Ingame worked examples support as an alternative to textual instructions in serious games about programming

Abstract

Serious games are considered an effective method to engage students in programming education and have been increasingly used in classrooms. An important part of the learning process with serious games involves the presentation of the new concepts and the provided support to encounter student difficulties. Although the most common approach is the use of instructional text, it comes with some drawbacks. This paper proposes an alternative method for user support in serious games about programming which mitigates current problems and provides improved learning efficiency. An experiment was conducted (N = 291) to test textual instructions learning efficiency compared to in-game worked examples. Two randomly assigned groups used different types of support in a block-based game, developed for the study. Our implementation provided support through a non-player character who executed worked examples inside the game world. The analysis showed a significant statistical difference in score performance between the two supports on both novice and experienced students. The results point to increased learning efficiency by students, when in-game worked examples and problem solving are combined. We further argue that the Cognitive Load Theory can provide adequate justification for the outcomes.

Running head: INGAME WORKED EXAMPLES IN SERIOUS GAMES

Toukiloglou Pavlos (corresponding author)
Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Street, GR-54636, Thessaloniki, Greece
toukiloglou@uom.edu.gr

Xinogalos Stelios
Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Street, GR-54636, Thessaloniki, Greece
stelios@uom.edu.gr

Keywords

serious games, support design, computer programming, worked examples

Stelios Xinogalos is an Associate Professor in the Department of Applied Informatics, University of Macedonia, Greece. His research interests include Programming Environments and Techniques, Object-oriented Design and Programming, Didactics of Programming, Computer Science Education, Educational Technology, and Serious Games. He has published more than 100 research papers in International journals, conferences and books.

Pavlos Toukiloglou received a BSc degree in Applied Informatics from the University of Macedonia and an MSc in Computer Graphics and Virtual Environments from the University of Hull. He is currently working towards a Ph.D. degree in serious games about programming at the University of Macedonia. His research interests include serious games, virtual learning environments, and pedagogical agents.

1. Introduction

Serious games describe a category of educational software designed to combine entertainment with learning. To achieve that, they are designed to be appealing to players and meet their didactic objectives at the same time (Bellotti et al., 2010). A high number of motivators are built-in serious games that increase student engagement, which is essential in educational activities (Laine & Lindberg, 2020). Laine and Lindberg (2020) define fifty-six motivators, which are taxonomized in fourteen classes and include concepts such as immersion, competition, emotions and challenge. Utilizing student engagement educational games can convey the learning content through their mechanics (Patino et al., 2016). They assist users to construct new knowledge through active learning, which stimulates cognitive processing. However, as the cognitive load theory (CLT) states, learning becomes inefficient when a trainee’s cognitive capacity overloads (Sweller et al., 1998). To compensate for the problem, games include supports to introduce new concepts while providing feedback in a constant response loop.

Common supports to help users reduce the cognitive load include tips or instructional text. Despite their presence in many serious games, the aforementioned methods fail to fulfill their purpose (White, 2014). Text and tips are designed as purely instructive means of support and display static information without a procedure to verify knowledge acquisition. If users fail to progress through the game, the same information will be presented to them again, resulting in frustration and learning inefficiency. Due to these problems alternatives like worked examples (WE) began to emerge in the tutorial designs (Spieler et al., 2020). Particularly in programming, worked examples is an important pedagogic strategy, as it promotes schemata construction by managing cognitive load during problem-solving (Chi et al., 1989). Most of the studies in the field conclude that further research is needed to find alternative support designs to improve learning effectiveness (Andersen et al., 2012; Spieler et al., 2020; Zhi et al., 2018). The paper proposes a variant of worked examples, where they are placed in-game and provide immediate feedback on how instructions are used. Two controlled studies were conducted for novice and experienced students in programming respectively, using a serious game, for investigating the following research questions:

RQ1: Do players with no previous experience in programming achieve better scores in serious games about programming when they are provided with in-game example based help compared to textual instructions help?
RQ2: Do players with previous experience in programming achieve better scores in serious games about programming when they are provided with in-game example based help compared to textual instructions help?

The goals of the two studies as expressed by their underlying research questions are the following:

1. Provide empirical evidence to the state of art in instructional design, by examining the effectiveness of in-game worked examples in terms of the support provided for comprehending programming concepts as an alternative to text in serious games about programming.
2. Explore the effects of the two aforementioned kinds of support on both novice and experienced students.

In order to investigate the research questions and achieve the goals of the two studies, a serious game called Dungeon class was built. This was necessary since we were not able to find any game that utilized in-game worked examples for the field of computer programming. The Dungeon class game provides a version that provides support through in-game worked examples presented by a non-player character (NPC) and a version that uses text-based instructions for the same purpose. Participants were randomly assigned to one of the two types of support and the efficiency of level completion on each implementation was investigated. The results were interpreted taking into account the cognitive load theory and more specifically the cognitive load during learning when in-game worked examples and textual instructions are used.

The rest of this paper is organized as follows: The next section presents the related work in the field, followed by an introduction to cognitive load theory and worked examples. It continues with a description of the serious game and the design process of it. Next, the methodology of the study and the results of the statistical analysis are presented. The last sections present a discussion of the results, limitations and final conclusions.

2. Related work

Novice students often use block-based environments to learn programming such as Scratch (*Scratch - About*, n.d.), Hour of Code (*Hour of Code*, n.d.) and App Inventor (*MIT App Inventor | Explore MIT App Inventor*, n.d.). Those environments provide learning material through text, tutorials and example code. However, students with no previous experience in programming have difficulties in understanding the example code, adapting it and integrating it into their code (Gross & Kelleher, 2010). Ichinco al. (2017) analyze in their study how novice students use examples in a block-based programming environment. They conclude that although examples and annotations help novice programmers to perform better in programming tasks, the support is not enough.

Zhi et al. (2018) designed BOTS, a serious game about programming to compare the in-game performance of participants in puzzle completion time and solution code length, for three kinds of support: instructional text, worked examples, and erroneous worked examples. In their study, 6-12th grade students used a LOGO-like environment to control a robot with movement commands, conditions, loops and functions. The worked examples were placed in a separate window and consisted of the optimal solution for the level. They used text popups to explain the code and a follow-up question related to the puzzle for checking player understanding. They concluded that the examples group performed better in loops and functions. Furthermore, they suggested interweaving problem solving with examples in future iterations. In another empirical study, Zhi et al. (2019) explored the impact of worked examples by designing a peer code helper which integrates worked examples into the programming environment Snap!. Their system presented students with coding exercises and included the first steps of the solution while guiding them with text messages through the final result. Scaffolded and stencil techniques were used by presenting WE in small steps via self-explanation prompts to control participants actions and reflect on their code. They concluded that although the intrinsic cognitive load of students was improved, student learning was not significantly altered by the new method. Spieler and Slany (2020) compared WE with text examples in a serious game about programming and found that the average scores between the two types of support were not significantly different. Nonetheless, participants from the worked example group finished significantly faster, concluding an overall better performance.

Concluding, the literature results on related work showed different approaches and limitations on the implementation of WE. In the study by Zhi et al. (2019) WE were used in the programming environment Snap!, which is not a serious game. Spieler and Slany (2020) had their WE appear as an image of command blocks. Students had to manually insert the commands into their program to discover their use. The functionality of executing the provided code in order to understand it was missing. Finally, Zhi et al. (2018) provided WE in the BOTS game as an instructional support, but they did not interleave it with problem solving as we did in our study. Therefore, this paper differs from the above research by placing WE inside the game and next to the player to reduce cognitive load even further. Moreover, this empirical study explores the effects of support in programming on both novice and experienced students.

3. Cognitive load theory and worked examples

Cognitive load theory (CLT) assumes that the human brain has a finite number of resources it can dedicate during the process of learning. It is a method of measuring the amount of load in the human mind and optimizing the efficiency of instructional systems, considering people's limited cognitive processing capacity (Sweller et al., 1998). According to CLT, there are three types of cognitive load: intrinsic, extraneous, and germane. Intrinsic, also called endogenous, is the load related to the difficulty level of a specific learning task, while extraneous refers to the load generated by distractions and interruptions during learning. Lastly, the germane load is the type of activity that occurs when the brain constructs new schemas and stores them from short-term memory to long-term memory. According to psychology, schemas are patterns of thought or behavior that organize categories of information and the relationships among them (DiMaggio, 1997). The core concept of designing effective instructions according to CLT is to simplify the complexity of new information (intrinsic load), reduce distractions while processing new information (germane load) and facilitate deep processing of new information (germane cognitive load). If the additive total of the three load types exceeds the trainee's maximum level, the learning procedure will stop because he or she will enter a state called cognitive overload.

Extraneous and intrinsic cognitive loads are both depending on how the learning activities are designed and experienced, with the extraneous load being associated with the design of the instructional materials and intrinsic load is generated by the inherent complexity of a specific instructional topic. Germane load relies on the trainee's reaction to the learning material and the active decision to devote finite working memory resources to the task at hand. Optimizing these loads during learning activities frees cognitive resources in the restricted working memory and as a consequence actual learning can take place with the construction of schemas. CLT achieves the above by managing the inherent complexity of what is being learned (Abdul-Rahman & du Boulay, 2014).

Programming is a complex cognitive skill and learning how to program is a difficult task (Xinogalos, 2016). Very often novice students encounter many difficulties when they try to implement the solution steps to a given problem (Proulx, 2000). CLT can be applied in the learning process with the use of worked examples (WE). It has been demonstrated that WE are an effective approach for supporting complex learning when it is guided under certain principles (*Learning from Examples: Instructional Principles from the Worked Examples Research - Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, Donald Wortham, 2000*, n.d.). The extraneous cognitive load is reduced and allows the learner to devote cognitive resources to useful loads (Magana et al., 2015). Also, the WE approach is guided by principles associated with CLT and has been recognized as a relevant strategy for supporting novices in learning tasks that involve a high cognitive load (Paas et al., 2003). Additionally, the borrowing principle when incorporated into CLT can further promote the effectiveness of WE when compared to simple problem-solving techniques. The borrowing principle states that new schemas are constructed by borrowing information from someone else’s long-term memory and combining it with information stored in the learner's long-term memory (Sweller, 2006). If there isn't someone to borrow from or information in the long-term does not exist, the procedure will have inevitable random components, created by the learner’s attempt to assimilate new knowledge. To solve the problem, the learner will have to randomly generate a solution and test its effectiveness by trial and error until it is viable. WE can reduce this random process and increase learning efficiency when compared with problem-solving since the latter elicits a higher cognitive load from random components.

According to Kalyuga et al. (2001), WE have beneficial results only when they are appropriately structured and refer to novice learners. Students with prior knowledge have already incorporated the task elements for solving the problem into cognitive schemas, so the intrinsic load introduced by the problem is lower (Kalyuga et al., 2001). Another factor to be considered is that a single WE per instructional area will not promote the expected results. After the WE is analyzed, learners require a problem to provide them with feedback and verify what they have learned. In this procedure where a problem succeeds the WE motivates learners to actively process newly taught instructions, especially when the material does not comprise previous knowledge from the curriculum (Sweller, 2006).

Most of the serious games about programming, present help to users with the same methods. Help instructions are given as text messages, flashcards, or short video presentations. Although the expository text is a very common method for introducing new instructions, students find it dry and boring (Ginns et al., 2013). As a result, they engage it superficially and construct insufficient or incorrect representations of the author’s argument (Ginns et al., 2013). A similar method, flashcards, usually appears in games as pop-ups on the screen and present new information required for the completion of the current level or puzzle. Though flashcards work great in education with skills that need memorization like multiplication tables, grammar rules, etc., they do not translate well with complex information. The didactic instructional strategy of a flashcard will fail when applied to more complex topics like programming (White, 2014). Flashcards present new information to users with the hope that they will absorb it, without verifying that any learning has taken place (White, 2014). Lastly, video presentations or tutorials are not easy to use because of their inherent difficulty to step forwards or backward (Abdul-Rahman & du Boulay, 2014).

Apart from how instructions are presented to the user, their content is also important. It is very common for help to mainly consist of instructions for solving the problem. In such cases, novices often resort to weak problem-solving strategies such as means-ends analysis, in which learners continuously search for operators to reduce the difference between the current problem state and the goal state (Chandler & Sweller, 1991). Even by applying a weak strategy, a learner can eventually succeed in solving the problem but it has been shown to contribute very little to learning. The high extraneous load on working memory imposed by the weak strategy prevents building a cognitive schema of how such problems should be solved. Contrarily, WE allow learners to study a worked-out solution, devoting all their available cognitive capacity and constructing the cognitive schema for solving this kind of problem (Cooper & Sweller, 1987).

4. Dungeon class

Dungeon Class is a block-based serious game inspired by Blockly Games (About Blockly Games, n.d.) and Hour of Code (Hour of Code, n.d.). It is an original game created by us from scratch and can be accessed via the following link: http://users.sch.gr/pavlos/dungeonclass/. Please note that there is not a multi-language support yet and all messages are in Greek. The editor of the programming environment was created using the Blockly library and the main game was developed in Unity (*Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*, n.d.), a popular game engine with a powerful suite of tools, allowing the rapid development of ideas. The game was exported in WebGL so it can be easily accessible to students via a web page. The learning objective is to teach the foundational coding skills of sequencing and loops to novice students with minimal or no previous experience in coding. It follows a puzzle-based level structure with a steady increase in difficulty, where new key programming concepts are gradually introduced. There are a total of 10 levels where the first five refer to sequencing and the rest to single or nested loops. The game takes place inside a dungeon and the user directs his or her avatar to the room exit by creating a single program for each level (Figure 1). Some rooms include obstacles, forcing players to take a longer route and imply the use of loop blocks. Additionally, in two of the levels, doors are locked and require the use of the pick key command from the corresponding tile, before moving towards the exit. Puzzle design and progressive difficulty were loosely based on Hour of Code Minecraft's (*Minecraft Hour of Code*, n.d.) serious games.

The game uses the Blockly library to create a programming area where command blocks are dragged and dropped into. The program runs when the corresponding button is pressed, followed by a step by step execution of the code. If a block is not directly connected with the rest of the program, an explanatory message will inform the user. If players fail to program a correct solution, they are prompted accordingly and allowed to try again. After the completion of each level, players are awarded gold coins according to their solutions and failed attempts. An optimal solution in the first try will grant them 300 coins, 200 coins for two or more tries, and 100 coins if the solution wasn't optimal. The reward system was designed as a gamification technique to encourage players to code the best possible solutions. Apart from that, coins are not used in any game mechanic or measure player performance. Also, in the level completion dialogue, a map of the dungeon appears, featuring the player's current position and path, enhancing the sense of progress (Figure 2).

-Insert Figures 1 and 2 here-

Dungeon class has two types of instructional support: text and examples. Each time the game executes, it assigns randomly one type of support and displays it accordingly. The first one provides textual information about new commands and suggestions relevant to the current level. It is positioned to the bottom of the screen and, in some cases, images are included to facilitate new command identification. Worked examples are implemented through a Non-Player Character (NPC) with autonomous behavior. The NPC moves in a room connected but separated from the player, inside the game world. It has the same goal of passing through the exit and creates small programs with blocks to achieve it. The rooms NPC populates have similar difficulty with the player's rooms but doors and obstacles are arranged differently which implies different paths and solutions. Consequently, each worked example presented by the NPC aims to support students in comprehending the semantics of the programming concepts used in it, which is a prerequisite for moving to more advanced programming concepts. It is clear that in the case that a student does not comprehend the underlying programming concepts s/he will not be able to correctly apply them in the problem s/he has to solve. The thinking process, code and step by step execution of the path are presented to the user in a floating window above the NPC, as s/he runs his programs in real-time inside the game world. The user interface allows players to zoom and pan the camera if they choose to isolate and observe the NPC's movement separately from their room.

5. Study methodology and results

5.1 Study design

As already mentioned, the objective of this paper was to compare the effect of support in serious games about programming in terms of learning efficiency. To measure the impact of the supports, two controlled studies were planned for novice and experienced students in programming respectively. Both studies intended to test if in-game help during problem-solving had increased beneficial results in learning when compared with the textual instructions counterpart. Our assumption based on the literature regarding CLT and WE, presented in section 3, was that students using the in-game WE would perform better than the textual support, due to the reduced cognitive load. Having both types of students in regard to programming experience in our study, was expected to provide an insight on the use of WE in serious games and in what target groups they can be applied.

It was important for the study to develop a method for measuring student learning within the game. This would validate the association between the type of help and the learning of new programming concepts. The most common methods in the literature measure the raw performance of players. The simplest implementation is the measurement of the last level or the total amount of levels completed. It was used by Delacruz (Delacruz et al., 2010) as a standalone assessment of student math ability and Andersen (Andersen et al., 2012) in a study of how efficient players were learning to play a game. Although this approach is suitable for Dungeon Class since it is divided into distinct levels, it will not provide evidence of understanding key programming concepts. Sequencing commands, for example, will provide the same results as using a loop in terms of level advancement. Consider an example problem where the player has to move 5 steps forward to complete the level and reach the exit. This can be accomplished either by using a loop statement or sequencing 5 forward commands (Figure 3). Both solutions will result in winning the level, however, the second one is inconsistent with programming efficiency and code optimization. Methods with similar measurements like time of level completion are prone to the same learnability problems.

-Insert Figure 3 here-

Since the main focus of the study was to measure the efficiency of in-game worked examples in conjunction with problem-solving, it was decided that experimentation should not be penalized. Students should be able to develop their solution strategies and have the freedom to learn new concepts at their own pace. Experimentation is most effective when mistakes are not punished and occur in a safe environment (Paas et al., 2003). Students were encouraged before the game session to take their time exploring the new commands presented in each one of the game's levels and debug the code by executing parts of it, if necessary. Hence, time of level completion and failed attempts were not directly included in the computation of player efficiency. Instead, it was designed to evaluate the student solution by dividing the number of blocks of the optimal solution on a level, by the number of blocks committed on that level. Since the student solution will always have equal or more blocks than the optimal, the calculation will result in a decimal number, where 1 is the maximum possible efficiency score per level. The Equation (1) can be further adapted by applying a difficulty coefficient (D_i) on the score of each level. For this study, difficulty coefficient was considered equal for all levels and had the value of 1 due to the scaffolded level design. Each level was based on the student's accumulated knowledge of the previous one. If a new concept was introduced, the complexity of the problem was decreased to maintain the same overall difficulty.

The sum of scores of each completed level defines the effectiveness per student. It is clear that the time of level completion is indirectly taken into account in the computation of the overall score, since less time of level completion results in more completed levels. An example of how the efficiency scores of 2 participants are calculated is illustrated in Table 1. It should be noted that completion of all levels was not a requirement for the computation of the equation and the participant's score was measured within the 45' session according to the last level reached. Also, the first 3 tutorial levels were not included in the calculation.

$$Efficiency = \sum_{i=1}^{n} \frac{OptimalSolution_i}{StudentSolution_i} \times D_i \qquad (1)$$

-Insert Table 1 here-

5.2 Procedure

The studies were conducted in the context of a computer science course in 5 Greek elementary schools. During the experiments, each class had 45 minutes to conclude the game and was administered by a teacher. In that period, students were instructed to rely solely on the game's help system they were randomly assigned to. Teachers were not allowed to guide or advise on puzzle solutions but only to assist in case of a technical problem. The first three levels of the game acted as a tutorial and included both types of help, textual and worked examples. Their purpose was to guide players on how to use block commands and demonstrated game mechanics.

After the end of the tutorial phase, students were randomly divided by the game into two groups according to help support. In the following levels (4-10) only one type of help was provided and each student used the remaining time to progress to the best of his/her ability. In the first study, 45.4% of the students used the in-game worked examples executed by an NPC and 54.6% of the students had textual instructions. In the second study, 53.3% used NPC and 46.7% textual support. Figures 4 and 5 show screenshots of the two kinds of support provided between levels 4 and 10.

-Insert Figures 4 and 5 here-

Throughout the session, game learning analytics were registered after each level completion automatically in an online database. This procedure had the advantage of gathering data without the assistance of local class administrators. Each student field in the database stored the type of support, school name, class, gender and also the following metrics per level: time of completion, the total number of blocks used and failed attempts. Data was saved in the JSON (JavaScript Object Notation) format in order to be human readable and easy to transmit/convert to statistical analysis applications.

5.3 Participants

The two studies were conducted by 291 students in total, aged from 9-12 in five elementary schools. All participants were under the same curriculum and distributed as shown in Tables 2a and 2b for the first study and Tables 3a and 3b for the second study. Schools belong to the same district and all students had similar characteristics in terms of education level in computer science and access to computer labs.

The sample for the first study consisted of 141 students, 84 of them males and 57 females from 4 elementary schools (Table 2a). The students attended 3rd and 4th grade (Table 2b) and had no previous experience in programming.

-Insert Tables 2a, 2b here-

The second group had a total of 150 students, 77 males and 73 females from 3 elementary schools (Table 3a). In the second study, students had already been taught basic programming concepts and had used educational software like Scratch, EasyLogo (*EasyLogo*, n.d.) and Kodu (*Kodu Game Lab | KoduGameLab*, n.d.). Through the 5th and 6th grade curriculum, most of them were already familiar with sequencing, loops and conditional statements. The class distribution of the students in the second study is presented in Table 3b.

-Insert Tables 3a, 3b here-

5.4 Data analysis

Although all schools followed the same curriculum, the programming experience of all participants before the studies was checked to ensure the validity of the experiment. A statistical analysis was conducted on the first three tutorial levels where both types of help were included. Firstly, the Shapiro-Wilk test was used to examine the null hypothesis that the scores sample came from a normally distributed population. In both studies the null hypothesis was rejected (p=0.000) which indicates that player scores for levels 1 to 3 were not normally distributed (Table 4a and Table 4b). Due to that fact, the non-parametric Kruskal-Wallis test was conducted to check whether efficiency scores originate from the same distribution. The tested null hypotheses were:

Null hypothesis for study 1 (H1a):
H_o: The distribution of efficiency score in players with no previous experience in programming is the same across all schools.
Null hypothesis for study 2 (H2a):
H_o: The distribution of efficiency score in players with previous experience in programming is the same across all schools.

Similarly, the Shapiro-Wilk test was used to check if efficiency scores for levels 4 through 10 were normally distributed (Study 1: Table 4a, Study 2: Table 4b). In both studies the null hypothesis was rejected (Study 1: NPC p=0.032, Textual p=0.043, Study 2: NPC p=0.000, Textual p=0.001). The non parametric method of Mann-Whitney was conducted to test the following null hypotheses:

Null hypothesis for study 1 (H2a):
H_o: Players with no previous experience in programming that had been provided with in-game example based help will perform the same scores like the ones that had been provided with textual help.
Null hypothesis for study 2 (H2b):
H_o: Players with previous experience in programming that had been provided with in-game example based help will perform the same scores like the ones that had been provided with textual help.

-Insert Table 4a here-
-Insert Table 4b here-

5.5 Results

The objective of the first test (H1a, H1b) was to confirm that students had the same experience in programming, despite the fact they came from 5 different schools. This verification was needed before we executed the main experiment. In the case students did not have the same experience in programming due to external factors, this would have resulted in an additional variable and altered the outcome. The results for study 1 (p=0.150, df=3) and study 2 (p=0.706, df=2) retained the null hypothesis. This was a confirmation that students understood the game mechanics and had the same experience in programming in each study equally across all schools before different types of support was provided.

The second test (H2a, H2b) answers the main research questions of the paper about the effectiveness in learning for both tested types of support. In both studies the null hypothesis was rejected (Study 1: p=0.016, df=1, Study 2: p=0.008, df=1), resulting in the two groups having significant statistical differences in scores. The mean comparison in scores between the two groups for both cases concluded that in-game worked examples provide students with better support than textual instructions (Study 1: Table 5a; Study 2: Table 5b).

-Insert Table 5a here-
-Insert Table 5b here-

6. Discussion

Programming consists of structured problems with an initial state and goal which can be reached through logical operators (Jonassen, 1997). Students during problem-solving, are mentally searching for similar problems they encountered in the past. If a matching representation of the solution is found, it can be implemented and further enforced in the existing knowledge. If not, the student will continue to search for a solution and try new strategies (Gick, 1986). The theoretical cognitive structures where learners organize the schematics of a problem's solution are called schemata and by adapting them learning is achieved. However, problem-solving is an active process and consumes cognitive resources, impairing schema acquisition in students (Meier et al., 2008).

In order to assist students in this complex mental process of learning programming, analogous support is required. Most of the serious games about programming still provide support in the form of plain text and flashcards. Although it is by far the easiest way to implement it and requires no changes in the game design, this study tries to prove that it is also ineffective. Moreover, support is placed outside the game world, thus the main area of focus directs the user away and increases extraneous load. Especially for novice users, text instructions can potentially act as a distraction, since they are not interactive and do not provide visual cues of how programming structures function.

An alternative and more efficient approach in comparison with text instructions is the use of WE, which assists students in the acquisition of problem-solving schemata, while as a passive process requires minimum cognitive resources. Nonetheless, some studies suggest that WE do not provide goals to students and as such, they have negative motivation effects (Sweller & Cooper, 1985) (Lunenburg, n.d.). This paper proposes a support of in-game WE that can be studied before or after problem-solving and compares it with textual instructions in terms of learning efficiency. The in-game WE are part of the game world and this passively assists students to construct new schemata through an NPC. This process helps students to progress with the game and negates any negative motivation effects. In implementations like DungeonClass, support in the form of an NPC who acts inside the game world with strong visual representation, seems to have the immediate user's attention. NPC's actions, scripted or not, can be observed, borrowed and ultimately used by the user's programs. The NPC acts like an expert who provides a mental framework for users to follow (White, 2014). Examining the worked example, students actively devote part of their working memory to understand the behaviour of the new programming commands. The problem solving that follows, provides immediate feedback and verifies learning (Sweller, 2006). This issues the construction of new schemas in the brain, which reduces intrinsic load and results in increased learning efficiency (Paas & van Gog, 2006).

Both studies in the paper aimed to investigate the support provided to students through in-game WE. Specifically, both studies had the same research question, namely to test if in-game help has better learning results than textual instructions. The analysis concluded significant statistical differences between the two types of support in each case. In the first study, the NPC in-game support was tested by students that were novices in programming. Their intrinsic cognitive load was expected to be high since it was the first time they encountered sequencing and loops. The mean efficiency score (optimal = 7) of the NPC version was 4.632 compared to 4.191 of textual instructions. This indicates a 9.52% increased efficiency, which implies better understanding and application of new commands by students. The second study tested students with previous experience in programming, so a lower intrinsic load was expected. The mean efficiency score of the NPC version (5.352) was 11.192% greater than textual instructions (4.753). Considering that the two groups had different intrinsic loads, it is assumed that the encouraging results of the in-game worked examples derive from the reduction of extraneous load and optimization of germane.

The aforementioned results of our study agree with the empirical study of Zhi et al. (2018), according to which, both WE and erroneous WE, a form of WE similar to debugging, performed better than textual support by reducing extraneous load. However, they reported that the germane load in WE was increased because students received the information passively and did not have the opportunity to engage a problem immediately after the presentation. They proposed interweaving problem solving with WE, a method implemented and tested in our current study successfully. In the study by Spieler and Slany (2020), a serious game was used to test instruction efficiency between WE and text examples. Although the WE group finished the problem faster, results show no significant difference in the average scores. The game did not provide a problem along with the WE, which signifies the importance of borrowing principle and compliance with our findings. In another paper, Zhi et al. (2019) integrated WE into the programming environment Snap! and concluded that WE may have improved students’ learning efficiency when programming and had an effect on students’ intrinsic cognitive load. Considering that intrinsic load depends on the difficulty level of the problem, they suggest that the use of scaffolded prompts in their WE led to faster problem solution times. They pointed out that further research is needed to investigate why and how this effect occurs. In our approach, the NPC presents a WE by splitting the solution into smaller blocks of code in a similar manner to the above method. Smaller problems create less intrinsic load and although we did not specifically measure it, it seems to contribute beneficially to the additive total load.

7. Limitations

In order to carry out the study the serious game Dungeon class was implemented from scratch, since we were not able to find any game for programming that utilizes in-game worked examples. The game was inspired by Blocky Games and Hour of Code Minecraft serious games and includes the programming concepts of sequencing and loops. It is clear that in the second study where the participants were familiar with the aforementioned concepts, a game incorporating more advanced concepts, such as conditions and functions, could have been utilized if it was available. However, the results of the study showed that students with prior experience in programming performed better with the support provided by in-game worked examples in comparison with textual support, even in the case of sequencing and loop constructs that can be considered less challenging for them. The results of the study indicate that students, either novices or experienced in programming, would benefit even more from the dynamic nature of in-game worked examples in comparison with the static nature of textual support for more advanced programming concepts. However, this has to be confirmed by replicating the study with a game that supports such programming concepts.

The difficulty coefficient (D_i) of each level was another limitation as we considered all levels equal. During the level design, an effort had been made to keep the rate of difficulty steady while introducing new programming concepts and scaffold knowledge construction. Although problems were becoming more complex in the course of the game, players gained experience from previous solutions and had in-game help to overcome the current ones. This compensated for any possible increase in complexity and justified our decision to assign D_i with the value of 1 for all levels. Nonetheless, the level design process relied mostly on experience in similar games and observation of student reactions, so it is accountable for error.

8. Conclusion

The work presented is an ongoing project in the field of instructional supports for serious games. The study presented has contributed to the existing body of knowledge in the field in the following ways: it has provided a promising approach for incorporating worked examples in serious games for programming; it has shown that in-game worked examples provide better results than textual support for the programming concepts of sequencing and loops that are typically used in the first levels of Hour of Code games, both for students without and with prior experience on programming.

Furthermore, we proposed an equation Equation (1) for measuring student performance in block-based serious games about programming with district-level design. However, future research is needed to enforce and validate our findings. In-game WE should be tested on other programming concepts like variables, conditions, subprograms and data structures. Also, adding an instrument to measure the perceived cognitive load of students on both types of supports, will determine more accurately the results and allow better refinement. Investigating the impact of other variables, such as students' familiarization with technology (e.g. mobile phones and video games), on the way they prefer to receive information and consequently their performance would also be interesting. Finally, we plan to continue our study with adaptive NPC help as a game mechanic.

References

Abdul-Rahman, S.-S., & du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. Computers in Human Behavior, 30, 286–298. https://doi.org/10.1016/j.chb.2013.09.007

Andersen, E., O'Rourke, E., Liu, Y.-E., Snider, R., Lowdermilk, J., Truong, D., Cooper, S., & Popovic, Z. (2012). The impact of tutorials on games of varying complexity. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 59–68. https://doi.org/10.1145/2207676.2207687

Bellotti, F., Berta, R., & Gloria, A. D. (2010). Designing Effective Serious Games: Opportunities and Challenges for Research. International Journal of Emerging Technologies in Learning (IJET), 5(2010). https://www.learntechlib.org/p/44949/

Chandler, P., & Sweller, J. (1991). Cognitive Load Theory and the Format of Instruction. Cognition and Instruction, 8(4), 293–332. https://doi.org/10.1207/s1532690xci0804_2

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. Cognitive Science, 13(2), 145–182. https://doi.org/10.1016/0364-0213(89)90002-5

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem-solving transfer. Journal of Educational Psychology, 79(4), 347–362. https://doi.org/10.1037/0022-0663.79.4.347

Delacruz, G. C., Chung, G. K. W. K., & Baker, E. L. (2010). Validity Evidence for Games as Assessment Environments. https://repository.isls.org//handle/1/2891

DiMaggio, P. (1997). Culture and Cognition. Annual Review of Sociology, 23(1), 263–287. https://doi.org/10.1146/annurev.soc.23.1.263

EasyLogo. (n.d.). Retrieved 25 November 2021, from http://edu.fmph.uniba.sk/~salanci/EasyLogo/

Gick, M. L. (1986). Problem-Solving Strategies. Educational Psychologist, 21(1–2), 99–120. https://doi.org/10.1080/00461520.1986.9653026

Ginns, P., Martin, A. J., & Marsh, H. W. (2013). Designing Instructional Text in a Conversational Style: A Meta-analysis. Educational Psychology Review, 25(4), 445–472. https://doi.org/10.1007/s10648-013-9228-0

Gross, P., & Kelleher, C. (2010). Non-programmers identifying functionality in unfamiliar code: Strategies and barriers. Journal of Visual Languages & Computing, 21(5), 263–276. https://doi.org/10.1016/j.jvlc.2010.08.002

Hour of Code: Join the Movement. (n.d.). Code.Org. Retrieved 13 March 2021, from https://hourofcode.com/

Ichinco, M., Harms, K., & Kelleher, C. (2017). Towards Understanding Successful Novice Example Use in Blocks-Based Programming. Journal of Visual Languages and Sentient Systems, 3(1), 101–118. https://doi.org/10.18293/VLSS2017-012

Jonassen, D. H. (1997). Instructional design models for well-structured and III-structured problem-solving learning outcomes. Educational Technology Research and Development, 45(1), 65–94. https://doi.org/10.1007/BF02299613

Kalyuga, S., Chandler, P., Tuovinen, J., & Sweller, J. (2001). When problem solving is superior to studying worked examples. Journal of Educational Psychology, 93(3), 579–588. https://doi.org/10.1037/0022-0663.93.3.579

Kodu Game Lab | KoduGameLab. (n.d.). Retrieved 25 November 2021, from http://www.kodugamelab.com/about/

Laine, T. H., & Lindberg, R. S. N. (2020). Designing Engaging Games for Education: A Systematic Literature Review on Game Motivators and Design Principles. IEEE Transactions on Learning Technologies, 13(4), 804–821. https://doi.org/10.1109/TLT.2020.3018503

Learning from Examples: Instructional Principles from the Worked Examples Research—Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, Donald Wortham, 2000. (n.d.). Retrieved 20 January 2021, from https://journals.sagepub.com/doi/abs/10.3102/00346543070002181

Lunenburg, F. C. (n.d.). Goal-Setting Theory of Motivation. 6.

Magana, A., Vieira, C., & Yan, J. (2015). Exploring Design Characteristics of Worked Examples to Support Programming and Algorithm Design. The Journal of Computational Science Education, 6(1), 2–15. https://doi.org/10.22369/issn.2153-4136/6/1/1

Meier, D. K., Reinhard, K. J., Carter, D. O., & Brooks, D. W. (2008). Simulations with Elaborated Worked Example Modeling: Beneficial Effects on Schema Acquisition. Journal of Science Education and Technology, 17(3), 262–273. https://doi.org/10.1007/s10956-008-9096-4

Minecraft Hour of Code. (n.d.). Code.Org. Retrieved 13 March 2021, from https://code.org/minecraft

MIT App Inventor | Explore MIT App Inventor. (n.d.). Retrieved 25 November 2021, from https://appinventor.mit.edu/

Molnar, A., & Kostkova, P. (2013). On Effective Integration of Educational Content in Serious Games: Text vs. Game Mechanics. 2013 IEEE 13th International Conference on Advanced Learning Technologies, 299–303. https://doi.org/10.1109/ICALT.2013.94

Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive Load Theory and Instructional Design: Recent Developments. Educational Psychologist, 38(1), 1–4. https://doi.org/10.1207/S15326985EP3801_1

Paas, F., & van Gog, T. (2006). Optimising worked example instruction: Different ways to increase germane cognitive load. Learning and Instruction, 16(2 SPEC. ISS.), 87–91. https://doi.org/10.1016/j.learninstruc.2006.02.004

Patino, A., Romero, M., & Proulx, J. (2016). Analysis of Game and Learning Mechanics According to the Learning Theories. 2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES), 1–4. https://doi.org/10.1109/VS-GAMES.2016.7590337

Proulx, V. K. (2000). Programming patterns and design patterns in the introductory computer science course. ACM SIGCSE Bulletin, 32(1), 80–84. https://doi.org/10.1145/331795.331819

Scratch—About. (n.d.). Retrieved 25 November 2021, from https://scratch.mit.edu/about

Spieler, B., Pfaff, N., & Slany, W. (2020). Reducing Cognitive Load through the Worked Example Effect within a Serious Game Environment. 2020 6th International Conference of the Immersive Learning Research Network (ILRN), 1–8. https://doi.org/10.23919/iLRN47897.2020.9155187

Sweller, J. (2006). The worked example effect and human cognition. Learning and Instruction, 16(2), 165–169. https://doi.org/10.1016/j.learninstruc.2006.02.005

Sweller, J., & Cooper, G. A. (1985). The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. Cognition and Instruction, 2(1), 59–89. https://doi.org/10.1207/s1532690xci0201_3

Sweller, J., van Merrienboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive Architecture and Instructional Design. Educational Psychology Review, 10(3), 251–296. https://doi.org/10.1023/A:1022193728205

Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. (n.d.). Retrieved 12 December 2021, from https://unity.com/

White, M. M. (2014). Learn to Play: Designing Tutorials for Video Games. CRC Press.

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

Zhi, R., Lytle, N., & Price, T. W. (2018). Exploring Instructional Support Design in an Educational Game for K-12 Computing Education. Proceedings of the 49th ACM Technical Symposium on Computer Science Education, 747–752. https://doi.org/10.1145/3159450.3159519

Zhi, R., Price, T. W., Marwan, S., Milliken, A., Barnes, T., & Chi, M. (2019). Exploring the

Impact of Worked Examples in a Novice Programming Environment. Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 98–104. https://doi.org/10.1145/3287324.3287385

Table 1
Example of efficiency calculation

Game Level	Optimal solution (in blocks)	Student 1 solution (in blocks)	Student 1 efficiency (Di=1)	Student 2 solution (in blocks)	Student 2 efficiency (Di=1)
4	7	9	0.777	8	0.875
5	8	11	0.727	10	0.800
6	3	4	0.750	6	0.5
7	5	5	1	9	0.555
8	6	6	1	-	-
9	6	10	0.600	-	-
10	11	14	0.785	-	-

Note. The efficiency for the first student is 5.639 (0.777+0.727+0.750+1+1+0.600+0.785) and for the second student is 2.730 (0.875+0.800+0.5+0.555)

Table 2a

The school distribution of students in study 1

Schools	Frequency	Percent
1st Elementary school of Peraia	31	22.0
2nd Elementary school of Peraia	29	20.5
5th Elementary school of Peraia	54	38.3
72nd Elementary school of Thessaloniki	27	19.1
Total	141	100

Table 2b

The class distribution of students in study 1

Grade	Frequency	Percent
3rd	57	40.4
4th	84	59.6
Total	141	100

Table 3a

The school distribution of students in study 2

Schools	Frequency	Percent
2nd Elementary school of Peraia	57	38.0
5th Elementary school of Peraia	73	48.7
72nd Elementary school of Thessaloniki	20	13.3
Total	150	100

Table 3b

The class distribution of students in study 2

Grade	Frequency	Percent
5th	81	54.0
6th	69	46.0
Total	150	100

Table 4a

Score distribution on tutorial levels across schools in study 1

Tutorial levels 1-3	1st Elementary school of Peraia	2nd Elementary school of Peraia	5th Elementary school of Peraia	72nd Elementary school of Thessaloniki
Score means	2.954	2.898	2.938	2.936
StD	0.094	0.282	0.152	0.127

Table 4b

Score distribution on tutorial levels across schools in study 2

Tutorial levels 1-3	2nd Elementary school of Peraia	5th Elementary school of Peraia	72nd Elementary school of Thessaloniki
Score means	2.953	2.956	2.905
StD	0.114	0.155	0.209

Table 5a

Total score means of supports in study 1

Levels 4 - 10	NPC	Textual
Score means	4.632	4.191
StD	1.409	1.247

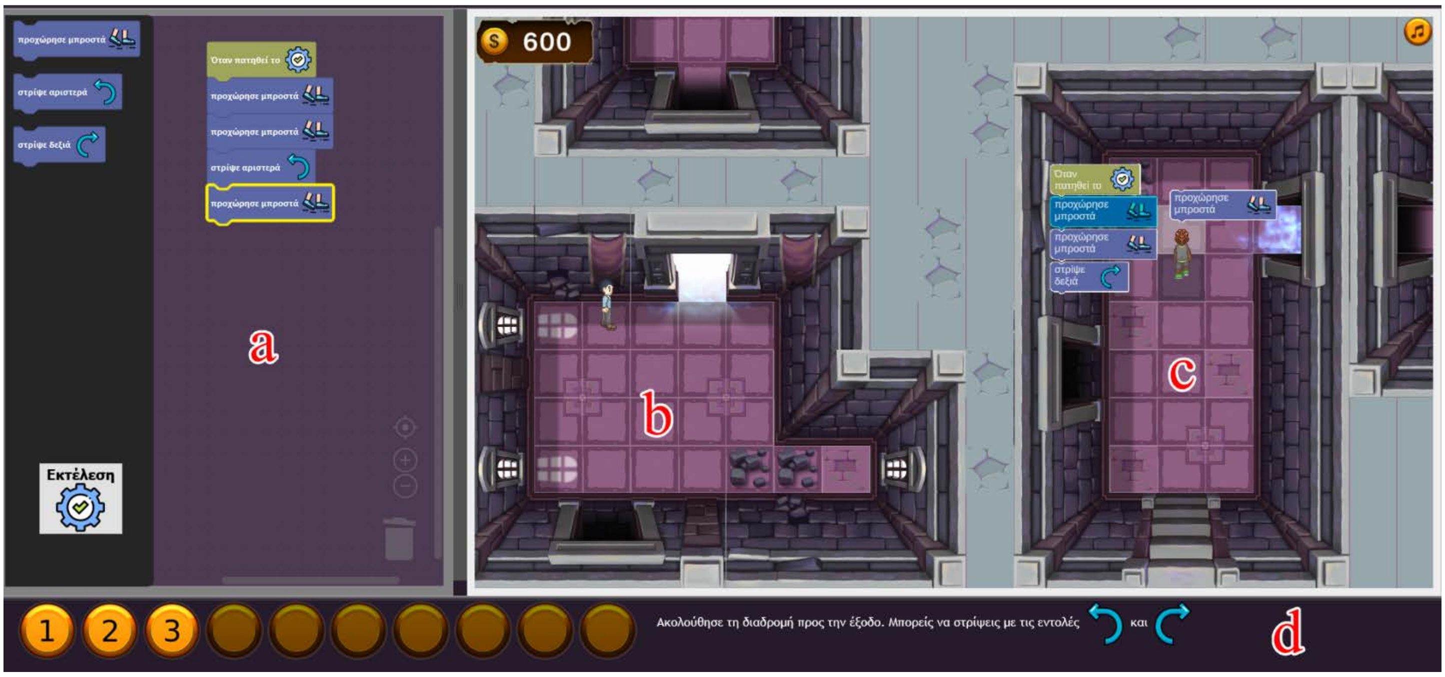
Table 5b

Total score means of supports in study 2

Levels 4 - 10	NPC	Textual
Score means	5.352	4.753
StD	1.269	1.257

Figure 1

A tutorial level: (1a) Programming area, (1b) Player puzzle, (1c) Worked example, (1d) Textual help

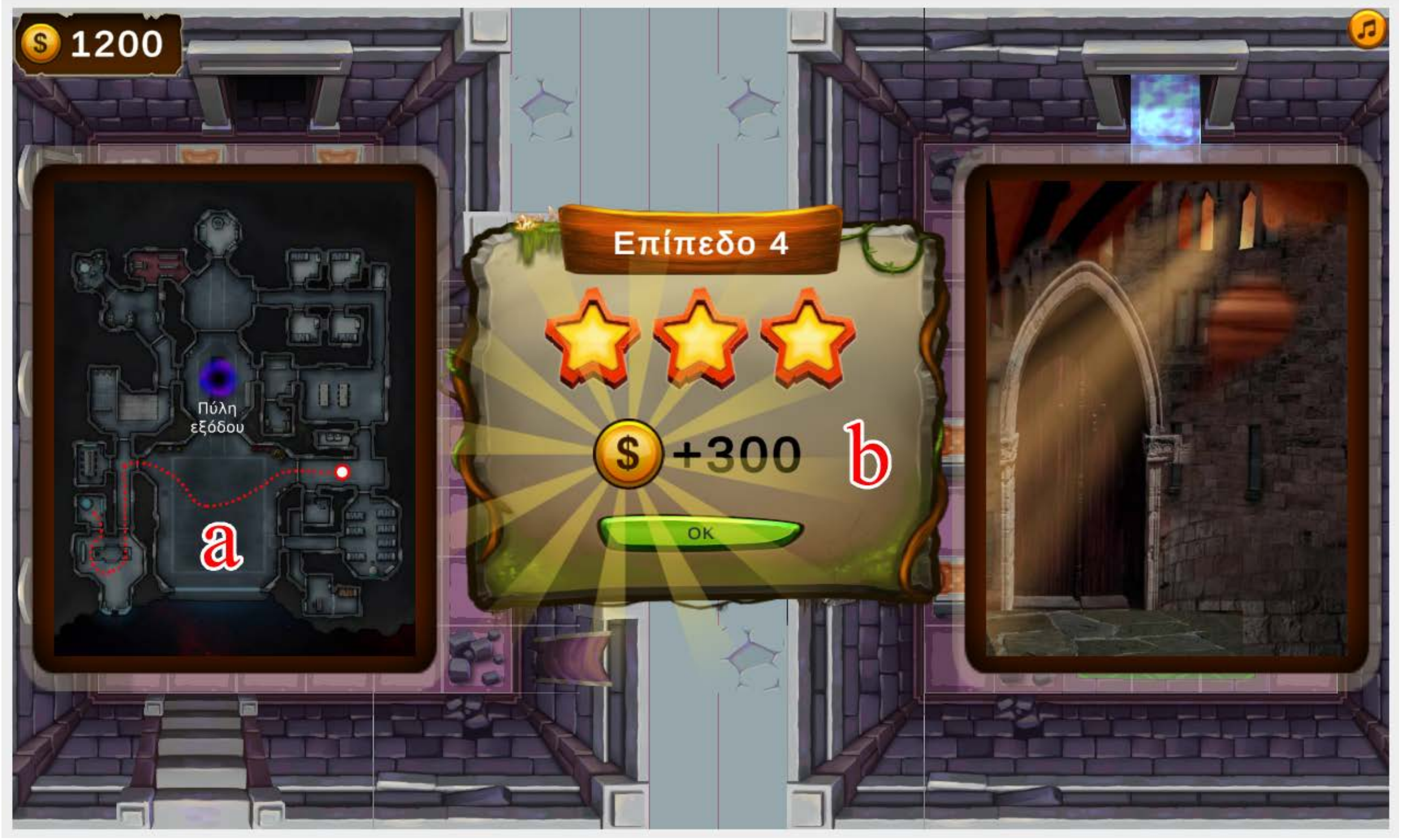


Ακολούθησε τη διαδρομή προς την έξοδο. Μπορείς να στρίψεις με τις εντολές και

Notes. The support text in the bottom right corner translates to: «Follow the path towards the exit. You can turn using the commands left and right».

Figure 2

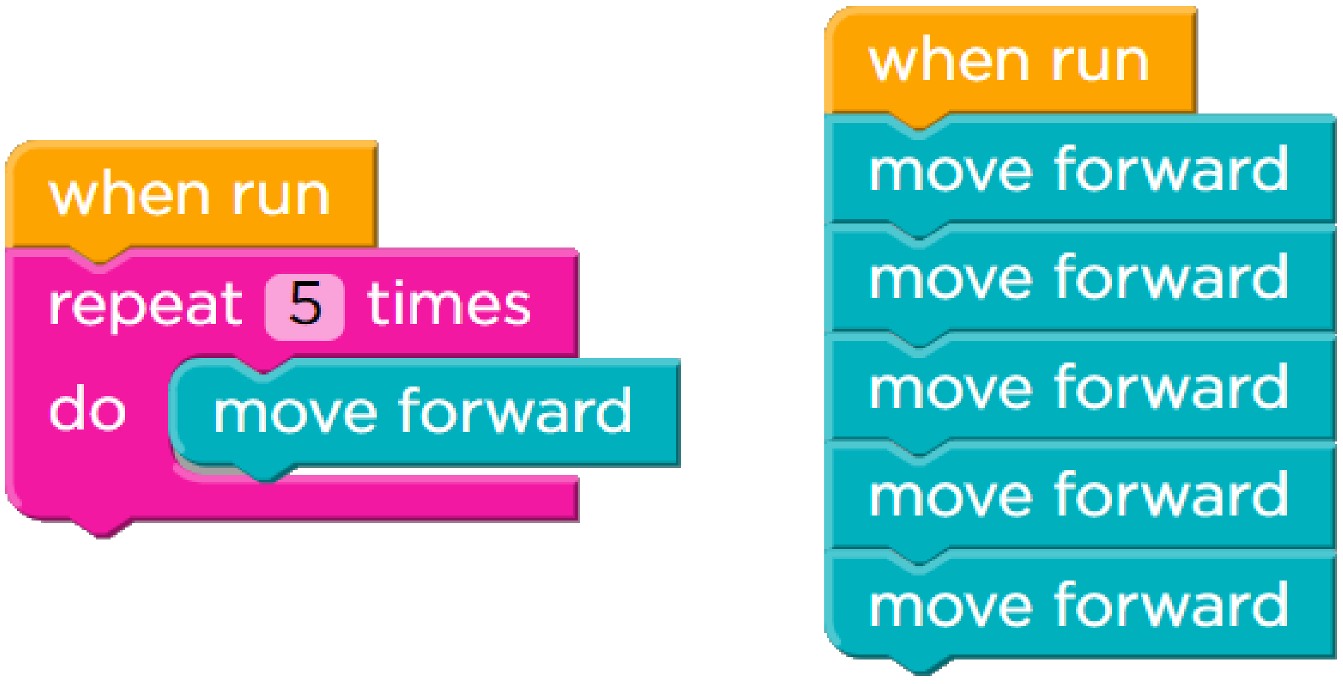
Winning message: (2a) Game room map, (2b) Score



Notes. The winning message after a successful solution. The text in the center translates to: «Level 4».

Figure 3

(3a) moving forward using a loop statement, (3b) moving forward using sequencing.

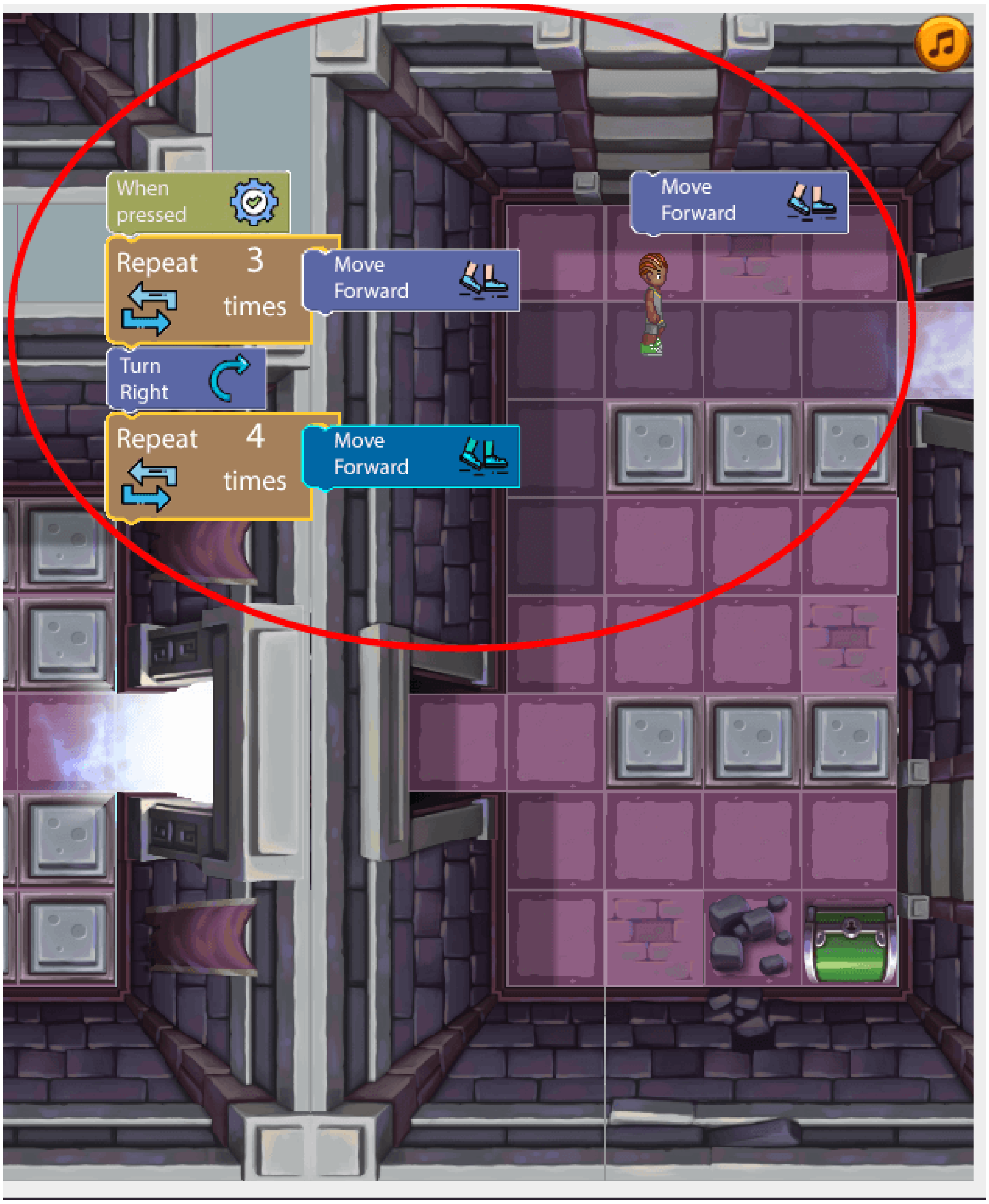


```
when run
repeat 5 times
do  move forward
```

```
when run
move forward
move forward
move forward
move forward
move forward
```

Figure 4.

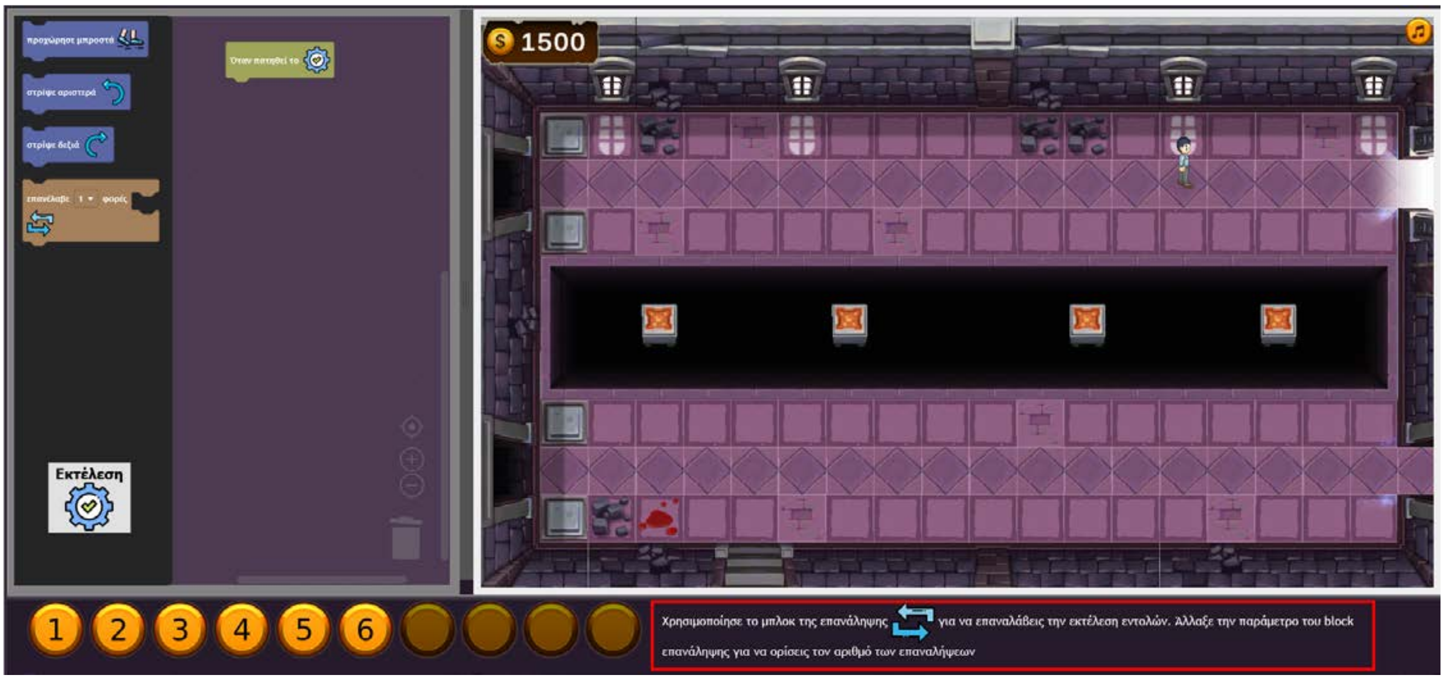
NPC executing a worked example



Notes. The command text has been translated to English for presentation purposes.

Figure 5

Textual instructions



Χρησιμοποίησε το μπλοκ της επανάληψης για να επαναλάβεις την εκτέλεση εντολών. Άλλαξε την παράμετρο του block επανάληψης για να ορίσεις τον αριθμό των επαναλήψεων

Notes. An example of textual instruction. The text in the bottom translates to: «Use the repeat block to repeat the execution of commands. Change the block parameter to set the number of iterations».