Journal of Information Technology Education: Research

An Official Publication of the Informing Science Institute
InformingScience.org

JITEResearch.org

Volume 16, 2017

Students Learning to Program by Developing Games: Results of a Year-long Project in Primary School Settings

Emmanuel Fokides University of the Aegean, Rhodes, Greece fokides@aegean.gr

Abstract

Aim/Purpose	The purpose of this study was to examine whether the authoring of computer games in a mainstream primary school setting can support the learning of game design and programming concepts.
Background	Despite the benefits for students when they learn how to program and the significant body of research regarding this matter, these benefits are still under debate, and the teaching of programming has a relatively undeveloped pedagogy. With this in mind, a project was designed and implemented, having constructionism as its theoretical framework. Also, Microsoft's Kodu Game Lab was used for the development of students' games.
Methodology	The project lasted for almost a school year (fifty two-hour sessions) and the target group was fifth-grade students (ages 10-11). A total of 138 students participated, coming from five schools in Athens, Greece. Students were divided into three groups. While all groups worked in pairs, to the first there was no teachers' guidance, to the second there was limited teachers' guidance, and to the third, a combination of teacher-led and pair work was used. Each group developed three games of escalating complexity and a total of 207 games were analyzed. Data were collected by analyzing students' games and with a short questionnaire.
Contribution	The study contributes to the debate surrounding the pedagogy of computer game authoring as a tool for teaching programming. That is because few studies have examined the above in mainstream settings, having as a target group primary school students. Furthermore, the present study is in contrast to most studies which involved intensive research projects, since it lasted for almost a school year.

Accepted by Editor Athanassios Jimoyiannis │Received: September 18, 2017│ Revised: November 21, 2017 │ Accepted: November 29, 2017.

Cite as: Fokides, E. (2017). Students learning to program by developing games. Results of a year-long project in primary school settings. *Journal of Information Technology Education: Research, 16,* 475-505. https://doi.org/10.28945/3893

Findings	It was found that the most commonly used programming concept was conditions, followed by variables and loops, while Boolean logic and functions were the least used ones. The most problematic concepts proved to be Boolean logic and loops, closely followed by functions. The least problematic concepts were conditions and variables. Also, the number of programming concepts that were used was increasing in each game, while the errors were decreasing. All in all, students' final games fall into the relational level according to a modified version of the SOLO taxonomy. While the findings indicated that, as well as learning some basic programming concepts, students enjoyed the activity and demonstrated positive attitudes to learning programming by developing games; it was also found that the teaching method did not have any effect on the learning outcomes nor in their views for game authoring.
Recommendations for Practitioners	Extended projects can be considered for teaching programming to primary school students, using visual programming tools that allow the development of games. The classes' teachers can undertake the task to teach programming if they are properly trained. The SOLO taxonomy can be used for assessing students' games.
Future Research	Future studies can examine a variety of game-like programming environments and the target group can be older or younger students. The assessment of students' games is also an interesting topic. Finally, research can be conducted by using other devices and compare the results.
Keywords	constructionism, game design, Kodu, primary school students, programming, SOLO taxonomy

INTRODUCTION

Technology has brought substantial changes in all aspects of our lives, education included. In fact, the educational systems, all around the world, are under a constant pressure to adapt to the needs of people who work and live in technology-enriched environments. Although the debates surrounding the use of technology and computers in education are still thriving, there is a significant shift in their focus. In the past, the focus was on how to render students adept users of devices and applications. Nowadays, much of the debate is about how students can become skilled designers and creators of digital artifacts (Organisation for Economic Co-operation and Development [OECD], 2015; Papert, 1993).

Evidently, the need for students to become content creators is closely related to the acquisition of computer programming skills (Resnick et al., 2009). Indeed, the benefits students have when they learn how to program were noted in the very early attempts to integrate computers in education (Papert, 1980). Programming helps students to develop their analytical and synthetic thinking, fosters their skills in designing and solving algorithms, and has a positive impact on their creativity and imagination (Liu, Cheng, & Huang, 2011). A number of programming languages were developed in order to teach programming to primary and secondary school students (e.g., Logo and Scratch). While there is a significant body of research regarding their educational benefits, it can be argued that these benefits have not been adequately researched in everyday school settings (Luckin, Bligh, Manches, Ainsworth, Crook, & Noss, 2012), and that the teaching of programming still has a relatively undeveloped pedagogy (Maguire, Maguire, Hyland, & Marshall, 2014). In addition, as students face quite a lot of problems when they learn how to program, researchers suggested that the teaching of programming should have game-like characteristics, so that the whole process becomes an enjoyable experience (e.g., Margulieux, Guzdial, & Catrambone, 2012) and that it should start as early as possible (Kalelioğlu, 2015).

Against this background, computer game authoring presents an interesting alternative method for teaching programming concepts and practices. Research regarding the use of digital games in education is extensive and diverse. A substantial body of the literature examined how digital games can be used as means for delivering the content of various learning/teaching subjects (e.g., Felicia, 2012; Gee, 2014; Hwang & Wu, 2012). The development of digital games, by the students, for learning about a range of subjects as well as for supporting the development of digital skills is another research field (e.g., Ke, 2014; Yang & Chang, 2013). In both cases, the results were interesting; the learning outcomes were good, students enjoyed the opportunity to be playful and creative, and this had positive effects in terms of increased commitment to learning.

Beyond enhancing knowledge in other subjects, there is a growing number of studies which examined how game authoring introduced students to programming. However, there are few studies which targeted primary school students (e.g., Baytak & Land, 2010). Also, few studies examined whether the authoring of computer games increases students' understanding of computer science concepts (e.g., Denner, Werner, & Ortiz, 2012) or what kind of knowledge students learn from such an activity (Koh, Basawapatna, Bennett, & Repenning, 2010). Moreover, there are few studies which examined the above within everyday classroom settings (Wilson, Hainey, & Connolly, 2012).

These were the areas of interest of the present study. It explored the introduction of a course in computer game authoring, addressed to fifth-grade primary school students (ages 10-11), having a two-fold purpose: (a) to examine whether primary school students can understand and use game design practices and programming concepts when they author their own games, and (b) to examine their views, attitudes, and perceptions regarding their involvement in game authoring activities.

The paper is organized as follows. First, a brief review of the literature on programming as a teaching/learning subject is presented, followed by a review of the literature on the use of games for teaching programming. Next, the project's rationale and methodology are analyzed, followed by results. Subsequently, results are discussed and the conclusion completes the work.

PROGRAMMING AS A TEACHING/LEARNING SUBJECT

The existing literature indicates considerable benefits for students when they learn how to program. Besides developing a positive attitude towards learning computing in general (Keren & Fridin, 2014), learning how to program has an impact on students' understanding of mathematical concepts and improves their problem-solving skills (Akcaoglu & Koehler, 2014). Positive effects on their creativity and imagination were also noted (Liu et al., 2011). Likewise, when students perform well in programming, they tend to use more meta-cognitive management strategies (Bergin, Reilly, & Traynor, 2005).

Unfortunately, the teaching of this subject is not an easy task, and students of all ages do face problems (Brennan, 2013; Saeli, Perrenet, Jochems, & Zwaneveld, 2011). Most problems arise because students have difficulties in understanding/learning (a) what programming is for, (b) what is going on inside the machine, (c) the syntax of a programming language, (d) certain programming concepts (e.g., loops and conditions), and (e) testing and debugging programs (du Boulay, 1986). In addition, students expect the computer to interpret correctly what they write (Pea, 1986), and they do not understand that everyday words have a different meaning/use in programming (du Boulay, 1986). Basic program planning is also a major source of difficulties for novice programmers (Robins, Rountree, & Rountree, 2003), and they need more instruction on how to put together the pieces of a program (Soloway, 2013). Others suggested that the difficulties in programming arise from students' dispositions, behavior, and attitudes (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986). For instance, students tend to delete their errors rather than try to fix them; they mess with the code unpurposely rather than try to understand what the cause of the problem is. Especially for young students, the lack of logical reasoning and the -still- undeveloped algorithmic and critical thinking, are the main reasons for all the above issues (Govender et al., 2014; Robins et al., 2003).

Despite the fact that most problems could have been eased if better teaching practices were implemented (Perkins et al., 1986), the teaching of programming still has a relatively undeveloped pedagogy, while inappropriate teaching methods and resources are often used (Maguire et al., 2014). Then again, the situation seems to slowly change, as in many countries and across all levels of education, the curriculum now includes the teaching of programming in a more systematic way (e.g., Grgurina, Barendsen, Zwaneveld, van Veen, & Stoker, 2014; Grout & Houlden, 2014; Lee, Martin, & Apone, 2014). Alas, this does not hold true for the Greek educational system. Indeed, in Greece's primary school curriculum, programming concepts are taught only to the last two grades (ages 10-12) as -a rather small- part of the ICT curriculum, which, in turn, is taught just for one hour a week. The objectives are for students to understand algorithms and variables and to be able to solve programming problems using Logo-like applications. One can easily understand that the above objectives cannot be achieved with such a minimal time allocation (Grigoriadou, Gogoulou, & Gouli, 2002) and, quite logically, students face problems (Fokides & Atsikpasi, 2017).

As for the programming tools that can be used, there is a variety of them, ranging from drag and drop applications to programming robots. For example, Alice, a 3D programming environment, helped students to learn fundamental programming concepts (Zhang, Liu, Ordóñez de Pablos, & She, 2014); robots' programming improved their geometric thinking and metacognitive tasks (Keren & Fridin, 2014); game development -through programming- supported their understanding of computer science concepts (Denner et al., 2012). The most commonly used programming languages are the visual ones, in which, instead of using text, the code is created with the use of drag and drop environments and graphical representations of program elements as the constituents of a program (e.g., icons). Because they are easy to understand and use, visual languages are preferred over textual systems for introducing primary school students to programming (Murnane, 2010).

LEARNING HOW TO PROGRAM BY AUTHORING COMPUTER GAMES

Computer games are probably the most common young people's entertainment medium; eight out of ten children and teenagers (ages 5-15) play games using a variety of electronic devices (Ofcom, 2013). The popularity of computer games has led to a surge in research in the area of game-based learning over the past twenty years (e.g., de Freitas, 2006; Felicia, 2011, 2012; Gee, 2014; Hwang & Wu, 2012; Ke, 2009; Prensky, 2007; Squire, 2005). While Prensky (2004) claims that games are the most powerful learning tools ever known, there is no common consensus regarding the extent to which computer games impact students' learning. Some researchers observed improved learning outcomes, others observed a negative impact, and others reported no effects at all (e.g., Perrotta, Featherstone, Aston, & Houghton, 2013). Nevertheless, the most common findings were that computer games had a positive impact on problem-solving skills, motivation, and engagement (e.g., Becta, 2006; Connolly, Boyle, MacArthur, Hainey, & Boyle, 2012). Also, researchers suggested that learning with games has to be supported by -equally- effective instructional measures (Egenfeldt-Nielsen, 2006) and a well-developed games' pedagogy (Ulicsak & Williamson, 2011).

The main argument for using computer games as part of the ICT curriculum, and strongly supported in the literature, is that students should be engaged in activities that lead to the development of digital products; students should not only be consumers of digital games but also producers of them (e.g., Caperton, 2012; Jones et al, 2011; Li, 2010). Much of the research focused on the literacy development when students developed games (e.g., Beavis, O'Mara, & McNeice, 2012; Merchant, 2013). In this respect, it was found that game authoring supported the learning of mathematics at primary level (Ke, 2014; Shaw, Boehm, Penwala, & Kim, 2012), narrative (Howland, Good, du Boulay, 2013), and the learning of science concepts (Baytak, Land, Smith, & Park, 2008; Yang & Chang, 2013).

However, there is another body of research which looked at how game authoring enabled students to understand programming concepts and practices. Accordingly, game authoring, as a programming activity, has been studied in tertiary and secondary education (e.g., Harteveld, Smith, Carmichael, Gee, & Stewart-Gardiner, 2014; Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012), but far less in primary education. It can also be argued that the evidence of the learning outcomes of game authoring is not well documented since the literature is not extended. Few studies examined the effects of computer game development as a pedagogical activity (Owston, Wideman, Ronda & Brown, 2009), and there is little evidence of the role of game design and programming in digital literacy development (e.g., Caperton, 2012. Moreover, studies that targeted children are scarce (e.g., Baytak & Land, 2010).

The most common arguments in favor of game design as a programming activity have to do with what is called "21st-century skills", a set of capacities that students need to develop in order to succeed in the information age. It is suggested that game design encourages systemic, critical, and computational thinking while fostering literacy skills (computational as well as language) (Hayes & Games, 2008; Salen, 2007). Since games are another form of media, their development can be seen as a form of digital literacy practice, which requires software design skills (Kafai & Peppler, 2011; Payton & Hague, 2010). Students need to develop a critical understanding of how this medium works not only by analyzing games but also by making them, given that creating games allows for a more engaging form of learning. In the same line of thinking, other researchers introduced the notion of "game literacy" as a subset of media literacy (Salen, 2007; Zimmerman, 2009). They considered game literacy as important because it enables children to view games as dynamic systems/structures with complex interactions and, because of that, they should be aware of how these structures function.

As for the learning outcomes when developing games, in terms of what programming concepts can be learned and to what extent, the results were mixed as in other tools for learning how to program. On the other hand, there is a consensus in the literature that students find game authoring motivating (e.g., Hwang, Hung, & Chen, 2014; Ke, 2014; Robertson, 2013). Students can pursue their own interests and develop a sense of ownership, which is a powerful lever for learning. Fun and enjoyment were also indicated as powerful motivating factors when developing games, which, in turn, led to increased commitment to learning (Sanford & Madill, 2007)

An important aspect of game authoring is that of "learning by design" (Ke, 2014). Students have little experience in following the design process (researching, planning, and bringing everything together) because conventional school assignments rarely give them the opportunity to spend an extended period of time on complex projects (Kafai, 1996). For Kafai, learning by design is important because it helps students to learn how to learn; since there is no single solution to the game design problems, students can choose their own strategies and solutions to deal with the complexity of the game making activity.

Extended projects were considered by Harel (1991) and Kafai (2012). They found that students developed an increased understanding of programming and mathematical concepts. A noteworthy development of metacognitive skills in planning and monitoring their work was also found. Constructionism provided the theoretical framework for their projects, and they emphasized that although extended projects were essential for students' learning, it would be difficult to integrate such an approach into the current ICT curriculum, which allocates just a few hours a week for this course. This is probably the reason why there is a lack of empirical evidence regarding the effects of game authoring in the learning of computer science concepts within everyday school settings. Yet, the existing studies suggested that using this approach is more effective and more motivational than a non-gaming approach and traditional lectures (Liu et al., 2011; Perrotta et al., 2013).

Constructionism as a Framework for Teaching Programming

The current study explores constructionism as a suitable approach for teaching primary school students how to design and program computer games. This focus originates from the bulk of constructionist research which has been conducted using Logo or Scratch to teach primary school students about mathematics (e.g., Kafai, 2012), programming and science (e.g. Baytak & Land, 2010) or how to create multimedia artifacts (e.g., Kafai & Peppler, 2011).

Constructionism was conceived more than thirty years ago by Papert (1980) and thereafter numerous researchers have been influenced by his ideas (Bulfin, Henderson, & Johnson, 2013). In essence, Papert extended the constructivist learning theory by suggesting that children build their own understandings more effectively when they actively construct artifacts that have some personal and cultural meaning for them. In making such artifacts, a process takes place, that of constructing their own understanding of the knowledge required to make them. Papert's views can be summarized into "the eight big ideas of constructionism" (Papert, 1999).

- Learning by doing. We learn better when learning is part of doing something we find really interesting or when we use what we learn to make something we really want.
- Technology as a building material. Technology can be used to make a lot of interesting things and we can learn a lot more by making them.
- Hard Fun. We learn best if we enjoy what we are doing. But enjoyment does not imply "easy"; the best fun is hard fun.
- Learning to learn. Nobody can teach us everything we need to know, we have to take charge of our own learning.
- Taking time, the proper time for the job. To do anything important we have to learn to manage time by ourselves.
- Freedom to get things wrong. Nothing important works the first time. We have to look carefully at what happened when it went wrong. To succeed, we need the freedom to make mistakes.
- Teacher as co-learner. The teacher is present as a co-learner and the mode of learning is less dominated by a strict curriculum. Students are encouraged to manage tasks and timing by themselves.
- Using computers to learn in a digital world. Learning about computers is essential but it is most important to use them for learning about everything else.

Acknowledging that making things with computers often involves some kind of programming and that such activities can be difficult, constructionism seeks to find ways to support learners. Among them are situating learning in the context of use, collaboration between teachers and peers and between peers, and making available computer-based learning environments that provide opportunities for learning. In particular, this theory of learning informs much of the research into computer game authoring from a programming perspective (e.g., Harel, 1991; Kafai & Peppler, 2011; Kafai & Resnick, 1996) and, thus, provides an appropriate theoretical framework for the current study. Consequently, this study embraces the notion that students should be producers as well as consumers of digital media and that they should be given the opportunity to use computers as a means of creative expression for making a product of personal relevance to them, an enterprise not -clearly- present in the in the current Greek program of study.

METHOD

Given that the development of digital games presents an interesting alternative method for teaching programming to students, as presented in the preceding section, a project was designed and implemented in order to examine what the learning outcomes of such an endeavor might be, having as a target group fifth-grade primary school students (ages 10-11). A quasi-experimental design, with one experimental and two control groups, was chosen because data from intact classroom groups were analyzed for their differences in the games they developed, as it will be further elaborated in the coming sections.

Research Hypotheses

The main purpose of the study at hand was to examine whether the design of digital games by students had an impact on their understanding and ability to use programming concepts. On this basis, the following hypotheses were formed:

- H1: Students can understand and effectively use basic programming concepts, such as Boolean logic, functions, conditions, loops, values, and variables, when they design their own digital games. As a result, their games are -up to a degree- functional and playable.
- H2: The learning outcomes, in terms of how correctly and efficiently the above programming concepts are used, and how complete and functional the games are, depend on the teaching approach that is used.
- H3: Students form positive attitudes and perceptions regarding their involvement in game authoring activities.
- H4: Their views depend on the teaching approach that is used.

It should be noted that, in this study, H2 and H4 had a substantial importance. In a previous project with similar settings as the present one, it was found that, if enough time is allocated for the teaching of programming concepts, the teaching method does not actually have an impact on the learning outcomes (Chatzigrigoriou & Fokides, 2016). Thus, it was considered interesting to check whether this finding was circumstantial or if it could be replicated.

Participants and Duration of the Project

As already mentioned, the target group was primary school students attending the fifth-grade. This grade/age group was selected because, according to the Greek primary school curriculum, at this grade students start to learn the basics of programming. An email invitation to participate in the project was issued, addressed to primary schools in Athens, Greece. Most of the schools that responded affirmatively had to be excluded because (a) they were too far apart, (b) they were private schools and, consequently, the sample would not be homogeneous in terms of the socioeconomic status of students, and (c) the computer labs did not have a sufficient number of computers so as to assign one to a pair of students or because the computers were outdated. A second set of selection criteria applied to students of the shortlisted schools: (a) to have never developed a computer game as part of their formal or informal ICT lessons, (b) to have no previous knowledge of programming, (c) to reflect the spread of ability in a typical mixed ability Greek fifth-grade class, and (d) the mix of genders to reflect the ratio of boys and girls in a typical Greek primary school. In Creswell's terms, the sample was achieved by selecting "ordinary", "typical", and "accessible" cases (Creswell & Poth, 2017).

Thus, a total of 138 students coming from eight fifth grades of five neighboring public primary schools were selected to participate in the project and, to each class, an instructional method, described in the "Procedure" section, was randomly assigned. Prior to the beginning of the project, students' parents were gathered and briefed about the project, its methodology, and objectives. Their written consent for their children's participation was obtained. Also, the fifth-grade teachers of the participating schools were briefed and they were asked to strictly follow the teaching method that was assigned to them.

The project lasted for almost a school year (from early September 2016 to mid-May 2017), as it will be further elaborated in a coming section.

Materials

There is a variety of tools for teaching programming to young students, though they prefer drag and drop applications, visual presentations, verbal explanations, discovering things on their own, and trial and error practices (Liu et al., 2011; Zhang et al., 2014). In addition, as presented in a previous section, by using game authoring tools, students form positive views regarding programming and the learning process becomes more effective (Margulieux et al., 2012).

Consequently, a number of programming environments were considered, that allow the development of games. Scratch (https://scratch.mit.edu/) is probably the most widely used application for teaching programming to students. It has been extensively studied and its usefulness is well documented (e.g., Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Flannery et al., 2013). Game Maker (http://www.yoyogames.com/gamemaker) is another application used in education with interesting results (e.g., Hernandez et al., 2010; Hoganson, 2010).

Although both applications are quite interesting per se, in the present study, an application which uses a totally different paradigm than the above was selected, namely, Microsoft's Kodu Game Lab (https://www.kodugamelab.com/). Kodu enables children and teenagers to create 3D games by offering an icon/tile-based visual language, cartoonish objects and characters, and a set of manipulation tools to build the games' landscape. The programming language is very close to the natural one. For example, it uses expressions like hear, see, bump, and combat, to trigger events and to implement interactions between objects and between objects and the user. Programs are composed of pages, which are broken down into rules, which are further divided into conditions and actions. Objects and characters run their own code, which is a simple list of conditions ("WHEN") and consequent behaviors ("DO") (Figure 1).

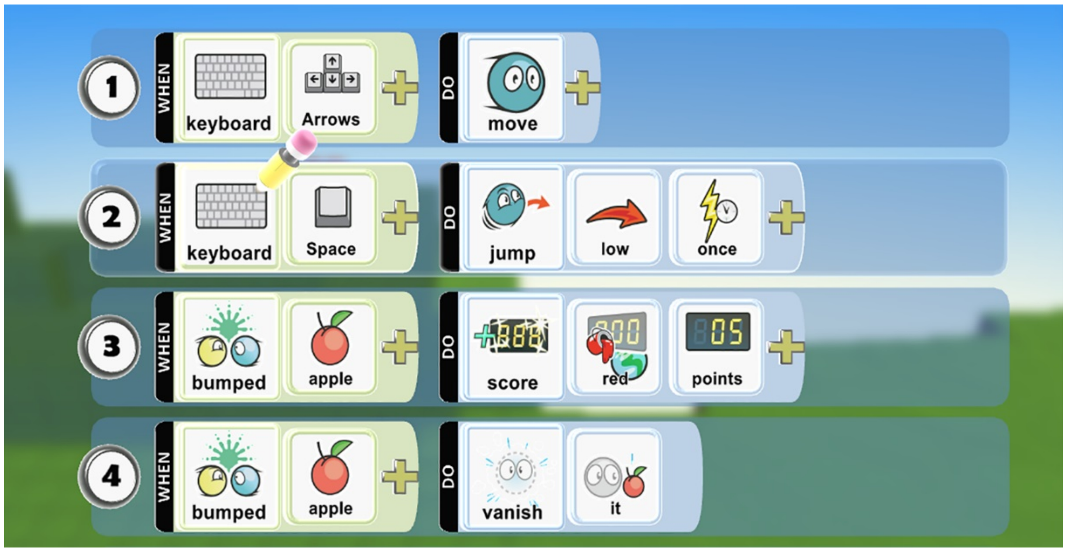

Figure 1. Sample of Kodu's programming language

While it was found that Kodu allows users to express several computer science concepts (e.g., variables, conditions, Boolean logic, and the flow of control) (Stolee & Fristoe, 2011), it has to be noted that Kodu's programming language is not a "general-purpose" one. The language is extremely simplified and game orientated; there are no loops, functions, exceptions, exception handling, or debugging (other than trial-and-error testing). Some programming concepts are implemented differently than in traditional languages. For instance, instead of using the AND operator, Kodu uses indentation to program multiple DO actions under the same WHEN condition, or for checking whether multiple WHEN conditions are true before executing an action. Creatables (objects that can be spawned multiple times during the game) can be viewed as -but are not exactly- instantiation (a programming principle difficult for beginners to grasp). Consequently, implementing some programming concepts is difficult and users are forced to find clever workarounds. For example, Kodu does not offer support for functions, although a combination of inline statements and pages can be used for that matter or a combination of switch statements, timers, and pages can be used for implementing loops.

As a result, one might argue that Kodu is suitable for the teaching of game design rather than the teaching of programming (Morris, Uppal, & Wells, 2017). Then again, the lack of programming technicalities means that students are not bound to problems related to the learning of other programming languages (e.g., syntax and use of symbols), and they can focus on the programming logic and the skills necessary to plan and implement their games. Moreover, it is Kodu's simplicity that renders it an appropriate tool for introducing novices (such as the fifth-grade students) to programming.

A book written by a team of authors (Aivalis et al., 2011) and freely available on the Internet under the Creative Commons-Noncommercial license (http://www.koduplay.gr/contents.html), was deemed as an excellent textbook for teaching programming concepts using Kodu. As it was written a few years ago and, meanwhile, quite a lot of new programming options were added to Kodu, and since it does not deal with game design concepts and principles, an additional booklet was written in order to fill these gaps. Finally, a teachers' guidance booklet was written which provided lesson plans, instructions, and more detailed examples regarding game design and the use of Kodu. It has to be noted that, prior to the beginning of the project, the teachers attended a thirty-hour intensive seminar, as all of them had never before used Kodu and were not familiar with game authoring.

PROCEDURE

A pilot study was completed with a small group of fifth-grade students (*N*=22) in spring 2016, preceding the main study, during which the research instruments were tested and the scheme of work was trialed. Students worked in pairs to create computer games using Kodu. From field notes recorded throughout the pilot study, three important issues emerged that were addressed in the main study: (a) the cognitive load (learning new concepts, new software, and new vocabulary) was a challenge for many students, (b) students had problems creating a coherent and original narrative/storyline for a game, and (c) students had difficulties in understanding what game elements to include and how to use them. Thus, it was decided to (a) significantly extend the duration of the main project, (b) place greater emphasis on the narrative aspects of game authoring, (c) provide an outline/theme for their games (e.g., the theme for their last game, in the main study, was the adventures of Ulysses, inspired from Homer's Odyssey), and (d) outline certain requirements that their games should have.

The next step was to decide on the project's duration. Following the contents' structure of both textbooks that were used in this study, fifteen teaching units were improvised and, to each, a number of two-teaching-hour sessions were allocated, as presented in Table 1. In addition, at certain milestones of the project, students were asked to develop three games of escalating complexity. As a result, the project lasted for a total of fifty sessions.

Since, in Greece, the official curriculum allocates just one hour per week for the ICT course in primary schools, arrangements were made, in collaboration with the teachers, so as to make possible to allocate two or four teaching hours per week for the project (e.g., by skipping or merging other courses and/or lessons). Coming to an agreement proved to be a difficult task; teachers voiced objections to the idea of changing their schedule. This was probably the most significant problem the project faced and the implications will be further elaborated in the "Discussion" section.

Table 1. Outline and duration of the teaching units

Unit	Duration (sessions)	Content/objectives
Let's talk about games	2	Introduction to the project, play sample games, identification of games' key components.
Game mechanics, playability, gameplay, game genres. Introduction to Kodu	2	Clarification of the terms game mechanics, playability, and gameplay. A rough categorization of games. The basics of Kodu (interface, main tools).
Game design. Terrain editing (Kodu)	2	How games are made. Review of the process and steps for developing a game. What makes a good game. Game success and game evaluation criteria. Kodu's terrain editing tools.
Storyline and interface. Objects and settings (Kodu)	2	Understand the importance of a game's storyline/narrative and of a game's interface. Kodu's objects, game and objects settings.
Events, actions, and rules	2	Understand the importance and use of events, actions, and rules in a game. Introduction to Kodu's programming language.
Controlling characters	2	The use of keyboard and mouse for controlling characters and for triggering events. Movement of characters and objects. Boolean logic (true/false operators). How to do many things at once in Kodu (AND operator).
Actions and events I	2	Clarification of the terms action and event. How to implement actions/events in Kodu. The trigger bump (collision). The action say. The NOT operator in Kodu.
Actions and events II	2	The actions/events see, hear, grab, got, and give. Sequencing events.
Shooting at things	2	How to implement a combat mechanic. Shooting and combating in Kodu.
Values and variables	4	What are variables and how they are used. How to use variables in Kodu, score mechanic and scores, health, and lives as variables.
Winning and losing	2	Understand the importance of a winning/losing mechanic. Winning or losing in Kodu.
Develop your own game I	2	Development of a simple game. Game's theme: "A shooting game." Basic game's requirements: Win/lose mechanic, combat mechanic, lives mechanic, power-ups, scores.
Creatables	2	What are creatables and how to use them in Kodu.
Functions and loops	4	What are functions and loops and how they are used. Using pages (inline & switch) in Kodu.
Time and timers in a game	2	Understanding the use of timers in a game. Time and timers in Kodu.

Unit	Duration (sessions)	Content/objectives
Develop your own game II	4	Development of a more complex game. Game's theme: "A racing game." Basic game's requirements: Win/lose mechanic, creatables, combat mechanic, lives mechanic, music, pages (inline/switch), time/timers, scores.
Use of music and sounds, and game levels	2	Understand the importance of a game's music and of the sounds assigned to characters and objects. Understand the need to use levels in a game. Use of sounds and music in Kodu. How game levels are implemented in Kodu.
Develop your own game III	10	Development of a complex game. Game's theme: "The adventures of Ulysses." Basic game's requirements: Win/lose mechanic, creatables, combat mechanic, lives mechanic, music, pages (inline/switch), power-ups, time/timers, scores, levels, environment change.

For selecting the teaching method for the experimental group, the constructionist learning/teaching principles were considered, as presented in a preceding section. In constructionist learning, students work on extended projects, learn by doing, and find by themselves the specific knowledge they need, while the teachers provide additional guidance and support (Papert, 1993). Therefore, it was decided students would work in pairs, a choice that could -potentially- bring several benefits because (a) partners are able to share ideas and complete tasks collaboratively, which is an important aspect of the constructionist theory (Kafai & Harel, 1991), (b) it promotes and sustains students' engagement, important for the completion of novel, complex, and open-ended activities (Kafai & Resnick, 1996), (c) it allows students to construct knowledge between and by themselves and provides a source of intellectual support (Vygotsky, 1978), (d) peer explanations are considered to be better matched to students' existing understandings compared to other resources (Lewis, 2011), and (e) larger groups are less flexible; students are likely to succeed in cognitive tasks when they work in pairs (Kutnick et al., 2005). In addition, the participating students had not been given, in the past, the opportunity to work on an extended project or in pairs, thus, it was considered important to check how students were going to respond to both.

The resulting scheme of work was a mix of teacher-led and pair work. The teachers made a short introduction to the programming or game design concept that students were about to study, presenting and discussing with them examples drawn from everyday life. Next, students worked in self-selected pairs, studying the relevant material from the textbooks, followed by the development of mini-games which were actually exercises for the concept they were taught (included in the first textbook). Students were free to work at their own pace, discuss, and collaborate. Also, the teachers acted as facilitators of the process; they were discussing with students by drawing their attention to important aspects of their work, and they provided help (without giving away the solution to an exercise or enforcing their views on how students should develop their games). When needed (e.g., when many students faced problems), the teachers paused students' work and provided guidelines, explanations, and examples to the whole class. At the end of a session or, in some cases, at the end of a teaching unit, students presented their work and discussed with others the problems they faced, their ideas, and/or the solutions they were able to find. The interesting ideas, good practices, and solutions that came to light during these discussions were summarized by the teachers and were handed to students, at the beginning of the next session, so that all could have a quick point of reference when needed.

For examining the impact of the above teaching method and for examining H2 and H4, two more groups of students were formed. The general idea was to compare the results of different teaching approaches. It should be noted that due to the findings of a previous study, as presented in the "Research Hypotheses" section, a "no teacher's intervention" group of students, as will be presented in a coming paragraph, was considered crucial for the study.

To the first group of students, the teachers made a short introduction, as in the experimental method, followed by examples and/or demonstrations (using the classes' video projectors) regarding game design concepts and/or how to implement certain programming concepts in Kodu. Next, students worked in pairs, by studying the relevant units in the textbooks and by solving the exercises. At this stage, the teachers' involvement was minimal; they did not intervene in students' work and they did not discuss or collaborate with them. Only when needed, they offered technical assistance or paused students' work in order to provide guidelines and examples to the whole class. At the end of each session, the teachers and/or the students presented the solutions to the exercises (mini-games) and the students were asked to check whether their answers were correct. It has to be noted that this teaching method, with the exception of students working in pairs, is the prevailing one in Greece's schools.

As for the second group, the teacher, as an element of the teaching process, was totally eliminated. The students, from the begging until the end of a session, worked in pairs, at their own pace, discussed, and collaborated with each other, but they had to rely solely on the two textbooks. There was no teachers' introduction, no form of help or guidance to students (with the exception of technical assistance), no discussions or exchange of ideas between students and teachers, and no checking whether students solved the exercises correctly. In a way, students were forced to develop, by themselves, their understanding regarding game design and programming.

As a result, three groups of students were taught the same game design and programming principles and concepts and worked in pairs for developing their own games, but their level of autonomy varied as did the teachers' role.

INSTRUMENTS AND DATA PROCESSING METHOD

The main instrument used for collecting data was students' games. While evaluation sheets/tests were an option, it was considered that they could provide only a fractional and limited picture of what students learned. On the other hand, students' games were complex artifacts representing the result of collaborative and creative work that took a significant amount of time to complete (in contrast with tests which are a "snapshot" in time). Furthermore, the analysis of games allows the assessment of students' work in terms of its quality (Biggs, 1989). In addition, this approach resonates with constructionist perspectives on assessment, which seek to evaluate learning outcomes holistically (Brennan & Resnick, 2012). The use of games as a source of data is becoming increasingly common (e.g., Brennan & Resnick, 2012; Denner et al., 2012). Then again, it is a time-consuming process and relies on the researcher having an in-depth knowledge of programming and of the software used to develop the games (Creswell, 2013).

The framework for the games' analysis was based on (a) frameworks for the analysis of commercially produced computer games (e.g., Consalvo & Dutton, 2006), (b) frameworks for analyzing computer games authored by children (e.g., Denner et al., 2012), and (c) documents defining computer programming concepts appropriate for primary and high school students (e.g., Saeli et al., 2011; Seehorn et al., 2011). On the basis of these frameworks, two sets of evaluation criteria were formed based on (a) problems related to game design features (Table 2) and (b) problems related to programming concepts (Table 3).

Two programming experts with experience in game design acted as raters. It has to be noted that they were trained prior to analyzing the games and their reliability was assessed (a) informally, during their training, (b) formally, during the pilot study, and (c) formally during the main project. An interrater reliability analysis using Cohen's kappa coefficient was performed to determine the consistency among raters. The interrater reliability was found to be $\varkappa = .87$ ($p < .001$), 95% CI (.89, .85), which was considered very good (Landis & Koch, 1977).

Table 2. Concepts used for the analysis of the games design features

Game design concepts	Errors/problems related to the:
Functionality	response to user input, interactions, gameplay, movement of characters/objects
Graphics	settings of the game, aesthetics, backgrounds, characters
Levels	use of levels
Lives and power-ups	lives mechanic, the use of power-ups (health, extra speed, etc.)
Narrative	storyline presentation and development, dialogs
Rules	obstacles, challenges, what the player is allowed to do
Score	score mechanic; the handling of scores
Sound/music	use of music and sounds; use of music for creating the game's atmosphere
Usability	game instructions, controls, and interface design
Win/lose	how a player can win or lose the game

Table 3. Categories of programming errors

Programming concept	Errors/problems related to the (or absence of):
Conditions	use of bump, eat, see, hear, say, combat, shot, and all other actions and events in conditional statements use of conflicting/duplicate/redundant conditions
Boolean logic	use of true/false/NOT operators use of the AND operator (use of indents for executing several commands at once) use of conflicting/duplicate/redundant Boolean logic statements
Functions	use of the "inline" statement for calling functions use of pages for writing functions use of conflicting/duplicate/redundant functions
Loops	use of the SWITCH statement use of pages for changing/altering the behavior of objects/characters use of conflicting/duplicate/redundant loops
Values/variables	assignment of a value in an action or expression use of conflicting/duplicate/redundant values use of variables use of conflicting/duplicate/redundant variables

For obtaining quantitative data, a quite complex procedure was followed. The first and second round of games' reviews, identified, in detail, problems related to game design features and programming concepts. Following that, the number of times a game design feature or a programming concept was used in a game and how many times it was wrongly used were calculated. This provided an initial overview of how good the games were. During the third round of reviews, a grading system, which was initially developed during the pilot study, was reconsidered, refined, and applied. It involved the allocation of penalty points, on a five-point scale, depending on the severity of a problem/error that was identified (e.g., an error in the use of a variable was considered as more severe than the use of a redundant one; an incomplete or non-functional winning/losing mechanism was considered as a less severe problem than the absence of such mechanism, etc.). It has to be noted that in all review cycles, all games were thoroughly play-tested and a printout of the program code was annotated, to identify the errors that were made.

When viewing the results of the above process, it was observed that there were cases in which non-functional or problematic games had received fewer penalty points than more complete ones. This was because in most problematic games only a handful of game design and programming concepts were used, and, therefore, the chances of making mistakes were far less than in complex ones. For instance, a non-functional game with just two actions/events had received three penalty points for problems in conditions, while an almost complete one with more than forty actions/events had received ten. As this could lead to a misinterpretation of the results, it was decided to divide the penalty points a game had received in a category by the maximum penalty points it could have received in this category. Thus, quotients close to one meant that games had many errors, while quotients close to zero indicated games with few problems. In the previous example, the first game, in conditions, had a quotient of .33 [3 penalty points/10 maximum penalty points (2 conditions X 5 penalty points maximum for each mistake)], while the second game had a quotient of .05 [10 penalty points/200 maximum penalty points (40 conditions X 5 penalty points maximum for each mistake)].

Even if this grading system provided a more accurate evaluation of games, it had a significant disadvantage; it did not provide an easy to read representation of how good a game was. Thus, the data that were obtained were reconsidered using the Structure of Learning Outcomes taxonomy (SOLO) (Biggs & Collis, 2014). The SOLO taxonomy is increasingly used to evaluate learning outcomes in computer science education (e.g., Brabrand & Dahl, 2009; Meerbaum-Salant, Armoni, Ben-Ari, 2011; Sheard et al., 2008) as well as for evaluating the learning outcomes when teaching specific programming concepts (Jimoyiannis, 2013). This taxonomy describes five levels of responses:

- Pre-structural. The responses present lack of understanding; inappropriate responses.
- Uni-structural. The responses demonstrate limited understanding; minimal relevant responses.
- Multi-structural. The responses are relevant but there may be no relationship between them; little internal coherence.
- Relational. The responses are related and appropriate and may contribute to a more coherent whole.
- Extended abstract. The responses are entirely appropriate and exceed expectations.

Accordingly, the SOLO taxonomy was adapted so that it could be used to evaluate the programming constructs and the game design concepts (Table 4). Each level of the SOLO taxonomy was divided into 10 sub-levels, to give greater accuracy in the evaluation and for reducing ambiguity (Chan, Tsui, Chan, & Hong, 2002). A score corresponding to a SOLO level was given for each game's feature (game design and programming) and an overall average was calculated. Thus, games with many errors and problems received a low SOLO score, while games with few problems were positioned high in the SOLO taxonomy.

Table 4. The adapted SOLO taxonomy

SOLO level (average score)	Game design	Programming
Pre-structural (0-10) No functionality, no user interaction, graphics only	The game is not playable, game assets do exist but are not organized or developed, irrelevant information, no or few interactions, only one level, poor game environment, no score mechanic	Many programming errors, no understanding of programming concepts, limited use or no logical sequence of events/ actions
Uni-structural (11-20) Some functionality and interaction, the game needs further development	The game is mostly unplayable, few game interactions, poor game environment but usable, levels do exist but are incomplete or progression through them is impossible, no functioning score mechanic	Limited understanding of programming concepts, events/ actions contain errors, the game partially works with significant problems, no use of important programming concepts
Multi-structural (21-30) More functionality, the game is playable but incomplete	Several aspects of the game are present but not all function correctly, game components are not connected, some parts of the game function correctly, the game environment is usable but requires further development, a score mechanic is present but does not function correctly	The game works with some problems, several objects are included, more confident use of events/actions, some of which work, conditional statements, variables, and other programming concepts are used, but partially correctly
Relational (31-40) The game is playable but certain details are missing	A playable game, most elements function correctly, the player can progress through levels, the game environment is reasonably well executed and acceptable	The game works with no significant problems, enough events/actions are used to control objects and operations in the game, programming concepts are used effectively most of the time
Extended abstract (41-50) A fully operational game	A complete, playable game with sufficient interactions to make it engaging, all elements function correctly, levels present and accessible, a clear win/lose state, the game environment is well-executed, the game is a coherent whole	No programming errors

One of the purposes of the study was to explore and assess students' perceptions about the process and outcomes of their learning during their game authoring activities. Thus, the second instrument used was a short questionnaire administered to students at the end of the project. It consisted of twenty 5-point Likert-type questions (worded "Strongly Agree", "Agree", "Neutral", "Disagree" and "Strongly Disagree") and eleven open-ended questions. Scores were obtained by allocating numerical values to responses: "Strongly Agree" scored 5, "Agree" scored 4; "Neutral" scored 3; "Disagree" scored 2 and "Strongly Disagree" scored 1. Finally, although it was beyond the scope of the present study to explore the teachers' perspective, they were asked to keep a log of incidents that took place during their teaching and to record their views on how well the whole process worked.

RESULTS

Since each pair of students developed three games (Game1 = simple game, Game2 = more complex game, and Game3 = a complex one), a total of 207 games were analyzed, developed by 138 students (62 boys and 76 girls) divided into three equal groups (Group1 = control group/no teachers' guidance, Group2 = control group/limited teachers' guidance, and Group3 = experimental group/combination of teacher-led and pair work). For examining H1 (students can understand and effectively use programming concepts and practices when they design digital games and their games are playable and functional) the SOLO taxonomy scores that were obtained from the evaluation of games were used. Mean SOLO scores per group of participants and per game are presented in Table 5.

Table 5. SOLO taxonomy. Means and standard deviations per game, per game's features, and per group of participants

Groups (games)	Mean SOLO scores								
	Game1			Game2			Game3		
	Average	Game design	Programming	Average	Game design	Programming	Average	Game design	Programming
Group1 *N* = 23	13.57 (3.23)	13.65 (3.74)	13.48 (3.33)	25.91 (5.27)	26.09 (5.97)	25.65 (5.00)	37.04 (4.51)	36.74 (4.78)	37.35 (4.98)
Group2 *N* = 23	16.96 (3.72)	17.09 (4.21)	16.83 (3.75)	30.30 (3.80)	30.00 (4.26)	30.61 (4.18)	38.83 (3.68)	37.87 (4.78)	39.78 (3.34)
Group3 *N* = 23	20.00 (3.28)	20.30 (2.91)	19.70 (4.06)	32.96 (4.23)	33.13 (5.12)	32.78 (4.33)	39.39 (2.73)	39.43 (3.99)	39.35 (2.72)

Notes. Standard deviations are reported in parentheses. The SOLO levels of Game1 are lower because all programming concepts were not taught during the time they were developed.

Additionally, in order to gain a deeper understanding of the programming errors students made and to conclude which programming concepts they were able to learn, the number of errors and the number of times a programming concept was used were calculated (Table 6). It has to be noted that this table presents the total number of errors/problems regardless of their severity.

Table 6. Programming concepts and errors per game and per group of participants

Concept	Programming concepts scores								
	Game1			Game2			Game3		
	Group1 *N* = 23	Group2 *N* = 23	Group3 *N* = 23	Group1 *N* = 23	Group2 *N* = 23	Group3 *N* = 23	Group1 *N* = 23	Group2 *N* = 23	Group3 *N* = 23
Boolean logic	12/30 (40%)	15/42 (36%)	12/51 (24%)	21/85 (25%)	18/112 (16%)	19/125 (15%)	20/158 (13%)	18/149 (12%)	24/161 (15%)
conditions	36/354 (10%)	45/440 (10%)	39/483 (8%)	85/769 (11%)	91/945 (9%)	98/1002 (10%)	99/1189 (8%)	89/1156 (8%)	96/1222 (8%)
functions	NA	NA	NA	2/4 (50%)	3/16 (19%)	3/18 (17%)	5/39 (13%)	7/48 (15%)	4/45 (9%)
loops	NA	NA	NA	45/98 (46%)	48/176 (27%)	38/202 (19%)	64/374 (17%)	72/442 (16%)	78/534 (15%)
Variables/values	38/112 (34%)	45/244 (18%)	51/294 (17%)	39/285 (14%)	25/381 (7%)	29/404 (7%)	24/395 (6%)	32/422 (8%)	19/439 (4%)

Notes. NA = Not applicable, A/B = number of errors/ number of times, percentages are reported in parentheses

Taken together, the results presented in Tables 5 and 6 confirm H1 because:

- All groups were not able to develop good games at their first attempt (Game1), as most of them belonged to the uni-structural category. This result was, more or less, expected, because, at the time these games were developed, important game design and programming concepts were not yet taught to them.

- The situation changed significantly in the following game (Game2). Students in all groups were able to present games that belonged to the -upper limit of- multi-structural and to relational categories.
- An even further improvement was noticed in students' final games, with the majority of them belonging to the -upper limit of- relational category. This finding is important considering the fact that the games were developed by fifth-grade students with no previous programming or game design experience.
- Similar were the results regarding the programming concepts present in students' games and the errors in their use.
- The most commonly used programming concept was conditions. Also, the errors in this programming concept were few.
- Variables and values were the second most commonly used programming concepts.
- The least commonly used concept was functions. On the other hand, loops were used several times more than functions and their error rates were not high. At Kodu, loops and functions work similarly (their only difference is the switch or inline commands for calling them), therefore, it is quite probable that students used loops as functions and vice versa.
- Functions, loops, and Boolean logic seemed to be the cause of most problems in games 1 and 2. These problems were eased in Game3.
- In Game3 and in all groups, the errors in all programming concepts were between 4 to 17%, meaning that in (almost) nine out of ten times, a concept was used correctly.

For examining H2 (the learning outcomes depend on the teaching approach that is used), the SOLO taxonomy scores were once again used, but this time for conducting one-way ANOVA tests. This was done in order to determine whether there were any statistically significant differences between the three groups of students. Prior to conducting these tests, it was checked whether the assumptions for ANOVA testing were violated. In two cases, minor issues regarding the normality of the data were found. Like other parametric tests, the analysis of variance assumes that the data fit the normal distribution. On the other hand, literature suggests that ANOVA is quite robust to moderate deviations from normality (the absolute values of the skewness and kurtosis for the data not to be more than double their respective standard errors) and the false positive rate is not affected very much by this violation (Glass, Peckham, & Sanders, 1972; Lix, Keselman, & Keselma, 1996). In the above cases, the violations were found to be minor rather than moderate, thus, they were considered as acceptable deviations from the assumptions for ANOVA testing. As all the other assumptions were met (equal number of games in all groups, no outliers, and no violation of the homogeneity of variance), the analysis was conducted (Table 7).

Table 7. One-way ANOVA results

Game	**SOLO scores**	**Result**
Game1	Average	*F*(2, 66) = 20.41, *p* < .001
	Game design	*F*(2, 66) = 19.01, *p* < .001
	Programming	*F*(2, 66) = 16.05, *p* < .001
Game2	Average	*F*(2, 66) = 14.54, *p* < .001
	Game design	*F*(2, 66) = 10.74, *p* < .001
	Programming	*F*(2, 66) = 15.08, *p* < .001
Game3	Average	*F*(2, 66) = 2.51, *p* = .089, NS
	Game design	*F*(2, 66) = 2.05, *p* = .136, NS
	Programming	*F*(2, 66) = 2.68, *p* = .076, NS

Note. NS = not statistically significant difference

Post-hoc comparisons were conducted using the Tuckey HSD test on all possible pairwise contrasts, except for the ones where no statistically significant differences were noted. The results were:

- Game1, average SOLO scores. Group3 (*M* = 20.00, *SD* = 3.28) outperformed Group2 (*M* = 16.96, *SD* = 3.72) (p = .014) and Group1 (*M* = 13.57, SD = 3.23) (*p* = .010 and *p* < .001 respectively). Group2 outperformed Group1 (*p* = .006)
- Game1, game design SOLO scores. Group3 (*M* = 20.30, *SD* = 2.91) outperformed Group2 (*M* = 17.09, *SD* = 4.21) (*p* = .010) and Group1 (*M* = 13.65, *SD* = 3.74) (*p* = .011 and *p* < .001 respectively). Group2 outperformed Group1 (*p* = .004)
- Game1, programming concepts SOLO scores. Group3 (*M* = 19.70, *SD* = 4.06) outperformed Group2 (*M* = 16.83, *SD* = 3.75) (*p* = .010) and Group1 (*M* = 13.48, *SD* = 3.33) (*p* = .030 and *p* < .001 respectively). Group2 outperformed Group1 (*p* = .009)
- Game2, average SOLO scores. Group3 (*M* = 32.96, *SD* = 4.23) outperformed Group1 (*M* = 25.91, *SD* = 5.27) (*p* < .001) but not Group2 (*M* = 30.30, *SD* = 3.80) (*p* = .118). Group2 outperformed Group1 (*p* = .004)
- Game2, game design SOLO scores. Group3 (*M* = 33.13, *SD* = 5.12) outperformed Group1 (*M* = 26.09, *SD* = 5.97) (*p* < .001) but not Group2 (*M* = 30.00, *SD* = 4.26) (*p* = .107). Group2 outperformed Group1 (*p* = .033)
- Game2, programming concepts SOLO scores. Group3 (*M* = 32.78, *SD* = 4.33) outperformed Group1 (*M* = 25.65, *SD* = 5.00) (*p* < .001) but not Group2 (*M* = 30.61, *SD* = 4.18) (*p* = .239). Group2 outperformed Group1 (*p* = .001)

The above results suggest that:

- In Game1, Group3 outperformed both groups 1 and 2 in all cases (average, game and programming concepts). Also, Group2 outperformed Group1 in all cases.
- In Game2, Group3 outperformed Group1 but not Group2 (in all cases). Also, Group2 outperformed Group1 (in all cases).
- In Game3, there were no statistically significant differences between all groups; all groups had the same results.

The data analysis, as presented above, partially confirms H2, depending on which game is taken into consideration. Indeed, in students' first game, the situation is quite clear; the combination of teacher-led and pair work yielded better results compared to the other methods. However, in the second game, this method and the limited teachers' guidance method yielded equally good results, and, in any case, better ones than the no teachers' guidance method. In the third game, the picture became blurred, as there were no differences between the three teaching methods. This finding will be further elaborated in the "Discussion" section.

Coming to the questionnaire that was given to students, its purpose was to record their experiences and views regarding the process of developing their games. It has to be noted that one-way ANOVA tests were run for each question, in order to determine if there were any differences in the opinions of the three groups of students and none were found. This lead to the rejection of H4 (students' views regarding programming and game development depend on the teaching approach that is used). Since there were no differences, Table 8 presents the averages from the responses of all groups.

Students' strong positive attitude towards the project was evident in most of their responses (see questions three through eight, and twelve through twenty), thus, H3 (students form positive attitudes and perceptions regarding their involvement in game authoring activities) was confirmed. Collaboration also seems to have worked well (*M* = 4.31, *SD* = .55) and students acknowledged how important their partner was in the development of their games (*M* = 4.50, *SD* = .50). Quite interestingly, students were cautious regarding the self-assessment of their games (see questions nine through eleven). It seems that they were moderately pleased with their games (*M* = 3.25, *SD* = .98), and they did not consider them very interesting and fun to play (*M* = 3.28, *SD* = 1.50). Also, it seems that they were not so pleased with the end result, compared to how they initially planned or thought that their games might be (*M* = 3.14, *SD* = 1.02).

Table 8. Students' questionnaire

	Question	Result (mean)
1.	The collaboration with my fellow student was very good.	4.31 (.55)
2.	I feel that working as a pair helped me to learn about making a computer game.	4.50 (.50)
3.	I think that making a game is a boring activity.*	4.42 (.48)
4.	I think that making a game is an enjoyable activity.	4.37 (.54)
5.	I think that making a game is a useful activity.	4.44 (.37)
6.	I think that programming is an interesting activity.	4.21 (.60)
7.	I think that programming is not an enjoyable activity.*	4.15 (.61)
8.	I think that programming is a useful activity.	4.11 (.52)
9.	I managed to develop my last game the way I initially conceived it.	3.14 (1.02)
10.	My last game was a good one (in terms of its complexity, gameplay, scenario, etc.).	3.25 (.98)
11.	I think that my last game was interesting and fun for others to play.	3.28 (1.50)
12.	Working with Kodu was difficult.*	3.78 (1.03)
13.	Learning how to program was easy.	3.60 (.96)
14.	I do not feel that I have learned new skills*.	4.26 (.44)
15.	I was eager each week to conduct the project's lessons.	4.30 (.51)
16.	I found the whole course (referring to the project) very interesting.	4.14 (.29)
17.	I did not like the course at all.*	4.25 (.48)
18.	I considered the course very important for me.	4.27 (.49)
19.	I am interested in developing other games in the future if I am given the chance do so.	4.35 (.55)
20.	I am interested in learning more about programming.	4.14 (.70)

Notes. * indicates a question for which its scoring was reversed; standard deviations are reported in parentheses

The open-ended questions were about game authoring, Kodu, and programming:

- Game authoring. Students found easy to understand almost all game design concepts; the use of lives and power-ups (*N* = 102), the importance of score mechanism (*N* = 100), music (*N* = 95), win/lose mechanism (*N* = 92), and the reason for using levels (*N* = 85). The majority of them also stated that no game design concept was difficult for them to grasp (*N* = 113).
- Kodu. Understanding how to use Kodu (in general) was easy for students (*N* = 122). On the other hand, a number of Kodu's features were a cause of some trouble; landscaping/creating the game's environment (*N* = 66), the use of settings (*N* = 32), and the handling of the camera (*N* = 25). Students enjoyed how easy they could "make things happen" (*N* = 101), Kodu's cartoonish characters and objects (*N* = 97), and their "funny" animations and sounds (*N* = 92). They disliked that "they could not do anything they wanted" (*N* = 87)

and that the trees were flickering (a problem with some graphics cards that was solved by a later update (*N* = 19).

- Programming. Students stated that conditions were the easiest programming concept to learn (*N* = 113), followed by variables (*N* = 92). On the other hand, functions and loops were the hardest (*N* = 54 and *N* = 50 respectively). Very few indicated Boolean logic as either an easy (*N* = 15) or a difficult concept (*N* = 18). In addition, timers and time were a problem for them (*N* = 48), probably because of the way that time is handled in Kodu. They enjoyed learning a new concept and applying it in order to make their games more interesting (*N* = 85). As expected, they expressed frustration when "something wasn't working" (*N* = 133, which represents the sum of diverse answers in this category, ranging from a specific programming concept to mistakes that students could not find).
- The last question was about which new skills students think that they have learned. Two were the most common responses (and almost the only ones) – "programming" (*N* = 130) and "making games" (*N* = 128) –, which, although they are valid, they are very vague statements. Very few students provided more detailed answers, and, out of them, the most common ones were about gameplay (*N* = 17) (e.g., "Because now I know what things make them interesting and fun to play, I know what to look for or expect when playing them.").

The teachers' logs confirmed students' enthusiasm and interest for the project and this applied to all groups. The most frequent problems that were reported were related to how focused students were at their tasks, collaboration problems (at the beginning of the project), and lack of discipline, but none were considered as major ones. Also, none of the participating teachers reported significant problems related to how well they were able to implement the scheme of work.

DISCUSSION

Game making within classrooms and within the context of ICT education at the primary level was the focus of relatively few studies (e.g., Wilson et al., 2012). Moreover, the literature regarding what kind of knowledge students learn when they develop digital games using a programming language is also limited. The present study contributes to the knowledge base of these still inadequately documented yet important areas by designing and implementing a project which had as a target group, 10-11-year-old students. Importantly, contrary to the existing body of the literature, which involved intensive and out-of-school research projects, the research focused on mainstream primary school settings and lasted for a good part of a school year. Due to the duration of the project, it can be argued that it was not just a few teaching interventions but that it introduced a new course which significantly deviated from the ICT course usually taught to Greece's primary schools.

While there are suggestions on how to deal with the errors students make with visual programming tools (e.g., Doran, Boyce, Finkelstein, & Barnes, 2012), there is a limited number of studies that identified the programming concepts they used or the errors they made (e.g., Good, Howland, & Nicholson, 2010). This study contributes to the relevant literature because it classified the programming constructs present in students' games and checked whether they were used correctly. On the other hand, the results have to be viewed with caution, due to Kodu's discrete features as presented in the "Materials" section. Having that in mind, it was found that the most commonly used programming concept was conditions, followed by variables and loops (but two to three times less than conditions). Boolean logic and functions were the least used concepts (eight and twenty-five times less than conditions respectively) (see Table 6, Game3). As for the programming errors students made, the most problematic concepts proved to be Boolean logic and loops, closely followed by functions. The least problematic concepts were conditions and variables. Conditions were expected to be used plenteously because Kodu games rely heavily on their use. Also, since the use of functions is not mandatory in Kodu, and because functions and loops have a similar use, it can be argued that the scarcity of functions' use is circumstantial, given that loops were used quite extensively. Most importantly, (a) the number of programming concepts that were used was increasing in each game, while (b) the errors were decreasing and, in Game3, ranged between 4 to 17% of the total times a concept was used.

Pea and Kurland (1984), described four levels of students' programming skills: (a) program user, (b) code generator, (c) program generator, and (d) software developer. As code generators, students know the syntax and semantics of the more common commands, can explain what each line of accomplishes, can locate bugs, and can write simple programs. On the other hand, the code is not optimized and the use of subroutines is limited, if non-existent. As program generators, students have mastered the basic commands, know the use of sequences of commands, and can write quite lengthy programs (but they tend not to be user-friendly). The results support the view that students certainly reached the third level, while they demonstrated elements of the fourth. Given that the students did not have any prior programming knowledge, this outcome can be considered as satisfactory.

As for the quality of students' games, in terms of their design and gameplay, and taking into account, once again, the limitations of Kodu, the results were also satisfactory. Thought in Game1 students were able to come up with games that belonged to the uni-structural category, therefore, not good ones, it is quite safe to assume that this result does not represent the full potential of students since (a) the development of Game1 took place at the early stages of the project, thus certain key programming concepts were not yet presented to students, (b) it was the first time students had to develop a game, and (c) the time that was available to them, to develop their games, was -by far- less than in Games 2 and 3. The situation changed dramatically in Game2, where groups 2 and 3 were able to present games that were mostly playable (upper limit of multi-structural category). In Game3 all groups developed playable games with most of their elements functioning correctly (upper limit of relational category).

Taken together the results in programming errors and game design, suggest that, while making their games, the participating students, in all groups, were able to learn and use certain programming concepts and game design practices. This brings the discussion to the results regarding the comparison between the instructional methods and to the project's theoretical framework. Although all three methods, as presented in the "Procedure" section, were based on constructionism's principles, in Group1 the students received no instruction by their teachers and had only the textbooks as a point of reference, in Group2 the teachers' role was limited, while in Group3 the teachers acted as facilitators of the learning process, by guiding students without enforcing their views or by giving away solutions to the problems that students faced. In Game1 the results indicated that the students in Group3 fared better compared to the other groups; in Game2, groups 2 and 3 had equally good results while both surpassed Group1 and, finally, in Game3 all groups had, statistically speaking, the same results.

The majority of the literature surrounding the use of visual programming languages emphasizes the need for direct instruction and formal introduction and demonstration of programming concepts before students are able to effectively implement them (Beynon & Harfield, 2010; Denner et al., 2012; Schelhowe, 2010). In the same line of thinking, there is the notion that programming concepts, such as conditions, loops, and variables, can be learned when students are taught them while they are involved in projects that use these concepts (e.g., Denner et al., 2012; Meerbaum-Salant, Armoni, & Ben-Ari, 2011). Other studies suggested that some concepts can be learned without instruction and some need a formal introduction (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Schelhowe, 2010). Finally, there is the view that computer game authoring imposes an additional obstacle since it does not allow the understanding of more complex concepts, at least without explicit teaching (Denner et al., 2012), because students are programming almost without knowing it (Good, 2011).

If only the results in Game1 and Game2 are taken into consideration, one can assume that they confirm the above views. Indeed, in both games, the groups of students who received some form of teacher's guidance and instruction were able to use more programming concepts with fewer mistakes and their games were better than those of Group1. Alas, the results in Game3 overturned this conviction; all groups were able to come up with good games, using a substantial amount of programming commands and with relatively few mistakes. In this respect, it can be argued that even without systematic instruction, students can, eventually, learn and use programming concepts (at least the ones that were involved in this project) and can successfully implement them for developing functional digital games. But there is one condition that has to be met and that is to provide students with enough time for practicing. In a way, the views of Harel (1991) and Kafai (2012), who supported the implementation of extended projects for teaching programming, are substantiated by the findings of the present study. Also, the fact that in Game3 students in groups 2 and 3 were not able to maintain their statistically significant difference from Group1, may imply that all groups reached a point, a plateau, where further improvement was not possible/negligible/very hard to achieve. The only difference is that Group3 was the first to reach this plateau, while Group1 was the last and this can be attributed to the different teaching methods.

The learning theory that framed the project was constructionism since it was theorized that it is suitable for teaching programming and how to design and make computer games. The results are in line with earlier research on students making games and learning programming, which embraced constructionism as a teaching approach (e.g., Harel, 1991; Kafai, 1996), but they are in contrast with other, more recent views on this matter. This dichotomy emerges from the core philosophy of constructionism. Besides the analytical learning style, Papert (1993) coined the term "bricolage" to describe a learning style in which one learns and solves problems by exploring alternative solutions, testing, and "learning by doing". However, researchers suggested that bricolage is not well suited for students still at their early stages of learning how to program (Guzdial, 2009) or that it might be suitable for some learners but not all (Stiller, 2009). Others suggested that such approaches are altogether inappropriate for learning programming concepts. They argued that the semantics and syntax of programming languages are non-negotiable and, thus, not well aligned to constructionist approaches (Beynon, 2009). They also argued that bricolage can lead to endless debugging and is therefore not an effective methodology and epistemology (Ben-Ari, 2001). The findings of this study, in terms of the learning outcomes that were achieved (especially those of Group1) and of students' positive attitudes and increased motivation, do not give support to such views.

Several of the questionnaire's items tried to record students' views regarding the project and high values in the affective domain (motivation and enjoyment) were observed in all groups (see Table 8, questions 3 through 8, 12 through 20, and the open-ended questions). Thus, it can be argued that one of the positive outcomes of the project was that students felt motivated and enjoyed making computer games. Enjoyment when making digital games is a common finding in many studies (e.g., Baytak, Land, & Smith, 2011; Ke, 2014; Li, 2010; Navarrete & Minnigerode, 2013; Yang & Chang, 2013). This finding is also in line with previous research, which widely reports that young individuals consider game authoring motivating and that this leads to positive attitudes to learning (e.g., Fowler & Cusack, 2011; Hwang et al., 2014; Ke, 2014; Li, 2010; Robertson, 2013).

As for why students felt motivated and enjoyed making games, several factors might have contributed. Kodu's colorful and easy to understand programming environment is undeniably one of them (Pilot, 2009). Also, the mode of work was, by itself, playful and certainly quite different than the -more formal kind of work students are used to. Work became a process of experimenting, creating, making mistakes, and playing. Exploratory learning, as well as the freedom to make mistakes, are two of constructionism's cornerstones, and Papert (1980) strongly supported that this kind of learning has to be given more status in schools. In addition, there were no grades involved, since the project was an unofficial course, and this may have been a contributory factor in some students' enjoyment of the game authoring activity. Closely related to the motivational affordances of game authoring are the feelings of achievement and fulfillment when students create an authentic product (Sanford & Madill, 2007). Thus, motivation might have been the result of students feeling proud and valuing their games because they had created something that had personal value to them. Also, making computer games is a form of creative expression, which is closely related to enjoyment (Buckingham &

Burn, 2007). The fact that the participating students were asked for the first time to be creators of digital artifacts probably boosted the impact of all the above.

Though students agreed that they enjoyed the experience, they also acknowledged that they faced difficulties. They expressed frustration when their games were not functioning properly, when they made mistakes, and when they could not implement a programming concept (see open-ended questions for programming and for Kodu). Fun co-existing with difficulties when making games is supported in the literature (e.g. Kafai, 1996; Li, 2010; McInerney, 2010; Navarrete & Minnegerode, 2013). In the words of Papert (1996), students' experience was "hard fun"; students prefer challenging activities over non-challenging ones, on condition that they are interesting and personally relevant.

The results in questions nine through eleven revealed an interesting aspect of students' views. They were cautious regarding the self-assessment of their games; they were not so pleased with the end result, they did not consider them very interesting and fun to play, and they stated that they were not able to implement all the features they initially planned. Two contradictory assumptions can interpret this finding. One is that students were over-enthusiastic, overestimated their skills, they made ambitious plans and did not take into account the limitations of Kodu. Not being able to make the "amazing" games they thought that they were able of developing led to dissatisfaction which was expressed in their responses to the relevant questions. Another interpretation is that students were aware of the limitations of their work and able to assess it on the basis of what they have learned during the course of the project. In this respect, their responses can be viewed as an indirect indication of the project's success. Since in the present study this issue was not studied in-depth, it is not clear which of the two assumptions is valid.

Implications for Practice

Whilst the literature supports the notion of students learning to program through game development, less attention is paid to the difficulties such projects face. The study identified four problems which arose in practice, as presented in the following paragraphs.

The cause of the project's most significant problem was its duration. As already mentioned, the teachers' objections to the idea of merging or skipping other lessons in order to fit one or two two-hour sessions per week (fifty in total) were strong and certainly justified; the insertion of a new course to an already saturated timetable can cause significant trouble. Indeed, the lessons' timetables of the participating schools were extensively rescheduled to suit the needs of the study. Although this might be -up to a degree- acceptable for conducting research, it is questionable whether it can be applied in larger scale projects or as part of the official ICT course. Thus, one has to come up with a more realistic duration and, at the same time, take into account the need for an effective instructional technique and a well-developed pedagogy (Egenfeldt-Nielsen, 2006; Ulicsak & Williamson, 2011) that, on the basis of the study's findings and as suggested by others (e.g., Harel, 1991, Kafai, 2012; Ke, 2014), will be based on constructionist principles. By examining the project's teaching units (see Table 1) it is observed that sixteen units were allocated for the development of students' games. Some of them can be removed (e.g., the development of Game1) and others can have a shorter duration (e.g., the development of games 2 and 3). The same applies to other teaching units (e.g., Values and variables, Functions and loops, etc.). As a result, a duration of twenty-five to thirty two-hour sessions is probably feasible.

The problem, at least for the Greek primary school curriculum, is that, instead of increasing the teaching hours of the ICT courses, recently (in 2016), it was decided to decrease them from two to one per week, as part of an overall reform of the curriculum in the first two levels of education. Thus, it is up to the education administrators and policymakers to reconsider this decision. Towards this end, the present study provided some useful insights on how the ICT curriculum can be reformed since (a) units of work to implement aspects of the ICT curriculum, especially for students who have no prior knowledge of programming were developed and (b) on the basis of the results, it suggests that game authoring is suitable for this kind of work. On the other hand, because the ICT curriculum covers many and diverse topics in addition to programming, it would be wise to -systematically- familiarize students with the use of computers at an earlier stage (probably at the beginning of primary education), so as to give space for the teaching of programming at a later stage.

Another issue that has to be discussed is whether Kodu is an appropriate tool for teaching programming. It is true that a number of programming concepts can be easily expressed as Stolee and Fristoe (2011) pointed out and was confirmed by the findings of the present study. It is also true that, on the basis of the study's findings, students mastered the use of Kodu, enjoyed the process, and were highly motivated and engaged, confirming previous research which also noted increased levels of enjoyment and motivation even of previously disengaged pupils when using Kodu (Pilot, 2009). On the other hand, Kodu's programming language significantly deviates from other traditional languages in terms of how some concepts are implemented. This might cause confusion to students when, at a later stage, start to learn one of the mainstream programming languages. In this respect, Kodu might be more suitable for teaching game design as was supported by Morris et al. (2017). Then again, Scratch, which is extensively studied and used for teaching programming to young students, also had its share of criticism for the programming paradigm it uses (e.g., Meerbaum-Salant, Armoni, & Ben-Ari, 2011, 2013). It seems that the question is not an easy one to answer and depends on what the objectives are. In this study, the objectives were to teach the basics of programming, motivate students, and stimulate their interest in programming. In this respect, Kodu proved to be a suitable tool.

Another concern was the scheme of work and the pedagogy of programming, especially taking into account the fact that, in this project, the classes' teachers were the ones that conducted the lessons and not the ICT teachers as it is the norm. This was done on purpose, because ICT teachers teach in many classes during the day (and sometimes in different schools) and, consequently, it would have been impossible to extensively rework their timetables in order to fit the project's needs. Thus, the training needs of teachers who do not have a computing background had to be taken into account, and, indeed, this was done by conducting a thirty-hour intensive seminar prior to the beginning of the project. On the basis of the results, this research illustrates that introducing basic programming concepts using Kodu is a viable approach for -properly trained- teachers and pupils who have little -or none- prior knowledge in this field.

Finally, for analyzing and evaluating the games students made, a quite complicated system was devised based on a modified version of the SOLO taxonomy (Biggs & Collis, 2014), which incorporated elements of programming and game design. This methodology for analyzing computer games made by students is an area not widely covered in the research literature. However, such a detailed assessment would be time-consuming for teachers in mainstream settings. On the other hand, some of the existing assessment frameworks are less useful when assessing extended projects as they ignore the design process and the development of skills which are important features of the constructionist learning theory (e.g., Dorling & Walker, 2014).

CONCLUSION

This study explored the introduction of a unit of work in which fifth-grade primary school students developed computer games as part of their ICT curriculum. The findings demonstrated that, as they made their games, students learned some basic game design and programming concepts, and became producers of software for the first time. All in all, the results can be considered as satisfactory and also thought-provoking. Indeed, the most significant finding was that the impact of the teaching method faded over time. On the basis of this finding, it can be argued that time for practice is more important than the method used for teaching young students the basics of programming. Then again, there are limitations to this study that merit further discussion. The instrument that was used for evaluating the games certainly needs refinement and further development. In addition, one cannot be certain whether the questionnaire accurately recorded students' views. The study's sample, although more than sufficient for statistical analysis, could have been larger and the participating students came from one city in Greece. Therefore, there are some reservations regarding the generalizability of the results. A number of the teaching units have to be reconsidered both in terms of their duration and content. Finally, since the focus was on students' performance, no data were collected on how well teachers were able to implement each teaching method.

Further studies are needed in order to identify differences or similarities to the findings of the present study. For example, other game-like programming environments can be examined. The target group can be younger and/or older students, so as to determine what programming concepts are suitable for teaching at each age and also to establish the appropriate teaching method. Future studies can use additional research tools, such as observations and interviews with students and teachers that will allow an in-depth understanding of how they view the idea of designing games for learning/teaching programming. Research into how to assess students' understanding of programming concepts when authoring games and their achievements in game design are fruitful topics for further investigation. In this respect, the study of students' programming misconceptions is an interesting topic. Finally, it would be interesting to conduct research by using other devices (such as tablets) and compare the results.

Nevertheless, taking into account all limitations and in conclusion, the experimental data that were obtained reinforced the view that game authoring is an interesting alternative method for teaching programming concepts to students. Considering the learning outcomes together with students' perceptions, motivation, and interest, it is hoped that this research will make a useful contribution to the ongoing debate surrounding the pedagogy of computer game authoring in mainstream primary school settings.

REFERENCES

Aivalis, S., Aivatis, Ch., Alexiou, V, Vouronikou, A., Gaki, S., Giaka, Ch., . . . Chalkias, C. (2011). *Δημιουργώ παιχνίδια με το MS Kodu* [I develop games with MS Kodu]. University of Thessaly.

Akcaoglu, M., & Koehler, M. J. (2014). Cognitive outcomes from the Game-Design and Learning (GDL) after-school program. *Computers & Education, 75*, 72-81. https://doi.org/10.1016/j.compedu.2014.02.003

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "real" programming. *ACM Transactions on Computing Education (TOCE), 14*(4), 25. https://doi.org/10.1145/2677087

Baytak, A., & Land, S. M. (2010). A case study of educational game design by kids and for kids. *Procedia-Social and Behavioral Sciences, 2*(2), 5242-5246. https://doi.org/10.1016/j.sbspro.2010.03.853

Baytak, A., Land, S., Smith, B. & Park, S. (2008). An exploratory study of kids as educational game designers. In: M. Simonson (Ed.) *Proceedings of the 31st Annual Convention of the Association for Educational Communications and Technology* (pp. 39-47). Bloomington, USA: AECT Publications.

Baytak, A., Land, S. M., & Smith, B. K. (2011). Children as educational computer game designers: An exploratory study. *The Turkish Online Journal of Educational Technology, 10*(4), 84-92.

Beavis, C., O'Mara, J., & McNeice, L. (Eds.). (2012). *Digital games: Literacy in action*. Wakefield Press.

Becta. (2003). *Computer games in education project report.* British Educational Communications and Technology Agency Archive.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45-73.

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. *Proceedings of the First International Workshop on Computing Education Research*, 81-86. https://doi.org/10.1145/1089786.1089794

Beynon, M. (2009). Constructivist computer science education reconstructed. *Innovation in Teaching and Learning in Information and Computer Sciences, 8*(2), 73-90. https://doi.org/10.11120/ital.2009.08020073

Beynon, M., & Harfield, A. (2010). *Constructionism through construal by computer.* Paper presented at Constructionism 2010. Paris, France

Biggs, J. (1989). Towards a model of school-based curriculum development and assessment using the SOLO taxonomy. *Australian Journal of Education, 33*(2), 151-63. https://doi.org/10.1177/168781408903300205

Biggs, J. B., & Collis, K. F. (2014). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome).* Academic Press.

Brabrand, C., & Dahl, B. (2009). Analyzing CS competencies using the SOLO taxonomy. *ACM SIGCSE Bulletin*, *41*(3), 1-1. https://doi.org/10.1145/1595496.1562879

Brennan, K. (2013). Learning computing through creating and connecting. *Computer*, *46*(9), 52-59. https://doi.org/10.1109/MC.2013.229

Brennan, K., & Resnick, M. (2012). *Using artifact-based interviews to study the development of computational thinking in interactive media design.* Annual American Educational Research Association meeting. Vancouver, BC, Canada.

Buckingham, D., & Burn, A. (2007). Game literacy in theory and practice. *Journal of Educational Multimedia and Hypermedia, 16*(3), 323.

Bulfin, S., Henderson, M., & Johnson, N. (2013). Examining the use of theory within educational technology and media research. *Learning, Media and Technology, 38*(3), 337-344. https://doi.org/10.1080/17439884.2013.790315

Caperton, I. H. (2012). Toward a theory of game-media literacy: Playing and building as reading and writing. In R. E. Ferdig & S de Freitas (Eds.), *Interdisciplinary advancements in gaming, simulations and virtual environments: Emerging trends* (pp. 1-17). Hershey, PA: IGI Global.

Chan, C. C., Tsui, M. S., Chan, M. Y., & Hong, J. H. (2002). Applying the structure of the observed learning outcomes (SOLO) taxonomy on student's learning outcomes: An empirical study. *Assessment & Evaluation in Higher Education, 27*(6), 511-527. https://doi.org/10.1080/0260293022000020282

Chatzigrigoriou, M., & Fokides, E. (2016). Κατασκευή ψηφιακών παιχνιδιών για την ανάπτυξη προγραμματιστικών δεξιοτήτων σε παιδιά. Αποτελέσματα από πιλοτικό πρόγραμμα σε μαθητές της Στ' τάξης [Using 3D games for the development of programming skills to sixth-grade primary school students. Results of a pilot program]. *Θεωρία και Έρευνα στις Επιστήμες της Αγωγής, 2*(5), 27-44.

Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012). A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education, 59*(2), 661-686. https://doi.org/10.1016/j.compedu.2012.03.004

Consalvo, M., & Dutton, N. (2006). Game analysis: Developing a methodological toolkit for the qualitative study of games. *Game Studies*, *6*(1), 1-17.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage Publications.

Creswell, J. W., & Poth, C. N. (2017). *Qualitative inquiry and research design: Choosing among five approaches.* Sage Publications.

De Freitas, S. (2006). *Learning in immersive worlds.* London: Joint Information Systems Committee. Overview of research on the educational use of video games.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, *58*(1), 240-249. https://doi.org/10.1016/j.compedu.2011.08.006

Doran, K., Boyce, A., Finkelstein, S., & Barnes, T. (2012, July). Outreach for improved student performance: A game design and development curriculum. *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, 209-214. ACM. https://doi.org/10.1145/2325296.2325348

Dorling, M., & Walker, M. (2014). *Computing progression pathways.* U.K.: Computing at School.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57-73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

Egenfeldt-Nielsen, S. (2006). Overview of research on the educational use of video games. *Digital kompetanse, 1*(3), 184-213.

Felicia, P. (Ed.). (2011). *Handbook of research on improving learning and motivation through educational games: multidisciplinary approaches*. Hershey, PA: IGI Global. https://doi.org/10.4018/978-1-60960-495-0

Felicia, P. (Ed.). (2012). *Developments in current game-based learning design and deployment*. Hershey, PA: IGI Global.

Flannery, L., Silverman, B., Kazakoff, E., Bers, M., Bonta, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. *Proceedings of the 12th International Conference on Interaction Design and Children*, 1-10. https://doi.org/10.1145/2485760.2485785

Fowler, A., & Cusack, B. (2011). Enhancing introductory programming with Kodu Game Lab: An exploratory study. In S. Mann, & M. Verhaart (Eds.), *Proceedings of the 2nd Annual Conference of Computing and Information Technology Education and Research in New Zealand* (pp. 69-79). Hamilton, New Zealand: CITRENZ.

Fokides, E., & Atsikpasi, P. (2017). Redefining the framework for teaching programming to primary school students. Results from three pilot projects. *Proceedings of the International Conference on Information, Communication Technologies in Education, ICICTE 2017*. Rhodes, Greece: ICICTE.

Gee, J. P. (2014). *What video games have to teach us about learning and literacy*. Macmillan.

Glass, G. V., Peckham, P. D., & Sanders, J. R. (1972). Consequences of failure to meet assumptions underlying fixed effects analyses of variance and covariance. *Review of Educational Research, 42*, 237-288. https://doi.org/10.3102/00346543042003237

Good, J. (2011). Learners at the wheel: Novice programming environments come of age. *International Journal of People-Oriented Programming, 1*(1), 1-24. https://doi.org/10.4018/ijpop.2011010101

Good, J., Howland, K., & Nicholson, K. (2010, September). Young people's descriptions of computational rules in role-playing games: An empirical study. *Proceedings of Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium*, 67-74. IEEE.

Govender, I., Govender, D. W., Havemga, M., Mentz, E., Breed, B., Dignum, F., & Dignum, V. (2014). Increasing self-efficacy in learning to program: exploring the benefits of explicit instruction for problem solving. *TD: The Journal for Transdisciplinary Research in Southern Africa*, *10*(1), 187-200. https://doi.org/10.4102/td.v10i1.19

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014, November). Computational thinking skills in Dutch secondary education: Exploring teacher's perspective. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 124-125. ACM. https://doi.org/10.1145/2670757.2670761

Grigoriadou, M., Gogoulou, A., & Gouli, E. (2002). Εναλλακτικές διδακτικές προσεγγίσεις σε εισαγωγικά μαθήματα προγραμματισμού: Προτάσεις διδασκαλίας [Alternative instructional approaches in introductory programming courses: Teaching suggestions. In A. Dimitrakopoulou (Ed.), *Proceedings of the 3ου Συνεδρίου ΕΤΠΕ, οι ΤΠΕ στην Εκπαίδευση* [3rd Conference on ICT in Education] (pp. 239-248).

Grout, V., & Houlden, N. (2014). Taking computer science and programming into schools: The Glyndŵr/BCS Turing project. *Procedia-Social and Behavioral Sciences*, *141*, 680-685. https://doi.org/10.1016/j.sbspro.2014.05.119

Guzdial, M. (2009). *Question everything: How we teach intro CS is wrong*. Computing Education Blog.

Harel, I. (1991). *Children designers: Interdisciplinary constructions for learning and knowing mathematics in a computer-rich school*. Ablex Publishing.

Harteveld, C., Smith, G., Carmichael, G., Gee, E., & Stewart-Gardiner, C. (2014). A design-focused analysis of games teaching computer science. *Proceedings of Games+ Learning+ Society*, 10.

Hernandez, C. C., Silva, L., Segura, R. A., Schimiguel, J., Ledón, M. F. P., Bezerra, L. N. M., & Silveira, I. F. (2010). Teaching programming principles through a game engine. *Clei electronic journal*, *13*(2), 3.

Hayes, E. R., & Games, I. A. (2008). Making computer games and design thinking: A review of current software and strategies. *Games and Culture, 3*(3-4), 309-332. https://doi.org/10.1177/1555412008317312

Hoganson, K. (2010). Teaching programming concepts with GameMaker. *Journal of Computing Sciences in Colleges*, *26*(2), 181-188.

Howland, K., Good, J., & du Boulay, B. (2013). Narrative threads: A tool to support young people in creating their own narrative-based computer games. *Transactions on Edutainment X*, 122-145. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37919-2_7

Hwang, G. J., Hung, C. M., & Chen, N. S. (2014). Improving learning achievements, motivations and problem-solving skills through a peer assessment-based game development approach. *Educational Technology Research and Development, 62*(2), 129-145. https://doi.org/10.1007/s11423-013-9320-7

Hwang, G. J., & Wu, P. H. (2012). Advancements and trends in digital game-based learning research: A review of publications in selected journals from 2001 to 2010. *British Journal of Educational Technology, 43*(1), E6-E10. https://doi.org/10.1111/j.1467-8535.2011.01242.x

Jimoyiannis, A. (2013). Using SOLO taxonomy to explore students' mental models of the programming variable and the assignment statement. *Themes in Science and Technology Education, 4*(2), 53-74.

Jones, S. P., Bell, T., Cutts, Q., Iyer, S., Schulte, C., Vahrenhold, J., & Han, B. (2011). *Computing at school. International comparisons*. U.K.: Computing at School.

Kafai, Y. B. (1996). Learning design by making games. In Y. B. Kafai, *Constructionism in practice: Designing, thinking and learning in a digital world* (pp. 71-96). Routledge.

Kafai, Y. B. (2012). *Minds in play: Computer game design as a context for children's learning*. Routledge.

Kafai, Y. B., & Harel, I. (1991). Learning through design and teaching: Exploring social and collaborative aspects of constructionism. In I. Harel, & S. Papert (Eds.) *Constructionism* (pp. 85-106). Norwood, USA: Ablex.

Kafai, Y. B., & Peppler, K. A. (2011). Youth, technology, and DIY: Developing participatory competencies in creative media production. *Review of Research in Education, 35*(1), 89-119. https://doi.org/10.3102/0091732X10383211

Kafai, Y. B., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Mahwah, USA: Lawrence Erlbaum Associates.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior, 52,* 200-210. https://doi.org/10.1016/j.chb.2015.05.047

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

Ke, F. (2009). A qualitative meta-analysis of computer games as learning tools. R. Ferdig (Ed.), *Handbook of research on effective electronic gaming in education*, (pp. 1-32). Hershey, PA: IGI Global. https://doi.org/10.4018/978-1-59904-808-6.ch001

Ke, F. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education, 73*, 26-39. https://doi.org/10.1016/j.compedu.2013.12.010

Keren, G., & Fridin, M. (2014). Kindergarten Social Assistive Robot (KindSAR) for children's geometric thinking and metacognitive development in preschool education: A pilot study. *Computers in Human Behavior, 35*, 400-412. https://doi.org/10.1016/j.chb.2014.03.009

Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010, September). Towards the automatic recognition of computational thinking for adaptive visual language learning. In C., Hundhausen, E., Pietriga, P. Díaz, & M. Rosson (Eds.), *Proceedings of the Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium,* 59-66. IEEE. https://doi.org/10.1109/VLHCC.2010.17

Kutnick, P., Sebba, J., Blatchford, P., Galton, M., Thorp, J., MacIntyre, H., & Berdondini, L. (2005). *The effects of pupil grouping: Literature review*. Research report RR688. Nottingham: DFES Publications.

Landis, J. R., Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics 33*, 159-174. https://doi.org/10.2307/2529310

Lee, I., Martin, F., & Apone, K. (2014). Integrating computational thinking across the K-8 curriculum. *ACM Inroads*, *5*(4), 64-71. https://doi.org/10.1145/2684721.2684736

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education, 21*(2), 105-134. https://doi.org/10.1080/08993408.2011.579805

Li, Q. (2010). Digital game building: Learning in a participatory culture. *Educational Research, 52*(4), 427-443. https://doi.org/10.1080/00131881.2010.524752

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

Lix, L. M., Keselman J. C., & Keselman H. J. (1996). Consequences of assumption violations revisited: A quantitative review of alternatives to the one-way analysis of variance F test. *Review of Educational Research, 66*, 579-619.

Luckin, R., Bligh, B., Manches, A., Ainsworth, S., Crook, C., & Noss, R. (2012). *Decoding learning: The proof, promise and potential of digital education*. London, UK: Nesta.

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using paired-programming: Who benefits? *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education, 6*(2), 1411-14125.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. In J. Dougherty, S. Rodger, S. Fitzgerald, & M. Guzdial (Eds.), *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 367-371). New York: ACM. https://doi.org/10.1145/1352135.1352260

Margulieux, L. E., Guzdial, M., & Catrambone, R. (2012, September). Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. *Proceedings of the Ninth Annual International Conference on International Computing Education Research,* 71-78. ACM. https://doi.org/10.1145/2361276.2361291

McInerney, C. (2010). Having fun with computer programming and games: Teacher and student experiences. *Teaching Fundamentals Concepts of Informatics*, 136-142. https://doi.org/10.1007/978-3-642-11376-5_13

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011, June). Habits of programming in Scratch. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 168-172. ACM. https://doi.org/10.1145/1999747.1999796

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

Merchant, G. (Ed.). (2013). *Virtual literacies: Interactive spaces for children and young people* (Vol. 84). Routledge.

Morris, D., Uppal, G., & Wells, D. (2017). *Teaching computational thinking and coding in primary schools*. London: Learning Matters.

Murnane, J. S. (2010). *Programming languages for beginners*. Lambert, Saarbrucken.

Navarrete, C. C., & Minnigerode, L. (2013, June). Exploring 21st century learning: Game design and creation, the students' experience. *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 282-293.

OfCom, U. K. (2012). *Children and parents: media use and attitudes report*.

Organisation for Economic Co-operation and Development-OECD (2015). *Students, computers and learning: Making the connection*. Paris: PISA, OECD Publishing.

Owston, R., Wideman, H., Ronda, N. S., & Brown, C. (2009). Computer game development as a literacy activity. *Computers & Education, 53*(3), 977-989. https://doi.org/10.1016/j.compedu.2009.05.015

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

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. Basic Books.

Papert, S. (1996). *The connected family: Bridging the digital generation gap*. Marietta, GA: Longstreet Press.

Papert, S. (1999). *Eight big ideas behind the Constructionist Learning Lab*. Retrieved from http://stager.org/articles/8bigideas.pdf

Payton, S., & Hague, C. (2010). *Digital literacy professional development resource*. Futurelab.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, *2*(1), 25-36. https://doi.org/10.2190/689T-1R2A-X4W4-29J2

Pea, R. D., & Kurland, D.M. (1984). On the cognitive effects of learning computer programming. *New Ideas Psychology, 2*, 137-168. https://doi.org/10.1016/0732-118X(84)90018-7

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research, 2*(1), 37-55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Perrotta, C., Featherstone, G., Aston, H., & Houghton, E. (2013). *Game-based learning: Latest evidence and future directions*. NFER Research Programme: Innovation in Education. Slough: NFER.

Pilot, K. (2009). *The impact of Web 2.0 technologies in the classroom*. Knowledge Bank: Next Generation research report. State of Victoria: Department of Education and Early Childhood Development, Melbourne, Australia.

Prensky, M. (2004). *What kids learn that's positive from playing video games*. Simon Fraser University, Surrey Campus Library.

Prensky, M. (2007). *Digital game-based learning* (Vol. 1). St. Paul, MN: Paragon House.

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. https://doi.org/10.1145/1592761.1592779

Robertson, J. (2013). The influence of a game-making project on male and female learners' attitudes to computing. *Computer Science Education, 23*(1), 58-83. https://doi.org/10.1080/08993408.2013.774155

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education-An International Journal*, *10*(1), 73-88.

Salen, K. (2007). Gaming literacies: A game design study in action. *Journal of Educational Multimedia and Hypermedia, 16*(3), 301.

Sanford, K., & Madill, L. (2007). Recognizing new literacies: Teachers and students negotiating the creation of video games in school. *Proceedings of the Digital Games Research Association Conference,* 583-589.

Schelhowe, H. (2010, February). *Using construction kits: Just learning how to program a computer-or is there more educational benefit?* Paper presented at the Digital Media and Learning Conference. La Jolla, USA

Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'grady-Cunniff, D., Boucher Owens, B., Stephenson, C. & Verno, A. (2011). *Computer science standards*. Computer Science Teachers Association.

Shaw, E., Boehm, Z., Penwala, H., & Kim, J. (2012, June). GameMath! Embedding secondary mathematics into a game making curriculum. *Proceedings of the 2012 ASEE Annual Conference and Exposition*. San Antonio, Texas.

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008, June). Going SOLO to assess novice programmers. *ACM SIGCSE Bulletin*, *40*(3), 209-213). https://doi.org/10.1145/1597849.1384328

Soloway, E. (2013). *Studying the novice programmer*. Psychology Press.

Squire, K. (2005). Changing the game: What happens when video games enter the classroom. *Innovate: Journal of Online Education, 1*(6).

Stiller, E. (2009). Teaching programming using bricolage. *Journal of Computing Sciences in Colleges, 24*(6), 35-42.

Stolee, K. T., & Fristoe, T. (2011, March). Expressing computer science concepts through Kodu game lab. *Proceedings of the 42nd ACM Technical Symposium on Computer science education*, 99-104. ACM. https://doi.org/10.1145/1953163.1953197

Ulicsak, M., & Williamson, B. (2011). *Computer games and learning: a handbook*. London: Futurelab.

Vygotsky, L. S. (1980). *Mind in society: The development of higher psychological processes*. Harvard University Press.

Wilson, A., Hainey, T., & Connolly, T. (2012, October). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. *Proceedings of* the *European Conference on Games Based Learning*, 549. Academic Conferences International Limited.

Yang, Y. T. C., & Chang, C. H. (2013). Empowering students through digital game authorship: Enhancing concentration, critical thinking, and academic achievement. *Computers & Education, 68*, 334-344. https://doi.org/10.1016/j.compedu.2013.05.023

Zhang, J. X., Liu, L., Ordóñez de Pablos, P., & She, J. (2014). The auxiliary role of information technology in teaching: Enhancing programming course using Alice. *International Journal of Engineering Education, 30*(3), 560-565.

Zimmerman, E. (2009). Gaming literacy: Game design as a model for literacy in the twenty-first century. *The Video Game Theory Reader, 2*, 23-31.

BIOGRAPHY

Dr. Emmanuel Fokides is an Assistant Professor in the Department of Primary School Education, University of the Aegean, Greece. His courses focus on the educational uses of Virtual Reality, digital storytelling, Augmented Reality, and Serious Games. Since 1994, he is involved in a number of research projects regarding distance and lifelong learning and the educational uses of Virtual and Augmented Reality. His work is published in several conference proceedings, international volumes, and journals. He is also the co-author of two books.