THE EFFECTS OF SERIOUS GAMES ON STUDENTS' CONCEPTUAL KNOWLEDGE OF OBJECT-ORIENTED PROGRAMMING AND COMPUTATIONAL THINKING SKILLS

ALİ AKKAYA

BOĞAZİÇİ UNIVERSITY

2018

THE EFFECTS OF SERIOUS GAMES ON STUDENTS' CONCEPTUAL KNOWLEDGE OF OBJECT-ORIENTED PROGRAMMING AND 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

Ali Akkaya

Boğaziçi University

2018

The Effects of Serious Games on Students'

Conceptual Knowledge of Object-Oriented Programming

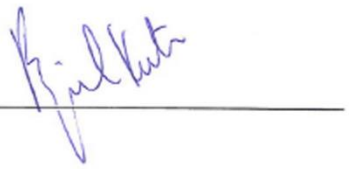
and Computational Thinking Skills

The thesis of Ali Akkaya

has been approved by:

Prof. Yavuz Akpınar
(Thesis Advisor)

Prof. Birgül Kutlu Bayraktar



Assist. Prof. Yavuz Samur
(External Member)

May 2018

DECLARATION OF ORIGINALITY

I, Ali Akkaya, 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 25.05.2018

ABSTRACT

The Effects of Serious Games on Students' Conceptual Knowledge
of Object-Oriented Programming and Computational Thinking Skills

The aim of this study is to investigate the effects of a serious game, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard), on undergraduate students' learning performance on conceptual knowledge of object-oriented programming (OOP) of and computational thinking (CT) skills in Turkish. The study was conducted with a pre-test and post-test quasi-experimental design model. Data were collected from 30 freshman students without programming experience and 31 sophomore students with procedural programming experience. Each student took a creative problem-solving test and a pre-test before playing the game. After playing the developed game, students took a post-test and an attitude scale for serious game assisted programming learning. Analyses showed that the game helped students to develop conceptual knowledge of OOP and improve their CT skills. Analysis also showed that there were no significant two-way or direct interactions among students' creative problem-solving skills, attitudes towards digital game-based learning of programming on the achievement scores of students. This study makes a significant contribution to the literature by providing empirical data about the effects of serious games on novice programmers' conceptual knowledge of OOP and CT skills. It is thought that in the lights of the findings of the study, serious game designers and instructors would have the opportunity to design effective games that help novice programmers to overcome their learning difficulties and improve their learning.

ÖZET

Eğitsel Oyunların Öğrencilerin Nesne Tabanlı Programlamanın Temel Kavramsal Bilgisi ve Bilgi İşlemsel Düşünme Becerilerine Etkisi

Bu çalışmanın amacı Türkçe dilinde geliştirilmiş olan Meraklı Robotlar: Operasyon Asgard isimli eğitsel oyunun üniversite öğrencilerinin nesne tabanlı programlamanın temel kavramsal bilgisi ve bilgi işlemsel becerileri üzerine olan etkisini incelemektir. Çalışma yarı-deneysel öntest – sontest araştırma deseninde tasarlanmıştır. Veriler Boğaziçi Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümündeki daha önce hiç programlama tecrübesi olmayan 30 birinci sınıf öğrencisi ve temel programlama eğitimini C dilinde almış olan 31 ikinci sınıf öğrencisinden toplanmıştır. İlk olarak, öğrencilere yaratıcı problem çözme testi ve nesne tabanlı programlama ve bilgi işlemsel düşünme becerilerini ölçen ön test uygulanmıştır. Öğrenciler ön testleri tamamladıktan sonra geliştirilen oyunu oynamışlardır. Sonrasında öğrencilere ilk testteki sırası değiştirilmiş sorulardan oluşan nesne tabanlı programlama ve bilgi işlemsel düşünme son testi ve eğitsel bilgisayar oyunları destekli kodlama öğrenimine yönelik tutum ölçeği uygulanmıştır. Yapılan istatistiksel veri analizleri, geliştirilen oyunun her iki öğrenci grubundaki öğrencilerin nesne tabanlı programlamanın temel kavramlarını öğrenmeye ve bilgi işlemsel düşünme becerilerini geliştirmeye yardımcı olduğunu göstermiştir. Ayrıca öğrencilerin yaratıcı problem çözme becerileri ve bilgisayar oyunu destekli programlama öğrenimine karşı tutumlarının nesne tabanlı programlamanın kavramsal bilgisi ve bilgi işlemsel becerileri üzerine birlikte ve ayrı ayrı etkisinin olmadığını göstermiştir. Bu çalışma sunduğu deneysel verilerle bilgisayar oyunu destekli programlama öğrenimi alanyazınına önemli bir katkıda bulunmuştur. Çalışmanın

bulguları vasıtasıyla eğitsel oyun geliştiricileri ve öğretim elemanlarının acemi programcıların problemlerini ortadan kaldırabilmelerine yardımcı olacak daha etkili eğitsel oyunlar geliştirebilmesine olanak sağlanması beklenmektedir.

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 thesis advisor, Prof. Yavuz Akpınar, for his commitment, motivation and meticulous guidance during my research study. Despite his heavy workload and limited time, it is his invaluable experience, ideas, and his continuous encouragement and belief in me that made this thesis possible.

I would like to express my greatest appreciation to each of the members of my thesis committee, Prof. Birgül Kutlu Bayraktar and Assist. Prof. Yavuz Samur, for their insightful comments, feedback, suggestions and their possitive attitudes that broadened my perspective and increased my motivation.

Moreover, I want to thank my friends and colleagues who made contributions to my research and made this thesis possible. First, I would like to offer my special thanks to my friend and colleague, Ekrem Kutbay, for his valuable feedbacks, suggestions and also for his support during the data collection phase of my research. I also want to thank my friends Yakup Adaklı, Hüseyin Demir and Barış Demirer for their valuable suggestions that improved the design of the game. Lastly, I am grateful to Oğuz Ak for his permission and support to conduct the experiments in his lecture.

Last but not least, I owe very special debt of gratitude to my beloved family: to my mother and father, Sırmalı and Mahmut Akkaya, to my older brother and sisters Adem, Elmas and Rükiye Akkaya all of whom were very supportive all the time. I, especially, owe a very important debt of gratitude to my nearest and dearest love, Gülhizar Bollu, who has always been there for me throughout this tough and tiring process, and had faith in me even when I had doubts about myself.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Motivation is the sine qua non of effective learning: nothing can stop a motivated learner (Prensky, 2003). For this reason, researchers put emphasis on developing learning environments that engage students and provide effective learning. Games, thanks to their motivating and engaging nature, started to be considered as learning environments by researchers (Prensky, 2003). The use of video games as a learning environment is one of the approaches that researchers have adopted. In the literature serious games are identified as video games which have educational goals and provide intriguing contexts (Gunter, Kenny, & Vick, 2008). Computer programming, on the other hand, is one of most important skills of today's world thanks to the industry 4.0 revolution. However, students who start to learn computer programming lose their motivation because of the monotonous lessons and the difficulties they encounter while learning to program. Hence, researchers developed tools that encouraged the digital game-based learning of computer programming.

Object-Oriented Programming (OOP) is the most popular programming approach, which has a broad range of use in many domains and is currently being taught in educational institutions (Kölling, 1999a). Students starting to learn computer programming are excited to develop applications and games or work on operating systems and web browsers. Unfortunately, programming courses include complex activities that can cause learning difficulties for novice programmers (Pellas, 2014), which results in demotivation. Some of the difficulties students face stem from the abstract nature of OOP, the distribution of control flows in OOP, the complexity of the syntax of programming languages and the programming

environments. Additionally, Computational Thinking (CT) skills are the fundamental skills that the lie at the bottom of students' problems. CT is a problem-solving approach in which solutions to problems are generated in a way that computers are able to perform (Wing, 2006). Students have problems in understanding and analyzing a problem, building step-by-step algorithmic solution designs for problems (Xinogalos, 2016) and visualizing the programming concepts from a problem situation (McCracken, et al., 2001).

Recently, researchers adopted a digital game-based learning approach to help novice programmers overcome their learning problems and increase their motivation for teaching computer programming. The current literature in digital game-based learning of programming has two focuses: using visual programming environments to teach programming and using serious games as learning environment to introduce programming learning through game-play experience.

Visual programming environments such as Scratch (Resnick, et al., 2009), Alice (Cooper, Dann, & Pausch, 2000), and Greenfoot (Kölling, 2010) were used by researchers to teach the basics of OOP and CT, and the findings of the research conducted on these programming environments were promising. Even though the visual programming environments had positive effects on students' learning performance, they still need to be used with well-designed teaching methods and learning materials (Meerbaum-Salant, Armoni, & Ben-Ari, 2013; Repenning, Webb, & Ioannidou, 2010). Moreover, these programming environments do not have a proper feedback mechanism to help novice programmers understand the errors in their algorithms (Meerbaum-Salant, Armoni, & Ben-Ari, 2011).

Using games as learning environment is the other main approach in the literature of digital game-based learning of programming. Jones (2000) states that

games are excellent examples of object-oriented environments if the aim is to teach object-orientation and programming. Similarly, Weintrop and Wilensky (2016) asserted that, with the integration of coding activities into game-play, students would become more familiar with fundamentals of programming. Consequently, many researchers studied the effects of serious games on learning computer programming in recent years. The majority of the findings in the literature agree on the positive effects of serious games on novice programmers' motivation (Barnes, et al., 2007; Liu, Cheng, & Huang, 2011; Mathrani, Christian, & Ponder-Sutton, 2016; Muratet, Torguet, Viallet, & Jessel, 2011; Ramírez-Rosales, Vázquez-Reyes, Villa-Cisneros, & De León-Sigg, 2016; Wong, Hayati, & Tan, 2016).

For this reason, instructors, educational technologists and serious game developers should develop appealing, meaningful and effective learning materials, contents and serious games to improve novice programmers' learning and help them overcome their difficulties.

1.1 Statement of the problem

Much of the research up to now has studied the effects of serious games on novice programmers' learning performance and motivation. However, there are still some crucial criticism of weaknesses in those studies.

Firstly, there is a need for research on the conceptual knowledge of OOP and CT skills because it is important for novice programmers to understand how problems are solved before learning specific programming languages (Liu, Cheng, & Huang, 2011). Hadjerrouit (1999) states that the fundamental concepts of OOP have a crucial role in understanding and analyzing problems and generating solutions to problems. CT, on the other hand, is a fundamental problem-solving skill in which the

solutions of problems are designed in a way that computers can perform effectively (Aho, 2012; Lu & Fletcher, 2009; Wing, 2006; Wing, 2008). According to McFarlane, Sparrowhawk, and Heald (2002), games can be helpful for students to enhance their critical thinking and problem-solving skills. Yet there has been little discussion about the effects of serious games on novice programmers' conceptual knowledge of OOP and CT skills.

Second, only a few studies provide a well-conceived experimental design and demonstrated inferential statistical analysis (Livovský & Porubän, 2014; Mathrani, Christian, & Ponder-Sutton, 2016; Miljanovic & Bradbury, 2017), even though there has been much research conducted on the topic. The majority of the findings in the literature are based either on anecdotal evidence or on initial evaluation results that do not provide substantial data on the learning performance of novice programmers. Furthermore, the majority of the studies conducted on the current issue are conducted with a limited number of participants, and the focal point of the studies was the motivation of students, not their learning performance (Barnes, Chaffin, Godwin, Powell, & Richter, 2007; Barnes, Richter, et al., 2007; Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012a; Livovský & Porubän, 2014; Ramírez-Rosales, et al., 2016).

Nevertheless, most of the research conducted on the current issue has been developed in English. Although there are a few studies examining the effects of serious games on novice programmers' performance and motivation using languages such as Spanish (Ramírez-Rosales, et al., 2016), there is not enough substantial data about the effects of serious games on students' programming skills in Turkish.

To conclude, by considering the findings of the studies in the current literature, it can be said that there has been little discussion about the effects of

serious games on novice programmers' learning performance on the conceptual knowledge of OOP and CT skills in Turkish.

1.2 Purpose of the study

The main purpose of this quasi-experimental study is to examine the effects of the use of a serious game which is in Turkish on undergraduate students' learning performance on conceptual knowledge of OOP of and CT skills.

1.3 Research questions

The current study is designed to answer the following research questions:

1. Is there any significant difference between post-test and pre-test scores on the conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience?
    a. Is there any significant difference between post-test and pre-test scores on conceptual knowledge of OOP of undergraduate students with no programming experience?
    b. Is there any significant difference between post-test and pre-test scores on CT skills of undergraduate students with no programming experience?
2. Is there any significant difference between post-test and pre-test scores on the conceptual knowledge of OOP and CT skills of undergraduate students with procedural programming experience?
    a. Is there any significant difference between post-test and pre-test scores on conceptual knowledge of OOP of undergraduate students with procedural programming experience?

b. Is there any significant difference between post-test and pre-test scores on CT skills of undergraduate students with procedural programming experience?

3. Is there any significant difference between the achievement scores on the conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience, and of undergraduate students with procedural programming experience?
4. To what extent do the students' creative problem-solving skills and attitudes towards digital game-based learning of programming influence achievement score on the conceptual knowledge of OOP and CT skills?

1.4 Significance of the study

A review of the literature shows that the majority of the research in this area is centered on the motivational effects of serious games on the learning of programming by novice programmers. There are a few studies that focus on the effects of serious games on improvement in students' learning from the proposed games. Moreover, there is not enough substantial data about the effects of serious games on novice programmers' programming skills in Turkish. The current study aims to contribute to the research area of the use of serious games to teach programming by providing empirical data on students' achievement scores on conceptual knowledge of OOP and computational thinking. The goal is to provide practical direction and knowledge for instructors, serious game designers, and educational technologists. With such knowledge, serious games for programming can be designed in a way that novice programmers have better and effective learning experience with high motivation.

1.5 Organization of the study

Chapter 2 introduces a literature review of digital game-based learning of programming, along with the problems of teaching and learning OOP, and the relationship between CT and OOP. Chapter 3 includes the research methodology: the design, participants, sampling procedures, developed serious game with the underlying theories, instruments, the data collection procedures and the data analysis procedures in detail. Chapter 4 contains the results of the data analyses. Finally, Chapter 5 focuses on the outcomes of the findings, the limitations of the study and recommendations for future research on the topic.

CHAPTER 2

REVIEW OF THE LITERATURE

2.1 Problems of teaching and learning programming

OOP is the most popular programming method that has a broad range of use, from education to different domains of industry, and almost every university in the computer science (CS) field covers OOP within its curriculum (Kölling, 1999a). Students engaging in the programming field for the first time are excited about the possibility of being the person to develop the next great action video game, or even work on an operating system or a Web browser. Although students are excited about learning to program, most of them soon realize that programming courses are monotonous and boring, and they start to lose their motivation and desire to learn to program (Prensky, 2003; Sarkar, 2006). Students' losing interest in learning OOP results in a decline in the number of the enrollments in CS courses in spite of the attempts to increase students' interest in CS (Ali & Shubra, 2010). There are several other reasons behind students' problems in learning OOP, and these reasons can be categorized based on the source of the problems. The problems emerge from the nature of the OOP, from fundamental problem-solving skills in computer programming, and from the programming environment.

The abstract nature of OOP itself causes problems for novice programmers (Kölling, 1999a; Xinogalos, 2016). For example, students have difficulties in understanding the nature and necessity of abstract concepts of OOP such as inheritance, polymorphism, overriding, abstract classes, and interfaces; hence, they cannot make use of these concepts properly (Xinogalos, 2016). Hadjerrouit (1999) emphasized the role of OOP concepts in understanding and analyzing problems,

generating solutions to problems, and in the implementation of the designed solutions of problems. Therefore, students' difficulties in understanding the fundamental concepts of OOP could give rise to much bigger problems.

The transition from procedural programming to OOP also causes problems for novice programmers in understanding OOP (Hadjerrouit, 1999). Students have difficulties in changing their mindset from focusing on defining special functions as a main way of forming a solution to a problem to trying to find a solution to a problem by creating functions that utilize classes and objects (Xinogalos, 2016). In other words, OOP requires one to generate functions making use of classes and objects for finding a solution to a problem, while procedural programming requires one to form a function that is special for that problem, and that is what poses a problem for students. Additionally, the distributed nature of control flows and functions of OOP is also what makes this transition a reason for the problems students come across. This difference in the nature of control flows and functions between OOP and procedural programming makes it harder for novice programmers to form mental representations of functions and control flows in OOP terms than in procedural programming (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999).

Another important problem that students encounter while learning OOP is the programming languages that they use while learning (Kölling, 1999a). Starting to learn OOP with a real programming language like C# or Java makes it harder for students to learn OOP than learning it via a pseudo language (Guzdial, 2008; Xinogalos, 2016). It is believed that using pseudo languages in teaching OOP helps students to focus on important aspects of OOP such as the algorithmic design of the

solution, OOP concepts and constructs instead of worrying about the syntax of a specific programming language (Xinogalos, 2016).

However, getting students not to worry about the syntax of a programming language and to focus on developing an algorithm for the solution does not solve all the problems. One of the reasons behind this situation is that students also have problems in developing algorithms and implementing these algorithms with a programming language (Xinogalos, 2016). In addition, novice programmers have difficulties figuring out what constructs to use and where to use them in their algorithms while they are programming (Kazimoglu, et al., 2012a; Soloway, 1986). Another reason behind students' difficulties in programming is that many students have problems in visualizing the execution of their programs. This problem of visualization prevents students from understanding what is wrong or what is missing in their program when their code is not executed as it is supposed to be (Cooper, Dann, & Pausch, 2000).

According to Kölling (1999b), another reason for the major problems of teaching OOP is the environments used. One of the main problems with these environments is the ease of use of the environment. Programming environments should not be very complicated so that novice learners will not have to worry about learning how to use the environment (Kölling, 1999b). Moreover, Watson, Li, and Lau (2011) claim that unspecific compiler messages cause difficulties for novice programmers because they need proper feedback from the compiler that guides them. Another problem about OOP is the lack of visual representation of classes. Kölling (1999b) suggests that programming environments should provide visual representation of classes and objects because it is difficult for students to think about problems in object-oriented terms when they first start learning OOP. The cost of

existing programming environments is another problem of teaching programming. According to Kölling (1999b), programming environments should be available at a lower cost, and they must also be compatible with computers with low-quality hardware.

2.2 Object-oriented programming concepts and computational thinking skills

Object-oriented programming is an approach where objects and interactions of objects in a real-world problem domain are modeled, and the production of a system via OOP is based on these objects and their interactions with the environment (Poo, Kiong, & Ashok, 2007). Abstract concepts such as class, objects, inheritance, encapsulation, polymorphism in OOP make it difficult for novice programmers to understand the nature of OOP. According to Hadjerrouit (1999), OOP concepts play an important role in understanding the problems, designing solutions to these problems and in the implementation of the proposed solution suggestions. In addition, students' CT skills such as conditional logic, algorithmic thinking, debugging and simulation are important factors in learning to program.

2.1.1 Object-oriented programming concepts

2.1.1.1 Objects and classes

Objects consist of methods that share a state and determine the calls to which the object can respond. The shared state of an object is hidden from the outside world and is accessible only to the object's operations (Wegner, 1990). Object attribute definitions enable objects to have different attribute values in each object instance (Poo, Kiong, & Ashok, 2007).

A class, on the other hand, is a definition which provides a structural template of an object and enables programmers to create instances of objects with the same attributes and methods (Poo, Kiong, & Ashok, 2007). One of the major differences between a class and an object is that classes are definitions of objects, and they do not have values in their attributes and methods. However, objects contain values in their attributes and methods because they are created as instances of a class (Poo, Kiong, & Ashok, 2007).

2.1.1.2 Methods

Methods are functions that include the behaviors and procedures that belong to a particular class. Communication between objects is done through a method call or a message with additional information called arguments from a message-sending object to a receiver object. Objects responds to messages only if they have a valid method which corresponds to the incoming method call, and the appropriate arguments of that call (Poo, Kiong, & Ashok, 2007).

2.1.1.3 Inheritance

Inheritance enables newly created classes to inherit the properties and the methods of previously defined classes. In other words, the properties of a superclass can be made available as part of the definition of its subclasses (Poo, Kiong, & Ashok, 2007). Thus, it enables programmers to make use of a behavior of a superclass in the newly created subclasses of it. Additionally, new attributes and methods can be added to these inherited classes (Wegner, 1990).

In OOP, classes can be in hierarchical relationships with other classes. Classes with more general attributes are called generalized classes, and they are

placed toward the top of the hierarchical relationship. Classes that have more specialized attributes are called specialized classes, and these are placed toward the bottom of the hierarchy. Specialized classes are a subclass of generalized classes, and generalized classes are superclass of subclasses (Poo, Kiong, & Ashok, 2007). The relationship between human beings and mammals is a good example of the inheritance concept. In this example, the mammals class is defined as having more general attributes, while human beings class have more special attributes along with inherited attributes and behavior of the mammals class.

2.1.1.4 Polymorphism

In procedural programming, it is not possible to have two methods with the same name, but this is possible in OOP. Polymorphism is the ability of different objects to perform a response to the same message (Poo, Kiong, & Ashok, 2007). For example if a programmer wants to define a function that calculates areas of different shapes, the programmer has to define different functions for each shape in procedural programming. In OOP, programmers can simply define multiple functions that have the same name (for example, calculateArea), but that performs different actions according to the inputs of methods such as triangle, square, rectangle, and so on.

2.1.1.5 Encapsulation

Defining attributes and methods of objects and hiding the implementation of these attributes and methods of an object from the users of objects — instances of other classes — is called encapsulation in OOP terms. In other words, the structure of an object and the implementation of its methods are not visible to other objects that interact with it. Other objects cannot directly access the attributes of an object, and

the data of an object can be manipulated through its public methods (Poo, Kiong, & Ashok, 2007).

2.1.2 Computational thinking

Computational thinking (CT) term has a long history in computer science, dating back to the 1950s and 1960s (Denning, 2009; Guzdial, 2008). However, CT became popular after Wing (2006) claimed that it is a fundamental skill which can be used by anyone in a variety of professions. Barr and Stephenson (2011) make a similar argument and claim that almost all of today's children will have a life that is influenced by computation, and some of these children will even have a profession related to computing. After the popularization of the term that resulted from Wing's claims in her seminal paper, researchers conducted studies to integrate CT into curricula (Perkovic, et al., 2010; Qualls & Sherrel, 2010). According to Guzdial (2008), teaching CT to people with different levels of background knowledge and different professions requires different approaches to learning. Therefore, understanding the nature of CT and skills that compose CT is important before integrating it into a curriculum and developing CT learning environments.

Although researchers aimed to integrate CT into a curriculum, there is not an agreed and clear definition of CT in the current literature (Guzdial, 2008; Berland & Lee, 2011). Various studies have aimed to construct a clear definition and the skill set that composes computational thinking (Perković, Settle, Hwang, & Jones, 2010). According to Wing (2006) CT is a problem-solving approach in which expressions of solutions of problems are presented in a way that a computer can perform them effectively. Wing (2006) claims that there are five main components of CT: conditional logic, distribution of a process, error debugging, simulation and building

algorithms. Dierbach et al. (2011) also proposed a model in which fundamental skills of CT are listed. These skills are decomposition of the problem, evaluating the problem, building algorithms, and developing computational solution methods for a problem. Similarly Berland and Lee (2011) asserted that CT involves five main concepts: conditional statements, building algorithms, debugging, simulation and distribution of a computation. According to Lee et al. (2011), defining, understanding and solving problems, abstraction, automation and analyzing the suitability of the abstractions define computational thinking. They also emphasize that the terms of abstraction, automation and analysis can be useful for understanding how young students make use of CT while solving problems.

Brennan and Resnick (2012) present a definition of CT that is specific to the Scratch visual programming environment; in this context, CT consists of three dimensions, namely, computational concepts, computational practices, and computational perspectives. Computational concepts are sequences, loops, parallelism, events, conditionals, operators and data. They claim that there are four main sets of computational thinking practices: being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing. The last dimension of computational thinking is the computational thinking perspective, which involves expressing oneself, connecting with others and questioning. Selby and Woollard's (2013) review of the literature on the definition of CT showed that there are three common terms in the definition of CT. These three terms are abstraction, decomposition and the concept of a thought process. Apart from these, there are other terms in the literature which are not supported by all researchers. Selby and Woollard listed these terms under four main categories: thinking terms, problem-solving terms, computer science terms and imitation terms. In the light of

their literature review, they claimed that CT involves problem-solving, abstraction, decomposition of a problem, algorithmic design, evaluation and generalization of the solution of a problem. A summary of CT skills are listed in Table 1.

Voogt, Fisser, Good, Mishra, and Yadav (2015) state that instead of propounding an exact definition of CT, researchers should try to find similarities and relationships among the definitions of CT that are made by different parties. Therefore, by considering all aforementioned studies, it can be said that the main skills which form CT are conditional logic, algorithmic thinking, debugging and simulation. Conditional logic is a problem-solving method in which logical thinking and different computational models are involved. Conditional logic includes the decomposition of the problem and evaluation of the problem to generate alternative representations (Berland & Lee, 2011). Algorithm building is the process of dividing the solution to a problem into step-by-step procedures. Selecting the most appropriate solution for a problem is important for the abstraction of the solution, because, in this way, the solution can be re-used in other problems (Barr, Harrison, & Conery, 2011). Debugging is the analysis of a solution to the problem and the process of correcting errors. The debugging process involves both critical and procedural thinking which makes it crucial for programming and computational thinking (Berland & Lee, 2011; Brennan & Resnick, 2012). Simulation is the demonstration of the solution to the problem, and the simulation skill includes the design and implementation of the solution based on the built algorithm (Basu, Dickes, Kinnebrew, Sengupta, & Biswas, 2013).

Table 1. CT Skillsets Defined in the Literature

Wing (2006)	Dierbach et al. (2011)	Berland & Lee (2011)	Lee et al. (2011)	Brennan & Resnick (2012)	Selby & Woollard (2013)
• Building algorithms • Conditional logic • Debugging • Distributed processing • Simulation	• Building algorithms • Decomposition of a problem • Developing computational solution methods • Evaluating a problem	• Building algorithms • Conditional logic • Debugging • Distributed computation • Simulation	• Abstraction • Analyzing the abstraction • Automation • Defining a problem • Understanding/Solving a problem	• Abstraction • Creative thinking • Debugging • Modularization • Reusing/Mixing • Testing	• Abstraction • Algorithmic design • Evaluation of a solution • Decomposition of a problem • Generalization • Problem-solving

2.1.3 The relationship between computational thinking and object-oriented programming

Contrary to the belief that CT and programming are the same, researchers have pointed out that these two terms are similar in nature but are not the same (Lu & Fletcher, 2009; Voogt, et al., 2015). Student problems in solving methods and skills in computer programming are identified as CT in the current literature (Aho, 2012; Lu & Fletcher, 2009; Wing, 2006). Hence, researchers put an emphasis on the early introduction and development of CT skills before students start to learn computer programming (Liu, Cheng, & Huang, 2011). In other words, it is important for novice programmers to understand how problems in a domain are solved conventionally before moving to understand how problems are solved in programming. Students' difficulties in learning computer programming are also connected to their problem-solving skills and to the computational learning environment where they learn programming (Gomes & Mendes, 2007). Therefore, it is critical to understand the relationship between CT and programming, especially OOP.

Problem-solving is one of the fundamental ways of having effective and meaningful learning (Jonassen, 2004). It is a way of thinking which has four main phases: understanding the problem, preparing a plan for the solution of the problem, implementing the proposed solution and reviewing the solution (Polya, 1957). It can be seen that Polya's definition of problem-solving is very similar to the CT skills that are defined in the literature. Lu and Fletcher (2009) have embraced Wing's idea about the importance of the CT, and they claim that CT should be taught to students before they start learning computer programming. They also claim that it is more crucial to understand the nature of CT rather than CT's manifestation in actual

programming languages because the integration of CT into a curriculum requires problem-solving skills. Therefore, activities involving CT should be introduced to students as early as possible (Qualls & Sherrel, 2010).

The idea of CT being a problem-solving approach is commonly held in the current literature, but CT is not limited to only a problem-solving method (Selby & Woollard, 2013). Along with being an approach to problem-solving, CT also involves skills such as abstract and logical thinking, decomposition, algorithm building, evaluating, and debugging. Computer science (CS), especially computer programming, is the field where the definition and the practice of CT emerged, yet it does not necessarily mean that CT requires the use of programming (Voogt, et al., 2015). Researchers point out that CT and computer programming do not mean the same thing (Lu & Fletcher, 2009; Voogt, et al., 2015), but computer programming and CT are intertwined concepts. In order to show the relationship between programming and CT, Kazimoglu (2013) used an onion metaphor which is a popular metaphor in the cognitive science community to represent the relationship between CT and computer programming (see Figure 1).

In the onion metaphor, there are three layers corresponding to the layers of abstractions. Machine coding is the core of the onion, which is referred as the physical layer. The layer on top of the core layer is the procedural layer where computer programming is located. The outmost layer of the onion metaphor is the conceptual or so-called operational layer in which CT is located. According to this model, novice programmers analyze a problem and design their solutions to the given problem at the CT layer. The implementation of novice programmers' solutions to a problem with a programming language occurs in the computer programming layer.

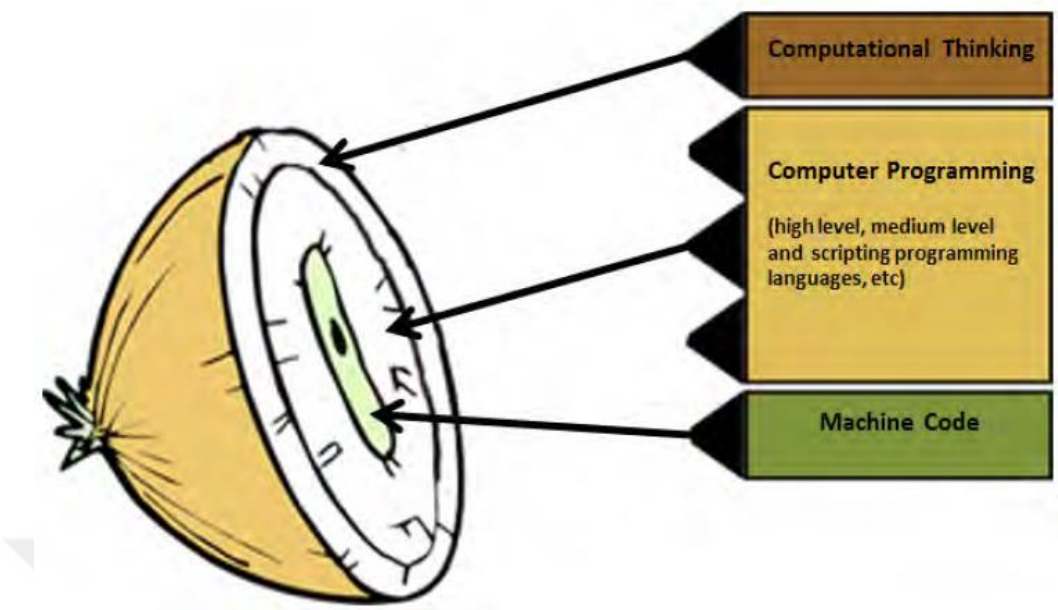


Figure 1. Layers of abstraction with onion metaphor

Source: Kazimoglu, 2013, p. 19

In the current literature of CT, researchers considered abstraction as the core concept of CT (Grover & Pea, 2013). Abstraction is the process of eliminating unnecessary details and selecting common and critical features and patterns of a problem to make a general representation of problems (Wing, 2008). Defining multiple layers of abstraction of problems and understanding the relationships among the different layers of abstractions are the basic practical facts of CT. Apart from being a key concept of CT, abstraction is also one of the fundamental features of OOP (Glasser, 2009; Poo, Kiong, & Ashok, 2007; Wegner, 1990). It plays a crucial role in the identification of the common features of objects and the categorization of similar objects into classes in object-oriented modeling. With the right abstraction method, objects can be grouped into classes, and programmers can form a hierarchical structure of superclasses and subclasses. Abstraction is not the only common ground between CT and OOP. According to Hadjerrouit (1999), it is

essential for novice programmers to have higher level of problem-solving skills in order to build object-oriented schemata rather than having technical coding skills. The researcher emphasized four main problem-solving skills for building well-structured object-oriented schemata; the analysis of the problem, the design of the solution, analogical thinking, and critical thinking skills.

Taking all of the aforementioned studies into consideration, it can be said that CT is a fundamental problem-solving skill approach in CS (Barr & Stephenson, 2011; Wing J. M., 2006; Wing J. M., 2008). Additionally, research in the current literature put emphasis on the importance of introducing CT as early as possible to novice programmers before letting them practice in actual programming languages. Similarly, OOP requires programmers to have high levels of problem-solving skills even before having met the technical competence requirements of programming with an actual programming language (Hadjerrouit, 1999). With such problem-solving skills, novice programmers would be able to build object oriented schemata and generic solutions to problems in the CS domain.

2.3 Digital game-based learning in introductory programming

The current literature in digital game-based learning for teaching programming to novice programmers focuses on two main approaches. These approaches are making use of visual programming environments to teach programming and learning programming through game-play experience.

2.3.1 Visual programming environments to teach programming

Kölling (1999b) claims that problems with the environment in which students learn object orientation are the most important problems, and therefore, researchers have

focused on developing different kinds of programming environments for teaching OOP (Carlisle, 2009; Cooper, Dann, & Pausch, 2003; Kölling, 2010; Kölling, Quig, Patterson, & Rosenberg, 2003).

BlueJ (Kölling, et al., 2003) is one of the first and the most popular educational programming environments. BlueJ has a window which presents a simple Unified Modelling Language (UML) class diagram to its users, and in this window, students can create objects from the classes of the UML diagram and interact with these objects via their methods. The BlueJ programming environment puts an emphasis on the visualization of OOP concepts, and it allows students to have dynamic interactions with classes to test those classes and methods (Kölling, et al., 2003). Van Haaster and Hagan (2004) conducted a survey to investigate the effects of the BlueJ programming environment on novice programmers. The respondents were 115 students who were taking the second compulsory programming course answered the survey. According to the responses, students made use of BlueJ features which helped them to facilitate higher order skill development in the cognitive, affective and psychomotor domains. Apart from that, participants had already been taught OOP with the Java programming language. Therefore, it is not clear whether or not students learnt OOP by using BlueJ, and the result of Van Haaster and Hagan's (2004) study depend on the perceptions of students, not on empirically collected achievement data.

Furthermore, Cooper et al. (2000) designed a 3-D interactive programming environment, Alice, which aims to help novice programmers to overcome their problems in learning OOP. In Alice, students can create their own 3D objects and manipulate these objects with drag-and-drop code blocks in the editor. Like the BlueJ programming environment, Alice enables programmers to see the immediate

results of their code blocks in their animation, which helps students easily understand the relationship between the code that they created and its results. Nowadays, the 3.2 version of Alice is available for Microsoft Windows, Mac and Linux. Wang et al. (2009) conducted a quasi-experimental study to investigate whether or not Alice could be used as a tool to teach fundamental programming concepts to high school students in Taiwan. Participants were 166 tenth-grade students from four different classes, 81 of whom were taught programming by using the Alice programming environment (experimental group), and 85 were taught programming with C++ (control group). The experiment lasted 8 weeks. Before the experiment, all participants took a motivation test which evaluated their motivation to learn programming, their self-efficacy of programming and their perception of computer programming. During the experiment, the same instructor taught both groups for about 50 minutes of lecture and 50 minutes of hands-on practice of programming each week. At the end of the eighth week, participants took the same motivation questionnaire, an experience questionnaire which assessed their experience of learning programming, and an achievement test on the programming concepts taught during the experiment. The results showed no statistically significant difference between the motivation and the learning experience of the experimental and control group students. The analysis of the achievement test showed that students who were taught programming with Alice performed significantly superior to the students in control group. Therefore, researchers claimed that visual programming tools can be a more effective way to teach basics of programming to novice learners. Florea et al. (2016) conducted another experiment with 60 undergraduate students. In this experiment, Alice was used as both a game and a game development tool for students while they are learning programming. Participants answered a survey which

evaluated their level of satisfaction with the proposed teaching method. The results showed that students had a positive perception of learning by playing and developing games. Therefore, researchers claimed that developing games with a visual programming environment, e.g. Alice, can be a good alternative to the current teaching methods for programming (Florea, Gellert, Florea, & Florea, 2016).

The Rapid Algorithmic Prototyping Tool for Ordered Reasoning (RAPTOR) (Carlisle, Wilson, Humphries, & Hadfield, 2005) is another visual programming environment that aims to help students improve their problem-solving skills and to reduce the problems of syntax of programming languages. In the RAPTOR programming environment, students create visual representations of algorithms to solve problems with the use of basic graphical symbols that the environment provides. RAPTOR provides an opportunity to run created algorithms either in a step-by-step mode or in a play mode to students. Researchers integrated RAPTOR programming tool into the spring 2003, fall 2003, and spring 2004 offerings of an introduction to computing course. In order to assess the effects of RAPTOR programming environment on students' problem-solving skills, three algorithmic design questions were asked to students at the final exam of the courses. The analysis of the results of the tests showed that there was a statistically significant increase in the performance of students on two of the questions. However, there was a statistically significant decrease in the performance of students on the array question. The results of this study indicated that students develop better problem-solving skills with the help of visual programming tools than with traditional non-visual methods. In 2009, RAPTOR was upgraded with the aim of teaching OOP to novice programmers (Carlisle M. C., 2009). The new RAPTOR opens with a UML diagram window. In this UML diagram, users can create classes just like the ones in

the BlueJ programming environment, but in RAPTOR there is another window, class editor window, for users to add methods and attributes to classes. One of the most important aspects of RAPTOR is that it supports OOP features such as polymorphism and inheritance, which were not supported in the previous programming environments. However, in the current literature, there is not enough substantial data on the effectiveness of the new version of the tool on students' OOP learning.

Another visual programming environment is Scratch, developed by the Lifelong Kindergarten Group at the MIT Media Laboratory. With Scratch, people are able to create different types of projects such as animated stories, mini games, music videos, simulations, etc. (Maloney, Resnick, Rusk, & Silverman, 2010). Scratch is designed to introduce the basics of programming to learners with little or no experience in programming. There were three main considerations in the development of Scratch: making it more tinkerable, more meaningful and able to create a bigger community of sharing than other programming environments (Resnick, et al., 2009). Developers of Scratch wanted users of Scratch to have the sense of playing and building as if they were playing with LEGO bricks. Therefore, Scratch has code blocks that are similar to LEGO bricks, and like LEGO bricks, these code blocks are designed in a way that they can be used only if they fit together. In Scratch, programming is done by snapping these code blocks together to control and manipulate 2D objects, sprites, in the stage. The developers of Scratch believed that these LEGO-like code blocks added more tinkerability, and students learn best when they work personally on meaningful projects (Resnick, et al., 2009). Maloney et al. (2008) introduced young learners aged 8 to 18 to a computer clubhouse with Scratch. The young learners worked with Scratch in extracurricular

activities without getting help from any instructors. The researchers collected students' Scratch projects to analyze how the young learners had worked with programming concepts such as interactions, loops, conditionals, booleans, variables, synchronization and random numbers. Their analysis showed that the majority of the students built Scratch projects by making use of these programming concepts without any formal help from instructors. Likewise, Meerbaum-Salant et al. (2013) conducted a mixed-method study to examine the effects of Scratch on the learning of CS concepts, namely, variables, loops, booleans, control structures, concurrency and message passing. Participants were 204 eighth and ninth grade students from a middle school, and Scratch was taught two hours per week for 20 weeks. Students took three different tests: a pretest, an interim test, and a posttest. The aim of the pretest was to measure students' abstract algorithmic skills. The interim and the posttest were aimed to measure the learning of these CS concepts. The results showed that students' learning of CS concepts was improved with Scratch. Yet the students had difficulty understanding concepts such as variables, concurrency and repeated executions. The researchers suggested that these difficulties could be overcome by explaining the relationship among between these concepts in detail. Even though Scratch visual programming environment has positive effects on the learning of CS concepts, this programming environment does not necessarily support object orientation.

An improved version of the BlueJ programming environment is Greenfoot (Kölling, 2010), and in this version, visualization of the current behavior and the state of the objects is instantaneous. In Greenfoot, users interact with classes in a specified world where students define behaviors of actors by calling methods in those actors through the editor and compiler windows. Begosso et al. (2012)

conducted a research to investigate the effects of the Greenfoot programming environment on students' conceptual knowledge of OOP. Participants in the study consisted of 30 first-year computer science undergraduate students. The study comprised four main phases. In the first phase, researchers conducted a survey to identify participants' level of OOP knowledge. The questionnaire consisted of 25 five-point Likert questions; 21 of the questions were about the knowledge of OOP, and the remaining 4 assessed knowledge of algorithms. According to the analysis of answers to the questionnaire, most of the participants did not have any knowledge of OOP and algorithms. After the first phase of the experiment, an instructor taught students the basics of OOP. After being taught about OOP, in the third phase of the experiment, students had hands-on practice with the Greenfoot environment and students had tasks about the process of developing of a game. While working with Greenfoot, students were first provided with examples to understand application of OOP concepts. Researchers asked students to solve problems about developing a game by using the concepts they had learnt after finishing each example. In the fourth and the last phase of the experiment, the researchers conducted an assessment test to find out how much students had learnt about OOP and algorithms. The results of the assessment test showed that students had an achievement rate of more than 60%. The researchers, based on their observations during the experiment, also claimed that students were motivated while they were learning OOP with Greenfoot.

In the current literature, visual programming environments used mostly as a platform for developing games. The researchers integrated visual programming environments into CS education in order to help novice programmers overcome their learning difficulties. Although the programming environments cited above have positive effects on teaching programming, research in the current literature reveals

that visual programming environments need to be used with a well-designed teaching methods, and learning materials should be provided to support their use (Meerbaum-Salant, Armoni, & Ben-Ari, 2013; Repenning, Webb, & Ioannidou, 2010). Otherwise, these programming environments will only bring a short burst of enthusiasm for novice programmers (Repenning, Webb, & Ioannidou, 2010). Furthermore, these programming environments lack the mechanism that provides feedback to students about their errors or the appropriate use of programming blocks (Meerbaum-Salant, Armoni, & Ben-Ari, 2011). Another concern about visual programming environments is that, even though these programming environments remove the extraneous cognitive load of syntax during programming process, there is still a need to write algorithms, which increases intrinsic cognitive load (Lister, 2011). Overall, developing programming environments is not the sole solution to the problems in teaching OOP. Teaching methods and the context such as games are also considered to be effective tools to teach programming to students. According to Jones (2000), games are wonderful examples of object-oriented environments if teaching object-oriented design and programming is the objective.

2.3.2 Serious games for learning programming

Serious games are defined as computer games which have educational goals and provide intriguing contexts with interactive, engaging and immersive activities (Gunter, Kenny, & Vick, 2008). Soflano (2011) argues that using games as learning environment is an efficient way to increase student engagement in courses because students of both genders and different ages can play games hour after hour, and they still can be in a teaching and learning environment while playing. Weintrop and Wilensky (2016) claim that the integration of coding into gameplay mechanics of the

game will enable students to become familiar with basic concepts of programming and allow them to develop programs in a more motivating and meaningful environment. Therefore, many researchers have showed interest in using games as learning environments for teaching programming to novice learners because of their engaging and motivational nature (Barnes, Chaffin, et al., 2007; Barnes, Richter, et al., 2007; Kazımoğlu, et al., 2012a; Mathrani, Christian, & Ponder-Sutton, 2016; Muratet, Torguet, Viallet, & Jessel, 2011; O'Kelly & Gibson, 2006). A review of the serious games and the corresponding CS concepts that games cover are presented in Table 2.

Phelps, Egert, and Bierre (2005) used a web-based 3D collaborative virtual environment, Multi User Programming Pedagogy for Enhancing Traditional Study (*MUPPETS)*, which aims to teach encapsulation, inheritance, and polymorphism concepts of OOP using the Java programming language. In this environment, students create their own robots that will fight with other players' robots in a virtual arena. In MUPPETS world, the difference between the concepts of class and object is visually represented so that students can define a class and create an instance of the class with specific attributes in the game world. Phelps et al. (2005) conducted a study in which MUPPETS is integrated as a tool for students to develop their final projects in one of the courses of first-year programming sequence at the Rochester Institute of Technology. In their study, students were asked to develop a final project, which was more complex than a weekly assignment and required a teamwork, in MUPPETS world. Before starting to develop their final projects, students developed chat applications, basic multiplayer games, voting machine simulations and so on via Robocode.

Students who developed final projects with MUPPETS world found the graphical system to be the key their enjoyment of programming. One of the students

Table 2. A Review of the Serious Games to Teach Programming

Serious Game	Computer science concepts	Programming language
MUPPETS (Phelps, Egert, & Bierre, 2005)	Fundamental OOP concepts	Java
RoboCode (O'Kelly & Gibson, 2006)	Fundamental OOP concepts	Java
Saving princess Sera – Catacombs (Barnes, Chaffin, et al., 2007)	Variables, conditionals, and loops	Micro-language
Second Life (Esteves, Fonseca, Morgado, & Martins, 2011)	Basics of computer programming	C-style micro-language
Prog&Play (Muratet, Torguet, Viallet, & Jessel, 2011)	Functions, recursion, data structures management	C, Java, Scratch, C++
TrainB&P (Liu, Cheng, & Huang, 2011)	Basic concepts of OOP, conditional logic, and loops	C like programming language
Program your robot (Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012a)	Computational thinking skills	Programming blocks
Alien Breed (Livovský & Porubän, 2014)	Fundamental OOP concepts	Java
ZTECH (Wong, Hayati, & Tan, 2016)	Fundamental OOP concepts	Micro-language
LightBot (Mathrani, Christian, & Ponder-Sutton, 2016)	Functions, conditional flows and recursion	Programming blocks
Software KIDS (Ramírez-Rosales, Vázquez-Reyes, Villa-Cisneros, & De León-Sigg, 2016)	Fundament OOP concepts and basics of software engineering	Micro-language
RoboBUG (Miljanovic & Bradbury, 2017)	Tracing codes, print functions, divide-and-conquer strategy, breakpoints	C++

that participated in the study felt uncomfortable about the sample materials used in the study by pointing out its being too male looking. Based on the anecdotal data, the researchers claimed that MUPPETS helped students to reach the cognitive learning objectives that had been specified in their course syllabi. Although the findings seem

promising, there is a need to provide empirical data on the issue by controlling all variables that influence the learning outcome.

RoboCode is another game and a game development environment developed by IBM that aims to teach students the basic concepts of structured programming and the fundamental concepts of OOP. In this game, students create robots by writing programs in the Java programming language, and players' robots fight each other in a small rectangular online environment. One of the most important features of this environment is that it enables a player to see instantly how the robots are affected by the player's codes. O'Kelly and Gibson (2006) held a competition using the RoboCode game development platform for students in their first year of programming. Students were introduced to the competition after the first half of the semester, which means that students were already familiar with basic programming concepts such as variables, iterations, control flows, and functions. The competition required students to work as a team. Additionally, since RoboCode was used in the competition, it offered students the opportunity to have a problem-based learning experience. Students submitted their robot-tanks, code and documentation of their robot-tanks to enter the competition. The competition consisted of a league in which the teams were divided into groups and matched with each other. Each team which became the leader in the groups proceeded to the next level in the competition and they were able to alter the code that defined their robot-tanks. Even though RoboCode gathers up different aspects such as fun, programming, games, artificial intelligence and competition, it does not free students from worrying about problems with syntax of a programming language because it uses a real OOP language, Java, not pseudo codes. The researchers also emphasized that in order to be successful in their game, students had to have prior experience in working as a team. Otherwise, if

students formed a team that was not balanced well, it was likely that there would be a decrease in their learning and success. Another important point that the researchers emphasized is that, with problem-based learning, the focus is shifted from teaching to learning, thus resulting in freedom for students to think for themselves, to use their existing knowledge on the topic and to gain new knowledge through explorations. However, the claims of the researchers were based mostly on their observations. Therefore, there is a need to do research on the effects of the problem-based learning of programming to provide empirical data on the issue.

Moreover, Saving Princess Sera (Barnes, Richter, et al., 2007) is a 2D role playing game (RPG) which aims to teach students variable declaration along with the simple usage of conditions, structures, and loops. In the game, a monster named Gargamel captures a princess named Sera, and the player's role is to help a man from the village, Arshes, to save the princess. Each of the tasks in the game involves programming concepts such as reordering while loop statements, correcting nested loops and solving picture puzzle of algorithms. The Catacombs is another game developed by Barnes, Richter, et al. (2007), but it is a 3D multiplayer game that shares the same objectives as Saving Princess Sera. In this game, users play the role of an apprentice wizard who is trying to help a mother to find her two children that are lost. The game has a linear structure in which students have only one option to perform in each task. The Catacombs game has two different versions. One of the versions has multiple-choice questions and dialogues with a spell book named Grimore. In the second version, Konijn, players are expected to select the correct scroll among the incorrect ones, and players receive game stones, which are code snippets that help players fill in the blanks in the codes. In order to examine the interface options of the games and overall feedback on the concept, Barnes, Richter,

et al. (2007) conducted a study with 13 students with prior knowledge in programming. The participants took a demographic questionnaire and a pre-test that assessed their existing knowledge in programming concepts. After completing the pre-test, the students were asked to play Saving Princess Sera for 20 minutes and The Catacombs for 20 minutes. When students finished playing the games, they took a post-test, after which they were interviewed for their opinions about the games. There was no meaningful difference between pre-test scores and post-test scores. Although there was no difference in achievement scores, the research showed that students liked the idea of games being used as a reinforcement tool for a programming class. In the light of the student comments, the researchers realized that these two games did not provide clear feedback to players, and therefore, they emphasized that such games should be providing clear and proper feedback (Barnes, Richter, et al., 2007).

According to the initial results, Barnes, Chaffin, et al. (2007) added a ranking system into Saving Princess Sera game. In this ranking system, players start as a rank 7 player, and by performing well on tasks, they ascend to rank 1. They also made modifications in Grimore version of The Catacombs game, but not in the Konijn version. A cut scene was added to the game that tells players they are a wizard who has a final task to complete in order to graduate from wizardry school. Players were also penalized for each of their mistakes in the game by a 20% decrease in their experience points. The final addition to the second game was another cut scene at the end of the game in which students were told how they performed in the game. With these modifications in their game prototypes, the researchers' had the goal was to investigate whether or not this explicit feedback would make a difference in game behavior and in the opinions of players. Barnes, Chaffin, et al., (2007)

conducted a second study with the same format and materials, except for the improvements in two games, with 8 participants. They compared the average time spent on each activity and the number of incorrect answers in both studies. Although there was no statistical difference between the average time spent on activities for the games, in the second study, the students completed the quests faster than the students in the first study. Five out of eight students in the second study reported that they were motivated by the feedback, the gold rewards and the ranking in the games. The researchers therefore claimed that the feedback in the two games may have affected the way students thought about using games to learn how to program. They suggested that the form of feedback in serious games should be examined in further studies to determine which one/s would be more effective with a broad range of students.

Esteves, Fonseca, Morgado and Martins (2011) analyzed how the teaching and learning of computer programming can be developed in the Second Life (SL) virtual world. SL is a 3D virtual world which enables students to program the behavior of objects by writing basic script codes. SL also provides immediate visualized feedback of written codes. Another important aspect of SL is the opportunity for collaboration in the 3D world. In the SL virtual world, more than one student can edit the same avatar by writing their own script and changing ideas via messaging through the system. The researchers conducted an action research which was a cyclical research that involved interventions or changes while it was being conducted. It consisted of four cycles, which started in March 2007 and ended in July 2008. Three different undergraduate student groups participated in the study: beginner students, students with a little previous knowledge of programming, and students with experience in semester-long programming projects. The participants

were asked to develop a semester-long project with their peers using a C-style scripting language called Linden Scripting Language (LSL). Once students were given an identical project description, they formed pair groups and started to develop their project in SL. The teachers met with students once a week in SL for 2 hours to monitor their progress and to help them with their problems in developing the project. Communication among students and between students and teachers was mostly text-based, although SL enables both voice-based and text-based communication. Data was collected through observations and a questionnaire that aimed to gather students' ideas about the difficulty level and the nature of the tasks in the project description. The findings of the study pointed to three important issues related to the subject: communication issues, students' process of learning, and the process of teaching programming. There were problems with communication among between students, and between students and teachers because text-based communication was preferred, and all of the messages appeared on the screen. Thus, students had difficulty following the conversations and instructions. Apart from that, students had the chance to talk with an instructor in a private conversation. Although students were grateful for the opportunity to talk with instructors, it was hard for the instructors to provide immediate feedback. Another problem that the researchers emphasized was the students' learning process. Student tasks were of two types: visual and nonvisual. In the visual tasks, students had immediate visual feedback of their program as the behaviors of the object that they defined in the project. On the other hand, in nonvisual tasks, students did not understand why they were doing the task; therefore, they had problems creating the right algorithm and finding the execution errors in their programs. The last issue that researchers emphasized is related to the process of teaching programming. The researchers claimed that the

teachers' physical presence at the beginning of the study was a crucial part of the learning process in case students have difficulty understanding the SL world. They also found that there was a feature that informed the teacher about the students' progress by email in the SL platform. With the use of such a feature, teachers could be guided through their teaching process by learning students' difficulties and attempts to solve these problems.

A multiplayer real time strategy game, Prog&Play, was developed by Muratet et al. (2011) to teach programming to novice programmers. The game motivates players by making them heroes in the story. Another important aspect of the game is that it gradually introduces programming concepts in its story, therefore enabling students to be masters at the end of the game. The game enables students to pause their program execution, to modify their codes and to execute the program again to see the effects of their changes in the codes. In order to study the possibility of a serious game being used to teach programming to draw computer science students' attention, Muratet et al. (2011) conducted three experiments. First, they selected 15 students, novice programmers, via a questionnaire that assessed motivation to playing video games and learning programming. Students who were interested in playing games but not interested in programming were selected for the first experiment, which consisted of two parts. In the first part, students played the game in a multiplayer session without focusing on programming tasks; they focused only on the game environment and the mechanics. In the second phase, Prop&Play was introduced and students were asked to play it. In the second experiment, the effects of the game on students' achievement scores and teachers' assessment were evaluated. The Prog&Play game was used in the first semester of the computer programming curriculum. There were 300 students taking the first-year programming

course, and half of these students continued to learn programming within traditional settings while the other half learned by playing the proposed game.

The third experiment tested the usability of the game by third parties, other instructors who wanted to adapt this serious game into their own pedagogy. Three teachers used the game to adapt their own pedagogy and conducted experiments with their students to analyze the effects of the game. The researchers defined four evaluation criteria to evaluate the results of the three experiments: (i) improvement in students' programming skills, (ii) usability of the game system, (iii) entertainment factor of the game and (iv) teachers' assessments. In order to evaluate the enhancement of programming skills, the researchers counted the number of missions completed by the students and the number of compiling instances was also recorded for each task in the game. With this data, the researchers defined the level of difficulty of the tasks in the game, and they saw that the fourth task was much harder than the first three. Although students had problems with the difficulty level of the tasks, the serious game reduced the failure rate of the students who were in the experimental group compared to students who were in the reference group. The usability and the entertainment aspect of the game was evaluated by a questionnaire administered in the first two experiments. The results of the questionnaire showed that students liked the way in which programming was taught with a game. Apart from that, students considered the game functional because there were no critical bugs that prevented them from playing the game. Overall, the results of the entertainment effect of the proposed game were also positive, which indicated that students found the game entertaining, and they appreciated being able to use their programming knowledge in such a context. The final criterion for the evaluation of the game was teacher assessment. Course sessions were filmed to observe teachers'

activities during the experiments. Apart from filming the lectures, a questionnaire was filled by the teachers about their perception of serious games before the experiments. After the experiments, the teachers were asked to submit a file to report their opinions about the experiments. Teachers' assessment of the game was positive, and they said the effect of the game on students' work was also positive.

Liu, Cheng, and Huang (2011) developed a 3D simulation game, Train: Build and Program It (TrainB&P*)*, and conducted an empirical study on the effects of simulation games on the computational problem-solving skills and learning experiences of novice programmers. The learning activities of the game were based on Papert's constructionism (1972). In the game, students were expected to design, develop and program the transportations in a railway network. One hundred seventeen freshman students participated in the study. Students had studied programming for one and a half months via traditional lessons before the experiment. The researchers proffered a learning experience survey to students in order to have a better understanding of the students' opinions of the learning experiences in the traditional lessons. After the traditional lectures, students worked with the game for two weeks. Students were asked to build and program a train that goes three rounds and stops at the starting point. The researchers stated that, in order for students to complete the mission, they needed to learn the basic concepts of OOP, conditional logic, and loops. Along with learning basic programming concepts, students also needed to consider physical laws because the game had a physics engine. Therefore, they needed to program their trains to complete the mission as quickly and as securely as possible. The researchers recorded students' railway network programs and the details of the design and development process of the railway systems. At the end of the game-based learning activity, they asked students to complete the same

learning experience survey to understand the students' perception of the learning experience in the game. The researchers also administered a survey to have a better understanding of students' motivation towards the game-based learning activity. Students responded to the motivation survey before and after the game activity phase of the study. The analysis of the learning experience survey suggested that students are more likely to be in a flow state when they practice computational problem-solving skills in a game rather than in traditional lectures. The same result was obtained from the analysis of the motivation survey. Thus, researchers suggested that integrating examples into games might help reduce student anxiety. Finally, a detailed analysis of the activity logs showed that a simulation game, which draws from constructionism, might enhance students' computational problem-solving skills. Additionally, the results suggested a help in the form of instructional support could help foster learning. Although the findings of the study are promising, it does not provide empirical data on the effects of the game on students' learning performance of OOP concepts. Furthermore, the researchers focused on the learning experiences, not the learning performance of novice programmers.

Kazımoğlu et al. (2012) developed a prototype game called Program Your Robot to study the effects of serious games on students' CT skills. They claim that developed serious game focus on abstract and conceptual knowledge of programming rather than on the functions of developing CT skills, which underlie the basics of programming. In order to have students acquire basic CT skills, researchers proposed a game which enables students to create algorithms of solutions of problems, to apply computational thinking methods to the given problems, to debug errors in their algorithms, and to observe visual representation of their algorithmic solution. They conducted an experiment with 25 undergraduate students

at different levels of classes within computer science discipline. In the experiment, students were asked to write feedback about the game they had played. The results revealed that the majority of students thought game was helpful in improving their problem-solving skills and understanding basic programming constructs.

Livovský and Porubän (2014) developed the Alien Breed game, a remake of a 1991 action-adventure game called Amiga, to teach the basic concepts of OOP to novice programmers without making students write programs in a real OOP language. The game is a top-view two-dimensional game in which players are expected to shoot their way through enemies and to find the exit in order to proceed to the next level. In the proposed game, researchers made use of an object-first approach, which aims to introduce fundamental concepts and principles of OOP to students before writing programs in OOP. The game focused on several concepts of OOP, namely, object, class, attribute, operation, encapsulation, and inheritance. The proposed game has three different levels which aim several learning objectives. These objectives are:

- Explaining the concept of object
- Understanding the relationship between objects and classes
- Knowing the roles of attributes and operation in terms of OOP
- Explaining why encapsulation is needed
- Explaining the inheritance concept by identifying relationships between subclasses and superclasses
- Understanding the inheritance concept's importance in OOP

Livovský and Porubän (2014) developed an educational tool called Object Access Tool (OAT) to support the interaction with concepts of OOP while students are playing the game, Alien Breed. The OAT has three components for

representations of object models in the UML class diagram: displaying instances of selected class, and displaying members of an object, or a static class. In order to measure the effects of Alien Breed on students' conceptual knowledge and understanding of OOP, a questionnaire was administered to students after they played the game for about an hour. Fourteen undergraduate students studying in the informatics program answered the questionnaire, which consisted of two types of questions: multiple-choice questions, which measured the basic conceptual knowledge of students; and open-ended questions, which asked students to write concepts of OOP in their own words, measured deeper information about students' conceptual knowledge of OOP. All of the respondents except two had no prior experience with OOP, but they did have experience with procedural programming. The results of the study showed that 5 students had scored 90% or higher on the multiple-choice questions and also had partially correct answers in the open-ended questions. Apart from the questionnaire, researchers also collected data by observing students. According to the observations, there were some problems with the presentation of tutorials, task instructions and conceptual knowledge of OOP in text format. The researchers stated that students were reluctant to read help texts, instructions for tasks, and to learn content. They believe that the length of those texts was the reason, and they recommend replacing these text tutorials with animations or video tutorials.

ZTECH (Wong, Hayati, & Tan, 2016) is a role-playing game (RPG) which aims to teach basic OOP concepts such as encapsulation, polymorphism, and inheritance in a fun, easy, and interactive environment. The game consists of ten mini-puzzle games with 8 main quests that cover some basic programming concepts. Sixty first-year students who study game development were asked to play the game

to evaluate the effects of the game. After completing game play, students were required to complete a 15-item questionnaire. The first three asked for personal details and existing programming knowledge; the other five questions were about the usability of the game. The remaining seven asked students' perceptions about the game-based approach used in learning object-oriented programming. The results showed that 65% of the students found ZTECH an effective tool to teach object-oriented paradigms. The researchers also stated that 15 of the participants had passed an OOP course at the university with grade of A. They claimed that pseudo codes should be used in introductory programming courses in order to overcome difficulties that are caused by the syntax of a real programming language.

Mathrani et al. (2016) used the LightBot game in their research on game-based learning issue. The LightBot game has a fictional story in which players program a robot to light all blue tiles in a specific path. Players accomplish tasks by using prefabricated commands that represent fundamental programming concepts such as functions, conditional flows, recursion. The researchers conducted a study with two different student groups. The first group consisted of 20 students with no prior programming experience but with a little computing experience. The second group had 24 students with basic programming knowledge. Students in both groups were asked to play the game, but the two groups had different orders in the game and they also had different time sets. The task and the time span were different because the groups differed in prior knowledge of programming. The researchers made the first group play the basic level first to learn the basic mechanics of the game and to learn functions, procedures and the sequential flow of execution of the program. After completing the basic level, students in the first group played the first level that covers recursions and conditional. On the other hand, students in group two played

the game as it was designed by the developers. Immediately after playing the game, students in both groups were asked to complete an online survey to collect qualitative and quantitative data. The qualitative data used open-ended questions about the students' opinion about the game. Qualitative data was collected with a 5-point Likert scale questionnaire. This questionnaire aimed to gather the understanding of students' perceptions of the game, conceptual knowledge on loops, conditionals and recursion, and also about the difficulty of the game. Students in the first group rated the basic level of the game as easy, recursions as difficult, and conditionals as very difficult. Thirteen of 20 students in the first group correctly answered the questions about their learning through game. Students in the second group thought that the game was an effective tool to learn programming concepts, and the game also clarified earlier conceptual difficulties in programming. Overall, the findings of the study indicated that students loved the game, and the researchers stated that games can be useful in learning basic programming concepts such as functions, procedures, conditionals, loops, and recursions. They concluded that LightBot was designed for basic-level programming and they suggested that another more complicated and intensive game should be used to study advanced programming.

Ramírez-Rosales, Vázquez-Reyes, Villa-Cisneros, and De León-Sigg (2016) developed a serious game, Software KIDS, to teach basic conceptual knowledge of OOP and the basics of software engineering (SE) to children older than eight years old. The game has ten levels that focus on fundamentals of OOP such as class, object, attributes, methods, inheritance, polymorphism, abstraction, encapsulation etc., and the basic concepts of SE, such as algorithms, conditional and iterative structures, and arrangements. Although the proposed game is aimed at children older than 8 years old, researchers conducted an experiment with 12 children aged between

6 and 12. The experiment consisted of two sessions: In the first session, the children were given a short lesson about technology creation for an hour. In the second part of the experiment, students played the Software KIDS game for about 50 minutes and then had 10 minutes to answer a questionnaire. Both sessions were held under the supervision of mentors, who supported the children while they were playing the game. Data collected through mentors' observations and a questionnaire that asked students' opinions about levels of difficulty, easiness, fun, and satisfaction with the game. The average score for the Software KIDS game was 9 out of 10. The researchers found that the Software KIDS game motivated children to play and learn programming even if they had difficulty in playing the game. The findings of the study showed that after playing the game, students' perceptions about software development had changed. Although the children enjoyed the game, they needed the help of a mentor to solve problems they encountered in the gameplay. Therefore, the researchers emphasised that the difficulty levels of the game needed to be suitable for age group. The researchers also stated that it was hard for novice programmers to understand, because the proposed game was highly based on OOP and specialized in the software engineering area.

RoboBUG (Miljanovic & Bradbury, 2017) is a puzzle-type serious game designed for first-year computer science students who are learning C++. The standard version of the game implements debugging in C++, but it also allows instructors to create new levels with different programming languages. In the proposed game, players have a role of a scientist who tries to save the world from invader alien bugs by purging bugs from the scientist's robotic suit of armor, Mech Suit. The player as a scientist purges the alien bugs from the armor by correcting errors in the infected source code of the different parts of the Mech Suit. The

RoboBUG game has four different levels that focus on code tracing, print statements, divide-and-conquer methods, and breakpoints in the C++ programming language. Each level in the game starts with a tutorial in which new debugging tools are introduced to the players, 2 or 3 sub problems involving small debugging tasks, and a final challenge in which students are expected to use newly introduced debugging tools along with the knowledge they gained while doing the small tasks. The researchers conducted a study to examine the effects of the game on students' understanding of debugging and the players' experience. The participants were 14 first-year computer science students at the University of Ontario Institute of Technology between 18 and 25 years old. They first completed a test, the Positive and Negative Affect Scale (PANAS), that assessed their feelings. After the completion of the PANAS test, the students took a pre-test of 10 multiple-choice questions to measure their existing knowledge on debugging techniques. After completing the pre-test, they had 30 minutes to play the game. At the end of the experiment, they completed a post-test on debugging and they completed the PANAS questionnaire one more time. The researchers conducted a paired t-test to analyze the scores of pre-test, post-test and PANAS tests. The results of the analysis showed that the proposed game helped students to acquire debugging skills. In addition to this, the RoboBUG game yielded more improvement in test scores of students with low prior knowledge on the topic than of students with a higher level of prior knowledge on debugging. An analysis of the PANAS test showed no significant difference in the positive and negative effects. The researchers claim that the game should include a hint system to relieve the frustration of players.

2.4 Summary of the literature

In order to overcome the problems of novice programmers in learning OOP and CT skills and to motivate them, researchers have adopted digital game-based learning approaches. There are two main approaches in the literature on the digital game-based learning of programming: learning by developing games and learning through gameplay experience.

In the learning-programming-by-developing game approach, researchers have made use of visual programming environments to teach programming to novice programmers. Although there is much supporting empirical data for the use of programming environments, there is still significant criticism of them. First, the research in the current literature points out that these visual programming environments will only bring a short burst of enthusiasm unless they are used with well-designed teaching methods and learning materials (Repenning, Webb, & Ioannidou, 2010). These programming environments lack the mechanism to provide feedback to students about their error or about the appropriate use of programming blocks (Meerbaum-Salant, Armoni, & Ben-Ari, 2011). Another important concern about these environments is that there is still a need to write algorithms for solutions to problems, which increases the intrinsic cognitive load (Lister, 2011). In other words, the visual programming environments do not provide guidance or assistance for developing algorithms even though they enable novice programmers to implement their algorithms in a more user-friendly way.

The other main approach in the digital game-based learning of programming uses serious games to teach programming to novice programmers by providing a gameplay experience for students. In recent years, many researchers investigated the effects of serious games on learning computer science skills and programming.

Although a considerable amount of experimental research has been conducted on this topic, few of the studies provided a well-prepared experimental design and demonstrated inferential statistical analysis (Livovský & Porubän, 2014; Mathrani, Christian, & Ponder-Sutton, 2016; Miljanovic & Bradbury, 2017). The findings of the majority of the studies on the current problem are based either on anecdotal evidence or on initial evaluation results that do not provide empirical data about what students learn from these proposed games. Additionally, there is not enough substantial data about the effects of serious games on Turkish students' programming skills. Finally, the majority of the studies in the current literature focus on what is being taught rather than how the developed games support novice programmers' learning by providing details about the instructional design of the game (Laporte & Zaman, 2018).

Nonetheless, there are some common findings in those studies. The most prominent one of those findings is that, thanks to the games developed, abstract programming concepts can be concretized and students are provided with chances to see the instant results of the programming activities in the games; both help students during the debugging phase. Additionally, as revealed by another finding in a number of studies, games particularly enhance motivation to learn programming, and some studies found out that games might even encourage students to study in the field of programming.

This thesis, by drawing from the findings of current literature on the problem, aims to examine the effects of a serious game, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard), on students' conceptual knowledge of OOP and CT skills. By considering all aforementioned issues, this study was designed to answer the following research questions:

1. Is there any significant difference between the post-test and pre-test scores on the conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience?
    a. Is there any significant difference between the post-test and pre-test scores on the conceptual knowledge of OOP of undergraduate students with no programming experience?
    b. Is there any significant difference between the post-test and pre-test scores on CT skills of undergraduate students with no programming experience?
2. Is there any significant difference between the post-test and pre-test scores on the conceptual knowledge of OOP and CT skills of undergraduate students with procedural programming experience?
    a. Is there any significant difference between the post-test and pre-test scores on the conceptual knowledge of OOP of undergraduate students with procedural programming experience?
    b. Is there any significant difference between the post-test and pre-test scores on CT skills of undergraduate students with procedural programming experience?
3. Is there any significant difference between the achievement scores on the conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience and of undergraduate students with procedural programming experience?
4. To what extent do students' creative problem-solving skills and attitudes towards digital game-based learning of programming influence their achievement score on the conceptual knowledge of OOP and CT skills?

CHAPTER 3

METHODOLOGY

This chapter presents the details of the methods and procedures followed in the current study. The chapter consists of the following sections: (1) research design, (2) participants and sampling procedures, (3) treatments, (4) instruments, (5) data collection procedures.

3.1 Research design

This study employs a quasi-experimental design approach in order to minimize the effects of extraneous variables on the outcome. A pre–test and post-test quasi-experimental design (Creswell, 2011) is used to study the effects of serious games on undergraduate students' conceptual knowledge of OOP and CT skills. Another reason for using a quasi-experimental design is that the researcher was unable to create groups with a random assignment method.

The independent variables of the study were the level of the students' creative problem-solving skill, students' attitude towards digital game based learning of programming and OOP and CT pre-test scores of the students. The dependent variable of the study is students' achievement scores in conceptual knowledge of OOP and CT skills test. The independent and dependent variables of the study are displayed in Table 3.

3.2 Participants and sampling procedure

The target population of the study was undergraduate students studying computer programming in non-engineering disciplines in Turkey. Identifying all the

individuals in the population would not be possible because the population of the study is extremely large. Sample undergraduate students who were accessible to the researcher were selected from the Computer Education and Educational Technology Department of Bogazici University. Convenience sampling (Creswell, 2011) was followed as the sampling method because there was no chance to access participants randomly. For selection of the participants, the main criterion was that students should not have had experience in OOP before the experiment.

Table 3. Variables of the Study

Independent variables	Dependent variables
The level of creative problem-solving skill Attitude towards digital game-based learning of programming Prior knowledge of OOP and CT skills	Achievement in conceptual knowledge of OOP Achievement in CT skills

The researcher selected two different student groups in the 2017-2018 academic year. The first group of the sample consisted of freshman students without prior experience in programming, while the students in the second group were sophomores with experience in procedural programming but not in object-oriented programming. Data were collected from the all 61 students in these two groups (see Table 4). A pre-test on the basic conceptual knowledge of OOP and CT skills is given to the students, and according to the results of this test, a student with higher level of conceptual knowledge on OOP and CT skills is excluded from the study.

Table 4. Participants of the Study

Freshman students		Sophomore students	
Female	Male	Female	Male
20	10	12	19

3.3 Treatments

In this study, a 2D science-fiction themed hybrid (puzzle-solving and simulation) serious game, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard), was developed by the researcher with Unity 3D game engine using C# programming language. The game was specifically designed to be a simulation game because simulation games allow players to explore virtual game world and interact with the other game objects to test their hypotheses (Kiili, 2005). The main objective of the developed game is to introduce fundamental concepts of OOP namely class, object, attribute, data, method, inheritance, polymorphism and encapsulation to students in a meaningful and fun environment. Apart from introducing OOP concepts, it also aims to enable students to practice CT skills: conditional logic, algorithm building, simulation and debugging, even if students have no programming knowledge.

After the development of the first version of the game, the researcher asked the opinion of three educational technology specialists and four software engineers in terms of usability, instructional design and the integration of programming concepts and procedures into gameplay. Additionally, a pilot study with 5 students with experience in OOP was also conducted to assess the usability of the game. In the light of the feedback from these initial evaluations, the gameplay, the screen and the message design of the game were revised.

3.3.1 Serious game design model

In the design and the development of a serious game using a game design model that successfully integrates game characteristics, educational theory is important. In order to ensure that students would accomplish the objectives of the learning unit, it was

necessary to choose a suitable serious game model. In choosing the conceptual design framework of the game developed, I took into consideration certain criteria that the framework should bear:

- a learning environment which would provide a high level of interactivity to motivate learners,
- a learning environment that would provide problems in an authentic context,
- learners would be able to analyze a problem situation and generate their solutions,
- learners would be able to actively test their solutions and discover,
- learners would be able to observe the outcomes of their solutions and alter them to get better solutions.

Even though there are many serious game development frameworks in the current literature, Kiili (2005)'s Experiential Gaming Model was selected as the conceptual design framework of the developed game because it was the one that most closely met the researcher's criteria. Another reason this framework was selected was that the experiential gaming model is one of the most widely accepted and referenced frameworks in the literature, even though there are several other serious game models.

This model aims to create a link between gameplay mechanics and experiential learning theory to enhance players' flow experience. Experiential learning theory stresses the importance of direct experience and reflective thinking in learning (Kolb, 1984). Flow, on the other hand, is the state of having optimal experience from an activity by being completely engaged (Csikszentmihalyi, 2014). Players are commonly in a state of flow when they play a game. By grounding the design of the game in this study according to an experiential gaming model, I aimed

to increase in the motivation of novice programmers because novice programmers often have low motivation to learn OOP (Prensky, 2003; Sarkar, 2006).

The experiential gaming model consists of three main cycles, namely preinvative idea generation, idea generation and the active experimentation cycle consisting of reflective observation and schemata construction (see Figure 2). Challenges or problems, which according to the researcher is the heart of the model in the games, are at the center of these three cycles. Challenges in a serious game play a crucial role in keeping players in a state of flow. The level of the challenges in game activities is important in the design of instructional games because easy challenges may bore players, while hard one may cause players to be anxious (see Figure 3). Therefore, it is important for a serious game to provide learners with challenges that will match the students' skill and knowledge level. Furthermore, challenges in a game should be designed in a way that the difficulty of the tasks will increase when players make progress in the game.

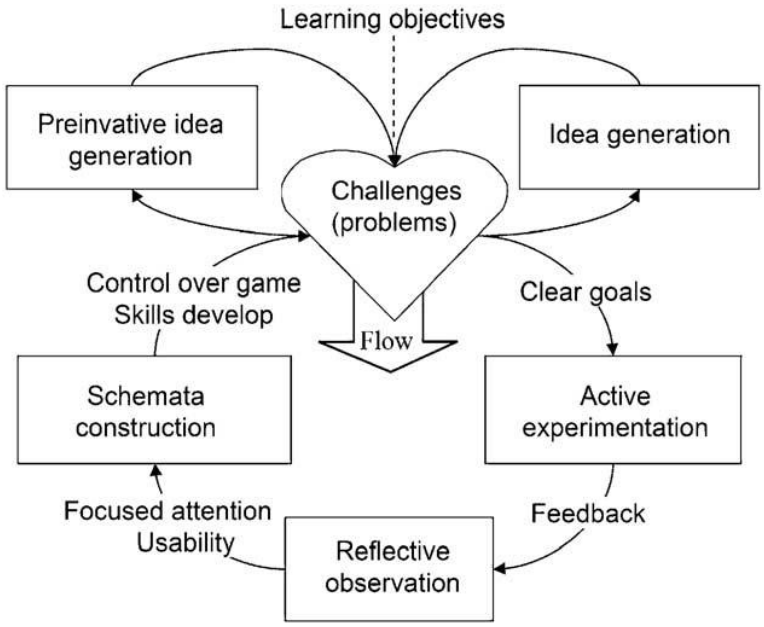


Figure 2. Experiential gaming model
Source: Kiili, 2005

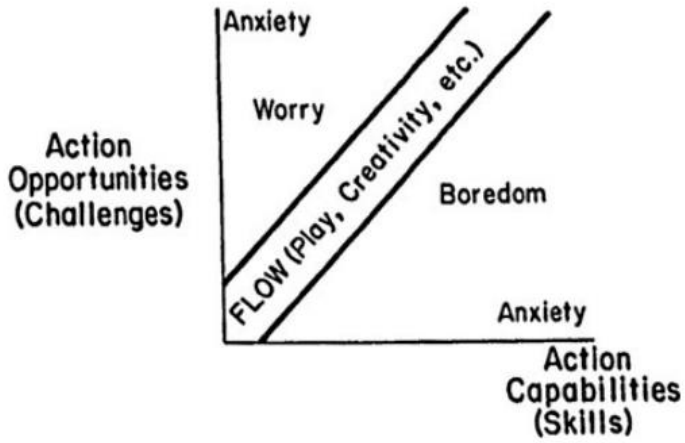


Figure 3. Model of the flow state

Source: Adapted from Csikszentmihalyi, 1975

There are two idea generation loops, preinvative idea generation and the idea generation loops, in the experiential gaming model in which players develop their solutions to the problems. The difference between the preinvative idea generation and the idea generation cycle is that the preinvative idea generation cycle has a disorganized structure which can usually be seen in the way children play. In the idea generation loop of the model, players analyze the problems and generate their solutions according to the rules and constraints of the game world.

After completing their solutions in the idea generation stage of the model, the players move on to the experimentation stage. They implement their solutions to the problems and observe their effects on the problem situation. In the reflective observation phase of the experimentation cycle, clear feedback plays a crucial role. With the help of feedback from the game world, learners may understand the deficiencies in their solutions and thereby improve their solutions to create more effective ones. This experimentation and observation process of solutions would help students to construct new knowledge schemata, consequently resulting in learning.

The researcher emphasized that it is important for learners to test different solutions to a problem to improve their creative problem-solving skills and current knowledge on the topic (Kiili, 2005). Although this model provides guidance and information about the fundamentals of designing serious games, it does not necessarily refer to the instructional design of the activities in a game. Therefore, along with a conceptual design framework of serious games, an instructional design model was used in the design and the development of the activities of the game.

3.3.2 Instructional design model

The instructional design of the activities of the game in this study is based on the four-component instructional design model (4C/ID model) (van Merriënboer, Clark, & de Croock, 2002). The 4C/ID model regards authentic learning tasks as the core of teaching and complex learning because with such tasks, learners are able to integrate their knowledge, skills and attitudes. The experiential gaming model and the 4C/ID model are similar in terms of the design of their learning activities. Both models encourage the use of ill-structured problems in a learning environment to support discovery learning. Furthermore, gradual increase in the level of difficulty of learning tasks are emphasized in both models. The experiential gaming model lays emphasis on clear feedback in supporting students. Similarly, the 4C/ID model also points out the role of providing support to learners in the form of procedural and supportive information.

The 4C/ID model consists of four major components and these are (1) learning tasks, (2) supportive information, (3) procedural information and (4) part-task practice. Learning tasks are authentic whole-task problems which are based on real-life situations. By working on learning tasks, learners build knowledge schemata

and integrate their current knowledge, skills and attitudes. Learning tasks are divided into task classes according to their level of difficulty. In other words, learning tasks should be organized in a way that students will start working on relatively easy tasks and finish with the difficult ones. According to this model, supportive and procedural information should be presented to students over the course of their learning experience. Supportive information is provided to help learners to perform nonroutine, complex and problem-solving parts of the learning tasks. Procedural information, on the other hand, provides help to students in routine aspects of learning tasks. In other words, procedural information indicates a step-by-step instruction about a routine task in learning process. While learning tasks in this model refers to whole-task activities, the last component of the 4C/ID, the part-task, refers to the practice of automated constituent skills. When a high level of automaticity is required to perform a task, the learning tasks may not be sufficient. In such circumstances, additional part-task practice should be provided for learners.

3.3.3 Conceptual design of the game

Based on the guidelines and information from the game model and the instructional design model, the Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard) game was developed. The game developed is similar to LightBot (Mathrani, et al., 2016) and Program Your Robot (Kazimoglu, et al., 2012) in terms of game play. Even though these two games are designed to teach the basics of procedural programming and computational thinking, neither aims to teach OOP concepts to novice programmers. Unlike the other two games, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard) includes a component that enables students to

build their own code blocks according to the needs of their missions as a game play experience.

One of the most important aspects of the game is that the fundamental concepts of OOP are integrated into the story of the game. Fantasy elements such as imaginary machines and planets were used to integrate OOP concepts into the story of the game because the use of fantasy elements may enhance students' learning (Garris, Ahlers, & Driskell, 2002). Using stories is one of the core components of the game design process (Rollings & Adams, 2003). Stories set the background for games, and using stories in games enables the integration of small tasks into a main goal. Therefore, I aimed to help students understand the need of such concepts and where to use them in authentic problem settings. An Animated Pedagogical Agent (APA), Professor Ekrem, plays a crucial role in the integration of OOP concepts into the story of the game by telling the story and providing information about the learning activities. APAs are on-screen characters that act as personal tutors in computer-based learning environments, and they provide feedback and contextualized information about the learning unit in an activity (Bates, 1994; Lester, et al., 1997). Using APAs in computer-based discovery-learning environments increases learners' motivation and leads to deep learning (Moreno, Mayer, Spires, & Lester, 2001). Furthermore, in the current literature, the chosen serious game and instructional design models emphasize the importance of feedback for novice programmers (Barnes, Chaffin, et al., 2007; Kiili, 2005; van Merriënboer, et al., 2002). Therefore, an immediate feedback mechanism in which students are informed about their mission is provided with hints about their mistakes, both visually and verbally.

3.3.3.1 Supportive information

In the developed game, supportive information is given via a panel called mission information. With such information, students were expected to construct a knowledge schemata by building a bridge between their current knowledge and the new ones. In the information panel, the English word of each OOP concept was provided — even though the language of the game is Turkish — along with the corresponding Turkish word and the definition of the concept (see Figure 4). The English word for the concepts was provided to minimize the problems that students might encounter in the future because they will be using these concepts in English when they start to work with real programming languages.

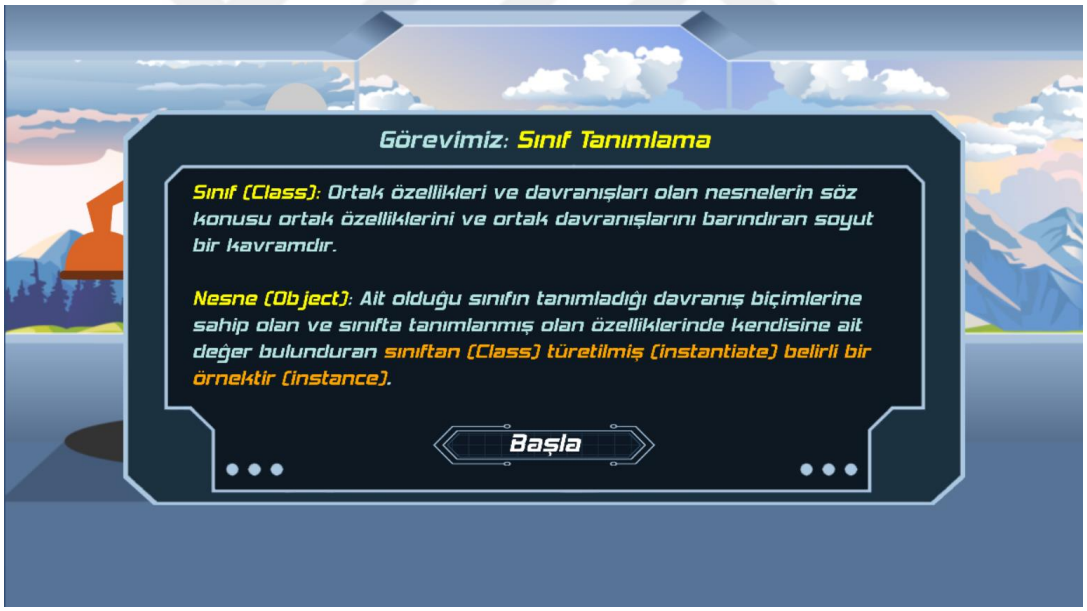


Figure 4. Mission information panel

A help menu that provides hints and examples for each mission in the game is another important feature of the game developed (see Figure 5). Thanks to the help menu of the game, students will be able to get help whenever they need. Finally, the researcher paid special attention to not to overwhelm learners' the cognitive load in

the course of designing the learning activities. According to Sweller, van Merriënboer and Paas (1998), human beings have a limited capacity for working memory, for this reason all instructional materials should be developed by considering the cognitive load of learners. Thus, some key points and concepts of OOP were highlighted in the instructions in order to lower the cognitive load of students.

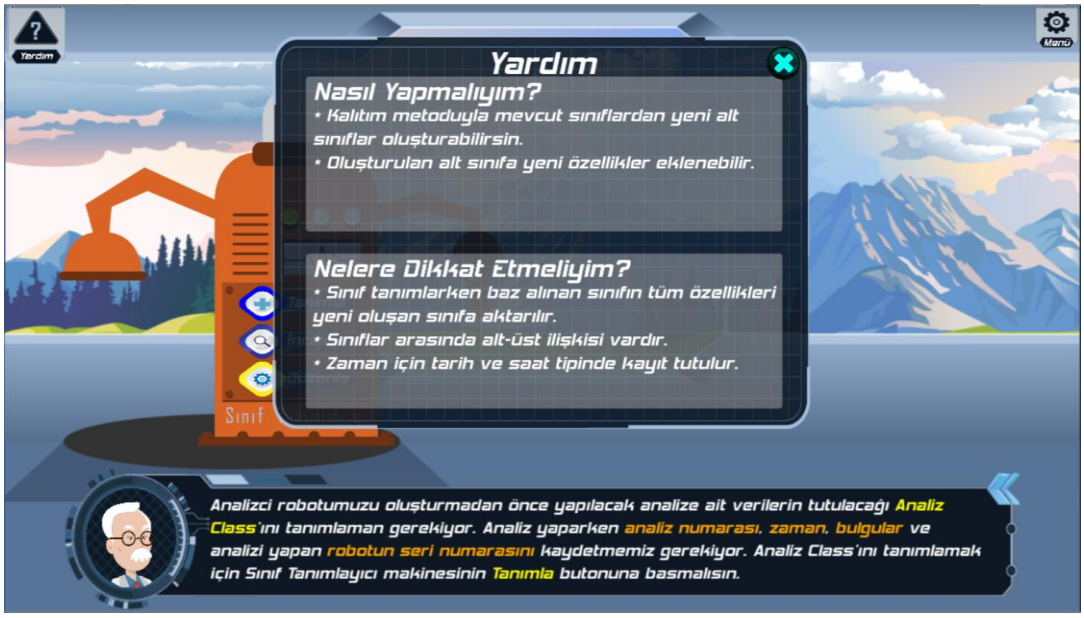


Figure 5. The help menu

3.3.4 Associating game-play with object-oriented concepts and computational thinking

The player's role in the game is to work in the Turkish Space Agency (TSA) as a programmer who is specialized in programming robots. According to the story of the game, in 2048 the world is on the verge of a crisis because of global warming, and scientists in the TSA are looking for a new planet for humans to live. The game is a puzzle-solving simulation game in which students first build their own robots and program them to explore the planet Asgard so as to decide determine whether

humans can live on that planet or not. There are eleven different activities with a gradual increase in the level of difficulty. Throughout the game, students are given instructions about their current mission through an APA, Professor Ekrem, in an instruction area at the bottom of the screen.

In the first mission of the game, students are expected to build a chip which will contain the specifications of a robot to be used in the exploration mission. The chip in this activity is an analogy that represents the class concept. One of the problems of teaching and learning of OOP stems from using a real programming language to learn because of the complex syntax of the programming language (Guzdial, 2008; Xinogalos, 2016). Therefore, in order to get students to be more comfortable and not to worry about the syntax of a real programming language, defining a class called robot is done through an imaginary machine called Class Definer Machine (see Figure 6).

Figure 6. Class definition activity

After defining the robot class, students will be directed to create an instance of the class by inserting the created chip into one of the robots. Three different robot options are presented, and one of them may be chosen. When students choose a robot to put their chip in, a pop-up window will appear. This pop-up window is used as an analogy for constructors in OOP. The students create their own robot by indicating the name, speed, and the carrying capacity of their robot (see Figure 7). The visual representations of abstract concepts of OOP play an important role in novice programmers' learning of OOP (Kölling, 1999b). Therefore, the visualization of abstract concepts of OOP are provided by an animation after students indicate the specifications of their robots. In the animation, the chip enters the robot and brings it to life. Thus, while the chip was representing an abstract concept, class, the live robot refers to a specific instance of the class, an object.

Figure 7. Creating a robot instance activity

The next mission is about defining methods of the created robot by using code blocks in the game. The APA in the instruction panel of the game presents the details

of methods to the students. The method creation window allows students to try their methods and see the immediate results of their coding in a simulation screen on the right side of the Method Definer Machine (see Figure 8). As stated earlier, novice programmers have difficulties in developing algorithms to solve a problem (Xinogalos, 2016). Therefore, this activity is specifically designed in a way that it will help novice programmers to think their daily motions critically and divide basic motions such as walking into small steps. The main purpose of the activity is to give the novice programmers a smooth introduction to algorithmic thinking by making them analyze their daily movements step-by-step. Another reason of letting students to develop methods of the robot is to design a solution to a problem which is independent of a particular situation. According to the literature, students have problems changing their mindsets from procedural programming to OOP (Hadjerrouit, 1999; Xinogalos, 2016). In other words, novice programmers develop the habit of generating problem specific solutions in procedural programming, but the nature of OOP requires programmers to develop generic solutions that could be applicable in different problem situations.

After defining the basic movement methods of their robots, students are asked to program their robot to walk towards the spaceship to start their journey in space. In this activity, the basic layout of their programming environment will be introduced in a concealable coding panel on the right side of the screen (see Figure 9). Methods that are created by students are shown on the upper side of the coding panel. By simply dragging and dropping these code blocks into the free space below the coding panel, students will be programming their robots. There are two buttons at the bottom of the coding panel: Run (Çalıştır) and Clear (Temizle). When the Run button is clicked, the code blocks in the coding area are executed, and the coding panel

becomes inactive. The aim of disabling the coding panel is to get students to pay close attention to the execution of their codes and find mistakes, if there are any.

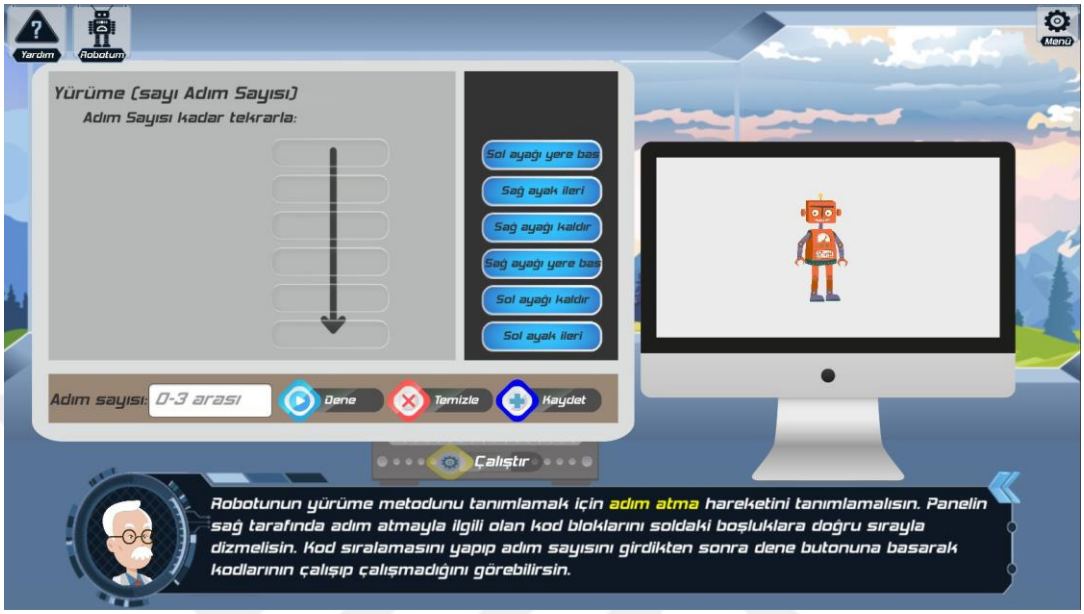


Figure 8. Defining method activity

Figure 9. Programming the robot to go to the spaceship

A visual and textual feedback mechanism, which is one of the most essential characteristics of the game, is specifically designed to help novice programmers to

understand the execution of their program and debug their codes, because the lack of such a mechanism in visual programming environments causes problems for novice programmers (Meerbaum-Salant, Armoni, & Ben-Ari, 2011). For example, if a code block runs properly, the block will turn green, or if there is an error with the code block, it will be red, and if an input is missing in any of the code blocks, the code blocks will be yellow. Along with the visual feedback for code blocks, a pop-up feedback message providing information about the error will also appear if there are any bugs in the code (see Figure 10).

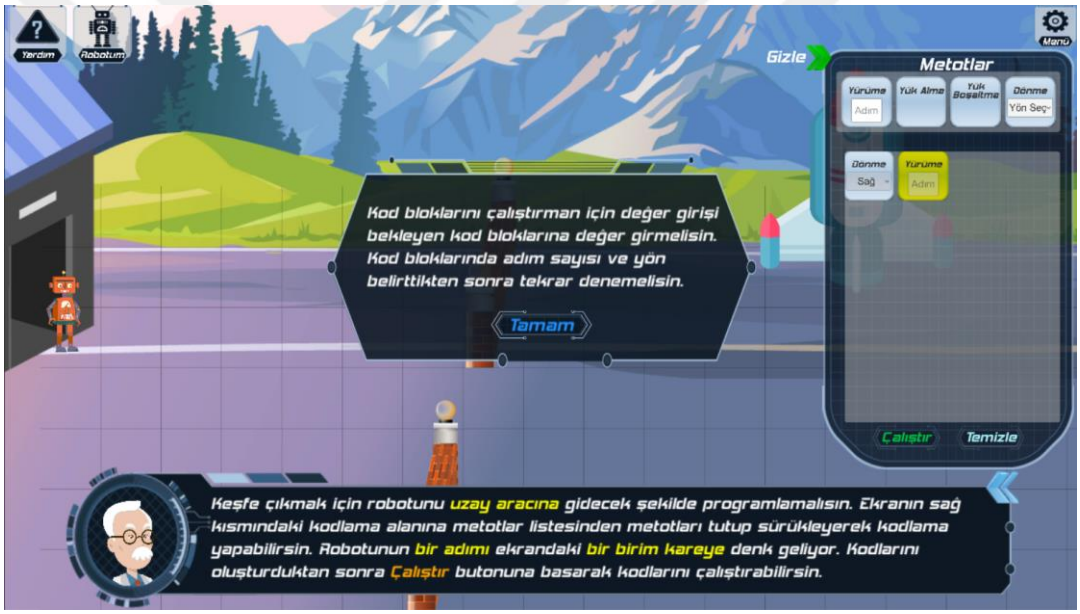


Figure 10. The feedback message

The next mission introduces the encapsulation concept of OOP. Students are asked to establish a connection between the created class, robot, and spaceship class so that their robot will be able to use methods of the spaceship to go to the planet Asgard. This activity introduces one of the main reasons for using the encapsulation process in programming, which is ensuring the security of a class. Students are able

to change access conditions of the methods of spaceship class by simply clicking on the radio buttons namely, as *public* and *private* (see Figure 11).

Figure 11. Encapsulation activity panel

After properly setting the access conditions of the methods of the spaceship, a mini space map will be displayed to students (see Figure 12). According to the story of the game, this mini map is shown in a simulation screen because the spaceship travels with the speed of light and it is not possible to drive the spaceship manually at such speed. Therefore, they need to program their journey from planet Earth to Asgard in advance. The same coding panel appears on the right side of the screen, but this time in the methods part of the coding panel students will see the methods of the spaceship. If students arrive at one of the other planets or leave behind the mini map, they will get a warning message reminding them of their mission objective.

When the spaceship arrives on Asgard, students start to program their robot to collect sample objects, small stones, from the planet's surface (see Figure 13). There are two different small stones to collect and huge rocks as obstacles on the surface of

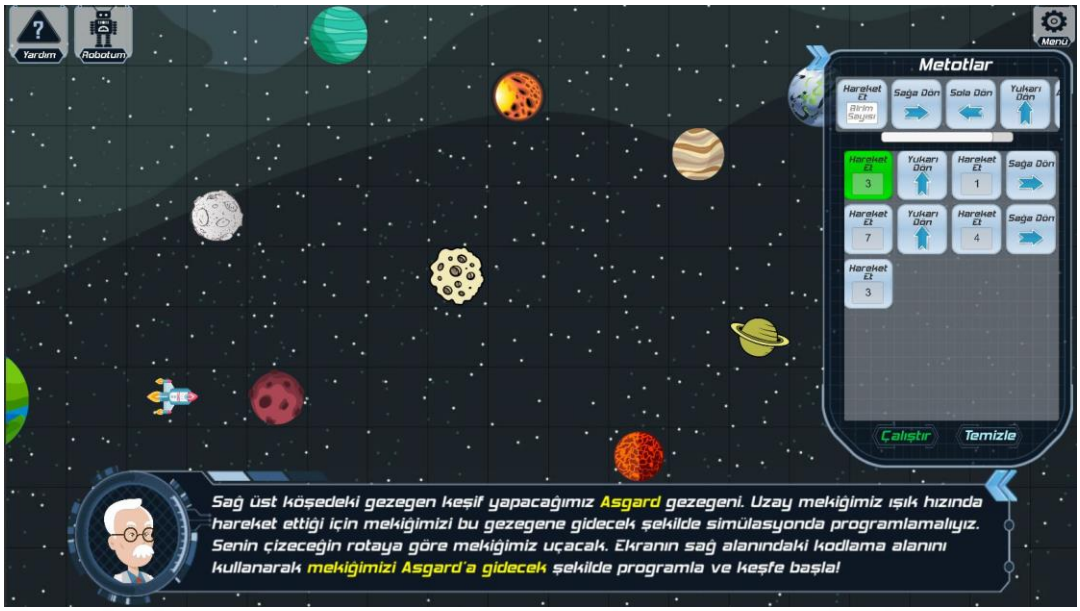


Figure 12. Going to Asgard mission

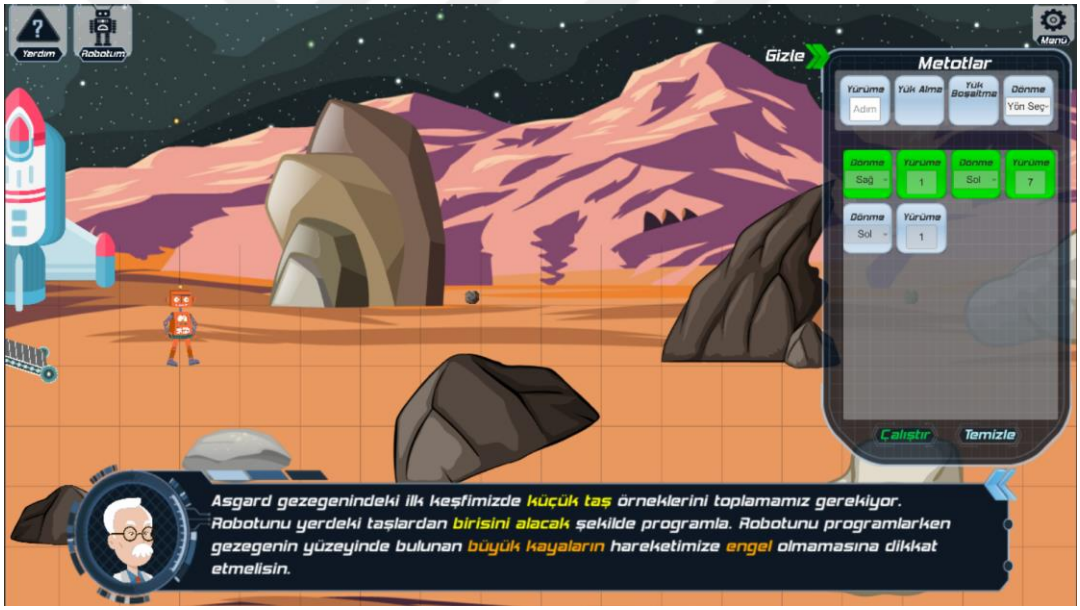


Figure 13. First exploration mission

the planet. Each of the stones has four alternative locations, and they are positioned randomly in one of the alternatives. This ensures that students are not able to use the same code sequence as their peers to solve the problem in this mission. In this activity, students need to program their robot so that it can collect the stones without hitting the huge rocks or leaving the exploration area. In order to write the code of their solutions to a problem, students need to understand the problem and divide it

into smaller parts and generate a strategic solution to these smaller parts (Gomes & Mendes, 2007; Lahtinen, Ala-Mutka, & Järvinen, 2005). Therefore, the collection task of the small stones is divided into two main parts. In the first part, students program their robot just to pick one of the stones and then clear the coding panel to program their robot to put the stone into the gathering point in front of the ship. In the second part of the mission, students are asked to program their robot to collect the other small stone and put it into to the gathering point within a single execution of their code.

When students finish collecting the stones and have put them in the gathering point, they will be asked to program the spaceship to travel back to the Earth from Asgard. Students program their spaceship according to the same principles that were used for going to the Planet Asgard.

In the next activity, students are introduced to the inheritance concept and why such a feature is needed within an authentic problem situation in the story. According to the story of the game, the professor welcomes the player's robot when it comes back to the TSA. In an animation, the professor informs students that the analysis of the stones from the first mission on Asgard was promising. The professor adds that scientists in the TSA need liquid and gas samples from Asgard in order to decide whether this planet is suitable for humans to live.

In the meantime, the professor states that "they have little time left and they need a new robot to analyze the samples and accelerate the exploration mission." Therefore, students are asked to create a new robot class, analyzer robot, for their final exploration mission. This new robot class needs to be able to collect solid and gas samples from Asgard, along with having all properties of previously defined robot class. Therefore, students will need to define the new class which will be

inherited from the robot class and have its own unique properties. The previous Class Definer Machine was improved with a new add-on that enables students to select the base class while defining a new one (see Figure 14). When students define the new class according to the instructions of the professor, they will instantiate an instance of the newly defined class the same way they did in the second activity.

Figure 14. Inherited class define activity

When students build their analyzer robot, they need to define new methods of the robot according to the given directions. In this activity, students use polymorphic methods by defining new methods and altering old ones. The professor indicates that the new robot should have the ability to collect and analyze the samples according to the physical state of the matter. The need and the underlying logic of polymorphic methods will be explained to students by embedding into the story of the game. The so-called Method Definer Machine was improved with a new add-on which shows the inherited methods of the class on the left side and the newly defined polymorphic

methods on the right side of the panel (see Figure 15). The same method defining procedures are held for the polymorphic method defining activity.

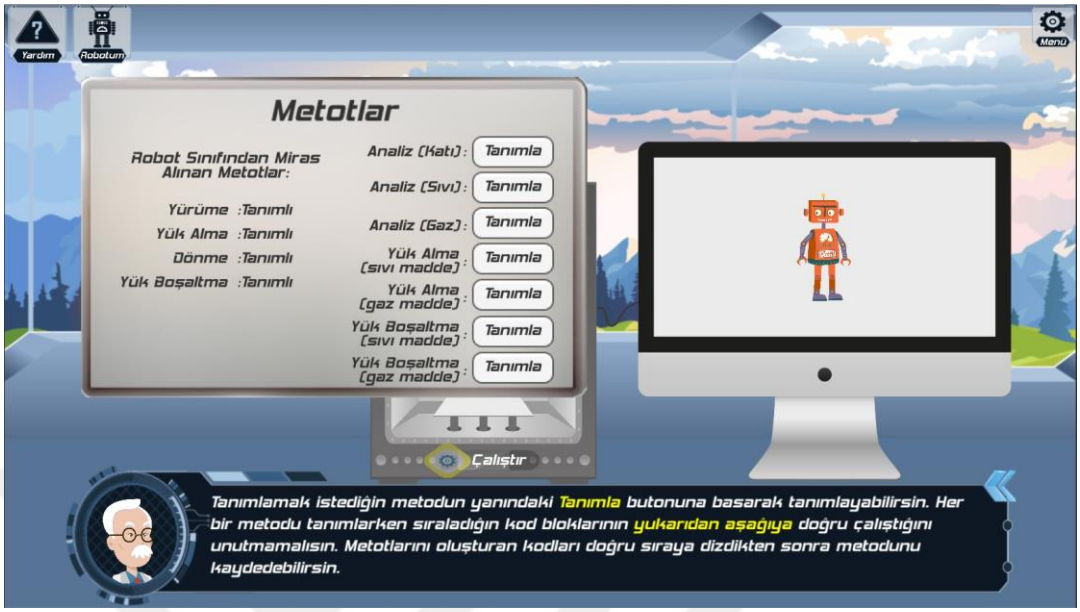


Figure 15. Defining polymorphic methods

After finishing the polymorphic method activity, students play a reinforcement activity for encapsulation concept. This time students try to establish a connection between the newly created analyzer robot and the analysis machine in the TSA station so that collected samples will be analyzed while it is in Asgard. The aim of this activity is to emphasize another use of encapsulation concept, which is hiding unnecessary details of a complex operation from users. This activity has the same layout as the previous encapsulation mission, except for the fact that the encapsulation panel is part of the analyze machine.

In the last activity of the game, students go on their final exploration mission on the planet Asgard with their freshly created analyzer robot. The second exploration on Asgard is in a different part of the planet so that students will encounter different obstacles on the planet's surface. Another important difference of

this activity is that students use polymorphic methods of previously defined as the "pick an object" method and the "put an object" method (see Figure 16).

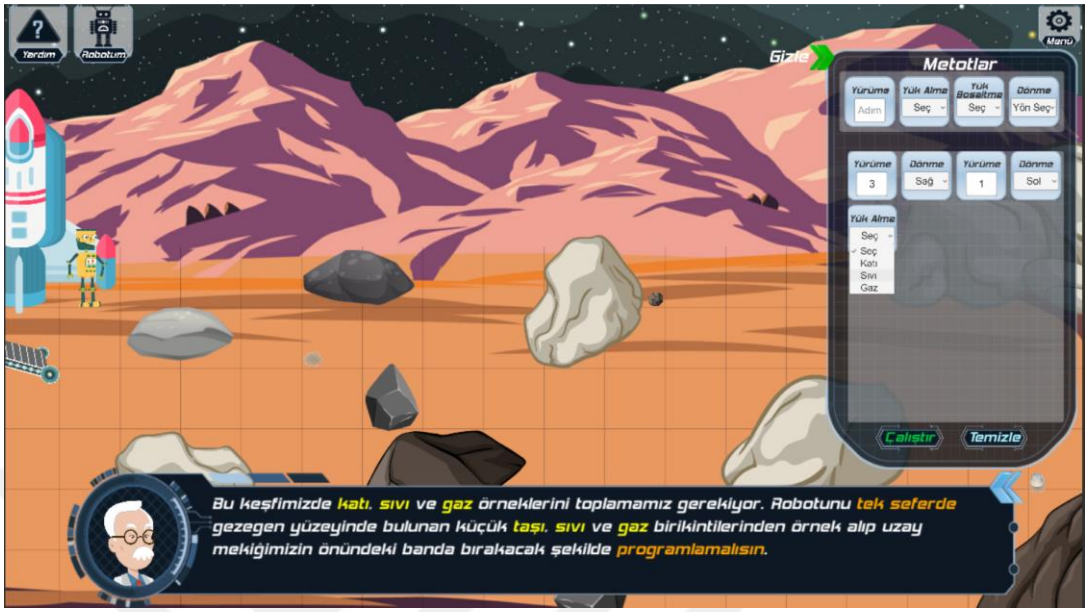


Figure 16. Final explorations in Asgard

With a dropdown list addition to the method blocks, students are able to select the physical state of the material that they need to pick or put. Thus, the necessity of the polymorphic methods are clarified in an authentic problem situation. In the final episode of the game, there is a small stone, a puddle, and a gas beam on the surface of the planet, and students have to program their robot to collect these three samples from the surface and put them in the gathering point in a single execution of their codes. When all three objects are collected, a final animation of the game will play.

3.4 Instruments

In the present study, four sets of data collection instruments were used: (1) a Creative Problem-Solving Test (Özkök, 2005), (2) an Attitude Scale for Serious Game Assisted Programming Learning (Keçeci, Alan, & Zengin, 2016), (3) a pre-test, and

(4) a post-test. The aim of the creative problem-solving test (see Appendix A) was to measure students' creative problem-solving skills. The test consists of 30 multiple-choice questions, each with 5 options; there is one correct answer and four distracters for each question. Each correct answer was assigned 1 point while each wrong answer was assigned 0 points. The Cronbach coefficient alpha of the test is 0.94. Ten of the questions in the test cover the identification of a problem, twelve involve the decomposition of a problem, and the remaining eight questions are about interpretation and making judgement skills.

The attitudes for serious game assisted programming learning were indicated on a 5-point Likert scale that ranged from (1) strongly disagree to (5) strongly agree (see Appendix C). There are 28 statements, 22 positive and six negative ones in the scale, and the Cronbach coefficient alpha reliability of the test is 0.833. Students were divided into two groups based on their attitude, either positive or with negative. Each statement was graded according to the weight of the items, with 1 point corresponding to strongly disagree to 5 points corresponding to strongly agree. The negative items in the scale were graded reversely.

There was no single test that evaluated both the conceptual knowledge of OOP and CT skills at the same time, for which reason the researcher prepared a test by adapting items from three different tests to measure students' conceptual knowledge of OOP and CT skills (see Appendix E). Eight questions of the test were adapted from an object-oriented computer programming semantic knowledge instrument (Gerola, 1997). The internal consistency reliability of alpha was 0.614. Eight questions were adapted from another instrument used in a doctoral dissertation (Pitsatorn, 2003). Pitsatorn (2003) and two instructors checked the tests and answers to assure reliability. The last three questions, questions 17 to 19, were adapted from

another study (Basu, 2016) in which students' science and CT learning were measured. The reliability of the instrument was not provided by the researcher. Table 5 shows the distribution of the adapted test items. After adapting all these items into one instrument, an instructor, a researcher and I matched learning objectives with measurement items (see Table 6) and checked pre- and post-tests in order to assure the reliability of the developed instrument. Additionally, the Cronbach alpha coefficient alpha of the instrument was also calculated for the pre-test and the post-test based on the answers of participants of the study. The Cronbach alpha coefficient of the pre-test was .83, and of the post-test was .63.

Table 5. Distribution of the Adapted Test Items

Test item number	Adapted from	Measured learning unit
1 - 8	Gerola, 1997	Conceptual knowledge of OOP
9 - 16	Pitsatorn, 2003	Conceptual knowledge of OOP
17 - 19	Basu, 2016	Computational thinking skills

While the aim of the pre-test was to measure students' existing conceptual knowledge of OOP and CT skills, the aim of the post-test was to measure students' conceptual knowledge of OOP and CT skills after they studied the learning unit with the game developed. The order of the questions and options were changed in the post-test. Seventeen of the questions were multiple-choice, each with four options; there was one correct answer and three distracters for each question. The last two questions of the test were open-ended questions which asks students to provide a solution to the given problems by building a solution algorithm and writing a simple program with pseudo codes. Each correct answer was graded out of 5 points; each wrong answer received 0 points.

Table 6. Learning Objectives and Corresponding Measurement Item Numbers

Instructional objective	Corresponding test item
Explain class concept	4
Identify object concept	5
Distinguish a class from an object	3
Distinguish object instantiation from class declaration process	10
Give an example of a class and instance from the class	12
State the roles of class attributes	8
Explain object instantiation process	2
State the difference between attributes of a class and attributes of an object	15
Define method concept	16
Explain how classes communicate with each other	13
Explain encapsulation concept	6
Explain the role of encapsulation in object-oriented programming	11
Explain polymorphism concept	7
Explain method overriding process	14
Differentiate a base (derived) class from a sub-class in an inheritance relationship	9
List the characteristics of object-oriented programming	1
Understand conditional statements	17
Write a conditional statement	18
Design a step-by-step solution to a problem	19

3.5 Data collection procedures

Prior to the data collection and treatments of the study, ethical approval (see Appendix F) was obtained from the Committee on Human Research of Boğazici University (İNAREK). The study was conducted in three sessions — pre-test, treatment and post-test — that took place on two different days over a period of 2 weeks, according to the availability of the participants.

Data collection and the experiment took place in the computer laboratories of the Faculty of Education at the university where the students study. Before starting

the experiment, the researcher informed the participants about the procedure of the experiment and indicated that they were free to participate (or not) in the experiment. The researcher obtained a written consent form from the students who volunteered to participate in the experiment (see Appendix G).

In two groups, freshman and sophomore students, all participants were given two tests: a Creative Problem-Solving Test (Özkök, 2005) and a pre-test on conceptual knowledge of OOP and CT skills. The pre-test session lasted approximately 50 minutes. A week after the pre-test phase, the second and third sessions of the study were performed on the same day. In the second part of the study, students in both groups played the game developed, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard), under the supervision of the researcher and the instructor of the course for about one and a half lesson period (90 minutes). After a 15-minute break, in the last session of the experiment the post-test was administered to students to measure conceptual knowledge of OOP and CT skills. At the end of the third phase, the researcher administered an attitude scale for serious game-assisted programming learning. The administration of the post-test and attitude scale took 40 minutes.

3.6 Data analysis

In order to answer the research questions, a series of different statistical tests were conducted. Data sets of the students' pre-test scores, post-test scores, creative problem-solving test scores and the attitude scale scores were first matched for each student in both groups. The data were then checked to ensure that each student had scores for all five measurements. Four students who did not have all of these scores were dropped from the study. For the data analysis of this study, quantitative

methods were used. First, the descriptive statistics of the achievement scores (difference between pre-test and post-test) and the normal distributions of each group's data were examined before conducting hypothesis testing through either parametric or nonparametric methods. In all statistical tests, the IBM SPSS statistical software (Version 24) was used.

The first question (Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience?) was answered as follows. First, the Shapiro-Wilk normality test was applied in order to check the distribution of students' pre-test scores and post-test scores. The post-test scores were normally distributed, but the pre-test scores were not distributed normally. Therefore, a nonparametric, Wilcoxon signed-rank test was conducted in order to analyze the difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills.

For the second question, (Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills of undergraduate students with procedural programming experience?), first, Shapiro-Wilk test was applied in order to check the distribution of the data. Then, a parametric, paired-samples t-test was conducted because the data was distributed normally.

In order to answer the third question, (Is there any significant difference between the achievement scores on conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience, and of undergraduate students with procedural programming experience?), the following methods were implemented. First, a Shapiro-Wilk test was conducted to check the distribution of

the achievement scores of students in both groups. Then, to compare the matched groups, an independent-samples t-test was used because the achievement scores of students with no programming experience and students with procedural programming experience were distributed normally.

The fourth question of the study (To what extend do the students' creative problem-solving skills and attitudes towards digital game-based learning of programming influence the students' achievement score on the conceptual knowledge of OOP and CT skills?) was answered by conducting a two-way ANOVA test. In order to test whether the level of creative problem-solving skills and attitudes towards digital game-based learning of programming together or pairwise influence the students' achievement scores, a general linear model 2x2 ANOVA test was conducted. Additionally, a series of Pearson's *r* and Spearman's rho tests were conducted to analyze the correlations among students' CPSS, attitudes towards digital game-based learning of programming and achievement scores in detail. First, a series of Shapiro-Wilk test was conducted to assess the distribution of students' data. Then, a Pearson correlation coefficient was computed if the data was normally distributed, and a Spearman's rho was computed if the data was not normally distributed.

CHAPTER 4

RESULTS

This chapter provides results of the data analyses conducted to answer the research questions. Specific findings for each research question are presented under title of the each group of research questions.

4.1 Learning gain of freshman and sophomore students

4.1.1 Learning gain of freshman students

Research question 1: Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience?

In order to analyze the pre-test and post-test scores of freshman students without programming experience, first normality of the data was checked to decide the type of statistical test to be conducted. A Shapiro-Wilk's test showed that the post-test scores were normally distributed ($p > .05$) but that pre-test scores were not (see Table 7). Therefore, a nonparametric, Wilcoxon signed-rank, test was conducted to analyze the difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills.

Table 7. Shapiro-Wilk Result of Pre-test and Post-test Scores

	Statistics	*df*	*Sig.*
Pre-test	.797	30	.000
Post-test	.959	30	.295

A Wilcoxon signed-rank test was conducted to examine the difference in the pre-test and post-test scores of students without programming experience. Table 8 shows the descriptive statistics of pre-test and post-test scores of freshman students. A Wilcoxon signed-rank test (z = -4.797, p = 0.000) revealed that there was a statistically significant increase in the post-test scores of students after playing the developed game with a large (r = .87) (Rosenthal & Rosnow, 1984) effect size (see Table 9). The median score on conceptual knowledge of OOP and CT skills test increased from pre-test (Md = 5.00) to post-test (Md = 40.00) after playing the developed game.

Table 8. Descriptive Statistics of the Pre-test and Post-test Scores of Freshman Students

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	10.67	5.00	30	12.229	2.233
Post-test	40.00	40.00	30	16.713	3.051

Table 9. Wilcoxon Signed Rank Test for Pre-test and Post-test

		N	Mean Rank	Sum of Ranks
Post-test – Pre-test	Negative Ranks	0[a]	.00	.00
	Positive Ranks	30[b]	15.50	465.00
	Ties	0[c]		
	Total	30		

a. post-test < pre-test; b. post-test > pre-test; c. post-test = pre-test

4.1.1.1 Freshman students' learning gain on conceptual knowledge of OOP

Research question 1a: Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP of undergraduate students with no programming experience?

In order to analyze freshman students' pre-test and post-test scores on conceptual knowledge of OOP, first the normality of the data was checked to decide which statistical test to be conducted. Table 10 shows the descriptive statistics of the pre-test and post-test scores. A Shapiro-Wilk's test revealed that the post-test scores were normally distributed but that the pre-test scores were not (see Table 11). Therefore, a nonparametric, Wilcoxon signed-rank test was conducted. The result of the test ($z = -4.793$; $p = 0.000$) showed that there was a statistically significant increase in the post-test scores of students after playing the developed game with a large effect size $r = .87$ (see Table 12). The median score on conceptual knowledge of OOP test increased from 0.00 to 35.00 after playing the developed game.

Table 10. Descriptive Statistics of Freshman Students' Pre-test and Post-test Results of Conceptual Knowledge of OOP

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	6.67	.00	30	9.589	1.751
Post-test	33.50	35.00	30	14.090	2.573

Table 11. Shapiro-Wilk Result of the Pre-test and Post-test Scores on Conceptual Knowledge of OOP

	Statistics	*df*	*Sig.*
Pre-test	.689	30	.000
Post-test	.966	30	.427

Table 12. Wilcoxon Signed Rank Test for Pre-test and Post-test Scores on Conceptual Knowledge of OOP

		N	Mean Rank	Sum of Ranks
Posttest - Pretest	Negative Ranks	0[a]	.00	.00
	Positive Ranks	30[b]	15.50	465.00
	Ties	0[c]		
	Total	30		

a. Post-test < pre-test; b. Post-test > pre-test; c. Post-test = pre-test

The frequency distribution of the number of correct answers to questions involving OOP concepts of freshman students are shown in the Table 13. Overall, there was an increase in the number of correct answers to all of the questions for the instructional objectives after playing the developed game. The learning objectives with the highest improvement were: stating the roles of class attributes, distinguishing object instantiation from class declaration process, stating the difference between attributes of a class and attributes of an object, explaining how classes communicate with each other, and listing the characteristics of OOP.

Table 13. Frequency Distribution of Freshman Students' Number of Correct Answers for OOP Concepts

Instructional Objective	Correct Answers	
	Pre-test	Post-test
Explain class concept	1	13
Identify object concept	4	12
Distinguish a class from an object	9	18
Distinguish object instantiation from class declaration process	1	15
Give an example of a class and instance from the class	1	6
State the roles of class attributes	1	16
Explain object instantiation process	2	11
State the difference between attributes of a class and attributes of an object	4	18
Define method concept	2	8
Explain how classes communicate with each other	4	18
Explain encapsulation concept	2	13
Explain the role of encapsulation in object-oriented programming	6	18
Explain polymorphism concept	0	4
Explain method overriding process	2	10
Differentiate a base (derived) class from a sub-class in an inheritance relationship	0	6
List characteristics of object-oriented programming	1	15

4.1.1.2 Freshman students' learning gain on CT skills

Research question 1b: Is there any significant difference between the post-test and pre-test scores on CT skills of undergraduate students with no programming experience?

In order to analyze the pre-test and post-test scores on CT skills of freshman students' without programming experience; the normality of the data was checked to decide which statistics to be used. The descriptive statistics of pre-test and post-test scores of sophomore students were presented in Table 14. Shapiro-Wilk's test showed that the pre-test scores and post-test scores were not normally distributed (see Table 15). Therefore, a Wilcoxon signed-rank test was conducted to analyze the difference between the post-test and pre-test scores on CT skills of freshman students.

Table 14. Descriptive Statistics of Freshman Students' Pre-test and Post-test Scores on CT Skills

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	4.00	5.00	30	3.806	.695
Post-test	6.50	7.50	30	5.438	.993

Table 15. Shapiro-Wilk Result of Pre-test and Post-test Scores on CT Skills

	Statistics	*df*	*Sig.*
Pre-test	.253	30	.000
Post-test	.240	30	.000

A Wilcoxon signed-rank test ($z = -2.500$; $p = 0.012$) revealed that there was a statistically significant increase in the post-test scores of students after playing the developed game with a medium ($r = .45$) effect size (see Table 16).

Table 16. Wilcoxon Signed Rank Test for Freshman Students' Pre-test and Post-test Scores on CT skills

		N	Mean Rank	Sum of Ranks
Post-test – Pre-test	Negative Ranks	3[a]	3.50	10.50
	Positive Ranks	10[b]	8.05	80.50
	Ties	17[c]		
	Total	30		

a. Post-test < pre-test; b. Post-test > Pre-test; c. Post-test = Pre-test

The frequency distribution of the number of correct answers to questions for the instructional objectives involving CT skills of freshman students are shown in the Table 17. Overall, there was an increase in the number of correct answers to all three questions for the instructional objectives after playing the developed game. The highest improvement was in the learning objective of writing a conditional statement.

Table 17. Frequency Distribution of Freshman Students' Number of Correct Answers for CT Skills

Instructional Objective	Correct Answers	
	Pre-test	Post-test
Understand conditional statements	18	20
Write a conditional statement	6	14
Design a step-by-step solution to a problem	0	5

4.1.2 Learning gain of sophomore students

Research question 2: Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills of undergraduate students with procedural programming experience?

In order to analyze the pre-test and post-test scores of sophomore students with procedural programming experience, the normality of the data was checked to decide which statistical test to be used. The descriptive statistics of pre-test and post-test scores of sophomore students are presented in Table 18. The students' pre-test scores and post-test scores were normally distributed, as assessed by a Shapiro-

Wilk's test (see Table 19). Therefore, a parametric, paired-samples t-test was conducted to analyze the difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills. The analysis ($t(30) = 4.558$, $p < 0.001$) showed that playing the developed game elicited a statistically significant increase in the post-test mean scores compared to the pre-test mean scores with a large effect size (Cohen's d = .92) (see Table 20).

Table 18. Descriptive Statistics of Pre-test and Post-test Scores of Sophomore Students

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	40.32	45.00	31	14.772	2.653
Post-test	53.48	50.00	31	13.721	2.464

Table 19. Shapiro-Wilk Result of Pre-test and Post-test Scores of Sophomore Students

	Statistics	*df*	*Sig.*
Pre-test	.955	31	.211
Post-test	.973	31	.614

Table 20. Paired Sample Test for Post-test and Pre-test of Sophomore Students

	Mean	Std. Deviation	*t*	*df*	Sig. (2-tailed)	Cohen's d
Posttest-Pretest	13.161	16.077	4.558	30	.000	.923

4.1.2.1 Sophomore students' learning gain on conceptual knowledge of OOP

Research question 2.a: Is there any significant difference between the post-test and pre-test scores on conceptual knowledge of OOP of undergraduate students with procedural programming experience?

In order to analyze the pre-test and post-test scores of sophomore students with procedural programming experience, the normality of the data was checked to

decide which statistical test to be used. The descriptive statistics of pre-test and post-test scores of sophomore students were presented in Table 21. Pre-test scores and post-test scores of students were normally distributed as assessed by Shapiro-Wilk's test (see Table 22). Therefore, a parametric, paired-samples t-test was conducted to analyze the difference between the post-test and pre-test scores on conceptual knowledge of OOP.

Table 21. Descriptive Statistics for Sophomore Students' Pre-test and Post-test Scores of Conceptual Knowledge of OOP

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	32.26	35.00	31	12.964	2.328
Post-test	42.58	45.00	31	11.963	2.149

Table 22. Shapiro-Wilk Result of Sophomore Students' Pre-test and Post-test on Conceptual Knowledge of OOP

	Statistics	*df*	*Sig.*
Pre-test	.935	31	.060
Post-test	.966	31	.413

The result of the test ($t(30) = 3.359$, $p = 0.002$) showed that playing the developed game elicited a statistically significant increase in the post-test mean scores compared to the pre-test mean scores, with a large ($d = .83$) effect size (see Table 23).

Table 23. Paired-Samples t-Test for Sophomore Students' Post-test and Pre-test Scores on Conceptual Knowledge of OOP

	Mean	Std. Deviation	*t*	*df*	Sig. (2-tailed)	Cohen's d
Post-test – Pre-test	2.839	4.705	3.359	30	.002	.827

The frequency distribution of the number of correct answers for questions involving OOP concepts of sophomore students are shown in Table 24. Overall, there was an increase in the number of correct answers to 15 of the questions for the instructional objectives after playing the developed game. The learning objectives with the highest improvement were: stating the difference between attributes of a class and attributes of an object, distinguishing object instantiation from class declaration process, explaining method overriding process and explaining class concept. However, there was not an increase or a decrease in one of the instructional objectives which is explaining encapsulation concept. In addition, the number of correct answers to pre-test and post-test questions for three instructional objectives was decreased after playing the developed game. These three learning objectives were: explaining polymorphism concept, differentiating a base (derived) class from a sub-class and listing characteristics of OOP.

Table 24. Frequency Distribution of Sophomore Students' Number of Correct Answers for OOP Concepts

Instructional Objectives	Correct Answers	
	Pre-test	Post-test
Explain class concept	12	21
Identify object concept	17	19
Distinguish a class from an object	22	28
Distinguish object instantiation from class declaration process	8	21
Give an example of a class and instance from the class	13	17
State the roles of class attributes	6	9
Explain object instantiation process	9	15
State the difference between attributes of a class and attributes of an object	8	23
Define method concept	9	13
Explain how classes communicate with each other	14	16
Explain encapsulation concept	14	14
Explain the role of encapsulation in object-oriented programming	15	16
Explain polymorphism concept	15	10
Explain method overriding process	10	20
Differentiate a base (derived) class from a sub-class in an inheritance relationship	16	12
List characteristics of object-oriented programming	12	10

4.1.2.2 Sophomore students' learning gain on CT skills

Research question 2.b: Is there any significant difference between the post-test and pre-test scores on CT skills of undergraduate students with procedural programming experience?

In order to analyze the pre-test and post-test scores of sophomore students with procedural programming experience, the normality of the data was checked to decide which statistics to be used. By inspecting the boxplots of the data, two outliers were found and excluded from the analysis. The descriptive statistics of pre-test and post-test scores of sophomore students are presented in Table 25. The Pre-test scores and post-test scores of students were not normally distributed as assessed by Shapiro-Wilk's test (see Table 26). Therefore, a nonparametric, Wilcoxon signed-rank test was conducted in order to analyze the difference between the post-test and pre-test scores on CT skills. The test ($z = -2.849$; $p = 0.004$) revealed that there was a statistically significant increase in the post-test scores of students after playing the developed game with a large effect size ($r = .53$) (see Table 27). The sophomore students' median score in CT skills test increased from pre-test (Md = 10.00) to post-test (Md = 13.00) after playing the developed game.

Table 25. Descriptive Statistics for Sophomore Students' Pre-test and Post-test Scores of CT Skills

	Mean	Median	N	Std. Deviation	Std. Error Mean
Pre-test	8.62	10.00	29	4.411	.819
Post-test	11.66	13.00	29	3.801	.706

Table 26. Shapiro-Wilk Result of Sophomore Students' Pre-test and Post-test Scores on CT Skills

	Statistics	*df*	*Sig.*
Pre-test	.830	29	.000
Post-test	.769	29	.000

Table 27. Wilcoxon Signed Rank Test Result for Sophomore Students' CT Skills

		N	Mean Rank	Sum of Ranks
Post-test – Pre-test	Negative Ranks	2[a]	11.50	23.00
	Positive Ranks	16[b]	9.25	148.00
	Ties	11[c]		
	Total	29		

a. Post-test < Pre-test; b. Post-test > Pre-test; c. Post-test = Pre-test

The frequency distribution of the number of correct answers for questions involving CT skills of sophomore students are shown in the Table 28. Overall, there was an increase in the number of correct answers to all three questions for the instructional objectives after playing the developed game. Designing a step-by-step solution to a problem learning objective was the one with the highest improvement.

Table 28. Frequency Distribution of Sophomore Students' Number of Correct Answers for CT Skills

Instructional Objective	Correct Answers	
	Pre-test	Post-test
Understand conditional statements	26	27
Write a conditional statement	16	20
Design a step-by-step solution to a problem	8	21

4.2 Comparison of the achievement scores of freshman and sophomore students

Research question 3: Is there any significant difference between the achievement scores on conceptual knowledge of OOP and CT skills of undergraduate students with no programming experience, and of undergraduate students with procedural programming experience?

First, the students' achievement scores were calculated by subtracting their pre-test scores from the post-test scores. In order to analyze the achievement scores

of freshman students without programming experience and of sophomore students with procedural programming experience, the normality of the data was checked to decide which statistical test to be applied. Achievement scores for each group were normally distributed, as assessed by a Shapiro-Wilk's test (see Table 29). Thus, to compare the matched groups, an independent-samples t-test was conducted.

Table 29. Shapiro-Wilk Result of Achievement Scores of Freshman and Sophomore Students

	Statistics	*df*	*Sig.*
Freshman Students	.936	30	.072
Sophomore Students	.977	31	.734

An independent-samples t-test was run to determine if there were any significant difference in the achievement scores of freshman and of sophomore students. Table 30 shows the descriptive statistics of the achievement scores of freshman and sophomore students. There was homogeneity of variances, as assessed by Levene's test for equality of variances (see Table 31). The independent-samples t-test revealed that there was a significant difference between the achievement scores of the freshman students (M = 29.33, SD = 14.55) and of the sophomore students (M = 13.16, SD = 16.08); t (59) = 4.115, p < .001 (see Table 31).

Table 30. Descriptive Statistics for Students' Achievement Scores

Groups	Mean	Median	Std. Deviation	N	Std. Error Mean
Freshman	29.33	30.00	14.547	30	2.656
Sophomore	13.16	15.00	16.077	31	2.888

Table 31. Independent-Samples t-Test for Students' Achievement Scores

Levene Statistics				Sig.	Mean	Std. Error
F	Sign	*t*	*df*	(2-tailed)	Difference	Difference
.721	.399	4.115	59	.000	16.172	3.930

4.3 Covariate effects on the achievement scores

Research question 4: To what extent do the students' creative problem-solving skills (CPSS) and attitudes towards digital game-based learning of programming influence the students' achievement score on the conceptual knowledge of OOP and CT skills?

In order to test whether CPSS and the attitudes towards digital game-based learning of programming together or pairwise influence the students' achievement scores, a general linear model 2x2 ANOVA test was conducted.

Before conducting the test of covariate effects, participants' CPSS test scores and attitudes survey results were inspected to categorize the students. To categorize CPSS, the CPSS test scores that were lower than 15 out of 30 constituted the low-level group (n = 17), and the ones higher than or equal to 15 out of 30 constituted the high-level group (n = 44). To categorize students according to their attitudes towards digital game-based learning of programming, 105 points out of 150 points was designated as threshold because a 70 percent is set as a success rate. Hence, 32 students were assigned to the low-attitude group, while 29 students were assigned to the high-attitude group. An inspection of boxplots of the data showed that there were two outliers, which were excluded from the analysis. The normality of the data was assessed using a Shapiro-Wilk's normality test for each group of the design. Residuals were normally distributed (see Table 32), and there was homogeneity of variances (F = 2.421, df1 = 3, df2 = 55, p = 0.76).

Table 32. Shapiro-Wilk Result of Residuals for Achievement Scores

The Level of Creative Problem-Solving Skills	Attitudes	Statistics	*df*	*Sig.*
Low	Negative	.921	10	.365
	Positive	.807	5	.093
High	Negative	.917	20	.088
	Positive	.983	24	.939

A two-way ANOVA test was conducted to examine the effects of the level of CPSS and attitudes towards digital game-based learning of programming on students' achievement scores. According to a general linear model 2x2 ANOVA test, the following statistical outcomes were found (see Table 33):

(1) There was no statistically significant two-way interaction between the students' level of CPSS and attitudes towards digital game-based learning of programming on achievement scores, $F(1, 55) = .229$, $p = .634$.

(2) There was no statistically significant interaction between the students' level of CPSS and achievement scores, $F(1, 55) = .299$, $p = .586$.

(3) There was no statistically significant interaction between the students' attitudes towards digital game-based learning of programming and achievement scores, $F(1, 55) = 1.124$, $p = .294$.

Table 33. Two-way ANOVA Test for Freshman and Sophomore Students' Achievement Scores

Source	Type III Sum of Squares	*df*	Mean Square	*F*	*Sig.*	Partial Eta Squared
CPSS	72.624	1	72.624	.299	.586	.005
Attitude	272.624	1	272.624	1.124	.294	.020
CPSS * attitude	55.603	1	55.603	.229	.634	.004
Error	13338.833	55	242.524			
Total	39774.000	59				

Additionally, a series of Pearson's *r* and Spearman's rho tests were conducted to analyze the correlation between students' CPSS, attitudes towards digital game-based learning of programming and achievement scores in detail. The normality of the data was checked to decide which statistical test to be applied. Students' all scores were normally distributed but achievement score in CT skills was not (see Table 34). A Pearson correlation coefficient was computed if the data was normally

distributed, and a Spearman's rho was computed if the data was not normally distributed. Three outliers were found in the CT skills achievement data and excluded from the analysis. The following statistical outcomes were found (see Table 35):

(1) A Pearson correlation coefficient was computed to assess the relationship between the students' overall achievement scores and their CPSS scores. There was a weak negative, statistically non-significant, correlation between the two variables, $r(59) = -.153$, $p = .239$.

(2) A Pearson correlation coefficient was computed to assess the relationship between the students' achievement scores in OOP concepts and their CPSS scores. There was a weak negative, statistically non-significant, correlation between the two variables, $r (59) = -.167$, $p = .197$.

(3) A Spearman's rho was computed to assess the relationship between the students' achievement scores in CT skills and their CPSS scores. There was a weak positive, statistically non-significant, correlation between the two variables, $r_s (56) = .032$, $p = .809$.

(4) A Pearson correlation coefficient was computed to assess the relationship between the students' overall achievement scores and their attitude scores. There was a weak negative, statistically non-significant, correlation between the two variables, $r(59) = -.157$, $p = .226$.

(5) A Pearson correlation coefficient was computed to assess the relationship between the students' achievement scores in OOP concepts and their attitude scores. There was a weak negative, statistically non-significant, correlation between the two variables, $r (59) = -.172$, $p = .184$.

(6) A Spearman's rho was computed to assess the relationship between the students' achievement scores in CT skills and their attitude scores. There was a weak negative, statistically non-significant, correlation between the two variables, r_s (56) = -.013, p = .924.

(7) A Pearson correlation coefficient was computed to assess the relationship between the students' overall achievement scores and achievement scores in OOP concepts. There was a strong positive, statistically significant, correlation between the two variables, r (59) = .96, p < .001

(8) A Spearman's rho was computed to assess the relationship between the students' overall achievement scores and achievement scores in CT skills. There was a moderate positive, statistically significant, correlation between the two variables, r_s (56) = .338, p < .05.

Table 34. Shapiro-Wilk Result of CPSS, Attitude and Achievement Scores

	Statistics	*Df*	*Sig.*
CPSS Score	.973	61	.189
Attitude Score	.986	61	.737
Achievement Score in OOP Concepts	.977	61	.321
Achievement Score in CT Skills	.836	58	.000
Achievement Score	.976	61	.275

Table 35. The Correlations Between Students' CPSS, Attitudes and Achievement Scores

	CPSS	Attitude	Achv. in OOP Concepts	Achv. in CT Skills	Achv. Overall
CPSS	—				
Attitude	.186	—			
Achv. in OOP Concepts	-.167	-.172	—		
Achv. in CT Skills	.032	-.013	.092	—	
Achv. Overall	-.153	-.157	.960	.338	—

CHAPTER 5

DISCUSSION AND CONCLUSION

Previous research on digital game-based learning of computer programming has focused mostly on the motivational effects of serious games, and the findings of the majority of the studies were either based on anecdotal evidence or on initial evaluation results that fail to provide enough empirical data about students' learning performance, particularly students in Turkey. Therefore, the current research examined the effects of a serious game, Curious Robots: Operation Asgard (Meraklı Robotlar: Operasyon Asgard), on undergraduate students' learning performance on conceptual knowledge of OOP and CT skills.

In this chapter, the results of the data are discussed by referring to the literature, and possible implications of the findings are presented. Finally, the suggestions for future research and limitations of the study are provided.

5.1 Effects of a serious game on students' conceptual knowledge of OOP and CT skills

Conceptual knowledge of OOP and CT skills play an important role in understanding how problems are solved in OOP (Hadjerrouit, 1999; Liu, Cheng, & Huang, 2011; Wing J. M., 2006; Wing J. M., 2008). Students' problem-solving methods and skills in computer science are referred as CT in the recent literature (Aho, 2012; Lu & Fletcher, 2009; Wing J. M., 2006; Wing J. M., 2008). Fundamental concepts of OOP, on the other side, also have a critical role in understanding problems, designing and implementing solutions of problems (Hadjerrouit, 1999). Furthermore, various researchers have emphasized that some of the learning difficulties of novice

programmers may be related to the computational learning environment in which they are introduced to programming (Gomes & Mendes, 2007; Kölling, 1999b). Yet there has been little discussion about the effects of serious games on novice programmers' both conceptual knowledge of OOP and CT skills. This study focused on this issue and made a significant contribution to the literature by demonstrating inferential statistics.

The first two questions of the study focused on the effects of the developed game on students' learning of conceptual knowledge of OOP and CT skills. In order to answer these questions, the pre-test and post-test scores of freshman students without programming experience and sophomore students with procedural programming experience were analyzed using a series of statistical tests. A Wilcoxon signed-rank test was conducted to examine freshman students' data, while a paired-samples t-test was used to analyze sophomore students' pre-test and post-test scores. The analyses of both groups' data showed that both freshman and sophomore students significantly improved their conceptual knowledge of OOP and CT skills after playing the developed game. This result is consistent with the idea that serious games can be effective in fostering novice programmers' programming knowledge (Livovský & Porubän, 2014; Mathrani, Christian, & Ponder-Sutton, 2016; Miljanovic & Bradbury, 2017; Muratet, Torguet, Viallet, & Jessel, 2011; O'Kelly & Gibson, 2006; Phelps, Egert, & Bierre, 2005).

A more detailed analysis of the pre-test and post-test scores on conceptual knowledge of OOP was also conducted for both groups. The results reveal that students with no programming experience and students with procedural programming experience significantly improved their understanding of fundamental concepts of OOP such as class, object, method, encapsulation, inheritance and

polymorphism. Such findings corroborate the findings of other studies in the current literature (Livovský & Porubän, 2014; O'Kelly & Gibson, 2006; Phelps, Egert, & Bierre, 2005; Wong, Hayati, & Tan, 2016) by demonstrating inferential statistical analyses. Similarly, both freshman and sophomore students' pre-test and post-test scores on CT skills were analyzed. The analysis revealed that the mean post-test scores of freshman and sophomore students were significantly higher than their mean pre-test scores on CT skills. The sophomore students had completed a semester-long course on procedural programming before the experiment, so it was assumed that there would be no significant difference in the achievement scores of sophomore students on CT skills. Yet the significant increase in sophomore students' mean CT skills scores were a delightful surprise.

Additionally, a detailed analysis of the number of correct answers of freshman students for pre-test and post-test showed that there was an increase in the number of correct answers for all of the questions after playing the developed game. Freshman students had significant improvement in learning objectives such as stating the roles of class attributes, distinguishing object instantiation from class declaration process, stating the difference between attributes of a class and attributes of an object, explaining how classes communicate with each other, listing the characteristics of OOP and writing a conditional statement.

On the other hand, a detailed analysis of the sophomore students' number of correct answers to pre-test and post-test questions showed that there was an increase in the number of correct answers to questions for 15 of the instructional objectives after playing the developed game. Sophomore students had significant improvement in learning objectives such as stating the difference between attributes of a class and attributes of an object, distinguishing object instantiation from class declaration

process, explaining method overriding process and class concept. However, there was not an increase or a decrease in one of the instructional objectives which is explaining encapsulation concept. In addition, there was decrease in three of the instructional objectives in the sophomore students' number of correct answers to pre-test and post-test questions. These three learning objectives were explaining polymorphism concept, differentiating a base class from a sub-class and listing characteristics of OOP. One of the possible reasons of this result is that the sophomore students may have not carefully read the instruction and mission information texts in the game because they might have thought they had already known the topic thanks to their prior knowledge in procedural programming. Another possible reason of this result is the complexity of the activities involving these learning objectives. For example, there were seven different methods in one of the game activities (the ninth activity, defining polymorphic methods) that introduced the polymorphism concept, and this may have been overwhelming for the novice programmers. Therefore, it can be said that the game activities involving these four learning objectives were not effective for students who started computer programming with procedural programming and shifted to OOP, and these game activities need to be revised. This result appears to support the idea that the transition from procedural programming to OOP may cause problems for novice programmers (Hadjerrouit, 1999; Xinogalos, 2016) because freshman students who has no prior programming knowledge had improvement in each of these learning objectives.

This study differs from the previous studies in terms of the conceptual design of the learning activities and the scope of learning objectives. The serious games that were developed in the current literature focused on the goals of teaching conceptual knowledge of OOP and developing CT skills separately. The game developed in this

study, on the other hand, aimed to teach fundamental concepts of OOP along with enabling students to improve their CT skills by providing authentic problem situations. In order to provide a constructivist learning experience for novice programmers the game is developed based on the Experiential Gaming Model (Kiili, 2005) and the 4C/ID model (van Merriënboer, Clark, & de Croock, 2002). Overall, both models encourages the use of ill-structured problems in a learning environment to support discovery learning. For example, students are asked to program their robot to collect objects from the surface of the Asgard without hitting the obstacles on its way. The fundamental concepts of OOP and CT skills were integrated into the story of the game, and the level of difficulty of tasks in the game increased gradually. Furthermore, fantasy elements such as imaginary machines and planets were used to integrate OOP concepts into the story of the game and to provide visual representations of abstract concepts of OOP. The imaginary machines in the game play crucial role in the concretization of abstract concepts of OOP by enabling novice programmers to not to worry about the syntax of a real programming language. For example, in the developed game students create their robots in a class definer machine, and program its behaviors in a method definer machine via dragging-and-dropping code blocks. The class definer machine is used to concretize the class concept as a programmable chip, and the object concept as a robot by visually representing the processess of defining a class and object instantiation in the panel of the machine. The method definer machine, on the other hand, visualized the execution of code blocks on students' robots to enable students to test codes and observe its results. Hence, a constructivist learning approach was followed in the developed game to help students understand the necessity and possible usages of such concepts and CT skills.

Another difference between this study and the previous studies is the design of the instructions. The instructions in the game were conveyed by an APA to increase learners' motivation and lead to deep learning. Livovský and Porubän (2014) claimed that long texts in instructions affected students' learning negatively. Similarly, Sweller, van Merriënboer, and Paas (1998) claimed that human beings have a limited capacity for working memory, for this reason instructional materials should be designed by considering the learners' cognitive load. Therefore, in the current study some key points and concepts of OOP were highlighted in the instructions to lower the students' cognitive load. For example, critical points in a problem situation were highlighted in the instruction text to help novice programmers understand and analyze a problem before finding a solution to it. Thus, though the length of the instruction texts were long, it did not affect students' learning performance adversely.

This study makes a significant contribution to the literature by providing empirical data about the effects of serious games on novice programmers' conceptual knowledge of OOP and CT skills. This study showed that teaching fundamental concepts of OOP and CT skills through a game play experience can foster novice programmers' learning performance and help them overcome their learning difficulties. The integration of fundamental concepts of OOP and CT skills into the story of the game can be an effective way to teach programming through game play experience. Additionally, it is important that a game, which aims to teach programming, should offer students an opportunity to implement their solutions to the given problems, observe and reflect the results on a problem situation, and make necessary changes in their solutions. In this reflective observation phase of the learning experience, clear feedback plays a crucial role. Therefore, an immediate,

visual and textual, feedback mechanism should be provided in the game to inform and guide students about their missions and mistakes. It is thought that, in the light of the findings, serious game designers and instructors will have the opportunity to design effective games that help novice programmers to overcome their learning difficulties and improve their learning.

5.2 Comparison of achievement scores of students without programming experience and students with procedural programming experience

The third question of the study explored the differences between the mean achievement scores of freshman students with no programming experience and sophomore students with procedural programming experience. In order to answer the question, the pre-test scores of students were first subtracted from post-test scores, and then an independent-samples t-test was conducted to compare the achievement scores of the two groups.

Hadjerrouit (1999) and Xinogalos (2016) stated that novice programmers are likely to have problems when they are first introduced to procedural programming and then move to OOP. However, there has been little discussion about the effects of the transition from procedural programming to OOP on novice programmers' understanding of fundamental concepts of OOP and CT skills. Therefore, in order to provide a fresh insight into the current problem, this study compared the mean achievement scores of freshman students with no programming experience and sophomore students with procedural programming experience. The results reveal that freshman students ($M = 29.33$, $SD = 14.55$) have higher achievement scores than the sophomore students ($M = 13.16$, $SD = 16.08$). However, the sophomore students' mean average scores on both pre-test ($M = 40.32$, $SD = 14.77$) and post-test ($M =$

53.48, SD = 13.72) were higher than the freshman students' mean average scores on pre-test (M = 10.67, SD = 12.23) and post-test (M = 40.00, SD = 16.71). Therefore, it can be said that the developed game was effective in fostering both groups' learning performance although both groups did not have high mean post-test scores. Moreover, developed game was more effective in teaching computer programming to students without programming experience than to students with procedural programming knowledge. One of the possible explanations of this result is that sophomore students' existing knowledge of programming might have been a factor. In other words, it was more likely for freshman students to have a higher achievement scores than sophomore students because the former had no previous experience of programming. Therefore, a burst in the learning performance of computer programming of students, who have no programming experience, to some extent is likely to be expected from. The findings of this study showed that serious games can foster novice programmers' OOP knowledge and CT skills, and help them to overcome the problems derive from the transition from procedural programming to OOP.

5.3 The relationship among students' creative problem-solving skills, attitudes towards digital game-based learning of programming and learning

The fourth question of the study was to what extent students' creative problem-solving skills and attitudes towards digital game-based learning of programming influence the students' achievement score on the conceptual knowledge of OOP and CT skills.

In the current literature, researchers have stated that the knowledge of fundamental concepts of OOP and CT skills play important role in understanding and

solving problems in computer programming (Aho, 2012; Barr & Stephenson, 2011; Wing J. M., 2006). Additionally, researchers have advised that CT should be introduced to students as early as possible (Liu, Cheng, & Huang, 2011; Lu & Fletcher, 2009; Qualls & Sherrel, 2010). Therefore, in order to test whether the level of CPSS and attitudes towards digital game-based learning of programming together or pairwise influence the students' achievement scores, a general linear model 2x2 ANOVA test was conducted. The test revealed that there were no significant two-way or one-way interactions among the level of CPSS and attitudes towards digital game-based learning of programming on students' achievement scores. Additionally, a series of Pearson's *r* and Spearman's rho tests were conducted to analyze the correlation between students' CPSS, attitudes towards digital game-based learning of programming and achievement scores in detail. The tests revealed that there were only weak correlations among students' CPSS, attitudes towards digital game-based learning of programming and learning. Although the current literature indicates that CT and fundamental concepts of OOP are closely related to students' programming performance, the findings of the present study did not reveal a significant relationship between students' CPSS and achievement scores, and thus contradicted such arguments. One possible explanation of this result is that the items in the CPSS test required mostly knowledge of symmetry. Therefore, in order to have a better understanding of the nature of the relationship between students' problem-solving skills and programming performance, a follow-up study could be conducted with another instrument measuring CPSS with a wide range of items.

Furthermore, many researchers have studied students' attitudes towards the digital game-based learning of programming and have agreed on the positive effects of games on novice programmers motivation (Barnes, Richter, et al., 2007; Liu,

Cheng, & Huang, 2011; Mathrani, Christian, & Ponder-Sutton, 2016; Muratet, Torguet, Viallet, & Jessel, 2011; Ramírez-Rosales, et al., 2016; Wong, Hayati, & Tan, 2016). Some of the studies (Phelps, Egert, & Bierre, 2005; Wong, Hayati, & Tan, 2016) claim that serious games could be effective in fostering novice programmers' learning of programming based on the data of students' attitudes. The findings of the current study contradict such claims, this study showed that there was not a significant interaction between students' attitudes towards digital game-based learning of programming and their achievement scores. This study makes a significant contribution to the literature by demonstrating that fun and engaging aspects of serious games might be motivating, but it does not necessarily improve novice programmers' learning performance at a university level. Therefore, it is important that serious game developers, instructors and educational technologists should pay more attention to the instructional design of the activities than the motivational fun aspects of the game.

5.4 Implication for practice and recommendations for further research

The present study is the first to directly examine the effects of serious games on undergraduate students' conceptual knowledge of OOP and CT skills of Turkish students. The findings of the study, which show serious games can be effective in fostering novice programmers' programming knowledge and CT skills, are consistent with the current literature (Livovský & Porubän, 2014; Mathrani, Christian, & Ponder-Sutton, 2016; Miljanovic & Bradbury, 2017; Muratet, Torguet, Viallet, & Jessel, 2011; O'Kelly & Gibson, 2006; Phelps, Egert, & Bierre, 2005). Additionally, by providing empirical data on the current issue, this study has beneficial theoretical and practical implications for digital game-based learning of

programming, and may provide valuable information and guidance for serious game developers, educational technologists and instructors.

The developed game shares the same objectives with the majority of the studies in the literature, which is improving students' programming skills. Yet this study differs from other studies in terms of the conceptual design of the game. The majority of the studies in the literature focus on the learning objectives of the developed serious games, but few provide information about the instructional design of the activities (Laporte & Zaman, 2018). With this in mind, the learning activities of the developed game were developed based on Kiili's (2005) experiential gaming model and 4C/ID model (van Merriënboer, Clark, & de Croock, 2002) to encourage discovery learning. Kiili (2005) advised that serious games should enable students to test different solutions in an authentic problem situation to improve students' problem-solving skills and current knowledge on the topic. Therefore, to encourage discovery learning, the developed game adopted a problem-based learning approach by introducing fundamental concepts of OOP in authentic problem situations. In addition, the difficulty of the tasks in the game increase gradually as students make progress in the game, as indicated by the experiential gaming model and 4C/ID model. The findings of the current study reveal that novice programmers' understanding of fundamental OOP concepts and CT skills improved after playing the developed game. Therefore, from a practical point of view, serious game designers should consider providing a learning environment with authentic problems to support discovery learning.

A well-designed visual and textual feedback mechanism is the other unique feature of the developed game. In the developed game, supportive and procedural information was provided to students via a mission information panel, an instruction

panel and a help menu. For example, while students are introduced with class concept in mission information panel, specific instructions and points to take into consideration to define a class are presented in the instruction panel and help menu. Abstract concepts of OOP were represented as concrete objects in the game. For example class concept is represented as a programmable chip which contains the specifications of a robot. Morevover, CT skills were practiced in a simulation environment to help students understand the necessity and the forms of utilization of such concepts and skills. For example, students are asked to program their robot to pick an object from the surface of the Asgard without hitting the obstacles on its way. Barnes, Chaffin, et al. (2007) and Esteves et al. (2011) advise that serious games for programming should have a well-designed feedback mechanism to help students overcome their learning difficulties, and the findings of this study are in agreement with these arguments. Similarly, Kiili (2005) reported that feedback in serious games helps learners understand the deficiencies in their solution and thus improves their solutions to create ones that are more effective. From a practical standpoint, these findings suggest that serious game developers should establish a well-designed feedback mechanism to help novice programmers to overcome their learning problems.

In order to provide guidelines that are more specific for serious game development, further research with a number different versions of the current game could be conducted to deeply analyze the effects of different components of serious games on novice programmers' learning performance. Additionally, more research could be conducted with different student groups to find out whether or not the effects of the developed game can be generalized to a greater population with different properties.

5.5 Limitations of the study

The first limitation of the study is about the generalizability of the findings because of the convenient sampling procedures that were used. In order to generalize the findings of the study to a larger population of novice programmers, a replication of the study with true experimental design should be conducted.

Secondly, using an immediate post-testing phase in the study may be considered as another limitation. A delayed post-test for measuring the students' conceptual knowledge of OOP and CT skills could be conducted. Nonetheless, this did not seem applicable in the present study due to practical constraints, particularly the lack of access to the students' class time.

Another limitation of the study is using the same instrument as a pre-test and a post-test. A follow up study could be conducted with two different instruments measuring the same learning objectives. However, this did not seem applicable in the present study due to the lack of a second instrument which evaluates the conceptual knowledge of OOP and CT skills.

In order to have a better understanding of the effects of the developed game on novice programmers' achievement scores of conceptual knowledge of OOP and CT skills, another study with a control group can be conducted. However, it did not seem feasible in the current study because of the limited number of participants.

Finally, it is better to be cautious to generalize the findings of this study to serious games for learning programming of all programming languages such as C#, Java or Python. More research is necessary to find out whether or not the effects of the design principles used in this game can be generalized to other serious games with different languages and age groups.

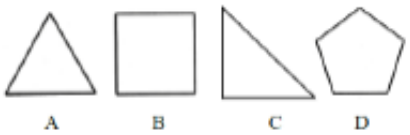
APPENDIX A

CREATIVE PROBLEM-SOLVING SKILL TEST

Creative Problem Solving Skill Test

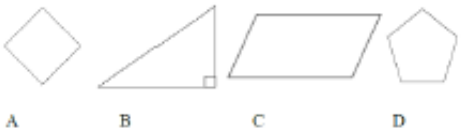
This test aims to measure your creative problem-solving skills and it consists of 30 multiple-choice questions. Please think about each question carefully. There is only one correct answer for each question. Mark the options that you think are correct with a circle for each question.

1. Which one/ones of the following shapes is/are not regular polygons?

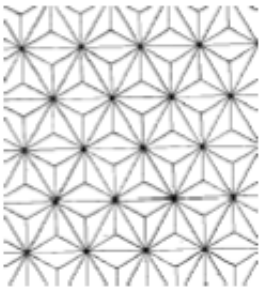


   a. C and D
   b. Only C
   c. A and B
   d. Only D
   e. All of them are regular polygons

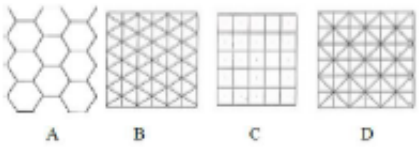
2. Which one/ones of the following polygons is/are symmetrical?
   a. Only A
   b. A and D
   c. Only C
   d. C and D
   e. All of them are symmetrical



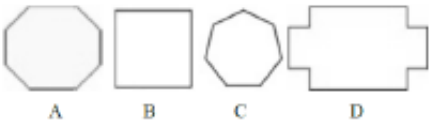
3. In which of the following, are the polygons seen in the picture below given correctly?
   a. A triangle and a square
   b. Only a hexagon
   c. Only a triangle
   d. A triangle, a hexagon and a parallelogram
   e. A triangle and a parallelogram



4. Which one/ones of the following shapes consist of hexagons?
   a. B and D
   b. A and C
   c. A and B
   d. Only A
   e. All of them

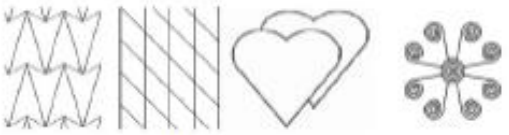


5. Symmetry: The quality of different objects matching proportionally.
   According to this definition which one/ones of the following terms are synonyms of symmetry?
   Pi, axis, identical, right angle
   a. Only Pi
   b. Only identical
   c. Axis and identical
   d. Axis and right angle
   e. None of them
6. Which one/ones of the following shapes is/are a regular quadrilateral polygon?
   a. A and B
   b. C and D
   c. Only B
   d. Only C
   e. A and C



7. Which one of the following gives the shapes that has symmetrical reflection?
   a. A and D
   b. Only A
   c. Only B
   d. C and D
   e. B and D

8. Which one of the following gives the shapes that does not have symmetrical shifting?
   a. Only C
   b. A and C
   c. B and C
   d. Only D
   e. All of them



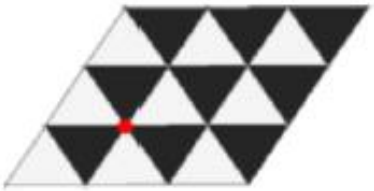
A B C D

9. Which one of the following gives the shapes that have symmetrical rotation?
   a. A and B
   b. A and D
   c. A and C
   d. Only D
   e. All of them

A B C D

10. Which of the following is an example of symmetry in nature?
    a. Honey comb
    b. Chamomile
    c. Pine cone
    d. Marble
    e. All of them

11. What is the angle of the junction point in the parallelogram below?
    a. 180°
    b. 270°
    c. 60°
    d. 120°
    e. 360°

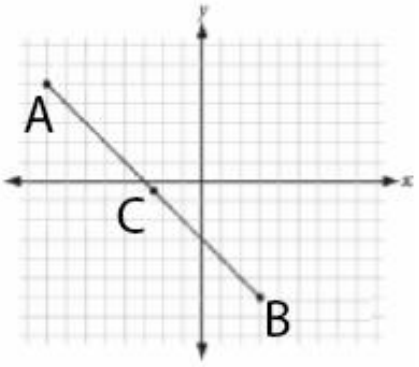


12. Which of the symmetry methods can be applied to the shape below?
    a. Only rotation
    b. Rotation and reflection
    c. Reflection and shifting
    d. Only reflection
    e. None

13. Which of the following gives the shapes that have the same number of rotation applied?
    a. B and C
    b. A and C
    c. A and B
    d. B and D
    e. None of them

A B C D

14. What are the coordinates of A, B and C points on the given line segment below?
    a. A (8, 5), B (3, 6), C(2.5, 0.5)
    b. A (5, -8), B (-6, 3), C(-0.5, -2.5)
    c. A (5, 8), B (6, 3), C(0.5, 2,5)
    d. A (-8, 5), B (3, -6), C(-2.5, -0,5)
    e. None of them



15. Which of the following statements is true for the shapes given below?
    a. Reflection symmetry method is used in A
    b. Shifting symmetry method is used in B
    c. Shifting and reflection symmetry methods are used C
    d. Reflection symmetry method was used in A and B
    e. None of them

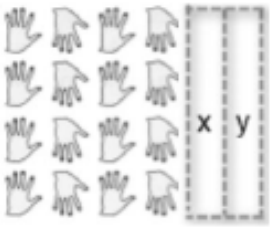
A B C

16. Which of the following can NOT be seen in the picture below?
    a. Rhtym
    b. Repetition
    c. Direction
    d. Balance
    e. Rotation

17. What are the angles of each marked point in A, B and C shapes?
    a. A (360°), B (180°) and C (270°)
    b. A (240°), B (180°) and C (120°)
    c. A (60°), B (45°) and C (30°)
    d. A (120°), B (90°) and C (60°)
    e. None of them

A B C

18. In the planary symmetrical shape below, which of the following shapes should be placed in x and y spots?



a. 

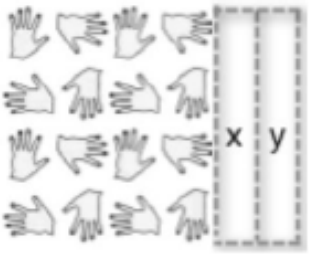
b. 

c. 

d.

e. None of them

19. In the planary symmetrical shape below, which of the following shapes should be placed in x and y spots?



a. 

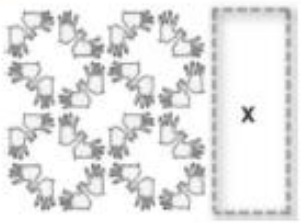
b. 

c. 

d. 

e. None of them

20. In the planary symmetrical shape below, which of the following shapes should be placed in the x spot?



a. 

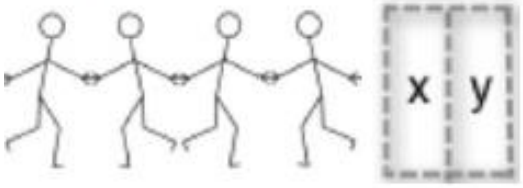
b. 

c. 

d. 

e. None of them

21. In the linear symmetrical shape below, which of the following shapes should be placed in x and y spots?



a. 

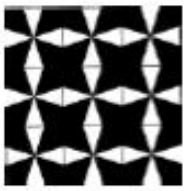
b. 

c. 

d. 

e. None of them

22. Which of the following forms the base of K shape?



K

a. 

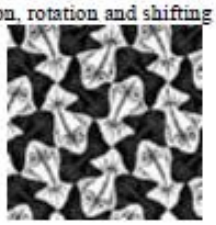
b. 

c. 

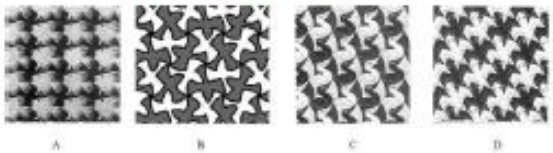
d. 

e. None of them

23. Which symmetry methods have been used in the following M. C. Escher's work of art named "Fishes"?
    a. Only rotation
    b. Only reflection
    c. Rotation and reflection
    d. Only shifting
    e. Reflection, rotation and shifting



24. Different symmetry methods have been used in M.C. Escher's following works.
In which one/ones of the following, rotation symmetry method is used?



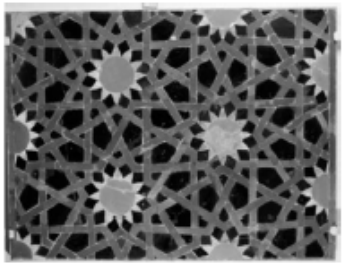
A B C D

    a. Only A
    b. Only B
    c. C and D
    d. A, C and D
    e. None of them

25. What is meant to be emphasized by the geometrical designes in works of art from different cultures?
    a. Eternity
    b. Hunger
    c. Environmental pollution
    d. Wealth
    e. None of them

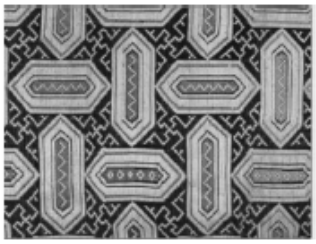
26. In which culture do we meet geometric designes based on the symmetry the most?
    a. European art
    b. Mesopotomian art
    c. Byzantium art
    d. Islamic art
    e. None of them

27. The reason why the following Anatolian Seljuk Empire work is considered as a work of art could be that.....



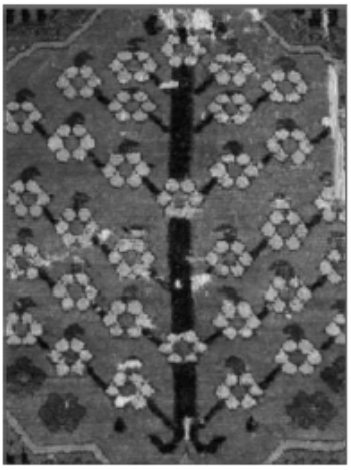
    a. It consists of geometric shapes.
    b. It covers the surface with no space left.
    c. It is original.
    d. It is functional.
    e. All of the above

28. Beside the fact that the following textile product from Ottoman Empire period is a work of art, ... symmetry method(s) were used in its design.



    a. only shifting
    b. only rotation
    c. shifting and rotation
    d. shifting, reflection and rotation
    e. None of the above

29. I think the reason for why the following Anatolian Seljuk Empire carpet is considered as an artwork is NOT that ....



    a. it has flowers.
    b. it symbolizes the tree of life.
    c. it uses reflection symmetry method in its design.
    d. it is functional.
    e. it is original.

30. Through symmetry, we feel that there is a distinct logical structure that exists independently in the universe and that we can perceive with our minds.
    With this proposition, which of the following branches of science has been tried to be associated with?
    a. Physics
    b. Chemistry
    c. Art
    d. Geometry
    e. All of them

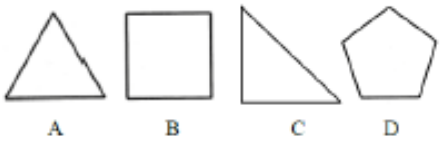
APPENDIX B

CREATIVE PROBLEM-SOLVING SKILL TEST (TURKISH)

Yaratıcı Problem Çözme Testi

Bu test, öğrencilerin yaratıcı problem çözme becerisi ölçmektedir ve 30 sorudan oluşmaktadır. Doğru olduğunu düşündüğünüz seçenek üzerinde düşünün. Her soru için tek bir doğru cevap vardır. Doğru olduğunu düşündüğünüz seçeneği soru üzerinde işaretleyin.

1. Aşağıdakilerden hangisi ya da hangileri düzgün çokgen değildir?



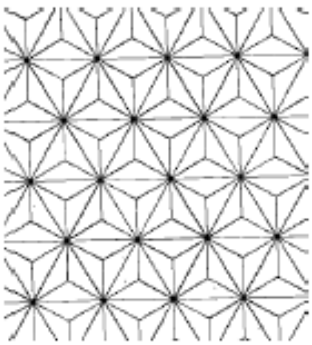
A B C D

a. C ve D
b. Yalnız C
c. A ve B
d. Yalnız D
e. Hepsi düzgün çokgendir

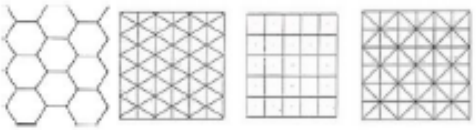
2. Aşağıdaki çokgenlerden hangisi ya da hangileri simetriktir?
   a. Yalnız A
   b. A ve D
   c. Yalnız C
   d. C ve D
   e. Hepsi simetriktir

A B C D

3. Aşağıdaki şekilde görülen çokgenler, seçeneklerden hangisindedir?
   a. Üçgen ve kare
   b. Yalnız altıgen
   c. Yalnız üçgen
   d. Üçgen, altıgen, paralelkenar
   e. Üçgen ve paralelkenar



4. Aşağıdakilerden hangisi ya da hangileri altıgenden oluşmuştur?
   a. B ve D
   b. A ve C
   c. A ve B
   d. Yanlız A
   e. Hepsi



A B C D

5. Simetri: Nesnelerin birbirine oranının aynı olmasıdır.
   Yukarıdaki önermeye göre, aşağıdakilerden hangisi ya da hangileri simetri ile eş anlama gelmektedir?
   Pi, eksen, özdeş, dik açı
   a. Yalnızca Pi
   b. Yalnızca özdeş
   c. Eksen ve özdeş
   d. Eksen ve dik açı
   e. Hiçbiri

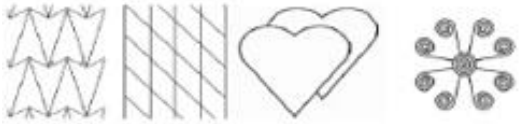
6. Aşağıdakilerden hangisi düzgün dörtgendir?
   a. A ve B
   b. C ve D
   c. Yalnız B
   d. Yalnız C
   e. A ve C

A B C D

7. Aşağıdakilerin hangisi ya da hangilerinde simetrik yansıma özelliği vardır?
   a. A ve D
   b. Yalnız A
   c. Yalnız B
   d. C ve D
   e. B ve D

A B C D

8. Aşağıdakilerden hangisi yada hangileri simetrik öteleme özelliği görülmemektedir?
   a. Yalnız C
   b. A ve C
   c. B ve C
   d. Yalnız D
   e. Yukarıdaki seçeneklerin hepsi doğrudur



A B C D

9. Aşağıdakilerin hangisinde simetrik döndürme özelliği vardır?
   a. A ve B
   b. A ve D
   c. A ve C
   d. Yalnız D
   e. Hepsi

A B C D

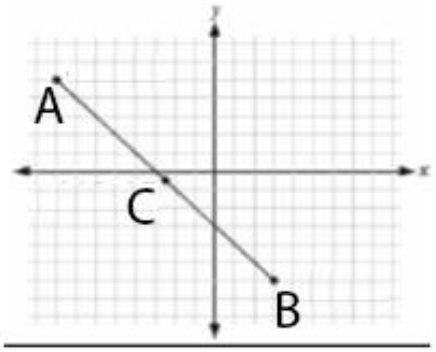
10. Aşağıdakilerden hangisi doğadaki simetriye bir örnektir?
    a. Bal peteği
    b. Papatya
    c. Kozalak
    d. Mermer
    e. Hepsi
11. Aşağıdaki paralelkenarda kesişim noktasındaki bir açının ölçüsü kaçtır?
    a. 180°
    b. 270°
    c. 60°
    d. 120°
    e. 360°

12. Aşağıdaki şekilde simetri yöntemlerinden hangisi ya da hangileri uygulanabilir?
    a. Yalnızca döndürme
    b. Döndürme ve yansıtma
    c. Yansıtma ve öteleme
    d. Yalnızca yansıtma
    e. Yukarıdaki seçeneklerin hiçbiri doğru değildir

13. Aşağıdakilerin hangilerinde aynı sayıda döndürme simetri yöntemi kullanılmıştır?
    a. B ve C
    b. A ve C
    c. A ve B
    d. B ve D
    e. Hiçbiri

A B C D

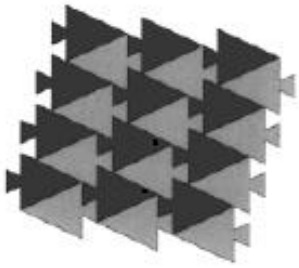
14. Doğru parçası üzerindeki A, B, C noktalarının koordinatları aşağıdakilerden hangisidir?
    a. A (8, 5), B (3, 6), C(2.5, 0.5)
    b. A (5, -8), B (-6, 3), C(-0.5, -2.5)
    c. A (5, 8), B (6, 3), C(0.5, 2,5)
    d. A (-8, 5), B (3, -6), C(-2.5, -0,5)
    e. Hiçbiri



15. Aşağıdaki şekillerden hangisi seçeneklerdeki ifadeyi desteklemektedir?
    a. A'da yansıma simetrisi kullanılmıştır.
    b. B'de öteleme simetrisi kullanılmıştır
    c. C'de öteleme ve yansıtma simetrisi kullanılmıştır
    d. A ve B'de yansıma simetrisi kullanılmıştır
    e. Hiçbiri

A B C

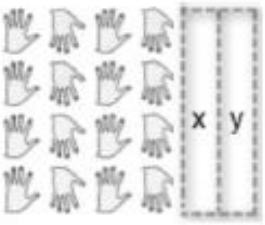
16. Aşağıdaki şekilde seçeneklerdeki hangi özellik görülmez?
    a. Ritm
    b. Tekrar
    c. Yön
    d. Denge
    e. Döndürme



17. Aşağıdaki A, B, C şeklinde her bir açının ölçüsü kaçtır?
    a. A (360°), B (180°) ve C (270°)
    b. A (240°), B (180°) ve C (120°)
    c. A (60°), B (45°) ve C (30°)
    d. A (120°), B (90°) ve C (60°)
    e. Hiçbiri

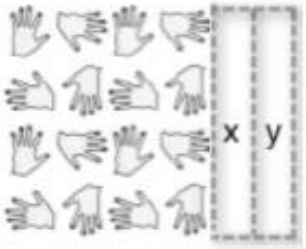
A B C

18. Aşağıdaki düzlem simetrisinde x ve y yerine gelmesi gereken şekil seçeneklerden hangisidir?



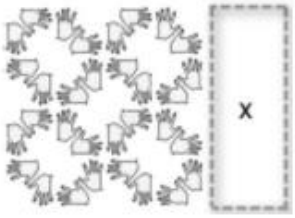
a. 
b. 
c. 
d. 
e. Hiçbiri

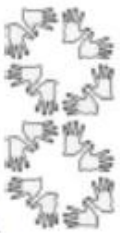
19. Aşağıdaki düzlem simetrisinde x ve y yerine gelmesi gereken şekil seçeneklerden hangisidir?



a. 
b. 
c. 
d. 
e. Hiçbiri

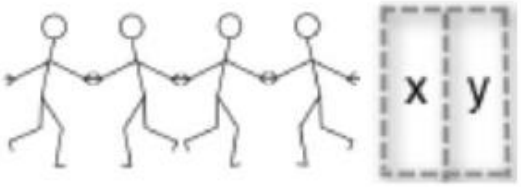
20. Aşağıdaki düzlem simetrisinde x yerine gelmesi gereken şekil seçeneklerden hangisidir?

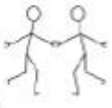


a. 
b. 
c. 
d. 

e. Hiçbiri

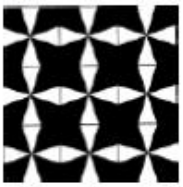
21. Aşağıdaki doğrusal simetrik şekilde x ve y yerine gelmesi gereken şekil seçeneklerden hangisidir?



a. 
b. 
c. 
d. 

e. Hiçbiri

22. Aşağıda verilen seçeneklerin hangisi K şeklinin temel birimidir?



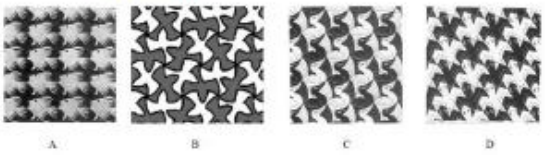
K

a. 
b. 
c. 
d. 

e. Hiçbiri

23. M. C. Escher 'in "Fishes" adlı eserinde, aşağıdaki simetri yöntemlerinden hangisi ya da hangileri kullanmıştır?
    a. Yalnız döndürme
    b. Yalnız yansıma
    c. Döndürme ve yansıma
    d. Yalnız öteleme
    e. Yansıma, döndürme ve öteleme

24. M.C. Escher' e ait aşağıdaki eserlerinde, farklı simetri yöntemleri teknikleri kullanmıştır. Hangisi ya da hangilerinde döndürme yöntemi kullanılmıştır?

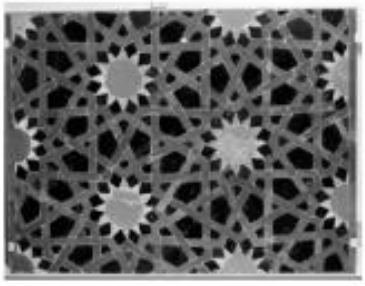


    a. Yalnız A
    b. Yalnız B
    c. C ve D
    d. A, C ve D
    e. Hiçbiri

25. Farklı kültürlere ait eserlerde görülen geometrik tasarımlar ile vurgulanmak istenen nedir?
    a. Sonsuzluk
    b. Açlık
    c. Çevre kirliliği
    d. Zenginlik
    e. Hiçbiri

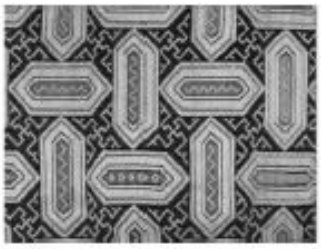
26. Simetri yöntemlerine dayalı geometrik tasarıma en fazla hangi kültürde karşılaşmaktayız?
    a. Avrupa sanatı
    b. Mezopotamya sanatı
    c. Bizans sanatı
    d. İslam sanatı
    e. Hiçbiri

27. Aşağıdaki Anadolu Selçuklu eserinin bir sanat yapıtı olarak kabul edilmesinin sebebi sanırım .....



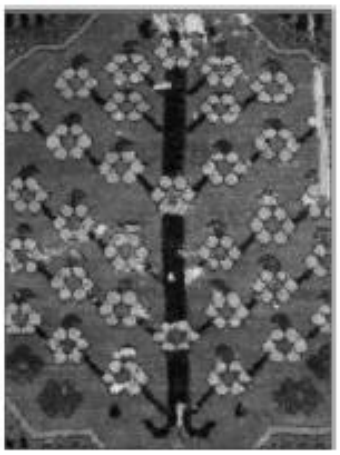
    a. Geometrik şekillerden meydana gelmesi
    b. Hiç boşluk bırakmadan yüzeyi kaplaması
    c. Özgün olması
    d. Işlevsel olması
    e. Hepsi

28. Aşağıdaki Osmanlılar dönemine ait tekstil eserinin önemli özelliği bir sanat eseri olmasının yanında, ... simetri yöntemi ya da yöntemlerinin kullanılmış olmasıdır.



    a. Yalnızca öteleme
    b. Yalnızca döndürme
    c. Öteleme ve döndürme
    d. Öteleme, yansıtma ve döndürme
    e. Hiçbiri

29. Aşağıdaki Anadolu Selçuklu halısının bir sanat yapıtı olarak kabul edilmesinin sebebi sanırım ..... değildir.



    a. Çiçekli olması
    b. Hayat ağacını sembolize etmesi
    c. Yansıma simetri yönteminin kullanılmış olması
    d. İşlevsel olması
    e. Özgün olması

30. Simetri aracılığı ile bizim dışımızda var olan, yine de kendi aklımızla kavrayabileceğimiz belirgin bir mantıksal yapının evrende işlemekte olduğunu hissederiz.
Önermesi ile aşağıdaki bilim dallarının hangisi ile ilişki kurulmaya çalışılmıştır?
    a. Fizik
    b. Kimya
    c. Sanat
    d. Geometri
    e. Hepsi

APPENDIX C

ATTITUDE SCALE FOR

SERIOUS GAME ASSISTED PROGRAMMING LEARNING

Attitudes Scale for Serious Game Assisted Programming Learning

Dear students, this survey aims to assess your attitudes for serious game assisted programming learning. Items in the survey focus on your attitudes towards learning programming, your interest in the use of video games in learning of programming and concerns about antisocial effects of computers. Please consider the following points while answering the items.

1. The games played on mobile phones, tablet computers, etc. should also be considered as video games.
2. Coding: A sequence of commands for making computer and computer like devices perform operations.
3. Name surname: Gender: Age:

	Items	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1	Playing games on my computer (tablet computer, phone, etc.) contributes to my success at school.					
2	It is useful to study by playing video games.					
3	I would like to learn by playing video games.					
4	I would like to adjust the physical appearance of characters in video games myself.					
5	I would like to design my own games on computer.					
6	I would like to learn computer programming.					
7	I think people who work in the field of computer science are not socially active.					
8	I find games with multiple levels intereseting.					
9	The multiplicity of the levels of a game increases my interest in playing it.					
10	I would like to decide what might happen to the character when I design a game.					
11	I would like my friends to play with the games that I developed.					
12	I am not interested in developing my own game on a computer.					
13	It would be fun to play games that are developed by my friends.					
14	It is hard for me to write codes in computer.					
15	I do not want to learn computer programming.					
16	I think that games played on streets are dangerous.					
17	I think that learning computer programming would be useful for me.					
18	I do not think that games can be used for teaching / learning.					
19	Because learning computer programming would increase my problem-solving skills, it would increase my success at exams.					
20	Designing my own games would improve my creativity.					
21	It is ideal to spend your free time by playing games on computer.					
22	The games played on computer are better than the games played on street (like football, basketball, etc.).					
23	I would rather play games on computer than play with my friends.					
24	I love video games with two-players more than the single player ones.					
25	Learning computer programming improves one's intelligence.					
26	Playing games on computer makes individuals lazy.					
27	Lessons with video games would increase my interests in a lesson.					
28	My academic sucess will increase if I learn how to program a video game.					

APPENDIX D

ATTITUDE SCALE FOR

SERIOUS GAME ASSISTED PROGRAMMING LEARNING (TURKISH)

Eğitsel Bilgisayar Oyunları Destekli Kodlama Öğrenimine Yönelik Tutum Ölçeği

Sevgili öğrenciler bu test eğitsel bilgisayar oyunları destekli kodlama öğrenimine karşı tutumlarınızı tespit etmeyi amaçlamaktadır. Ölçekteki maddeler kodlama öğrenimine karşı istek, bilgisayar oyunlarının derslerde öğrenme amaçlı kullanımına yönelik ilgi ve bilgisayarın asosyalleştirmesine yönelik endişeleri kapsayacak şekilde hazırlanmıştır. Ölçeğe cevap verirken aşağıdaki hususlara dikkat ediniz.

1. Ölçekte geçen bilgisayar oyunu sadece bilgisayarda oynanan oyunlar değil tablet, telefon vs. gibi ortamlarda oynanan oyunlar olarak düşünülmelidir.
2. Kodlama: Bilgisayar ve benzeri düzeneklere bir işlem yaptırmak için verilen komut dizisidir.
3. İsim soyisim: Cinsiyet: Yaş:

	Maddeler	Kesinlikle Katılıyorum	Katılıyorum	Kararsızım	Katılmıyorum	Kesinlikle Katılmıyorum
1	Bilgisayarla (tablet, telefon vb) oyun oynamak okul başarıma katkı sağlar.					
2	Bilgisayar oyunlarından yararlanılarak ders çalışmak faydalıdır.					
3	Dersleri bilgisayarla oyun oynayarak işlemeyi isterim.					
4	Bilgisayar oyunlarındaki kahramanların dış görünüşünü kendim ayarlamak isterim.					
5	Bilgisayarda kendi oyunumu tasarlamak isterim.					
6	Bilgisayarda kodlama yapmayı öğrenmek isterim.					
7	Bilgisayar ile ilgili meslek sahibi olanlar **aktif değildir.**					
8	Oyunların seviyeli olması ilgimi çeker.					
9	Oyunlardaki seviyelerin çokluğu oyuna olan ilgimi artırır.					
10	Oyun hazırlarken kahramanın başına neler gelebileceğini kendim belirlemek isterim.					
11	Kendi hazırladığım bilgisayar oyununu arkadaşlarımın da oynamasını isterim.					
12	Bilgisayarda kendi oyunumu hazırlamak ilgimi **çekmez.**					
13	Arkadaşlarımın tasarladığı oyunları oynamak eğlenceli olabilir.					
14	Bilgisayarda kod yazmak benim için **zordur.**					
15	Bilgisayarda kod yazmayı öğrenmek **istemem.**					
16	Sokakta oynanan oyunlar benim için tehlikelidir.					
17	Kodlama öğreniminin benim için faydalı olacağını düşünüyorum.					
18	Bilgisayar oyunları ile eğitim/öğretim **olmaz.**					
19	Kodlama öğrenimi problem çözme becerimi geliştireceği için sınavlarda başarım artar.					
20	Kendi oyunumu tasarlamak yaratıcılığımı geliştirecektir.					
21	Bilgisayarla oyun oynamak boş zamanları değerlendirmek için idealdir.					
22	Dışarıda (futbol, basketbol, evcilik vb.) oynanan oyunlardansa bilgisayarla oynanan oyunlar daha iyidir.					
23	Arkadaşlarımla oyun oynamaktansa bilgisayar oyunu oynamayı tercih ederim.					
24	Bilgisayar oyunlarında ikili oyunları daha çok severim.					
25	Kodlama öğrenimi zeka geliştirir.					
26	Bilgisayarda oyun oynamak bireyleri **tembelleştirir.**					
27	Derslerin bilgisayar oyunları ile işlenmesi derse olan ilgimi artırır.					
28	Bilgisayarda oyun kodlamayı öğrenirsem derslerim de başarım artar.					

APPENDIX E

PRE/POST TEST

Pre-Test

Hello, welcome to our pre-game test. This test is designed to assess your current knowledge on object oriented programming concepts and computational thinking. Please be assured that your personal information is **confidential** and no attempts will be done to identify you.

1. Which is NOT one of the most important characteristics of object-oriented programming?
   a. Encapsulation
   b. Information hiding
   c. Inheritance
   d. Static binding
2. What may be concluded when an instance of a class is created?
   a. A new class has been defined
   b. An object has been instantiated
   c. A set of superclasses is now available for use
   d. Protocols become unique
3. A primary distinction between a class and an object is that:
   a. A class is horizontal metaphor and an object is a hierarchical construct
   b. An object is a general category and a class is a specific instance
   c. A class is a general category and an object is a specific instance
   d. An object is a singular member of a class only
4. A basic function of a class is to do what?
   a. Maintain encapsulated structure
   b. Allow use of subroutines
   c. Set up data types for compression
   d. Define a particular type of object
5. An object can be simply defined as:
   a. Parameter creation through structural analogy
   b. An example of class compaction
   c. Any variable attribute
   d. An instance of a particular class
6. The act of separating external aspects of an object from its internal implementation details is known as:
   a. Application-domain abstraction
   b. Encapsulation
   c. Functional model methodology
   d. Enumerated data flow processes
7. "Polymorphism" can be defined as:
   a. The same operation applied to different classes using different forms
   b. The same operation applied to objects asynchronously
   c. The same operation applied to different homogenous variable groups
   d. The same sequence of operations applied to one class repeatedly
8. The role of class variables in object-oriented programming are the:
   a. Objects to be manipulated
   b. Classes to be manipulated
   c. Functional parameters to be developed
   d. Data typecasting directives
9. A class which inherits another class is thereafter known as a:
   a. Base class
   b. Derived class
   c. Substitution class
   d. Functional parameter class
10. Which of the following statement is **NOT** true?
    a. Objects are instantiated from objects
    b. Class is a prototype for objects
    c. Class is a prototype for classes
    d. Objects are created from classes

11. What is the benefit of encapsulation?
    a. When information in a class is modified, it will not affect the other classes.
    b. When information in a class is modified, it will automatically change information in the related classes
    c. When information in an object is modified, it will not affect other classes
    d. When information in an object is modified, it will automatically change information in related classes
12. Can wished to send a flower to his grandmother, Zeynep, on her birthday. Since his grandmother lives in Ankara, he has to ask Elif, a salesperson who works at the flower shop to send the flower to his grandmother in Ankara. Which of the following relationships is correct?
    a. Class = Flower shop, Instance = Can
    b. Class = Salesperson, Instance = Elif
    c. Class = Salesperson, Instance = Can
    d. Class = Flower shop, Instance = Elif
13. How can objects communicate with each other?
    a. Broadcast a signal throughout the interface network
    b. Use pipeline to synchronize an object's clock
    c. Call another object's method
    d. Place a message in a software bus
14. What is "method overriding"?
    a. To define a method having the same name and functionality as one of its parent class
    b. To define a method having the same functionality but a different name from one of its parent class
    c. To define a method having the same name but a different functionality from one of its parent class
    d. To define a method having a different name and functionality from one of its parent class.
15. What is the difference between "class variable" and "instance variable"?
    a. Class variable refers to a data item associated with a particular class while instance variable refers to a data item associated with a particular object
    b. Class variable refers to a data item associated with a parent class while instance variable refers to a data item associated with a child class
    c. Class varaible refers to a data item used within a particular class while instance variable refers to a data item used within a particular object
    d. None of the above are true, class variable and instance variable have no difference
16. Which of the following pairs of words are synonyms?
    a. Behavior and method
    b. Information hiding and instantiation
    c. Polymorphism and dynamic binding
    d. Encapsulation and interface
17. Consider the following program:

    If (time is after 6 pm)

    Then: Work on science project

    Else: If (time is after 3 pm)

    Then: Play with friends

    Else: ____

    Güliz is in London and it is 4 pm, while Gözde is in Muğla and it is 7 pm. What are Güliz and Gözde doing based on the given code above?

    a. Güliz: work on science project; Gözde: play with friends.
    b. Güliz: work on science project and play with friends; Gözde: play with friends
    c. Güliz: work on science project; Gözde: work on science project and play with friends
    d. Güliz: play with friends; Gözde: work on science project.

18. You are training a robot to avoid obstacles as it moves. To make things more interesting you tell the robot to turn right to go around the obstacle if the color of the obstacle is red. If the obstacle is of any color other than red, the robot should turn left to go around the obstacle. How will you program your robot to follow these instructions using a **When-Do-Otherwise do** structure?

When:

_____

_____

Do:

_____

_____

_____

Otherwise do:

_____

_____

_____

19. Imagine you have established a colony on the moon, and have robots helping you with your tasks. Today you need to program your robot to go test an instrument that is 5 miles away. Your robot can only travel 1 mile on a fully charged battery. Fortunately, you have a charging station every mile long the way. Using the loop structure (the **Repeat** command), write a program to command your robot to successfully reach the instrument so it can conduct the test. Assume your robot is fully charged when it starts its mission.

APPENDIX F

ETHICAL APPROVAL

T.C.
BOĞAZİÇİ ÜNİVERSİTESİ
İnsan Araştırmaları Kurumsal Değerlendirme Alt Kurulu

Sayı: 2018 - 02 -

1 Şubat 2018

Ali Akkaya

Bilgisayar ve Öğretim Teknolojileri Eğitimi

Sayın Araştırmacı,

"Eğitsel oyunların öğrencilerin nesne tabanlı programlamanın kavramsal bilgisi ve bilgi işlemsel düşünme becerilerine etkisi" başlıklı projeniz ile ilgili olarak yaptığınız SBB-EAK 2017/79 sayılı başvuru İNAREK/SBB Etik Alt Kurulu tarafından 1 Şubat 2018 tarihli toplantıda incelenmiş ve uygun bulunmuştur.

Doç. Dr. Ebru Kaya

Doç. Dr. Gül Sosay

Doç. Dr. Mehmet Yiğit Gürdal

Yrd. Doç. Dr. Bengü Börkan

Dr. Nur Yeniçeri

APPENDIX G

PARTICIPANT INFORMATION AND CONSENT FORM

KATILIMCI BİLGİ ve ONAM FORMU

Araştırmayı destekleyen kurum: Boğaziçi Üniversitesi
Araştırmanın adı: Eğitsel oyunların öğrencilerin nesne tabanlı programlamanın kavramsal bilgisi ve bilgi işlemsel düşünme becerilerine etkisi
Proje Yürütücüsü: Prof. Dr. Yavuz Akpınar (Tez Danışmanı)
E-mail adresi: akpinar@boun.edu.tr
Telefonu: 0 212 359 67 88
Araştırmacının adı: Ali Akkaya
E-mail adresi: ali.akkaya1@boun.edu.tr
Telefonu: 0 212 359 67 89

Proje konusu: Nesne tabanlı programlama eğitimden sanayiye birçok alanda kullanılan en popüler programlama yaklaşımıdır. Bu çalışma "eğitsel oyunların öğrencilerin nesne tabanlı programlamanın kavramsal bilgisi ve bilgi işlemsel düşünme becerileri üzerine etkisini incelemeyi" amaçlamaktadır. Çalışma sonucunda programlamaya yeni başlayan öğrencilere nesne tabanlı programlamanın temel kavramlarının ve kullanım alanlarının öğretilmesi ve öğrencilerin bilgi işlemsel düşünme becerilerinin geliştirilmesi hedeflenmektedir. Deneysel çalışmalar Boğaziçi Üniversitesi etik kurulu onayı ile Bilgisayar ve Öğretim Teknolojileri Eğitimi bölümünde yapılacaktır.

Onam: Eğitsel oyunların öğrencilerin nesne tabanlı programlamanın kavramsal bilgisi ve bilgi işlemsel düşünme becerilerine etkisi üzerine yapmak istediğimiz bilimsel araştırmaya katılmaya sizi davet ediyoruz. Bu araştırma programlama alanında kendini geliştirmek isteyen acemi programcılara temel programlama beceri ve bilgilerini eğlenebilecekleri bir ortam sunarak kazandırmayı hedeflemektedir. Araştırma ayrıca programlama öğreten eğitsel oyunlar geliştiren öğretim tasarımcılarına, eğitim teknolojileri uzmanlarına ve araştırmacılara programlama öğretiminde uygulanabilir bir eğitsel oyun yapısı ve bu yapının özelliklerini sunmayı hedeflemektedir.

Araştırmaya katılmayı kabul ettiğiniz takdirde çalışma öncesinde size problem çözme testi ve eğitsel bilgisayar oyunları destekli kodlama öğrenimine yönelik tutum ölçeği verilecektir. Tutum ölçeği ve testi yanıtladıktan sonra sizlere nesne tabanlı programlama bilgilerinizi ve bilgi işlemsel düşünme becerilerinizi ölçmek için bir ön test verilecektir. Ön test aşamasından sonra sizlerden araştırma kapsamında geliştirilen Meraklı Robotlar: Operasyon Asgard isimli eğitsel oyunu bir buçuk saat içerisinde oynayıp bitirmeniz istenecektir. Oyundaki her bir etkinlikte geçirdiğiniz süre ve deneme sayınız oyun tarafından kayıt altına alınıp son etkinliğin bitmesiyle birlikte araştırmacıya e-posta aracılığıyla iletilecektir. Bu nedenle oyuna başlarken sizden isim ve soyisminizi girmeniz istenecektir. Çalışmanın son aşamasında ise sizlere nesne tabanlı programlama bilgilerinizi ve bilgi işlemsel düşünme becerilerinizi ölçmek için çalışma öncesinde verilen ön teste paralel olan bir son test verilecektir. Ayrıca, ekteki formda istenen bilgileri de sağlamanızı rica ediyoruz. İsminiz ve bu bilgiler tamamen gizli tutulacaktır.

Çalışmaya katılmanız tamamen isteğe bağlıdır. Sizden ücret talep etmiyoruz ve size herhangi bir ödeme yapmayacağız.

Testlere vermiş olduğunuz yanıtlar ileride başka çalışmalar için de kullanılabilir. İstediğiniz zaman çalışmaya katılmaktan vazgeçebilirsiniz. Bu durumda yanıtlamış olduğunuz testler ve oyundaki performans kayıtlarınız imha edilecektir.

Yapmak istediğimiz araştırmanın sizlere herhangi bir risk getirmesi beklenmemektedir. Söz konusu araştırmanın tez çalışmasına olduğu kadar sizlere ve programlama dersleri veren öğretmen ve akademisyenlere de katkısının olacağı düşünülmektedir.

Bu formu imzalamadan önce, çalışmayla ilgili sorularınız varsa lütfen sorun. Daha sonra sorunuz olursa, proje yürütücüsüne (Ofis Telefonu: 0 212 359 67 88) sorabilirsiniz. Araştırmayla ilgili haklarınız konusunda yerel etik kurullarına da danışabilirsiniz.

Ben, (katılımcının adı) ....., yukarıdaki metni okudum ve katılmam istenen çalışmanın kapsamını ve amacını, gönüllü olarak üzerime düşen sorumlulukları tamamen anladım. Çalışma hakkında soru sorma imkanı buldum. Bu çalışmayı istediğim zaman ve herhangi bir neden belirtmek zorunda kalmadan bırakabileceğimi ve bıraktığım takdirde herhangi bir olumsuzluk ile karşılaşmayacağımı anladım.

Bu koşullarda söz konusu araştırmaya kendi isteğimle, hiçbir baskı ve zorlama olmaksızın katılmayı kabul ediyorum.

Formun bir örneğini aldım / almak istemiyorum (bu durumda araştırmacı bu kopyayı saklar).

Katılımcının Adı-Soyadı:……..

İmzası:……

Tarih (gün/ay/yıl):...../...../.....

Varsa Katılımcının Vasisinin Adı-Soyadı:.....

İmzası:.....

Tarih (gün/ay/yıl):...../...../.....

18 YAŞ ALTI KATILIMCI VARSA:

Varsa Katılımcının VELİSİNİN Adı-Soyadı:.....

İmzası:.....

Tarih (gün/ay/yıl):...../...../.....

REFERENCES

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832-835.

Ali, A., & Shubra, C. (2010). Efforts to reverse the trend of enrollment decline in computer science programs. *The Journal of Issues in Informing Science and Information Technology, 7*, 209-225.

Barnes, T., Chaffin, A., Godwin, A., Powell, E., & Richter, H. (2007). The role of feedback in Game2Learn. In M. B. Rosson, & D. Gilmore (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1-5). New York: ACM.

Barnes, T., Richter, H., Chaffin, A., Godwin, A., Powell, E., Ralph, T., . . . Jordan, H. (2007, March). Game2Learn: A study of games as tools for learning introductory programming concepts. Paper presented at SIGCSE '07: The 38th ACM Technical Symposium on Computer Science Education, Covington, KY.

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology, 38*(6), 20-23.

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.

Basu, S. (2016). *Fostering synergistic learning of computational thinking and middle school science in computer-based intelligent learning environments.* (Doctoral dissertation). Retrieved from ProQuest Dissertations & Theses Global. (10151674)

Basu, S., Dickes, A., Kinnebrew, J. S., Sengupta, P., & Biswas, G. (2013). CTSiM: A computational thinking environment for learning science through simulation and modeling. In M. Helfert, O. Foley, M. T. Restivo, & J. Uhomoibhi (Eds.), *Proceedings of the 5th International Conference on Computer Supported Education* (pp. 369-378). Aachen, Germany: SciTePress.

Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM, 37*(7), 122-125. doi:10.1145/176789.176803

Begosso, L. C., Begosso, L. R., Gonçalves, E. M., & Gonçalves, J. R. (2012). An approach for teaching algorithms and computer programming using Greenfoot and Python. In R. Leblanc, & A. Sobel (Eds.), *Proceedings of the 2012 IEEE Frontiers in Education Conference (FIE)* (pp. 1-6). Seattle: IEEE.

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.

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

Carlisle, M. C. (2009). RAPTOR: A visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges, 24*(4), 275-281.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: A visual programming environment for teaching algorithmic problem solving. *ACM SIGCSE Bulletin, 37*(1), 176-180.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges, 15*(5), 107-116.

Creswell, J. W. (2011). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (4th ed.). Boston, MA: Pearson.

Csikszentmihalyi, M. (1975). *Beyond boredom and anxiety.* San Francisco: Jossey-Bass Publishers.

Csikszentmihalyi, M. (2014). Toward a psychology of optimal experience. In M. Csikszentmihalyi, *Flow and the foundations of positive psychology: The collected works of Mihaly Csikszentmihalyi* (pp. 209-226). Dordrecht, Netherlands: Springer.

Denning, P. J. (2009). The profession of IT beyond computational thinking. *Communications of the ACM, 52*(6), 28-30.

Dierbach, C., Hochheiser, H., Collins, S., Jerome, G., Ariza, C., Kelleher, T., . . . Kaza, S. (2011). A model for piloting pathways for computational thinking in a general education curriculum. In T. J. Cortina, E. L. Walker, L. S. King, & D. R. Musicant (Eds.), *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 257-262). New York: ACM.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology, 42*(4), 624-637.

Florea, A., Gellert, A., Florea, D., & Florea, A.-C. (2016). Teaching programming by developing games in Alice. In I. Roceanu, D. Dubois, D. Beligan, F. Moldoveanu, M. I. Dascalu, I. Stanescu, & D. Barbieru (Eds.), *The International Scientific Conference eLearning and Software for Education. 1*, pp. 503-510. Bucharest: "Carol I" National Defence University Publishing House.

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & Gaming, 33*(4), 441-467.

Gerola, R. J. (1997). *Identification of object-oriented computer programmer mastery status through evaluation of object-oriented programming semantic knowledge.* (Doctoral dissertation). Retrieved from ProQuest Dissertations & Theses Global. (304370368)

Glasser, M. (2009). Fundamentals of object-oriented programming. In M. Glasser, *Open Verification Methodology Cookbook* (pp. 27-48). New York: Springer-Verlag.

Gomes, A., & Mendes, A. J. (2007, September). Learning to program-difficulties and solutions. Paper presented at the International Conference on Engineering Education–ICEE, Coimbra, Portugal.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38-43.

Gunter, G. A., Kenny, R. F., & Vick, E. H. (2008). Taking educational games seriously: using the RETAIN model to design endogenous fantasy into standalone educational games. *Educational Technology Research and Development, 56*(5-6), 511-537.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25-27.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *ACM SIGCSE Bulletin, 31*(3), 171-174.

Jonassen, D. H. (2004). *Learning to solve problems: An instructional design guide.* San Francisco: Pfeiffer.

Jones, R. (2000). Design and implementation of computer games: A capstone course for undergraduate computer science education. *ACM SIGCSE Bulletin, 32*(1), 260-264.

Kazimoglu, C. (2013). *Emprical evidence that proves a serious game is an educationally effective tool for learning computer programming constructs at the computational thinking level.* (Doctoral dissertation, University of Greenwich).

Kazımoğlu, Ç., Kiernan, M., & Bacon, L. (2012b). Understanding computational thinking before programming: Developing guidelines for the design. In P. Felicia, *Developments in Current Game-Based Learning Design and Deployment* (p. 316). IGI Global.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012a). A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences, 47*, 1991-1999.

Keçeci, G., Alan, B., & Zengin, F. K. (2016). Eğitsel bilgisayar oyunları destekli kodlama öğrenimine yönelik tutum ölçeği: Geçerlilik ve güvenilirlik çalışması. *Education Sciences, 11*(4), 184-194.

Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education, 8*, 13-24.

Kolb, D. (1984). *Experiential learning: Experience as the source of learning and development.* New Jersey: Prentice Hall.

Kölling, M. (1999a). The problem of teaching object-oriented programming. *Journal of Object Oriented Programming, 11*(8), 8-15.

Kölling, M. (1999b). The problem of teaching object-oriented programming, Part II: Environments. *Journal of Object-Oriented Programming, 11*(9), 6-12.

Kölling, M. (2010). The greenfoot programming environment. *ACM Transactions on Computing Education, 10*(4), 14:1-21.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13*(4), 249-268.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*(3), 14-18.

Laporte, L., & Zaman, B. (2018). A comparative analysis of programming games, looking through the lens of an instructional design model and a game attributes taxonomy. *Entertainment Computing, 25*, 48-61.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32-37.

Lester, J., Converse, S. A., KAhler, S. E., Barlow, S. T., Stone, B. A., & Bhogal, R. (1997). The persona effect: Affective impact of animated pedagogical agents. In S. Pemberton (Ed.), *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (pp. 359-366). New York: ACM.

Lister, R. (2011). Programming, syntax and cognitive load (part 2). *ACM Inroads, 2*(2), 21-22.

Liu, C.-C., Cheng, Y.-B., & Huang, C.-W. (2011). The effect of simulation games on the laerning of computational problem solving. *Computers & Education, 57*(3), 1907-1918.

Livovský, J., & Porubän, J. (2014). Learning object-oriented paradigm by playing computer games: concepts first approach. *Central European Journal of Computer Science, 4*(3), 171-182.

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260-264.

Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. In J. D. Dougherty, S. Rodger, S. Fitzgerald, & M. Guzdial (Eds.), *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (pp. 367-371). Portland: ACM.

Maloney, J., Resnick, M., Rusk, N., & Silverman, B. E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16.

Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game Based Learning Approach for Teaching Programming Concepts. *Journal of Educational Technology and Society*, 5-17.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In H. M. Walker (Eds.), *Working group reports from ITiCSE on Innovation and Technology in Computer Science Education* (pp. 125-140). New York: ACM.

McFarlane, A., Sparrowhawk, A., & Heald, Y. (2002). *Report on the educational use of games.* Cambridge: Teachers Evaluating Educational Media.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In G. Rößling, T. Naps, & C. Spannagel (Eds.), *Proceeedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168-172). New York: ACM.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239-264.

Miljanovic, M. A., & Bradbury, J. S. (2017). RoboBUG: A serious game for learning debugging techniques. In J. Tenenberg, D. Chinn, J. Sheard, & L. Malmi (Eds.), *Proceeding of the 2017 ACM Conference on International Computing Education Research* (pp. 93-100). New York: ACM.

Moreno, R., Mayer, R. E., Spires, H. A., & Lester, J. C. (2001). The case for social agency in computer-based teaching: Do students learn more deeply when they interact with animated pedagogical agents? *Cognition and Instruction, 19*(2), 177-213. doi:10.1207/S1532690XCI1902_02

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. P. (2011). Experimental feedback on Prog&Play: a serious game for programming practice. In E. Gröller, & H. Rushmeier (Eds.), *Computer Graphics Forum. 30*, pp. 61-73. Blackwell Publishing Ltd.

O'Kelly, J., & Gibson, J. P. (2006, June). RoboCode & problem-based learning: A non-prescriptive approach to teaching programming. *ACM SIGCSE Bulletin, 38*(3), 217-221.

Özkök, A. (2005). Disiplinlerarası yaklaşıma dayalı yaratıcı problem çözme öğretim programının yaratıcı problem çözme becerisine etkisi. *Hacettepe Üniversitesi Eğitim Fakültesi Dergisi, 28*, 159-167.

Papert, S. (192). Teaching children thinking. *Programmed Learning and Educational Technology, 9*(5), 245-255.

Pellas, N. (2014). Exploring interrelationships among high school students' engagement factors in introductory programming courses via a 3D multi-user serious game created in open sim. *Journal of Universal Computer Science, 20*(12), 1608-1628.

Perković, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational thinking across the curriculum. In R. Ayfer, J. Impagliazzo, & C. Laxer (Eds.), *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 123-127). New York: ACM.

Phelps, A. M., Egert, C. A., & Bierre, K. J. (2005). MUPPETS: multi-user programming pedagogy for enhancing traditional study: an environment for both upper and lower division students. *Proceedings of the 4th Conference on Information Technology Curriculum* (pp. 100-105). New York: ACM.

Pitsatorn, P. P. (2003). *Object-oriented programming training: Bottom-up versus top-down approach.* (Doctoral dissertation). Retrieved from ProQuest Dissertations & Theses Global. (305334331)

Polya, G. (1957). *How to solve it: A new aspect of mathematical method.* Princeton, New Jersey: Princeton University Press.

Poo, D., Kiong, D., & Ashok, S. (2007). *Object-oriented programming and Java.* London: Springer Science & Business Media.

Prensky, M. (2003). Digital game-based learning. *ACM Computers in Entertainment*, 1-4.

Qualls, J. A., & Sherrel, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Computing Sciences in Colleges, 25*(5), 66-71.

Ramírez-Rosales, S., Vázquez-Reyes, S., Villa-Cisneros, J. L., & De León-Sigg, M. (2016). A Serious Game to Promote Object Oriented Programming and Software Engineering Basic Concepts Learning. In R. Juárez-Ramírez, S. J. Calleros, H. J. Oktaba, C. F. Fernández, R. A. Vera, G. L. Sandoval, . . . J. A. (Eds.), *2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)* (pp. 97-103). Los Alamitos: IEEE.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. In G. Lewandowski, S. Wolfman, T. J. Cortina, & E. L. Walker (Eds.), *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 265-269). New York: ACM.

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.

Rieber, L. P. (1996). Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational Technology Research and Development, 44*(2), 43-58.

Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on game design.* Indianapolis: New Riders.

Rosenthal, R., & Rosnow, R. L. (1984). *Essentials of behavioral research: Methods and data analysis.* New York: McGraw-Hill.

Sarkar, N. I. (2006). Teaching computer networking fundamentals using practical laboratory exercises. *IEEE Transactions on Education, 49*(2), 285-291.

Selby, C. C., & Woollard, J. (2013). Computational thinking: The developing definition. In J. Carter, I. Utting, & A. Clear (Eds.), *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (p. 6). Canterbury: ACM.

Soflano, M. (2011). Modding in serious games: Teaching structured query language (SQL) using neverwinter nights. In M. Ma, A. Oikonomou, & L. Jain (Eds.), *Serious Games & Edutainment Applications* (pp. 347-368). London: Springer.

Soloway, E. (1986). Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850-858.

Sung, K., Hillyard, C., Angotti, R. L., Panitz, M. W., Goldstein, D. S., & Nordlinger, J. (2011). Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses. *IEEE Transactions on Education, 54*(3), 416-427.

Sweller, J., van Merriënboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational Psychology Review, 10*(3), 251-296.

Van Haaster, K., & Hagan, D. (2004). Teaching and learning with BlueJ: An evaluation of a pedagogical tool. *Issues in Informing Science & Information Technology, 1*, 455-470.

van Merriënboer, J. J., Clark, R. E., & de Croock, M. B. (2002). Blueprints for complex learning: The 4C/ID-model. *Educational Technology Research and Development, 50*(2), 39-61.

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.

Wang, T.-C., Mei, W.-H., Lin, S.-L., Chiu, S.-K., & Lin, J. M.-C. (2009). Teaching programming concepts to high school students with Alice. In J. Froyd (Ed.),

Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference (pp. 955-960). Piscataway, NJ: IEEE Press.

Watson, C., Li, F. W., & Lau, R. W. (2011). Learning programming languages through corrective feedback and concept visualisation. In H. Leung, E. Popescu, Y. Cao, R. W. Lau, & W. Nejdl (Eds.), *Proceedings of the 10th International Conference on Web-Based Learning* (pp. 11-20). Heidelberg: Springer-Verlag.

Wegner, P. (1990). Concepts and paradigms of object-oriented programming. *ACM SIGPLAN OOPS Messenger, 1*(1), 7-87.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers, 11*(3), 255-282.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences, 366*, 3717-3725. doi:10.1098/rsta.2008.0118

Wong, Y. S., Hayati, M. Y., & Tan, W. H. (2016). A Propriety Game-Based Learning Game as Learning Tool to Learn Object-Oriented Programming Paradigm. *Joint International Conference on Serious Games* (pp. 42-54). Brisbane: Springer International Publishing.

Xinogalos, S. (2016). Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy. *Education and Information Technologies, 21*(3), 559-588.