Adaptive support with working examples in serious games about programming

Abstract

Serious games are a growing field in academic research and they are considered an effective tool for education. Game-based learning invokes motivation and engagement in students resulting in effective instructional outcomes. An essential aspect of a serious game is the method of support for presenting the teaching material and providing feedback. A support design that evaluates students' progress and adapts accordingly, has the potential of producing better learning results. This paper presents an adaptive model based on fuzzy logic that adjusts the support acquisition according to student knowledge level. A serious game for teaching the concepts of sequence and iteration in programming to novice students was built to assess the model. It employs working examples as a support method since previous research indicated that it produced less cognitive load during problem-solving. An empirical study with 102 students has been conducted to evaluate the learning efficiency of the model. The analysis indicates positive results and a potential solution for balancing the amount of assistance in serious games.

Running head: ADAPTIVE SUPPORT IN SERIOUS GAMES

Pavlos Toukiloglou (corresponding author)
Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Street, GR-54636, Thessaloniki, Greece
toukiloglou@uom.edu.gr
https://orcid.org/0000-0003-4702-0159

Stelios Xinogalos
Department of Applied Informatics, School of Information Sciences, University of Macedonia, 156 Egnatia Street, GR-54636, Thessaloniki, Greece
stelios@uom.edu.gr
https://orcid.org/0000-0002-9148-7779

Keywords

Serious games, manual vs adaptive support, worked examples, programming, cognitive load theory

Stelios Xinogalos is a 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 120 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 support systems.

1. Introduction

Serious games (SGs) can be defined as software that merges a non-entertaining purpose with a video game structure (Felicia, 2011). Their main purpose is to train the player on a specific topic and they can be applied to a variety of fields like the military, health, and business sectors (Charsky, 2010). A SG is designed to be entertaining and attractive like a commercial game while maintaining learning as a primary objective. Students are motivated to play games and this incentive is considered the main determinant of effective learning (Westera, 2019). Therefore, education is a field in which SGs have been used and investigated in several studies with positive results (Kara, 2021). Many SGs have been developed for computer science learning focusing on programming, especially for novice users (Miljanovic & Bradbury, 2018).

Support systems are an essential part of SGs (Shannon et al., 2013) as they provide the necessary instructions to fulfill the required educational goals. Those systems can be non-interactive, presenting the instructional material with text and video, or interactive utilizing an Intelligent Tutoring System (ITS). An ITS (Crow et al., 2018) mimics the one on one tutoring model by adapting to the learner's performance and providing the appropriate response. A support mechanism can be more effective if the SG tracks the learning process of the user so it can provide the appropriate personalized feedback. This type of adaptability to different student learning profiles is difficult to be offered by human teachers in large classes (De Gloria et al., 2014). The learning value of those systems becomes even greater in situations where students need to study at home. The coronavirus pandemic forced millions of students around the world to receive formal education from home, challenging teachers, learners, and parents alike ('Impact of Coronavirus Pandemic on Education', 2020).

Adaptive learning environments for teaching programming have been developed (Crow et al., 2018) with positive results but mostly in the context of University courses. On the contrary, adaptive SGs implementations are scarce, employing various supports and techniques with few empirical studies. Additionally, an important problem in educational software design is balancing the amount of assistance that is given to students to optimize learning acquisition. This problem is often called the "assistance dilemma" (Koedinger & Aleven, 2007) and remains open in instructional science as a diversity of methods can be applied to provide guidance and feedback. This paper aims to explore the effects of adaptivity in a SG and how they relate to the assistance dilemma when the estimation of student knowledge level is taken into account.

In this study, we created a SG for teaching programming to novice students that implements an adaptive support system with worked examples (WEs). A WE guides the user by providing an expert solution to a similar problem and it is a very effective learning strategy, particularly for novice students (Sweller, 2006). The WEs have been used before as a support mechanism for teaching programming both in a SG (Toukiloglou & Xinogalos, 2022) and in a programming environment (Zhi et al., 2019) with positive results. This paper explores the impact of adaptive WEs on learning efficiency by conducting an empirical study on primary school students. The term efficiency can describe various aspects of measurement in educational procedures. In the context of our research, we define educational efficiency as the ability of a SG to assist students in learning programming. This can be estimated by examining students' solutions to a problem

and comparing the programming structures used to an expert solution. The study examines the following research questions:

RQ1: What is the effect of adaptive vs manual support on educational efficiency in a serious game about programming?

RQ2: How does adaptive support impact the amount of assistance received by students?

The rest of this paper is structured as follows: The next section presents an introduction to cognitive load theory and worked examples, followed by a review of the state of the art in adaptive support in serious games. After that, we present the serious game NanoDoc and its design process and continue with the adaptive support model that is based on fuzzy logic. Next, we examine the study methodology, following the results of statistical analysis. We conclude with a discussion of the results and the research limitations.

2. Related Work

2.1 Cognitive load theory and worked examples

Cognitive psychology studies internal mental processes and explores how the human brain and memory work. The computational metaphor (Ormerod, 1990) relates cognitive processes with operations performed inside a computer where the brain acts like a central processing unit (CPU). Learning can occur when new information flows as input in symbolic form and is processed by the brain through a series of cognitive procedures before being stored in long-term memory. According to Cognitive Load Theory (CLT), humans have finite working memory. In the course of a learning process, a cognitive load is produced from the required mental effort. The cognitive load is divided into three categories: intrinsic, extraneous, and germane. The intrinsic load is generated from the complexity of the learning activity, therefore the schemas must be acquired. The extraneous load relates to how tasks, designed procedures, and learning material are presented. Lastly, the germane load refers to the mental effort needed to acquire new schemata and create permanent knowledge in long-term memory (Paas et al., 2010). Every student has a maximum cognitive capacity and if that level is reached it will cause a cognitive overload and learning will cease. To optimize the learning process, the germane load should be maximized as it is the only one that is useful and contributes to learning (Clark et al., 2006).

When support is poorly designed it will generate extraneous load as the student will divert resources from the working memory to handle the insufficient instructional material. Since intrinsic and extraneous cognitive loads are additive (Paas et al., 2010), decreasing extraneous load is very important. That will free resources from working memory to handle the intrinsic load, hence providing more cognitive capacity for germane load and learning to take place. WEs can greatly contribute to that procedure since studies have shown that they decrease the extraneous load on working memory, reducing the overall cognitive load and producing better learning results (Sweller, 2006, 2011; Sweller et al., 1998). This is called the worked example effect and occurs when students are exposed to an expert solution to a problem allowing them to solve similar challenges by analogy (Magana et al., 2015).

The submitted programs are solutions to clearly defined problems also known as transformation problems since they require the learner to transform an initial state into an end goal (Jonassen, 1997). All elements of the problem are known to the learner and the correct solution requires the employment of specific knowledge domains. If the student has encountered a similar problem in the past, a representation of that knowledge called schema is already present and will lead to a solution (Gick, 1986). Schemas are cognitive concepts or patterns of thought of organized information stored in our long memory. WEs help students with the construction of schemas by providing passively all the required information

(Sweller & Cooper, 1985). During that process all the cognitive resources of the learner are directed into the study of the examples, formalizing the general solution strategy efficiently.

2.2 Adaptive support

The field of adaptive support in serious games is relatively new, so there are only a few implementations and empirical studies to validate the results. However, depending on the support method previous research showed that a diversity of systems had been employed to provide adaptiveness in game-based learning. A paper from Min (Min et al., 2014) discusses adaptive hint generation in ENGAGE, a game to teach computer science principles to middle-grade students. Their approach offered support by a data-driven technique that generated adaptive hints depending on the student and the problem being solved. The targeted guidance produced hints for students in a series of subgoals toward the solution instead of a single hint strategy referring exclusively to the end goal.

Hicks (Hicks et al., 2015) proposed automatically generated feedback in the form of hints in BOTS, a blocked-based puzzle game for teaching programming fundamentals to novice users. The feedback can be considered adaptive since hints were generated by observing the state of a game after each compilation of a student's solution. The method compares the current world state after a user submission with a state graph and, depending on the state transition type, classifies the provided hints. The state graph also referred to as an interaction network in the paper needs prior gameplay data to provide feedback for future students. However, Hicks (Hicks et al., 2014) mention that useful hints are generated even with a small amount of previous data.

Papadimitriou (Papadimitriou et al., 2019; Papadimitriou & Virvou, 2017) developed an online adventure game, HTML Escape, to teach the HTML language. The game consists of a series of rooms in a maze, with some of them containing quizzes. Each quiz correlates with a specific programming concept and the answers are used to update the player’s knowledge level of that concept. A fuzzy inference system builds a student model and dynamically controls the difficulty of the next quiz. Additionally, it controls the flow of the game by adding new rooms with items and non-player characters for the player to interact with. If the system detects a deficiency in a knowledge domain and the player keeps failing to complete a quiz, the game will adapt by providing tips and study material through game items. The feedback collected from students on a questionnaire survey had positive results. According to the responses, the game had excellent educational effectiveness by adapting quiz questions to student needs and offering further training while maintaining user motivation.

A more recent study by Hooshyar (Hooshyar et al., 2021) developed an adaptive educational game called AutoThinking to teach Computational Thinking. The adaptivity algorithm was based on a Bayesian network which dictated the support given to the player after the solution submission for each level. The system made non-invasive assessments and updated the player model from user actions and the game state. The feedback includes hints, images, or videos and depends on the student’s skill level. To evaluate the effectiveness of the proposed method, they conducted an empirical study with two groups of elementary school students. In the first group, the teaching material was presented with traditional methods utilizing a multimedia presentation whereas the second group followed the adaptive game approach. A pre/post-test analysis revealed that students who used AutoThinking outperformed the control group in knowledge gain. The authors argue that the main reason for this result was the individual support provided by the game.

A different approach was taken in Minerva, a game aiming to teach programming concepts to elementary school students (Lindberg & Laine, 2018). In Minerva, the content is adapted to the player's game and learning style to increase

engagement. Players were initially classified according to their play and learning styles with a questionnaire in the login process. The playstyle (Killer, Achiever, Socializer, Explorer) was dynamically updated during play from user interactions with the game world and affected various game elements such as the number of enemies or the obstacles. On the other hand, learning support adaptation was static. It was divided into four categories: Why (text), How (video), What (images), and Do (gameplay), and was displayed depending on the player's learning style (Activist, Reflector, Theorist, Pragmatist). The learning style retained the initial values from the questionnaire despite player actions and no adaptation occurred during playtime. A subsequent formative evaluation was conducted between a group of students who played the game and a group who studied the same concepts using handouts. It was noted that the sample size (64 students) was small. Nonetheless, the results showed a similar level of performance and retention between the two groups, although learning with the educational game was reported as more enjoyable.

According to the aforementioned serious games for programming, there is not a single approach to the implementation of adaptivity. Instead, a variety of models are employed to adapt different types of support as shown in Table 1. All studies reported positive results although not all of them had empirical data to back up those claims. As far as we know, no study to date has investigated neither the use of WEs as a support for an adaptive serious game about programming nor the potential differences in learning efficiency stemming from the way of receiving the support, namely manually vs adaptively. Previous research on a non-adaptive implementation revealed preferable learning results for WEs in comparison to textual support (Toukiloglou & Xinogalos, 2022). This paper will examine the impact of adaptive instructional support on learning efficiency and its association with the improvement of student assistance.

Table 1. Summary of adaptive serious games for programming.

Serious Game	Method of adaptation	Adaptation objective	Study
ENGAGE	Dynamic Bayesian network	Hints and task selection	(Min et al., 2014)
BOTS	Interaction Network	Hints	(Hicks et al., 2015)
HTML Escape	Fuzzy Knowledge State Definer	Hints and study material	(Papadimitriou et al., 2019; Papadimitriou & Virvou, 2017)
AutoThinking	Bayesian network	Hints and study material	(Hooshyar et al., 2019; 2021)
Minerva	Questionnaire classification	Game style	(Lindberg & Laine, 2018)

3. The game NanoDoc

The game was developed specifically for the study and encapsulates all the characteristics needed to explore the research questions. The design process is described in the following six steps:

1. Learning objectives definition: To teach the programming concepts of sequence, iteration, and condition.
2. Target audience definition: Novice students with basic or no previous knowledge of programming.
3. Game concept declaration: Players take the role of a doctor who was shrunk in nanoscale and has been transferred inside a sick creature to cure it. They explore a maze world, fight enemies, avoid traps, and solve programming puzzles to unlock the next rooms of the maze. The game will follow the third-person shooter conventions in terms of camera positioning and controls. Assistance will be provided by a Non-Player Character (NPC) who acts as a companion that guides and supports players.
4. Prototype creation: Developed in Unity engine (Unity Real-Time Development Platform | 3D, 2D VR & AR Engine, n.d.) based on a microgame template (*Unity Creator Kit: FPS*, 2021) provided freely by the engine.

5. Playtest and iteration: A group of 85 (41 male, 44 female) elementary students volunteered and played the game providing feedback and suggestions.
6. Final asset creation: Deployed as a WebGL application which can be accessed at the following link: http://users.sch.gr/pavlos/nanodoc.

The game consists of two modes, the exploration which takes place in a 3D world, and the puzzle which uses a mixed 2D/3D environment. During the exploration mode, users move inside a humanoid creature's body eliminating viruses with a special gun and avoiding obstacles or traps on their path (Figure 1). In this mode, the companion who takes the form of a red cell follows the player and provides useful tips about the game interface or upcoming events. The game world is a maze of corridors and rooms, some of them being dead ends. A mini-map in the screen's upper left corner mitigates disorientation and helps with navigation. Some of the rooms are sealed with a substance that key proteins can only dissolve. Each key is positioned strategically near the corresponding sealed door and users have to solve a programming puzzle to obtain it. There are 12 puzzles that teach specific programming concepts as the user gradually unlocks all doors and progresses throughout the game.

When the user finds a protein key, the game switches to a puzzle-solving mode as shown in Figure 2. The programming environment is block-based and consequently, each command is represented with a block. Users create solutions connecting the blocks together through a visual interface that is similar to the educational programming environment of Scratch (*Scratch - About*, n.d.). The command set is progressively made available as problems become more complex and new programming concepts are introduced. The puzzle consists of a grid on which the player’s avatar and the key are placed. For a successful solution, the avatar has to move through the grid as the program executes the user commands and stand in front of the key to grab it. Moreover, the puzzle grid contains obstacles that prevent direct access to keys, forcing the player to find alternative routes.

The game utilizes worked examples as instructional support. The adaptive technique described in the next section detects when the user needs help in understanding the concepts that have to be implemented for solving the current puzzle. In that case, the NPC offers a worked example with a solution to a similar problem. During that phase, the worked example replaces the current user solution in the grid with a new problem and a provided solution. The player can examine, edit and execute the worked example until s/he is confident that the programming concept was understood. When ready, users can switch back to their program and continue with their solution. If the support is activated for a current level, the above process of switching between the worked example and the user solution can be repeated freely until the goal is fulfilled.

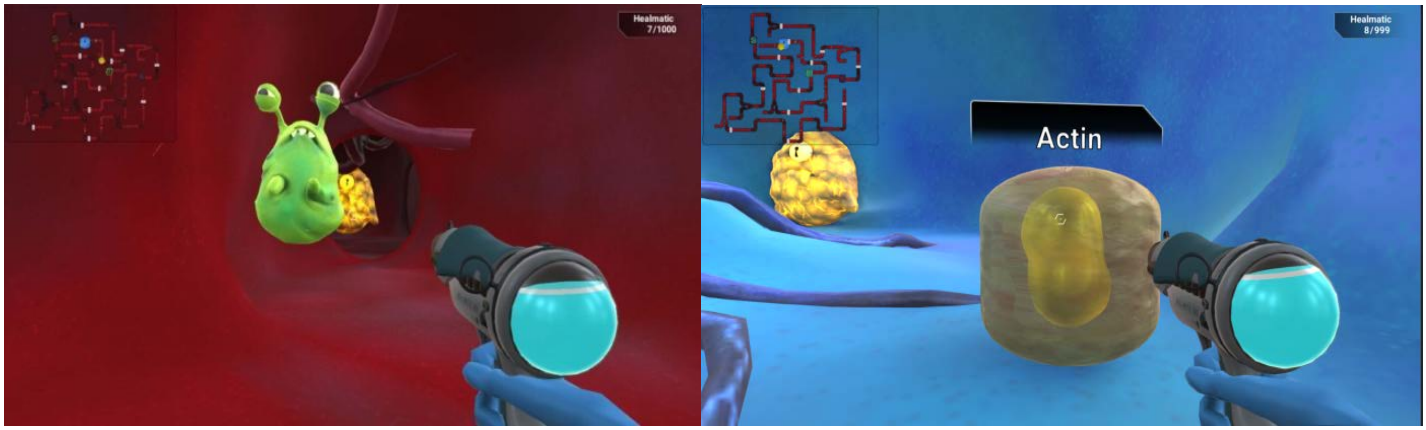

Figure 1. The 3D environment of the game.

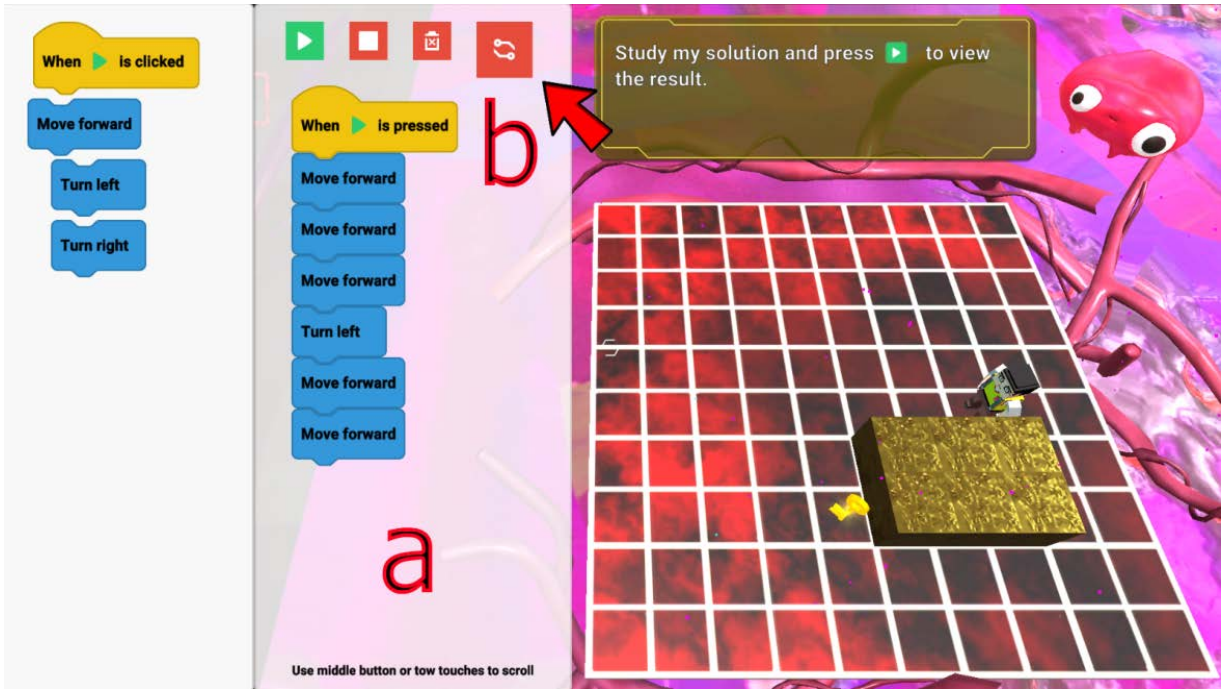

Figure 2. The puzzle mode of the game, (a) support is enabled (b) the WE button.

4. The adaptation model of NanoDoc

In this section the adaptation model of NanoDoc is presented. Since the adaptation model is based on fuzzy logic we first provide a brief overview of the underlying process and terminology and then we describe the implemented model.

4.1 Fuzzy Logic

The adaptive model utilizes fuzzy logic to represent student knowledge and adjust the support accordingly. Therefore, a brief description of the main principles of fuzzy systems is provided. Fuzzy logic was introduced by Zadeh (Zadeh, 1999) to address the concept of partial truth in a problem. In contrast to boolean logic in which the value of a variable can only be the integer 0 or 1, in fuzzy logic, the value can be a real number ranging between 0 and 1. As a consequence, the values of partial true or false allow the representation of imprecise data and non-numerical information. Fuzzy logic handles non-numeric values as linguistic variables that can be modified with adverbs. Thus, decision rules are formed similarly to natural language and mimic human decision processes. Fuzzy systems are rule-based and interpret data following a three-step approach. Firstly, the input data is fuzzified by assigning the numerical value to a set of functions called fuzzy sets. During this procedure, fuzzy sets estimate the degree of membership in the given data and output a linguistic expression as a verbal description. Secondly, all rules in the rule base are executed using IF-THEN

statements and logical operators. The last step called De-Fuzzification converts the fuzzy output to a crisp value so that it can be used to make a decision. Depending on the system, there are several defuzzification strategies with the most common being the maxima and centroid methods. Fuzzy logic has many real-world applications with main uses in control systems and artificial intelligence.

Fuzzy logic was selected for the development of the adaptive algorithm for a number of reasons. Firstly, it is a data-driven method and this approach allows the algorithm to process player data during gameplay. In contrast, model-based methods such as intelligent tutoring systems or artificial neural networks require training before execution (Hooshyar et al., 2019). Additionally, model-based methods have a higher cost of development considering factors such as paid expert knowledge, complexity, and the amount of data needed. As already discussed (Table 1) a diversity of other methods were implemented to adapt learning content like the Bayesian network, Questionnaire classification, and Hint factory. A common characteristic of all the mentioned methods is that they are also data-driven approaches. However, fuzzy logic is able to handle partial truth statements, and that allows it to mimic the uncertainty of modeling student knowledge and resemble real-life support in teaching.

4.2 Adaptation model

The adaptation model of NanoDoc extends previous work from the game FuzzEg (Papadimitriou et al., 2019) and the Fuzzy Knowledge State Definer model (Chrysafiadi & Virvou, 2015). The referred papers applied Fuzzy Logic to describe student knowledge as a way to counterbalance the uncertainty of learning representation. This uncertainty derives from the fact that student knowledge can not always be defined with accuracy as it can be partially known or the provided data for its calculation is imprecise. The adaptation model consists of the steps shown in Figure 3:

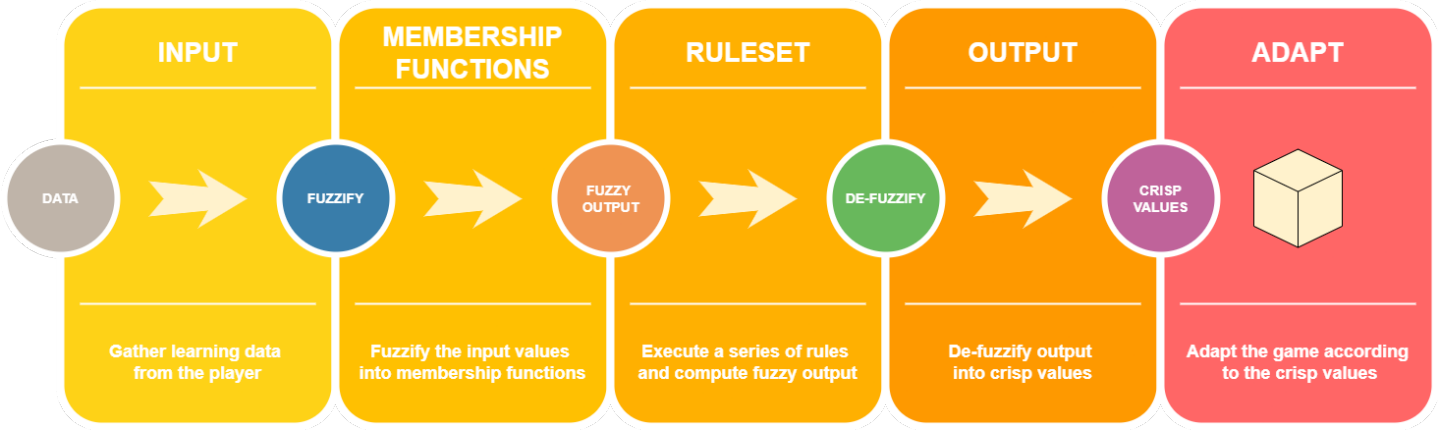

Figure 3. The adaptation model.

Although the main structure of our model is similar to the aforementioned (Chrysafiadi & Virvou, 2015), we had to adjust several aspects to comply with our programming environment and game characteristics. In our game, students solve programming puzzles in a block-based environment. It is expected that data generated from this process will make up the input source of the model. However, selecting the suitable metric is critical during a solution submission as it can alter the adaptation results. We reject measuring the time of puzzle completion because it will penalize students who prefer to experiment and explore different strategies before submitting their final submission. Allowing players the freedom to make mistakes in a risk-free environment when they encounter new concepts is a practice that leads to more

efficient tutorials (Gee, 2005). Another candidate metric is to verify if the puzzle goal is met and the player reached the designated destination. This technique is very common and has been used in the past in various games or academic research (Andersen et al., 2012). Even though this approach intuitively feels simple and correct, it is subject to a severe problem due to the nature of programming. Some puzzles of the game teach students the iteration structure and attempt to display the benefits of its use. Nevertheless, since iteration is defined as a repetition of statements, the puzzles can also be solved as a series of sequence statements, bypassing iteration completely. If the model relied only on checking if the player reached the level goal for calculating the student's knowledge of iteration in those puzzles, it would predict the wrong values. Figure 4 shows an example of two solutions to a puzzle that although both reach the end goal, only one is correct regarding code efficiency/quality.

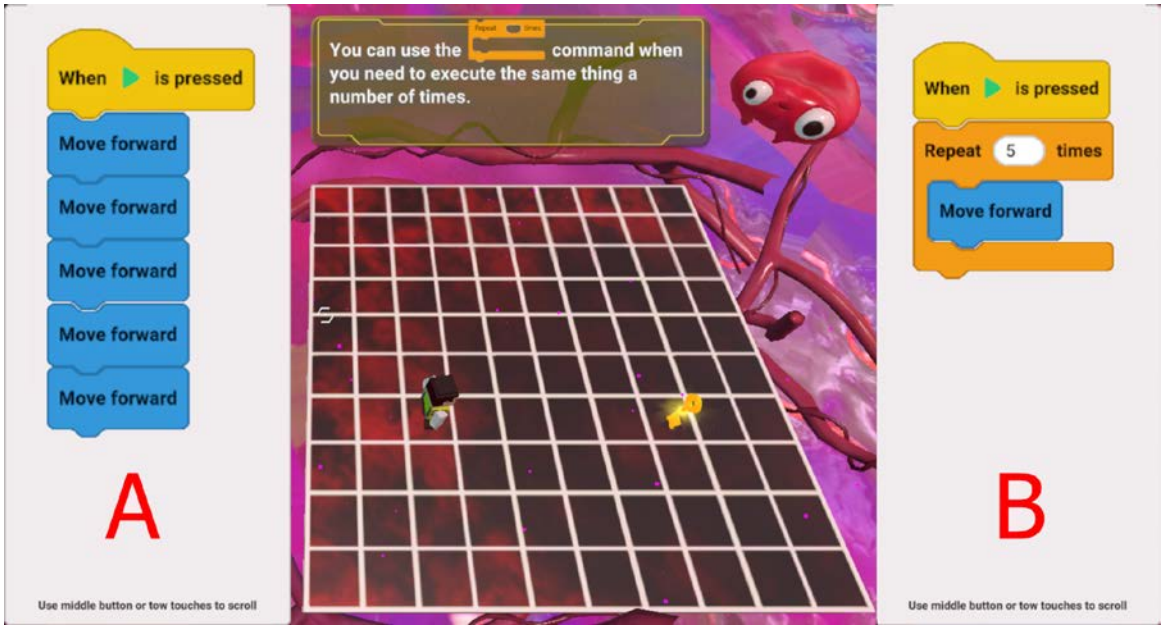

Figure 4. An example of two solutions to a puzzle: (A) uses the sequence structure, (B) uses the iteration which is more efficient programmatically.

In a previous study (Toukiloglou & Xinogalos, 2022) we used Equation (1) to calculate the learning efficiency of supports in a block-based game. Efficiency is calculated by dividing the number of blocks of the optimal solution by the number of blocks of the solution committed by the student. A result is a decimal number indicating the degree of code efficiency in terms of programming structures used. The maximum value is 1 which signifies the best solution, hence the maximum learning efficiency achieved by the student. The puzzle difficulty coefficient (D) is equal to 1 for all puzzles in this game. This is because the solution to each puzzle was based on the student's accumulated knowledge of the previous ones. Furthermore, to maintain the overall difficulty steady, when a new programming structure or technique was introduced, the complexity of the problem was decreased.

$$\text{Efficiency} = \frac{OptimalSolution}{StudentSolution} \times D \qquad (1)$$

The model utilizes the concept of knowledge domains to define the learner's competence on a specific topic. Our game includes puzzles for three domains: sequence, condition, and iteration. Each game puzzle corresponds to a specific domain as shown in Table 2.

Table 2. Knowledge domain of each level puzzle.

Level	Knowledge Domain
1, 2, 3, 4	Sequence
5, 6, 7, 10, 11, 12	Iteration
8, 9	Condition

To estimate the current knowledge level of a student on a domain, firstly the algorithm calculates the efficiency score of the current puzzle solution. Then, during the fuzzify step, that number is received as input by the next four membership functions:

$$\mu\text{Un}(x) = \begin{cases} 1, & x < 45 \\ 1 - \frac{x-45}{5}, & 45 < x < 50 \\ 0, & x \geq 50 \end{cases}$$

$$\mu\text{Uk}(x) = \begin{cases} \frac{x-45}{5}, & 45 < x < 50 \\ 1, & 50 \leq x \leq 65 \\ 1 - \frac{x-65}{5}, & 65 < x < 70 \\ 0, & x \leq 45, x \geq 70 \end{cases}$$

$$\mu\text{K}(x) = \begin{cases} \frac{x-65}{5}, & 65 < x < 70 \\ 1, & 70 \leq x \leq 85 \\ 1 - \frac{x-85}{5}, & 85 < x < 90 \\ 0, & x \leq 65, x \geq 90 \end{cases}$$

$$\mu\text{L}(x) = \begin{cases} \frac{x-85}{5}, & 85 < x < 90 \\ 1, & 90 \leq x \leq 100 \\ 0, & x \leq 85 \end{cases}$$

The fuzzy output describes the knowledge level of the student on a domain as a quadruplet (μUn, μUK, μK, μL) of the following fuzzy sets:

- Unknown (Un): The domain concept is unknown to the player. The efficiency score is from 0.0 to 0.5
- Unsatisfactory Known (Uk): Player has some knowledge of the concept. The efficiency score is from 0.45 to 0.7
- Known (K): Player knows the concept sufficiently. The efficiency score is from 0.65 to 0.9
- Learned (L): Player knows the concept completely. The efficiency score is from 0.85 to 1.0

Every quadruplet complies with the following rules which originate from the fuzzy set definitions and the fact that the knowledge level on a domain can not exceed 100%.

- The value of each quadruplet component is limited to the range from 0 to 1, μU n(x), μUK(x), μK(x), μL(x) [0, 1]
- The sum of all quadruplet components is equal to 1, μUn(x) +μUk(x) + μK(x) + μL(x) = 1
- Each quadruplet can have up to 2 non zero values and they must be adjoined

In the Defuzzification process, a crisp output value is computed in the form of a linguistic term. The fuzzy set is converted to a defuzzified value with the Maximum-Membership method. Through this technique, the biggest of all values in the quadruplet defines the knowledge level of the reviewed domain. According to the above description, a

classification example of a student who completes a puzzle corresponding to the sequence knowledge domain with an efficiency score of 0.86 will result in a fuzzy output of (0,0,0.8,0.2). This translates to the concept being 80% Known and 20% Learned and results in the linguistic value of Known.

The procedure of evaluating student knowledge level is repeated after each solution submission throughout all the programming puzzles of the game. Thus, the model can update the estimations on knowledge domains and adapt the support activation. Each domain evaluation re-estimates student knowledge level taking into account the previous and current levels. The initial knowledge level values are set by players during the game setup or otherwise, the default value of unknown is given. To calculate the knowledge level, previous and current values are compared to estimate the cognitive state of the student. If the current knowledge level is greater than the previous in a domain then this fact is interpreted as progress in learning and the current value becomes the new knowledge level. The puzzle structure of the game is created in such a way that each level envelops and advances previously acquired knowledge. Therefore, when a student solves a puzzle with a high-efficiency score, it is logical to assume that the most recent calculated knowledge level is valid since previous puzzles required equal or less expertise on the domain to be solved.

On the contrary, if the most recent knowledge level estimation is less than the previous, that translates to knowledge insufficiency of a new command or programming structure utilization. Considering that the main objective of the adaptive algorithm is to provide support, detecting an insufficiency in a new concept has a higher priority than previous knowledge on a domain. Hence, the current knowledge level is estimated according to Table 3.

Table 3. Evaluation of knowledge level in the case that the current value is less than the previous one

Previous knowledge level	Current knowledge level	Output
Learned	Known, Unsatisfactory known	Known
Learned	Unknown	Unknown
Known	Unsatisfactory known	Unsatisfactory known
Known	Unknown	Unknown
Unsatisfactorily known	Unknown	Unknown

Additionally, this approach solves the potential problem of decreased sensitivity to knowledge estimation due to the previous results, especially in the early game where puzzles are relatively easier as new concepts are introduced. However, the aforementioned method assumes that each new concept on a domain is introduced through previously established knowledge, scaffolding learning in a predetermined manner.

After a knowledge level estimation, the game adapts the support according to the ruleset shown in Table 4. The ruleset takes into account the failed attempts of the student over a single puzzle solution. Support triggers after at least one solution attempt, allowing students to probe the problem even if the current knowledge level in that domain is unknown. After the first submission, support activation is analogous to the number of attempts and the knowledge level of the domain to which the puzzle is related.

Table 4. Support activation ruleset.

Knowledge domain level	Failed attempts
Learned	3
Known	2
Unsatisfactory Known, Unknown	1

Finally, support can be presented passively by the system without a direct association with the adaptation model. When the student is confronted with a new programming structure or the knowledge level of that structure is inadequate, it is possible to completely be stumped. In that case, the system can detect inaction after 120 seconds and activate the support even before the first solution attempt. The inactivity duration was measured empirically by the authors during test runs of the game in a group of students not participating in the empirical study. The following collection of three cases demonstrates the effects of support during gameplay. Each example of support activation was extracted from anonymous student data and refers to different students with compatible characteristics on the same puzzle level.

The first case (Table 5) refers to level 5 of the game where players are introduced to the iteration structure which is a new concept and so the knowledge level of both students is set to unknown. Student A received support after the first failed attempt and by exploring the working example s/he submitted an optimal solution. Student B did not ask for support and solved the puzzle using only sequential logic instead of the iteration block command, registering a much lower efficiency score and missing an opportunity to advance learning on a new domain.

Table 5. Case 1, Knowledge level: Unknown, Knowledge domain: Iteration, Puzzle level: 5.

Support activation	Student	Number of submitted attempts			Score efficiency on submitted solution
		Before support is presented	After support is presented	Total	
Adaptive	Student A	1	1	2	1
Manual	Student B	2	-	2	0.4

The second case (Table 6) refers to level 7 of the game that requires a combination of sequence and multiple iteration structures to be constructed for a successful solution. Both students had a fair knowledge of the domain and submitted the same, although not a perfect solution. The first student received support after two failed attempts while the second asked manually after the fourth attempt. The working example had the same effect on both students as in each case a successful solution was submitted after its presentation. We argue that a much-extended delay in support reception might lead to frustration or unnecessary mental exhaustion.

Table 6. Case 2, Knowledge level: Known, Knowledge domain: Iteration, Puzzle level: 7.

Support activation	Student	Number of attempts			Score efficiency on submitted solution
		Before support is presented	After support is presented	Total	
Adaptive	Student A	2	1	3	0.88
Manual	Student B	4	1	5	0.88

The third case (Table 7) refers to level 10 of the game which introduces the concept of nested loops which is generally considered a difficult subject for novice programmers. The game, taking into account the knowledge level of Player A, accepted three failed attempts before offering help, allowing time for experimentation with various strategies. This delayed feedback resulted in a stronger effect of the provided support on the efficiency of the solution submitted by student A in comparison with that of student B. The latter needed three attempts after the support presentation and moreover submitted a less efficient solution.

Table 7. Case 3, Knowledge level: Learned, Knowledge domain: Iteration, Puzzle level: 10.

Support activation	Student	Number of attempts			Score efficiency on submitted solution
		Before support is presented	After support is presented	Total	
Adaptive	Student A	3	1	4	1
Manual	Student B	1	3	4	0.45

5. Study Methodology and Results

5.1 Study design

The empirical study was conducted in a computer science lab of a Greek elementary school. A teacher was administering the procedure during a 45-minute game session for each class. Before starting, students received an introduction to the user interface, game objectives, and mechanics. Throughout the experiment, students relied solely for support on the game as the administrator was not allowed to give any kind of assistance with puzzle solutions. The game provided two types of support, one in the form of worked examples and the other as adaptive worked examples. Upon initialization of the application, the type of support was randomly assigned to each student. The support is accessible from a button, clearly distinctive in terms of size and color, placed in the upper right corner of the designated programming area (Figure 2). Both types of support display the same predetermined WE according to the puzzle level. The WE presents a solution that can be examined, edited, and executed for a similar problem to the current puzzle. Although the function of both types of support is identical, they deviate in availability. The first one is always visible and students can use it when they need assistance. However, in the adaptive worked examples, the support button remains disabled and activates only when the adaptive algorithm concludes a student learning deficiency in a domain. This setup allows the direct comparison of specific metrics between the two types of supports by registering their impact on logged learning analytics.

The first metric we used is the total efficiency score of all individual levels per user and it- is computed by Equation (2) which is a variation of Equation (1) we discussed in section 4. This allows us to measure the efficiency of puzzle solutions in terms of programming and consequently estimate the knowledge level of students after the game completion in the taught programming structures. Users who manage to progress further into the game will have better total efficiency scores than the ones with the same knowledge level but fewer completed puzzles. However, simply advancing through the puzzles will not guarantee a higher total score since each distinct puzzle score affects the total. Table 8 shows a computational example for equation (2) of two users who completed 7 and 5 levels respectively.

Table 8. The computation of educational efficiency for two users. The solutions are measured in command blocks.

Level	Number of blocks for optimal solution	Number of blocks of user 1 solution	Number of blocks of user 2 solution	User 1 efficiency	User 2 efficiency
1	3	3	3	1	1
2	5	5	5	1	1
3	8	8	16	1	0.5
4	4	10	11	0.4	0.36
5	9	18	9	0.5	1
6	8	12	-	0.67	-
7	12	12	-	1	-
Total efficiency				5.57	3.86

$$\text{Efficiency score} = \sum_{i=1}^{n} \frac{OptimalSolution_i}{StudentSolution_i} \times D_i \quad (2)$$

Additionally, we examined the correlation between the efficiency score and the total number of supports received during the play session per user. Since in both cases of manual and adaptive groups the support was in the form of a WE, it can be safely concluded that it had the same impact on players regarding the submitted programming solution. However, considering that in the manual support selection group, users chose when they received help from the game, analysis is needed to explore how it affected their progress.

5.2 Participants

The game focuses on supporting novices in learning the basic concepts of programming through adaptive support provided in the form of working examples. Consequently, in order to investigate the impact of the game and its support system it has to be evaluated by novice programmers. Additionally, the game is designed to be appealing to younger ages and the programming interface mimics block-based environments used in primary education. Taking into account the aforementioned characteristics, the best candidates for the study were students in primary school. Consequently, the study was conducted with 102 elementary students, 53 males and 49 females aged from 9 to 12 years old. All students attend the same school, have access to computer labs, and follow the same curriculum in computer science (CS). They are distributed in 3 classes as shown in Table 9. The students played the game in the context of their CS course carried out according to the school timetable. In terms of programming experience, all study subjects were novices. They were taught sequencing programming with the educational programming environment Scratch (Scratch - About, n.d.) and EasyLogo (*EasyLogo*, n.d.). In addition, they participated in an EU Code Week (Europe Code Week, n.d.) event organized by the school, resulting in a preliminary introduction to the iteration structure.

Table 9. The class distribution of students.

Grade	Frequency	Percent
4th	33	32.35
5th	39	38.23
6th	30	29.41
Total	102	100

5.3 Data Collection

Participants were under the age of 18 so their parents or legal representative had to provide signed consent about the collected data and their usage. The study design and the consent form were approved by the Committee for Research Ethics of the University of Macedonia during the planning stage of our study. The form indicated the limited rights, terms, and conditions that were given to our team to handle the data and also information about the research. Data was obtained anonymously and automatically in real-time by the application during the game session. It was stored in an online database in a JSON (JavaScript Object Notation) format for easier processing and statistical analysis. User record information consisted of grade, date, type of support, and learning data. The database was updated after a successful solution with the user actions, the knowledge level per programming structure, and the calculated current efficiency score.

5.4 Data analysis and Results

The Kolmogorov-Smirnov test was used to check for normality in the knowledge efficiency scores of the sample. The results for the manual support group, $D(53) = 0.024$, $p < .05$, and for the adaptive support group, $D(49) = 0.039$, $p < .05$, indicated that the data was not normally distributed in both groups. Therefore, the non-parametric method of Mann-Whitney was conducted to test the following null hypothesis:

H_o: The efficiency scores of players that had been provided with adaptive support will be similar to the ones that chose the support manually

The Mann-Whitney U test indicated that the difference between the two groups was statistically significant ($U=405$, $z=-5.989$, $p = 0.00 < .05$) and the null hypothesis was rejected. It revealed that the efficiency scores were significantly greater in the adaptive support group ($Md=10.33$, $n=49$) compared to the manual support group ($Md=8.00$, $n=53$).

Analysis of the highest level reached by students for manual support ($M=8.00$, $SD=2.40$) and adaptive support ($M=11.00$, $SD=1.30$) revealed that students in the adaptive support group managed to complete more levels of the game. The adaptive support group ranged significantly higher (min=7, max=12) than manual support (min=4, max=12). Figure 5 displays the distribution of the maximum level reached by students per support type.

A Pearson correlation coefficient was computed to assess the linear relationship between the efficiency score and the number of times support was used per user. There was a positive correlation between the two variables, $r(102) =.37$, $p =.00$. Also, the means comparison of the total number of supports used for manual support ($M=1.68$, $SD=1.68$) and adaptive support ($M=2.90$, $SD=1.31$) showed that the adaptive support group received more help during the game. We must note that the support is registered once per level, although the student can revisit a support as many times as needed.

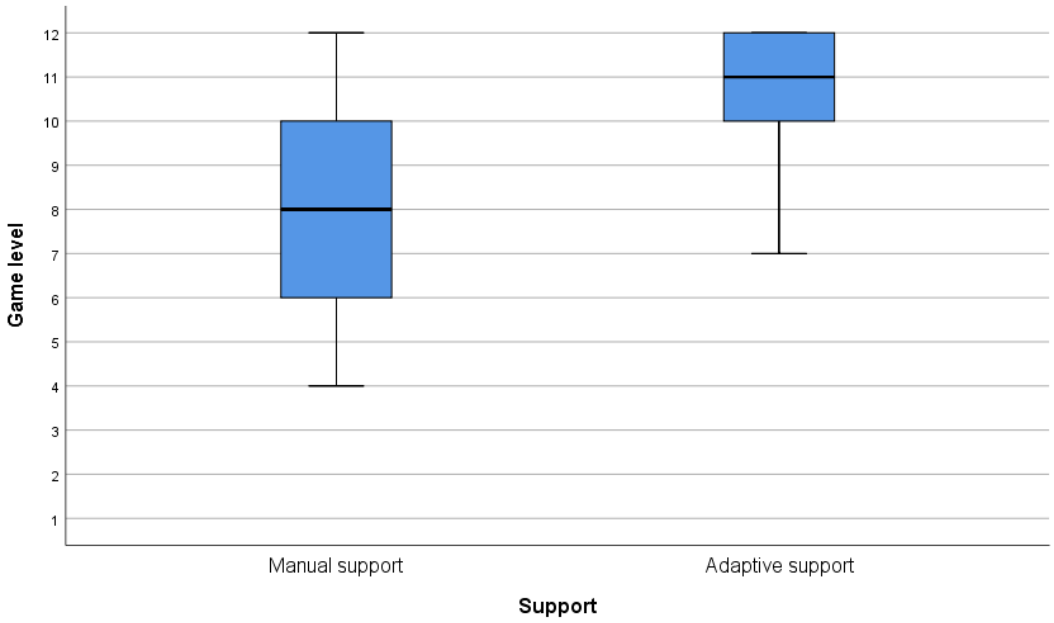

Figure 5: Distribution of highest level completion per support type

6. Discussion

Support is an essential part of a serious game as it provides the means to achieve its learning targets. There are three possible approaches for a serious game concerning the availability of the given support. In the first type, the support is shown as an introduction with tips and instructions on how to solve the problem at hand. The support is triggered every time the user encounters a new problem regardless of whether it is needed or not. This situation may cause frustration to some students who either possess the required knowledge or could have derived it on their own (Shannon et al., 2013). An alternative is to let the student decide when to seek support from the game. Although this technique seems logical and intuitive there are some drawbacks deriving mostly from user behavior patterns. Ryan (Ryan et al., 2001) analyzed why students avoid seeking help and concluded that students who are focused on their reputation prefer to solve a problem without external support. Even though the research was referring to the classroom, the argument could be valid since subjects remain the same. Asking for help might be considered evidence to themselves and others that they lack the ability to solve the puzzle or comprehend the current programming function. Another reason for not receiving help is that students might simply neglect its existence. Also, a complex user interface design with incorrect placement of components and inappropriate graphical elements could cause the support icons to be overlooked.

The last approach of support presentation is when it is determined by the game based on the user knowledge level. Shannon (Shannon et al., 2013) in their study on effective practices in in-game tutorial systems, mention the importance of well-timed feedback in intelligent tutoring systems. In manual support, users could ask for support in their first failed attempt without fully understanding the problem. Adaptive support by tracking the knowledge level of users could let them experiment with solutions and provide help when it is needed in order for it to be more effective. This delayed feedback can promote better self-regulated learning skills, such as error detection and self-correction (Corbett & Anderson, 2001). We applied this principle in our adaptive implementation by disabling help in the first submission even when the user knowledge level for the current structure was unknown.

The results of our study agree with the aforementioned observations as they indicate that the educational efficiency of the adaptive support group was greater than the manual (adaptive Md = 10.33, n = 49, manual Md=8.00,

n=53) (RQ1). Furthermore, by examining the relationship between educational efficiency and the amount of support received by students we found a positive correlation (RQ2). This implies that the reason the adaptive group performed better was that it received more support from the game, an assumption that was confirmed by statistical data (manual support M=1.68, SD=1.68, adaptive support M=2.90, SD=1.31). Additionally, players in the adaptive support group (M=11.00, SD=1.30) completed more levels than the manual support group (M=8.00, SD=2.40). The vast majority of players with adaptive support reached the challenging final two puzzles of the game. On the contrary, players in the manual support group dropped out of the game earlier, possibly due to frustration or lack of guidance. We argue that adaptive support can result in better educational efficiency than the manual by ensuring the presence of help at the correct time.

7. Limitations

The empirical study was conducted with primary school students so the results refer to this specific age group. Although it is possible that other age groups could have a similar response to adaptive WEs we do not have data to back up this claim. Future research could explore the adaptive impact on secondary school or higher education students. Also, the study did not track time-related data. As a consequence, comparisons such as time spent during game and puzzle modes or time for solution submissions between manual and adaptive supports could not be done. Finally, it should be noted that although it was not observed during the experiment, some of the students might have difficulties with the nature of the gameplay between the puzzles. Traps, enemies or even the maze design could prevent or significantly delay players from progressing throughout the game and reaching new puzzles. Future research may validate the study results with different game genres or strip gameplay entirely by focusing exclusively on programming puzzles.

8. Conclusion

In this paper we presented an empirical study on adaptive support in a serious game about programming. The game NanoDoc was created according to the research specifications and an adaptive method based on fuzzy logic was implemented. A formula (2) was used to calculate the educational efficiency of two groups of primary school students. Both groups utilized WEs as a support method during the play session to minimize cognitive load and differ only on how the support was initiated. In the first group, the choice of when to receive help was upon the students whereas in the second it was determined by the adaptive algorithm. Results indicate that the adaptive support had higher educational efficiency as it provided help more often and with better accuracy. We conclude with a recommendation to serious game designers about programming for novice users. We propose to apply adaptive WEs as a support method since it has a great potential of succeeding in the educational goals while taking into account the cognitive load.

Acknowledgment

This work is part of a project that has received funding from the Research Committee of the University of Macedonia under the Basic Research 2020-21 funding program.

References

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

Charsky, D. (2010). From Edutainment to Serious Games: A Change in the Use of Game Characteristics. *Games and Culture*, *5*(2), 177–198. https://doi.org/10.1177/1555412009354727

Chrysafiadi, K., & Virvou, M. (2015). Fuzzy logic for adaptive instruction in an e-learning environment for computer programming. *IEEE Transactions on Fuzzy Systems*, *23*(1), 164–177. https://doi.org/10.1109/TFUZZ.2014.2310242

Clark, R. C., Nguyen, F., Sweller, J., & Baddeley, M. (2006). Efficiency in learning: Evidence-based guidelines to manage cognitive load. *Performance Improvement*, *45*(9), 46–47. https://doi.org/10.1002/pfi.4930450920

Corbett, A. T., & Anderson, J. R. (2001). Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement, and attitudes. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 245–252. https://doi.org/10.1145/365024.365111

Crow, T., Luxton-Reilly, A., & Wuensche, B. (2018). Intelligent tutoring systems for programming education: A systematic review. *Proceedings of the 20th Australasian Computing Education Conference on - ACE '18*, 53–62. https://doi.org/10.1145/3160489.3160492

De Gloria, A., Bellotti, F., & Berta, R. (2014). Serious Games for education and training. *International Journal of Serious Games*, *1*(1). https://doi.org/10.17083/ijsg.v1i1.11

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

Europe Code Week. (n.d.). Retrieved 2 April 2022, from https://codeweek.eu/about

Felicia, P. (Ed.). (2011). *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*. IGI Global. https://doi.org/10.4018/978-1-60960-495-0

Gee, J. P. (2005). Learning by Design: Good Video Games as Learning Machines. *E-Learning and Digital Media*, *2*(1), 5–16. https://doi.org/10.2304/elea.2005.2.1.5

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

Hicks, A., Peddycord, B., & Barnes, T. (2014). Building Games to Learn from Their Players: Generating Hints in a Serious Game. In S. Trausan-Matu, K. E. Boyer, M. Crosby, & K. Panourgia (Eds.), *Intelligent Tutoring Systems* (Vol. 8474, pp. 312–317). Springer International Publishing. https://doi.org/10.1007/978-3-319-07221-0_39

Hicks, D., Dong, Y., Zhi, R., Cateté, V., & Barnes, T. (2015). BOTS: Selecting next-steps from player traces in a puzzle game. *CEUR Workshop Proceedings*, *1446*.

Hooshyar, D., Yousefi, M., & Lim, H. (2019). A systematic review of data-driven approaches in player modeling of educational games. Artificial Intelligence Review, 52(3), 1997-2017.

Hooshyar, D., Pedaste, M., Yang, Y., Malva, L., Hwang, G. J., Wang, M., Lim, H., & Delev, D. (2021). From Gaming to Computational Thinking: An Adaptive Educational Computer Game-Based Learning Approach. *Journal of Educational Computing Research*, *59*(3), 383–409. https://doi.org/10.1177/0735633120965919

Impact of Coronavirus Pandemic on Education. (2020). *Journal of Education and Practice*. https://doi.org/10.7176/JEP/11-13-12

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

Kara, N. (2021). A Systematic Review of the Use of Serious Games in Science Education. *Contemporary Educational Technology*, *13*(2), ep295. https://doi.org/10.30935/cedtech/9608

Koedinger, K. R., & Aleven, V. (2007). Exploring the Assistance Dilemma in Experiments with Cognitive Tutors. *Educational Psychology Review*, *19*(3), 239–264. https://doi.org/10.1007/s10648-007-9049-0

Lindberg, R. S. N., & Laine, T. H. (2018). Formative evaluation of an adaptive game for engaging learners of programming concepts in K-12. *International Journal of Serious Games*, *5*(2), 3–24. https://doi.org/10.17083/ijsg.v5i2.220

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

Miljanovic, M. A., & Bradbury, J. S. (2018). A Review of Serious Games for Programming. In S. Göbel, A. Garcia-Agundez, T. Tregel, M. Ma, J. Baalsrud Hauge, M. Oliveira, T. Marsh, & P. Caserman (Eds.), *Serious Games* (Vol. 11243, pp. 204–216). Springer International Publishing. https://doi.org/10.1007/978-3-030-02762-9_21

Min, W., Mott, B., & Lester, J. C. (2014). Adaptive Scaffolding in an Intelligent Game-Based Learning Environment for Computer Science. *The Second Workshop on AI-Supported Education for Computer Science*, 41–50.

Ormerod, T. (1990). Human Cognition and Programming. In *Psychology of Programming* (pp. 63–82). Elsevier. https://doi.org/10.1016/B978-0-12-350772-3.50009-4

Paas, F., Gog, T. van, & Sweller, J. (2010). Cognitive Load Theory: New Conceptualizations, Specifications, and Integrated Research Perspectives. *Educational Psychology Review*, *22*(2), 115–121. https://doi.org/10.1007/s10648-010-9133-8

Papadimitriou, S., Chrysafiadi, K., & Virvou, M. (2019). FuzzEG: Fuzzy logic for adaptive scenarios in an educational adventure game. *Multimedia Tools and Applications*, *78*(22), 32023–32053. https://doi.org/10.1007/s11042-019-07955-w

Papadimitriou, S., & Virvou, M. (2017). Adaptivity in scenarios in an educational adventure game. *2017 8th International Conference on Information, Intelligence, Systems and Applications, IISA 2017*, *2018-Janua*, 1–6. https://doi.org/10.1109/IISA.2017.8316453

Ryan, A. M., Pintrich, P. R., & Midgley, C. (2001). Avoiding Seeking Help in the Classroom: Who and Why? *Educational Psychology Review*, 22.

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

Shannon, A., Boyce, A., Gadwal, C., & Barnes, T. (2013). Effective practices in game tutorial systems.

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. (2011). CHAPTER TWO - Cognitive Load Theory. In J. P. Mestre & B. H. Ross (Eds.), *Psychology of Learning and Motivation* (Vol. 55, pp. 37–76). Academic Press. https://doi.org/10.1016/B978-0-12-387691-1.00002-8

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

Toukiloglou, P., & Xinogalos, S. (2022). Ingame Worked Examples Support as an Alternative to Textual Instructions in Serious Games About Programming. Journal of Educational Computing Research, 60(7), 1615–1636. https://doi.org/10.1177/07356331211073655

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

Unity Creator Kit: FPS (n.d.). Retrieved 12 December 2021, from https://assetstore.unity.com/packages/templates/tutorials/creator-kit-fps-149310#description

Westera, W. (2019). Why and how serious games can become far more effective: Accommodating productive learning experiences, learner motivation and the monitoring of learning gains. Journal of Educational Technology & Society, 22(1), 59-69.

Zadeh, L. A. (1999). Fuzzy Logic = Computing with Words. In L. A. Zadeh & J. Kacprzyk (Eds.), Computing with Words in Information/Intelligent Systems 1: Foundations (pp. 3–23). Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-1873-4_1

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