter 3 discusses the overview of the development background for the design the application used in the work, Microcity Academy. The initial design of the application was derived from the questions addressed in the study discussed in Chapter 3 where the focus was the relationship between programming modality and its effects on how well students learn and understand basic programming concepts. This was important to discern as previous work investigating modality suggested that there is was a need to explore blended or hybrid programming environments to assist in the transfer from blocks-based to text-based programming languages (Tabet, Gedway, Alshikhabobakr, Razak, 2016; Weintrop &Wilensky, 2015; Weintrop &Wilensky 2018a; Weintrop &Wilensky 2018b).

The findings of this pilot study suggest that there is a difference in how students perform on the learning measure based on the programming modality even though the outcome that students in the free choice condition outperformed those in guided was not expected. However, the rigidity of the guided condition and the flexibility of the free choice condition in conjunction with the lack of scaffolds built into the application may have contributed to the guided condition's lower performance. Additionally, there was a significance difference in performance for students in the game condition. This was expected and supported the work in Study 2 focused on modality solely within the game application.

The findings of Study 2, presented in Chapter 4, suggest that programming modality matters and has a significant impact on how novice programmers learn, develop CT practices and solve new challenging coding tasks. Students who were guided through the tasks and transitioned from blocks to text-based programming outperformed students who were given free choice of their modality on learning, transfer and challenge measures. There are several implications from these outcomes. It is possible that due to the guided transition the students in the guided condition were guaranteed a more balanced learning experience than students who may have chosen to remain in one modality throughout the course of the game. This means that there are students who would not necessarily do well on the learning post-test items that were in a modality they were not familiar with since they were not actively practicing in the application.

Additionally, it is important to note that the restriction of the guided condition required that the students rely heavily on the educational resources that were in place for them to reference. The analyses in Chapter 4 show that there were differences in the average amount of resources being used between the two conditions. Therefore, future work should include a more in-depth investigation into the usage differences between modality and effects of the use of these resources on learning outcomes. This dissertation attempted to start this work into the free choice condition to compare the highest and lowest performing students. Findings show that students who use the toggle button in conjunction with the resources available to them in the game will be more successful.

This is modeled by the high performing free choice student, Laney, who utilized the tools in the console more often than the lower performing free choice student, Justin. Laney used the glossary as a reference and the Sim Lab as a practice space for how to incorporate new coding concepts into her program. In both tasks, the lower performing student stayed within his comfort

level and only referenced the glossary for syntactical support in completing basic command functions. When discussing their work flow, both high and low performing free choice students made use of the toggle feature, there were clear differences in how it was utilized and there are some effective methods for its usage in this game that led to a deep and establish knowledge of basic CS concepts. Generally, there are two overarching behaviors, a high performing student would use the toggle to check her work while the low performing student explained he typically used it to make the code process itself faster. I expect that there is a larger behavioral practice split to explore in this condition along with the several other factors, such as the pair effect, that may be more indicative of student conceptual knowledge and how apt they are at discovering the new information. More research is need on discerning the implications for design and content structure for free choice learning as these types of games continue to be developed.

Lastly, a potential implication of this work relates to the age of the students in Study 2. Although, age had no effect on the performance outcomes and while there is little research regarding what level students across the 6-12 age range should be performing at in a supplemental coding game, the performance typically varies. This may be an indication that there is no advantage to being older and learning these introductory computer science concepts when prior experience across all ages is the same. This drives the question of how to design and implement instruction in supplemental educational tools for different age groups with similar knowledge background so that there is a level of advancement for the older students. The outcome of this work would suggest that early exposure is necessary to gaining these skills. More research on the effect of age on computer science introductory learning and how well students can apply that new knowledge across contexts is needed to determine whether this is a

result that may better inform parents, game developers and educators when designing new games for all age groups.

There is a growing investment in exposing young children to computer science concepts, skills and practices to fostering computational thinking skills (Grover & Pea, 2013). Programming modality is a small component of that learning and is hardly studied in computer science research. This dissertation contributed to the few studies focused on discerning differences in how students learned to code using distinct programming modalities and ensure that there was a true comparison within the same hybrid application. This work suggests that students can learn basic computer science concepts and programming skills in a simple block-based programming modality and be able to apply those concepts successfully in a text-based programming modality without the explicit instruction of a full course. Supplemental learning for K-8 is how this field will gain student interest, not just in programming, but in the skills, they develop and in these types of games.

As game developers, educators continue to collaborate in this space, it is important that researchers provide a framework for implementation of novice coding curriculum and instructional resources in a supplemental educational game design. This work can now contribute to addressing the gap between blocks-based learning and text-based programming. Students who are exposed to new concepts in blocks-based programming languages and master those concepts can apply that same knowledge when confronted with a similar, more challenging task, but in a text-based programming language without explicit instruction in a game. While these constructs still need further study, this work largely provides a new point of view when considering introductory programming interventions and development of novice computer science curricula in supplemental game-based environments.

REFERENCES

- Alaswad, Z., & Nadolny, L. (2015). Designing for game-based learning: The effective integration of technology to support learning. *Journal of Educational Technology Systems, 43*(4), 389-402.
- Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. (2015). From Scratch to “real” programming. *ACM Transactions on Computing Education, 14*(4).
- Atmatzidou, S., & Demetriadis, S. (2016). Advancing students’ computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*, 661-670.
- Bandura, A. (1986). *Social Foundations for of thought and action*. Prentice Hall, Englewood Cliffs, NJ.
- Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the ACM, 60*(6), 72-80.
- Bau, D. (2015). Droplet, a blocks-based editor for text code. *Journal of Computing Sciences in Colleges, 30*(6), 138-144.
- Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145-157.
- Bevan, B., Gutwill, J. P., Petrich, M., & Wilkinson, K. (2015). Learning Through STEM-Rich Tinkering: Findings From a Jointly Negotiated Research Project Taken Up in Practice. *Science Education, 99*(1), 98-120.

- Black, J.B., Segal, A., Vitale, J. & Fadjo, C. (2012). Embodied cognition and learning environment design. In D. Jonassen and S. Lamb (Eds.) *Theoretical Foundations of Student-centered Learning Environemtns*. New York: Routledge.
- Brandt, D. S. (1997). Constructivism: Teaching for understanding of the Internet. *Communications of the ACM*, 40(10), 112-117.
- Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (Vol. 1, p. 25).
- Chambers, B., Cheung, A. C., Madden, N. A., Slavin, R. E., & Gifford, R. (2006). Achievement effects of embedded multimedia in a success for all reading program. *Journal of Educational Psychology*, 98(1), 232.
- Chan, C., Burtis, J., & Bereiter, C. (1997). Knowledge building as a mediator of conflict in conceptual change. *Cognition and instruction*, 15(1), 1-40.
- Chetty, J., & van der Westhuizen, D. (2014, June). Toward a Pedagogy Centering on Computer Programming for Learners in South Africa: An Educational Design Research Approach. In *EdMedia: World Conference on Educational Media and Technology*(pp. 732-740). Association for the Advancement of Computing in Education (AACE).
- Chi, M.T.H. & Wylie, R. (2014). The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational Psychologist*, 49(4), 219-243.
- Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research*, 1(3), 253-274.

- Devolder, A., van Braak, J., & Tondeur, J. (2012). Supporting self-regulated learning in computer-based learning environments: systematic review of effects of scaffolding in the domain of science education. *Journal of Computer Assisted Learning*, 28(6), 557-573.
- Douadi, B., Tahar, B., & Hamid, S. (2012). Smart edutainment game for algorithmic thinking. *Procedia-Social and Behavioral Sciences*, 31, 454-458.
- Ertmer, P. A., & Ottenbreit-Leftwich, A. (2013). Removing obstacles to the pedagogical changes required by Jonassen's vision of authentic technology-enabled learning. *Computers & Education*, 64, 175-182.
- Grover, S. (2014). *Foundations for advancing computational thinking: Balanced designs for deeper learning in an online computer science course for middle school students* (Doctoral dissertation, Stanford University).
- Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267-272). ACM.
- Grover, S. & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38-43.
- Hahn, J.H., Mentz, E., & Meyer, L. (2009). Assessment strategies for pair programming. *Journal of Information Technology Education*, 8, 273-284.

- Hannafin, M., Hannafin, K., & Gabbitas, B. (2009). Re-examining cognition during student-centered, Web-based learning. *Educational Technology Research and Development*, 57(6), 767-785.
- Hannafin, M. J., Hannafin, K. M., Land, S. M., & Oliver, K. (1997). Grounded practice and the design of constructivist learning environments. *Educational technology research and development*, 45(3), 101-117.
- Hansen, A. K., Iveland, A., Carlin, C., Harlow, D. B., & Franklin, D. (2016, June). User-Centered Design in Block-Based Programming: Developmental & Pedagogical Considerations for Children. In *Proceedings of the The 15th International Conference on Interaction Design and Children* (pp. 147-156). ACM.
- Henderson, P. B., Cortina, T. J., Hazzan, O., and Wing, J. M. (2007) Computational thinking. In *Proceedings of the 38th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*, 195–196. New York, NY: ACM Press.
- Honebein, P. (1996). Seven goals for the design of constructivist learning environments. In B. Wilson (Ed.), *Constructivist learning environments* (pp.17-24). Englewood Cliffs, NJ: Educational Technology Publications.
- Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). Computational Thinking Patterns. *Online Submission*.
- Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, 82, 263-279.

- Johnson, G. M. (2005). Instructionism and Constructivism: Reconciling Two Very Good Ideas. *Online Submission*.
- Jonassen, D. H. (1991). Objectivism vs. constructivism. *Educational Technology Research and Development*, 39(3), 5-14.
- Kafai, Y. B. (2006). Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and culture*, 1(1), 36-40.
- Kafai, Y.B. & Burke Q. (2013). Computer programming goes back to school. *Kappan*, 61-65.
- Kafai, Y. B., & Burke, Q. (2015). Constructionist gaming: Understanding the benefits of making games for learning. *Educational psychologist*, 50(4), 313-334.
- Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education*, 8(1), 13-24.
- Kiili, K. (2007). Foundation for problem-based gaming. *British journal of educational technology*, 38(3), 394-404.
- Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*, 72, 558-569.
- Lewis, C. M. (2010, March). How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 346-350). ACM.
- Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, 20(4), 191-206.

- Linderoth, J. (2012). Why gamers don't learn more: An ecological approach to games as learning environments. *Journal of Gaming & Virtual Worlds*, 4(1), 45-62.
- Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*, 41(1), 260-264.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4), 16.
- Merrill, M. D. (1991). Constructivism and instructional design. *Educational technology*, 31(5), 45-53.
- Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, 23(4), 1483-1500.
- Moreno, R., Mayer, R. E., Spires, H. A., & Lester, J. C. (2001). The case for social agency in computer-based teaching: Do students learn more deeply when they interact with animated pedagogical agents?. *Cognition and instruction*, 19(2), 177-213.
- Nadolny, L., Alaswad, Z., Culver, D., & Wang, W. (2017). Designing with game-based learning: Game mechanics from middle school to higher education. *Simulation & Gaming*, 48(6), 814-831.
- NGSS Lead States. (2013). Next Generation Science Standards: For states, by states. Washington, DC: The National Academies Press.

- Palincsar, A. S. (1998). Social constructivist perspectives on teaching and learning. *Annual review of psychology*, 49(1), 345-375.
- Papastergiou, M. (2009). Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation, *Computers & Education*, 52(1), pp. 1–12
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.
- Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York: Basic Books.
- Pila, Sarah, Aladé, Fashina, Sheehan, Kelly J., Lauricella, Alexis R. and Wartella, Ellen A. (2019). "Learning to code via tablet applications: An evaluation of Daisy the Dinosaur and Kodable as learning tools for young children." *Computers & Education*, 128, 52-62.
- Powers, K., Ecott, S., Hirshfield, L.M. (2007). Through the looking glass: teaching CS0 with Alice. *SIGCSE Bull.* 39, 1, 213-217.
- Puzziferro, M., & Shelton, K. (2008). A model for developing high-quality online courses: Integrating a systems approach with learning theory. *Journal of Asynchronous Learning Networks*, 12, 119-136.
- Resnick, M. (2012). Reviving Papert's dream. *Educational Technology*, 52(4), 42-46.
- Rideout, V. J., Foehr, U. G., & Roberts, D. F. (2010). *Generation M 2: Media in the Lives of 8-to 18-Year-Olds*. *Henry J. Kaiser Family Foundation*.

- Ritterfeld, U., & Weber, R. (2006). Video games for entertainment and education. *Playing video games: Motives, responses, and consequences*, 399-413.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer science education*, 13(2), 137-172.
- Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. Teoksessa JM Hoc, TRG Green, R. Samurçay, & DJ Gillmore (toim.), *Psychology of programming* (s. 157–174).
- Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on game design*. New Riders.
- Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, 72, 678-691.
- Ross, S. M., & Morrison, G. R. (1989). In search of a happy medium in instructional technology research: issues concerning internal validity, media replications, and learner control. *Educational Technology Research and Development*, 37, 19-24.
- Ross, S.M. & Lowther, D.L. (2009). Effectively using technology in education. *Better Evidence-Based Education*, 2(1), 20-21.
- Ross, S. M., Morrison, G. R., & Lowther, D. L. (2010). Educational technology research past and present: Balancing rigor and relevance to impact school learning. *Contemporary Educational Technology*, 1(1), 17-35.

- Salleh, N., Mendes, E., & Grundy, J. (2011). Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review. *IEEE Transactions on Software Engineering*, 37(4), 509-525.
- Saye, J., & Brush, T. (2001). The use of embedded scaffolds with hypermedia-supported student-centered learning. *Journal of Educational Multimedia and Hypermedia*, 10(4), 333-356.
- Shaffer, D. W. (2007), *How Computer Games Help Children Learn*, New York: Palgrave Macmillan.
- Sharma, P., & Hannafin, M. J. (2007). Scaffolding in technology-enhanced learning environments. *Interactive learning environments*, 15(1), 27-46.
- Slavin, R. E. (1980). Cooperative learning. *Review of educational research*, 50(2), 315-342.
- Slavin, R. E. (2009). *Educational Psychology: Theory into Practice* (Ninth Edition). Upper Saddle River, NJ: Pearson.
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142-158.
- Smith, M. (2016). *Computer science for all*.
- Sung, W., Ahn, J. & Black, J.B. (2017). Introducing computational thinking to young learners: Practicing computational perspectives through embodiment in mathematics education. *Technology, Knowledge and Learning*, 22(3), 443-463.
- Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016, July). From alice to python. Introducing text-based programming in middle schools. In *Proceedings of the 2016 ACM*

- Conference on Innovation and Technology in Computer Science Education* (pp. 124-129). ACM.
- Thomas, L., Ratcliffe, M., & Robertson, A. (2003, February). Code warriors and code-a-phobes: a study in attitude and pair programming. In *ACM SIGCSE Bulletin* (Vol. 35, No. 1, pp. 363-367). ACM.
- Tomayko, J.E. (2002). A comparison of pair programming to inspection for software defect reduction. *Computer Science Education*, 12(3), 213-222.
- Tsai, C. W., & Fan, Y. T. (2013). Research trends in game-based learning research in online learning environments: A review of studies published in SSCI-indexed journals from 2003 to 2012. *British Journal of Educational Technology*, 44(5), E115–E119.
- U.S. Department of Labor, Bureau of Labor Statistics. (2015). *Employment Projections for 2016-2026*. Retrieved from <https://www.bls.gov/emp/tables/emp-by-detailed-occupation.htm>.
- VanDeGrift, T. (2004). Coupling pair programming and writing: Learning about students' perceptions and processes. *Special Interest Group on Computer Science Education*, 2-6.
- Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: A meta-analysis. *Journal of Educational Computing Research*, 34(3), 229-243.
- Vygotsky, L. S., Cole, M., John-Steiner, V., Scribner, S., & Souberman, E. (1978). *Mind in society*. Cambridge, MA: Harvard University Press.
- Vygotsky, L. (1987). Zone of proximal development. *Mind in society: The development of higher psychological processes*, 5291, 157.

- Weintrop, D. & Wilensky, U. (2015). The challenges of studying blocks-based programming environments. In *2015 IEEE Block and Beyond Workshop*. IEEE.
- Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1), 3.
- Weintrop, D., & Wilensky, U. (2018a). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*.
- Weintrop, D., Hansen, A. K., Harlow, D. B., & Franklin, D. (2018b, August). Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 142-150). ACM.
- Weintrop, D. (2016). *Modality matters: Understanding the effects of programming language representation in high school computer science classrooms* (Doctoral dissertation, Northwestern University).
- Werhane, P. H., Hartman, L. P., Moberg, D., Englehardt, E., Pritchard, M., & Parmar, B. (2011). Social constructivism, mental models, and problems of obedience. *Journal of business ethics*, 100(1), 103-118.
- Williams, L., & Upchurch, R. L. (2001, February). In support of student pair-programming. In *ACM SIGCSE Bulletin* (Vol. 33, No. 1, pp. 327-331). ACM.
- Wing, J.M. (2006). Computational thinking, *Communications of the ACM*, 49(3), 33-35.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, 28(3), 17-22.

Wouters, P., Van Nimwegen, C., Van Oostendorp, H., & Van Der Spek, E. D. (2013). A meta-analysis of the cognitive and motivational effects of serious games. *Journal of educational psychology*, 105(2), 249.

Yaroslavski, D. (2014). *Lightbot*. Retrieved from <http://lightbot.com>.

0-8 Common Sense Media. (2017). *The Common Sense Census: Zero to eight: Media Use by Kids Zero to Eight*. San Francisco, CA: Author.

APPENDIX A: STUDY 1 PRE-TEST

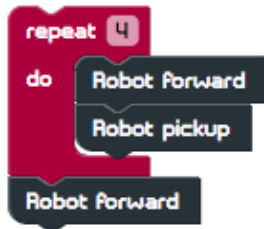
TEACHERS COLLEGE

COLUMBIA UNIVERSITY

Name: _____

Date: _____

Age: _____



What is the above an example of?

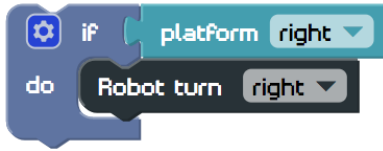
- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

```
1 Repeat(4){
2   Robot.forward
3   if(chip){
4     Robot.pickup()
5   }
6 }
```

What is the above an example of?

- a. Conditional

- b. Debugging
- c. Repeat/Loop
- d. Nesting code



What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

```
1 Repeat (4) {  
2   Robot.forward  
3 }
```

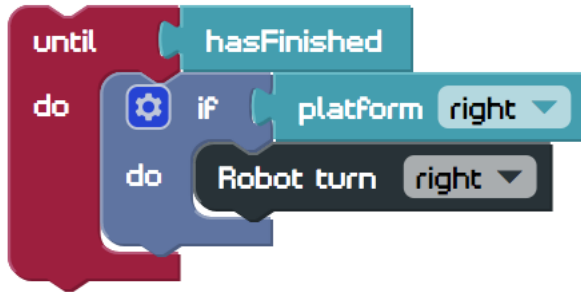
What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

```
1 if (chip) {  
2   Robot.pickup()  
3 }
```

What is the above an example of?

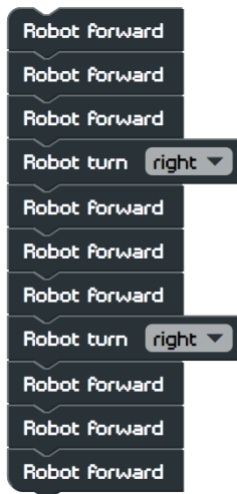
- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code



What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

EXPLAINING CODE (reading and explain code)



Bot Start			
End			

(1) Describe what the program above does and whether it correctly solves the task given above?

(2) Is the most *efficient* way this code could be written?

- a. Yes
- b. No
- c. I don't know

(3) If yes, why is this the best way to write the code for solving the above grid?

(4) If no, what would you do to improve it?

```
1 Until(End){  
2   Robot.forward()  
3   if(platform.right){  
4     Robot.turn(left)  
5   }  
6   if(platform.left){  
7     Robot.turn(left)  
8   }  
9 }
```

Bot Start				
				End

(1) What will be the result of running the above code? Explain your answer.

Use the code given below to answer the following questions.

```
1 Repeat(4){
2   Robot.forward()
3   if(missing){
4     Robot.jump()
5   }
6   if(chip){
7     Robot.pickup()
8   }
9 }
```

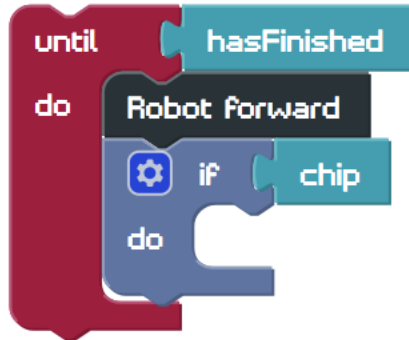
How many times will the Robot run this program?

- a. 20
- b. 2
- c. 7
- d. 4
- e. I don't know

What will happen if the Robot walks to a *missing* platform?

- a) It will fall through it
- b) It will pick up a microchip
- c) It will jump over it
- d) It will repair it
- e) I don't know

Use the code given below to answer the following questions.



How many times will the Robot run this program?

- a) Forever
- b) Until it reaches the end of the task
- c) 5
- d) Whenever the Robot feels like stopping
- e) I don't know

What will happen if the Robot sees a *chip* on the platform?

- a) It will pick up the microchip
- b) It will jump over it
- c) It won't do anything
- d) It will repair it
- e) I don't know

APPENDIX B: STUDY 1 POST-TEST

TEACHERS COLLEGE

COLUMBIA UNIVERSITY

Name: _____

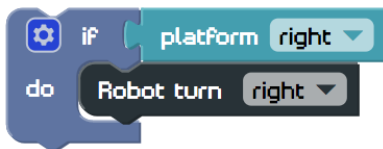
Date: _____

Age: _____

```
1 Repeat(4){
2   Robot.forward
3   if(chip){
4     Robot.pickup()
5   }
6 }
```

What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code



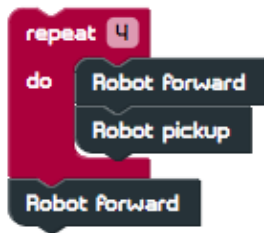
What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code


```
1 if(chip){  
2   Robot.pickup()  
3 }
```

What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code



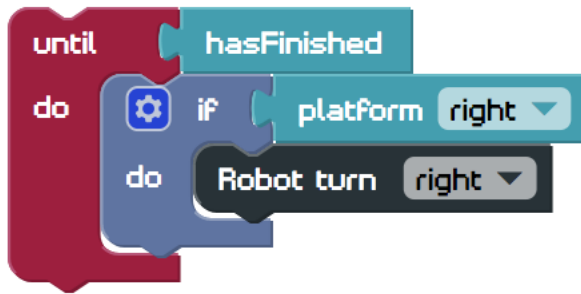
What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

```
1 Repeat(4){  
2   Robot.forward  
3 }
```

What is the above an example of?

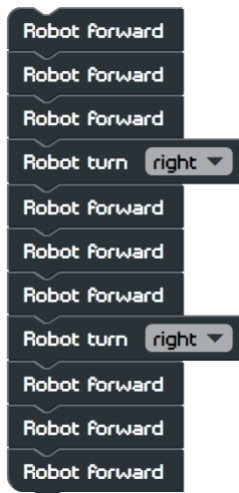
- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code



What is the above an example of?

- a. Conditional
- b. Debugging
- c. Repeat/Loop
- d. Nesting code

EXPLAINING CODE (reading and explain code)



Bot Start			
End			

(1) Describe what the program above does and whether it correctly solves the task given above?

(2) Is the most *efficient* way this code could be written?

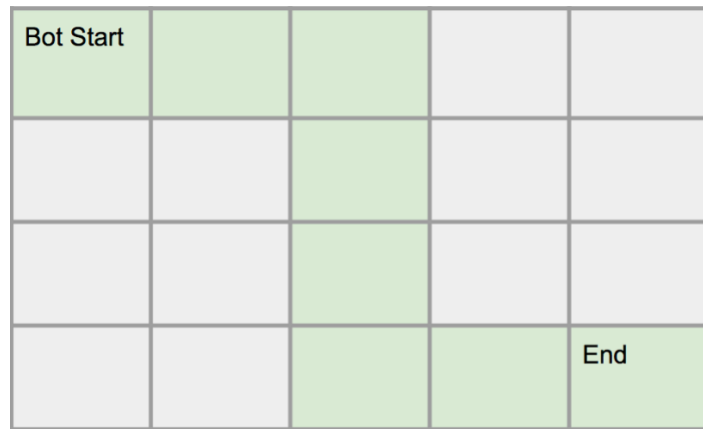
- a. Yes
- b. No

c. I don't know

(3) If yes, why is this the best way to write the code for solving the above grid?

(4) If no, what would you do to improve it?

```
1 Until(End){  
2   Robot.forward()  
3   if(platform.right){  
4     Robot.turn(left)  
5   }  
6   if(platform.left){  
7     Robot.turn(left)  
8   }  
9 }
```



(1) What will be the result of running the above code? Explain your answer.

Use the code given below to answer the following questions.

```
1 Repeat(4){
2   Robot.forward()
3   if(missing){
4     Robot.jump()
5   }
6   if(chip){
7     Robot.pickup()
8   }
9 }
```

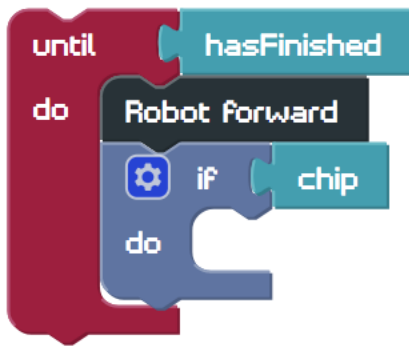
How many times will the Robot run this program?

- a) 20
- b) 2
- c) 7
- d) 4
- e) I don't know

What will happen if the Robot walks to a *missing* platform?

- a) It will fall through it
- b) It will pick up a microchip
- c) It will jump over it
- d) It will repair it
- e) I don't know

Use the code given below to answer the following questions.



How many times will the Robot run this program?

- a) Forever
- b) Until it reaches the end of the task
- c) 5
- d) Whenever the Robot feels like stopping
- e) I don't know

What will happen if the Robot sees a *chip* on the platform?

- a) It will pick up the microchip
- b) It will jump over it
- c) It won't do anything
- d) It will repair it
- e) I don't know

APPENDIX C: STUDY 1 ONLINE PRIOR KNOWLEDGE SURVEY

Q1 Name: _____

Q2 Age

- a) 11
- b) 12
- c) 13
- d) 14
- e) 15
- f) 16
- g) 17
- h) 18
- i) Other ____

Q3 Gender

- a) Male
- b) Female
- c) Other

Q4 Grade

- a) 6th
- b) 7th
- c) 8th
- d) Other ____

Q5 Have you written a computer program before?

- a) Yes
- b) No

(If Q5 is YES)

Q6 What language(s) have you used? List below.

[_____]

Q7

In the future, can you see yourself...

Strongly Disagree	Disagree	Neither	Agree	Strongly Agree
Taking more classes about computers or computer science?				
Becoming a computer programmer or engineer of some sort?				
Becoming a graphic designer or web designer?				
Becoming a computer or technology teacher?				
Becoming a computer game designer?				
Becoming an app developer?				
Becoming a computer scientist?				
Becoming a scientist?				
Becoming a teacher?				
Becoming a doctor or nurse?				
Becoming an artist?				
Becoming a designer?				
Starting a business?				

Q8 Please describe your ideal job for the future:

[_____]

Q9 What are your top 3 favorite subjects in school?

[_____]

[_____]

[_____]

Q10 How MANY TIMES do you use a computer (anywhere) to do each of the following...

Never	Once a month	A few times a Month	Once a Week	A few times a Week	Daily	Several times a day
Play games (on the computer, online or on a game console).						
Participate in multi-user online games.						
Work on your own digital media projects outside of school assignments?						
Conduct research on the Internet for school.						
Collect/view/organize images or music (e.g. put your photos, images or sounds from the Web into folders).						
Write for fun.						
Read or send email.						
Read comics (e.g. Manga).						
Do some artwork.						
Doing homework, checking grades.						
Watching movies and online music videos.						
Take online courses in science/math/other.						
Watch online academic videos and lectures (e.g. Khan Academy).						
Social networking (e.g. Facebook).						
Do computer programming.						

Q11 How MANY TIMES have you EVER CREATED the following using some software on the computer?

0 times	1-2 times	3-5 times	5 + times
A multimedia presentation (e.g. PowerPoint).			
Written computer program (code) using a computer language (e.g. LOGO, Java, Javascript, Python).			
Computer creations using Scratch, Alice or Tynker (block-based programming).			
A website using HTML.			
An app for iPhone or Android.			
A piece of art using a software application (e.g. Photoshop, Illustrator)			
Built a robot or other invention of any kind using electronics and technology.			
A digital movie (e.g. iMovie or MovieMaker).			
An animation (e.g. Flash, Alice, Scratch).			
A computer or video game (e.g. Stagecast, GameStar, Scratch, Kodu).			
Created a piece of music (e.g. GarageBand, FruityLoops).			
A spreadsheet, graph, or chart (e.g. Excel).			

Q12 What is your level of experience with the following computer applications or equipment?

I don't know what this is	I have no experience, but I have heard of it	I've played around with it	I have used it to make something	I'm an expert and can teach someone how to use it
Flash				
Photoshop/Illustrator				
Scratch/Tynker				
Alice				
LOGO				

Swift/ Swift Playground
Java programming
Python programming
Javascript programming
HTML/XML
iPhone SDK/Objective C
GameStar Mechanic
FruityLoops/Audacity/GarageBand
iMovie/MS MovieMaker
Arduino
Microsoft Word/PowerPoint
C or C++ programming

Q13 How often do you use a computer in the following places?

Never	Once a month	A few times a Month	Once a Week	A few times a Week	Daily	Several times a day
At home.						
At school during class.						
At school on your own time.						
At a relative's house.						
In an after-school program/club.						
At a friend's house.						
At the library.						

APPENDIX D: COMPUTATIONAL THINKING BEHAVIOR INDEX

Computational Thinking Skill	Behavior Descriptions
Iteration	Multiple submissions, switching blocks/text, restarting code
Pattern Recognition	Write code, simplify long code with loop or another controller method/block (if/then)
Decomposition	Breaking down the problem, splitting code to different parts of grid
Algorithmic Thinking	Reviewing problem, talking about the grid, merging repetitive code, sequencing steps

APPENDIX E. COMPUTATIONAL THINKING BEHAVIOR INDEX RUBRIC

Code Index Used to Indicate Computational Thinking Behaviors During the Challenge Tasks

CT Behavior	Description	Example
Iteration	Multiple submissions, switching blocks/text, restarting code.	
Pattern Recognition	Write code, simplify long code with loop or another controller method/block (if/then).	
Decomposition	Breaking down the problem, splitting code to different parts of grid.	<p>“There are 3 missing platforms, I’ll have to jump.”</p> <p>“Ah, I forgot I have to turn right. How do I do that with a repeat? I already have the first part, can I put 2 repeats?”</p>
Algorithmic Thinking	Reviewing problem, talking about the grid, merging repetitive code, sequencing steps.	<p>“That path isn’t missing...that’s something different. [tries to have robot walk over it] Oh it died, ok so I have to jump over then turn and pick up the chips, that’s a lot.”</p>

APPENDIX F. SUMMARY OF CT FACETS AND DEFINITIONS
(SHUTE, SUN, & ASBELL-CLARKE, 2017)

Facet	Definition
Decomposition	Dissect a complex problem or system into manageable parts. The divided parts are not random pieces, but functional elements that collectively comprise the whole system or problem.
Abstraction	<p>Extract the essence of a (complex) system. Abstraction has three subcategories:</p> <ul style="list-style-type: none"> a) <i>Data collection and analysis</i>: Collect the most relevant and important information from multiple sources and understand the relationships among multilayered tasks. b) <i>Pattern Recognition</i>: Identify patterns and rules underlying the information structure. c) <i>Modeling</i>: Building models or simulations to represent how a system operates, and/or how a system will function in the future.
Algorithms	<p>Design logical and ordered instructions for rendering a solution to a problem. The instructions can be carried out by a human or computer. There are four subcategories:</p> <ul style="list-style-type: none"> a) <i>Algorithm design</i>: Create a series of ordered steps to solve a problem b) <i>Parallelism</i>: Carry out a certain number of steps at the same time. c) <i>Efficiency</i>: Design the fewest number of steps to solve a problem, removing redundant and unnecessary steps. d) <i>Automation</i>: Automate the execution of the procedure when required to solve similar problems.
Debugging	Detect and identify errors, and then fix the errors, when a solution does not work as it should.
Iteration	Repeat design processes to refine solutions, until the ideal result is achieved.

APPENDIX G: STUDY 2 ATTITUDINAL SURVEY

ONLY AT PRE: Demographic Questions

1. Name
2. Study ID
3. Birthday
4. Grade
5. Gender

ASKED AT PRE/MID/POST: The following questions are asked on a 5-point Likert scale



6. Programming is fun.
7. I will be good at programming.
8. Programming is hard.
9. I know more than my friends about programming.
10. In the future, I would like a job that involves programming.
11. I like programming.
12. My family encourages me to learn to program.
13. I think knowing how to program is important.
14. I like using computers.
15. I can become good at programming.
16. I like the challenge of programming.
17. I think programming will be useful in the future.
18. I cannot learn to program well if the teacher does not explain thing well.
19. Computer Science is all about programming.
20. I plan to continue to learn more about computer science after this activity.
21. I will do well in these programming activities.
22. I am excited about this activity.
23. I think learning to program can help me with other classes.
24. I think learning to program will help me with things outside of school.
25. I think about the programs that control the devices I use in my everyday life.

ONLY AT PRE: Multiple choice questions

26. How much time do you spend on a computer at home each day?
 - a. I don't use a computer
 - b. Less than 1 hour
 - c. Between 1 and 2 hours
 - d. Between 2 and 3 hours
 - e. More than 3 hours
27. What do you do on the computer outside of school?
28. What types of computational devices do you own or use regularly? *Check all that apply:*
 - a. Laptop computer
 - b. Desktop computer
 - c. Tablet (iPad, Surface tablet, etc.)
 - d. Smartphone (iPhone, Google Pixel, Samsung Galaxy, etc.)
 - e. Portable Media Player (iPod, portable movie player, etc.)
 - f. Game console (Xbox, Play Station, Wii, etc.)
29. Have you tried learning how to program on your own before? If yes, using what resources? (Courses, online classes, apps/games)
30. Have you ever used these languages/programming tools? *Check all that apply:*
 - a. Scratch or Snap!
 - b. App Inventor
 - c. Alice
 - d. HTML, CSS or JavaScript
 - e. Java, C++ or C#
 - f. Python, Lisp or Scheme
 - g. Pencil Code
 - h. Tynker
 - i. Code.org
 - j. CodeHS
 - k. Other: _____
31. Do you know any professional programmers? If yes, who?

ONLY AT MID: Open-Ended

32. The thing I like most about Microcity Academy is...
33. The thing I like least about Microcity Academy is...
34. Something about how they feel working with their partner.

The following questions are asked on a 7-point Likert scale

35. Microcity Academy is making me a better programmer
36. I think Microcity Academy was a good use of class time.
37. I will do well in this activity.
38. I am excited about this game.

7-point Likert questions (conceptual ease)

39. I think it is it to learn how to use the game?
40. I have used the Lab Sim to practice coding.
41. I think the resource/help glossary is useful.
42. I use the block/text glossary frequently.
43. I like that I can choose the path to navigate the bot to the exit.
44. I like working with my partner on this activity.
45. This would be harder to learn without my partner.
46. I think I could do this activity and learn how to program well on my own.

7-point Likert questions (conceptual ease) and free response

47. What do for loops and while loops do? How are they used in programs?
48. How easy was it to use loops (for and while) in Microcity Academy?
49. What do if and if/else statements do? How are they used in programs?
50. How easy was it to use if and if/else statements in Microcity Academy?

ONLY AT POST:

7-point Likert questions

51. What I learned in blocks helped me learn the MCA language.
52. Microcity Academy made me a better programmer.
53. I think Microcity Academy was a good use of class time.
54. Microcity Academy helped me learn what real programmers know.
55. I did well in this activity.
56. I am excited about this game.
57. I am more excited about programming now than I was when we started this activity.

APPENDIX H: STUDY 2 PRE/POST-TEST (ENGLISH)

TEACHERS COLLEGE
COLUMBIA UNIVERSITY

Name: _____

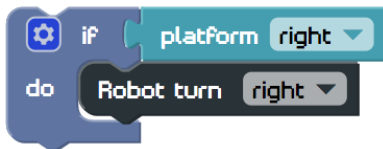
Date: _____

Age: _____

```
1 Repeat(4){
2   Robot.forward
3   if(chip){
4     Robot.pickup()
5   }
6 }
```

What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code



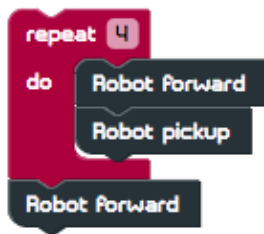
What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code

```
1 if(chip){  
2   Robot.pickup()  
3 }
```

What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code



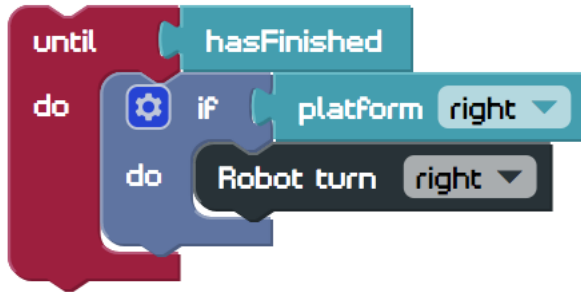
What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code

```
1 Repeat(4){  
2   Robot.forward  
3 }
```

What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code

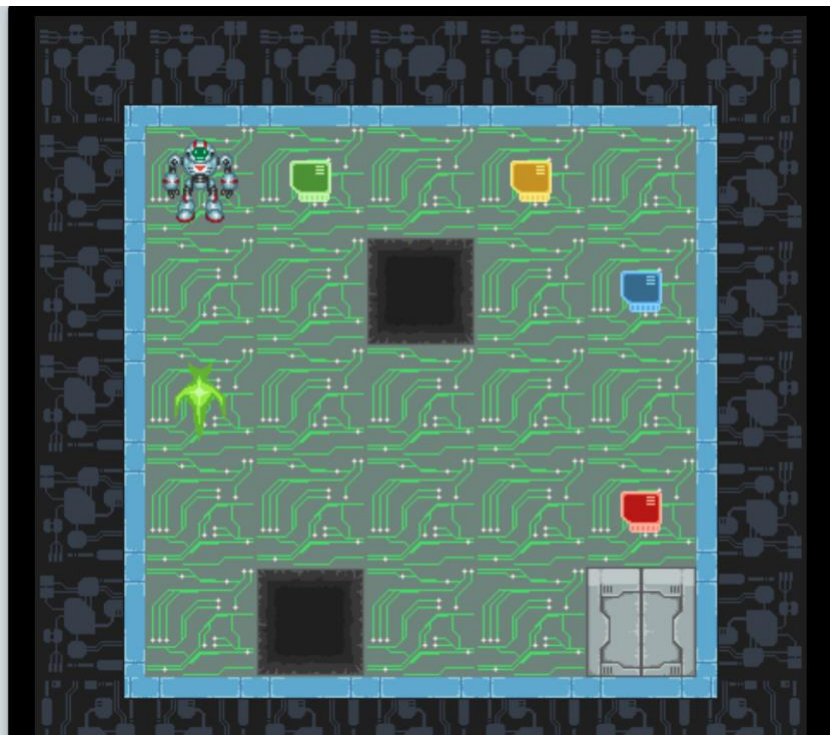


What is the above an example of?

- a) Conditional
- b) Debugging
- c) Repeat/Loop
- d) Nesting code

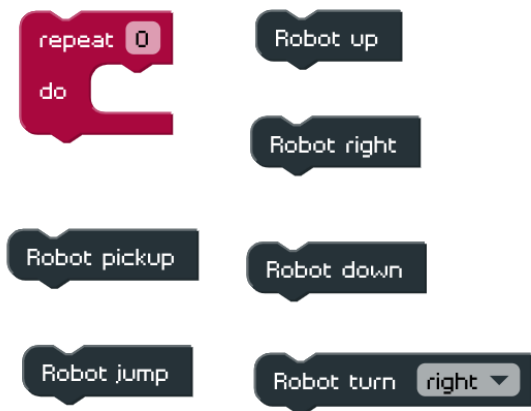
EXPLAINING CODE (reading and explain code)

- Robot right
- Robot pickup
- Robot right
- Robot right
- Robot right
- Robot down
- Robot pickup
- Robot down
- Robot down
- Robot down



- 1) Look at the code provided above. Describe what the program does and whether it is the best way to solve the task given.

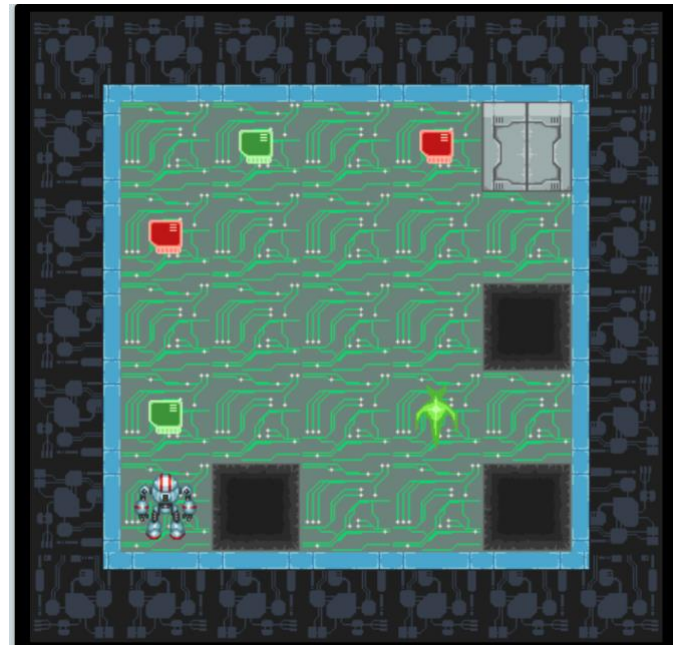
- 2) Was this the most *efficient* way this code could be written?
- a. Yes
 - b. No
 - c. I don't know



3) If yes, why is this the best way to write the code for solving the above grid?

4) If no, what would you do to improve it? Use the code bank to help explain.

```
1 until(hasFinished) {  
2   Robot.forward()  
3   if (platform.right) {  
4     Robot.turn(left)  
5   }  
6 }  
7  
8
```



1) What will be the result of running the above code? Explain your answer.

Use the code given below to answer the following questions.

```
1 Repeat(4){
2   Robot.forward()
3   if(missing){
4     Robot.jump()
5   }
6   if(chip){
7     Robot.pickup()
8   }
9 }
```

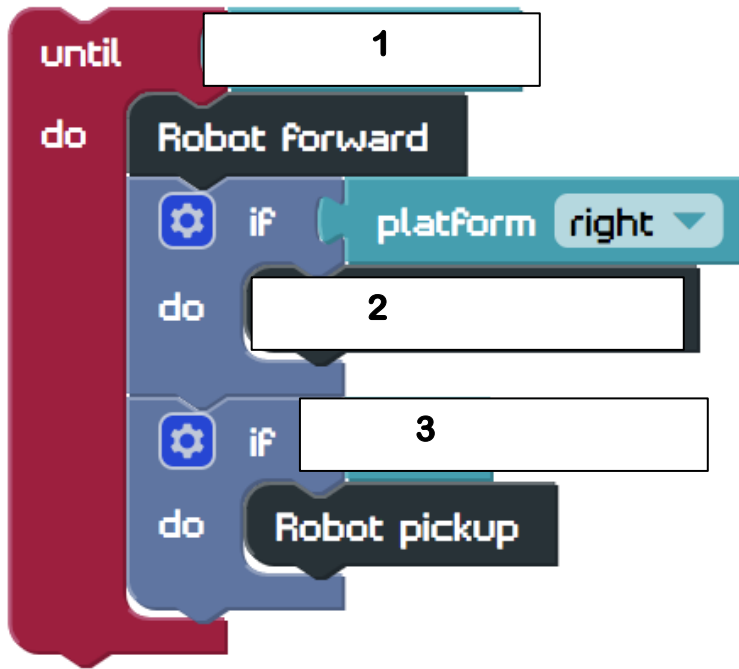
How many times will the Robot run this program?

- a) 20
- b) 2
- c) 7
- d) 4
- e) I don't know




What will happen if the Robot walks to a *missing* platform?

- a) It will fall through it
- b) It will pick up a microchip
- c) It will jump over it
- d) It will repair it
- e) I don't know

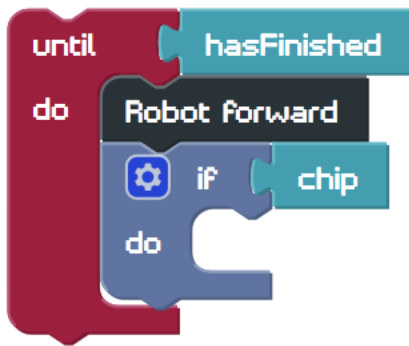
Use the code presented below to answer the following questions.



Below are the three missing instructions. In the spaces provided next to each, write the number for the missing command to match the instruction.

	<input type="text"/>
	<input type="text"/>
	<input type="text"/>

Use the code given below to answer the following questions.



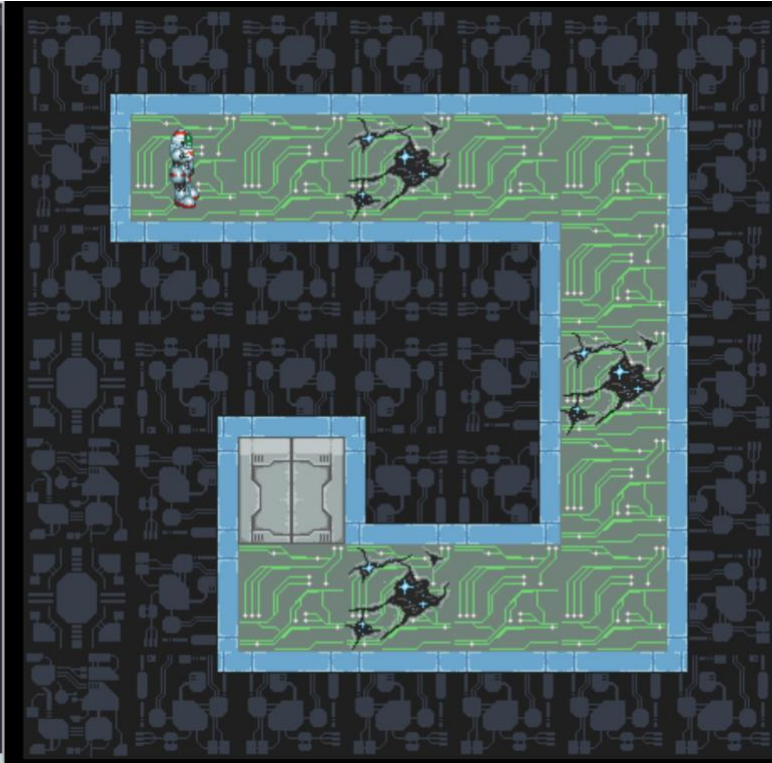
How many times will the Robot run this program?

- a) Forever
- b) Until it reaches the end of the task
- c) 5
- d) Whenever the Robot feels like stopping
- e) I don't know

What will happen if the Robot sees a *chip* on the platform?

- a) It will pick up the microchip
- b) It will jump over it
- c) It won't do anything
- d) It will repair it
- e) I don't know


```
1 Robot.forward()  
2 Robot.forward()  
3 Robot.repair()  
4 Robot.forward()  
5 Robot.turn(right)  
6 Robot.forward()  
7 Robot.forward()  
8 Robot.turn(right)  
9 Robot.repair  
10 Robot.forward  
11 Robot.turn(right)
```



(1) This code does not work. Can you figure out why?

(2) Was this the most efficient way this code could be written?

- a. Yes
- b. No
- c. I don't know

(3) If you answered yes, explain why.

(4) If you answered no, what would you do to improve it? Use the code provided below as reference for your response.

Commands

```
platform()
```

```
Robot.turn()
```

```
if(){} 
```

```
Robot.repair
```

```
until(){} 
```

```
Robot.forward()
```

Data

```
hasFinished
```

```
broken
```

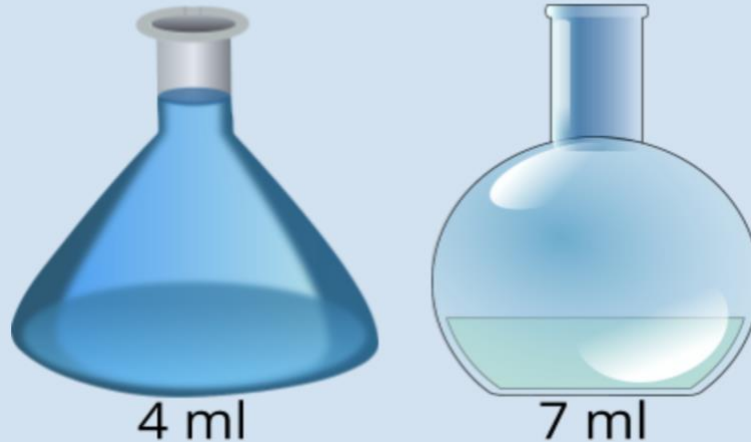
```
missing
```

APPENDIX I: STUDY 2 RUBRIC EXAMPLE RESPONSE

Written Response		
Correct	Partial	Incorrect
Correct answer and correct explanation Example: The code has the Bot moving forward until it has reached the door, but it will not work. It doesn't have the correct turn direction set.	Correct answer or correct explanation Example: The Bot will move forward until it has reached the door. <i>OR</i> Bot will not reach the door at the end.	Wrong answer and wrong explanation Example: The Bot will reach the end because the code has the Bot moving forward until it gets there.

APPENDIX J: STUDY 2 TRANSFER TASKS

You've run into a predicament. You want to do a science experiment. Given two containers, one that can hold 4 millilitres and one that can hold 7 millilitres, how can you get exactly one of these containers to hold exactly 5 millilitres?



These containers are blacked out and oddly shaped, so you can't tell by weight or watermark how much is in the container - you can only accurately tell if it is empty or if it is full. You can't look at the container to ascertain any other quantities.

“What do you know about the final desired volume?” “How can you express that value other than ‘5’?” “How do you think you can break this big problem into smaller problems?”

Describe the solution in specific steps and record the amount of water in each container (A and B) contains at each step.

You can do this by first noting that at any point there are only three actions you can do: completely fill a container, completely empty a container, or move the contents of one container to another (for this, you can use the shorthand A -> B to mean pouring the contents of A into B).

So an instruction to fill up the first container (A), pour it into the second (B) and then empty container B would look like:

Instruction	Quantity in A (max 4)	Quantity in B (max 7)
START	0	0
Fill A	4	0
A -> B	0	4
Empty B	0	0

Create your own table to describe the solution. An example solution might look like:

Instruction	Quantity in A (max 4)	Quantity in B (max 7)
START	0	0

Q1: How do you know how to start?

Q2: How do you know if you should use the method that pours A into B or the method that pours B into A?

Read the following word problems and write an algorithm for the solution or to indicate the pattern.

For example:

There are 90 people in line at a theme park ride.

Every 5 minutes, 40 people get on the ride and 63 join the line.

Estimate how long it would take for 600 people to be in line."

```
starting_peeps = 90
```

```
time = 0
```

```
new_peeps = 63
```

```
leaving_peeps = 40
```

```
while starting_peeps <= 600:
```

```
    starting_peeps = starting_peeps + new_peeps - leaving_peeps
```

```
    time = time + 5
```

- a) Sam has a jar with 5 cups of fresh lemonade.
Jack has some glasses which hold 1.5 cups each of liquid.
How many glasses of lemonade can Jack serve of Sam's lemonade?

- b) Charisse is buying two different types of cereals from the bulk bins at the store. Granola costs \$2.29 per pound, and muesli costs \$3.75 per pound. She has \$7.00. Use x as the amount of granola and y as the amount of muesli. How many pounds of granola can she buy if she buys 1.5 pounds of muesli?

APPENDIX K: STUDY 2 INTERVIEW PROTOCOL – CHALLENGE PROBLEMS

This interview is modeled the same way as the pair programming procedure except that the student is being asked to start as the navigator and tell the interviewer what to do as they go about solving the problem.

Tell me what you know about programming?
Do you think you could use what you have learned to solve a new task?

Okay, we are going to work together on a few problems. I'll start as the driver and you as the navigator. If at any point you want to switch roles, let me know and I'll do the same.

[Challenge Level 1 – Blocks Only]

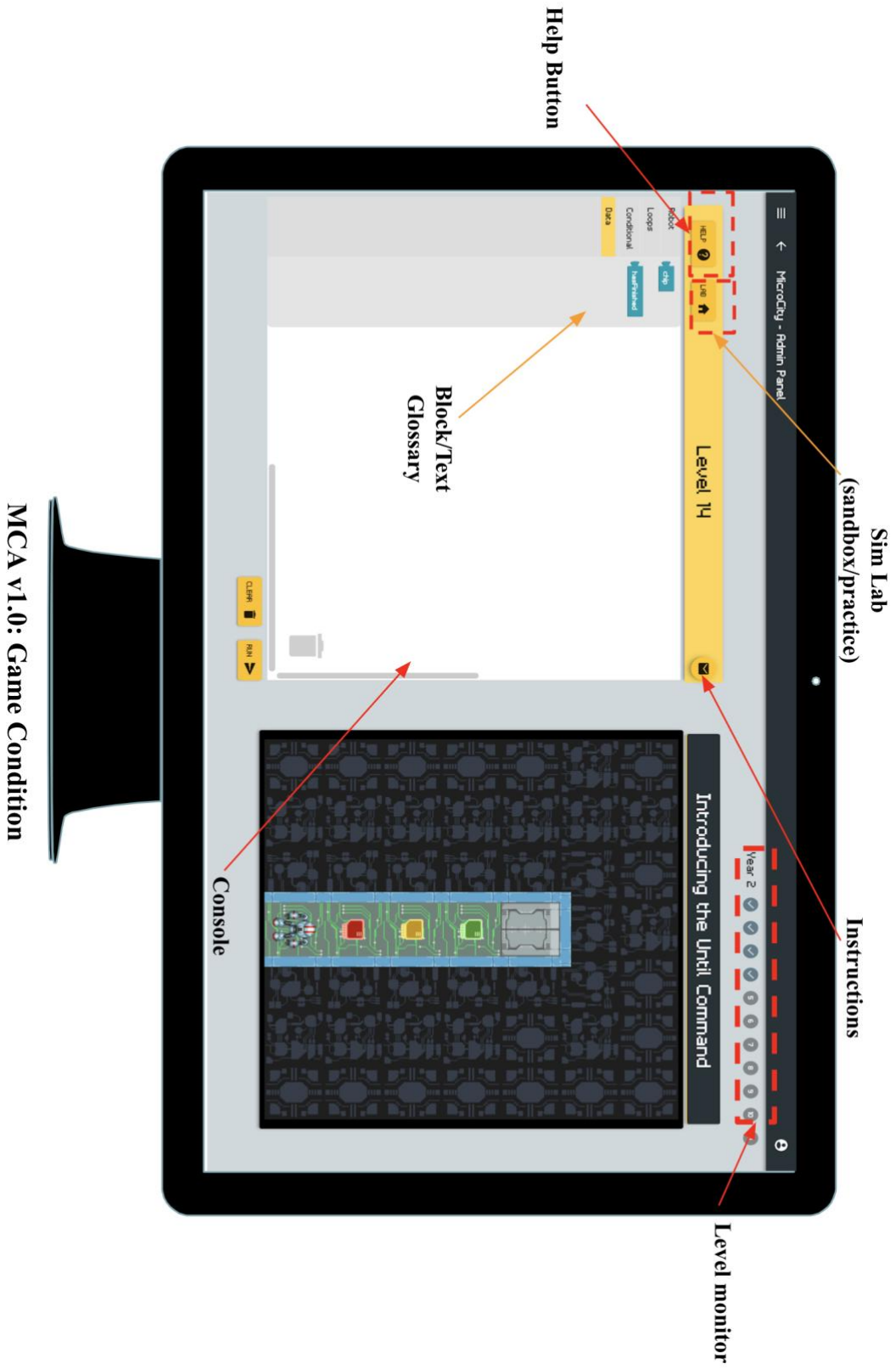
[Challenge Level 2 – Text Only]

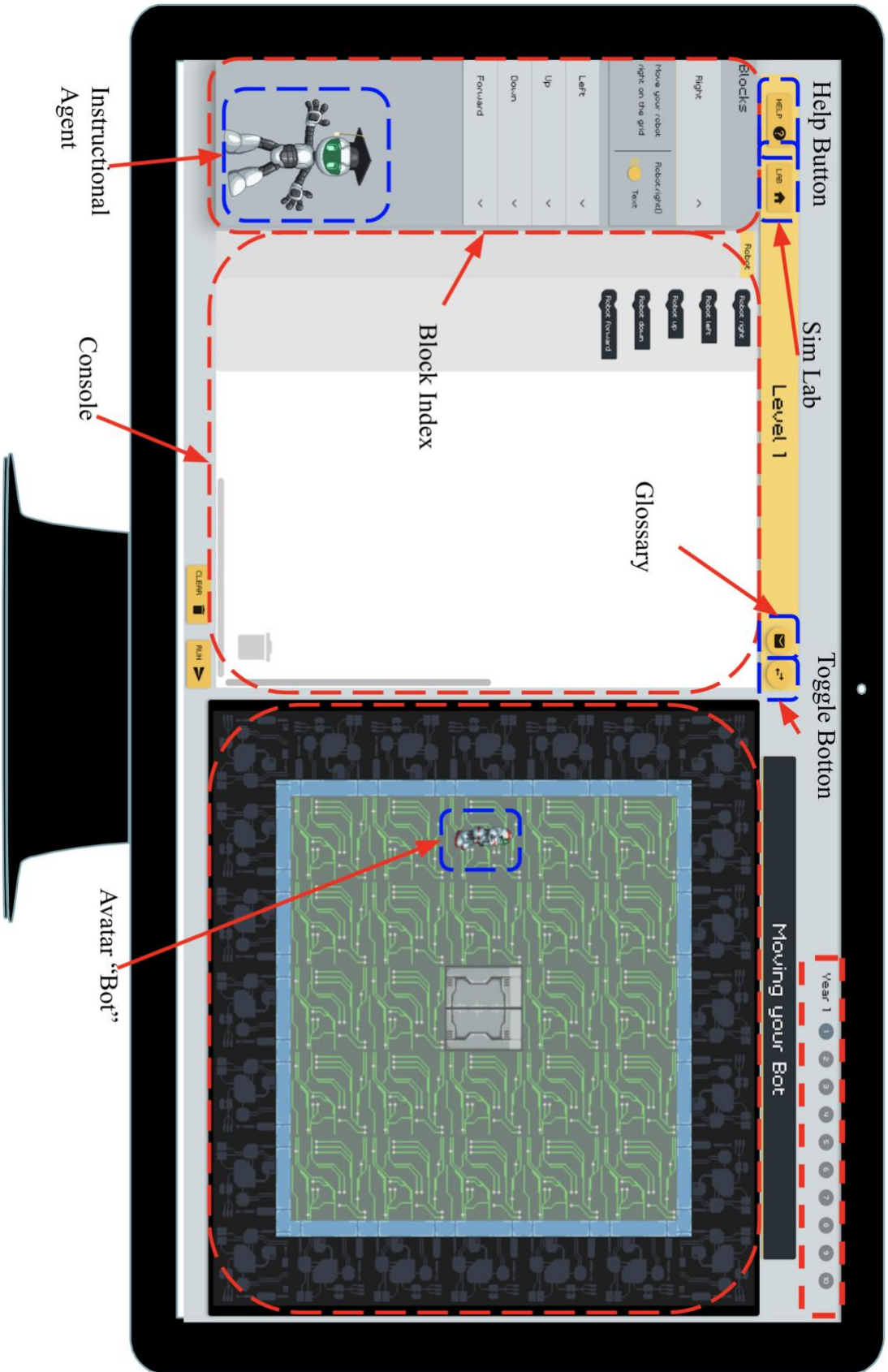
Prompt for each Challenge:

I would like you to look at this task. Try to solve it, but step-by-step, tell me what you are doing. Pretend I'm your partner and you have to tell me how to write the code. What's the first thing you want me to do?

Simple	Complex – attempt	Complex
The code uses the simple sequence code to solve the problem. NO "C-BLOCK/TEXT"	The code incorrectly incorporated the C-blocks/text to solve the problem.	The code correctly incorporated the C-blocks/text to solve the problem.

APPENDIX L: ANNOTATED SCREENSHOTS OF GAME VERSION 1 AND 2





MCA v2.0: Game Condition