Computer Science Education

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/ncse20

Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills

Ali Akkaya & Yavuz Akpinar

To cite this article: Ali Akkaya & Yavuz Akpinar (2022): Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills, Computer Science Education, DOI: 10.1080/08993408.2022.2044673

To link to this article: https://doi.org/10.1080/08993408.2022.2044673

COMPUTER SCIENCE EDUCATION
https://doi.org/10.1080/08993408.2022.2044673

Routledge
Taylor & Francis Group

Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills

Ali Akkaya and Yavuz Akpinar

Computer Education & Educational Technology Department, Boğaziçi University, Etiler, Istanbul, Turkey

ABSTRACT

Background and Context: Though still a nascent area of research, serious games have been presented as means of engaging students in computer programming and computational thinking due to their immersive and interactive nature. Existing research is limited in its ability to provide systems based on sound instructional design models, and only a few studies validate their design with statistical support.

Objective: This study investigated the effects of a game which is based on experiential learning theory under framework of the four-component instructional design model on undergraduate students' learning performance in conceptual knowledge of object-oriented programming and computational thinking skills.

Method: A pre-test and post-test quasi-experimental design was used to study the effects of the experiential serious games on conceptual knowledge of OOP and CT skills of 61 non-engineering students with and without prior programming knowledge.

Findings: The statistical analyses reveal that students with and without programming experience significantly improved their understanding of fundamental concepts of OOP. There were only weak correlations among students' creative problem solving, attitudes towards digital game-based learning of programming, and learning.

Implications: We provide several recommendations for researchers and practitioners for designing and developing an effective serious game to teach novice programmers computer programming.

ARTICLE HISTORY
Received 24 August 2021
Accepted 17 February 2022

KEYWORDS
Serious game; computational thinking; object-oriented programming; instructional design

Introduction

Computer programming is one of most important skills of today's world because modern societies rely on computer systems to drive industry, trade and nearly every aspect of human life. Additionally, computational thinking (CT) skills are the fundamental skills that lie at the bottom of computer programming. CT is a problem-solving approach in which

CONTACT Yavuz Akpinar akpinar@boun.edu.tr Computer Education & Educational Technology Department, Boğaziçi University, Kuzey Kampüs, ETA-B Blok, Oda:508, Etiler, Istanbul, Turkey

Please note that ethical committee approval of the research reported in this manuscript, participants' consent information and research data are available athttps://drive.google.com/drive/folders/1JR1mtlfgIwmjRb-XnJLrNHb-0iJtv2Dr?usp=sharing

solutions to problems are generated in a way that computers are able to perform (Wing, 2006). However, students have problems in understanding and analyzing a problem, building step-by-step algorithmic solution designs for problems (Guenaga et al., 2021; Xinogalos, 2016) and visualizing the programming concepts from a problem situation (Al-Sakkaf et al., 2019; Mladenovic et al., 2021) which result in demotivation. Hence, researchers adopted a digital game-based learning approach to help novice programmers overcome their learning problems and improve their performance on computer programming.

Programming for K-12 can be traced back to 1960s when Logo programming was first introduced to develop powerful intellectual thinking skills; though studies of teaching programming at early ages were not high in number, in the recent years, renewed interest in introducing programming to K-12 students, and students of non-computer science programs in universities is observed. This is encouraged with the availability of relatively easy-to-use visual programming languages. Visual programming environments such as Scratch (Resnick et al., 2009), Alice (Cooper et al., 2000), and Greenfoot (Kölling, 2010) were used by researchers to teach the basics of object-oriented programming (OOP) and CT, and the findings of the research 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 (Chen et al., 2019; Meerbaum-Salant et al., 2013; Mladenovic et al., 2021). Moreover, these programming environments do not have a proper feedback mechanism to help novice programmers understand the errors in their algorithms (Meerbaum-Salant et al., 2011). 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, the effects of serious games on learning computer programming attracted researchers in recent years; the majority of the findings agree on the positive effects of serious games on novice programmers' motivation (Barnes, Richter et al., 2007; Liu et al., 2011; Mathrani et al., 2016; Paiva et al., 2020; Ramírez-Rosales et al., 2016; Wong et al., 2016). Much of the research up to now has studied the effects of serious games on novice programmers' learning performance and motivation. However, only a few studies provide a well-conceived design and demonstrate inferential statistical analysis (Livovsky & Poruban, 2014; Mathrani et al., 2016; Miljanovic & Bradbury, 2017). Hence, the main purpose of this study is to examine the effects of the use of a serious game on undergraduate students' learning performance on conceptual knowledge of OOP, and CT skills.

Literature review

Problems of teaching and learning programming

Simula, developed in early 1960s, was the first OOP; however, teaching OOP become an essential part in computer science education a couple of decades ago (Kölling, 1999a). Commonly an easy programming language was used to introduce computer programming to students at schools and universities, and then move to a more difficult programming language. Therefore, procedural programming concepts and methods which are easier than OOP are used to be taught first, and later OOP are introduced to students.

However, OOP have benefits of software reuse, modularization, programming in teams and maintenance of large systems. Many researchers and teachers (for details, see, Kölling, 1999a, 1999b; Pitsatorn, 2003;) suggest to first teach OOP if the aim is to teach them OOP. Because shifting from procedural programming to OOP takes long time (Kölling, 1999a; Livovsky & Poruban, 2014; Xinogalos, 2016) many schools and universities are introducing computer programming by using OOP. The abstract nature of OOP itself causes problems for novice programmers (Abbasi et al., 2021; Kölling, 1999a; Weintrop & Wilensky, 2019). 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 defining special functions as a main way of forming a solution to a problem to form a solution to a problem by creating functions that utilize classes and objects (Xinogalos, 2016). Starting to learn OOP with a real programming language like C# or Java makes it difficult for students to learn OOP (Guzdial, 2008). 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). In addition, novice programmers have difficulties figuring out what constructs to use and where to use them in their algorithms while they are programming (Guenaga et al., 2021; Kazimoglu et al., 2012). Moreover, Watson et al. (2011) state that unspecific compiler messages cause difficulties for novice programmers because they need proper feedback from the compiler that guides them. Further, many students have problems even in visualizing the execution of their programs (Cooper et al., 2000; Grover & Basu, 2017). To overcome this, providing visual representation of classes and objects to think about problems in object-oriented terms is suggested (Kölling, 1999b; Toth & Lovaszova, 2021).

Computational thinking and object-oriented programming

Students' problem 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 et al., 2011; Weintrop & Wilensky, 2019). The idea of CT being a problem-solving approach is commonly held, but CT is not limited to only a problem-solving method (Selby & Woollard, 2013). CT also involves skills such as abstraction, decomposition, algorithm building, evaluating, and debugging.

Researchers point out that CT and computer programming are not the same (Lu & Fletcher, 2009; Voogt et al., 2015), nonetheless computer programming and CT are intertwined concepts (Kazimoglu, 2013). 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 solutions. With CT skills and knowledge of OOP concepts, novice programmers would be able to build object oriented schemata and generic solutions to problems in the computer science (CS) domain.

Visual programming environments to teach programming

Studies focused on developing different kinds of programming environments for teaching OOP (Carlisle, 2009; Cooper, et al., 2000; Kölling, 2010; Kölling et al., 2003). For example, The BlueJ programming environment allows students to have dynamic interactions with classes to test them (Kölling et al., 2003). As a result of a survey, Van Haaster and Hagan (2004) reported positive effect of the BlueJ environment on novice programmers. An improved version of the BlueJ programming environment, Greenfoot (Kölling, 2010), provide instant visualization of the current behavior and the state of the objects. Begosso et al. (2012) investigated Greenfoot programming environment on students' conceptual knowledge of OOP showed that students had an achievement rate of more than 60%, and that students were motivated while they were learning OOP with Greenfoot.

Furthermore, to overcome problems in learning OOP, Cooper et al. (2000) designed a 3-D interactive programming environment with drag-and-drop code blocks, Alice. Wang et al. (2009) showed that students who were taught programming with Alice performed significantly superior to the students in control group. Another experiment (Florea et al., 2016) used Alice as both a game and a game development tool for students in learning programming. The study showed that students had a positive perception of learning by playing and developing games.

The researchers integrated visual programming environments into CS education to help novice programmers overcome their learning difficulties. For example, a study (Meerbaum-Salant et al., 2013) with Scratch showed that students' learning of CS concepts was improved. Yet the students still had difficulty in understanding concepts such as variables, concurrency and repeated executions: Although Scratch has some positive effects on the learning of CS concepts, it does not properly support OOP.

Current research 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 et al., 2013; Repenning et al., 2010; Weintrop & Wilensky, 2019) even though they have positive effects on teaching programming. Otherwise, these programming environments will only bring a short burst of enthusiasm for novice programmers (Repenning et al., 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 et al., 2011). Another concern is that, 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.

Serious games for learning programming

Serious games are computer games with educational goals, and provide intriguing contexts with interactive, engaging and immersive activities (Gunter et al., 2008). Expectations from serious games encouraged to initiate large scale projects; For example, The European Network of Excellence for Game-based Learning (GaLA – Game and Learning Alliance; see http://www.galanoe.eu) project aimed to stimulate the effective use of games in education and training. The subjects at which serious games developed in the GaLA projects include supply chain management, cultural heritage, healthcare, ethics, probability, CT, communication and co-operation in complex distributed production systems. Project reports suggested linking learning and game mechanics (Arnab et al., 2015), providing an unplugged and plugged activities (Tsarava et al., 2019), providing immediate feedback (Marfisi-Schottman et al., 2019), using metaphors (Eleftheriadis & Xinogalos, 2020). Pilot evaluations of the settings in these studies have provided promising results.

Another large scale endeavor to pinpoint the most pioneering solutions that use gamification for the development and certification of certain 21st century skills including CT was recently launched by the Inter-American Development Bank (IADB, 2020); Results of the project is not unveiled yet. Further, many researchers used games as learning environments for teaching programming to novice learners (e.g. Abbasi et al., 2021; Barnes, Chaffin et al., 2007; Barnes, Richter et al., 2007; Kazimoglu et al., 2012; Mathrani et al., 2016; Muratet et al., 2011; O'Kelly & Gibson, 2006; Paiva et al., 2020). For example, Phelps et al. (2005) used a web-based 3D collaborative virtual environment, MUPPETS, to teach fundamental concepts of OOP using the Java programming language. Another game development environment, RoboCode, aims to teach the basic concepts of structured programming and the fundamental concepts of OOP in which students create robots by writing programs in Java, and players' robots fight each other in a small rectangular online environment. It enables a player to see instantly how the robots are affected by the player's codes. Based on the anecdotal data (Phelps et al., 2005) and observations in students' first year of programming in RoboCode (O'Kelly & Gibson, 2006) the findings without empirical data seem promising. 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, Java.

Moreover, Barnes et al. (2007) conducted a study with a 2D role playing game, Saving Princess Sera, to teach variable declaration along with the simple usage of conditions, structures, and loops. The study showed that students liked the idea of games being used as a reinforcement tool for a programming class, but no achievement increase was reported. ZTECH (Wong et al., 2016) is another role-playing game for teaching basic OOP concepts. Sixty freshman students evaluated the game, two-third of the students found ZTECH an effective tool to teach object-oriented paradigms. Researchers claimed that pseudo codes should be used in introductory programming courses to overcome difficulties that are caused by the syntax of a real programming language.

In a similar vein, Livovsky and Poruban (2014) developed the 2D Alien Breed game to teach the basic concepts of OOP to novice programmers without making students write programs in a real OOP language. The game focused on several concepts of OOP, namely,

object, class, attribute, operation, encapsulation, and inheritance. The researchers stated that students were reluctant to read help texts, instructions for tasks, and to learn content. They recommend replacing these text tutorials with animations or video tutorials because they believe that the length of those texts was the reason.

A 3D virtual world, the Second Life, which enables students to program the behavior of objects by writing basic script codes, was investigated by Esteves et al. (2011). The study pointed to three important issues, first, students had difficulty following the conversations and instructions of the settings. Second, although students were grateful for the opportunity to talk with instructors, it was hard for the instructors to provide immediate feedback. Third, they had problems creating the right algorithm and finding the execution errors in their programs. Another study with a 3D simulation game, TrainB&P (Liu et al., 2011), 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. Similarly, a multiplayer real-time strategy game, Prog&Play, (Muratet et al., 2011) to teach programming, was found motivating by the students and teachers.

Kazimoglu et al. (2012) studied the effects of a serious game, Program Your Robot, on students' CT skills. The game focused on abstract and conceptual knowledge of programming rather than on the functions of developing CT skills. The results revealed that the majority of students thought game was helpful in improving their problem-solving skills and understanding basic programming constructs. In a similar study, Mathrani et al. (2016) used the LightBot, a fictional story in which players program a robot to light all blue tiles in a specific path by using prefabricated commands that represent fundamental programming concepts like functions, conditional flows, recursion. The study indicated that students loved the game, and the researchers found it useful in learning basic programming, but suggested more complicated game for advanced programming.

Ramírez-Rosales et al. (2016) developed a serious game, Software KIDS, to teach fundamentals concepts of OOP and the basic concepts of software engineering, such as algorithms, conditional and iterative structures, and arrangements. Although the children enjoyed the game, they needed the assistance of a mentor to solve problems they encountered in the gameplay. Another serious game, RoboBUG, (Miljanovic & Bradbury, 2017) was designed for computer science students who are learning C++. The standard puzzle-type version of the game implements debugging in C++, but it also allows instructors to create new levels with different programming languages. However, no significant effects were observed, and the researchers argued that the game should include a hint system to relieve the frustration of players.

Recently, Abbasi et al. (2021) conducted an experimental study with a serious game to teach OOP concepts to students in computer science programs. The game incorporated three different stories as playing scenarios, which include management systems of a hospital and a library, and an online-shopping system. The game with various levels focused on concepts of object, methods, attributes, class, and relationship between the classes; the game story required the player to identify the correct candidates for the class in a given domain. The main finding was that the control group who studied the same content in a traditional manner performed worse than the experimental group's performance at the posttest. The study also reported that the game motivated students to learn OOP.

The improvement of students' CT skills, engagement and affected outcomes through educational games have been reported in many recent studies (see a recent meta-analysis conducted by Sun, Guo & Hu, 2021), however the evidence for accomplishment in this area is not yet convincing.

Statement of the problem

The current literature points out that visual programming environments will only bring a short burst of enthusiasm unless they are used with interactive teaching methods and learning materials (Repenning et al., 2010; Sun et al., 2021). These environments lack the mechanism to provide feedback to students about their error or about the appropriate use of programming blocks (Abbasi et al., 2021). Moreover, there is still a need to write algorithms for solutions to problems, which increases the intrinsic cognitive load (Lister, 2011). Many researchers, on the other hand, investigated the effects of games on developing CS skills and learning computer programming. Few studies provided a well-prepared experimental design and demonstrated statistical findings, and few followed a sound gaming approach and 4C instructional design (Abbasi et al., 2021; Livovsky & Poruban, 2014; Mathrani et al., 2016; Miljanovic & Bradbury, 2017). Also, research in computer programming education has stressed that many novice students lack problem solving skills, one of the prerequisites of computing (Veerasamy et al., 2019), and fail in formulating a problem and communicating its solution in code, and unable to utilize key concepts, such as loops and conditionals in programming (Koulouri et al., 2014). In turn, in programming processes, knowing programming codes is not sufficient while solving the problems; Having creative problem solving skills (CPSS) is needed (Akar & Altun, 2017). CPSS is a way of solving problems when usual way of thinking fails, it encourages students to bring new perspectives and find new solutions. Hence, it is also important to examine the relationship between students' CPSS and their performance in OOP.

This study aimed to examine the effects of a serious game (Curious Robots: Operation Asgard), based upon Kiili (2005)'s Experiential Gaming Model and 4C instructional design model (Van Merriënboer et al., 2002), with experiential gaming features on students' conceptual knowledge of OOP and CT skills. The study particularly answered the following questions:

(1) Is there any significant effect of the game on the development of conceptual knowledge of OOP and CT skills of students without programming experience, and of students with procedural programming experience?
(2) Is there any significant effect of the game on the difference between the achievement scores on the conceptual knowledge of OOP and CT skills of students without programming experience and of students with procedural programming experience?
(3) To what extent do students' CPSS and attitudes towards digital game-based learning of programming influence their achievement score on the conceptual knowledge of OOP and CT skills?

Methodology

Research design and sampling

A pre–test and post-test quasi-experimental design is used to study the effects of serious games on undergraduate students' conceptual knowledge of OOP and CT skills. The target population of the study was undergraduate students studying computer programming in non-engineering disciplines. A sample of undergraduate students were selected from the Computer Education and Educational Technology Department in a Turkish state university where the medium of instruction is English, permitted to collect data. Convenience sampling in this quasi-experimental design (Creswell, 2011) was followed as the sampling method because there was no chance to access participants randomly. The first group of the sample consisted of 30 freshman students without prior experience in programming, while the second group included 31 sophomores with experience in procedural programming but not in OOP. 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.

Treatments

In this study, a 2D science-fiction themed hybrid (puzzle-solving and simulation) serious game, Curious Robots: Operation Asgard, was developed by the researchers with Unity 3D game engine using C# programming language (Akkaya, 2019). The game was specifically designed to be a simulation game because it allows players to explore a virtual game world and interact with the other game objects to test their hypotheses (Kiili, 2005). The objective of the developed game was in two-folds; one 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, and the second is to enable students to practice CT skills: conditional logic, algorithm building, simulation and debugging.

Following the development of the first version of the game, the researchers 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.

Serious game design model

Using a game design model that successfully integrates game characteristics and educational theory is important in the development of a serious game. In order to ensure that students would accomplish the objectives of the learning unit in the game, the conceptual design framework of the game developed included the following features:

high level of interactivity to motivate learners,

providing problems in an authentic context,
chance of analyzing a problem situation,
actively testing generated solutions and discover,
chance of observing the outcomes of solutions through feedback, and improve them.

Even though there are many serious game development frameworks in the current literature (see, Krath et al., 2021), Kiili (2005)'s Experiential Gaming Model was selected as the conceptual design framework because it was the one that most closely met criteria of interactive learning environments. 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). By grounding the design of the game to the experiential gaming model, we also aimed to increase the motivation of novice programmers because they often have low motivation to learn OOP (Kong & Wang, 2019; Prensky, 2003)

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 1). At the center of these cycles there are challenges which 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 ones may cause players to be anxious. Therefore, it is important to provide learners

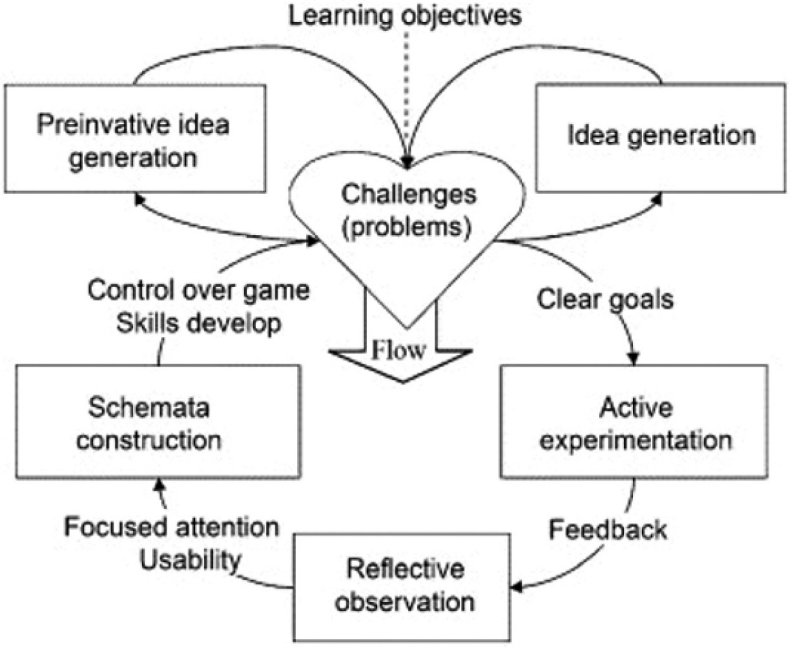

Figure 1. Experiential gaming model (Kiili, 2005).

with challenges that will match their skill and knowledge level. Furthermore, challenges should be designed in a way that the difficulty of the tasks will increase when players make progress in the game.

The preinvative idea generation loop has a disorganized structure which can usually be seen in the way children play. In the idea generation loop, players analyze the problems and generate their solutions according to the rules and constraints of the game world. In the experimentation stage, players implement their solutions 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 them. This experimentation and observation process of solutions would help students to construct new knowledge schemata, consequently resulting in learning. Kiili (2005) emphasized that it is important for learners to test different solutions to a problem to improve their creative problem-solving skills (CPSS) and current knowledge on the topic. 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, the four-component instructional design model (4C/ID model; Van Merriënboer et al., 2002) was used in the design and the development of the activities of the game.

Instructional design model

The 4C/ID model consists of four major components: (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, starting from 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, 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 part-task practice 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.

Overall, both models encourage the use of ill-structured problems in a learning environment to support exploratory 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 metaphorical machines (Eleftheriadis & Xinogalos, 2020) were used to integrate OOP concepts into the story of the game, and to provide visual representations of abstract concepts of OOP. The metaphorical 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 processes of defining a class and object instantiation in the panel of the machine (see, Figure 2). 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.

The game developed in this study differs from others in two folds: One is that it teaches fundamental concepts of OOP along with enabling students to improve their CT skills by providing authentic problem situations. Second is the design of interactive tools of the game. Context of learning environment provided screen operators (ie., metaphorical machines) and tasks: Both of which together supported particularization of meanings of OOP and CT concepts and brought coherence with the whole. Activities on operators supported understanding how all the pieces of an entire concept fit together. In addition to providing meaning to the abstract concepts, and to proceduralization of them, the used context also showed the relevance of the concepts. To make it more appealing and more relevant to computation and automation, the metaphorical machines are used. In the game, once students specify the features of their robots, they start to program its behaviors in a method definer machine via dragging-and-dropping code blocks. Unlike the other games, the method definer machine, along with other machines, enables students to build their own code blocks according to the needs of their missions as a game play experience. Method creation activities in this machine are designed to help novice programmers think their daily motions critically and divide basic motions such as

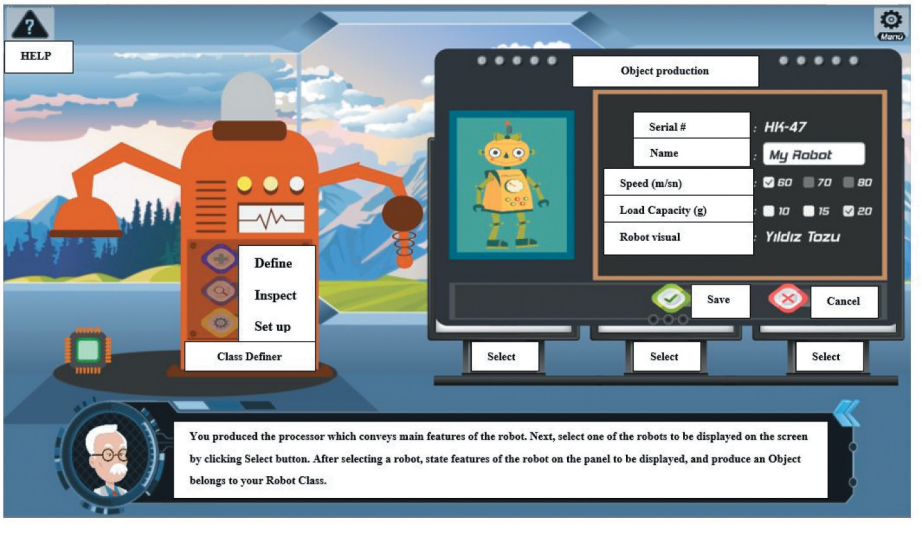

Figure 2. Object instantiation (interface is translated into English).

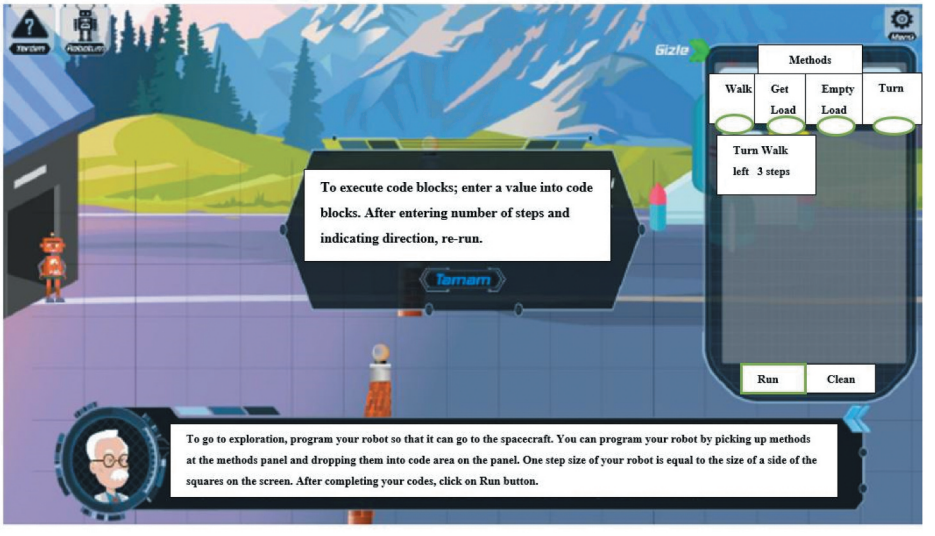

Figure 3. Feedback mechanism.

walking or picking up an object into small steps. The main purpose of these activities is to give the novice programmers a smooth introduction to algorithmic thinking by making them analyze their daily movements step-by-step. By enabling students to develop methods of the robot it is also aimed to help them design a solution which won't be specific to a single situation but will be used in the solution of different problems as the nature of OOP requires.

An animated pedagogical agent (APA) presents instructions of each mission, supportive information and guidance prior to each new task. It is also responsible for giving feedback and contextualized information about an activity in the learning unit. The APA with its additional help section assists integration of OOP concepts into the story of the game by telling the story and providing information about the learning activities. It delivers hints and examples for each mission in the game in order to supply complementary scaffolds in novice programmers' learning experience. In the help menu there are two main topics to guide students in their journey, one to provide procedural information on using the elements in the game environment, and the other to highlight the important points and provide examples specific to the given programming tasks. The instructions and the information in help menu designed in a way to reduce learners' cognitive load. For instance, some key points and concepts of OOP were highlighted in the instructions prior to a task in order to lower learners' cognitive load.

The lack of proper feedback mechanisms in visual programming environments causes problems for novice programmers (Marfisi-Schottman et al., 2019; Meerbaum-Salant et al., 2011). Therefore, a visual and textual feedback mechanism is specifically designed to help novice programmers to understand the execution of their program and debug their codes. 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 (see, Figure 3).

Data collection instruments

The study used four sets of data collection instruments: (1) a Creative Problem-Solving Test (Özkök, 2005) to measure students' CPSS which consists of 30 multiple-choice questions with single response, each with 5 options. For each question the correct answer was assigned 1 point while each wrong answer received 0 points (alpha = 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 judgment skills. (2) an Attitude Scale for Serious Game Assisted Programming Learning (Keçeci et al., 2016) is a 5-point Likert scale with 22 positive and six negative statements (alpha = 0.83). Students were divided into two groups based on their attitude, either positive or with negative. (3) a pre-test to measure students' existing conceptual knowledge of OOP and CT skills, and (4) a post-test to measure students' conceptual knowledge of OOP and CT skills after playing the game. There was no single test that evaluated both the conceptual knowledge of OOP and CT skills at the same time therefore we prepared a test by adapting items from three different tests to measure students' conceptual knowledge of OOP and CT skills. Sixteen questions of the test were adapted from two tests (Gerola, 1997; Pitsatorn, 2003) for measuring object-oriented computer programming semantic knowledge. The last three questions measuring CT skills were adapted from another instrument (Basu, 2016). After adapting all these items into one instrument, an instructor and two researchers re-matched learning objectives with measurement items. Additionally, the Cronbach-alpha coefficient of the instrument for the pre-test was 0.83 based on the answers of participants of the study. Seventeen of the questions were multiple-choice, each with four options. 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. The order of the questions and options were changed in the post-test.

Data collection procedures

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. Data collection and the experiment took place in the computer laboratories of the university where the students study.

In two groups, freshman and sophomore students, all participants were given 50 minutes to answer Creative Problem-Solving Test and the pre-test. A week after the pre-test phase, the second and third sessions of the study were performed consecutively. In the second part of the study, both groups played the game developed, under the supervision of the researcher and the instructor of the course for about two lesson period (90 minutes). Students played the game individually, they didn't interact or cooperate

with other students online or offline during the study. After introducing the game, the teacher did not communicate any content with the students. After a 15-minute break, the researcher administered an attitude scale for serious game-assisted programming learning and the post-test which took 40 minutes.

Data analysis

In order to answer the research questions, a series of different statistical tests were conducted. Data sets of the students' scores of pre-test, post-test, creative problem-solving test and the attitude scale were examined before conducting hypothesis testing through either parametric or nonparametric methods. 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 to analyze the difference between the post-test and pre-test scores on conceptual knowledge of OOP and CT skills. Further, 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 2 × 2 ANOVA test was conducted.

Results

Learning gain of students without programming experience

A Wilcoxon signed-rank test ($z = -4.797$, $p < 0.001$) revealed that there was a statistically significant increase from the pre to the post-test scores of students after playing the developed game with a large ($r = .87$) effect size. The median score on conceptual knowledge of OOP and CT skills test increased from pre-test (Md = 5.00, SD = 12.229) to post-test (Md = 40.00; SD = 16.713).

Learning gain on conceptual knowledge of OOP of students without programming experience

A Wilcoxon signed-rank test ($z = -4.793$; $p < 0.001$) showed that there was a statistically significant increase in the post-test scores on conceptual knowledge of OOP of students after playing the developed game with a large effect size ($r = .87$). The median score on conceptual knowledge of OOP test increased from 0.00 (SD = 9.589) to 35.00 (SD = 14.090). 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 (see, Table 1, Table 2).

Learning gain on CT skills of students without programming experience

A Wilcoxon signed-rank test ($z = -2.500$; $p = 0.012$) revealed that there was a statistically significant increase in the CT skills post-test scores of students after playing the developed game with a medium ($r = .45$) effect size. Their median CT skills score increased from pre-

Table 1. Frequency distribution of participants' number of correct answers for OOP concepts.

	Freshman Students		Sophomore Students	
	Correct Answers		Correct Answers	
Instructional Objective	Pre-test	Post-test	Pre-test	Post-test
Explain class concept	1	13	12	21
Identify object concept	4	12	17	19
Distinguish a class from an object	9	18	22	28
Distinguish object instantiation from class declaration process	1	15	8	21
Give an example of a class and instance from the class	1	6	13	17
State the roles of class attributes	1	16	6	9
Explain object instantiation process	2	11	9	15
State the difference between attributes of a class and attributes of an object	4	18	8	23
Define method concept	2	8	9	13
Explain how classes communicate with each other (calling the members of one class from another class by creating an object)	4	18	14	16
Explain encapsulation concept	2	13	14	14
Explain the role of encapsulation in object-oriented programming	6	18	15	16
Explain polymorphism concept	0	4	15	10
Explain method overriding process	2	10	10	20
Differentiate a base (derived) class from a sub-class in an inheritance relationship	0	6	16	12
List characteristics of object-oriented programming	1	15	12	10

Table 2. Frequency distribution of participants' number of correct answers for CT skills.

	Freshman Students		Sophomore Students	
	Correct Answers		Correct Answers	
Instructional Objective	Pre-test	Post-test	Pre-test	Post-test
Understand conditional statements	18	20	26	27
Write a conditional statement	6	14	16	20
Design a step-by-step solution to a problem	0	5	8	21

test (Md = 5.00 SD = 3.806) to the post-test (Md = 7.50 SD = 5.438). Overall there was an increase in the number of correct answers to all three questions for the instructional objectives after playing the developed game (Table 2).

Learning gain of students with procedural programming experience

A paired-samples t-test ($t(30) = 4.558$, $p < 0.001$) showed a significant difference between the post-test (M = 53.48 SD = 13.721) and pre-test (M = 40.32 SD = 14.772) scores on conceptual knowledge of OOP and CT skills, with a large effect size (Cohen's d = .92). The developed game elicited a statistically significant increase in scores of the students who even attended a semester long programming course.

Learning gain on conceptual knowledge of OOP of students with procedural programming experience

A paired-samples t-test ($t(30) = 3.359$, $p = 0.002$) showed a significant difference between the post-test ($M = 42.58$ $SD = 11.963$) and pre-test ($M = 32.26$ $SD = 12.964$) scores on conceptual knowledge of OOP of students with procedural programming experience, with a large ($d = .83$) effect size. Overall, there was an increase in the number of correct answers to questions for the instructional objectives after playing the developed game (see, Table 1).

Learning gain on CT skills of students with procedural programming experience

A Wilcoxon signed-rank test ($z = -2.849$; $p = 0.004$) revealed that there is a significant difference between the post-test ($Md = 13.00$; $SD = 3.801$) and pre-test ($Md = 10.00$; $SD = 4.11$) scores on CT skills of students with procedural programming experience, with a large effect size ($r = .53$). There was an increase in the number of correct answers to all three CT questions for the instructional objectives after playing the developed game (Table 2).

Comparison of the achievement scores students with and without programming experience

An independent-samples t-test ($t(59) = 4.115$, $p < .001$) revealed that there was a significant difference between the achievement scores of the freshman students without programming experience ($M = 29.33$, $SD = 14.55$) and of the sophomore students with procedural programming experience ($M = 13.16$, $SD = 16.08$).

Covariate effects on the achievement scores

A general linear model 2×2 ANOVA test investigated whether CPSS and the attitudes towards digital game-based learning of programming together or pairwise influence the students' achievement scores. The following statistical outcomes were found: *(i)* 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, $F(1, 55) = .229$, $p = .634$, between the students' level of CPSS and achievement scores, $F(1, 55) = .299$, $p = .586$, and between the students' attitudes towards digital game-based learning of programming and achievement scores, $F(1, 55) = 1.124$, $p = .294$. *(ii)* There was a strong positive, statistically significant, correlation (Pearson $r(59) = .96$, $p < .001$) between the students' overall achievement scores and achievement scores in OOP concepts. *(iii)* There was a moderate positive, statistically significant, correlation between (Spearman's rho $r_s(56) = .338$, $p < .05$) the students' overall achievement scores and achievement scores in CT skills.

Discussion and conclusion

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 problems, designing and implementing solutions to problems in CS (Laakso et al., 2021; Liu et al., 2011; Wing, 2008). Therefore, the first two questions of the study focused on the effects of playing the developed game on students' learning of conceptual knowledge of OOP and CT skills. 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 (Abbasi et al., 2021; Guenaga et al., 2021; Livovsky & Poruban, 2014; Mathrani et al., 2016; Miljanovic & Bradbury, 2017; Muratet et al., 2011; O'Kelly & Gibson, 2006; Phelps et al., 2005; Sun et al., 2021).

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 both freshman and sophomore students 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 (Abbasi et al., 2021; Livovsky & Poruban, 2014; Mladenovic et al., 2021; O'Kelly & Gibson, 2006; Phelps et al., 2005; Wong et al., 2016) by demonstrating inferential statistical analyses. Similarly, the analysis of students' scores on CT skills revealed that the mean post-test scores of both groups 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 was 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 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 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 four learning objectives were explaining

polymorphism concept, differentiating a base class from a sub-class and listing characteristics of OOP. A 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 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 learning objectives were not effective for students who started computer programming with procedural programming and shifted to OOP, and 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; Toth & Lovaszova, 2021; Weintrop & Wilensky, 2019; Xinogalos, 2016) because freshman students who have no prior programming experience had improvement in each of these learning objectives. However, it seems that more and varied activities along with/without teacher and peers' scaffolds may be needed to compare and contrast information, and to analyze code sets in their parts or kinds, in order to develop sufficient mindful abstraction in contents (i.e. explaining polymorphism concept, differentiating a base class from a sub-class) at which novice students' progress was poor.

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 et al., 2002) which encourage the use of ill-structured problems in a learning environment to support discovery learning.

Livovsky and Poruban (2014) claimed that long texts in instructions affected students' learning negatively. Similarly, Sweller et al. (1998) argued that human beings have a limited capacity for working memory, so 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 each activity 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, by providing empirical data, 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 with serious games. Additionally, it is important that a serious game 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 games to inform and guide students about their missions and mistakes.

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

Novice programmers are likely to have problems when they are first introduced to procedural programming and then move to OOP (Hadjerrouit, 1999; Xinogalos, 2016). To provide a fresh insight into the current problem, this study compared the mean achievement scores of freshman students without programming experience and sophomore students with procedural programming experience. The findings 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. Moreover, the results reveal that freshman students (M = 29.33, SD = 14.55) made more progress than the sophomore students (M = 13.16, SD = 16.08). One of the possible reasons of this result is that freshman students' lack of prior knowledge on programming compared to sophomore students.

Interaction among CPSS, attitudes towards digital game-based learning of programming and learning

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 (Abbasi et al., 2021; Aho, 2012; Barr & Stephenson, 2011; Wing, 2006). Additionally, researchers have advised that CT should be introduced to students as early as possible (Chen et al., 2019; Liu et al., 2011; Lu & Fletcher, 2009; Qualls & Sherrel, 2010). However, in this study, a general linear model 2×2 ANOVA 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 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. One possible explanation of this result is that the items in the CPSS test mostly covers symmetry algebra. Therefore, in order to have a better understanding of the nature of the relationship between students' CPSS and programming performance, a follow-up study could be conducted with another instrument measuring CPSS with a wide range of items, and with a more heterogeneous group of students (because the student sample of this study was drawn from a single department)

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 et al., 2011; Mathrani et al., 2016; Muratet et al., 2011; Paiva et al., 2020; Ramírez-Rosales et al., 2016; Wong et al., 2016). Some of the studies (Phelps et al., 2005; Wong et al., 2016) claim that serious games could be effective in fostering novice programmers' learning of programming based on the data of students' perceptions. The findings of the current study contradict such claims by showing 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. Therefore, more attention should be paid to the instructional design of activities in a serious game more than the motivational aspects.

Implication for practice and recommendations for further research

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 (Livovsky & Poruban, 2014; Mathrani et al., 2016; Miljanovic & Bradbury, 2017; Sun et al., 2021). Additionally, by providing empirical data on the current issue, this study has validated the instructional design model which is a synthesis of experiential gaming framework and 4C/ID model: This study has beneficial theoretical and practical implications for digital game-based learning of programming, and may provide valuable information and guidance for researchers and practitioners.

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 established based on Kiili's (2005) experiential gaming model and 4C/ID model (Van Merriënboer et al., 2002) to encourage exploratory 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 both models. 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.

In the developed game, supportive and procedural information was provided to students via a mission information panel, an instruction panel and a help menu. Additionally, abstract concepts of OOP were represented as concrete objects in the game like representing an abstract concept, class, as a programmable chip/processor which contains the specifications of a robot. Moreover, CT skills were practiced in a simulation environment to help students understand the necessity and the forms of utilization of such concepts and skills. The findings of this study support the idea that serious games for programming should have a feedback mechanism to help students understand the deficiencies in their solutions, improve these solutions, and thus overcome their learning difficulties (Barnes, Chaffin et al., 2007; Esteves et al., 2011; Kiili, 2005).

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 such as experience in CT or studying the game with different task regimes.

Further research should also consider observing student performance and progress in the game-based assessment in a stealth assessment format: The game may have an integrated logging system that records every interaction of participants with the platform, user's performance and progress through the tasks can be measured with the data collected automatically from the logging system. Such data may show fine-grained edits made by students, amount of students' work and its typology, the order of dragging a given code, the number of times a student clicks "run" or the use of those metaphorical machines for each task, and referring to supportive information in each task. Having students' path towards a task solution helps to reveal the students' misconceptions. Moreover, with large data sets of logging systems, learning analytics techniques may be used to identify learning behavior patterns in these environments.

This study is aware that learning from games often needs complementary teaching and scaffolds. For instance, to meet each student's needs, a teacher's and peers' support, social scaffolds, play a crucial role in complementing and prolonging the support provided by the APA and other tools of the game. Effective implementation and orchestration of distributed scaffolding by relevant game components, teachers and peers may be the subject of new studies. Also, as the game scaffolds fade-off, when students are required to transfer OOP knowledge acquired in the game environment to code outside the game, students' understanding and readiness for that fading should be monitored, and complementary support needed should be provided by a teacher. Conditions of a teacher's scaffold, e.g. additional elaborative activities, as an extension to and as a complement the support already built into the game environment rather than just replicating existing support should not be overlooked and warrants further investigation.

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, and because of absence of a control group. 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 with control groups should be conducted: This would also increase internal validity of the studies. 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. Third, using the same instrument as a pre-test and a post-test is another limitation. A follow up study could be conducted with two different instruments measuring the same learning objectives.

Finally, depth, nature (selection, integration, organization) and timing (before, during and after the game activities) of supportive information along with curricular task regimes are crucial in exploratory learning setting. As a critical limitation, this study investigated merely one type of supportive information in helping selection of relevant information.

Our further work will focus upon influence of different type of supportive information considering depth, nature and timing, on different types of learning outcome such as production of code in and outside of the game environment, incorporation of OOP knowledge and CT skills, and in-game performance. Further investigations should broaden understanding of the interactions between the varieties of scaffolds provided by the teacher and/or the game not only in increasing students' cognitive engagement in selection but also in organization and integration of new programming knowledge. In a similar vein, additional work should identify which scaffold characteristics effectively promote learning of OOP procedures tailored in classroom curriculum whilst maintaining flow and enjoyment in the game.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Ali Akkaya http://orcid.org/0000-0001-7955-7407
Yavuz Akpinar http://orcid.org/0000-0002-9406-3795

References

Abbasi, S., Kazi, H., Kazi, A. W., Khowaja, K., & Baloch, A. (2021). Gauge OOP in student's learning performance, normalized learning gains and perceived motivation with serious games. *Information*, *12*(3), 101. https://doi.org/10.3390/info12030101

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832–835. https://doi.org/10.1093/comjnl/bxs074

Akar, S. G., & Altun, A. (2017). Individual differences in learning computer programming. *Contemporary Educational Technology*, *8*(3), 195–213.

Akkaya, A. (2019). Eğitsel oyunların öğrencilerin nesne tabanlı programlamanın temel kavramsal bilgisi ve bilgi işlemsel düşünme becerilerine etkisi. *Unpublished master thesis*. Bogazici University. Turkey.

Al-Sakkaf, A., Omar, M., & Ahmad, M. (2019). A systematic literature review of student engagement in software visualization. *Computer Science Education*, *29*(2–3), 283–309. https://doi.org/10.1080/08993408.2018.1564611

Arnab, S., Lim, T., Carvalho, M. B., Bellotti, F., De Freitas, S., Louchart, S., Suttie, N., Berta, R., & De Gloria, A. (2015). Mapping learning and game mechanics for serious games analysis. *British Journal of Educational Technology*, *46*(2), 391–411. https://doi.org/10.1111/bjet.12113

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). NY: ACM.

Barnes, T., Richter, H., Chaffin, A., Godwin, A., Powell, E., Ralph, T., . . . Jordan, H. (2007). Game2Learn: A study of games as tools for learning introductory programming concepts. In I. Russel, S. Haller, J. D. Dougherty, & S. Rodger (Eds.), *The 38th ACM Technical Symposium on Computer Science Education* (pp. 7–9). ACM.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12. *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Basu, S. (2016). *Fostering synergistic learning of computational thinking and middle school science in computer-based intelligent learning environments* [Unpublished PhD thesis]. Vanderbilt University.

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* (pp. 1–6). Seattle: IEEE.

Carlisle, M. C. (2009). RAPTOR: A visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges*, *24*(4), 275–281. https://dl.acm.org/doi/abs/10.5555/1516546.1516591

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: A comparison of graphical and textual languages. *Computer Science Education*, *29*(1), 23–48. https://doi.org/10.1080/08993408.2018.1547564

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. https://dl.acm.org/doi/10.5555/364133.364161

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

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

Eleftheriadis, S., & Xinogalos, S. (2020). Office madness: Design and pilot evaluation of a serious game for learning the C++ programming language. In I. Marfisi-Schottman, F. Bellotti, L. Hamon, and R. Klemke (Eds.), *Games and learning alliance. GALA 2020. Lecture notes in computer science* (Vol. 12517, pp. 389–394). Springer.

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. https://doi.org/10.1111/j.1467-8535.2010.01056.x

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 Proceedings of International Scientific Conference eLearning and Software for Education. 1* (pp. 503–510). Bucharest: "Carol I" National Defence University Publishing House.

Gerola, R. J. (1997). *Identification of object-oriented computer programmer mastery status through evaluation of object-oriented programming semantic knowledge* [Unpublished PhD thesis]. University of Southern California.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming. In *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*. (pp. 267–272). NY: ACM.

Guenaga, M., Eguíluz, A., Garaizar, P., & Gibaja, J. (2021). How do students develop computational thinking? *Computer Science Education*, *31*(2), 259–289. https://doi.org/10.1080/08993408.2021.1903248

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. https://doi.org/10.1007/s11423-007-9073-2

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27. https://doi.org/10.1145/1378704.1378713

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *ACM SIGCSE Bulletin*, *31*(3), 171–174. https://doi.org/10.1145/384267.305910

IADB. (2020). *Play Challenge*. from https://comunidad.socialab.com//challenges/PLAY

Kazimoglu, C. (2013). *Empirical evidence that proves aserious game is an educationally effective tool for learning computer programming constructs at the computational thinking level* [Unpublished PhD thesis]. University of Greenwich.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences*, *47*, 1991–1999. https://doi.org/10.1016/j.sbspro.2012.06.938

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

Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education*, *8*(1), 13–24. https://doi.org/10.1016/j.iheduc.2004.12.001

Kolb, D. (1984). *Experiential learning: Experience as the source of learning and development*. 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), 1–21. https://doi.org/10.1145/1868358.1868361

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. https://doi.org/10.1076/csed.13.4.249.17496

Kong, S. C., & Wang, Y. Q. (2019). Positive youth development from a "3Cs" programming perspective. *Computer Science Education*, *29*(4), 335–356. https://doi.org/10.1080/08993408.2019.1599646

Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education*, *14*(4), 1–28. https://doi.org/10.1145/2662412

Krath, J., Schürmann, L., & von Korflesch, H. F. (2021). Revealing the theoretical basis of gamification: A systematic review. *Computers in Human Behavior*, *125*. https://doi.org/10.1016/j.chb.2021.106963

Laakso, N. L., Korhonen, T. S., & Hakkarainen, K. P. J. (2021). Developing students' digital competences through collaborative game design. *Computers & Education*, *174*. https://doi.org/10.1016/j.compedu.2021.104308

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. https://doi.org/10.1016/j.entcom.2017.12.005

Lister, R. (2011). Programming, syntax and cognitive load (part 2). *ACM Inroads*, *2*(2), 21–22. https://doi.org/10.1145/1963533.1963539

Liu, -C.-C., Cheng, Y.-B., & Huang, C.-W. (2011). The effect of simulation games on the learning of computational problem solving. *Computers & Education*, *57*(3), 1907–1918. https://doi.org/10.1016/j.compedu.2011.04.002

Livovsky, J., & Poruban, J. (2014). Learning object-oriented paradigm by playing computer games: Concepts first approach. *Central European Journal of Computer Science*, *4*(3), 171–182. https://doi.org/10.2478/s13537-014-0209-2

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

Marfisi-Schottman, I., George, S., & Leconte, M. (2019). TurtleTable: Learn the basics of computer algorithms with tangible interactions. In M. Gentile, M. Allegra, and H. Söbke (Eds.), *Games and learning alliance. GALA 2018. Lecture notes in computer science* (Vol. 11385, pp. 291–300). Springer.

Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). Playt: Game based learning approach for teaching programming concepts. *Educational Technology and Society*, *19*(2), 5–17.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In G. Rößling, T. Naps, & C. Spannagel (Eds.), *Proceedings of the 16th annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 168–172). NY: ACM.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022

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). NY: ACM.

Mladenovic, M., Zanko, Z., & Aglic, M. (2021). The impact of using program visualization techniques on learning basic programming concepts. *Computer Applications in Engineering Education*, *29*(1), 145–159. https://doi.org/10.1002/cae.22315

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* (Vol. 30, pp. 61–73). Blackwell Publishing Ltd.

O'Kelly, J., & Gibson, J. P. (2006). RoboCode & problem-based learning. *ACM SIGCSE Bulletin*, *38*(3), 217–221. https://doi.org/10.1145/1140123.1140182

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

Paiva, J. C., Leal, J. P., & Queirós, R. (2020). Fostering programming practice through games. *Information*, *11*(11), 498–517. https://doi.org/10.3390/info11110498

Phelps, A. M., Egert, C. A., & Bierre, K. J. (2005). MUPPETS: Multi-user programming pedagogy for enhancing traditional study. *Proceedings of the 4th Conference on Information Technology Curriculum* (pp. 100–105). NY: ACM.

Pitsatorn, P. P. (2003). *Object-oriented programming training: Bottom-up versus top-down approach* [Unpublished PhD thesis]. Claremont Graduate University.

Prensky, M. (2003). Digital game-based learning. *ACM Computers in Entertainment*, *1*(1), 1–4. https://doi.org/10.1145/950566.950596

Qualls, J. A., & Sherrel, L. B. (2010). Why computational thinking should be integrated into the curriculum. *Computing Sciences in Colleges*, *25*(5), 66–71. https://dl.acm.org/doi/10.5555/1747137.1747148

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. Cisneros (Eds.), *4th International Conference in Software Engineering Research and Innovation* (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). NY: ACM.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

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.

Sun, L., Guo, Z., & Hu, L. (2021). Educational games promote the development of students' computational thinking. *Interactive Learning Environments*, 1–15. https://doi.org/10.1080/10494820.2021.1931891

Sweller, J., van Merriënboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, *10*(3), 251–296. https://doi.org/10.1023/A:1022193728205

Toth, T., & Lovaszova, G. (2021). Mediation of knowledge transfer in the transition from visual to textual programming. *Informatics in Education*, *20*(3), 489–511. https://doi.org/10.15388/infedu.2021.20

Tsarava, K., Moeller, K., & Ninaus, M. (2019). Board games for training computational thinking. In M. Gentile, M., , Allegra, and H. Söbke (Eds.), *Games and learning alliance. GALA 2018. Lecture notes in computer science* (Vol. 11385, pp. 90–99). Springer.

Van Haaster, K., & Hagan, D. (2004). Teaching and learning with BlueJ. *Issues in Informing Science & Information Technology*, *1*, 455–470. https://doi.org/10.28945/752

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. https://doi.org/10.1007/BF02504993

Veerasamy, A. K., D'Souza, D., Lindén, R., & Laakso, M. J. (2019). Relationship between perceived problem-solving skills and academic performance of novice learners in introductory programming courses. *Journal of Computer Assisted Learning*, *35*(2), 246–255. https://doi.org/10.1111/jcal.12326

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

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* (pp. 955–960). Piscataway, NJ: IEEE.

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.

Weintrop, D., & Wilensky, U. (2016). Playing by programming: Making gameplay a programming activity. *Educational Technology*, *56*(3), 36–41. https://www.jstor.org/stable/44430491

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, *142*, 1036–1046. https://doi.org/10.1016/j.compedu.2019.103646

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

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*(1881), 3717–3725. https://doi.org/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.

Xinogalos, S. (2016). Designing and deploying programming courses. *Education and Information Technologies*, *21*(3), 559–588. https://doi.org/10.1007/s10639-014-9341-9