USING UNPLUGGED AND BLOCK-BASED ACTIVITIES THROUGH VIDEO CONFERENCING TO DEVELOP SECONDARY SCHOOLERS' COMPUTATIONAL THINKING SKILLS

ELİF ÇİL BİLGİN

BOĞAZİÇİ UNIVERSITY

2022

USING UNPLUGGED AND BLOCK-BASED ACTIVITIES THROUGH VIDEO CONFERENCING TO DEVELOP SECONDARY SCHOOLERS' COMPUTATIONAL THINKING SKILLS

Thesis submitted to the

Institute for Graduate Studies in Social Sciences

in partial fulfillment of the requirements for the degree of

Master of Arts

in

Educational Technology

by

Elif Çil Bilgin

Boğaziçi University

2022

DECLARATION OF ORIGINALITY

I, Elif Çil Bilgin, certify that

- I am the sole author of this thesis and that I have fully acknowledged and documented in my thesis all sources of ideas and words, including digital resources, which have been produced or published by another person or institution;
- this thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- this is a true copy of the thesis approved by my advisor and thesis committee at Boğaziçi University, including final revisions required by them.

Signature……..

Date…….

ABSTRACT

Using Unplugged and Block-Based Activities through Video Conferencing to Develop Secondary Schoolers' Computational Thinking Skills

The aim of this study is to design, develop, and evaluate a learning module that brings together unplugged computing with block-based activities to develop Computational Thinking (CT) and arouse interest in computer science, especially for children from socioeconomically disadvantaged backgrounds. A mixed method study was used to investigate quantitative and qualitative data. Participants of the study were secondary school students (n=62) from different public schools. 8-week learning module consists of 8 unplugged and 7 block-based activities that were implemented through video conferencing. A CT skills test consisting of 12 questions selected from Bebras tasks and 3 questions developed by the researcher, the Self-Efficacy Perception Scale for Computational Thinking Skills developed by Gülbahar et al., (2018) and the rubrics for the assessment of the students' final Scratch projects were used as data collection instruments. Feedback questionnaires and interviews with mentors were carried out for examining the students' learning. The results showed a statistically significant increase in the students' CT skills and self-efficacy perceptions. Based on an analysis of the Scratch projects, most frequent components were events, conditional statements, and loops and the overwhelming majority of the students used an incorporated theme in their projects, which were also functional. The qualitative findings demonstrated that the majority of the students learned about Computer Science, and gained new skills such as basics of coding, and coding in Scratch.

ÖZET

Video Konferans Aracılığıyla Sunulan Bilgisayarsız ve Blok Temelli Etkinliklerin Ortaokul Öğrencilerinin Bilgi İşlemsel Düşünme Becerilerini Geliştirmek için Kullanımı

Bu çalışmanın amacı, sosyoekonomik açıdan dezavantajlı çocuklar için, Bilgi İşlemsel Düşünme (BİD) becerilerini geliştirmek ve bilgisayar bilimine ilgi uyandırmak için bilgisayarsız kodlama ve blok tabanlı programlama etkinliklerini bir araya getiren bir öğrenme modülünü tasarlamak, geliştirmek ve değerlendirmektir. Nicel ve nitel verileri işe koşan karma yöntem çalışması kullanılmıştır. Araştırmanın katılımcıları farklı devlet okullarında okuyan ortaokul öğrencileridir (n=62). 8 haftalık öğrenme modülü, video konferans yoluyla gerçekleştirilen 8 bilgisayarsız kodlama ve 7 blok tabanlı programlama etkinliklerinden oluşur. Bebras görevleri arasından seçilmiş 12 soru ile araştırmacı tarafından geliştirilen 3 sorudan oluşan BİD beceri testi, Gülbahar vd., (2018) tarafından geliştirmiş olan BİD Becerisine Yönelik Öz Yeterlik Algısı Ölçeği ve öğrencilerin Scratch final projelerini analiz etmek için rubrikler veri toplama araçları olarak kullanılmıştır. Geri bildirim anketleri ve mentorlarla yapılan görüşmeler, öğrencilerin öğrenmesini incelemek için analiz edilmiştir. Sonuçlar, öğrencilerin BİD becerilerinde ve öz yeterlik algılarında istatistiksel olarak anlamlı bir artış olduğunu göstermektedir. Scratch projelerinin bulguları ise, en yaygın kullanılan bileşenlerin olaylar, koşullu ifadeler ve döngüler olduğunu göstermiştir. Ayrıca öğrencilerin büyük çoğunluğu projelerinde birleştirilmiş bir tema kullanmıştır ve projeleri işlevseldir. Nitel bulgular, öğrencilerin çoğunluğunun Bilgisayar Bilimi hakkında bilgi edindiğini, Scratch'te kodlama ve kodlamanın temelleri gibi yeni beceriler öğrendiğini göstermektedir.

ACKNOWLEDGEMENTS

I am pleased to acknowledge the substantial contributions of those who helped me with my thesis. First and foremost, I would like to express my sincere gratitude to my advisor, Assoc. Prof. Günizi Kartal for her continuous patience, encouragement, support, and motivation during my research study. Throughout my graduate work, she was always there to answer my questions and resolve any issues that arose. This study would not be possible without valuable expertise of my advisor.

I would like to express my greatest appreciation to each of the members of my thesis committee, Assist. Prof. Mutlu Şen-Akbulut and Assoc. Prof. Filiz Kalelioğlu, for their valuable comments, feedback, suggestions, and their positive attitudes that broadened my perspective in this period. I would also like to thank Assist. Prof. Duygu Umutlu, who I consulted her content knowledge and expert opinions while preparing Scratch lesson plans used in this study.

I also thank Banu Aykın Köylüer, for helping me to reach the participants in School Support Foundation (SSF) to conduct this research.

I have to express my immense gratitude to my father, Hasan Çil and my mother Sevgi Çil who have always been sources of continuous support for me throughout my entire life. I am very thankful for my precious love Erdem, for his encouragement, continuous support, and patient.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In an increasingly digitized world, individuals are expected to be digitally literate, problem solve using technological tools, and even have basic programming skills. Being a digital citizen requires individuals to have computational thinking skills (Kalelioğlu, Gülbahar, & Kukul, 2016). Education programs that focus on science and technology, and computer science are recommended from an early age in order to achieve a welfare level for social progress and individual development to ensure scientific and technological inventions. Computational thinking skills are being integrated into school curricula around the world (Grover & Pea, 2013, Seow et al., 2019), as well as nationally (Gülbahar & Kalelioğlu, 2018). The importance of such attempts is underscored by the abrupt switch to remote delivery of education since the outbreak of the Covid-19 pandemic in Spring 2020, which made digital solutions mandatory in many fields around the world.

The necessity for integrating computer science and computational thinking in K-12 education was first raised by Papert (1980) in his seminal book *Mindstorms Children, Computers, and Powerful Ideas*. More recently, Wing (2006) emphasized the importance of computational thinking beyond computation and the need to include it in the curriculum. She defined computational thinking skill as the process of formulating problems and generating solutions using computers and argued that teaching computational thinking as one of the basic skills in the school curriculum will help students learn abstract algorithmic and logical thinking and solve complex problems.

Angeli et al. (2016) focused on how to prepare students to enhance the knowledge they need to effectively tackle the 21st century's technological challenges. It has been argued that such an educational goal can be achieved by integrating computer science as a separate discipline and a school subject into the K-12 curriculum, as many educators strongly advocate (Angeli et al., 2016; Barr & Stephenson, 2011; Grover & Pea, 2013; Rich, Spaepen, Strickland, & Moran, 2019) Computer scientists emphasized the necessity of increasing students' interest in computer science by including computational thinking skills so that students pursue their careers in this field (Lu & Fletcher, 2009; Voogt et al., 2015).

A variety of teaching methods are recommended in the literature for the development and analysis of computational thinking skills. These methods are unplugged programming activities (Bell, Alexander, Freeman, & Grimley, 2009; Taub, Armoni, & Ben-Ari, 2012), visual block based programming (Berland & Lee, 2011), the use of simulations and video games (Berland & Lee, 2011; Grover, Pea, & Cooper, 2015; Kalelioğlu, 2015; Lye & Koh, 2014) the use of robotics (Bers, 2010), and text-based programming (Bocconi, Chioccariello, Dettori, Ferrari, & Engelhardt, 2016). Programming is also included in the Turkish Ministry of National Education's curriculum, which involves the use of unplugged computing activities, block-based programming, text-based programming, and robotics (Milli Eğitim Bakanlığı, 2018).

Unplugged computing activities give a chance to younger students to build computational thinking skills and learn about basic computer science concepts without having to use computers (Bell et al., 2009). These game-like activities can help demonstrate how to deal with difficulties in the programming process while enhancing creativity. Some computational concepts such as if/then statements,

debugging, problem solving, abstraction, and decomposition can be taught through unplugged computing (Kalelioğlu, 2018). Block-based programming, on the other hand, enables students to create animated games or stories with programming blocks without having to learn the complex code structures of traditional programming languages (Resnick et al., 2009).

The purpose of this study is to design, develop, and evaluate a learning module that brings together unplugged computing with block-based activities to develop computational thinking and arouse interest in computer science, especially for children from socioeconomically disadvantaged backgrounds. The module is designed so that it can be delivered online by volunteers with little teaching experience through a video-conferencing tool or be used by 5th or 6th graders who are new to Computer Science on their own without the presence of an instructor.

1.1 Statement of the problem

Despite the efforts to integrate computational thinking skills in K-12 education, few middle school curricula cover a meaningful introduction to computer science as a discipline, and rarely contain contexts to enable the transfer of introductory programming skills to the next level. Particularly, the coding activities at the K-8 level rarely focus on understanding, computer science concepts, and transfer of learning to relevant fields (Grover, Pea, & Cooper, 2015). For example, in Turkey, programming was not included in the curriculum for many years, and the term Computer Science was not used until very recently. Currently, its use is reserved for 9th-10th grade courses in the curriculum, while the term "Information Technologies and Software" is still preferred for middle school level.

When the “IT and Software” curriculum for 5th-6th graders was renewed in 2018, the foundations of computer science and computational thinking skills were included. However, the allocation of only 2 hours per week during the first 2 years of middle school makes it still difficult to make a meaningful introduction to computer science. For this reason, unplugged computing and few block-based programming activities adopted in the program rarely reach the gains in practice mentioned by Grover et al. (2015), and Kalelioğlu (2018).

This study will address this problem by designing activities with intriguing questions and videos that arouse learners’ curiosity in the field of computer science, and by covering topics from daily life to improve their computational thinking skills through learning by doing. It is hoped that the implementation of the module will provide for some of the ICT learning loss resulting from a year spent out of school due to the Covid-19 pandemic, during which the ICT and Software class was one of the first to be canceled, as it is still not considered an essential subject in the Turkish curriculum.

1.2 Purpose of the study

The primary purpose of this study is to design, develop, and implement an online learning module to help 6th graders develop their computational thinking skills and increase their computational thinking self-efficacy, and the learning loss experienced during the COVID-19 pandemic. The module is intended for use by children on their own or with the help of a remote mentor who will scaffold the process as a knowledgeable other (Vygotsky, 1978). The secondary purpose of the study is to evaluate the module from the students’ and mentors’ point of view to unveil the strengths and weaknesses during the implementation process. It is hoped that this

effort will also help guide volunteers from educational foundations delivering online support during the COVID-19 pandemic to socioeconomically disadvantaged children.

1.3 Significance of the study

As mentioned above, computational thinking comprises an important set of skills in the 21st century, and the development of these skills from an early age is recommended around the world. However, there are not sufficient opportunities to do so in middle school, given the limited amount of time dedicated to the IT/CS class in the curriculum. The learning module prepared as part of this study will be built around the main computational concepts to develop computational thinking skills and is intended to help meet the need identified.

In the literature, there are studies on unplugged computing activities, block-based programming, and text-based programming and others. However, these studies generally focus on which method is more effective, or whether or not one type of computing (i.e., unplugged or block-based) improves computational thinking. In other words, there are not many studies involving unplugged computing activities that lead to block-based coding activities and are designed to complement one another. This study, on the other hand, will bring together unplugged computing and block-based programming to help develop computational thinking in a progressive manner, as the module starts out with unplugged activities and builds upon these skills in the following block-based activities. Therefore, it is hoped that this study will offer a new approach, and thus contribute to the literature. Also, during the COVID-19 pandemic, much learning loss resulted during (emergency) remote

teaching, as expected. This study will also be helpful in filling the gaps that occurred in learning in middle school.

CHAPTER 2

LITERATURE REVIEW

Computational thinking skills, and the development of computational thinking (CT) in elementary and secondary schooling as part of the Computer Science curriculum will comprise the conceptual background of the study. Constructionism, as defined by Papert (1980), and promoted by Bers (2020), is the learning theory that provides the theoretical framework of the study and is adopted during the entire design process.

According to Papert's theory of constructionism, which is based on Piaget's constructivist theory, learning occurs through doing. Papert (1980) argued that children learn by making things with whatever is available in the culture around them, and children should learn Computer Science by doing it, as learning by doing will support creating their own knowledge. The importance of "making" in learning is emphasized (Ackermann, 2001), and "learning to learn" is fostered. As pointed out by Bers (2020), according to constructionism, learning occurs when learners engage with concrete objects to explore and build meaningful things using them, share what they have built with the community, and engage in self-reflection. This approach supports that the best way to learn and explore independently happens when learners can create, make, program, and design their own products in an enjoyable way (Bers, 2008). When learners are given "small objects" to acquire knowledge, by making, repairing, and playing with them, they become not only persons who create a new product, but also discover new knowledge themselves, such as coding and programming (Bers, 2020).

The literature review below starts with a definition of computational thinking in Computer Science, and the subcomponents of CT, such as algorithmic thinking, abstraction, decomposition, debugging, and generalization. Then the techniques and procedures employed in the development of CT are described; unplugged and block-based computing studies are reviewed as ways of developing CT. In addition, ways of assessing the changes in computational thinking are discussed. Finally, further empirical studies on computational thinking are reviewed.

2.1 Computational thinking

Although Seymour Papert was the first to use the term computational thinking (CT), he did not create a definition precisely during his career (Lodi et al., 2020). The term has drawn more attention in recent years after being discussed in Wing's (2006) short article, where she described computational thinking as a way of "solving problems, designing systems, and understanding human behavior by drawing on the concepts fundamental to computer science." (p.33).

After Wing's definition of CT, many researchers conducted studies to create a definition of CT (Weese, 2017), which is a multi-dimensional skill, as evident in the variety of definitions available in different studies (Gülbahar, Kert, & Kalelioğlu, 2018). The most important common aspect of these definitions is problem solving, which is based on problem-solving processes such as understanding and formulating problems (Wing, 2006).

Similarly, ISTE (International Society for Technology in Education) and CSTA (Computer Science Teachers Association) define computational thinking as a problem-solving process that includes the following features (CSTA & ISTE, 2011; ISTE, 2021):

- “Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data;
- Representing data through abstractions such as models and simulations;
- Automating solutions through algorithmic thinking (a series of ordered steps);
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources;
- Generalizing and transferring this problem-solving process to a wide variety of problems”.

Systematically following a conceptual path and correctly processing tasks to solve complex problems are also underlined in definitions of CT (Voogt et al., 2015; Faber, Wierdsma, Doornbos, & van der Ven, 2017).

According to Weinberg’s definition (2013), the process of problem-solving in CT includes the formulation of the problems to facilitate the process, visualization of the problems via models, and identifying and implementing solutions to reach more effective ways and resources. Wing (2008) expanded her first CT definition to include analytical thinking, mathematical thinking, engineering thinking and scientific thinking.

Tsarava et al. (2017) stated that the development of computational thinking enables the creation of new concepts that are coding and algorithmic thinking. Computational thinking donates the idea of decomposing a problem, identifying variables, and creating algorithmic solutions that resemble the steps of coding.

In a meta-analysis of 125 articles, Kalelioğlu et al. (2016) listed the most common subskills of CT as abstraction, problem-solving, algorithmic thinking, and pattern recognition. Angeli et al. (2016) identified the CT sub skills as abstraction,

generalization, decomposition, algorithmic thinking, and debugging and proposed a CT curriculum framework based on these skills for K-6. They also identified the type of knowledge needed to teach CT using this curricular framework, and tired it out with elementary school teachers.

Table 1. Concepts Related to Computational Thinking

Grover and Pea (2013)	Selby and Woollard (2013)	Angeli et al. (2016)	Kalelioğlu et al. (2016)	Shute et al. (2017)	Relkin, Ruiter and Bers (2021)
abstractions and pattern generalizations	abstraction	abstraction	abstraction	abstraction	
algorithmic notions of flow of control	algorithmic thinking	algorithmic thinking	algorithmic thinking	algorithm design	algorithms
conditional logic					
debugging and systematic error detection		debugging			debugging
structured problem decomposition	decomposition	decomposition		problem decomposition	modularity
					design process
efficiency and performance constraints	evaluation				
iterative, recursive, and parallel thinking	generalization	generalization		recursion	
			pattern recognition		control structures
			problem-solving	problem reformulation	
				system testing	
systematic processing of information					
symbol systems and representations					representation
					hardware /software

Based on a synthesis of Table 1, and in line with Angeli et al. (2016), Grover and Pea (2013), and Kalelioğlu et al. (2016), the main CT skills covered in this study were algorithmic thinking, abstraction, decomposition, debugging, and generalization. These skills are discussed below in more detail.

2.1.1 Algorithmic thinking

Algorithmic thinking, one of the important components of computational thinking (Brown, 2015), is defined as the abilities to understand and construct an algorithm (Futschek, 2006). An algorithm is a step-by-step procedure to accomplish a task, and is not just related to computer science, but used in other disciplines (Selby & Woollard, 2013) such as mathematics (Lockwood & Dejarnette, 2017), and STEM related fields (Rich, Yadav, & Schwarz, 2019).

Futschek (2006) listed the abilities involved in algorithmic thinking as analysis of a given problem, precise specification of a problem, finding a basic solution to a given problem, developing a correct algorithm to a given problem using the basic solution, thinking about all possible solutions to a problem, and enhancing the efficiency of an algorithm. In addition, Brown (2015), and Cansu and Cansu (2019) defined algorithmic thinking as a process of understanding, implementing, assessing, and designing algorithms to create solutions for problems.

2.1.2 Abstraction

According to Wing (2008), abstraction is a crucial sub-skill of computational thinking. Abstraction is a process of making problems more understandable by reducing unnecessary details (National Research Council, 2010; Wing, 2006).

The important aspect of abstraction is to select the unnecessary details without losing the necessary parts. In this way, the problem-solving process is simplified (Csizmadia et al., 2015), by eliminating unnecessary details and complexity to reduce a problem to its core (Yadav, Stephenson, & Hong, 2017)

The abstraction that allows finding relationships between problems and reviewing them in solution processes, has been stated as the most difficult sub-skill of computational thinking (Booth, 2013).

2.1.3 Decomposition

Decomposition is defined as a process of breaking down large problems into simpler parts (Rich, Egan, & Ellsworth, 2019; Wing, 2006). Solving the problem by dividing it into small pieces instead of solving it as a whole makes the problem-solving process easier (Liskov & Guttag, 2000). However, small parts must be solved independently from each other and when small parts are joined together, the solution of the main problem should be provided (Liskov & Guttag, 1986). For example, programmers can design and create different parts of a game independently, and then bring the pieces together to compose a whole game (Csizmadia et al., 2015).

2.1.4 Debugging

Debugging is the process of systematically determining problems in the code and fixing malfunctioning code. It is described as an important part of programming and computer science (McCauley et al., 2008). Debugging is mostly used in the process of improving the prepared codes or algorithms (Brennan & Resnick, 2012), as it is a systematic process of eliminating existing bugs, or errors, in a code when the program does not provide actions (Wong & Jiang, 2019).

2.1.5 Generalization

According to Csizmadia et al. (2015) generalization is about determining patterns and similarities and reaching the result by using them. It is a way of solving a new problem quickly based on the solution of a previous problem and prior knowledge (Csizmadia et al., 2015). Selby and Woollard (2013) defined generalization as the "understanding how to draw a square by defining internal angles, then applying the same algorithm to produce an approximation of a circle." (p.4)

2.2 Developing CT skills

Over the last two decades, the use of block-based programming tools such as Scratch, Alice, Game Maker, Code and Greenfoot in schools has increased. The use of such block-based platforms at the elementary and middle school levels were found to develop CT skills (Lye & Koh, 2014; Román-González, Pérez-González, & Jiménez-Fernández, 2016; Shute, Sun, & Asbell-Clarke, 2017). Unplugged activities have also been used for CT development, and studies showed that after school programs of unplugged computing have been effective (Delal & Oner, 2020; Kalelioğlu, 2018; Rodriguez, Rader, & Camp, 2016). Apart from these, web-based simulation development tools and Arduino and robotic kits that allow tangible programming to have gained popularity in programs for elementary children (Grover et al., 2015).

Repenning, Webb, and Ioannidou (2010) indicated that children were introduced to computational thinking in K-12 education with "low threshold, high ceiling" tools and environments, which are suitable for programmers at all levels. In a meta-analysis on the development of CT skills in school, Kalelioğlu et al. (2016) found that both block-based, and unplugged activities were used in the curricula to

teach computational thinking skills to elementary and secondary students. However, Grover and Pea (2013) argued that developing CT will not be enough without an introduction to computing, only with unplugged CS activities.

In a more recent review on CT instruction, Gülbahar et al. (2018) reported that computer games and simulations were commonly used to teach computational thinking skills at the elementary and secondary levels.

Since both types of activities were generally found useful for the development of CT skills, the findings from research on the development of CT through both unplugged computing and block-based programming are presented below.

2.2.1 Unplugged computing

Unplugged computing or unplugged computer science involve programming or coding activities kinesthetically without using computational devices. These unplugged activities that make use of computing concepts may be done through physical actions, the use of real-world objects, role play, using paper and pencil, etc. (Aranda & Ferguson, 2018). These activities may not simulate the work process of a computer, but they are designed for students to discover the principal ideas of computing without learning the procedures of writing code (Bell et al., 2009).

Unplugged activities are considered a framework for a constructivist approach to teach computer science without computers (Weigend, Vaníček, Pluhár, & Pesek, 2018). Fundamental steps and principles of computing processes can be learned through paper-pencil and handmade activities. By observing these activities done in the classroom and by applying it themselves, students can conceptualize this knowledge with minimal help. It has been shown that unplugged activities can

enhance computational thinking (Brackmann et al., 2017; Faber et al., 2017), and can be transferred to learning programming skills in the future (Gaio, 2018). Additionally, contextualizing computer science concepts in these activities promotes conceptual learning by making use of real-world contexts (Grover, Jackiw, & Lundh, 2019; Weigend et al., 2018). They provide a chance for students to focus more deeply on the concepts of the field by eliminating the focus on computers, which are actually considered a toy by many children (Bell & Vahrenhold, 2018; Bell et al., 2009). Another important goal of this approach is to decrease the gap between K-12 teachers who may not have a technical background (Rodriguez, Kennicutt, Rader, & Camp, 2017).

Quasi-experimental design research that explores the effects of unplugged activities on the students' computational thinking shows that unplugged activities remove deficiencies in computational science, and they help to develop computational thinking skills for students by allowing them to see fundamental steps in computing (Looi et al., 2018; Brackmann et al., 2017; Song, 2019; Takaoka et al., 2014).

Brackmann et al. (2017) conducted a quasi-experimental design research with 73 students in the 5th and 6th grades from two different public primary schools in Spain to investigate the effectiveness of unplugged programming activities to develop CT skills at the elementary level. The experimental group worked with unplugged activities developed or adapted by the researcher as part of their regular classes, while the control group did not receive an intervention. A computational thinking test, developed by Roman-Gonzalez et al. (2016), was used as pre- and post-test to assess decomposition, pattern recognition, abstraction, and algorithmic design.

The results showed that students in the experimental group developed their skills more than the students in the control group (Brackmann et al., 2017).

Faber et al. (2017) designed an introductory course in computational thinking for students in their final year in primary school. Classes will last for six weeks, and each lesson will be 90 minutes in 26 schools in the north of the Netherlands. Researchers used different sources while preparing the content of the unplugged programming lessons. First, they identified the programming concepts they would teach: algorithms, variables, repetition and conditionals. Afterwards, learning outcomes and activities were determined. Each lesson started with an explanation of the concept to be learned, followed by a discussion of how to use this concept to make daily life easier. Focus group interviews were conducted with teachers to elicit their evaluation of student performance and the six-week course. Evaluations from the focus group discussions were used to develop lesson plans. At the end of the study, researchers stated that the parts of the course materials containing the unplugged coding activities seemed to receive positive reactions from both teachers and students.

Tsarava et al. (2017) combined both unplugged and computer-based activities to develop an extracurricular CS course to teach primary school students CT concepts. The course consisted of eight lessons that began with unplugged activities and continued with application design, using the AppInventor, which provides a platform to develop applications for Android phones. Unplugged scenarios were used in class to develop students' decomposition and generalization skills. For example, the students first studied events and parallelism, and then they were expected to transfer these computational thinking sub skills to designing an application in the AppInventor, by reusing or modifying their learning in the new environment. A

gamified assessment framework, based on Dorling and Walker's (2014)'s framework (cited in Tsarava et al., 2017) was used for evaluation, where badges were awarded for successful acquisition of coding concepts, core CT processes, STEM specialization, creativity, and social skills. The results showed that the students learned to create their own solutions in the unplugged activities, and they were able to transfer their learning to a coding program.

Bebras is an international organization that aims to teach computational thinking and computer science to students. The idea of international challenges was proposed by Valentina Dagiene in Lithuania in 2004 (Kalelioğlu, Doğan, & Gülbahar, 2021). Translated into Turkish as "Bilge Kunduz," the Bebras challenges are also held online in many countries to examine students' abilities on computer science and computational thinking skills (Gülbahar, Kalelioğlu, Doğan, & Karataş, 2020), and are performed as "Bilge Kunduz" competition in Turkey (https://bilgekunduz.org). The competition is held for children between the ages of 6 and 18 and there are 15 questions/tasks consisting of 3 difficulty levels as easy, medium, and hard to be solved in 45 minutes. Bebras competition questions include 5 sub-dimensions of computational thinking skills; abstraction, decomposition, evaluation, generalization and algorithmic thinking (Kalelioğlu et al., 2021).

Bebras tasks can be completed by students with no prior knowledge of informatics. In order to solve these tasks, students need to do calculations, make decisions, establish cause-effect relationships, use high-level thinking skills such as analytical thinking and problem solving (Gülbahar et al., 2020).

2.2.2 Block based programming

Block-based learning environments involve animated stories where students need to decompose and recompose screens and movements of characters. For kindergarten children, toy robots like Bee-Bot are used to teach coding (Atmatzidou & Demetriadis, 2016). This kind of program gives a chance to practice abstraction and decomposition by splitting the actions to move the robot (Brennan & Resnick, 2012).

The first drag and drop block-based program was Logo Blocks, and it resembled Alice and Scratch that enable novice learners to code without using textual language (Duncan, Bell, & Tanimoto, 2014). In Papert's turtle robot, logo commands were used to control it and its movements could be seen physically on the floor (Papert, 1980).

Scratch is a one of the most widely used block-based programming tools (Zhang & Nouri, 2019), developed by the Lifelong Kindergarten research group at the MIT Media Lab (Brennan & Resnick, 2012; Resnick et al., 2009). The Scratch platform also provides an opportunity for the users to share their projects, see other users' projects, and interact with each other, and share the source codes of their projects. Most of the users are young people in the 8-16 age group, but a large group of adults also uses the program. Research showed that using the Scratch program helped develop creative thinking skills, collaborative working skills, computational and mathematical skills, and reasoning skills, namely the essential skills of the 21st century (Resnick et al., 2009).

Wong and Jiang (2019) designed a CT curriculum employing Scratch and tested it with 85 fifth graders in Hong Kong to see whether or not it helped develop algorithmic thinking and debugging skills. Before starting the 5th grade curriculum,

this group of students completed the 4th grade lessons during the first year of this longitudinal study. The 5th grade curriculum was prepared according to the feedback received from the teachers and students. In 4th grade, the students took coding courses that comprised unplugged computing activities and coding in Scratch to develop basic knowledge of programming. The 5th grade curriculum was designed to develop students' algorithmic thinking and debugging skills. The 6-week curriculum began with a review of basic programming concepts and continued with algorithm examples from daily life. The students then tried to create algorithms for a given game using the code blocks from Scratch and learned debugging by finding the errors in the given Scratch examples. Then they designed games according to the instructions given, and finally they created their own games. The results of the pretest and posttest revealed significant differences in algorithmic thinking and debugging skills, before and after the CT lessons. In addition, the results showed that the CT curriculum helped students to analyze given problems deeply and produce adequate solutions for the problems. (Wong & Jiang, 2019).

Kalelioğlu and Gülbahar (2014) examined the effect of Scratch on 5th graders' problem-solving skills and their views about programming in a 5-week study, where the participants developed various applications with Scratch. The "Problem Solving Inventory for Children" developed by Serin, Bulut Serin, and Saygılı (2010) was used to test the participants' problem-solving skills. According to the results, the programming in Scratch did not cause a significant difference in the problem-solving skills of primary school students, but the participants stated that they liked developing programs with Scratch, and they wanted to further develop their programming skills.

In another study with 5th graders, Oluk, Korkmaz and Oluk (2018) investigated the effects of using Scratch on computational thinking skills and algorithm development. Computational Thinking Scale developed by Korkmaz, Çakır and Özden (2015) and Algorithm Development Achievement Test (AGBT) developed by the researchers were used as assessment instruments. AGBT consisted of 27 multiple-choice questions measuring ICT learning objectives, such as algorithm design, debugging and rearranging an algorithm. The results of the study showed that the students who learned algorithms in Scratch significantly increased their algorithmic thinking compared to the students who learned through direct instruction.

Erdem (2018), who also worked with 5th grade students, used two different delivery formats, face to face regular instruction and flipped classroom techniques to teach Scratch programming to see whether or not the teaching method made a difference for CT development. The results showed no statistically significant difference between the groups according to self-efficacy perceptions and CT skills. However, there was a significant difference in the pretest and posttest scores within the groups. Focus group interviews revealed that the students in the flipped classroom enjoyed learning on their own, even if they did not get help immediately. The students in the face-to-face class, on the other hand, said that learning with the teacher was effective, despite difficulties due to class factors. However, both groups argued that discovery learning was better, and they said that they improved their game development and animation skills.

Ozer (2019) studied the impact of using robots for coding on 5th and 6th grade students' problem-solving skills in a quasi-experimental study. The experimental group were taught using block-based robotic programming (Scratch +

WeDo 2.0), while the control group used a block-based programming tool (Scratch) for 12 weeks. Based on an achievement test developed by the researcher, a statistically significant difference was found between the pre-test and post-test scores of both groups. The programming achievement test scores of the students in the experimental group was higher than that of the students in the control group. In addition, “Problem Solving Inventory for Primary School Age Children,” developed by Serin et al. (2010), was given to measure the students' problem-solving skills. The experimental group’s posttest scores of the inventory were found significantly higher than that of the control group, when the pretest scores were controlled for (Ozer, 2019).

Zhang and Nouri (2019) examined the effects of Scratch, which is one of the block-based programming environments, on participants' CT skills by using systematic review methods. In the literature review, the findings of studies examined the effects of using Scratch on students' CT skills through the differences between the control and experimental groups, or the differences before and after using Scratch showed that Scratch helped students discover and develop their computational thinking skills (Zhang & Nouri, 2019).

2.3 Assessment of CT skills

In recent years, different methods have been used to evaluate computational thinking skills. In this section, research focused on measuring the development of computational thinking skills and the methods used will be reviewed.

New approaches have emerged to enable individuals to acquire computational thinking skills, and several approaches are considered for their assessment. Evaluating an abstract skill such as thinking is considered a difficult process.

However, it is possible to evaluate CT skills with indirect methods. One approach is to collect perceptions of their own skills and what they think they are capable of, through surveys, such as those developed by Korkmaz et al. (2017) and Gülbahar et al. (2018).

In a study of self-efficacy perceptions of secondary school students on CT skills, İbili, and Günbatar, (2020) used the Self-Efficacy Perception Scale for Computational Thinking Skill developed by Gülbahar, Kert, & Kalelioğlu (2018) to investigate different variables such as gender, time spent block-based programming, and ease of access to computers at home. The analysis of the data from 232 students showed different results according to the variables. Gender did not seem to make a difference for CT skills. Usage of computers at home did not have any impact either on the students' self-efficacy perception. However, it was observed that the results of the students who used block-based programming tools showed higher results than the students who never use block-based programming tools. Time spent using a block-based programming tool did not affect the results.

The Computer Thinking Skills Scale, also cited in Shute et al. (2017), was developed by Korkmaz, Çakır and Özden (2017) to measure the CT skills of university students. The five-point Likert type scale consists of 29 items and five factors, Creativity, Algorithmic Thinking, Collaboration and Critical Thinking and Problem Solving. The consistency coefficient was 0.822. The construct validity of the scale was investigated by both confirmatory and exploratory factor analysis, and it was concluded that it is a valid and reliable tool for measuring the CT skills. The same scale was adapted for use with elementary and secondary school students by the same authors. This version is a five-point Likert type scale consisting of 22 items. The validity and reliability of the scale was tested with 241 seventh and eighth

graders, Exploratory and confirmatory factor analysis, item distinctiveness analyses, internal consistency coefficients and constancy analyses were conducted. The scale was found valid and reliable to measure the CT skills of secondary school students. Cronbach alpha reliability coefficient of the scale was 0.809 (Korkmaz et al., 2015)

Lee et al. (2011) indicate that students' CT skills and self-efficacy perception can be enhanced by activities that enhance students' experiences or by group work. Also, encouraging students with supportive feedback may help students to develop their CT self-efficacy perceptions. Activities both in school and out of school are also suggested to develop students' computational thinking skills. The environments in which students use their existing knowledge and develop new products by developing this knowledge can be supportive to develop their computational thinking skills (Lee et al., 2011).

Rather than depending on the students' perceptions of their own skills, many researchers developed achievement tests to assess students' CT skills in order to see whether or not their interventions did actually help students build these skills. Seiter and Foreman (2013) designed a Scratch based assessment tool, "Progression of early computational thinking," to assess elementary school students' CT abilities. The test included evidence variables such as operators and conditionals, pattern variables such as motion and interactivity, and CT concepts such as algorithms, abstraction, and decomposition.

Yavuz Mumcu and Yıldız (2018) developed an "Algorithmic Thinking Test" which consisted of 12 open-ended questions to measure 5^{th} and 6^{th} graders' algorithmic thinking skills. This instrument comprises four parts, algorithmic tasks, tracing tasks, logic tasks, and analysis tasks. In the algorithm task questions, students need to develop algorithms to solve the problems given. Students use given

algorithm steps to develop an algorithm to solve a problem in the tracing task questions. In logic tasks, students create appropriate algorithms by reasoning. Students question the effectiveness of the given algorithms and reorganize the steps to create an algorithm to get the expected solution. Ozer (2019) developed a "Programming Achievement Test" for use at the 5th and 6th grade levels. The first part of the achievement test addressed algorithm learning objectives, and the second part was constructed with questions related to Scratch, using pictures of code blocks.

A validated reliable instrument to test CT skills did not exist in the literature until recently when Roman-Gonzalez et al (2016; 2017) developed the Computational Thinking Test (CTt). This test was not available in Turkish at the time this study was conducted.

Researchers suggested that multiple choice tests will not be sufficient for a rich assessment of CT skills (e.g., Grover et al., 2015). Thus another approach recommended for assessing CT skills is formative assessment, and evaluation of student projects and portfolios, which can be supported by student interviews (Brennan & Resnick, 2012; Grover et al., 2015) In a meta-analysis of 45 papers published after 2006, Shute et al. (2017) identified the computational sub-skills of decomposition, abstraction, algorithms, debugging, and generalization, and listed the intervention tools: Scratch, C programming language, Alice, and CT module. They identified three types of assessment tools: Scratch based assessment, game-based assessment, and validated CT scales for generic use. As Shute et al. (2017) indicated, Brennan and Resnick (2012) adopted a CT assessment which consists of formative analysis tools, interviews, and design projects to assess 8- 17 years old Scratch users' CT abilities and evaluate users' portfolios and projects. To assess users' thinking processes, interviews were used. Also, three sets of design projects were used to

assess users' debugging and programming skills. Each set consisted of two tasks and the user had to select one from each set.

Grover (2014), who argued for a multi-faceted assessment of CT skills, developed both a CT curriculum and assessment of CT. In Grover et al. (2015) they proposed "a system of assessments" to measure computational thinking skills, as part of their research effort designing and testing a Computer Science course entitled FACT (Foundations for Advancing Computational Thinking). In this approach, formative and summative quizzes and tests, open-ended programming assignments, a transfer test and product evaluation were used to assess skills development. In the formative assessment part, the questions allowed students to use their creativity and add their own distinctive elements in problem solving. The summative assessment consisted of multiple choice and open-ended questions about code tracing and debugging to assess students' CT ability. A transfer test was developed to measure how students transfer their computational understanding developed in Scratch to the Java/ Pascal programming language (Grover et al., 2015).

Denner, Werner and Ortiz (2012) presented strategies to evaluate students' games created in block-based programming platforms. Through a literature review, Denner et al. (2012) deduced that computer game programming can be useful for engaging students in three core abilities, programming, design for usability and organizing and documenting code. A total of 108 student projects developed using the Creator program were examined according to these competencies. In addition to tutorials about the platform, students were given courses related to game design, with handouts containing the rules necessary to design games step by step. At the end of the implementation, students were asked to prepare 5 games of 4 different genres based on predetermined criteria. The findings from students' projects were grouped

under three core abilities. As a result, the features used in student projects were analyzed under the categories that emerged based on the features themselves. The results showed that conditions or events, door functionality and conditional character interaction were the most common programming elements. In addition, 84% of the projects contained multiple stages of play feature, and 54% of the projects had clear goals, and 53% used rule grouping (rule boxes) in their projects, evaluated under the criterion of organizing and documenting code.

2.4 Computer science and CT in formal education

The development of CT skills and an understanding of Computer Science require much more than just a few hours of coding lessons, and therefore any attempt to integrate CT into the curriculum will need to take into account multiple factors. It is generally recommended that children be introduced to CT concepts at an early age (Bers, 2018; Bers & Sullivan, 2019; Papadakis, Kalogiannakis, & Zaranis, 2016; Relkin, Ruiter, & Bers, 2020).

In the U.S. and around the world, the standards set by the International Society for Technology in Education (ISTE) have been influential in technology integration efforts in education, including CT skills. Recently, ISTE collaborated with the Computer Science Teachers Association (CSTA) to determine competencies for educators. CSTA developed and revised Computer Science standards for all grades starting with kindergarten through 12th grade (CSTA, 2017), with 4 major levels, based on 5 sets of skills, ranging from computing systems, and data analysis to programming, and impact of computing. In the EU countries, an effort has been undertaken to promote CT skills in formal education.

In a comprehensive study of the state of affairs regarding the integration of CT into the school curricula in European and other countries, Bocconi et al. (2016) and Bocconi et al. (2021) reviewed CT from various perspectives as part of the European Union CompuThink project. The study aimed to reveal the common concepts and features of CT and point out the necessity of CT skills for both individuals and countries. The authors argued that a focus on CT may make a significant contribution to economic growth, by filling the unemployment in the field of information and communication technologies and preparing for the future occupational groups. The development of computational thinking skills in individuals will help the development of different thinking skills and problem solving and produce multiple solutions in daily problems (Bocconi et al., 2016). According to the report, England was one of the first European countries to include primary and secondary level coding and CT lessons in its curriculum. In Finland, Norway and other Northern European countries computer education started in lower age groups, while in France, Italy, and Spain, it started at the secondary school level. Not only European countries, but other countries have introduced coding and programming education at primary and even kindergarten levels, such as Australia, Japan, Singapore, South Korea, the U.S., and Turkey (Bocconi et al., 2016). The changes that occurred between 2016 and 2021 as a result of the integration of CT in the curricula were reported in Bocconi et al. (2021): 25 countries renewed their CT curricula between 2016 and 2021. Countries cited a number of reasons for including CT skills in their curricula and the most commonly cited reasons were that CT improved logical thinking and problem-solving skills (Bocconi et al., 2021).

While CS curricula and CT skills were being integrated into formal education, a major concern was about how to best teach CS at various levels of

schooling in a developmentally appropriate way. The National Centre for Computing Education argued that teachers needed to have certain strategies to teach computing effectively to students. They recommended 12 principles which can be used to understand the pedagogy needed to teach computing effectively. These principles are “lead with concepts”, “work together”, “get hands-on”, “unplug, unpack, repack”, “model everything”, “foster program comprehension”, “create projects”, “add variety”, “challenge misconceptions”, “make concrete”, “structure lessons” and “read and explore code first”. Since most of the computing concepts are abstract, technological tools, concepts of programming language are needed to help students enhance their understanding. Also, student collaboration in projects is important to share their understanding among their peers. Educators were expected to benefit from the pedagogical framework suggested by structuring their lessons based on the “use-modify-create” cycle; using the existing examples, modifying them, and finally, creating their own example. Concrete examples such as real-world examples, such as kinesthetic activities are needed to make abstract and complex concepts more concrete in teaching computing. (Teaching Computing, 2020)

Six different computing concepts and five approaches which used to teach computational concepts were defined. These five approaches are tinkering, creating, debugging, persevering, and collaborating. Tinkering is discovering something new and understanding what it is and how it works. It contributes to understanding by establishing a cause-effect relationship. Creating is about planning and creating projects. Students need to design a project by breaking down small parts and then finding the relevant materials and integrating what they find and finally test the projects. Debugging is about finding and fixing bugs in code. Students can find and solve errors by making assessments on their own or with their peers. Persevering is

about being determined to code because computing is a complex process. That's why it's necessary to encourage students to write code and encourage them to break problems down into small parts while solving them. Collaboration is important for students to solve problems in large groups and to share their ideas with each other (Barefoot, 2014)

Code.org curriculum aims to enhance students' CT skills by encouraging students to think like computer scientists and improve CT practices which are problem solving, creativity, collaboration, communication, and persistence. Code.org activities are a blend of online and unplugged coding activities. Each activity is prepared in such a way that students can do it without the need for help from others and at their own pace by following the instructions provided to explore and apply algorithmic thinking. Unplugged computing lessons, on the other hand, consist of activities in which hands-on exercises are used to teach computational concepts. The course contents in the curriculum differ according to the needs of age groups. Each lesson consists of approximately 18 to 22 lessons and each activity in the lessons aims to use what they learned in the previous activity. Although the Code.org curriculum has been prepared taking into account ISTE and CSTA standards, Math, English Language Arts and Science standards are also integrated into the lessons according to the purpose (Code.org, 2022).

Despite efforts to integrate CT and CS courses in the curriculum, there is still a lack of clarity of what Computer Science is at the elementary and even secondary levels of schooling (Grover & Pea, 2013). Grover et al. (2015) stated that the students' lack of knowledge about CS affects their educational choices, and therefore only a few students may end up studying CS as a major.

In a study to address misconceptions and negative attitudes about computer science, Bell et al. (2009) developed a series of unplugged learning activities to address these misconceptions and difficulties. The activities were designed to explain computer science as a discipline in an entertaining way. Questionnaires and interviews were used to examine the students' perceptions and attitudes towards CS. Contrary to the hypothesis of the study, the results showed that although students generally understand what computer science is, they saw the computer as the essence of the field, rather than a tool for computing. That's why the researchers concluded that more activities need to be developed to change this view of computer science.

In the introductory CS middle school curriculum designed and tested by Grover et al. (2015) the cognitive and affective aspects of learning CS through well-designed activities and instruction. They also focused on misconceptions of computing and strategies for assessing transfer from block-based programming activities to text-based programming to attain an extensive picture of students' deeper CS learning. One of the main goals of the study was to help students understand CS as a "problem solving discipline with applications in the real world," and that "computer scientists are creative problem solvers who work on problems to make our lives better and easier using computation." (p.209). In addition to CT skills, the changes in students' views of computer science were also measured. The results showed that students developed their algorithmic thinking skills, and they were able to transfer their learning from block-based programming "Scratch" to a text-based programming and they developed their perspective through computer science as a discipline (Grover et al., 2015).

Relkin et al. (2021) conducted a study to investigate the effect of a seven week "Coding as Another Language" (CAL) curriculum on first and second graders'

CT skills. KIBO robotics platforms were used to teach programming via wooden blocks that scans using a barcode scanner. The CAL curriculum was designed to enhance students' CT skills which are algorithms, modularity, hardware/software, control structure, debugging, representation and design process. CAL curriculum consists of activities that support students' collaboration and creativity. As an example activity, the students were asked to write compositions for the part of a story consisting of 6 pages of illustrations using robots. Students programmed KIBO robots and tested whether the code blocks worked. "TechCheck" unplugged computing scale instruments for the younger age group was used to examine the change in students' CT skills. The result showed improvement in students' scores. In addition, improvement in algorithms, modularity and representation skills were examined. They found that young children's computational thinking skills can be developed when an appropriate curriculum for coding is used (Relkin et al., 2021).

2.5 Computer science and computational thinking in the Turkish curriculum

With the decision of the Board of Education and Discipline, in 1997, the "Computer" course was added to the curriculum as an elective course in primary schools starting in the 4th grade (Tebliğler Dergisi, 1997). In 2005, the Board of Education and Discipline expanded the course for grades 1-8 and the choice of the course was left to the teachers' board. In 2007, it was updated as an "Information Technologies" course, as 2 hours per week in the 4th and 5th grades and 1 hour in other grade groups (Tebliğler Dergisi, 2007). In 2010, the time allocated for the course was lowered to 1 hour per week in grades 6-8, and the course was removed from other grade levels. The name of the course was changed again in 2012 to "Information Technologies and Software," and turned into a 2-hour weekly elective course in

grades 5-8 (Tebliğler Dergisi, 2012). After the update in 2012, the focus was on "problem solving and programming", and it also aimed to provide students with knowledge, skills and attitudes about IT literacy and digital citizenship. In other words, Computer Science was also included in an "Information Technologies and Software" course (Gülbahar & Kalelioğlu, 2018). With further changes in 2013, the course was accepted as a compulsory course in the 5th and 6th grades, and as an elective course in the 7th and 8th grades (Tebliğler Dergisi, 2013).

By the year 2016, a commission consisting of academics and teachers was established within the Ministry of National Education to update the “Information Technologies and Software” course. There was a need to revise the knowledge and skills covered in the curriculum, determine the structure, content and functionality of the curriculum, and develop appropriate teaching materials, and means for assessing learning gains. After a review of the literature, and interviews with experts, it was decided that the Information Technologies and Software course focus on “Computational Thinking”, “Information and Communication Technologies” and “Computer Science,” and be taught starting in the first grade through 10th grade (Gülbahar & Kalelioğlu, 2018).

2.6 Summary

The first formal CT definition was proposed by Wing (2006) as a way of thinking that everyone can and should have (Wing, 2006). Many researchers carried out studies to create a definition of CT (Weese, 2017), and the components of CT were identified (Kalelioğlu et al., 2016). Selby and Woollard (2013) identified the most commonly used CT sub skills as algorithmic thinking, abstraction, decomposition, generalization and evaluation. Angeli et al. (2016) identified CT components as

algorithmic thinking, abstraction, decomposition, debugging, and generalization, and others have proposed similar components (Grover & Pea, 2013; Kalelioğlu et al., 2016; Relkin et al, 2021).

As there is no one single definition of computational thinking, approaches to how to teach and develop CT skills are also varied. Unplugged computing activities and block-based programming tools have been used to improve CT skills. Researchers showed that fundamental principles and steps of computing processes can be taught through unplugged computing activities, and students' CT skills can be developed (e.g. Battal et al., 2021; Brackmann et al., 2017; Delal & Oner, 2020; Faber et al., 2017; Kalelioğlu, 2018; Rodriguez, Rader, & Camp, 2016). These skills can be transferred while learning programming in the future (Gaio, 2018). Additionally, Bell and colleagues claimed that students may focus more deeply on the concepts by disposing of computers (Bell & Vahrenhold, 2018; Bell et al., 2009).

It has been demonstrated that block-based programming platforms were also effective in teaching programming and CT skills to younger students (Amnouychokanant, Boonlue, Chuathong, & Tamwipat, 2021; Lye & Koh, 2014; Román-González, Pérez-González & Jiménez-Fernández, 2016; Shute, Sun, & Asbell-Clarke, 2017). Researchers argued that Scratch, one of the most commonly used block-based programming tools (Zhang & Nouri, 2019), can also help enhance creative thinking skills, collaborative working skills, computational and mathematical skills, and reasoning skills, namely the essential skills of the 21st century (Resnick et al., 2009). Findings showed that CT curriculum based on Scratch activities can help students to analyze given problems deeply and generate sufficient solutions, and enhance students' CT skills, such as algorithmic thinking, debugging,

and problem solving (e.g., Çatlak, Tekdal, & Baz, 2015; Grover et al., 2015; Kalelioğlu & Gülbahar, 2014; Oluk, Korkmaz, & Oluk, 2018; Wong & Jiang, 2019).

Assessing CT skills has been an important aspect of research and teaching in CT (e.g. Bocconi et al., 2021; Grover et al., 2015). Assessment methods have been developed and used to measure enhancement of CT skills. These methods generally focus on measuring students' knowledge through achievement tests, their final products and examining their perception of their own CT skills through Likert-scale surveys. Bebras international challenge is one of the assessment methods commonly used to measure students' CT skills (Gülbahar et al., 2020). The tasks in the competition consist of three difficulty levels and address 5 sub-dimensions of CT skills, such as abstraction, decomposition, evaluation, generalization, and algorithmic thinking (Kalelioğlu et al., 2021).

The integration of CT skills and Computer Science into formal education was recommended to develop students' CT skills and help students to understand computer science at an early age (Bers, 2018; Bers & Sullivan, 2019; Papadakis et al., 2016; Relkin et al., 2020). In Turkey, although the "Computer" course was added to the curriculum as an elective course in primary schools starting in the 4th grade in 1997 (Tebliğler Dergisi, 1997), the focus was on "problem solving and programming" after an update in 2012. By the year 2016, the Information Technologies and Software course started to focus on "Computational Thinking", "Information and Communication Technologies" and "Computer Science," and be taught starting in the first grade through 10th grade in 2016 (Gülbahar & Kalelioğlu, 2018).

Strategies have been put forward for how to best teach CT at different levels of schooling (Barefoot, 2014; Code.org, 2022; Teaching Computing, 2020). Barefoot

(2014) also provided five main activities, tinkering, creating, debugging, persevering, and collaborating, to teach computational concepts effectively. The Code.org curriculum gives an opportunity for students to develop their CT skills by thinking like computer scientists (Code.org, 2022). The National Center for Computing Education provided 12 strategies to teach computing effectively, which are adopted in this study, too.

The aim of this study was to design, develop, and evaluate a learning module that combines unplugged computing and block-based activities to improve CT skills, and arouse interest in CS, especially for children from socioeconomically disadvantaged backgrounds. The strategies suggested by the National Center for Computing Education were used in this study as guidelines to prepare lesson plans and in the training provided to the tutors who carried out the lesson plans. The Self-Efficacy Perception Scale for Computational Thinking Skill developed by Gülbahar et al. (2018), and a CT skills test mainly adapted from Bebras tasks, and final projects developed by the students were used as assessment instruments. This study aimed to answer the following research questions:

i. What are the effects of unplugged and block-based programming activities delivered online by mentors on the computational thinking (CT) skills of middle school students?
    - Is there a statistically significant difference in the middle school students' general CT scores before and after they participated in the remote modules?
    - Is there a statistically significant difference in the middle school students' perceptions of their CT self-efficacy before and after they participate in the remote module?

ii. What kind of CT learning occurred as a result of the children’s participation in the program, based on their final products?

iii. What are the learners' evaluation of the online CT module delivered remotely, based on their feedback?

iv. According to the mentors, how effective was the CT module, regarding the unplugged and block-based activities, and what design improvements are needed?

CHAPTER 3

METHOD

In this chapter, research design, details about participants, the process of content development and data collection tools, data scoring and finally data analysis are covered.

3.1 Research design

In this study, both quantitative and qualitative data were collected to answer the research questions. This mixed-method triangulation research adopted the data transformation model, where quantitative and qualitative data are collected separately, and after an initial analysis, the qualitative data are quantified (Creswell, 2006; Creswell & Plano Clark, 2011).

The quantitative component was designed as a pre-experiment, with a single group pretest posttest design. The independent variable was an 8-week learning module developed by the researcher consisting of unplugged computing activities and block-based programming designed to develop the computational thinking skills of 6th grade students. The dependent variables were the students' computational thinking skills and their perception of self-efficacy of CT.

The qualitative component consisted of data collected through interviews, open ended and Likert-type questionnaires, as well as a rubric-based assessment of student final projects. Another purpose of collecting qualitative data was to answer the secondary research question about evaluation of the instructional design delivered online through video-conferencing.

In this research, the researcher's role was threefold: designing the CT learning module, providing training for the mentors, and conducting the research.

3.2 Participants

The participants of the study were 6th grade students from socio economically disadvantaged backgrounds who received small group or one-on-one online tutoring from a group of volunteers from the Okul Destek Derneği (School Support Foundation). The purpose of the School Support Foundation (SSF) is to provide online academic support to elementary and secondary students from low SES backgrounds, especially to make up for the learning losses due to the school closures during the Covid-19 pandemic. The participants of this study were students who enrolled in the volunteer summer school on computational thinking. All of the participants were enrolled in public schools in 20 different provinces of Turkey (see in Table 2). These students followed online courses for the first time during the COVID-19 pandemic.

Table 2. Information About the Students' Location

Province	Number of the Students	Percentage	Number of Students who Took ICT		
			First Semester	Second Semester	Both Semesters
İstanbul	25	40.3%	1	3	11
Ankara	9	14.5%			2
Gaziantep	3	4.8%			2
İzmir	4	4.8%			4
Kocaeli	3	4.8%			3
Antalya	2	3.2%			2
Mersin	2	3.2%			1
Muğla	2	3.2%	1		1
Other (Adana, Aydın, Balıkesir, Bursa, Kahramanmaraş, Karabük, Kilis, Malatya, Mardin, Sakarya, Sivas, Tekirdağ)	12 (1 from each)	1.6%			7
Total	62		2	3	33

While 27 students (43.5%) did not take ICT courses at school during the 1st semester, 22 (35.4%) of the 62 students took ICT courses in the 1st semester, and 13 students (20.9%) took the courses for a few weeks. In the second semester, 27 students (43.5%) did not take the ICT courses in the 2nd semester. Additionally, 21 students (33.8%) who took the ICT courses in the first semester took the lesson and 14 students (22.5%) took the courses for a few weeks.

Table 3. Parents' Education Background

Graduation Degree	Fathers	Mothers
Master's degree	5%	6.4%
University	35.4%	38.7%
High school	40.3%	25.8%
Primary school	10%	29%

A typical father in this group graduated from high school, and a typical mother was a college graduate: 40% of fathers graduated from high school and 39% of mothers graduated from college. The second most common school graduation level for mothers was primary school (29%), while for fathers it was college (35.4%). In addition, 25.8% of mothers graduated from high school and 10% of fathers graduated from primary school. The least common education level was a master's degree; 6.4% of mothers and 5% of fathers held a master's degree.

As for the parents' profession, the majority of the mothers were home-makers (43 out of 62), and the majority of the fathers were self-employed (14 out of 62). The second most common employed was the health sector; 5 of the fathers and 4 of the mothers were health care professionals. There were also civil workers, police officers, and engineers among fathers (4 from each). The rest of the professions varied from plummer, cook, and teacher to logistics manager.

Although 86 students attended the online classes throughout the program, the number of students who completed the program was 62. Therefore, the data from 62 students were included in this study.

3.2.1 Mentors

The lessons were given by 10 mentors who were volunteers recruited by the WTech Foundation, Turkey. While 2 of the mentors were senior students, 8 were college graduates, with undergraduate degrees in engineering, computer science, or teaching. All had an interest in computer science, regardless of their field of study, and many of them had either a private tutoring background or teaching experience.

Table 4. Information About the Mentors

Mentor ID	Profession	Undergraduate Degree	Graduation Year	Teaching Experience	Relation to Computer Science
1	Business analyst	Business	2019	no	Personal interest
2	Mind games designer; Robotics coding instructor	Mathematics	2001	tutoring	Personal interest
3	RPA developer	Electrical electronics engineering	2020	tutoring	Graduate of the field
4	Cyber Security Researcher	Information systems engineering	2021	tutoring	Graduate of the field
5	ICT Teacher	Computer Education & Educational Technology	2019	school teacher	Graduate of the field
6	Computer engineer	Senior student in Computer engineering		no	Prospective graduate
7	Computer engineer	Computer engineering	2018	tutoring	Graduate of the field
8	Business analyst	Industrial engineering	2019	tutoring	Graduate of the field
9	Business analyst	Economics	2016	volunteer teacher	Graduate of the field
10	Cyber security intern	Senior student in Information Systems Engineering	-	no	Prospective graduate

The mentors were provided with 5 hours of training delivered online via a video-conferencing tool, and an instructor's guide with lesson plans. In the first week, 2 hours training about the purpose of the implementation was given by the researcher, the pedagogical approach behind it, and the content of the unplugged programming activities, and a demo lesson was held. In the second week, a 2-hour Scratch training was given by the researcher. Before coming to this training, mentors were asked to familiarize themselves with the Scratch program and note down their questions. During this session, anticipated problems and possible solutions, and

questions children may ask were also discussed. In the last week, a 1-hour training session was held by the researcher and an expert in the field to explain the approaches necessary to teach computational thinking skills to students.

In both the unplugged and Scratch training sessions, sample lessons were taught by the researcher where the mentors assumed the role of the students, so that they were able to see how the lesson plans can be implemented online through video-conferencing. The instructor's guides included sections such as how lesson plans should be handled, what important points should be emphasized, and the triggering questions mentors can ask. In this way, the mentors who did not have sufficient knowledge about the subject were able to prepare for the lessons by adhering to these guidelines and by examining the lesson slides prepared for them.

3.3 Content development

The 8-week CT learning module developed by the researcher started with unplugged computing activities and continued on the Scratch platform at the end of the first 4 weeks. It consisted of activities that 6th grade students can implement both at home and by remote access under directional guidance. The majority of the activities were selected and adapted from the content of the “Information Technologies and Software” (ICT) course in the 5th and 6th grade curriculum.

The underlying theory of learning in the design process was constructionism, as defined by Papert (1980) and promoted by Bers (2020). Activities were prepared by using contexts and materials of daily life (e.g. kitchenware commonly found in the home), as well as topics that could be of interest to children (e.g. brushing teeth in a spacecraft) so that the students got opportunities to learn by using the materials around them. These activities aimed to support students to learn about computer

science concepts, as suggested by Bers (2020) and Grover et al. (2015), while solving situations related to daily life using their prior knowledge. and exploring information to construct their own knowledge.

While preparing the content, the pedagogies recommended by Teaching Computing (2020) were taken into consideration. The module was designed to support student learning by making inferences from the information provided or by researching the desired information. For example, in the “maze in the house” activity, students actually learn to develop algorithms by inferring from the questions asked or, in the "message in the mosaics" activity, after researching the word “pixel”, students discover a picture by coding with pixels. In addition, the activities were prepared in such a way as to enable the students to find common answers to the questions asked by sharing their thoughts, as in group work, and by doing kinesthetic activities. For example, in the "Find the rule of the game, start coding" activity, students sequenced glasses they could find in their homes in various ways following the rules given.

Several activities were based on the principle of "unplug, unpack, repack," a recommended computing pedagogy (Teaching Computing, 2020). First, new information or term was posed, then this new information was associated with what the students are likely to know in daily life, and then they were asked to provide their own examples. Since it was difficult to understand abstract concepts, concepts related to programming and computer science were brought concrete through examples from daily life. At the same time, reinforcement activities with different difficulty levels and scaffolding were held. Finally, the students worked on a final project with the information they learned in unplugged computing activities and Scratch lessons.

The unplugged programming activities and block-based programming activities contained the same sub dimensions of Computational Thinking to support skills development during both phases. Recommendations of Grover et al. (2015) on computer science education for secondary schools were followed. Nine unplugged programming activities were prepared based on the 5th Grade Teacher's Guide published by the Ministry of National Education, as well as web resources such as CS Unplugged (https://csunplugged.org) and Teaching London Computing (https://teachinglondoncomputing.org). The activities in these sources were adapted carefully to be associated with the daily lives of the students and for live delivery via video conferencing tools. The activities based on the teacher's guide by the Ministry of National Education were rearranged to support students' thinking about computer science. The unplugged CS activities developed and adapted by the researcher are listed in Table 5, with information about its source. A sample activity can be found in Appendix S.

Table 5. Activities and Sources

Activities	Researcher developed?	Sources
Maze in the house	yes	
Secret message	partial adaptation from:	Create Your Own Code (info.thinkfun.com)
Caesar Cipher		
Line it up and complete it!	adaptation from:	Squeezing pictures into codes (csunplugged.org)
Message in the mosaics	partial adaptation from:	Pixel Puzzles (teachinglondoncomputing.org)
Surprise question	adaptation from.	Roman Mosaic (teachinglondoncomputing.org)
Find the rule of the game, start coding	partial adaptation from:	Milli Eğitim Bakanlığı (2018). Bilişim teknolojileri ve yazılım dersi öğretim programı (ortaokul 5. ve 6. sınıflar)
Bug hunting	yes	
Pattern in the tile		
Program the cat	Partial adaptation from:	
Exploration Hunt		
Create an animation		
Save the chicklet		
Talking Objects		
Be Careful		
I'm coding the game		

In addition to the unplugged activities, 7 block-based programming activities were created by considering the Scratch lesson plans in 6th grade Teacher's Guide published by the Ministry of National Education in 2018. The design of the Scratch activities followed the order of the code blocks introduced in this Teacher's Guide. The activities were organized through question posing to support students' thinking on computer science. At the same time, the 7 activities were designed so that they made sense when brought together as a whole, unlike those in the 2018 curriculum or the web resources available to teachers. Table 6 below shows which unplugged and

block-based programming activities support which computational thinking skills, presented in the order of the lesson plans provided to the mentors to follow.

Table 6. Unplugged Computing and Scratch Activities, and CT Skills Addressed in Each

Activities	Abstraction	Algorithmic Thinking	Decomposition	Debugging	Generalization
Maze in the house		X		X	
Secret message		X			
Caesar Cipher	X				
Line it up and complete it!			X		
Message in the mosaics			X		
Surprise question					
Find the rule of the game, start coding	X				X
Bug hunting				X	
Pattern in the tiles					X
Program the cat	X	X			
Exploration hunt	X		X		
Create an animation		X		X	
Save the chicklet		X			
Talking Objects				X	
Be Careful		X	X		X
I'm coding the game					X

The purpose of the unplugged CS activities was to improve students' computational thinking skills by playing games about coding on paper or physically in front of the computer during video-conferencing. The activities were designed to increase the interest and curiosity of the students on computer science while

supporting their computational thinking skills. Each of the 9 unplugged lessons (Appendix P) was built on and supported the previous week's activity, and hence made sense when brought together.

The block-based coding activities (Scratch) (Appendix R) were designed in parallel with the unplugged programming activities and were intended to follow up and support the CT skills addressed in the unplugged activities. In the following Table 7 the unplugged activities are matched with the relevant block-based programming activities.

Table 7. Unplugged Computing & Scratch Activities and, Matching CS Concepts

Unplugged Computing Activities	Learning outcomes	CS Concepts	Scratch Activities	Learning outcomes	CS Concepts
Maze in the house	Write a daily life algorithm Develop students' algorithmic thinking skills.	Sequences	Program the cat	Recognize the interface and features of the Scratch programming tool. Explain the functions of the presented code blocks in the Scratch.	Events
Caesar Cipher	Develop students' abstraction skills		Exploration Hunt	Explain the functions of a program presented in the Scratch. Develop and organize a project in Scratch according to given criteria.	Sequences
Secret message,	Develop students' algorithmic thinking skills Solve an algorithm that include decision making	Conditions Sequences	Create an animation	Debug a program presented in Scratch. Develop and organize a desired program in Scratch according to the given criteria.	Sequences, Loops
Pattern in the tile	Develop students' generalization skills Solve a problem that include loop structure	Sequences, Loops			
Message in the mosaics,	Develop students' decomposition skills Solve a given algorithm that include decision making	Conditions	Save the chicklet	Develop and organize a desired program in Scratch according to the given criteria.	Events
Line it up	Develop students'	Sequences			

and complete it!	algorithmic thinking and decomposition skills Write a daily life algorithm Gives daily life algorithm examples				
Bug hunting,	Develop students' debugging skills Write a daily life algorithm Solve a problem that include loop structure	Sequences, Loops	Talking Objects	Create programs that contain logic structure. Test and debug programs that contain logic structure.	Sequences
Message in the mosaics	Develop students' decomposition skills Solve a given algorithm that include decision making	Conditions			
Message in the mosaics,	Develop students' decomposition skills Solve a given algorithm that include decision making	Conditions	Be Careful	Create programs that include decision structure. Create programs with multiple decision structures.	Conditions, Loops
Secret message	Develop students' algorithmic thinking skills Solve an algorithm that include decision making	Conditions Sequences			
Find the rule of the game, start coding	Develop students' abstraction and generalization skills Write an algorithm that include decision making	Conditions	I'm coding the game	Select the most appropriate decision structures to adapt an algorithm.	

The purpose of designing matching unplugged with plugged activities was for students to be able to transfer the skills they learned in unplugged computing to block-based programming. For example, students developed algorithmic thinking skills in the "Secret Message" unplugged programming activity, and in the Scratch lesson they began to learn the if / then condition in coding. Later they needed to use the condition commands they had learned while doing the "Be careful" and "I'm coding the game" activities in the Scratch program.

3.4 Implementation

SSF Coding Summer School on computational thinking was a 2-month program carried out in the summer of 2021 (July 1 – August 31), after a school year dominated by the Covid-19 pandemic, infested with school closings and quarantines around the world. The implementation took 8 weeks, and the lessons were delivered through video-conferencing via Zoom, 2 hours per week, with a total of 16 lesson hours. Each mentor worked with a group of 8-10 students. It was ensured that the students who enrolled in the block-based coding sessions had access to technological devices such as tablets or computers. For 6 students who used parent's phones to participate in the program with no access to a computer or tablet, further unplugged activities were developed and carried out by the researcher so that they completed the same number of hours of computing, albeit unplugged. No data was collected from these students.

Google Classroom was used to share activity files with students and mentors. A class folder was opened on Google Classroom for each mentor. Instructor lesson plans and presentations that they can use in the lessons were shared with mentors in a separate folder. In addition, each student had their own folder, and the activity pages of that week were shared in these folders with the students.

After 4 weeks of 8 unplugged computing activities, Scratch lessons were started. In the last week of Scratch lessons, students created their own Scratch games and presented it. Before launching on a Scratch project, the mentors dedicated a session to the discussion of game design, such as the goal and rules of the game, and the students came up with game ideas and shared them with the group. While creating their games, the mentors answered questions and helped them throughout the lesson. The students who could not complete their projects due to limited class

hours continued to do their projects at home, and in the last lesson, each student presented their project by share screen.

3.5 Data collection

3.5.1 Procedures

Before the CT module was implemented by the mentors of the SSP, several of the activities were piloted with 3-4 children, and necessary changes were made.

Before the research was conducted, approval was obtained from the Institutional Review Board and the Ethics Committee of the University. Consent forms were signed by the students' parents and the mentors.

At the beginning of the implementation at the SSF, the students and mentors were given a demographic questionnaire to fill out. Computational thinking self-efficacy perception scale, and computational thinking skill tests were given as pre-test to students. The students took the computational thinking skill test as a post-test and were also asked to fill out the self-efficacy scale again, as well as a feedback questionnaire about the module. The mentors were given a feedback questionnaire, and then interviewed. Finally, the projects developed by the students were collected for further coding and analysis.

Table 8. The Data Collection Process

Pre-test	Implementation	Post-test	Interview/ Questionnaire
1st week 1st Lesson	Weeks 1-7	8th week	8th week
CT skill test	Unplugged computing and block-based computing activities	CT skill test	Feedback questionnaire
Self-Efficacy Perception Scale for Computational Thinking Skills		Self-Efficacy Perception Scale for Computational Thinking Skills	Interview questions

3.5.2 Instruments

The data collection instruments were demographic questionnaires, feedback questionnaires, interview questions for the mentors, CT skills test, CT self-efficacy perception scale, and the students' final projects.

Table 9. Data Collection Instruments Throughout the Data Collection Phase

Instruments	Students	Mentor	Pre-Intervention	Post-Intervention
Demographic Questionnaire_S	✓		✓	
Demographic Questionnaire_M		✓	✓	
Self-Efficacy Perception Scale for Computational Thinking Skills (Bilgi Işlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı Ölçeği (BİDBÖA))	✓		✓	✓
CT skill test	✓		✓	✓
Feedback questionnaire_S	✓			✓
Feedback questionnaire_M		✓		✓
Final projects	✓			✓
Interviews		✓		✓

Before starting the implementation, both mentors and students were asked to fill out a demographic questionnaire in order to obtain information about the participants. This form contained different questions for mentors and students.

A computational thinking skill test was given to the students as a pre and post test to collect data about their CT skills before and after the implementation. The test was developed by the researcher based on Grover et al. (2015) and Gülbahar et al. (2018). It consisted of 15 questions with 3 difficulty levels; 12 tasks were selected from the Bebras tasks, and 3 were created by the researcher for block-based programming skills. In the selection of the Bebras tasks, the computational thinking skills to be evaluated were taken into consideration. In the block-based questions section, questions were prepared using code.org visuals, similar to the Scratch program.

Bebras competition questions were prepared by experts in different countries and were translated into Turkish and held as a large competition event named “Bilge Kunduz " in Turkey. The CT skills test items were checked by an expert, and revisions were made based on the feedback. After the test was administered with students, its reliability was calculated, and the Cronbach alpha value was .583. The low Cronbach alpha value is consistent with the findings from an assessment of Bebras challenges using data from 113,653 students in Turkey in 2019 (Kalelioğlu et al., 2021).

The names of the Bebras challenges used in the CT test, their difficulty levels, the CT sub skills engaged, and Bebras challenge years are shown in Table 10.

Table 10. Pretest and Posttest

Task Number	Task Name	Difficulty Level	CT Component Covered	Bebras Challenge Year
1	Five Sticks	easy	Algorithmic thinking, Decomposition, Abstraction, Debugging	2017
2	Theater Performance	easy	Abstraction, Algorithmic thinking, Decomposition	2020
3	Scratch Art Paper	easy	Debugging, Generalization, Decomposition	2019
4	Message Service	easy	Abstraction, Generalization	2017
5	Squirrel and Acorn	easy	Algorithmic Thinking, Abstraction, Debugging	-
6	Choose a Way	medium	Algorithmic thinking	2020
7	Processing Objects	medium	Debugging, Algorithmic thinking, Decomposition	2020
8	Birthday Cake	medium	Algorithmic thinking, Decomposition	2019
9	Chain	medium	Algorithmic thinking, Generalization, Abstraction, Decomposition	2016
10	Robot BB-8	medium	Algorithmic Thinking, Abstraction, Generalization	-
11	Jigsaw Puzzle	hard	Decomposition	2020
12	Give Me a Smile	hard	Decomposition, Generalization, Algorithmic thinking	2017
13	Mistakes	hard	Debugging, Algorithmic Thinking	2015
14	Infinite Ice-Cream	hard	Algorithmic Thinking, Decomposition, Generalization	2018
15	Fish Game	hard	Algorithmic thinking, Abstraction, Decomposition	-

In addition to the CT achievement test, the final projects prepared by the students during the last week of the implementation were collected and analyzed for a holistic assessment of student learning, as recommended by Grover et al (2015).

In pre and post testing, the students filled out a computational thinking Self-Efficacy Perception Scale for Computational Thinking Skills (Bilgi Işlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı Ölçeği—BİDBÖA) developed by

Gülbahar et al. (2018). The scale includes 36 items with five-factors and has “yes”, “partially” and “no" options. Students marked the most appropriate option for each sentence. Cronbach Alpha coefficients were between 0.762 and 0.930 (Gülbahar et al., 2018). The Cronbach Alpha coefficients of the Self-Efficacy Perception Scale for Computational Thinking Skill (CTSSP) that was used in this study were calculated after the implementation and it was .963 with a total of 36 questions. In addition, the Cronbach’s alpha for all sub factors (i.e., algorithm design, problem solving, data processing, basic programming, and self-confidence) were calculated (see Table 11).

Table 11. Cronbach’s alpha value of the 5 Factors in SPSCTS

Factors	Number of Items	Cronbach’s Alpha Value
Efficacy for Algorithm Design	9	.936
Efficacy for Problem Solving	10	.917
Efficacy for Data Processing	7	.909
Efficacy for Basic Programming	5	.891
Efficacy for Self-Confidence	5	.784
Total	36	.963

The students and the mentors were asked to fill out a feedback questionnaire after the module was completed to receive feedback about instructional design and implementation via video-conferencing. The feedback questionnaire for the students consisted of the questions about the activities they liked the most, what they learned, the difficulties they experienced, their suggestions for the lesson and 7-question Likert scale to evaluate lessons and activities. The feedback questionnaire for the mentors contained specific open-ended questions about the module and also included a 7-item scale.

Finally, interviews were conducted with the mentors to get more detailed feedback from them about the implementation and students learning. Although the

study was carried out with a total of 10 mentors, only 8 mentors were available for an interview.

3.6 Data scoring and analysis

3.6.1 Fidelity check

Although all 10 mentors were supplied with the same training and the same instructor's guide and lesson plans, fidelity of implementation was a concern since their experience and areas of expertise varied. An ID was assigned to each of the mentors to use in the analysis, and the automatically recorded sessions (about 60%) were observed to see the extent to which they followed the instructor's guides and adopted the recommended pedagogy for the delivery of the lessons to teach computational thinking skills.

During this observation, the lessons were assigned scores 1 or 2 depending on the level of fidelity of implementation. Thus, two groups of mentors emerged were identified to test whether the lessons were delivered as planned. Then the pretest and posttest differences of the students taught by the mentors in these 2 groups were compared. Although the majority of mentors did not come from teaching backgrounds, there was no difference between the test results of the 2 groups of students. This might be due to the training provided at the beginning, the instructor's guides, and the ongoing support. Since there was no difference in the 2 mentor groups, the data of all students in all the mentors' groups were included in the same analysis.

3.6.2 Pretest and posttest scores in the CT achievement test

Data from the CT test were coded in SPSS 27 to calculate the students' pretest and posttest scores. In this table, the student names were hidden, and an ID was assigned to each student. In order to distinguish the instructors of the students, an ID number from 1 to 10 was assigned to each instructor in the data table.

The scoring method used in Bebras tasks was used. As recommended by Bebras Challenges (bilgekunduz.org), the Easy questions that were marked "Easy" were scored 6 points. Medium difficulty questions were scored 9 points and "Difficult" questions scored 12 points. The maximum score that could be achieved was 135. As with the scoring in the Bebras tasks, no points were deducted for wrong answers.

Students' answers to each question in the pretest and posttest exams were tabulated, and the total scores for the pre and post tests were calculated for each student, as well as the difference between the posttest and pretest results.

Finally, a paired samples t-test was conducted in SPSS 27 to determine whether there was a significant difference between the students' pretest and posttest scores in the CT test.

3.6.3 Self-efficacy scale

The pre and post test scores from the self-efficacy scale were also coded in SPSS 27. The answers were entered 3 for "yes", 2 for "partially", and 1 for "no," Negative sentences were reverse coded: 1 point for "yes," and 2 points for "partially," and 3 points for "no," Then total scores for each student were calculated for the pre and post test. In addition, the total scores for each factor were calculated for the pre and posttest.

The non-parametric Wilcoxon signed-rank test was used to analyze the effects of unplugged and block-based programming activities on students' perceptions since the data was not normally distributed.

3.6.4 Students' Scratch projects

All students were expected to create a final project at the end of the Scratch module. However, it was possible to collect and access 20 final projects. The coding and analysis reported in this study is therefore based on 20 students' projects.

The three competencies identified by Denner et al. (2012) based on a literature review were used to develop rubrics for the assessment of the projects. The students' final Scratch projects were assessed using an adapted version of Denner et al. (2012)'s rubric. The assessment was done in 2 stages, first the code blocks used, and then the usefulness of design accordingly. In the first step, a rubric for programming was used to score the code blocks used in each project. In the second step, a rubric for "designing for usability" was used to score 5 features.

The rubrics were based on the coding categories and definitions in Denner et al. (2012). For example, the variable test coding category allowed checking for whether the "if else" code command was used. All code commands used by the students were listed to examine the students' projects in more detail. These commands were grouped according to the appropriate code categories that Denner et al. (2012) found as a result of their work, and the names of the code blocks in Scratch. Likewise, the projects were also evaluated according to the "design for usability" in Denner et al. (2012). The features unrelated to Scratch platform were not used.

In the rubric for programming (see Table 12), the projects were examined according to the learning outcomes in the Scratch lessons. It was first examined whether they used the commands they had learned during the lessons. According to their learning outcomes, there were 23 code blocks the students learned in the Scratch module that they could use in their projects. Each code block was scored 1 point. The projects lost 1 point for each code block added to the code blocks section but never used.

The programming rubric consisted of criteria such as motion, character variable use, speech, background scene, sound use, conditions or events use, variable test, control, global variable use, operators, sensing, and music. Table 12 presents information about these criteria and the code blocks they are found in.

Table 12. Rubric for Programming Based on the Code Blocks Covered and not Covered in the Lesson Plans

Criteria	Functions	Code Blocks Covered	Code Blocks not Covered
Motion	The character moves in accordance with the story. (e.g., when the right arrow key is pressed, the character moves to the right)	move …. steps turn …. degrees point in direction …. if on edge, bounce set rotation style ……	point towards mouse pointer go to x: …. y…. change x by …. set x to … change y by…
Character Variable Use	Information about individual characters (e.g., change in appearance) is used in game (either changed and/or displayed). A new definition is given to a variable changed the default initialization of a character	switch costume to ……. next costume show/hide	change size by 10 set size to 100%
Speech		say Hello for 2 seconds say Hello think…. for … seconds think …….	
Background Scene	Changes about the scene	switch backdrop to ……. next backdrop	
Sound Use	Sound is used appropriately for its function (e.g., sound of rain when	play sound … until done	

	it rains)		
Conditions or Events Use	Player can make something happen by interacting with the program	when … clicked/ when …. key pressed/ when this sprite clicked when I receive "message1"	broadcast "message"
Variable Test	An "and-if test" is used on a variable in the game (e.g., more than one condition must be in place for a rule to fire).	if…then / if…. then else	
Control	Student recognizes and applies "control" commands in Scratch	wait … seconds repeat …. forever	when I start as a clone create clone of myself stop all
Sensing	Interaction of the character by the characters on the screen, the characteristics of the characters, or the user gestures	color… is touching ….? touching ….	ask ……. and wait answer key …. pressed? mouse x
Global Variable Use	Uses information that affects every stage and every character worldwide (e.g., score variable)		my variable set my variable to 0 change my variable by 1
Operators	Providing operational checks on the screen (e.g., whether two values are equal or performing mathematical operations such as addition)		pick random 1 to 10 ….. + ……
Music	Musical instrument commands are used in accordance with their function (e.g., hearing the "re" sound when pressing the re note on the piano)		play note … for … beats

Apart from these, there were other code blocks the students used in their projects, though not covered in the Scratch module, such as movement, size, sensing music, control, and variable code blocks. These code blocks are given in a separate column in Table 12. Each extra code block scored 2 points. The maximum point that projects could get from the code blocks learned in the module was 26. The maximum point that projects could get from code blocks that were not covered in the module was 42.

The criteria in the rubric for "designing for usability" were incorporate themes, character appearance changes, goal (how to win-lose) and functionality,

based on Denner et al. (2012). Projects that had clear instructions at the beginning scene received bonus points. Points were given based on whether the students used these criteria fully, partially or none in their projects. The projects received 2 points if the criterion was fully implemented, 1 point if the criterion was partially implemented, and 0 points if the criterion was not implemented at all. For example, if the students used a theme aligned with the purpose of their projects, they received 2 points. If they used a theme that was not relevant for the purpose of their projects, they scored 1 point. If the goal of the game and the theme were not related to each other, no points were given. To illustrate, if a student designed a maze game and there was a maze, a character, and an endpoint in the project, the project received 2 points. If there were no ending or starting point in the maze, i.e., the theme did not fully match the maze game, it received 1 point, and no elements appropriate for a maze game, scored 0 points. Likewise, if the character's appearance was changed in accordance with the theme of the game, the project received 2 points, but if the character's new appearance did not match the project's theme, it received 1 point. If the projects had winning and losing, the projects got 2 points. If there was only winning or only losing, the projects received 1 point. If losing or winning were not used in the game, the project received 0 points.

Although 19 of the 20 students wrote the instructions on paper, they did not include these in their projects. If the students' projects fulfilled all the purpose-oriented functions, their projects received 2 points. However, if some features were not working in the project, projects got 1 point, and if none of the features worked, projects got 0 points.

Table 13. Rubric for Designing for Usability

Criteria	Explanations
Incorporates Themes	The given theme is important and incorporated into the game (e.g., astrobiology, adventure, trivia).
Character appearance changes	Characters change their appearance during game play (e.g., flap wings, change direction).
Instructions clear	There are clear instructions for playing the game and rules for user input.
Goal (how to win-lose) clear	There are clear instructions on how a player wins and/or loses the game.
Functionality	There are few or no defects in the programming.

Finally, the scores received from two rubrics were added up and the final score was calculated for each project.

3.6.5 Feedback data

Separate thematic analyses were conducted for the students' and the mentors' responses to the open-ended questions in each feedback questionnaire. The answers given by 62 students to the 4 open-ended questions (Appendix I) were collected in an Excel sheet. A table with 2 columns was created to group the answers given by the students. Starting from the first question, the students' answers were entered in the first column of the table in order. If there were students who gave the same answer, the number of students was entered in the column next to that answer. In this way, the same answers were tabulated with the number of students who gave this answer.

Then, the answers of the students were rearranged based on common themes and a matrix was created. A third column was added to indicate the percentage of students who gave the same answer.

The answers to the 5 Likert-scale questions were tallied. A table was created to score these responses, based on the 5 options, which were strongly agree, agree,

undecided, disagree and strongly disagree. Then, the number of students who gave that answer was note and the percentages were calculated.

As for the data from the mentor feedback questionnaire, the responses to the 6 open-ended questions were collected in a matrix. As in the student feedback data, all the answers were entered and gathered under relevant themes, based on the questions, and then tallied. A table was created for the 7 Likert scale questions with 5 options and tallied.

3.6.6 Mentor interview data

First, the mentors' responses to the 5 interview questions (Appendix E) were fully transcribed by the researcher. The interview data was then coded and analyzed using the Partially Ordered Meta-Matrix suggested by Miles and Huberman (1994, cited in Miles, Huberman, & Saldana 2016) for qualitative analysis.

The themes were based on the questions asked in the interview, such as effective activities, difficulties, suggestions for modules, advice to the instructors and students' development. Then, the common answers given by the mentors were identified.

CHAPTER 4

RESULTS

In this chapter, the results will be reported in the order of research questions. First, the results of the statistical analyses are presented for the students' scores on the CT test and the CT self-efficacy scale to answer the first research question. Then descriptive statistics and results from the scoring of the students' final projects are reported, and the thematic analyses of the students' and the mentors' responses to the feedback questionnaires as well as the data from the interviews are presented.

4.1 The effects of the unplugged and block-based programming activities delivered online

4.1.1 Research question 1a: Is there a statistically significant difference in the middle school students' general CT scores before and after they participate in the remote modules?

Before carrying out a parametric test, the assumption of normality and skewness and kurtosis values were checked. Kolmogorov-Smirnov's test for normality was carried out to check the normal distribution of the data, because the sample size was larger than 50. Based on the results of Kolmogorov-Smirnov's test, pretest and posttest scores were normally distributed (p >.05) (see table 14). The pre-test scores skewness was found .499 (SE = .304) and kurtosis was found -.325 (SE = .599). The skewness for post-test scores was -.188 (SE = .304) and kurtosis was -1.029 (SE = .599).

Table 14. Test of Normality of CT Scores

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
Pre CT Scores	.088	62	.200*
Post CT Scores	.092	62	.200*

The Kolmogorov-Smirnov test and the skewness and kurtosis values showed that both the pre and post test data met the normality assumption. Since the data were normally distributed, a paired samples t-test was conducted to analyze the effect of unplugged and block-based programming activities on CT scores.

Table 15. Descriptive Statistics for Students' CT Scores

	n	Mean	Std Deviation.	Std. Error Mean
Pre-Total	62	60.44	24.365	3.094
Post-Total	62	67.89	22.445	2.851

Paired samples t-test results showed that there was a significant difference between the students' pretest and posttest scores (see Table 15 for descriptive statistics). The students' CT scores significantly increased at the end of the CT module (M= 7.45, SD=24.340), p=.019. See Table 16.

Table 16. Paired Sample Test

Paired Differences				
		Mean	Std. Deviation	Sig. (2-tailed)
Pair 1	Post CT Score - Pre CT Scores	7.452	24.340	.019

4.1.2 Research question 1b: Is there a statistically significant difference in the middle school students' Self-Efficacy Perception Scale for Computational Thinking Skill (CTSSP) before and after they participate in the remote module?

To check for normality of the CTSSP scores, skewness and kurtosis values were calculated, and Kolmogorov-Smirnov's test for normality was carried out. Based on the results of Kolmogorov-Smirnov, the data from the survey were not normally distributed (p <.05) (see table 17). At pretest, the skewness was -.919 (SE = .304) and kurtosis .483 (SE = .599) for the whole scale, while at posttest skewness was -1.115 (SE = .304) and kurtosis.605 (SE = .599) for the whole scale. Although these values were within the acceptable range, the Kolmogorov-Smirnov test showed that the data did not meet the normality assumption.

Table 17. Test of Normality of Students' Perception Scale

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
Pre CT Self-efficacy perception scale results	.152	62	.001
Post CT Self-efficacy perception scale results	.175	62	.000
Pre Efficacy for Algorithm Design (Factor 1)	.139	62	.005
Post Efficacy for Algorithm Design (Factor 1)	.224	62	.000
Pre Efficacy for Problem Solving (Factor 2)	.200	62	.000
Post Efficacy for Problem Solving (Factor 2)	.276	62	.000
Pre Efficacy for Data Processing (Factor 3)	.173	62	.000
Post Efficacy for Data Processing (Factor 3)	.269	62	.000
Pre Efficacy for Basic Programming (Factor 4)	.160	62	.000
Post Efficacy for Basic Programming (Factor 4)	.218	62	.000
Pre Efficacy for Self-Confidence (Factor 5)	.185	62	.000
Post Efficacy for Self-Confidence (Factor 5)	.289	62	.000

The results of descriptive statistics showed that the mean of the total post test scores and sub factor scores were greater than the mean of total pre-test scores and sub factor scores. It can be stated that there was a significant difference between pre test and post test scores according to the total scores and sub factor scores (see Table 18).

Table 18. Descriptive Statistics for CTSSP Self-efficacy Scale

Factors	Tests	n	Mean	Std Deviation	Std. Error Mean
Whole Scale	Pre-Test	62	82.42	18.681	2.372
	Post-Test	62	99.45	9.147	1.162
Efficacy for Algorithm Design (Factor 1)	Pre-Test	62	17.91	6.260	.795
	Post-Test	62	24.88	2.788	.354
Efficacy for Problem Solving (Factor 2)	Pre-Test	62	25.62	5.198	.660
	Post-Test	62	28.06	2.475	.314
Efficacy for Data Processing (Factor 3)	Pre-Test	62	16.88	4.361	.553
	Post-Test	62	19.45	2.358	.299
Efficacy for Basic Programming (Factor 4)	Pre-Test	62	10.27	3.613	.458
	Post-Test	62	13.17	2.329	.295
Efficacy for Self-Confidence (Factor 5)	Pre-Test	62	11.70	2.870	.364
	Post-Test	62	13.87	1.408	.1788

Because the data was not normally distributed, the non-parametric Wilcoxon signed-rank test was used. The Wilcoxon signed-rank test demonstrated a positive improvement in total scores and sub factor scores of the students ($p < .05$) (see Table 19).

Table 19. Wilcoxon Signed Ranks Test for CTSSP Self-efficacy Scale

		N	Mean Rank	Sum of Ranks
Post CT Self-efficacy perception scale results - Pre CT Self-efficacy perception scale results	Negative Ranks	9[a]	12.00	108.00
	Positive Ranks	51[b]	33.76	1722.00
	Ties	2[c]		
	Total	62		
Post Efficacy for Algorithm Design – Pre-Efficacy for Algorithm Design	Negative Ranks	3[d]	5.83	17.50
	Positive Ranks	45[e]	30.29	1635.50
	Ties	5[f]		
	Total	62		
Post Efficacy for Problem Solving-Pre-Efficacy for Problem Solving	Negative Ranks	11[g]	20.32	223.50
	Positive Ranks	36[h]	25.13	904.50
	Ties	15[i]		
	Total	62		
Post Efficacy for Data Processing-Pre-Efficacy for Data Processing	Negative Ranks	14[j]	19.32	270.50
	Positive Ranks	39[k]	29.76	1160.50
	Ties	9[l]		
	Total	62		
Post Efficacy for Basic- Pre Efficacy for Basic	Negative Ranks	9[m]	19.72	177.50
	Positive Ranks	45[n]	29.06	1307.50
	Ties	8[o]		
	Total	62		
Post Efficacy for Self-Confidence-Pre Efficacy for Self-Confidence	Negative Ranks	11[p]	12.36	136.00
	Positive Ranks	41[q]	30.29	1242.00
	Ties	10[r]		
	Total	62		

Table 20. Test Statistics[a]

	Post CT Self-efficacy perception scale results - Pre CT-Self-efficacy perception scale results	Post Efficacy for Algorithm Design – Pre-Efficacy for Algorithm Design	Post Efficacy for Problem Solving- Pre-Efficacy for Problem Solving	Post Efficacy for Data Processing- Pre-Efficacy for Data Processing	Post Efficacy for Basic- Pre Efficacy for Basic	Post Efficacy for Self-Confidence - Pre Efficacy for Self-Confidence
Z	-5.942[b]	-6.432[b]	-3.613[b]	-3.950[b]	-4.876[b]	-5.072[b]
Asymp. Sig. (2-tailed)	.000	.000	.000	.000	.000	.000

Wilcoxon Signed Ranks Test
Based on negative ranks.

4.2 What kind of CT learning occurred as a result of the children's participation in the program, based on their final products?

Given the remote teaching conditions and limited access to student work, it was possible to collect final projects from 20 of the 62 students who participated in the online summer program. Therefore, the results reported in this section are based on 20 student projects.

Nine out of 20 students had not taken an ICT course in the first or second semester, and 3 students had attended the ICT course only for a few weeks, which means that 12 of the 20 students whose projects were examined were not very familiar with the content of the ICT classes in the curriculum.

Two types of analysis were done to answer this research question. First, the number and variety of code blocks used in the final projects were counted. Then the projects were evaluated based on the rubric adapted from Denner et al. (2012).

The number of students who used the code blocks covered in the CT module, and the most frequently also used code blocks were counted and percentages were calculated. The code blocks covered in the lessons and those not covered during the

modules but used by the students were tallied separately (see table 21 and table 22). Thus, the effect of the activities in the CT module on student learning was observed.

Table 21. Scratch Code Blocks Covered in the CT Module

Scratch Code Block	Number of Students Using the Code Block	Number of Students who took ICT Courses During the School Year			Scratch Activities	CT Skills Addressed in the Activity
		Yes	No	A few weeks		Algorithmic Thinking, Abstraction
When … clicked/ when … key pressed	20	8	9	3	Program the Cat Save the Chicklet	Algorithmic Thinking, Generalization, Decomposition
If…then / if…. then else	17	7	7	3	Be Careful	Algorithmic Thinking, Abstraction
Move … steps	13	4	7	2	Program the cat	Algorithmic Thinking, Debugging
Forever	11	3	5	3	Create an Animation	Algorithmic Thinking, Generalization, Decomposition
Color… is touching ….?	10	2	5	3	Be Careful	Algorithmic Thinking
Point in direction …	9	2	6	1	Save the chicklet	Decomposition, Abstraction
Show/hide	9	4	4	1	Exploration Hunt	Debugging
Wait … seconds	8	4	3	1	Talking Objects	Algorithmic Thinking, Abstraction
Say hello for 2 seconds /say hello	8	4	4	0	Program the cat	Debugging
Play sound… until done	6	2	3	1	Talking Objects	Debugging
Next backdrop	5	3	1	1	Talking Objects	Algorithmic Thinking, Debugging
Next costume	5	4	1	0	Create an Animation	Algorithmic Thinking, Debugging
If on edge, bounce	5	2	3	0	Create an Animation	Decomposition, Abstraction
Repeat ….	5	3	1	1	Exploration Hunt	Debugging
Switch backdrop to…	4	1	2	1	Talking Objects	Algorithmic Thinking, Generalization, Decomposition

When I receive “message1”	4	2	2	0	Be Careful	Algorithmic Thinking, Generalization, Decomposition
Broadcast “message”	4	2	2	0	Be Careful	Algorithmic Thinking, Debugging
Switch costume to…	3	1	1	1	Create an Animation	Decomposition, Abstraction
Think…for … seconds/think	2	0	2	0	Exploration Hunt	Algorithmic Thinking, Debugging
Set rotation style…	2	1	1	0	Create an Animation	Decomposition, Abstraction
Turn … degrees	1	0	1	0	Exploration Hunt	CT Skills Addressed in the activity
Total	151	55	74	22		

The most frequently used code block by the students was "when … clicked/ when …. key pressed,". All of the 20 students used one of these code blocks since they needed a code block from the "events" menu to be able to run the code blocks they wrote. The second most used code block was “if …then/ if… then else,” used to control situations, 17 out of these 20 students used this code block in their projects. Eight of 17 students who used “if …then/ if… then else” code blocks had not taken an ICT course in the first or second semester at school. While 13 students used the "move ... steps" command used to move a sprite, 11 students used the "forever" code block used when an infinite loop is needed, in their projects. The code block "color... is touching?" used to detect if a sprite is touching a specific color, was used by 10 students in their projects. Again, 5 of these students who used the “forever” code block did not take ICT courses, while 3 had done so only for a few weeks.

Table 22. Scratch Code Blocks Not Covered in the CT Module

Scratch Code Blocks that Were Extra Used	Number of the Students Using the Code Block	Number of the Students Took ICT Courses in the First and Second Semester		
		Yes	No	A few weeks
go to x: …. y….	12	5	5	2
change y by…	11	4	5	2
key …. pressed?	10	4	4	2
change x by …. / set x to …	9	3	4	2
variables (my variable, set my variable to 0, change my variable by 1)	8	4	3	1
Pick random 1 to 100	7	3	4	0
Stop all	7	3	3	1
….. + ……. / ….. = ……..	6	3	2	1
when I start as a clone / create clone of myself	4	3	1	0
touching? (object)	4	0	4	0
point towards mouse pointer	3	1	2	0
mouse x	3	2	1	0
1.ask ……. and wait / 2. answer	3	3	0	0
change size by 10	2	2	0	0
set size to 100%	1	1	0	0
play note … for … beats	1	1	0	0

The code blocks that were not studied in the CT module but discovered and used by the students in their projects are listed in Table 19. The 3 most frequently used code blocks by the students were “go to x: …. y….” used to set a sprite’s x and y position, “change y by…” used to move a sprite in a Y axis, and “key …. pressed?” used to run a project with a key. 12 students used the go to x... y... code block. 11 students used “change y by…” blocks and 10 students used “key …. pressed?”.

Then, the students' projects were examined in terms of the usability of the design, using the rubrics prepared.

Table 23. Number of Projects that Implemented Designing for Usability

Criteria	Implementation of the Criteria		
	Full	Partial	None
Incorporates Themes	13	7	-
Character appearance changes	11	-	9
Goal (how to win-lose) clear	8	6	6
Functionality	8	10	2
Instructions clear (Bonus)	4	-	16

(N=20)

Using the designing for usability rubric, projects were scored according to the extent to which the students' projects met each of the 5 criteria listed in the rubric, fully, partially, or not at all. All the students applied fully incorporated or partially incorporated themes (which is explained in the data scoring section). 11 of the 20 students used character appearance changes in their projects. 14 students had a goal in their projects fully or partially. 18 of the 20 students' projects were fully or partially functional, and 4 students added clear instruction in their projects.

Table 24. Students' Projects, Goals, and Evaluations From the Designing for Usability Rubric

Name of the Scratch Projects	Goal of the projects	Incorporates Themes	Character appearance changes	Goal Clear	Functionality	Instructions clear	Total Point
Alien Invasion	collecting 75 points by killing aliens and protecting the earth from aliens	√	-	√	√	√	48
Zombies Battle	shooting zombies before 3 lives run out.	√	√	√	√	√	48
Collect the desserts	collecting the desserts falling from the top of the screen into the bowl	√	√	-	√	-	45
Flappy Bird	crossing the bird without hitting the pipes	√	√	√	√	-	35
Click To Catch the Balloon	popping 70 balloons in 75 seconds	√	√	√	√	√	31
Soccer Game	scoring a goal into the opponent's goalposts	√	√	√	√	-	31
Reach for the Cake	getting the character to the cake across the road without hitting the vehicles	√	√	√	√	√	29
A ball game	trying not to drop a ball by hitting the board	√	√	Partial	Partial	-	28
Flappy Bird	crossing the bird without hitting the pipes	√	-	Partial	Partial	-	28
Escape the Bus	gaining points by jumping the dragon over the incoming buses	√	-	Partial	Partial	-	23
Maze Game	getting the cat to the bread at the end of the maze	√	-	√	√	-	23
Maze Game	move the mouse to the finish point without hitting the black lines	Partial	√	Partial	Partial	-	18
Maze Game	getting the chicklet to the green line without touching the maze lines	Partial	-	√	-	-	18
Maze Game	reaching the finish point without crashing the apples in the maze	Partial	-	Partial	Partial	-	17
Pelican and Hunter	conversation among a pelican, 2 characters and a hunter	√	√	Partial	Partial	-	16
Mathematical Operations	solving 6 mathematical operations	Partial	√	-	Partial	-	14

Maze Game	taking the cat to the finish point	-	-	-	Partial	-	11
Escape from an aquarium	escaping from the aquarium by moving the fish with the arrow keys	Partial	-	-	Partial	-	10
Piano	playing a piano	√	√	-	Partial	-	10
Maze Game	taking the ball to the finish without touching the maze	Partial	-	-	-	-	10

Note: Projects are listed in the order from the one with the highest total points to lowest

In 3 of the 20 projects, the criteria in the design rubric were fully met. In 2 projects, all criteria were met, except for clear instructions, and the games worked perfectly for the purpose. In one project, the Zombies Battle game, the instructions were given in detail at the beginning of the game, and there was even a story written about the game, and gifs and music were used in the introduction. The use of visual elements in this game was consistent with the aim of the game. In the "Reach for the Cake", "Zombies Battle", "Click to Catch the Balloon", "Soccer Game", "Flappy Bird" and "Alien Invasion" projects, students drew their own characters using the drawing feature in the Scratch program. In addition, in these 5 projects, the students used the game elements of winning and losing. The rest did not include scoring.

Two of the 6 maze games lacked most of the criteria. In one, a maze was drawn, but the code blocks required to move the ball were not provided, neither were the necessary features added, so it only consisted of maze and ball visuals. In the other maze game, a maze with missing parts was drawn, an end point was set, but no code blocks were used, so the game did not work.

4.3 What are the learners' evaluation of the online CT module delivered remotely based on their feedback?

All 62 students filled out the feedback form, but 1 student's responses were not about the CT module, and therefore were not included in the analysis. In other words, the results reported in this section are based on the data from 61 students.

First the answers for each of the 4 open-ended questions are reported, and then the responses to Likert-scale type questions are reported.

4.3.1 Students' responses to open ended questions

The answers given by the students to the question "Which of the activities did you like the most? Why?" were tabulated so that the common answers given by the students were identified and classified into themes. The number of responses in each theme were listed.

While 47.5% (29/61) of the students gave a general answer for the activities they liked, other students wrote the names of the activities. 24.5% (15/61) of the students who gave general answers indicated that they liked all Scratch activities, and 22.9% (14/61) stated that they liked all Scratch and unplugged computing activities (N=61). While 9.8% (6/61) of the students indicated that all activities were entertaining, 8.1% (5/61) expressed that all activities were instructive, 16.3% (10/61) of the students said the reason they liked the Scratch activities was because Scratch was entertaining, and one student said because it was instructive. Of those who liked both unplugged activities and Scratch activities, 6.5% (4/61) said the reason was because these were instructive. In addition, 6.5% (4/61) of the students stated that they liked the Scratch program because they learned how to create games. Thus 8.1% (5/61) of the students emphasized learning in their responses, while 16.3% (10/61)

emphasized entertainment.

Apart from the general answer, 52.4% (32/61) of the students gave the name of their favorite activity in response to this question. Among the Scratch activities the most popular was “I'm coding the game”, as 14.7% (9/61) of the students named it as the one they liked the most. This was followed by Caesar's Cipher (13.1%), one of unplugged computing activities, and Save the Chicklet (9.8%). The most frequent answers given by the students to the question why they liked these activities were that the activities were entertaining (37.7%), and instructive (9.8) as stated by the students who indicated that they liked the Scratch and/or unplugged computing activities. In addition, the students stated that the reason for liking the activity was that it was difficult, or it was a new language, and in one case, moving the character.

All 61 students answered the question about what they learned in the activities and listed the 3 things they learned. The students identified 4 areas of learning in the responses they gave to this question: Learning Scratch in general, learning code blocks in Scratch, gaining CT skills (design of algorithms and debugging), and more generally learning about CS (basics of coding, and forming an algorithm in more than one way).

The majority of the students, 91.8% (56/61), stated that they learned to code in the Scratch program while doing the activities. Among these students, 12.5% (7/56) of them wrote the names of the code blocks they learned. These were “switch costume”, “switch backdrop”, “if command”, “if on edge, bounce”, and “turn degrees”. In addition, 21.4% (12/56) of these students listed the skills they learned from the activities. These were moving characters, making characters talk, adding voices, adding animations and creating their own games. Those who stated that they

learned how to animate in the Scratch program comprised 21.4% (12/56) of the students.

Students who said they learned the basics of coding constituted 40.9% (25/61) of the students. Apart from this, students who wrote that they learned to create algorithms or algorithms comprised 36% (22/61) of all students. Among these students, 2 students stated that they learned the algorithm in daily life. 2 other students stated that they learned that a question has many answers and that an algorithm can be created in different ways. There were also 2 other students who said they learned how to debug while building an algorithm.

In addition, some students indicated more generally that they learned the Scratch program and how to use the Scratch program. This comprised 21.3% (13/61) of all students.

In addition, 12.9% (8/61) of students wrote that they learned Caesar's Cipher, which was one of the unplugged activities. Finally, one student stated that they learned an algorithm for brushing teeth in space, referring to a video they watched as part of another unplugged activity.

Table 25 summarizes the skills that the students stated that they learned while doing the activities and the number of students.

Table 25. Student' Responses to Skills Learned During the CT Module

What did you learn?	Number of Students	Percentage
Coding in Scratch	56	91.8%
Basics of coding	25	40.9%
Algorithm, creating algorithm	22	36%
Scratch / How to use Scratch	13	21.3%
Caesar Cipher	8	12.9%
What computer scientists do	2	3.2%
Thinking, Logical thinking	2	3.2%
Problem Solving	1	1.6%
Developing a strategy	1	1.6%
Pixel	1	1.6%
Unplugged computing	1	1.6%

N=61

The students' most frequent answer to the question "what difficulties have you experienced during the CT module?" was that they had no difficulty during the activities. Twenty-six out of 61 (42.6%) students who responded as such.

The rest listed the difficulties they experienced. These fell into 3 categories: code writing, using Scratch, and infrastructural problems. The most frequently experienced difficulty was related to writing code: 15 students (24.6%) indicated that they had difficulties with code writing. These were related to specific activities covered in the lessons, such as flying the parrot, sequencing conversation balloons, creating animation in Scratch, writing an algorithm incorrectly, and starting to code.

Eight students (13.1%) stated more generally that they had difficulty using the Scratch program. These students wrote that they had difficulty finding code blocks, placing code blocks, and using Scratch.

The number of students who stated that they had technical problems was 4 (6.5%). The students listed these technical problems as computer malfunctions and Internet connection problems.

When the students were asked how they would conduct the lessons if they were the teacher, the majority, 48 out of 61 students (78.6%), responded that they would not change anything. A one fifth (21.3%) of the students made suggestions for instructional strategies that could be applied in the lessons. It was possible to classify these suggestions into 4 groups listed below.

Suggestions about classroom management:

- have students turn on their cameras
- have the students share the screen and tell them what they are doing

Suggestions for instructional strategies that the teachers may follow:

- start with the easy activities and continue with the harder ones
- provide shorter explanations, and do not show everything in Scratch
- ask questions and explain the lessons in more detail
- refrain from assigning homework, instead, do homework during class

Suggestions about content:

- do more Scratch activities and activities with more code
- show all code blocks available in Scratch

General suggestions:

- make the lessons more fun
- lengthen the duration of the lessons

4.3.2 Student responses to Likert-scale questions

The answers given by the students for the Likert scale are given below.

Of the 61 students, 60.5% (37/61) strongly agreed and 24.5% (15/61) agreed that the activities in the program contributed to learning new skills. Thus, the percentage of students who said that these activities helped them learn new skills totaled 85.2% (52/61). A total of 5% disagreed (2%) or strongly disagreed (3%) that these activities helped build new skills, while 9% indicated that they were uncertain.

The majority (85.2%) of the students indicated that it was easy to follow the flow of the lessons: 16.3% (10/61) agreed and 68.8% (42/61) strongly agreed with this statement. The percentage of those who disagreed or strongly disagreed constituted 9.8% (6/61) of the total, while 6.5% (4/61) of the students were uncertain.

There were two statements about the mode of delivery (i.e., remotely through video conferencing). 77% (47/61) of the students marked that it was not difficult to do the lessons online: 54% (33/61) strongly agreed and 22.9% (14/61) agreed with the statement about the difficulty involved in following the lessons online remotely. Those who disagreed (6.5%) or strongly disagreed (4.9%) that remote lessons were not difficult to follow constituted 13.1% (8/61) of the students. Students who stated that they were uncertain were 9.8% (6/61) of the total. It is possible to conclude that 22.9% (14/61) of all students experienced some level of difficulty following the lessons through video conferencing.

The second statement involved a comparison to face-to-face classroom instruction. 65.5% (40/61) of the students strongly agreed (49.1%) or agreed (16.3%) that the activities would be easier had they been conducted face-to-face in the classroom. 16.3% (10/61) of the students stated that they were uncertain. In addition to this, 16.3% (10/61) of the students stated that a face-to-face environment would

not make it easier to do the activities, as 6.5% (4/61) disagreed and 9.8% (6/61) strongly disagreed.

The percentage of the students who indicated that they had no difficulties understanding the activities were 73.7% (45/61): 44.2% (27/61) of these students said they strongly agreed and 29.5% (18/61) said they agreed that they did not have difficulty. 18% (11/61) of the students had difficulty understanding the activities, as 3.2% (2/61) disagreed and 14.7% (9/61) strongly disagreed with this statement. The students who were uncertain about whether they had problems understanding the activities were 8.1% (5/61).

There was one statement about the students' use of the skills they gained in the unplugged activities while doing the Scratch activities. 90.1% (55/61) of the students indicated that they agreed (19.6%) and strongly agreed (70.4%) with this statement. Thus, 90.1% (55/61) of the students indicated that they were able to use the skills they learned in unplugged computing activities in Scratch. In contrast, 4.9% (3/61) of the students strongly disagreed with this statement. The students who were uncertain constituted 3.2% of the total.

While 91.8% (56/61) of the students agreed (24.5%) or strongly agreed (67.2%) that they could do the activities without assistance of someone else, 24.5% (15/61) stated that they agreed, one of the students were undecided, and 4.9% (3/61) of the students disagreed that they could do the activities without help.

4.4 Mentors' feedback: Effectiveness of the CT module and design improvements needed

In order to gather feedback and suggestions for improvement, all mentors filled out a feedback form. Then interviews were held with 8 of the 10 mentors. First, the results

from the 6 open-ended questions, and the Likert-scale questions are provided below, and then the analysis of the interviews are given.

4.4.1 Mentors' responses to open ended questions

All mentors listed the 3 difficulties they experienced most during their implementation of the computational thinking module via video conferencing. The mentors identified 6 difficulties in the responses they gave to this question: monitoring students, student tardiness or absence, keeping track of students' homework, students' internet problems, and their own areas of improvement (i.e., lack of teaching experience or insufficient knowledge of Scratch).

The findings are summarized in Table 26 and described below.

Table 26. The Mentors' Responses to 4 Open Ended Questions on the Questionnaire

	Difficulties / Problems	How problem was resolved	Difficulties that could not be overcome	Transition from unplugged to Scratch activities	Help provided for smooth transition	Suggestions to improve the module
Zeynep	student tardiness or absence monitoring students (because of not turning on camera)	taking attendance at the beginning and end of the lessons	students' not turning on their cameras	no difficulty	none	register and save work in SSF's LMS
Selin	student tardiness or absence monitoring students (because of not turning on camera)	addressing the students by their names asking questions individually	student tardiness or absence	understanding code blocks	explaining the subject again and giving examples	include more entertaining and interesting examples
Gamze	monitoring students (because of not turning on camera) keeping track of homework	addressing the students by their names	keeping track of homework	no difficulty	explaining how to save projects step by step	well designed (none)

Zehra	monitoring students (because of not turning on camera) keeping track of homework students' internet problem	asking questions individually	students' not turning on the camera submission of homework students' internet problem	choosing the code blocks and combining them according to the scenario	selecting code blocks with students and explaining the code blocks at every stage while creating the animations	additional activities that explain the usage of code blocks
Ahmet	student tardiness or absence keeping track of homework	reminding the students to do the homework consistently	lack of maximum participation in the lesson	no difficulty	none	visuals to support the verbal instruction; more video and Web resources
Büşra	insufficient Scratch knowledge	doing Web research about Scratch	none	no difficulty	none	a platform for students to share their work
Begüm	internet problem keeping track of homework		students' internet problem	creating a Scratch account	asking students to create a Scratch account before class	increase the number of Scratch lessons
Burcu	students already familiar with Scratch keeping track of homework	assigning students to help other students who were not familiar with Scratch reminding the students to do the homework continuously	none	saving the projects	direct students to save projects over the phone	well designed (none)
Nisan	no difficulty	no difficulty	none	no difficulty	none	increase the number of examples, and extend the module content
Ayşe	lack of teaching experience	training provided for the mentors, teacher guidelines & lesson plans	lessons that rely on storytelling	understanding code blocks	explaining the purpose of the activities	Reduce story-like content, due to age group

The most frequent difficulty experienced by the mentors was related to keeping track of students' homework, as indicated by the 5 of the 10 mentors who responded to this question. One of these mentors also stated that she had difficulties in checking homework. Another mentor stated that she had difficulty in identifying students who submitted homework, because they submitted their assignments anonymously.

Four mentors had difficulty in monitoring students because they did not see whether the students were following the lessons or not. 3 out of 10 mentors stated that the difficulties they experienced most was related to students' lateness in signing on to class or their absence. In addition, 2 mentors stated that the students could not attend the lesson, or they had difficulty following the lessons due to internet connection problems.

Another 3 mentors stated that they had difficulties because some of the students already knew Scratch. Since the students who were already familiar with Scratch were in the same group as those who were not, it was difficult to teach. One of these mentors stated that she had difficulty in helping students who knew Scratch because her own Scratch knowledge was insufficient. In addition to this, this mentor also stated that students who know Scratch were sometimes bored. While one mentor stated that she had difficulties because she did not have teaching experience, Nisan stated that she did not have any difficulties.

The answers given by the mentors to the question "Which of the challenges you listed were possible to overcome? How?" were listed so that the common themes in the answers given by the mentors were identified and the frequencies calculated.

Three of the mentors, who stated that they had difficulty monitoring students because they did not see what the students were doing as the cameras were off,

indicated that they ensured students' participation in the lesson by addressing them by their names and/or asking questions individually, while one mentor stated that they could not solve this problem because students did not turn on their cameras. One of the mentors also indicated that she took attendance at the beginning and end of the lesson to ensure student participation.

Two of the 3 mentors, who stated that they had problems because the students already knew Scratch, gave students duties in the lesson so that they would not get bored or lose interest. They asked students to demonstrate certain features to other students who were not familiar with Scratch. These students shared their screens and explained the parts that their friends did not understand. Büşra, who had the same problem, stated that she was doing Web research about Scratch to help students.

In addition, Ahmet and Burcu who had problems related to keeping track of students' homework indicated that they consistently reminded the students to do homework. Ayşe, who had difficulties since she had no teaching experience, stated that she was able to solve this problem thanks to training, instructor's guide, and lesson plans. One of the mentors indicated that the students had difficulty in understanding how coding worked, so she explained the lesson objectives at the beginning of the lessons and the logic behind the code blocks.

Another question asked whether there were any difficulties they could not overcome, and if so, which ones. Three mentors, who had problems with submission homework submission, stated that the homework was not submitted despite reminding parents and students. Two out of 10 mentors stated that they had no difficulties that they could not solve. In addition, 1 of the mentors did not face any difficulties during the implementation of the module. The difficulties that could not be overcome by the 7 mentors were the problems of students' tardiness, problems

related to the internet, the lack of maximum participation in the lesson and the problem of students not turning on the camera.

The mentors' most frequent answer to the question "in what ways did your student have difficulty in transitioning from unplugged computing activities to Scratch activities?" was that they had no problems. Five out of 10 mentors who responded to this question said that students did not have any problems in transitioning to the Scratch program. In addition, 2 mentors stated that the students had problems with the code blocks. These mentors indicated that the students had problems in understanding the code blocks and choosing the code blocks and combining them according to a scenario. One of the mentors stated that the students had difficulties in opening a Scratch account, while another mentor indicated that students had problems in saving the projects.

The mentors' responses were specific to the question "How did you help students?" to provide a smooth transition. Four of the 10 mentors answered that they had no problems in transitioning from unplugged programming activities to Scratch activities. Two of the 10 mentors stated that they helped students who didn't save their projects by directing them to save projects over the phone. One of the mentors, who stated that the students had difficulties in understanding the code blocks, helped the students by explaining the subject again and giving examples. The other mentor stated that she helped the students by selecting code blocks with students and explaining the code blocks at every stage while creating the animations. The mentor who said that the students had difficulty in understanding lessons when presented within a story context explained the purpose of the activities and stated that it helped the students to understand. The mentor stated that she helped students who had

difficulties opening a Scratch account by asking students to create a Scratch account before class.

When the mentors were asked what they would recommend to the designer to improve the module, the majority of the mentors, 6 out of 10, made suggestions about the content. These suggestions are listed below.

- increase the number of examples and extend module content
- use more visuals to support the verbal instruction, video, and Web resources
- use more entertaining and interesting examples
- increase the number of Scratch lessons and show more features about the Scratch
- use additional activities that explain the usage of code blocks
- explain the lesson outcomes to the students and why they are doing these activities

In addition to this, two out of 10 mentors stated that the module was well prepared, and their students were satisfied with the lessons. Zeynep made a suggestion unrelated to the module's implementation that the students can be taught how to register and save work in SSF's LMS. Additionally, Büşra stated that a platform can be created for students to share their work.

4.4.2 Mentors' responses to Likert-scale questions

The findings from the mentors' responses given to the Likert scale questions are reported below.

All of the mentors stated that it was easy to follow the flow provided by the lesson plans of the CT module. Half of these mentors (5/10) strongly agreed with this statement, while the other half (5/10) expressed that they agreed with this idea. When asked about teaching remotely via video conferencing, 5 indicated that they agreed

(2/10) or strongly agreed (3/5) that it was not difficult to remotely teach the activities in the module. The other 5 mentors stated that they were uncertain.

Nine out of 10 mentors indicated that it would be easier to implement the activities in the module in a face-to-face environment, as 3 mentors agreed, and 6 strongly agreed with this statement, while Büşra disagreed with it, implying that it would not be easier in a classroom.

Nine of the 10 mentors stated that the module helped students to develop their computational thinking skills: 4 mentors agreed and 5 strongly agreed with the statement. Ayşe was uncertain about this statement. Another nine stated that the activities in the module were appropriate for the age of the students. While 5 of these mentors strongly agreed with this idea, 4 stated that they agreed. Ayşe stated that she was undecided about whether the activities in the module were appropriate for the age of the students.

All of the mentors agreed (6/10) or strongly agreed (4/10) that the unplugged computing activities were designed to support Scratch activities. All of the thought that it is possible for children to practice the activities in the module without the instructor. 6 of them strongly agreed with this idea while 4 of them agreed.

4.5 Results from the interview data

The findings from the interview data are summarized in Table 27 and described below.

In response to the interview question about the most effective aspects of the CT module for teaching computational thinking, the teachers listed the names of the activities they found most effective and explained why these were effective. The paper cup game in the activity “Find the rule of the game, start coding” and bug

hunting activity were found the most effective by the mentors. In the “find the rule of the game, start coding” activity, students wrote an algorithm to get the shapes created with paper cups and they used what they learned about the algorithm in previous lessons. This activity helped to develop students’ abstraction and generalization skills. In the bug hunting activity, students found and corrected the errors in three activities: the growing cycle of the tomato, activity related to weight and sudoku. These activities helped to enhance the students’ debugging skills.

While Ayşe and Zehra stated that Caesar's Cipher activity contributed to the development of students' computational thinking skills, Zehra and Gamze indicated that "Line it up and complete it!" activity was effective. In the Caesar's Cipher activity, students developed their abstraction skills by deciphering examples encrypted with the Caesar cipher and decrypting examples encrypted with emojis and geometric shapes. Gamze indicated that in "Line it up and complete it!" activity, students create algorithms consisting of different steps while writing a tooth brushing algorithm. I think I contributed to their learning by giving more details.

Büşra and Nisan thought that unplugged computing activities were effective. In addition, Begüm said that the Scratch and unplugged computing activities were interconnected so these were important for the development of students' computational thinking skills. Thanks to this, students did not have any difficulties when they were learning the Scratch program.

The mentors recounted several difficulties when asked “which concepts and/or skills did your student have difficulty understanding during the module?”. The common difficulties listed by the mentors were technical difficulties, lack of prior knowledge, and difficulties in understanding the terms. Zehra and Begüm mentioned that students have technical difficulties when painting pixels which is a part of the

message hidden in mosaics activity. Since the students had not painted something in a word document before, they had difficulty painting the pixels. Zehra and Gamze also stated that the students had difficulties while doing Caesar's Cipher activity. On the other hand, Selin expressed that since the students did not know the coordinate system, they had difficulty understanding the commands containing the x and y coordinate system. Additionally, Gamze said that some students had difficulties in understanding the terms in the paper cup game, the need to advance the length of only half of a paper cup, rather than that of one paper cup. Another difficulty that was mentioned by Zeynep was related to the computer science terms, because some of the students heard terms such as algorithm for the first time.

Büşra said that when they closed their eyes in the maze in the house activity, they had a hard time imagining their home. They could not draw their homes very well on paper, so activities that required imagination were challenging for them. In addition, Ayşe stated that students also had difficulty creating animations as they saw it for the first time in Scratch. Another difficulty mentioned was difficulty in understanding lessons that were built around a story. Ayşe said that instead of using lessons that rely on story-like activities, explanations of the purpose of the lessons could be more supportive to teach the topics. Nisan said that the students did not have any difficulties.

When the mentors were asked what they advise to the instructors who may teach the same module in the future, the mentors answers fell into two categories: recommendations regarding teacher preparation and those about instructional strategies. The majority of the mentors (6/10) recommended that the trainers prepare for the lessons using teachers' guidelines before they teach each lesson. As for instructional strategies, requiring that students turn on their cameras was a popular

suggestion, because it was difficult to monitor students when one could not see whether the students were following the lessons or not. Selin said that mentors should prepare in advance and give importance to time management, and she advised that mentors do not forget to mention the important points in the teacher's guidelines. They also suggested that it would be good for the instructors to study for the Scratch lessons. Also, Zehra thought that mentors can be provided with more detailed Scratch, and the logic behind the usage of code blocks can be explained to the mentors in order to transfer to students.

The last interview question was “How much do you think students have learned? Do you think there is development in understanding and thinking styles? In which areas have you observed this?” All of the 8 mentors said that the students learned new information and used it in the activities. Zeynep and Begüm said that even if some students had used the Scratch program before, they did not learn how to use code blocks. Thanks to the activities, they learned the logic of using code blocks.

Zehra said that the students took an active role in Scratch and participated in the classes. She indicated that “I could not get answers from the students at first, but later on, they started to participate in the lesson and started making comments. That's why it was good that we did the unplugged computing activities first, it helped students understand the subject and apply it in Scratch.” Gamze agreed that the students who did not respond at first began to attend and participate later on. Ayşe also agreed that there was a gradual increase in student response, that they began to raise more hands. She commented that the students could eventually create very nice games in Scratch.

Additionally, Nisan thought that the students also improved creating algorithms. She saw that students did not have difficulty in writing algorithms about

daily life. Even in Scratch, students easily created the algorithm when they were coding their games.

When the mentors were asked which aspects of the CT module they would change if they were the instructional designer, half (4/8) said that they would not want to change anything since the module was very descriptive, entertaining and educational. In addition, Büşra wanted to increase the number of the activities in Scratch. She said that students wanted to do more advanced projects, so further activities could be added, especially about detection and control blocks.

Selin suggested adding images, animations, and gifs to make the activities more colorful. She stated that more attention-grabbing stories and games can be used, and that the content can be more appealing for students. She said that the Maze game, Alice in Wonderland, was a good idea but the children were not familiar with the story. Begüm suggested that the concept of variables could be taught in both unplugged computing and Scratch lessons. Finally, Ayşe argued that the story/scenario-based activities were not appropriate for this age group, and she thought that the logic of the code blocks should be directly explained, with no context needed because children were old enough.

Table 27. Mentors' Assessment of the Module and Recommendations

	Most effective activities in CT building	Concepts/skills students had difficulty with	Development in student thinking and understanding	Recommendations for future instructors	Suggestions for the designer of the module
Zeynep	Paper cup game Secret message game	new terms like algorithm	the logic of using code blocks	require students to turn on cameras	keep as is, no change needed
Büşra	Unplugged computing activities, especially paper cup game, and	imagination code blocks in Scratch	-	prepare for the lessons using teachers' guidelines	increase the number of the Scratch activities

	maze in the house				more activities especially regarding detection and control blocks.
Ayşe	Paper cup game, Bug hunting, Caesar cipher.	Storytelling creating animations	making very nice games in Scratch raising hands and responding questions	prepare with teachers' guidelines	exclude story-like activities
Selin	The Caesar cipher, Bug hunting	the commands containing the x and y coordinate system	-	require students to turn on cameras prepare in advance and give importance to time management not forget to mention the important points in the teacher's guidelines	adding images, animations and gifs to make the activities more colorful. more attention-grabbing stories and games
Zehra	Bug hunting, Find the rules of the game, start coding, Line it up and complete it!	Caesar cipher Activities that required abstract thinking hidden message activity in mosaics using a computer	students took an active role in Scratch	more detailed Scratch training can be given to mentors and the logic behind the usage of code blocks can be explained	keep as is, no change needed
Gamze	Line it up and complete it!	the expression of half a paper cup in the paper cup game activity Caesar Cipher	students who did not answer before began to attend and respond to lessons	preparing with teachers' guidelines	keep as is, no change needed
Nisan	unplugged computing activities	did not have any difficulty	creating algorithms writing algorithms about daily life	prepare for the lessons using teachers' guidelines	keep as is, no change needed
Begüm	that all activities were interconnected	painting the pixels using a computer	the logic of using code blocks.	prepare for the lessons using teachers' guidelines	include the concept of variables

4.6 Integration of findings

When the students were asked what they learned while doing the activities, they stated that they learned the code blocks in Scratch, the Scratch program, and information about CS such as algorithms. The code blocks that the students used in their final Scratch programs and the fact that some of the students who did not take ICT lessons at school used new code blocks support this statement. Likewise, the mentors stated that the students learned new information and used it in their activities. In fact, 2 mentors stated that students who have used the Scratch program before also learned the logic of using code blocks thanks to this module. In addition, 2 mentors indicated that there was a gradual increase in student response, that they began to raise more hands. The positive improvement in students' self-efficacy perceptions after the module also supports the development in students' learning: there was an increase in the students' perceptions of their efficacy not only in algorithm design, problem solving, data processing, and basic programming, but also the self-confidence factor in the scale. Likewise, there was an increase in CT skill test results after students participated in the module.

90.1% of the students indicated that they were able to use the skills they learned in unplugged computing activities in Scratch. Likewise, all the mentors agreed that the unplugged computing activities were designed to support Scratch activities. In addition, all mentors thought that it is possible for children to practice the activities in the module without the instructor. Also, 91.8% of the students agreed that they could do the activities without assistance. Mainly, the answers of students and mentors support each other. In addition, 85.2% of the students stated that it was easy to follow the flow of the lessons and 9 mentors also indicated that the activities in the module were appropriate for the age of the students. In addition, 9 mentors

indicated that the module helped students to develop their computational thinking skills. Again, the students' and mentors' answers support each other.

The most popular answers given by the instructors for the most effective activities were "Find the rule of the game, start coding" in which the students developed an algorithm, and "Bug hunting" where the students found and corrected the errors in a connect the dots activity. The second popular answer was "Caesar's Cipher," which the mentors believed that it contributed to the development of students' computational thinking skills. Also, many mentors indicated that the "Line it up and complete it!" activity was effective. The students were not asked about the activities that were effective but were asked about the activities they liked the most. While 24.5% of the students stated that they liked Scratch activities, 22.9% stated that they liked Scratch and unplugged computing activities. In addition, 13.1% of the students stated that they liked the Caesar's Cipher, which the mentors found effective. Most of the students said that they liked these activities because they were entertaining and instructive. At the same time, 8 students stated that they learned Caesar's Cipher during this module.

42.6% of the students stated that they did not experience any difficulties while doing the activities. Other students stated that they had problems with writing code, using Scratch, and technical problems. The mentors, on the other hand, stated that students had technical problems, problems due to lack of prior knowledge, and difficulties in understanding new terms. While the students indicated the difficulties related to writing code with specific activity examples such as flying the parrot, sequencing conversation balloons, creating animation in Scratch, writing an algorithm incorrectly, and starting to code, the mentors did not mention these difficulties experienced by the students. Only Ayşe stated that the students had

difficulties because they were creating animation for the first time. While the technical problems that the students mentioned were related to the internet connection and computer malfunctions, the technical problems mentioned by the mentors were related to the students' not knowing how to use the Word document for example.

CHAPTER 5

DISCUSSION AND CONCLUSIONS

This study investigated the effects of an online computational thinking module that consists of unplugged and block-based activities on the CT skills of 6th grade students. The CT module consisted of 9 unplugged computing activities and 7 block-based coding (Scratch) activities and was delivered online by 10 mentors using a video conferencing tool to 62 students in groups of 6-7. The activities in the module were selected from different sources and each activity was designed to improve two or more CT skill categories, which are algorithmic thinking, abstraction, debugging, decomposition, and generalization. The main research questions were whether unplugged and block-based programming activities delivered through video conferencing in small groups were effective in developing the computational thinking (CT) skills of 6th graders, and how participation in such a module affected their perception of CT self-efficacy.

The results demonstrated that computational thinking skills significantly improved at the end of the program, and that the students' perception of their self-efficacy in CT skills also improved after participating in unplugged and block-based programming activities. The analysis of the students' final projects unveiled the code blocks those students learned in and out of the CT module. The feedback survey revealed which activities the students liked the most, the skills they thought they learned throughout the module, and the difficulties they experienced. The interviews with the mentors helped identify which activities were the most effective, the extent to which the students were able to develop CT skills as well as the difficulties

experienced by the students. The mentors also offered recommendations for future mentors and suggestions for the improvement of the module.

In the following section the findings in terms of research questions, limitations of the study and suggestions for the future research are discussed.

5.1 The effect of the CT module on CT skills and CT self-efficacy

The findings showed that the participants improved their CT skills after they attended the CT module comprising unplugged computing and block-based coding activities delivered online via video conferencing. This finding is consistent with the findings of other studies, such as those using various tools for block-based programming (e.g., Brennan & Resnick, 2012; Grover, 2014, Lye & Koh, 2014; Zhang & Nouri, 2019), as well as those using unplugged computing activities (e.g., Brackmann et al., 2017; Delal & Oner, 2020; Kalelioğlu, 2018; Rodriguez, Rader, & Camp, 2016).

What is unique about this study is that it made sequential use of unplugged computing activities and block-based programming in a complementary fashion, and it was delivered entirely online through video-conferencing, and targeted middle schoolers who were socioeconomically disadvantaged. While there are many studies on the effect of unplugged computing activities and the effect of teaching programming using block-based coding (usually with Scratch) on CT skills, there are not many studies examining how students' computational thinking can be supported online using both unplugged computing and block-based activities in a complementary way. Therefore, this study brought new findings to the literature, showing that starting with unplugged computing activities and then continuing with

Scratch activities can have a positive effect on developing students' computational thinking skills.

The activities in the CT module were prepared according to primarily five component skills of CT that were identified based on a synthesis of the CT literature: algorithmic thinking, abstraction, decomposition, debugging, and generalization. The unplugged computing and block-based activities in the module were prepared to address primarily these skills, among others, which were also included in the CT test questions. Therefore, it can be concluded that the module was instrumental in the development of the five mostly cited CT skills in the frameworks identified in research literature (Angeli et al., 2016; Grover & Pea, 2013; Kalelioğlu et al., 2016; Relkin et al., 2021; Selby & Woollard, 2013; Shute et al., 2017)

The findings also showed that participation in this CT module significantly improved the students' self-efficacy perception scores for all sub factors which are algorithmic thinking, problem solving, data processing, efficacy for basic, efficacy for self-confidence. This result is consistent with the findings reported in Erdem (2018), and İbili and Günbatar (2020). Similarly, Erdem (2018), and İbili and Günbatar (2020) reported increased self-efficacy perceptions of CT skills for middle school students who participated in block-based programming activities delivered face-to-face, though they had not used unplugged computing.

Some students may have difficulties in computer science lessons because of their negative attitudes towards the field (Bell et al., 2009). Researchers argued that computer science lessons with unplugged computing activities may change students' negative attitudes in a positive way, since unplugged computing activities contribute by decreasing difficulty and supplying entertaining games (Kalelioğlu, 2018; Rodriguez, Rader, & Camp, 2016). The results of the current study not only support

these findings, but also build upon them further by extending the positive findings to combined unplugged and block-based activities that are designed to complement one another. Also, this study might be one of the first to reveal positive effects of an online CT summer program delivered by volunteers through videoconferencing, as mandated by the conditions of the Covid-19 pandemic in 2021.

5.2 CT learning in students' Scratch projects

The second question of the study focused on examining the CT learning that occurred as a result of the students' participation in the module, based on their final products. The findings of many studies demonstrated that participating in block-based coding activities helps students to enhance their CT skills (Grover & Pea, 2013; Lee et al., 2011; Roman-Gonzalez et al., 2016). According to the study of Brennan and Resnick (2012), studying with block based computing tools and creating projects via these tools contributed development in students' CT skills. However, there are not many studies in the literature examining the products produced by students who learned block-based programming after unplugged computing activities. The findings of this study add to the literature in this way.

When the findings from the analysis of the 20 student projects are compared to those in Denner et al (2012), it can be said that students can use similar CS concepts and develop similar CT skills when developing projects in different block-based platforms.

The most frequently used Scratch code block in the student projects was "when … clicked/ when …. key pressed". This must be because they needed event code blocks to run the program. In the first week of Scratch lessons, students learned in the "Program the cat" lesson that they needed a start command or trigger for their

project to work, and this knowledge was used throughout all scratch lessons. When the students' use of the event code blocks was checked, it was observed that this code block was used correctly in all of the 20 projects. Denner et al. (2012) noted that students had difficulty with conditional statements and only 1% of the students in their study used conditional statements. Similarly, in Maloney et al. (2008) not many students (26%) used conditional statements in their artifacts. However, in this study, the second most used code block was "if …then/ if… then else". The majority of the students (85%) used a conditional code block, and more than half of these students (58.8%) had not taken an ICT course, or they had done so for a few weeks during the school year. It is an encouraging finding that students were able to include a conditional structure in their projects, even if they had not studied Scratch in school. In more than half of the projects (55%) the "forever" code block was used—the 4th most used code block in this study. Again, 72.7% (8/11) of the students who used this code had not taken ICT courses, or they had done so for a few weeks. Brennan and Resnick (2012)'s tinkering as a CT component may have been actualized in the case of these students.

Although variables and operators were not covered in the Scratch lessons in this study, 40% of the students (8/20) were able to use variables code blocks, and 40% (8/20) of the students use operators in their projects when they needed to, and half of these students had not taken an ICT course, or they had done so for a few weeks. This seems encouraging when compared to Maloney et al. 2008), who found that the operators and variables were the least used code commands in middle schoolers projects. It may be possible that some of the students were able to transfer what they learned in the unplugged computing activities and help discover and employ the lesser used code blocks. There is some evidence for this conclusion as the

majority of the students 90.1% (55/61) stated that they were able to use the skills they learned in unplugged computing activities in Scratch, even though 26 students had not taken an ICT course, and 14 students had taken it for a few weeks. Additionally, 35% (7/20) of the students who did not take ICT lessons at school used different code commands that they did not learn during the module. In addition, 3 students who stated that they took ICT lessons for a few weeks also used code commands that they did not learn in the module. It can be concluded that the use of unplugged computing activities before teaching coding with block-based computing tools may help students to discover new code blocks.

When the students' projects were evaluated in terms of the usability of design, it was found that the overwhelming majority (90%) of the students used an incorporated theme, which was also functional. In addition, 70% of the students defined a goal in their projects. However, few students added instructions to their projects. The reason for this may be that since the students wrote the purpose and rules of their games on paper while planning the game, they may have thought it unnecessary to include these in their actual design in Scratch.

5.3 Learners' evaluation of the online CT module

The findings from student feedback demonstrated that most of the students liked all Scratch activities, and close to a quarter of them (23%) liked both unplugged and Scratch activities. The most frequent answers given by the students to the question of why they liked these activities were that the activities were entertaining, and that they were instructive. This is consistent with Kalelioğlu and Gülbahar (2014)'s findings from a study carried out with 49 primary school students, and Brennan and Resnick (2012)'s study with 31 students, who all stated that they liked Scratch. It

was also important for the students to be able to create their own games, as was the case in this study.

When the students were asked what they learned while doing the activities, the most popular answers were coding in Scratch, coding, algorithm, usage of the Scratch program, and Caesar Cipher. The students who indicated that they learned coding in Scratch listed the code blocks they learned in Scratch, while students who said that they learned coding indicated that they learned the basics and logic of coding. Since the students learned the term algorithm and how to create algorithms in the unplugged coding activities, this finding supported that the unplugged computing activities were helpful to teach algorithms.

While 26 students did not face any difficulties, 15 students had problems while writing code, from which one might conclude that the unplugged activities posed no difficulty, and those related to Scratch were specifically about code blocks. This can be understandable, because the syntax of the code blocks is the reverse of the syntax of the Turkish language. As Scratch is based on English, a language with the Subject-Verb-Object word order and Turkish is a Subject-Object-Verb language, some students may have had difficulty with the configuration of the code blocks.

When the students were asked how they would teach the lessons if they were teachers, 48 students did not want to make any changes, while 13 students made suggestions about instructional strategies and assignments, all of which seemed useful, except showing all available Scratch codes.

When the Likert scale results were examined, it was shown that the majority of the students learned new skills, and this response of the students coincides with the increase in the Computational Thinking test results. The increase in the students' self-efficacy perception scale results is also consistent with the students' response. Of

the students who delivered their Scratch projects, 9 were students who did not take an ICT course, and 3 were students who took ICT courses for a few weeks. These students also completed their Scratch projects by learning new information throughout the module.

The activities in the module were designed to teach simple programming skills first and then move on to the more difficult ones. In addition, content that students are familiar with from daily life and that can trigger student interest was included. The majority of the students (85.2%) agreed that it was easy to follow the lessons. This supported that the content and processing of the module were suitable for students' level. This response by the majority of students is confirmed by the increase in CT scores.

Although many students preferred face-to-face instruction (65.5%), even a larger majority (77%) said they did not have any problems while doing the activities online over video conferencing. This may be because the students were accustomed to this kind of delivery by the end of the Spring 21 semester, when school closings and remote teaching were still common. Another reason might be related to the fact that the module was prepared in a way that supported student understanding and was designed specifically for remote delivery via video-conferencing. The activities were accompanied by instructions so that they can be delivered without the presence of a mentor. In fact, the overwhelming majority of the students (91.8%) indicated that they could do the activities on their own without the help of an instructor.

Nonetheless, it is possible to conclude that over one fifth (22.9%) of all students experienced some level of difficulty following the lessons through video conferencing. Also, 18% (11/61) of the students had difficulties understanding the activities and 8.1% (5/61) of the students stated that they were undecided about

whether they had problems understanding the activities.

5.4 The mentors' assessment of the CT module

The findings from the mentor feedback questionnaire complemented the findings from the interviews with them. While the open-ended questions in the feedback form were mostly about the difficulties experienced and their solutions, the interview questions included effective activities, challenges, and student learning, as well as suggestions for future mentors who might use the same module, and for the improvement of the module.

The difficulties stated by the mentors were mostly related to the participation of the students in the lesson, and infrastructural problems that are not directly related to the content of the module. However, three instructors pointed out that they had difficulty in helping students because their Scratch knowledge was insufficient, but they solved this problem as a result of their own research and by getting support from the students in their group. Another difficulty was the students' lack of prior knowledge of the basic underlying concepts. The students who did not know the order of the letters in the alphabet had difficulties while doing the Caesar's Cipher activity. Likewise, students who did not know the coordinate system had difficulty understanding the commands related to x and y axis in Scratch. In addition, the difficulties that could not be resolved by the majority of the mentors (7/10) were the students' tardiness, problems related to the internet, lack of full participation in the lesson, and the problem of students not turning on the camera.

The content-related suggestions were generally about increasing the examples in the lessons and using more visuals. All of the mentors, except one, stated that the activities in the module were suitable for the age group of the students. The mentor

who thought that the scenario-based activities were not suitable for the age group said she would give the objectives and the purpose of the lesson directly. This mentor did not have a background in education. It can be concluded that the training for the mentors should spend more time on conveying the pedagogical approach underlying the module and be available throughout the program.

As for the CT module's support for student learning, the data from the mentors confirmed that from the students. Most of the mentors stated that the module contributed to the development of students' computational thinking skills. Likewise, when asked whether there was any improvement in student learning, many mentors said that the students' participation in the course and their frequency of response increased compared to the first couple of sessions, and that the students learned the logic of using code blocks and writing algorithms. In the same way, the students were able to list the code blocks they learned in the lessons, used these and other code blocks in their projects, and responded that they acquired coding skills during the module, all which actually supports the answers of the mentors. The improvement in the CT self-efficacy perception scale also supports this result. In addition, the fact that the students who did not take ICT courses at school used code blocks that were not covered in the module also supports the mentors' assessment regarding student improvement. The mentors' views that the unplugged computing activities supported the learning outcomes in Scratch, may support the conclusion that these students learned how to use these code blocks on their own.

The mentors named many unplugged computing activities as most effective, where the students learned coding with games, using daily objects, or kinesthetically. The findings of the study confirmed previous findings that most of the students were satisfied with the way programming concepts were taught by playing unplugged

computing games, such as card games (Faber et al. 2017). It can be said that increasing the number of unplugged computing activities that emphasizes coding in the curriculum can help improve students' computational thinking skills.

5.5 Implications for research and practice

This study differs from the previous ones because it provided a module to support students' computational thinking skills by using a combination of unplugged computing and Scratch, a block-based coding tool. In addition, it consists of computer science activities that will attract students' attention. In previous research, block-based platforms (e.g. Lye & Koh, 2014; Román-González, Pérez-González, & Jiménez-Fernández, 2016; Shute, Sun, & Asbell-Clarke, 2017), unplugged activities (e.g. Delal & Oner, 2020, Kalelioğlu, 2018; Rodriguez et al., 2016), web-based simulation development tools and Arduino and robotic kits (e.g. Grover et al., 2015) were used to develop students' computational thinking skills. However, there are not many studies that integrate unplugged computing activities and block-based activities to teach computer science and examine students' CT skills.

The fact that the study was carried out remotely via the Zoom platform and the module was prepared with content suitable for distance education are also the differences that the study brings to the literature. Unplugged computing activities were designed in a way that students can easily do at home by using the objects in the house.

Although in this study, the content of the unplugged computing activities and Scratch activities in the module consisted of real-world problems, the effect of using real world problems on the students' computational thinking skills was not investigated separately. Findings of the previous studies stated that using real world

problems helps to develop students' computational thinking skills (Csernoch, Biro, & Math, 2021; Curzon, Selby, & Woollard, 2014; Sezer & Namukasa, 2021). Based on the increase in the students' CT scores at the end of the implementation, it can be concluded that this study provided indirect evidence about the effectiveness of real-world problems in teaching computational thinking.

Apart from this, the classes were formed with groups of 6-8 students, since students needed the support from the mentor while doing the Scratch activities. In this way, students were able to interact with the mentor through screen sharing.

This study has also demonstrated the importance of teacher preparation and training in pedagogies of teaching computing. The answers given by most of the mentors in this study, emphasized the importance of training and teacher's guidelines, and confirm the findings that pre-service teachers need to understand computational thinking skills and have the pedagogical knowledge necessary to teach them (Good, Sands, & Yadav, 2018).

Since this module was delivered online via a video-conferencing tool, the findings can be informative for situations where face-to-face education should be interrupted, such as during the COVID-19 pandemic. The results of the studies conducted to investigate the problems experienced by the teachers during the COVID-19 pandemic showed that the most frequently encountered problems were both teachers' and students' internet connection problems and drop in participation (Aytaç 2021; Eşici et al., 2021; Kamal & Illiyan, 2021; Tosun, Mıhcı, & Bayzan, 2021). It was stated that the reason for the decrease in student participation was that the students had other siblings going to school and did not have enough electronic devices (Tosun et al., 2021). Apart from this, the students' low motivation for learning, varied learning styles, and the families' inability to create a learning

environment at home were among the reasons given for the decrease in the students' participation in the lessons (Aytaç, 2021). Teachers had difficulty in monitoring students because they did not see whether the students were following the lessons or not. In addition, teachers' not knowing how to use distance education materials is among the problems experienced by teachers (Tosun et al., 2021). Also, in this study, the unstable internet connection, little participation in the lesson, students not turning on the camera, and student tardiness, cited as problems by the mentors they experienced sheds light on the problems experienced by the teachers during the pandemic.

According to the results of this study, some students had difficulties because they did not have sufficient prior knowledge in other fundamental fields (mathematics and Turkish). In future research, it can be examined whether having sufficient prior knowledge in other fields such as not knowing the coordinate system has an effect on the development of CT skills and learning to code, and how this learning need be best addressed within the CT curriculum.

This study can be inspiring for the design of content integrating block-based programming with unplugged computing activities, as such integration studies are rare in the literature. It would be worthwhile to study an integration that is not sequential but interlaced, as well as a blended version of the CT module used in this study.

5.6 Limitations of the study

As stated by the mentors and trainers, more activities could be done in a longer period of time, and thus more code blocks could be covered in Scratch, allowing students to create more comprehensive games. At the same time, students could be

given longer time to create projects because 2 sessions were limited for students to plan and create. However, since the module was implemented during the summer break, the content was prepared according to the maximum time available.

Another limitation was some of the students were already familiar with Scratch. In order for these students not to get bored in the lessons and to improve themselves, the mentors gave them tasks, and had them share their screens to explain how to implement a feature, just as an instructor would.

Although there were 61 students whose data was collected in the study, there were only 20 students who submitted their Scratch projects. Among these projects, there were well-equipped projects made using code blocks not covered in the course. An important limitation was that the resources students might have accessed outside of the module were unknown to the mentors.

Only one of the mentors was an ICT teacher, while the other 9 were not teachers, although they were graduates or interested in computer science-related fields, since they were all volunteers. The training, teacher guidelines and lesson plans provided for the mentors were intended for address this shortcoming.

There were difficulties associated with the implementation of the unplugged computing activities during a live session over video conferencing. Actually, these activities were prepared in such a way that they can be done using materials at home or only using paper. The mentor supported the students to produce ideas together by asking questions and to contribute to the ideas of others by listening to what they say and reflecting on it. At the same time, if students were doing a problem-solving activity such as Caesar' Cipher, they shared their screens, show their work to other students, and improve working together or solve mistakes together. Mainly, the

process of working together was formed by producing a common idea rather than creating a common physical product in unplugged computing activities.

APPENDICES

APPENDIX A

DEMOGRAPHIC QUESTIONNAIRE FOR MENTORS

Profession:
Grade:
Teaching Experience: ( ) Tutoring ( ) Private Teaching Institution ( ) Volunteer Teacher ( ) Other _____
Do you have teaching experience with IT and Software or Computer Science courses? ( ) Yes ( ) No If yes, please specify: _____

APPENDIX B

DEMOGRAPHIC QUESTIONNAIRE FOR MENTORS (TURKISH)

Bölüm:
Sınıf:
Geçmiş öğretmenlik deneyimi: ( ) Özel ders ( ) Dershane ( ) Gönüllü proje ( ) Diğer _____
BT ve Yazılım veya Bilgisayar Bilimi dersleriyle ilgili öğretmenlik deneyiminiz var mı? ( ) Evet ( ) Hayır Evet ise açıklama: _____

APPENDIX C

AN FEEDBACK QUESTIONNAIRE FOR MENTORS

Dear mentors,

Your suggestions for improving the computational thinking module are invaluable. After answering the questions below, we ask you to evaluate the module by filling out the 7-question survey form. Thank you.

Questions:

1. List 3 difficulties you experienced while applying the computational thinking module remotely.

2. Which of the challenges you listed were able to overcome? How?

3. Were there any difficulties that you could not overcome? What?

4. In what ways did your student have difficulty in transitioning from unplugged computing activities to Scratch activities?

5. How did you help them during this time?

6. What would you suggest to the designer to improve the module? Write your 3 suggestions.

	Strongly Disagreed	Disagreed	Undecided	Agreed	Strongly Agreed
The module helped students to develop their computational thinking skills.					
It was easy to follow the flow					

provided by the lesson plans of the CT module.					
It was not difficult to remotely teach the activities in the module.					
It would be easier to implement the activities in the module in a face-to-face environment.					
The activities in the module were appropriate for the age of the students					
The unplugged computing activities were designed to support Scratch activities.					
It is possible for children to practice the activities in the module without the instructor.					

APPENDIX D

AN FEEDBACK QUESTIONNAIRE FOR MENTORS (TURKISH)

Değerli yönderler,

Bilgi işlemsel düşünme modülünü iyileştirmek için sizin önerileriniz çok değerli. Aşağıdaki soruları cevapladıktan sonra 7 soruluk anket formunu doldurarak modülü değerlendirmenizi rica ediyoruz. Teşekkürler.

Sorular:

1. Bilgi işlemsel düşünme modülünü uzaktan uygularken yaşadığınız 3 zorluğu listeleyin.
2. Listelediğiniz zorluklardan hangilerini aşmak mümkün oldu? Nasıl?
3. Aşamadığınız zorluklar oldu mu? Neler?
4. Öğrenciniz bilgisayarsız kodlama etkinliklerinden Scratch etkinliklerine geçişte hangi açılardan zorlandı?
5. Bu sırada kendisine nasıl yardımcı oldunuz?
6. Modülü iyileştirmek için tasarımcıya neler önerirsiniz? 3 önerinizi yazın.

	Kesinlikle Katılmıyorum	Katılmıyorum	Kararsızım	Katılıyorum	Kesinlikle Katılıyorum
Bu modül bilgi işlemsel düşünmeyi öğrenmeye yardımcı oluyor.					
Modülün akışını takip					

etmek kolaydı					
Modüldeki etkinlikleri uzaktan uygulamak zor değildi.					
Yüz yüze ortamda olsaydık modüldeki etkinlikleri uygulamak daha kolay olurdu.					
Modüldeki etkinlikler öğrencinin düzeyine uygundu.					
Bilgisayarsız kodlama etkinlikleri Scratch etkinliklerini destekleyecek şekilde tasarlanmıştı.					
Modüldeki etkinlikleri yönder olmadan çocukların uygulaması mümkün.					

APPENDIX E

INTERVIEW QUESTIONS

1. What were most effective aspects of the CT module for teaching computational thinking?
2. Which concepts and/or skills did your student have difficulty understanding during the module?
3. If you were the designer of the module, which aspects of the CT module would you change?
4. What is your advice to the mentors who may teach the same module in the future?
5. How much do you think students have learned? Do you think there is development in understanding and thinking styles? In which areas have you observed this?

APPENDIX F

INTERVIEW QUESTIONS (TURKISH)

1. Modül boyunca bilgi işlemsel düşünmeyi öğretmek için en etkili kısımlar hangileriydi?
2. Modül boyunca öğrencinizin anlamakta zorlandığı kavramlar ve/veya beceriler hangileriydi?
3. Modülü tasarlayan siz olsaydınız hangi kısımlarını değiştirirdiniz?
4. Sizden sonra bu modülü uygulayacak olan yönderlere tavsiyeniz neler?
5. Öğrencilerin ne kadar öğrendiğini düşünüyorsunuz? Anlama düşünme biçimlerinde gelişim olduğunu düşünüyor musunuz? Bunu hangi alanlarda gözlemlediniz?

APPENDIX G

DEMOGRAPHIC QUESTIONNAIRE FOR STUDENTS

School:
Grade:
Mother's Educational Background: ( ) Primary School ( ) High School ( ) University ( ) Master's Degree **Mother's Profession: …….**
Father's Educational Background: ( ) Primary School ( ) High School ( ) University ( ) Master's Degree **Father's Profession: …….**
Do you have computer? ( ) Yes ( ) No
Do you have internet connection? ( ) Yes ( ) No
Applications you use: ☐ WhatsApp ☐ Instagram ☐ TikTok ☐ Facebook ☐ Youtube ☐ Games ☐ Others: …….

APPENDIX H

DEMOGRAPHIC QUESTIONNAIRE FOR STUDENTS (TURKISH)

Okul:
Sınıf:
Annenizin eğitim durumu: ( ) İlkokul Mezunu ( ) Lise Mezunu ( ) Üniversite ( ) Yüksek Lisans **Annenizin mesleği: …….**
Babanızın eğitim durumu: ( ) İlkokul Mezunu ( ) Lise Mezunu ( ) Üniversite ( ) Yüksek Lisans **Babanızın mesleği: …….**
Bilgisayarınız var mı? ( ) Evet ( ) Hayır
İnternet bağlantınız var mı? ( ) Evet ( ) Hayır
Kullandığınız Uygulamalar: ☐ WhatsApp ☐ Instagram ☐ TikTok ☐ Facebook ☐ Youtube ☐ Oyunlar ☐ Diğer: …….

APPENDIX I

A FEEDBACK QUESTIONNAIRE

1. Which of the activities did you like the most? Why?

2. What did you learn while doing the activities? Can you write 3 of them?

3. What difficulties did you experienced while doing the activities?

4. If you were the teacher, how would you conduct the lessons?

	Strongly Disagreed	Disagreed	Undecided	Agreed	Strongly Agreed
The activities in the module contributed to learning new skills.					
It was easy to follow the flow of the lessons.					
It was not difficult to do the lessons online.					
If we were in a face-to-face environment, it would be easier to do the activities.					
I had no difficulties understanding the activities.					

I was able to use the skills I learned in unplugged computing activities in Scratch activities.					
I was able to do the activities without assistance of someone else.					

APPENDIX J

A FEEDBACK QUESTIONNAIRE (TURKISH)

1.Etkinliklerden en çok hangilerini sevdin? Neden?

2.Etkinlikleri yaparken neler öğrendin? 3 tanesini yazar mısın?

3.Etkinlikleri yaparken yaşadığın zorluklar nelerdir?

4.Sen öğretmenin yerinde olsaydın dersleri nasıl yapardın?

	Kesinlikle Katılmıyorum	Katılmıyorum	Kararsızım	Katılıyorum	Kesinlikle Katılıyorum
Etkinlikler yeni beceriler öğrenmeme katkı sağladı.					
Derslerin akışını takip etmek kolaydı.					
Etkinlikleri uzaktan yapmak zor değildi.					
Yüz yüze ortamda olsaydık etkinlikleri yapmak daha kolay olurdu.					
Etkinlikleri anlamakta zorluk yaşamadım.					
Bilgisayarsız kodlama etkinliklerinde öğrendiğim becerileri Scratch etkinliklerinde kullanabildim.					

Etkinlikleri başkasının yardımı olmadan yapabildim.					

APPENDIX K

SELF-EFFICACY PERCEPTION SCALE FOR COMPUTATIONAL THINKING SKILLS (BİLGİ İŞLEMSEL DÜŞÜNME BECERİSİNE YÖNELİK ÖZ YETERLİK ALGISI ÖLÇEĞİ (BİDBÖA))

Ortaokul Öğrencileri İçin

Bilgi İşlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı Ölçeği

Değerli katılımcı, aşağıda yer alan ifadelere ilişkin, her madde içerisinde sunulan 3 seçenekten (Evet-1, Kısmen-2, Hayır-3) size en uygun olanı işaretleyiniz. Tercihlerinizin doğru ya da yanlış olarak bir değerlendirilmesi yapılmayacaktır. İfadelere, düşünerek ve içtenlikle vereceğiniz cevaplar için teşekkür ederiz.

Sıra No	Soru Metni	Evet	Kısmen	Hayır
1	Algoritmanın ne olduğunu biliyorum.	1	2	3
2	Algoritmaların hangi amaçla kullanıldığını anlıyorum.	1	2	3
3	Basit algoritmalar oluşturabilirim.	1	2	3
4	Yönergelerin ve işlem adımlarının önemini biliyorum.	1	2	3
5	Problem çözme sürecinde hatalarımı nasıl düzelteceğimi biliyorum.	1	2	3
6	Algoritmaların dijital araçlar için nasıl koda dönüştürüleceğini anlıyorum.	1	2	3

7	Döngü yapısında algoritmalar oluşturabilirim.	1	2	3
8	Koşullu algoritmalar oluşturabilirim.	1	2	3
9	Algoritma oluştururken mantıksal sorgulama yapabilirim.	1	2	3
10	Bir algoritmanın çıktısının ne olacağını tahmin edebilirim.	1	2	3
11	Algoritmada bulunan hataları ayıklayabilirim.	1	2	3
12	Çözümleri göstermek için şemalar kullanabilirim.	1	2	3
13	Dijital araçlar tarafından en iyi başarılan işlemlerin ne olduğunun farkındayım	1	2	3
14	Aynı problem için farklı çözümler üretilebileceğinin farkındayım.	1	2	3
15	Verinin ne olduğunu biliyorum.	1	2	3
16	Veri toplamanın önemini anlıyorum.	1	2	3
17	Dijital verinin farklı biçimlere dönüşebileceğini biliyorum.	1	2	3
18	Verinin farklı türleri olduğunun (sayı ve metin) farkındayım.	1	2	3
19	Problemlerin çözümünde farklı veri türleri kullanılabileceğini biliyorum.	1	2	3
20	Verilerin tablo yapısında daha anlamlı sunulabildiğini biliyorum.	1	2	3
21	Veri ve bilgi arasındaki farklı açıklayabilirim.	1	2	3
22	Koşullu yapıları ve döngüleri oluştururken aritmetik operatörleri kullanabilirim.	1	2	3
23	Değişkenleri tanımlayıp kullanabilirim.	1	2	3
24	Farklı kontrol durumları için değişik döngüler oluşturabilirim.	1	2	3
25	Bir döngüyü sonlandırmak için değişken ve ilişkisel operatörleri kullanabilirim.	1	2	3

26	Belirli işlemler için hazır fonksiyonları kullanabilirim.	1	2	3
27	Bir problemi okuduğumda, çözüm için hangi bilgiye ihtiyacım olduğunu düşünürüm.	1	2	3
28	Bir problemi okuduğumda, çözüm için gerekli ve gereksiz olan bilgiyi ayırt edebilirim.	1	2	3
29	Bir problemi çözebilmem için yeterli veri sunulup sunulmadığına karar verebilirim.	1	2	3
30	Bir problemi okuduğumda, daha önce çözdüğüm problemleri düşünerek benzerlik ve farklılıklarına göre aralarında ilişki kurarım.	1	2	3
31	Problem çözüm sürecinde işlem önceliklerine dikkat ederim.	1	2	3
32	Problemi çözüp sonucunu bulduktan sonra yaptığım işlemleri kontrol ederim	1	2	3
33	Farklı çözüm yollarını inceleyerek daha iyi bir çözüm bulmaya çalışırım.	1	2	3
34	Problem çözerken, hangi işlemi neden yaptığımı sürekli sorgularım.	1	2	3
35	Problemi çözüp sonucunu bulduktan sonra yaptığım işlemleri kontrol eder varsa hataları düzeltirim.	1	2	3
36	Bir problem için ürettiğim çözümü farklı problemlere genelleyebilirim.	1	2	3

APPENDIX L

COMPUTATIONAL THINKING SKILL TEST

Bilgi İşlemsel Düşünme Beceri Testi

Sevgili Öğrenciler,

Bu test 14 sorudan oluşuyor. Her soruda 4 şık var ve sadece 1 doğru cevap var.

Doğru olduğunu düşündüğünüz şıkkın üstüne basın. Bazı sorular zor olabilir.

Yapamadığınız sorular olursa boş bırakıp geçebilirsiniz. Başarılar.

1. Beş Çubuk

Ali'nin beş tahta çubuğu vardır. Bu çubukları masaya koyarak bir şekil oluşturur.

Nil masaya gelerek bir çubuğu alır ve aşağıdaki şekilde görüldüğü gibi yerini değiştirir:

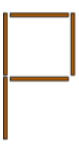
Daha sonra Burak masaya gelir, o da bir çubuğu alır ve yerini değiştirir.

Soru

Aşağıdakilerden hangisi, Burak'ın yaptığı şekil olamaz?

A) 

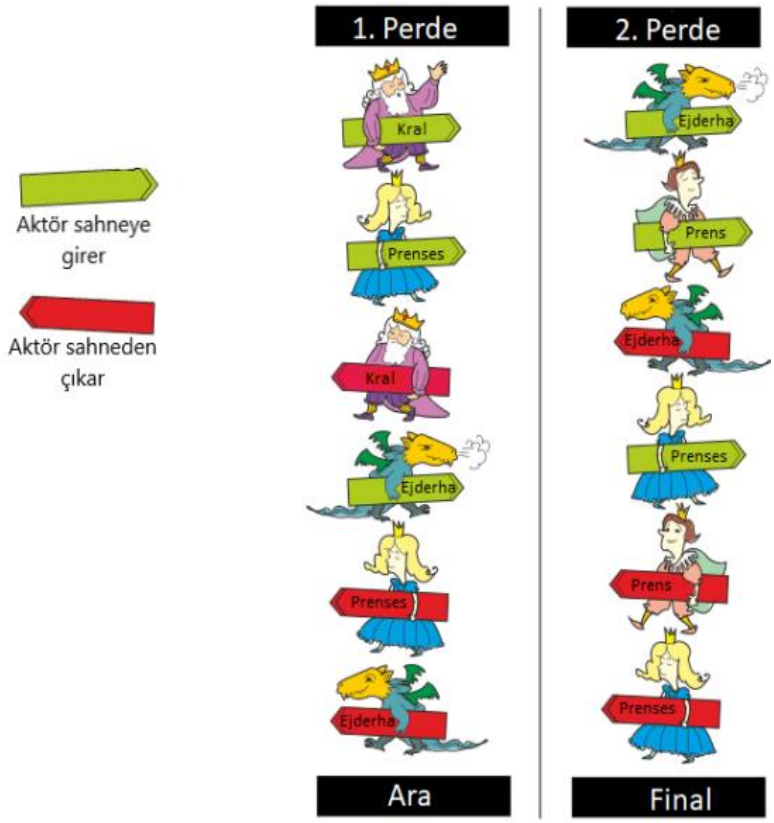
C) 

B) 

D) 

2. Tiyatro Oyunu

Bir tiyatro oyunundaki aktörler, resimde gösterilen sıraya göre (yukarıdan aşağıya) sahneye girer ve çıkar. Oyunda 2 perde ve bir ara bulunmaktadır.



Buna göre aşağıdaki ifadelerden hangisi doğru değildir?

A) Prens ve prenses sahnede birlikteydi.

B) Prens ve prenses sahnede birlikteydi.

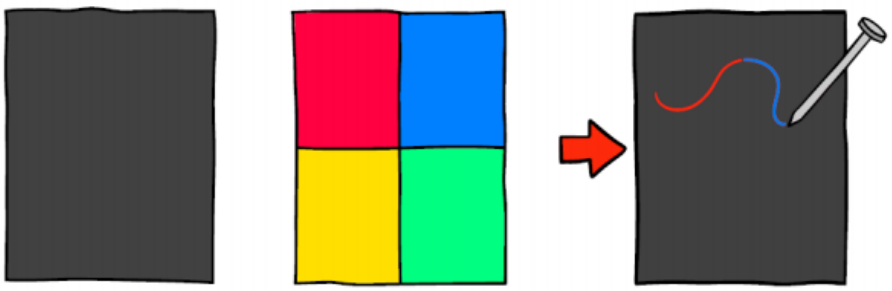
C) Kral ve ejderha sahnede birlikteydi.

D) Prens aradan sonra sahneye çıktı.

E) Prens ve ejderha sahnede birlikteydi.

3. Kağıt Kazıma Sanatı

Kazınabilen sanat kağıdını keskin bir nesneyle çizerek güzel resimler oluşturabilirsiniz.



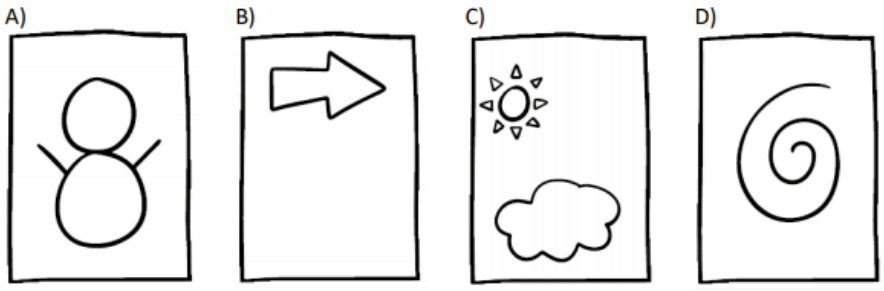
Öncelikle sanat kağıdını siyah olarak görürüz.

Bu katmanın altında 4 farklı renk gizlidir.

Kazımaya başladığımızda alttaki renkler ortaya çıkar.

Soru

Aşağıdaki resimleri sivri uçlu bir nesneyle kazınabilen sanat kağıdına çizerseniz, hangisi tam olarak üç renk gösterir?



4. Mesaj İletimi

Ayşe arkadaşı Can'a postacı ile uzun bir mesaj göndermek ister. Ayşe, mesajları her bir kartta en fazla 3 harf olacak şekilde gruplara ayırır ve bu kartların her birini bir postacıya verir.

Mesaj taşıyan postacılar kartları Can'a farklı zamanlarda iletmektedir. Bu yüzden, Ayşe kartları postacıya vermeden önce her birine sıra ile numara verir. Can, mesajı anlayabilmek için kartları sıralamak zorundadır.

Örneğin, KELİMEBUL mesajı için, Ayşe'nin oluşturduğu 3 kart aşağıdaki gibidir.

1 KEL	2 İME	3 BUL

Can aşağıdaki mesajı almıştır.

3 İŞB	5 NDA	2 DUZ	1 KUN	4 AŞI

Bu durumda Can'a gelen mesajın ilk hali nedir?

A) NDADUZİŞBKUNAŞI

B) İŞBAŞINDAKUNDUZ

C) KUNDUZİŞBAŞINDA

D) KUNDUZAŞINDAİŞB

5. Sincap ve Palamut

Aşağıdaki bilgisayar oyununda sincap karakteri çok sevdiği palamuta ulaşmaya çalışıyor, ancak bu oyunu yazan bilgisayar programcısı bir hata yapmış ve komutları yanlış yazmış.

Aşağıdaki program komutlarından hangisi sincabın palamuta ulaşmasını sağlar?

A)

B)

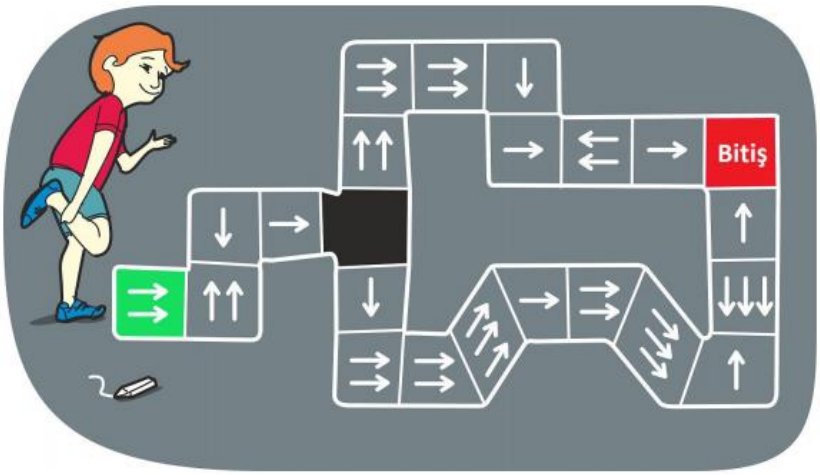
C)

D)

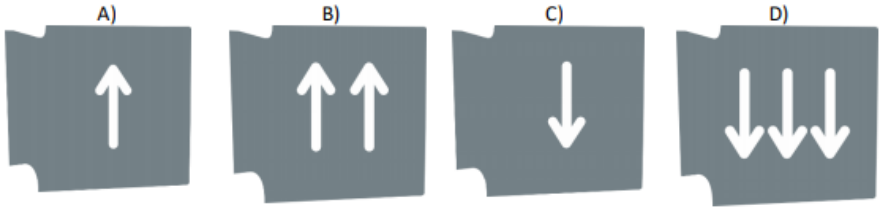
6. Yol seçimi

Burak bahçede seksek gibi atlamalı bir oyun çizdi.

Burak oyuna ilk (yeşil) bölmeden başlayacak. Sonra şu kuralı tekrarlayacak: üzerinde durduğu bölmeden, o bölmedeki okların yönünde ve ok sayısı kadar atlayacak.



Bitişe ulaşmak için Burak'ın boş bölmeye ne çizmesi gerekir?



7. Robot Kol

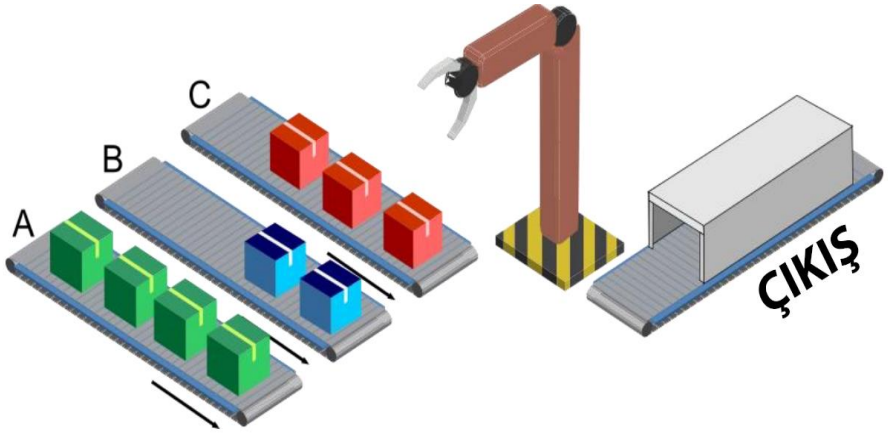
Bir fabrikada işlem ünitesinde kullanılan robot kolu, nesneleri son ÇIKIŞ bandına taşımak için üç farklı taşıma bandından (A, B, C) nesneler alır. Robot kolu şu şekilde çalışır:

Adım 1: önce A'dan bir nesne alır ve onu ÇIKIŞ'a taşır,

Adım 2: sonra B'ye hareket eder, B'den bir nesne alır ve ÇIKIŞ'a taşır,

Adım 3: son olarak, A bandına tekrar başlamadan önce C'de aynı adımları gerçekleştirir.

Taşıma bandına koyulacak bir nesne olmadığında robot kol bir nesne gelene kadar bekler, çünkü işlem ünitesinin ilerlemek için her bantta bir nesneye ihtiyacı vardır.



Yukarıda gösterilen durumda, taşıma bantlarına (A, B, C) daha fazla yeni nesne gelmeyecek. Öyleyse çalışma kurallarına göre robot kol kaç nesneyi hareket ettirilecektir?

A) 5

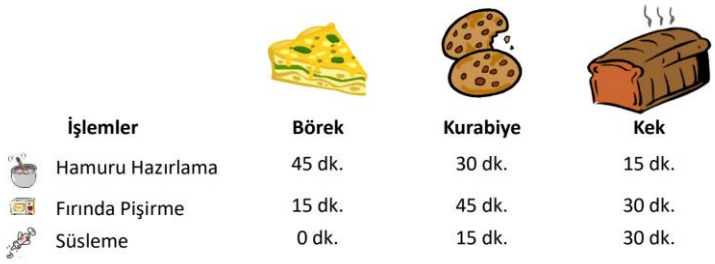
B) 6

C) 7

D) 8

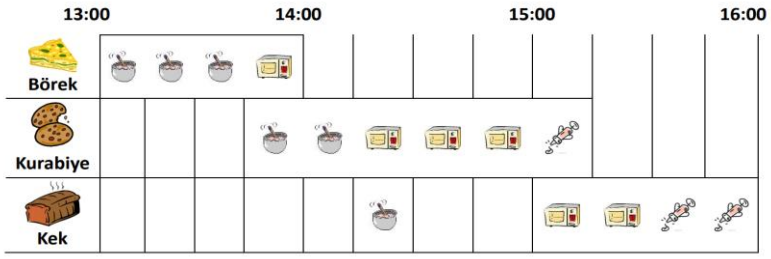
8. Doğum Günü Kutlaması

Sibel ve Levent öğleden sonra saat 15:00'de başlayacak olan doğum günü kutlamasına gidecek. Giderken taze yapılmış kurabiye, kek ve börek götürmeye karar verdiler. Hamur İşleri tarifi kitabından aşağıdaki pişirme sürelerini öğrendiler.



İşlemler	Börek	Kurabiye	Kek
Hamuru Hazırlama	45 dk.	30 dk.	15 dk.
Fırında Pişirme	15 dk.	45 dk.	30 dk.
Süsleme	0 dk.	15 dk.	30 dk.

Hazırlık sürecini 3 aşamaya böldüler. Levent hamuru hazırlayacak ve sonra fırına verecek. Kek piştikten sonra Sibel krema ile süsleme yapacak. Fırında aynı anda sadece tek bir yiyecek için yer var. Sibel ve Levent aynı anda yalnızca tek bir yiyeceği hazırlayabilir. Saat 13:00 olunca işe başlayıp saat 15:00'de kutlamaya gitmek istiyorlar. O yüzden aşağıdaki planlamayı yaptılar:



Pişirme sürelerinin uzunluğu dikkatlerini çeker ve saat 15:00'e nasıl yetişeceklerini düşünürler. Bu yüzden zamanında hazır olabilmek için pişirme sırasını değiştirmeye karar verirler.

Sibel ve Levent bu 3 hamur işini en erken saat kaçta hazırlayabilir?

A) 14:15

B) 14:30

C) 14:45

D) 15:00

9. Geometrik Süsler

Bilgisayar programcısı bir bilgisayar programı yazmış. Bu program kare ve üçgenlerden oluşan geometrik şekilleri verilen yönergelere göre sıralamaktadır. Programda şekillerden desen oluşturmak için aşağıdaki yönergeneler kullanılmaktadır.

bK: büyük kare

kK: küçük kare

bU: büyük üçgen

kU: küçük üçgen

Bir yönergeyi tekrar etmek için T [Y] kullanılmaktadır. Buna göre T bir işlemin tekrar sayısını Y ise tekrar edilecek yönergeyi belirtmektedir.

Örneğin; kK 2 [bU kU] bK yazıldığında aşağıdaki şekil oluşmaktadır.

Soru

Aşağıdaki şeklin oluşturulması için yazılması gereken yönerge nedir?

A) kK 2 [kU kK bU] kU Bk

B) kK 3 [kU kK bU] bK

C) bK 3 [kU kK bU] kU bK

D) bK 2 [kU kK bU] kU bK

10. Robot BB-8

Yukarıdaki program komutlarını yazan bilgisayar programcısı Robot BB-8'in ilerleyerek önündeki metalleri toplamasını istiyor. Yazdığı program çalıştığında aşağıdakilerden hangisi doğru olur?

A) BB-8 hareket etmez.

B) BB-8 4 kez ilerler ve tüm metal parçaları toplar.

C) BB-8 5 kez ilerler ve tüm metal parçalarını toplar

D) BB-8 4 kez ilerler ve 2 metal parça toplar.

11. Yapboz

Küçük kardeşim Ali, oyuncak dükkanının duvarında bir resim gördü. Bu resimdeki şekle sahip bir oyuncak monte etmek istiyor. Oyuncağı yapmak için tezgahtaki şekillerden satın alacak. Her şeklin fiyatı farklı, 1 jeton olan da var, 7 jeton olan da var; hepsinin üstünde kaç jeton olduğu yazıyor. Şekilleri istediğimiz yönde döndürebiliyoruz. Ali'nin harçlığı az olduğu için bu oyuncağı monte edebilmek için çok jeton harcamak istemiyor.

Ali'nin istediği oyuncağı monte etmek için en az kaç jeton harcaması lazım?

A) 20 jeton

B) 16 jeton

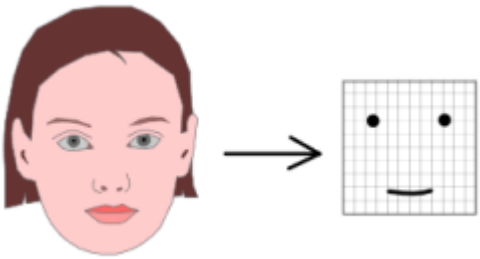
C) 13 jeton

D) 14 jeton

12. Mutlu Yüz

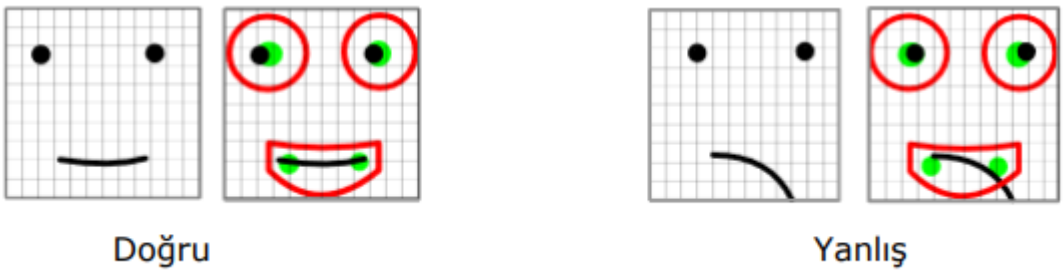
Bilge, kamerada insan gülümsemesini algılayan bir sistem geliştirmiştir. Bu sistem insan gülümsemesini iki adımda algılamaktadır:

1) Ön-işlem: Yüzün resmi gözlere karşılık gelen iki nokta ve ağıza denk gelen bir çizgiden oluşan mutlu yüz modeline çevrilir.

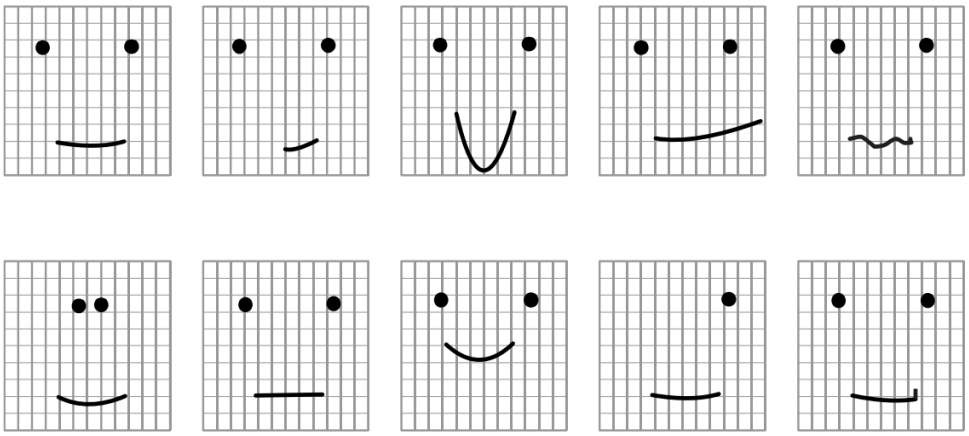


2) Yüz-algılama: Mutlu yüz modeli kırmızı çizgiler ve dört yeşil noktayı içeren bir desenle karşılaştırılır.

3) Yüz modeli ancak bütün yeşil noktalar kırmızı çizgiye değmediği sürece mutlu olarak kabul edilir.



Aşağıdaki yüzlerin ön-işlem sürecinden geçtiği varsayılarak, kaç tanesi mutlu olarak algılanır?



A) 4

B) 5

C) 6

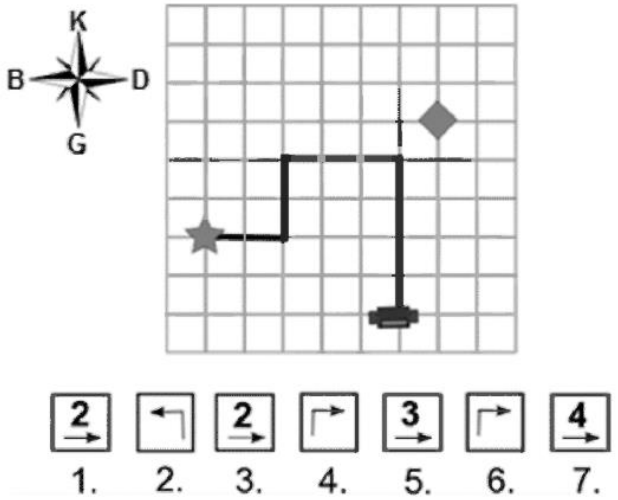
D) 8

13. Hatalar

Bir robotu kontrol eden 3 tür buton vardır:

	Robot sola döner.
	Robot sağa döner.
	Robot X birim kadar bulunduğu yönde ilerler.

Robot yıldızın olduğu konumdan doğu yönüne doğru hareket etmeye hazırdır. Bilge Kunduz aşağıda verilen 7 butona sıra ile (soldan sağa doğru) basar ve robotu eşkenar dörtgenin bulunduğu konuma getirmeye çalışır. Ancak 2 butona yanlışlıkla basar.



Soru Bilge Kunduz hangi 2 butona yanlışlıkla basmıştır?

A) 1 ve 2 numaralı butonlar

B) 1 ve 4 numaralı butonlar

C) 3 ve 4 numaralı butonlar

D) 2 ve 6 numaralı butonlar

14. Sonsuz Dondurma

İki dondurma standı vardır. İkisi de, aynı dört rengi kullanırlar:

Birinci stand, dondurma yapmak için aşağıdaki talimatları kullanır:

1. Boş bir külah ile başla.

2. Rastgele bir renk seç ve bu renkten iki top ekle.

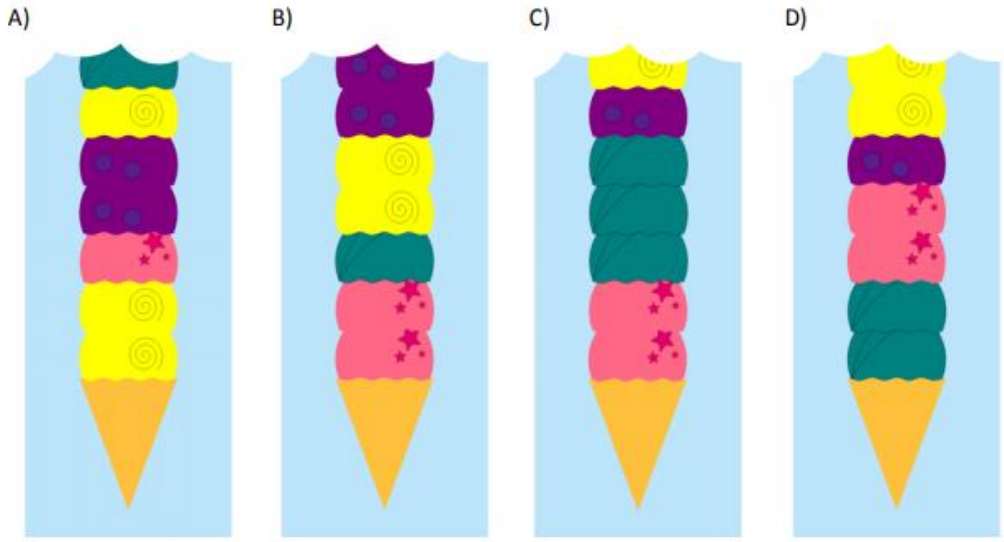
3. Farklı herhangi bir renkten bir top ekle.

4. İstenen yüksekliğe ulaşılmışsa, dur, aksi takdirde Adım 2'e geç.

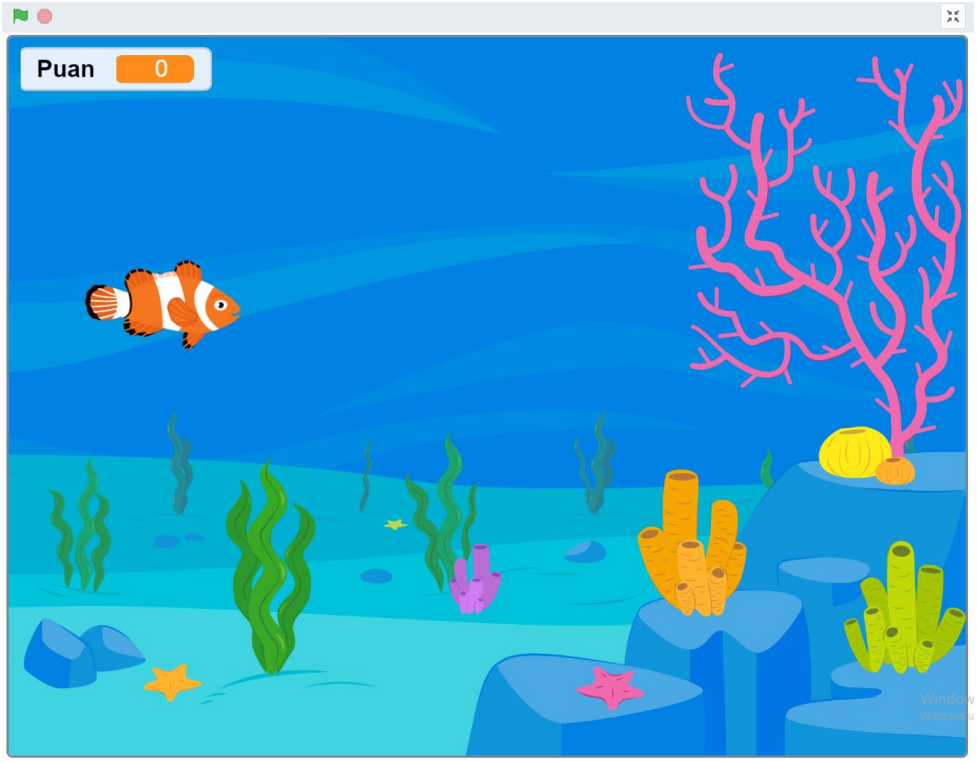
İkinci stand herhangi bir talimatı takip etmez.

Her dondurma külahının sadece ilk birkaç top dondurmasını görebilirsiniz.

Bunlardan hangisi kesinlikle ikinci stanttandır?



15. Balıklı Oyun



Yukarıdaki oyun Scratch programı ile yapılmıştır. Oyunumuzun kuralları aşağıdaki gibidir:

1. Oyun klavyeden “boşluk” tuşuna basıldığında başlar.
2. Oyun başlatıldığında puan her zaman 0’dır.
3. Oyun durdur düğmesine (yeşil bayrağın yanındaki kırmızı düğme) basmadıkça devam eder.
4. Balık ekranda sürekli 10 adım ilerlemektedir.
5. Balığa tıkladığımızda balık 1 saniye boyunca gizlenir ve 1 saniye sonra herhangi bir yerde görünür olur.
6. Balığa tıklandığında puan değişkeni 10 artar.

Bilgisayar programcısı balıklı oyunu bu 6 kurala göre yapmış. Aşağıdaki komutlardan hangisi bu oyunun çalışmasını sağlar?

A)


boşluk tuşuna basılınca
Puan i 0 yap
sürekli tekrarla
10 adım git
eğer fare-imlecine değiyor mu? ise
Puan i 10 kadar değiştir
gizle
1 saniye bekle
rastgele konuma 'e git
göster

B)


boşluk tuşuna basılınca
Puan i 0 yap
sürekli tekrarla
eğer fare-imlecine değiyor mu? ise
Puan i 10 kadar değiştir
gizle
1 saniye bekle
rastgele konuma 'e git

C)


boşluk tuşuna basılınca
Puan i 0 yap
sürekli tekrarla
10 adım git
eğer fare-imlecine değiyor mu? ise
Puan i 10 kadar değiştir
göster
1 saniye bekle
rastgele konuma 'e git
gizle

D)


boşluk tuşuna basılınca
Puan i 0 yap
sürekli tekrarla
10 adım git
eğer kenara değiyor mu? ise
Puan i 10 kadar değiştir
göster
1 saniye bekle
rastgele konuma 'e git
gizle

APPENDIX M

ETHICS COMMITTEE APPROVAL

Evrak Tarih ve Sayısı: 26.03.2021-9631

T.C.
BOĞAZİÇİ ÜNİVERSİTESİ
SOSYAL VE BEŞERİ BİLİMLER YÜKSEK LİSANS VE DOKTORA TEZLERİ ETİK İNCELEME KOMİSYONU
TOPLANTI TUTANAĞI

Toplantı Sayısı	:	14
Toplantı Tarihi	:	25.03.2021
Toplantı Saati	:	13:00
Toplantı Yeri	:	Zoom Sanal Toplanti
Bulunanlar	:	Dr. Öğr. Üyesi Yasemin Sohtorik İlkmen, Prof. Dr. Ebru Kaya, Prof. Dr. Fatma Nevra Seggie
Bulunmayanlar	:	

Elif Çil

Bilgisayar ve Öğretim Teknolojileri Bölümü

Sayın Araştırmacı,

"Uzaktan eğitim yoluyla ortaokulda bilgi işlemsel düşünme becerilerinin geliştirilmesi" başlıklı projeniz ile ilgili olarak yaptığınız SBB-EAK 2021/16 sayılı başvuru komisyonumuz tarafından 25 Mart 2021 tarihli toplantıda incelenmiş ve uygun bulunmuştur.

Bu karar tüm üyelerin toplantıya çevrimiçi olarak katılımı ve oybirliği ile alınmıştır. COVID-19 önlemleri kapsamında kurul üyelerinden ıslak imza alınamadığı için bu onam mektubu üye ve raportör olarak Ebru Kaya tarafından bütün üyeler adına e-imzalanmıştır.

Saygılarımızla, bilgilerinizi rica ederiz.

Prof. Dr. Ebru KAYA
ÜYE

e-imzalıdır
Prof. Dr.Ebru KAYA
Raportör

SOBETİK 14 25.03.2021

APPENDIX N

PERMISSION LETTER FOR MENTORS

T.C.
BOĞAZİÇİ ÜNİVERSİTESİ
Sosyal ve Beşeri Bilimler İnsan Araştırmaları Etik Kurulu
KATILIMCI BİLGİ ve ONAM FORMU

Araştırmayı destekleyen kurum: Boğaziçi Üniversitesi
Araştırmanın adı: Uzaktan eğitim yoluyla ortaokulda bilgi işlemsel düşünme becerilerinin geliştirilmesi
Proje Yürütücüsü: Günizi Kartal

E-mail adresi:

Telefonu:

Araştırmacının adı: Elif Çil

Adresi:

E-mail adresi:

Telefonu:

Değerli Öğrenciler,

Eğitim Teknolojisi yüksek lisans öğrencisi olarak 6. sınıf öğrencilerinin bilgi işlemsel düşünme becerilerini geliştirmek için bilimsel bir araştırma projesi yürütüyorum. Sizleri bu çalışmaya katılmak için davet ediyorum.

Katılmayı kabul ettiğiniz takdirde sizden kısa bir demografik bilgi anketi doldurmanızı isteyeceğim. Ortaokul öğrencileriyle uygulayacağınız bilgi işlemsel düşünme becerisi geliştirme modülü tamamlandıktan sonra çalışmaların faydasını/eksiklerini değerlendirmeniz için geri bildirim soruları cevaplamanızı ve online görüşme yapmak isteyeceğim. Bu yaklaşık 30 dakikanızı alacaktır.

Bu araştırma bilimsel amaçla yapılmaktadır ve katılımcı bilgilerinin gizliliği esas tutulmaktadır. Anket, form ya da sınavlardaki isminiz ya da herhangi bir kişisel veriniz benim haricimde kimse tarafından görülmeyecektir. Çalışma süresince elde edilen veriler yüksek lisans tezimde kullanılacaktır. Aynı veriler daha sonra bilimsel makale veya sunumlarda kullanılabilir. Bu durumlarda da isminiz geçmeyecektir.

Araştırma projesi hakkında ek bilgi almak istediğiniz takdirde lütfen benimle ya da tez danışmanım Boğaziçi Üniversitesi Bilgisayar ve Öğretim Teknolojileri Bölümü Öğr. Üyesi Dr. Günizi Kartal ile temasa geçiniz. Ayrıca araştırma çalışması ile ilgili haklarınız konusunda Boğaziçi Üniversitesi Sosyal ve Beşeri Bilimler Yüksek Lisans ve Doktora Tezleri Etik İnceleme Komisyonu ile görüşebilirsiniz.

Yukarıdaki açıklamaları okudum ve anladım. Çalışmaya katılmayı kabul ediyorum. Formun bir örneğini aldım / almak istemiyorum.

Katılımcının Adı-Soyadı:.....

İmzası:.....

Telefon No :.....

Tarih (gün/ay/yıl):...../...../.....

APPENDIX O

PERMISSION LETTER FOR STUDENTS

BOĞAZİÇİ ÜNİVERSİTESİ
Sosyal ve Beşeri Bilimler İnsan Araştırmaları Etik Kurulu
KATILIMCI BİLGİ ve ONAM FORMU

Araştırmayı destekleyen kurum: Boğaziçi Üniversitesi
Araştırmanın adı: Uzaktan eğitim yoluyla ortaokulda bilgi işlemsel düşünme becerilerinin geliştirilmesi
Proje Yürütücüsü: Günizi Kartal

E-mail adresi:

Telefonu:

Araştırmacının adı: Elif Çil

Adresi:

E-mail adresi:

Telefonu:

Değerli Veliler ve Öğrenciler,

Eğitim Teknolojisi yüksek lisans öğrencisi olarak 6.sınıf öğrencilerinin bilgi işlemsel düşünme becerilerini geliştirmek için bilimsel bir araştırma projesi yürütüyorum. Sizleri bu çalışmaya katılmak için davet ediyorum.

Proje konusu: Bu araştırma projesinin amacı 6. sınıf öğrencilerinin uzaktan erişimle katılabileceği bir eğitim programıyla bilgisayar bilimi alanına merak uyandırmak ve bilgi işlemsel düşünme becerilerini geliştirmektir.

Onam:

Çalışmaya katılmayı kabul ettiğiniz takdirde öğrencilerden bilgi işlemsel düşünmeyle ilgili bir anket doldurmanızı isteyeceğim. Ayrıca bu konuda test sorularını cevaplamanızı isteyeceğim. Bu ölçekler toplamda 35-40 dakikanızı alacaktır. Aynı ölçekleri uzaktan eğitim programının sonunda tekrarlayacağız. Program sonunda öğrencilerden bir de geri bildirim anketi doldurmalarını isteyeceğim.

Bilgi işlemsel düşünmeye yönelik anket öğrencilerinizin bilgisayar bilimiyle ilgili fikirlerini öğrenmek için hazırlanmış 39 maddeden oluşmaktadır. “Evet”, “Kısmen”, “Hayır” seçeneklerinden uygun olanı işaretleyerek anketi tamamlayacaksınız. Bilgi işlemsel düşünme testinde ise doğru şıkları işaretlemeniz beklenmektedir. Buradaki soruların okul başarınızla ya da ders notlarınızla bir ilgisi yoktur. Not kaygısı olmadan cevaplamanızı bekliyorum.

Çalışmaya katılmanız tamamen isteğe bağlıdır. Katılımcı bilgilerinin gizliliği esastır ve isminiz tamamen gizli tutulacaktır. Katıldığınız takdirde çalışmanın herhangi bir aşamasında sebep göstermeden onayınızı çekebilirsiniz.

Araştırmanın size herhangi bir risk getirmesi kesinlikle beklenmiyor. Araştırma çerçevesinde toplanan verilerden ulaşılan sonuçlar eğitimle ilgili bilimsel konferanslarda sunulabilir, yayınlanabilir.

Bu formu imzalamadan önce çalışmayla ilgili sorularınız varsa sormaktan çekinmeyin. Daha sonra bilgi almak istediğiniz takdirde numaralı telefondan benimle iletişime geçebilirsiniz. Araştırma projesi hakkında ek bilgi almak için tez danışmanım Boğaziçi Üniversitesi Bilgisayar ve Öğretim Teknolojileri Bölümü Öğr.

Üyesi Dr. Günizi Kartal ile temasa geçiniz. Ayrıca araştırma çalışması ile ilgili haklarınız konusunda Boğaziçi Üniversitesi Sosyal ve Beşeri Bilimler Yüksek Lisans ve Doktora Tezleri Etik İnceleme Komisyonu ile görüşebilirsiniz.

Yukarıdaki açıklamaları okudum ve anladım. Çalışmaya katılmayı kabul ediyorum.
Katılımcı Adı-Soyadı:……..
İmzası: ……
Tarih (gün/ay/yıl):...../...../.....

VELİSİNİN Adı-Soyadı:.....

İmzası:.....

Tarih (gün/ay/yıl):...../...../.....

Araştırmacının Adı- Soyadı:
.....
İmzası:.....
.....
Tarih (gün/ay/yıl): ..... / …… / ……

APPENDIX P

UNPLUGGED PROGRAMMING LESSON PLANS

Weeks	Name of the activity	Learning outcomes	What did students do?
1st Week 1st and 2nd Lesson	Maze in the house and Secret message	Write a daily life algorithm Develop students' algorithmic thinking skills. Solve an algorithm that include decision making	-draw a maze of their house -write an algorithm to get from the living room to the kitchen -write an algorithm to get from the bedroom to the kitchen -find the longest and shortest path in the maze -find the secret password given by color coding
1st Week 3rd and 4th Lesson	Caesar Cipher	Develop students' abstraction skills	-watch a video about Caesar's cipher -answer questions about the video -decipher examples encrypted with the Caesar cipher -decrypt examples encrypted with emojis and geometric shapes
2[2d] Week 1st and 2nd Lesson	Line it up and complete it!	Develop students' algorithmic thinking and decomposition skills. Write a daily life algorithm Gives daily life algorithm examples	-find the shapes that create a given house -construct an algorithm to draw a given house -write a daily life algorithm -write an tooth brushing algorithm -Find pictures encrypted with numbers
2[2d] Week 3rd and 4th Lesson	Message in the mosaics	Develop students' decomposition skills Solve a given algorithm that include decision making	-search on the internet what pixel is -find hidden message by painting pixel by pixel
3rd Week 1st and 2nd Lesson	Surprise question		-watch 2 videos about what computer engineers do -answer 2 questions about videos -watch a video about what a computer scientist does -answer question about the video
3rd Week	Find the rule	Develop students'	-talk about the game and the

3rd and 4th Lesson	of the game, start coding	abstraction and generalization skills Write an algorithm that include decision making	rules of the game -write the rules of the game of hide and seek -write an algorithm to get the shapes created with paper cup game
4th Week 1st and 2nd Lesson	Bug hunting	Develop students' debugging skills Write a daily life algorithm Solve a problem that include loop structure	-talk about the plant growth process -find the given errors in the growing cycle of the tomato and correct the cycle -correct errors in activity related to weight -solve sudoku by finding the errors in the given sudoku
4th Week 3rd and 4th Lesson	Pattern in the tile	Develop students' generalization skills Solve a problem that include loop structure	-discuss the question of what is tile -complete the missing tile pattern -answer questions about the pattern completion process

APPENDIX R

BLOCK BASED PROGRAMMING (SCRATCH) LESSON PLANS

Weeks	Name of the activity	Learning outcomes	What did students do?
5th Week 3rd and 4th Lesson	Program the Cat	Recognize the interface and features of the Scratch programming tool. Explain the functions of the presented code blocks in the Scratch.	-reach the glass on the table with their eyes closed -guess how many seconds passed while saying their names, and confirm whether their guesses were correct by using a stopwatch -introduce the interface of the Scratch by using the theater analogy. -move the cat 10 steps
6th Week 1st and 2nd Lesson	Exploration Hunt	Explain the functions of a program presented in the Scratch. Develop and organize a project in Scratch according to given criteria.	-select 10 code blocks -answer 7 questions to proceed -use appropriate code blocks to create small projects
6th Week 3rd and 4th Lesson	Create an animation	Debug a program presented in Scratch. Develop and organize a desired program in Scratch according to the given criteria.	-watch a video about animations prepared by Pixar -create an algorithm to fly a parrot
7th Week 1st and 2nd Lesson	Save the chicklet	Develop and organize a desired program in Scratch according to the given criteria.	-draw a sprite to create a fox hut -create an animation to move a chicklet with arrow keys
7th Week 3rd and 4th Lesson	Talking Objects	Create programs that contain logic structure. Test and debug programs that contain logic structure.	-answer questions about what is conversation -change to the properties of the sprites (horizontal flip feature) -use code blocks to make the characters talk to each other.
8th Week 1st and 2nd Lesson	Be Careful	Create programs that include decision structure. Create programs with multiple decision structures.	-discuss how we move without hitting the objects in our daily life -create animation about the meeting of 2 characters in the forest

8th Week 3rd and 4th Lesson	I'm coding the game	Select the most appropriate decision structures to adapt an algorithm.	-talk about the rules needed to create a game. -determine the rules of the game were determined and the features that should be in the games were listed -create a game as a project

APPENDIX S

AN EXAMPLE ACTIVITY

Line it up and complete it! (4th activity)

In this activity, students were expected to learn and develop algorithms. Before the mentor explained the definition of "algorithm" to the students, the lesson started with an activity for the students to better understand the definition of algorithm. At the beginning of the lesson, the mentor showed the students a simple picture of a house he had drawn.

Then he asked 6 questions about this house drawing. Thus, the students did an analysis by breaking the shape into parts, and then they created an algorithm to draw a house themselves. The mentor said the students that all the steps they follow to draw a simple house are called algorithms. Then, mentor gave examples of algorithms from daily life so that students could better understand them, such as brewing tea, brushing teeth. Then, the students were asked to write an algorithm for something they do in daily life. After examining the examples of the students, a tooth brushing algorithm was written in the classroom so that the students could understand it better. Then, the mentor wrote 2 algorithms with more detailed and simpler steps and explained to the students that the same result could be achieved with different steps. The mentor showed the students a video about brushing teeth in

space and asked the students to write an algorithm for brushing teeth in space. After the main activity, a reinforcement activity was also carried out. The students were asked to take a sheet from the math notebook. Then, the rules of the activity were explained to the students.

1. The first number in each line indicates the number of squares you should leave blank on the squared paper, and the second number indicates the number of squares you need to scribble.
2. The first thing you should always do is leave the squares blank. So, if the 1st digit is 2 for example, you need to leave 2 squares blank. After leaving 2 squares blank, you should draw as many squares as the 2nd digit says. For example, if the 2nd digit is 7, we should scribble the 7 squares after the first 2 squares.

Resim 1	**Resim 3**
2, 7 2, 7 2, 1, 1, 3, 1, 1, 1, 1 0, 11 0, 1, 1, 7 3, 1, 3, 1 2, 2, 3, 2	2, 1 2, 5, 5, 4 3, 5, 3, 6 2, 7, 1, 8 4, 5, 1, 4, 2, 2 3, 11, 1, 3 4, 15 5, 12 7, 8 0, 16 1, 16 2, 15 1, 5, 6, 6 3, 2, 7, 8 3, 1, 9, 1, 1, 3 15, 1, 1, 2

Resim 2	**Resim 4**
5, 6 4, 1, 6, 1 3, 1, 7, 2 2, 2, 9, 1 1, 1, 12, 1 0, 1, 4, 1, 4, 1, 4, 1 0, 1, 14, 1 0, 1, 14, 1 0, 1, 3, 1, 6, 1, 3, 1 1, 1, 3, 1, 4, 1, 3, 1 2, 1, 3, 4, 4, 1 2, 1, 10, 1 3, 1, 8, 1 4, 2, 4, 2 6, 4	6, 3 5, 5 4, 7 3, 9 3, 9 3, 3, 3, 3 3, 3, 3, 3 3, 3, 3, 3 3, 9 2, 11 1, 13 0, 15 0, 15 0, 15 0, 2, 1, 9, 1, 2 0, 2, 1, 9, 1, 2 0, 1, 2, 9, 2, 1 0, 1, 3, 8, 2, 1 0, 1, 4, 5 6, 3

At the end of the lesson, the video was watched to examine the robots that appeared in the hidden pictures and the mentor concluded the lesson by saying that algorithms are needed for robots to move.

REFERENCES

Ackermann, E. (2001). Piaget's Constructivism, Papert's Constructionism: *What's the difference?*

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). International Forum of Educational Technology & Society A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge. Source: *Journal of Educational Technology & Society, 19*(3), 47–57.

Amnouychokanant, V., Boonlue, S., Chuathong, S., & Tamwipat, K. (2021). Online learning using block-based programming to foster computational thinking abilities during the COVID-19 pandemic. *International Journal of EmergingTechnologies in Learning (iJET), 16*(13), 227–247.

Aranda, G., & Ferguson, J.P. (2018). Unplugged Programming: The future of teaching computational thinking?

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*, Part B, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Aytaç, T. (2021). The Problems Faced by Teachers in Turkey During the COVID-19 Pandemic and Their Opinions. *International Journal of Progressive Education, 17,* 2021. https://doi.org/10.29329/ijpe.2020.329.26

Barefoot, C. A. S. (2014). Computational Thinking. Retrieved from http://barefootcas.org.uk/barefootprimary-computing-resources/concepts/computational-thinking

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K 12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2* (1), 48 54.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology, 13*(1), 20-29.

Bell, T., & Vahrenhold, J. (2018). CS Unplugged—How is it used, and does it work? In H. J. Böckenhauer, D. Komm, & U. W. (Eds.), Adventures between lower bounds and higher altitudes. Cham: Springer. doi.org/10.1007/978-3-319-98355-4_29

Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning, 1*(2), 65-81.

Bers, M. U. (2008). Blocks to robots: *Learning with technology in the early childhood classroom.* New York, NY: Teachers College Press.

Bers, M.U. (2010). The TangibleK Robotics program: Applied computational thinking for young children. *Early Childhood Research & Practice*, *12*(2).

Bers, M. U. (2018). *Coding as a playground: Programming and computational thinking in the early childhood classroom.* New York, NY: Routledge. https://doi.org/10.4324/9781315398945

Bers, M. U., & Sullivan, A. (2019). Computer science education in early childhood: The case of scratchjr. *Journal of Information Technology Education: Innovations in Practice, 18,* 113–138. https://doi.org/10.28945/443710.28945/4437

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education-Implications for policy and practice (No. JRC104188).* Luxembourg: Publications Office of the European Union. Retrieved from http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf.

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M., Jasutė, E., Malagoli, C., Masiulionytė-Dagienė, V., & Stupurienė, G. (2021). State of play and practices from computing education reviewing computational thinking in compulsory education.

Booth W. A., (2013). Mixed-methods study of the impact of a computational thinking course on student attitudes about technology and computation. Unpublished doctoral dissertation, Baylor University, Texas USA.

Brackmann, C. P., Moreno-León, J., Román-González, M., Casali, A., Robles, G., & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. *ACM International Conference Proceeding Series,* (January 2018), 65–72. https://doi.org/10.1145/3137065.3137069

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceeding of the 2012 annual meeting of the American Educational Research Association,* Vancouver, Canada.

Brown, W. (2015). Introduction to algorithmic thinking. Retrieved from https://raptor.martincarlisle.com/Introduction%20to%20Algorithmic%20Thinking.doc

Cansu, F. K., & Cansu, S. K. (2019). An Overview of Computational Thinking. *International Journal of Computer Science Education in Schools*, *3*(1), 17. https://doi.org/10.21585/ijcses.v3i1.53

Creswell. J. W. (2006). Qualitative inquiry and research design: Choosing among five approaches. London: Sage Publications.

Creswell, J. W., & Plano Clark, V. L. (2011). Designing and conducting mixed methods research. Thousand Oaks, CA: Sage.

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational Thinking a Guide for Teachers. 02.01.2019,* http://eprints.soton.ac.uk/id/eprint/424545

Csernoch, M., Biro, P., & Math, J. (2021). Developing Computational Thinking Skills with Algorithm-Driven Spreadsheeting. *IEEE Access, 9,* 153943–153959. https://doi.org/10.1109/ACCESS.2021.3126757

Code.org (2019). Curriculum Overview Available at: https://code.org/curriculum/docs/k-5/overview

Computer Science Teachers Association. (2017). Progression of Computer Science Teachers Association (CSTA) K-12 Computer Science Standards, Revised 2017. *Csta, 3017.* Retrieved from https://www.csteachers.org/page/standards

CSTA & ISTE. (2011). Operational definition of computational thinking for K-12 education. Available at: https://cdn.iste.org/www-root/ct-documents/computational-thinking-operational-definition-flyer.pdf

Curzon, P., Selby, C., & Woollard, J. (2014). *Developing computational thinking in the classroom: a framework!* http://www.digitalschoolhouse.org.uk

Çatlak, Ş., Tekdal, M., & Baz F.Ç. (2015). Scratch yazılımı ile programlama öğretiminin durumu: Bir doküman inceleme çalışması. *Journal of Instructional Technologies & Teacher Education, 4*(3), 13-25.

Delal, H., & Oner, D. (2020). Developing middle school students' computational thinking skills using unplugged computing activities. *Informatics in Education,19*(1), 1–13. https://doi.org/10.15388/infedu.2020.01

Denner, J., Werner, L. & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240-249. Elsevier Ltd. Retrieved May 10, 2022, from https://www.learntechlib.org/p/50683/.

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should your 8-year-old learn coding? *In Proceedings of the 9th Workshop in Primary and Secondary Computing Education,* (pp. 60–69). Berlin, Germany: ACM.

Erdem E. (2018). *Blok Tabanlı Ortamlarda Programlama Öğretimi Sürecinde Farklı Öğretim Stratejilerinin Çeşitli Değişkenler Açısından İncelenmesi.* Yüksek Lisans Tezi. Başkent Üniversitesi. Eğitim Bilimleri Enstitüsü. Ankara.

Eşici, H., Ayaz, A., Yetim, D., Çağlar, S., & Bedir, N. (2021). Teachers in COVID-19 period: Psychological effects, practices and career needs. *Turkish Journal of Education,* 157–177. https://doi.org/10.19128/turje.855185

Faber, H. H., Wierdsma, M. D. M., Doornbos, R. P., & van der Ven, J. S. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. *Journal of the European Teacher Education Network, 12*, 13–24.

Futschek, G. (2006). Algorithmic Thinking: *The Key for Understanding Computer Science. In R.T. Mittermeir (Ed.),* ISSEP 2006, LNCS 4226, (pp. 159–168), Berlin: Springer.

Gaio, A. (2018). Programming for 3 rd graders, Scratch-based or unplugged? To cite this version: HAL Id: hal-01950502.

Good, J., Sands, P. & Yadav, A. (2018). Chapter 8: Computational Thinking in K-12: Inservice Teacher Perceptions of Computational Thinking. *Computational Thinking in the STEM Disciplines.* (978-3-319-93566-9), 151-164.

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, Stanford, CA, United States). Retrieved from Stanford Digital Repository.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S, R Pea and S Cooper (2015). 'Designing for Deeper Learning in a Blended Computer Science Course for Middle School Students' *25*(2) Computer Science Education 199–237.

Grover, S., Jackiw, N. & Lundh, P. (2019). Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school, *Computer Science Education, 29*(2-3), 106-135. doi.org/10.1080/08993408.2019.1568955

Gülbahar, Y., Kalelioğlu, F. (2018). BİLİŞİM TEKNOLOJİLERİ VE BİLGİSAYAR BİLİMİ: ÖĞRETİM PROGRAMI GÜNCELLEME SÜRECİ. *Milli Eğitim Dergisi, 47* (217), 5-23. Retrieved from https://dergipark.org.tr/tr/pub/milliegitim/issue/39632/462315

Gülbahar, Y., Kalelioğlu, F., Doğan, D. & Karataş, E. (2020). Bilge Kunduz: Enformatik ve bilgi-işlemsel düşünmeyi kavram temelli öğrenme için toplumsal bir yaklaşım. *Ankara Üniversitesi Eğitim Bilimleri Fakültesi Dergisi, 53*(1), 241 – 272. Erişim adresi: https://dergipark.org.tr/tr/download/article-file/1032946.

Gülbahar, Y., Kert, S. B. ve Kalelioğlu F. (2018). Bilgi Işlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı Ölçeği (BİDBÖA): Geçerlik Ve Güvenirlik Çalışması. *Türk Bilgisayar ve Matematik Eğitimi Dergisi.* Advance online publication. doi:10.16949/turkbilmat.385097

ISTE (International Society for Technology Education) (2021). Computational Thinking Competencies. Retrieved from https://cdn.iste.org/standards/computational-thinking

İbili, E., & Günbatar, M. S. (2020). Computational thinking skills self-efficacy perceptions in secondary education: A review of the effectiveness of the new information technology and software curriculum. *Trakya Journal of Education, 10*(2), 303-316

Kalelioğlu, F. (2018). Bilgisayarsız Bilgisayar Bilimi (B3) Öğretimi. In Y. Gülbahar (Eds.), *Bilgi işlemsel düşünmeden programlamaya kitabı* (pp. 183-204). Ankara, Turkey: PEGEM Akademi.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior, 52*(2015), 200-210.

Kalelioğlu, F, Doğan, D. & Gülbahar, Y. (2021). Snapshot of Computational Thinking in Turkey: A Critique of 2019 Bebras Challenge. *Informatics in Education.* https://doi.org/10.15388/infedu.2022.19

Kalelioğlu, F. & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in Education, 13*(1), 33-50.

Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A Framework for Computational Thinking Based on a Systematic Research Review. *Baltic Journal of Modern Computing, 4*(3), 583.

Kamal, T., & Illiyan, A. (2021). School teachers' perception and challenges towards online teaching during COVID-19 pandemic in India: an econometric analysis. *Asian Association of Open Universities Journal, 16*(3), 311–325. https://doi.org/10.1108/aaouj-10-2021-0122

Korkmaz, Ö., Çakır, R., & Özden, M. Y. (2015). Bilgisayarca düşünme beceri düzeyleri ölçeğinin (bdbd) ortaokul düzeyine uyarlanması. *Gazi Eğitim Bilimleri Dergisi, 1*(2), 143-162.

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. https://doi.org/10.1016/j.chb.2017.01.005

Liskov, B., & Guttag, J. (1986). Abstraction and Specification in Program Development. Cambridge, MA: MIT Press.

Liskov, B., & Guttag, J., (2000). Program development in java: Abstraction, specification, and object-oriented design (1st ed.). Boston, MA: Addison-Wesley.

Lockwood, E., & Dejarnette, A. F. (2017). Algorithmic thinking: An initial characterization of computational thinking in mathematics Investigating Student Learning and Sense-Making from Instructional Calculus Videos View project. (November). Retrieved from https://www.researchgate.net/publication/317570108

Lodi, M. (2020). Introducing Computational Thinking in K-12 Education: *Historical, Epistemological, Pedagogical, Cognitive, and Affective Aspects. Computers and Society [cs.CY]. Dipartimento di Informatica - Scienza e Ingegneria, Alma Mater Studiorum - Università di Bologna, 2020. English. ffftel-02981951f*

Looi, C. K., How, M. L., Wu, L., Seow, P., & Liu, L. (2018). Analysis of linkages between an unplugged activity and the development of computational thinking. *Journal of Computer Science Education* (JCSE). doi.org/10.1080/08993408.2018.1533297

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking about computational thinking. *SIGCSE Bulletin Inroads, 41*(1), 260–264. https://doi.org/10.1145/1539024.1508959

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*(2014), 51-61.

Miles, M. B., & Huberman, A. M. (1994). Qualitative Data Analysis: An Expanded Sourcebook. Thousand Oaks, CA: Sage Publications.

Milli Eğitim Bakanlığı (2018). Bilişim teknolojileri ve yazılım dersi öğretim programı (ortaokul 5. ve 6. sınıflar). Retrieved from http://mufredat.meb.gov.tr/Dosyalar/2018124103559587-Bili%C5%9Fim%20Teknolojileri%20ve%20Yaz%C4%B1l%C4%B1m%205-6.%20S%C4%B1n%C4%B1flar.pdf

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *SIGCSE Bulletin, 40, 367*–371.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., et al. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education, 18*(2), 67–92.

Moreno León, J., Robles, G., & Román González, M. (2015). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *RED. Revista de Educación a Distancia,* (46), 1–23.

National Research Council. (2010). Committee for the workshops on computational thinking: Report of a workshop on the scope and nature of computational thinking. Washington, DC: National Academy doi: 10.17226/12840

Oluk, A., Korkmaz, Ö., & Oluk, H. A. (2018). Effect of scratch on 5th graders' algorithm development and computational thinking skills. *Turkish Journal of Computer and Mathematics Education, 9*(1), 54–71. https://doi.org/10.16949/turkbilmat.399588

Ozer, F. (2019). *Kodlama eğitiminde robot kullanımınnn ortaokul öğrencilerinin erişi, motivasyon ve problem çözme becerilerine etkisi.* Unpublished master's thesis, Hacettepe University.

Papadakis, S. J., Kalogiannakis, M., & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: *A case study Developing fundamental programming concepts and computational thinking with ScratchJr in preschool education: a case study Stamatios Papadakis *, Michail Kalogiannakis and Nicholas Zaranis*. (July). https://doi.org/10.1504/IJMLO.2016.077867

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York, NY: Basic Books.

Relkin, E., de Ruiter, L. E., & Bers, M. U. (2021). Learning to code and the acquisition of computational thinking by young children. *Computers and Education, 169.* https://doi.org/10.1016/j.compedu.2021.104222

Relkin, E., de Ruiter, L. E., & Bers, M. U. (2020). TechCheck: Development and Validation of an Unplugged Assessment of Computational Thinking in Early Childhood Education. *Journal of Science Education and Technology*, *29*(4), 482–498. https://doi.org/10.1007/s10956-020-09831-x

Repenning, A., Webb, D. & Ioannidou, A., 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on computer science education.* pp. 265–269.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

Resnick, M. (2013). *Learn to code, code to learn: How programming prepares kids for more than math.* EdSurge (May 8, 2013). Retrieved from https://www.edsurge.com/n/2013-05-08-learn-to-code-code-to-learn

Rich, K. M., Spaepen, E., Strickland, C., & Moran, C. (2019). Synergies and differences in mathematical and computational thinking: *Implications for integrated instruction. Interactive Learning Environments, 28*(3), 272-283.

Rich, K. M., Yadav, A., & Schwarz, C. V. (2019). Computational Thinking, Mathematics, and Science: Elementary Teachers' Perspectives on Integration. *Journal of Technology and Teacher Education, 27*(2), 165-205.

Rich, P. J., Egan, G., & Ellsworth, J. (2019). A framework for decomposition in computational thinking. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, (August), 416–421. https://doi.org/10.1145/3304221.3319793

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2016). Assessing computational thinking in CS unplugged activities. Proceedings of the *Conference on Integrating Technology into Computer Science Education, ITiCSE,* 501–506. https://doi.org/10.1145/3017680.3017779

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior, 72,* 678-691.

Sayın, Z. (2018). Bilgisayar bilimi eğitimi kapsamı. In Y. Gülbahar (Eds.), *Bilgi işlemsel düşünmeden programlaya kitabı* (pp. 133-153). Ankara, Turkey: PEGEM Akademi.

Seiter, L., Foreman, B., 2013, August. Modeling the learning progressions of computational thinking of primary grade students. In: *Proceedings of the ninth annual international ACM conference on international computing education research. ACM*, pp. 59-66

Selby, C. and W., John (2013) Computational thinking: the developing definition University of Southampton (E-prints) 6pp.

Serin, O., Serin, N. B., & Saygılı, G. (2010). İlköğretim düzeyindeki çocuklar için problem çözme envanteri'nin (ÇPÇE) geliştirilmesi. *İlköğretim Online, 9*(2).

Seow, P., Looi, C.-K., How, M.-L., Wadhwa, B., & Wu, L.-K. (2019). Educational policy and implemen-tation of computational thinking and programming: Case study of Singapore. In S.-C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 345–361). Springer Singapore.

Sezer, H. B., & Namukasa, I. K. (2021). Real-world problems through computational thinking tools and concepts: the case of coronavirus disease (COVID-19). *Journal of Research in Innovative Teaching & Learning, 14*(1), 46–64. https://doi.org/10.1108/jrit-12-2020-0085

Shute, V., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22,* 142-158.

Song, J. B. (2019) The Effectiveness of an Unplugged Coding Education System that Enables Coding Education without Computers. *Universal Journal of Educational Research 7.5A* 129 - 137. doi:10.13189/ujer.2019.071514.

Takaoka, E., Fukushima, Y., Hirose, K., & Hasegawa, T. (2014). learning based on computer science unplugged in computer science education: Design, Development, and Assessment.

Taub, R., Ben-Ari, M., & Armoni, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education, 12*(2), 1-29.

Tebliğler Dergisi (1997-2013). T.C. MEB Tebliğler Dergisi. http://tebligler.meb.gov.tr/

Teaching Computing (2020). How we teach computing. Retrived from https://blog.teachcomputing.org/how-we-teach-computing/

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017). Training computational thinking: game-based unplugged and plugged-in activities in primary school.

Tosun, N., Mıhcı, C., & Bayzan, Ş. (2021). Challenges encountered by in-service k12 teachers at the beginning of the covid-19 pandemic period: The case of turkey. *Participatory Educational Research, 8*(4), 359–384. https://doi.org/10.17275/per.21.95.8.4

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies, 20*(4), 715-728. doi.org/10.1007/s10639-015-9412-6

Yadav, A., Stephenson, C. & Hong, H., 2017, Computational Thinking for Teacher Education, *Communications of the ACM, 60* (4), 55-62.

Yavuz Mumcu, H., & Yıldız, S. (2018). The investigation of algorithmic thinking skills of fifth and sixth graders at a theoretical dimension. *MATDER Journal of Mathematics Education, 3*(1), 41-48.

Zhang, L. C., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers and Education, 141*(June). https://doi.org/10.1016/j.compedu.2019.103607

Weese, J. L. (2017). *Bringing computational thinking to K-12 and higher education* (Doctoral dissertation, Kansas State University, Manhattan, Kansas).

Weigend, M., Vaníček, J., Pluhár, Z., & Pesek, I. (2019). Computational thinking education through creative unplugged activities.

Weinberg, A. E. (2013). Computational thinking: An investigation of the existing scholarship and research. (Unpublished Doctoral Thesis), Colorado State University, School of Education, Colorado.

Weintrop, D., & Wilensky, U. (2013). Robobuilder: a computational thinking game. SIGCSE.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-36.

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: *Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wong, G. K. W., & Jiang, S. (2019). Computational Thinking Education for Children: Algorithmic Thinking and Debugging., 328–334. https://doi.org/10.1109/TALE.2018.861523