# Introducing Secondary Education Students to Programming through Sound Alerts

Theofani S. Sklirou, Areti Andreopoulou, Anastasia Georgaki, and Nikolaos D. Tselikas

***Abstract* — It is considered hard to teach programming in secondary education while achieving the aims of curriculum. However, when teaching is supported by suitable methodologies, learning can be ameliorated. Under this premise, this paper discusses different teaching approaches to programming in secondary education and examines the potential benefit of sound-alerts as a complementary teaching tool. Such alerts were created by pairing different sound stimuli to specific programming actions and operations. Both the selection of the sound stimuli, as well as the potential impact of the use of sound alerts on programming are evaluated through subjective studies. Results showed that participants preferred synthesized to natural (pre-recorded) stimuli for all types of alerts. It was also revealed that users prefer sound-alerts associated to pending actions, errors, successful code execution, conditional statements and code looping over alerts highlighting the step-by-step execution of the code. Finally, the test results showed that students understand both the meaning and the use of code commands more clearly when they use a sound-enriched programming environment instead of a conventional one. These results were the motivation for the initial creation of an audio and voice messages' data base and the initial design of a new comprehensive educational tool using sound.**

***Index Terms* — Programming teaching, Psychoacoustic perception, Secondary education, Sound-alerts.**

## I. Introduction

According to a popular definition, programming is the process of writing, testing, debugging/troubleshooting, and maintaining the source of code of computer programs [43], [47]. Computer science is rapidly becoming critical for generating new knowledge and should be taught as a distinct subject or content area, especially in secondary schools [19].

In Greece, programming courses were added to the secondary education curriculum 25 years ago. Since then, students have been confronted with problems concerning their communication with the machine through coding, which involves a precise way of thinking realized through specific syntax [6], [23], [40], [53]. Nevertheless, while the most common difficulties/misconceptions students encounter when learning how to program have been identified, clear strategies for addressing them still remain to be established [16].

High-school students should be taught programming concepts independently of specific applications and programming languages [49], [50]. They all have different needs and difficulties, which can be divided into 5 categories [15]: 1) orientation: discovering the usefulness and benefits of programming 2) notional machine (the general properties of the machine): Realizing how the behaviour of the physical machine relates to the notional machine 3) notation: includes problems related to syntax and semantics 4) structures: understanding the schemas or plans that can be used to reach small-scale goals (e.g., using a loop) 5) mastering the pragmatics of programming: learning the skill to specify, develop, test and debug a program using the available tools.

Pea [36] has identified some persistent, conceptual, language-independent "bugs" in how novices program and understand programs. Students believe that computers "go beyond the information given" in the program. In addition, it has been observed that several students fail to "translate" a conceptual solution to a problem into correct code [55]. The reason might be that students are not trained to transform mental intuitions into code [55]. Such obstacles could be overcome by helping students develop problem-solving skills in addition to logical reasoning.

Artificial languages have a limited vocabulary compared to natural ones and teachers use natural language to decode and communicate the meaning of programming operations [15]. On the other hand, it has been shown that multimodal interactions facilitate the understanding of programming concepts operations [52], [54]. The most common practices in schools involve environments with audiovisual feedback [5], [26]. This paper explores the use of sound-alerts as a complementary tool to programming courses and discusses its potential impact on the students' development of problem solving skills. Specifically, the rest of this paper is organized as follows: Section 2 presents an overview of educational tools, teaching approaches, difficulties in programming learning and environments using sound. Section 3 presents the introduction of a new sound-based learning approach, analyzes, and presents the selection of the utilized sound stimuli. In Section 4, we present the psychoacoustic evaluation study of six teaching scenarios, which use sounds-alerts as a complementary tool for teaching computer programming. The experimental results and the effectiveness of sound-alerts are depicted in Section 5 and conclusion and future work are summarized in Section 6.

Published on December 30, 2020.
Theofani S. Sklirou, Department of Informatics and Telecommunications, University of Peloponnese, Tripoli, Greece.
(e-mail: sklirou@uop.gr)
Areti Andreopoulou, Laboratory of Music Acoustics and Technology (LabMAT), National and Kapodistrian University of Athens, Greece.
(e-mail: aandreo@music.uoa.gr)
Anastasia Georgaki, Laboratory of MusicAcoustics and Technology (LabMAT), National and Kapodistrian University of Athens, Greece.
(e-mail: georgaki@music.uoa.gr)
Nikolaos D. Tselikas, Department of Informatics and Telecommunications, University of Peloponnese, Tripoli, Greece.
(e-mail: ntsel@uop.gr)

## II. Educational Tools and Teaching Approaches

Coding is part of the logical thinking, but programming learning is difficult for secondary school students, because there are difficulties in program writing, locating, and designing, as we will present below, after the presentation of the overview of educational tools and teaching approaches, which used in education.

### *A. An Overview of Educational Tools*

The first attempts to present programming in a more engaging and effective way started in the early '70s [34], [29]. Among the most popular were Logo and its derivatives [34]. Some of these proposals promoted the use of visual or virtual programming languages and the simulation of dynamic auralization of the program execution [8], [33].

The educational platform LEGO Mindstorms NXT of the homonymous company Lego simulates almost all modern automation and approaches automatic control systems through a creative and enjoyable learning environment. Its use includes both programming and simple mechanical character applications [39], [11].

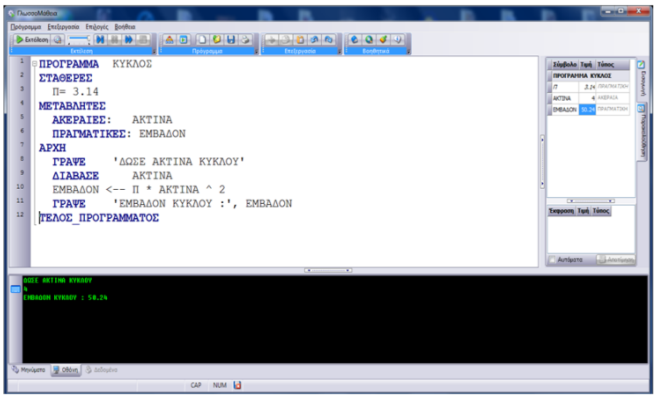
Pascal is an easy programming language, for teaching structured programming techniques [3], a first contact with the algorithmic programming. It is easy to learn programming concepts and commands.

C is a powerful programming language with many possibilities but on the other hand it has a difficult syntax. C provides a limited number of keyword and a large number of arithmetic and logical operators. C is a difficult programming language regarding both teaching and learning, because of its native minimalism [27].

Python is a widely used high-level, general-purpose, dynamic programming language [2]. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Students learn programming or start programming learning by using Ardiuno and build entire projects for automation and robotic applications [35]. Arduino software development environment based on Wiring, a C-like open source programming language, but it is easier in programming understanding [38]. Research studies carried out in last ten years look at the positive effect of robots, on children perception and education [51].

Glossomatheia *(Γλωσσομάθεια* in Greek*)* is a programming environment (using pseudocode), based on the Pascal programming language, used mainly in secondary education (high school) in Greece [48], [4], [17]. It is a training package, developed with a focus on laboratory support courses related to the cultivation of algorithmic and analytical thinking and the development of methodological skills for students. During program editing and execution, the screen is full of information. It is possible to present flow charts, but a lot of messages that are displayed on the screen, are not perceived by students, e.g., waiting for data entry, logical data, successful data entry etc. [48]. Glossomatheia environment (Fig. 1) does not support any sound-alerts.



```
ΠΡΟΓΡΑΜΜΑ  ΚΥΚΛΟΣ
ΣΤΑΘΕΡΕΣ
  Π= 3.14
ΜΕΤΑΒΛΗΤΕΣ
  ΑΚΕΡΑΙΕΣ:   ΑΚΤΙΝΑ
  ΠΡΑΓΜΑΤΙΚΕΣ: ΕΜΒΑΔΟΝ
ΑΡΧΗ
  ΓΡΑΨΕ    'ΔΩΣΕ ΑΚΤΙΝΑ ΚΥΚΛΟΥ'
  ΔΙΑΒΑΣΕ     ΑΚΤΙΝΑ
  ΕΜΒΑΔΟΝ <-- Π * ΑΚΤΙΝΑ ^ 2
  ΓΡΑΨΕ    'ΕΜΒΑΔΟΝ ΚΥΚΛΟΥ :', ΕΜΒΑΔΟΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

| Σύμβολο | Τιμή | Τύπος |
|---|---|---|
| ΠΡΟΓΡΑΜΜΑ ΚΥΚΛΟΣ | | |
| Π | 3.14 | ΠΡΑΓΜΑΤΙΚΗ |
| ΑΚΤΙΝΑ | 4 | ΑΚΕΡΑΙΑ |
| ΕΜΒΑΔΟΝ | 50.24 | ΠΡΑΓΜΑΤΙΚΗ |

| Έκφραση | Τιμή | Τύπος |
|---|---|---|

```
ΔΩΣΕ ΑΚΤΙΝΑ ΚΥΚΛΟΥ
4
ΕΜΒΑΔΟΝ ΚΥΚΛΟΥ : 50.24
```

Fig. 1. The Glossomatheia programming environment.

### *B. Teaching Approaches*

The traditional teaching approach has been to learn specific programming languages, which is a closed and teacher-centered behavioral model. Effectiveness depends on the amount of knowledge that the teacher has on the subject.

According to [25], there are as many ways to learn as there are students. In the context of these efforts, the idea of programming by discovery emerges. According to this model, the students discover their mistakes and misunderstandings, and the role of the student is upgraded.

Another model is the constructive one, where knowledge is not transmitted, but built. The actions and needs of each student are the center of gravity.

In the new teaching approaches, the elements-principles assimilation of exploratory and collaborative learning is sought. The aim of the new model is to improve the learning process and to overcome the obstacles that arose during the implementation of the previous teaching approaches. The process consists of five distinct but repetitive stages: Questioning, active research, creation, discussion, evaluation.

The research of Hsu & Hwang [24] has been based on this methodology. We relied on this teaching approach and the above new combinatorial model. In our study, the course unit "basic concepts of structured programming" (serial, selective, and repeated structure) in high school was adopted for conducting the experiment using sound in our case.
The following research questions were investigated:

- Is there relevant software for better understanding and learning algorithmic programming?
- What are the weaknesses of the existing programs?
- What is the originality of the new proposal?
- What is the purpose and contribution of the new proposal?
- What are the pedagogical benefits of the new approach?
- What are the perceptions of the students who learn programming with sound?
- Do the students who learn with the structured sound-based programming show better learning than those who learn with the conventional teaching without sound?

Our research questions and the recording of the difficulties preceded, and we developed the following learning model that will be described below, with the cooperation of our students.

The basic ideas of Piaget are adopted, according to which learning is not transmitted but built through the active cooperation between the teacher and the students, and the students with each other. It is also based on J. Bruner's discovery theory, according to which students discover knowledge (search) through discovery processes such as the execution of a ready-made program with different input data through testing and experimentation.

### C. *Difficulties in Programming Learning*

Difficulties in programming understanding are observed in all educational environments. The difficulties encountered by students in programming learning are related to:

a) the misunderstanding of the basic structures,

b) the rules of programming language,

c) the properties of the "machine" that the students control and

d) the ability to determine, develop, control, and debug a program with the available tools [56].

More specifically, regarding the properties of the "machine", students [14] cannot easily identify the relationship between the program (written in a programming language) and its execution mechanism. They usually make their own assumptions about how the computing system works internally. They cannot realize the degree of detail, required by programming and the rigor with which the rules of the programming system must be followed.

Sometimes students [57] have difficulty with English words that include programming languages because they misinterpret them. Some words have a different meaning in the programming language than the current English language. Moreover, the standard language use is difficult to interpret by students, who are unaware of the mechanisms by which the Information System works.

The most common category of misconceptions among students has its source in everyday life. Students interpret the commands in a programming language, with reference to everyday conversations between people – which are often elliptical, metaphorical, etc.

They fail to define the output condition of a loop successfully. They cannot determine the range of the loop and its beginning – end, as well as the repeated commands.

It’s difficult for them to understand: the way that variables work, the concept and role of the variable (algorithmic and programming variable) have been some of the subjects of extensive research (for example [45], [31], [7], [42], [44], [13], [41]). The variables’ use to enter data misinterprets. They interpret the "immobility" of the screen as a kind of fault or in any case as a kind of problem. They find it difficult to explain the movement of information (keyboard input-data entry and storage in a memory location A).

### D. *Environments Using Sounds*

The ear plays a fundamental role during information processing, as it is the main source of information to our brain. The ear selects the surrounding sounds, but also those of our own voice, converts them into electrical stimuli, and transmits them to the brain, which processes and analyzes them.

Applications have developed that use sound to develop programs, like Scratch, Sonic Pi and MAX/MSP, to understand computational thinking, like Tune Pad and to simulate actions, like Peep.

An educational tool for programming, employing audio cues is Scratch. Scratch is used in education and entertainment [46]. The system offers support for music blocks controlling Loop Sets, play-back etc. [28]. Study findings reveal that educational games add to the fun element in learning, and students rated the game as an effective way to learn programming and enhance their programming skills. Students could easily relate game elements to difficult programming constructs [30].

Sonic Pi is a live coding environment, originally designed to support both computing and music lessons in schools, developed by Sam Aaron and his group at the Computer Laboratory of University of Cambridge [1]. Thanks to the use of the Super Collider platform and of an accurate timing model, it is also used for live coding and other forms of algorithmic music performance and production, including “algoraves”. The Sonic Pi programming environment is user-friendly, where the students learn music by writing code and programming using music. Sonic Pi includes a multitude of commands and structures that allow students to develop musical compositions and/or understand algorithmic structures, as well.

The MAX / MSP is a high-level programming environment (Fig. 2), mainly used by composers, musicians, performers. It is used to build interactive music and multimedia applications in real time and gives direct audio-visual effect [12]. Over its more than thirty-year history, MAX/MSP is a modular environment, with most routines existing as shared libraries. An application programming interface allows third-party development of new routines. Thus, it is characterized by a large community that supports and enhances it with both commercial and non-commercial extensions. Because of its extensible design, which represents both the structure of the program as well as its graphical user interface, it has been described as the lingua franca for developing interactive music performance software [37].

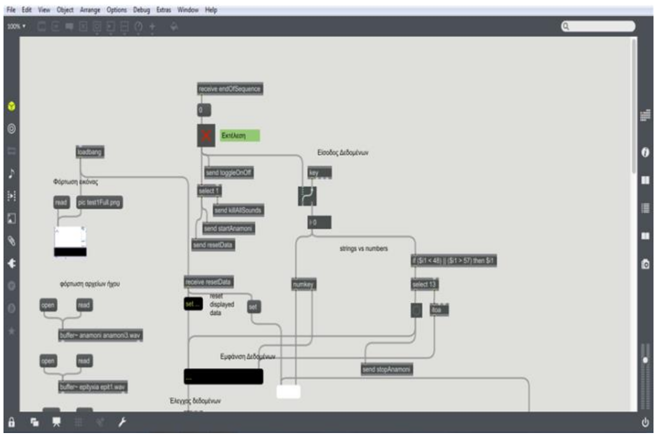


Fig. 2. The MAX/MSP programming environment.

TunePad is a sound composition tablet application controlled by a block-based programming environment. TunePad is designed to introduce learners to computational thinking and to prepare them for text based coding environments. This design actively engages learners and communicates how the programming blocks control the sounds being played. It engages students in computational thinking who may not have otherwise been exposed, giving the opportunity to more people to enter the computer science field [21].

Peep is a Network Auralizer that replaces visual monitoring with a sonic "ecology" of natural sounds [22]. Each sound type represents a specific kind of network event. This system combines network state information from multiple data sources, by mixing audio signals into a single audio stream in real time. Using Peep, one can easily detect common network problems such as high load, excessive traffic, and email spam, by comparing sounds being played with those of a normally functioning network. This allows the system administrator to concentrate on more important things while monitoring the network via peripheral hearing. After visualization, the next step will be a process of transmitting information through sounds by using parameters that the human ear responds to, such as intensity and frequency [10].

## III. Teaching Programming Using bi-modal Interactions

Research teaching scenarios have been conducted evaluating students' perceptual and critical ability in basic algorithmic structures [24]. On the other hand, according to [32], students in Greek high schools learn only basic elements of programming, due to the inefficient and, in some cases, outdated teaching methods and the absence of an interconnection between education and the industry. In attempt to alleviate the current situation, this paper discusses a methodology for teaching programming through sound-alerts stimulating psychoacoustic perception. The perceived sound quality depends, among other things, on loudness, roughness, and sharpness [18]. The use of physiological measures sensitive to attention and arousal, in conjunction with behavioural and subjective measures, led to the design of auditory warnings that produce a sense of diversity of programming commands [9], [18]. Knowing that acoustic and visual memory make up 90% of the sensory memory can be hypothesized that students using both their optical and acoustic perception, will better comprehend programming structures [18], [58]. The goal is to assist students understand different algorithmic procedures such as relational and arithmetic operations, successful and unsuccessful code execution, assignments, loading data etc., through bimodal (both visual and aural) feedback. Our hypothesis is that through visual and aural interactions students will be able to comprehend the taught programming structures more effectively.

In this direction, we use the MAX/MSP environment to design teaching scenarios for sound-alert production and MAX/MSP patches creation to simulate commands with sound. Each Max/MSP patch was designed to have an identical user-interface to Glossomatheia, which is taught in Secondary Education in Greece. Moreover, Glossomatheia is the basis for the evaluation of the effect of auditory cues on computer programming comprehension and the idea to test auditory feedback using sounds in the system presented was highly motivated by Peep, since our main goal was to achieve a better understanding in programming teaching by using sounds.

### *A. Stimulus Selection*

One crucial component in the design of a bimodal programming educational environment is the selection of the utilized sound stimuli, which, in order to be conducive to the students' learning, they should reflect in a clear and concise manner the algorithmic action they correspond to [9]. In addition, they should also reflect the aesthetics of the target group, which, in our case, consists of senior high-school students.

Hence, a preliminary study was conducted aiming at the selection of the most appropriate sound stimuli, given the aforementioned criteria [9], [18], [20]. Two different categories of sounds were selected, natural (actually birds' sounds) and synthesized, to represent 5 procedures and/or operations a) reading data from keyboard, b) successful data assignment, c) error, d) arithmetic operation, and e) relational operation. For each procedure, a pair of sound stimuli were selected, (one natural and one synthesized), such that the sounds shared common auditory characteristics in terms of pitch, timbre, speed and contour. Participants of the study were then asked to select the best fitting sound alerts for the tested programming procedures.

### *B. Protocol Overview*

One hundred and forty six (146) senior high-school students (67 male), 16 to 17 years old, participated in this preliminary study. Participants were divided into 8 groups. In MAX/MSP environment, groups were presented with a pair of stimuli (sound-alerts) for each of the 5 coding procedures and were asked to select the alert that best fitted each procedure in a 2AFC task with no repetitions. Each group took the test twice, once in the beginning and a second time at the end of a class, to evaluate response repeatability. The approximate duration of each test was 5 minutes. Between groups the order of stimuli as well that of the test procedures was fully randomized, but remained the same within groups, for practical reasons.

### *C. Preliminary Study Results*

The stimulus type selection results of the preliminary study are summarized in Fig. 3. As depicted, the vast majority of the students ranging between (82% and 96%) preferred synthesized alert sounds to natural ones. No significant deviations were observed as a function of stimulus and/or procedure presentation order. Student preference remained roughly unchanged across the two tests' repetitions (see error-bars in Fig. 3). Variations were less than 3% across all tested procedures. These results highlight that the sound-alerts are preferred for bimodal educational software for programming. Bars correspond to the selection rate of each sound stimulus grouped per process, averaged across test repetitions. Variations between repetitions are marked with error-bars.

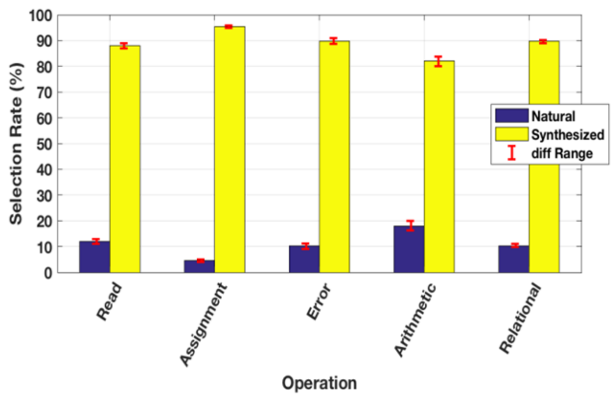


Fig. 3. Preliminary test result overview of the 1st experiment.

## IV. Evaluation Study of Second Experiment

Following the preliminary study, a second experiment was conducted to assess the effectiveness of sound-alerts as a complementary tool for teaching computer programming. Students participated in six (6) teaching scenarios.

Fifty three (53) senior high-school students (24 male), who have previously participated in the preliminary study, volunteered to participate in the three first scenarios and 80 students (38 male) in the next three ones. All of them were, at the time, taking a programming course at school using the Glossomatheia environment.

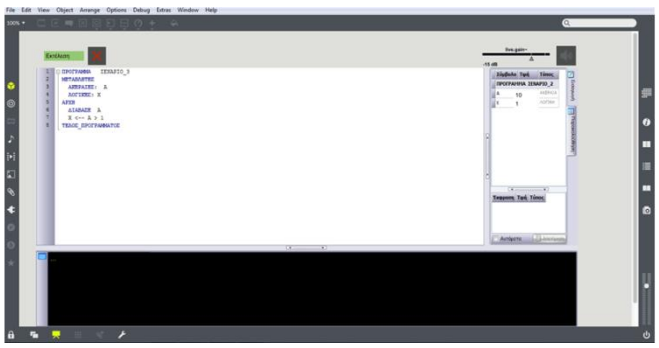


Fig. 4. Max/MSP Environment – Program execution by sound (2nd experiment).

### A. Experimental Protocol

Participants were presented with 6 ready-to-run programming scenarios both in Glossomatheia as well as in Max/MSP. The Max/MSP patch was designed (Fig. 4) to have an identical user-interface to Glossomatheia. Its only difference was that it was complemented by visual feedback with sound-alerts. The sound stimuli utilized for the alerts selected amongst the most preferred sounds of the aforementioned preliminary study.

Participants were allowed to interact with both environments and were afterwards asked to fill a questionnaire evaluating the effect of auditory feedback on the comprehension of the code functionality. For each tested scenario, user ratings were collected on a 5AFC Linkert scale with the following anchors: no affect, minor affect, neutral, moderate affect, major effect. The questionnaire concluded with a "general comments" section, where participants could share their thoughts and feedback on the tested system.

### B. Teaching Scenarios

The following 6 teaching scenarios were tested in the second study. These scenarios use simple input/output operations, arithmetic/relational operations, as well as conditional statements and iteration loops. The selected teaching scenarios target to give a better understanding to the students regarding the difference between e.g. an arithmetic operation and a logical or a relational one, to highlight the differences between the iteration loops etc., by using sound too.

*Scenario 1: Data entry*

The code in the first program scenario performs an assignment of a numerical value to a pre-defined variable (Fig. 5). Upon code execution, the program waits for user-input from the keyboard. If the input value is a numeric one, the assignment is performed, and the program concludes. In case that user-input is not numeric, the code returns an error and waits for a new input value.

```
! Data entry!
Program Scenario_1
Variables
      Integer: A
Begin
      Read A
End
```

Fig. 5. Data entry.

When this scenario is evaluated in the Glossomatheia programming environment, the waiting time for user-input is indicated with a flashing cursor on the computer screen. In the Max/MSP environment, except for the flashing cursor, users hear a sound-alert informing them of a pending action. If user-input is numeric, Glossomatheia, prints the assignment on screen and the code concludes, while the Max/MSP test environment complements the Glossomatheia visualization with a sound-alert indicating successful assignment. If user-input is not numeric, Glossomatheia prints an error message on screen, while the Max/MSP test environment also produces a sound-alert indicating an erroneous action. Each sound-alert was selected in the preliminary study.

*Scenario 2: Arithmetic operation*

The code in the second program scenario (Fig. 6) performs an assignment of a numerical value to a pre-defined variable followed by a simple numerical operation (addition to a constant). Its first part is identical to scenario 1 since it includes the assignment of user-input to a variable. Hence the code works exactly as described in the previous sub-section (Scenario 1: Data entry), and both Glossomatheia and Max/MSP code alerts remain the same. When the arithmetic operation is executed Glossomatheia prints the result on the computer screen, while the Max/MSP test environment also produces a sound-alert, indicating that an arithmetic operation has taken place.

```
! Arithmetic operation!
Program Scenario_2
Variables
       Integer: A
Begin
                   Read A
       A ← A + 1
End
```

Fig. 6. Arithmetic operation.

*Scenario 3: Relational operation*

The code in the third program scenario (Fig. 7) performs the assignment of a numerical value to a pre-defined variable followed by a simple relational operation, i.e., the comparison of the input variable against the numerical value of 1. The first part of this including the assignment of user input to a variable is identical to scenario 1. Hence the code works exactly as described in Scenario 1: Data entry, and both Glossomatheia and Max/MSP code alerts remain the same. When the relational operation is executed, Glossomatheia prints the Boolean result on the computer screen, while the Max/MSP test environment produces a sound-alert too, indicating that a relational operation has taken place.

```
! Arithmetic operation!
Program Scenario_2
Variables
       Integer: A
Begin
       Read A
       A ← A + 1
End
```

Fig. 7. Relation operation.

*Scenario 4: Conditional statements*

This scenario is a combination of the 1st and 3rd scenarios for a better understanding of the conditional statement "if…then…else" (Fig. 8**)**. The code performs an assignment of a numerical value to a pre-defined variable followed by a simple relational operation (comparison of the input variable against the numerical value of 9.5). Its first part is identical to scenario 1 since it includes the assignment of user-input to a variable. Hence the code works exactly as described in Section Scenario 1: Data entry, and both Glossomatheia and Max/MSP code alerts remain the same. When the relational operation is executed, Glossomatheia prints the Boolean result on the computer screen (i.e., true, or false) and a different message accordingly. If the Boolean result is true, the Max/MSP test environment produces a success sound-alert too, otherwise a failure sound-alert.

```
! Conditional statements!
ProgramScenario_4
Variables
       Real: A
Begin
       Read A
       If A>9.5 then
              Write 'Past'
       Else
              Write 'Failure'
       End _ If
End
```

Fig. 8. Conditional statements.

*Scenario 5: Iteration Loops- "while ...do"*

This scenario is a combination of the 1st, 3rd and 4thscenarios (Fig. 9). The code performs an iteration loop based on "while…do" statement. In the first five iterations, the Boolean result is `true`, so the Max/MSP test environment produces each time a success sound-alert. When the Boolean result becomes false, the loop is terminated, and the Max/MSP test environment produces a failure sound-alert.

```
    ! Looping - "while...do"!
        Program Scenario_5
             Variables
                Integer: A
               Begin
               A ← 1
          While A <= 5 Do
              Write A
            A ← A + 1
                End
                End
```

Fig. 9. Iteration Loops - "while...do".

*Scenario 6: Iteration Loops- "repeat…until"*

The opposite result appears exactly in this experiment (Fig. 10). In each iteration, the Boolean result is false, so the Max/MSP test environment produces a failure sound-alert. When the Boolean result is true, the loop is terminated and the Max/MSP test environment produces a success sound-alert.

```
! Looping - "repeat...until"!
Program Scenario_6
Variables
       Integer: A
Begin
              A ← 1
       Repeat
              Write A
              A ← A + 1
       Until A > 5
End
```

Fig. 10. Iteration Loops - "repeat…until".

### *C. Second Experiment's Results*

The participants' evaluations of the effect of sound-alerts on code comprehension are summarized in Table I and Table II. We can see in Table I that more than 60% of them indicated that the use of sound-alerts had a positive effect (either moderate or major) on code comprehension for three tested operations/actions, i.e., waiting for data input (read), successful assignment of data to a variable and erroneous code execution (error), while in Table II, we can also see that more than 85% of them indicated that the use of sound-alerts had a positive effect (either moderate or major) on code comprehension for all three statements, i.e., the conditional statement (if…then…else) as well as both iteration loop statements (while…do and repeat…until). In addition, the results underline that the use of sound-alerts had no effect (neutral) to users in the case of arithmetic and relational operations. This could be interpreted in two different ways: either users generally prefer sound-alerts for events related to the correct or erroneous execution of the code, and for notifications about pending actions, or the specific experiment design and rating question was not appropriate for testing the effectiveness of sound-alerts on such operations.

We can also see that 12% of the 53 participants indicated that the complementary use of sound-alerts had no effect on code comprehension (Table I) on more trivial programming issues like the ones of the first three Scenarios. This observation is also reflected on Fig. 11, which plots user evaluations averaged across the first three tested scenarios. As can be seen, approximately 57% of the students felt that the effect was moderate, or major compared to 19% who felt that the effect was minor or non-existent.

On the other hand, only less than 4% of the 80 participants say the same for more complicated programming topics, such as conditional and iteration statements (Table II). The remaining students did feel that the auditory cues had an impact on their learning. This observation is also reflected on Fig. 12. Interestingly enough, participants showed stronger preference for sound-alerts related to pending activities and correct or erroneous executions of the code than to other operations. This can be related to the fact that sound alerts work well as memory boost, hence notifying users of any code related events that require action. Our interpretation of the ratings is further supported by some of the participants' written comments. For example, some students wrote: "I prefer the sounds for success and error", "the pending action sound helped me understand that was time to input some data", "sound alerts for numerical and relational operations were not so important", "we understand when a repetitive process is performed and the differences between the iteration loops "while…do" and "repeat…until", respectively".

TABLE I: RESULTS OF THE FIRST THREE SCENARIOS - UNDERSTANDING OPERATIONS WITH SOUNDS

| Operation | No effect | Minor effect | Neutral | Moderate effect | Major effect |
|---|---|---|---|---|---|
| Read | 11,32% | 0,00% | 24,53% | 30,19% | 33,96% |
| Assignment | 11,32% | 1,89% | 7,55% | 35,85% | 43,40% |
| Error | 9,43% | 3,77% | 15,09% | 39,62% | 32,08% |
| Arithmetic | 13,21% | 15,09% | 37,74% | 16,98% | 16,98% |
| Relational | 9,43% | 18,87% | 35,85% | 28,30% | 7,55% |

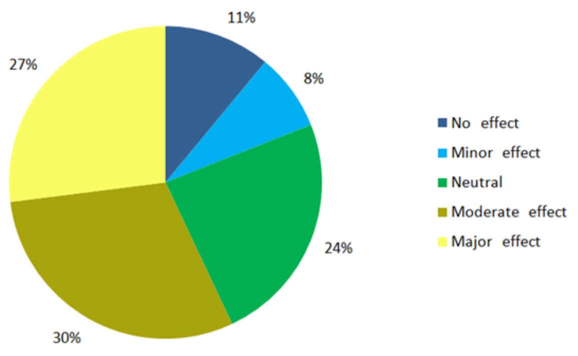


Fig. 11. Overall evaluation of thw use of sound-allerts on code comprehension averaged across the three first teaching scenarios.

TABLE II: RESULTS OF THE LAST THREE SCENARIOS - UNDERSTANDING COMMANDS WITH SOUNDS

| Programming Statement | No effect | Minor effect | Neutral | Moderate effect | Major effect |
|---|---|---|---|---|---|
| If…then…else (conditional statements) | 2,50 % | 0,00% | 10,00% | 47,50% | 40,00% |
| While…do (iteration loops) | 3,75 % | 0,00% | 8,75% | 50,00% | 37,50% |
| Repeat…Until (iteration loops) | 5,00 % | 2,50% | 6,25% | 38,75% | 47,50% |

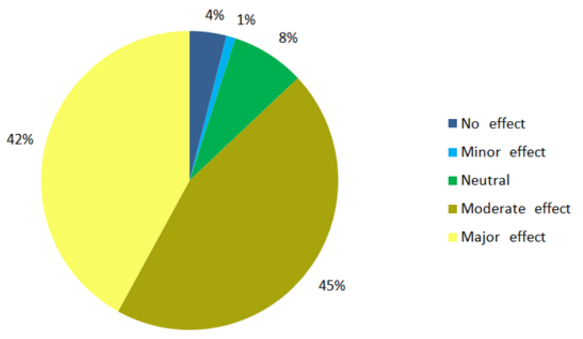


Fig. 12. Overall evaluation of the use of sound-alerts on code comprehension averaged across the last three teaching scenarios.

## V. EVALUATION STUDY OF THIRD EXPERIMENT

Then a third experiment was conducted to assess the effectiveness of sound-alerts.100 students had all the same test on iteration loops. Half of them had been taught the corresponding programming topic without sound-alerts, while the rest students had also participated in the sound-alert experiments presented in the previous sections. In order to be as unbiased as possible, we intentionally chose both groups of 50 students to be equivalent, based on their past performance (both groups had performed the same average score in the corresponding course, up to this experiment).

### *A. Experiment Protocol*

The test consisted of 10 looping procedures and students had to answer the number of looping. They were rated on the centigrade level.

### *B. Test*

The test consisted of 10 questions regarding iteration statements, i.e., either "while…do" or "repeat… until" loop.

Students had to find out whether a procedure was repeated or not and, in the first case, how many iterations are taking place. Furthermore, students had to understand that the "while…do" loop is repeated when the logical condition is true, while the "repeat…until "loop terminates when the corresponding logical condition is true.

### C. *Test Results*

The test results are really impressive and are depicted in Table III and Fig. 13. The very high percentage of 92% of the students who participated in all six (6) enriched with sound-alerts teaching scenarios, were rated over 50/100. On the other hand, the corresponding percentage of the students who did not participated in our scenarios was only 40%. The results are even more impressive, i.e., 72% against 12%, respectively, if we focus on the highly rated students (i.e., students rated over 70/100), Thus, the results regarding low-rated students, i.e., students rated with less than or equal to 50/100, are anticipated (8% against 60%, respectively).

TABLE III: TEST RESULTS

| Student grade | Students– programming (iteration loops topic) taught *without* sound-alerts | Students – programming (iteration loops topic) taught *with* sound-alerts |
|---|---|---|
| 0-50 | 60 % | 8 % |
| 51-70 | 28 % | 20 % |
| 70-100 | 12 % | 72 % |

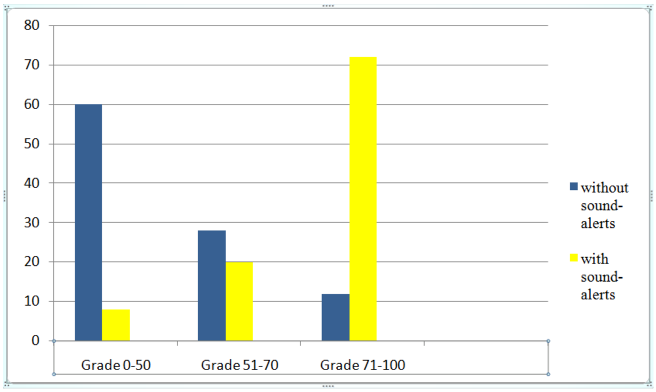


Fig. 13. Evaluation of the participants and non-participants in the research teaching scenarios (Grading Rates).

The test results underline the importance of the sound-alerts' role in teaching basic programming topics, such as iteration statements. It is clear that students participated in the experiments with the embedded sound-alerts present much better understanding regarding iteration loops than students who didn't participate in the experiment.

## VI. CONCLUSION AND FUTURE WORK

This paper discussed the potential benefits of using auditory cues (sound-alerts) as a complementary tool for teaching programming in secondary education. The work is based on the hypothesis that bimodal user interactions could positively impact the students' development of problem solving skills and improve their comprehensions of programming code.

Three studies were presented. The first assessed the type of sounds which would be preferable for such a task, while the second and the third whether the use of sound-alerts affects code comprehension, or not.

In the first study, two different sound categories were considered: recorded excerpts of birds' sounds, referred as natural sounds in the text, and composed electronic sounds from synthesizers, referred as synthesized sounds, respectively. 5 different computational procedures and/or operations were defined (i.e., reading data from keyboard, successful data assignment, error, arithmetic operation, and relational operation). For each procedure, a pair of sound stimuli were selected and paired to one sound from each of the two categories. The sound pairs shared common musical and psychoacoustic properties. Participants showed very strong and consistent preference towards synthesized sounds, rejecting almost unanimously the corresponding natural ones across all tested procedures.

In the second study participants had to evaluate the effect of sound-alerts on code comprehension, through 6 given programming scenarios. Overall, students rated the use of auditory feedback positively. In the first three scenarios57% of them indicated that the cues had a moderate or major effect, while only 11% indicated no effect at all. In the last three scenarios 87% of them indicated that the cues had a moderate or major effect and only 4% indicated no effect at all, respectively.

In the third study an excellent improvement in the test score was observed. 92% of the 50 students, who participated in six (6) teaching scenarios enriched with sound-alerts, were rated over 50/100, while only 40% of the rest 50 students, who did not participate in scenarios, had the same rate.

The first step of our future work would be to include sound-alerts for more programming procedures and operations, and test them against more complex teaching scenarios, e.g., arrays manipulation, searching and sorting data algorithms, etc., in order to find out whether our proposed methodology could be also efficient to harder to understand programming concepts, or not. Finally, since some of the participants indicated in their comments that a combination of sound-alerts and voice messages could be also effective, this could certainly be a route worth exploring in our project's future extension. The ultimate goal is the design of a complete instructional tool that interprets and simulates commands, syntax and runtime errors, using sound and achieves the best understanding of programming teaching in Secondary education.

## ACKNOWLEDGMENT

The authors would like to thank all students who participated on the studies presented in this paper.

## REFERENCES

[1] Aaron, S., Blackwell, A. F., & Burnard, P. (2016). The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. Journal of Music, Technology & Education. 9 (1), 75–94.
[2] Aggelidakis, N. (2015). Introduction in programming with Python. Retrieved on 30 June, 2020 from: http://aggelid.mysch.gr/pythonbook/.
[3] Alexandrakis, N.P. (2001). Introduction to Structured Programming and the Programming Language Pascal. Athens.
[4] Algorithmos. Retrieved on 30 June, 2020 from: http://www.algorithmos.gr/glossomat heia.html.

[5] Aslanidou, S. (2010). Educational Technology, From the audiovisual in the digital treatment.
[6] Avouris, N. (2000). Introduction to Human Communication – Computer. Athens: Diavlos.
[7] Bayman, P., & Mayer, R. E. (1983). A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements, Communications of the ACM, vol. 26, n° 9, pp. 677-679.
[8] Beanz (2013). The magazine for kids, code, and computer science. Retrieved on 30 June, 2020 from: https://www.kidscodecs.com/resources/programming/education/.
[9] Burt, J. L., Bartolome, D. S., Burdette, D. W., & Comstock, J. R. Jr. (1995). A psychophysiological evaluation of the perceived urgency of auditory warning signals. Ergonomics, 38(11), 2327-2340.
[10] Cerf ,V.G. (2018).The sound of programming. Communications of the ACM, 61(4), 6.
[11] Chronaki, A., & Kourias, S. (2011). Kids, Robots and Lego Mindstorms: The Recording of an interactive relationship start. Proceedings of the 2nd Panhellenic Conference on ICT in Education (in Greek), pp 1009 – 1022, Patras, Greece.
[12] Cipriani, A., & Giri, M. (2010). Electronic Music and Sound Design - Theory and Practice with Max/MSP - volume 1, 2nd edition. Rome: ConTempoNet.
[13] Dagdilelis, V. (1986). Conceptions des eleves a propos des notions fontamentales de la programmation informatique en classe de Troisieme, Memoire D.E.A., Universite Joseph FOURIER, Grenoble, France.
[14] Dagdilelis, V. (1996). Teaching computer science. The teaching of programming: students' perceptions of the construction and validation of programs and teaching situations for their formation, PHD, Department of Applied Informatics, University of Macedonia.
[15] Du Boulay, B. (1989). Some difficulties of learning to program. In Soloway, E., Spohrer, J.C. (Eds.), Studying the Novice Programmer, London, Lawrence Erlbaum Associates (pp. 283–299).
[16] Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A. & Basnet, R. B. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance. Journal of Computers in Education, 4, 355–3.
[17] Ebooks.edu.gr. Retrieved on 30 June, 2020 from: http://ebooks.edu.gr/courses/DSGL-C101/document/4c65902ff3dk/4e52d483egdp/4e52e406mj6l.pdf.
[18] Fastl, H. (2006). Psychoacoustic basis of sound quality evaluation and sound engineering. Proceedings of the International Congress on Sound and Vibration, 2006. Vienna, Austria.
[19] Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J., & Zagami, J. (2016). Arguing for Computer Science in the School Curriculum. Educational Technology & Society, 19 (3), 38–46.
[20] Genuit, K. (2004). Sound quality in environment: Psychoacoustic mapping. ASA 2004, San Diego, CA, USA.
[21] Gorson, J., Patel, N., Beheshti, E., Magerko, B., & Horn, M. (2017). TunePad: Computational Thinking Through Sound Composition. IDC '17: Proceedings of the 2017 Conference on Interaction Design and ChildrenJune 2017 Pages 484–489 https://doi.org/10.1145/3078072.3084313.
[22] Gilfix, M., & Couch, A. Peep (The Network Auralizer): Monitoring Your Network With Sound. Retrieved on 30 June, 2020 from: https://www.usenix.org/legacy/events/lisa00/gilfix/gilfix_html/.
[23] Grigoriadou, M., Gogolou, A., Gouli, E., Glezos, K., Boubouka, M., Papanikolaou, K., Tsagkanou, C., Kanidis, E.,Dukakis, D., Fragkou, S., & Verginis, H. (2009). Teaching Approaches and Tools for teaching IT. Athens:New Technologies.
[24] Hsu, T.-C., & Hwang, G.-J. (2017). Effects of a Structured Resource-based Web Issue-Quest Approach on Students' Learning Performances in Computer Programming Courses. Educational Technology & Society, 20 (3), 82–94.
[25] Jerinic, L., Ivanovic, M., Puntik, Z., Budimac, Z., & Savic, M. (2014). e-Education in Teaching Programming - Forty Years of Promises? International Conference on e-Learning'14. Retrieved on 30 June, 2020 from:https://www.academia.edu/9828622/e-Education_in_Teaching_Programming_-_Forty_Years_of_Promises.
[26] Kelleher, C., & Pausch,R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. ACM-Computer-Surveys.
[27] Kernighan, B.W., & Ritchie, D. M. (1988). The C Programming Language. Prentice-Hall.
[28] Kirn, P. (2018). Roland and MIT want to use music to teach kids programming. Retrieved on 30 June, 2020 from: http://cdm.link/2018/01/roland-mit-want-use-music-teach-kids-programming/.
[29] Lockwood, J., & Mooney A. (2018) Computational Thinking in Secondary Education: Where Does It Fit? A Systematic Literary Review, International Journal of Computer Science Education in Schools.
[30] Mathrani, A., Christian, S., & Ponder-Sutton, A. (2016). PlayIT: Game Based Learning Approach for Teaching Programming Concepts. Educational Technology & Society, 19 (2), 5–17.
[31] Mayer, R. (1981). The Psychology of How Novices Learn Computer Programming, Computing Surveys, Vol. 13, n° 1, pp. 121-141.
[32] Milne, I., & Rowe, G.(2002). Difficulties in Learning and Teaching Programming—Views of Students and Tutors. Education and Information Technologies 7(1), 55-66.
[33] Noone, M., & Mooney, A. (2018).Visual and textual programming languages: a systematic review of the literature. Journal of Computers in Education, 5, 149–174.
[34] Noss, R. (1984). Children Learning Logo Programming. Interim Report No. 2 of the Childern Logo Project, Advisory Unit for Computer Based Education, Hatfield, United Kingdom.
[35] Papazoglou, P., & Lionis, S.–P. (2014). Applications Developing with Arduino. Publisher: Tziolas
[36] Pea, R.D. (1986). Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research (pp. 2, 25–36).
[37] Place, T., & Lossius, T. (2006). Jamoma:A modular standard for structuring patches in Max: Proc. of the International Computer Music Conference, pp. 143–146, New Orleans, US.
[38] Poulakis, E. (2015). By programming the Arduino microcontroller. Heraklion.
[39] Ringwood, J. V., Monaghan, K., & Maloco, J. (2005). Teaching engineering design through Lego Mindstorms. European Journal of Engineering Education, 30:1, (pp. 91 – 104)
[40] Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137-172.
[41] Rogalski, J. (1989). Problem-Solving in Mathematics and in Informatics: Differencies and Invariants. In J. Hoc, T Green, R. Samurçay & D. Gilmore (Eds.), Psychology of Programming, pp. 236-247.
[42] Rouchier, A., & Samurçay, R. (1984). Concepts Informatiques et programmation: Une premiere analyse en classe de seconde des Lycees, Rapport de recherche CNRS.
[43] Saeli, M., Perrenet, J., Jochems, W., & Zwaneved, B. (2011). Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective, Informatics in Education,(10), 73-88.
[44] Samurçay, R. (1989). The Concept of Variable in Programming: its Meaning and Use in Problem-Solving by Novice Programmers, In E. Soloway, J. Sprohrer (Eds.) Studying The Novice Programmer, 161-178, Lawrence Erlbaum Associates.
[45] Schneiderman, B. (1980). Software Psychology, Human Factors in Computer and Informtion Systems, Winthrop Publishers Inc', Cambridge.
[46] Scratch Wiki (2015). Retrieved on 30 June, 2020 from: https://wiki.scratch.mit.edu/wiki/ ScratchX.
[47] Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition. University of Southampton (E-prints) 6pp.
[48] Spinet. Retrieved on 30 June, 2020 from: http://spinet.gr/glossomatheia/.
[49] Stephenson, C., Gal-Ezer, J., Haberman,B., & Verno,A. (2005). The New Educational Imperative: Improving HighSchool Computer Science Education (Rep. No. Final Report of the CSTA Curriculum Improvement Task Force, 2005).
[49] SuperCollider platform, available at: https://supercollider.github.io/
[50] Szlávi, P., & Zsakó, L. (2006). Programming versus applicationIn: Mittermeir, R.T. (Ed.), ISSEP 2006, LNCS 4226 (48–58).
[51] Toh, L. P. E., Causo, A., Tzuo, P. W., Chen, I. M., & Yeo, S. H. (2016). A Review on the Use of Robots in Education and Young Children. Educational Technology & Society, 19 (2), 148–163.
[52] Tsolakidis, K., & Fokidis, M. (2007). Virtual reality in education.Athens.
[53] Tzimogiannis, A., Tsiotakis, P., Tsakonas, P., & Vraxnos, P. (2019). Basic Misconceptions in Programming Teaching.
[54] Vakaloudi, A. (2003). Teaching and learning with new technologies theory and practice. Athens: Patakis.
[55] Weigend, M. (2006). From intuition to program. Programming versus application. In: Mittermeir.
[56] Xynogalos, S. (2000). Programming Microcosms: Another Approach to Teaching Programming, Proceedings of the Conference "Informatics and Education", Thessaloniki, November 2000.
[57] Xynogalos, S. (2002). Educational Technology: A Teaching Microcosm for Introduction to Object Oriented Programming,

PHD, Department of Applied Informatics, University of Macedonia.
[58] Yarbrough, D. (2017). Sound the alarm: how sounds affect our memory and emotions. Retrieved on 30 June, 2020 from: https://www.voxmagazine.com/music/sound-the-alarm-how-sounds-affect-our-memory-and-emotions/article_153c4146-be25-11e7-b9ab-8b1620bcc28d.html.

**Theofani S. Sklirou** is a teacher of Computer Science and a Mathematician. She received her BA-degree from the University of Patras (1986), her MSc in Advanced Telecommunication Systems and Networks from the University of Peloponnese - Department of Informatics and Telecommunications (2014). She is currently a PhD student (since 2016) at the same Department and her research is based on the algorithmic programming in education.

She has worked several years as a computer analyst - programmer in hotel computerization and taught computer science at the Technological Educational Institute of Athens- Department of Informatics as well as in Public Secondary Education (High School). She served as an Assistant Director of Aspropyrgos High School and she is responsible for High School computerization. Her main research interests are New Technologies in Education. She has also a particular interest in Music and New Technologies

**Areti Andreopoulou** is an Assistant Professor in the Laboratory of Music Acoustics and Technology (LabMAT) at the University of Athens, Greece. She has a bachelor's degree in music studies from the University of Athens (2005) and a Master's (2008) and a PhD degree (2014) in music technology from New York University. Her fields of interest include spatial audio, the design and evaluation of immersive environments, auditory displays, acoustics, and data sonification.

**Anastasia Georgaki** is a Professor in the Laboratory of Music Acoustics and Technology (LabMAT) at the University of Athens, Greece. She has a bachelor's degree in Physics (National and Kapodistrian University of Athens,1986), Music Studies, Hellenic Conservatory of Athens (1981-1990), Master's Degree (1991) and PhD in "Music Technology and Musicology of the XXth Century», IRCAM/EHESS, PARIS, France (1992-1997). Her research projects focus on Acoustic analysis of the singing voice Creative technologies in education Visual music and configurative sounds Acoustic ecology and Cicada song's study Singing voice synthesis Musicological issues in Electroacoustic music Music Technology in Archeomusicology.

**Nikolaos D. Tselikas** is Associate Professor in the Department of Informatics and Telecommunications of the University of Peloponnese. He holds a BSc (1999) and a Ph.D. (2004) from the National Technical University of Athens (NTUA) Department of Electrical and Computer Engineering. His research activity focuses mainly on the design and implementation of web applications and services as well as software, middleware, and internet technologies.