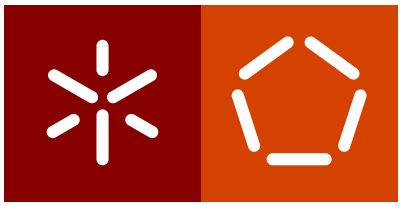
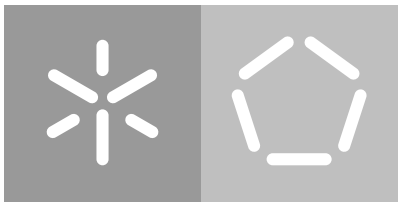


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Diogo da Silva Costa

Melodic, Using Music to train Visually Impaired Kids in Computational Thinking

April 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Diogo da Silva Costa

Melodic, Using Music to train Visually Impaired Kids in Computational Thinking

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
Pedro Rangel Henriques
Cristiana Araújo

April 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Rui Diogo da Silva Costa

_____

ACKNOWLEDGEMENTS

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to thank my thesis advisor, Professor Pedro Rangel Henriques, and my supervisor, Professor Cristiana Araújo, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. You also provided me with the tools that I needed to choose the right direction and successfully complete my dissertation

I would like to acknowledge the people from Íris Inclusiva, Gabinete para Inclusão da Universidade do Minho and Escola Secundária de Maximinios for their wonderful collaboration. I would particularly like to single out Isabel Barciela, Marta Paço and Rui Silva, I want to thank you for your patient availability and for all of the opportunities I was given to further my research.

In addition, I would like to thank my parents and girlfriend for their wise counsel and sympathetic ear. You were always there for me.

ABSTRACT

This document, in context of second year of Integrated Master of Informatics Engineering, reports the development of a project that intends to teach Computational Thinking to kids with special educational needs, in this case blindness. The aim of this research is to characterize both subjects, Computational Thinking and Blindness, and identify what are the current most used and best practises to teach this different way of thinking to kids with special needs. To achieve this, Melodic was created. This is a system composed by a software and a hardware where the user must create sequences with the tactile blocks (the hardware) and then read them with the mobile application (the software), that converts the sequence created into sound. With this, the user can easily hear the differences that the changes in the blocks sequence can make. This can be compared to the Computational Thinking teaching through the use of robots, because in that case, users can see the result of their instructions in the robot movement and with Melodic, the user can hear the result of their instruction with the musical note sequence played by the app. In this document more technical aspects such as the architecture of the application that is proposed to accomplish the goal of the present Master's project, will also be discussed. After this, the project development process that lead to the creation of Melodic is described as well as all the decisions taken. A description of all functionalities of this system can also be seen in this document. To prove the research hypothesis initially stated, some exercises were created and described. The referred exercises were designed to access if Melodic actually develops Computational Thinking.

Keywords: Computational Thinking, Visual Impaired Students Education, Teaching through Music

RESUMO

Este documento, no contexto do segundo ano do Mestrado Integrado em Engenharia Informática, relata o desenvolvimento de um projecto que pretende ensinar Pensamento Computacional a crianças com necessidades educativas especiais, neste caso a cegueira. O objetivo desta investigação é caraterizar tanto o Pensamento Computacional como a Cegueira, e identificar quais são atualmente as práticas mais usadas para ensinar esta forma diferente de pensar a crianças com necessidades especiais. Para o conseguir, foi criado o Melodic. Este é um sistema composto por software e hardware onde o utilizador deve criar sequências com os blocos tácteis (o hardware) e depois lê-los com a aplicação móvel (o software), que converte a sequência criada em som. Com isto, o utilizador pode facilmente ouvir as diferenças que as alterações na sequência dos blocos podem fazer. Isto pode ser comparado com o ensino Pensamento Computacional através do uso de robôs, sendo que nesse caso os utilizadores podem ver o resultado das suas instruções no movimento do robô e com Melodic, os utilizadores podem ouvir o resultado das suas instruções com a sequência de notas musicais tocadas pela aplicação. Este é um sistema composto por um software e um hardware onde o utilizador deve criar sequências com os blocos tácteis (o hardware) e depois lê-los com a aplicação móvel (o software), que converte a sequência criada em som. Neste documento serão também discutidos aspetos mais técnicos, tais como a arquitetura da aplicação que é proposta para atingir o objectivo deste projecto de mestrado. Depois disto, descreve-se o processo de desenvolvimento do projecto que levou à criação da Melodic, bem como todas as decisões tomadas. Uma descrição de todas as funcionalidades deste sistema também pode ser vista neste documento. Para comprovar a hipótese de investigação inicialmente referida, foram criados e descritos alguns exercícios. Os referidos exercícios foram concebidos para verificar se o Melodic realmente se treina o Pensamento Computacional.

Palavras-Chave: Pensamento Computacional, Educação de Estudantes Invisuais, Ensino através da Música

CONTENTS

LIST OF FIGURES

1

INTRODUCTION

This document describes the process of planning and all the research and implementation done in order to be able to create a system capable of teaching Computational Thinking to visually impaired children.

In this chapter a general introduction for the project is done, mentioning the overall context, motivation, project objectives, research approach, and finally the document structure.

1.1 CONTEXT

In today's world technology is everywhere we look. As a consequence employers look for people with strong skills in computational field. This fact leads many young people to choose programming courses. Programming students normally face a great difficulty due to the traditional way of thinking they are taught in conventional schools, while they should be trained in Computational Thinking (CT) that is the key to solve general or programming problems. CT is based on some key concepts such as *Logical Reasoning*, *Algorithm Design*, *Decomposition*, *Pattern Recognition*, *Abstraction* and *Evaluation* and changes the whole way of thinking about the resolution of a given problem (Silva et al., 2019; CEA, 2018; Wolfram, 2016). This change of the way of thinking is not an easy task so the sooner it is introduced to people the better.

That said, it is of uttermost importance to research for a strategy to teach children in a captivating way to keep them motivated. Currently one technique that is very used is "Game Based Learning" Plass et al. (2015).

This technique is characterized by being a type of game play with defined learning outcomes. Usually the game used is a digital one, but this is not always the case. There are many arguments in favour of game based learning, such as:

- **Motivation**: Normally games for entertainment are able to motivate learners to stay engaged over longer periods of time through a series of features of a motivational nature. These include incentive structures, such as stars, points, leader boards, badges,and trophies, as well as game mechanics and activities that learners find interesting;

- **Player Engagement**: It is related to motivation and relies on design decisions that reflect the specific learning goal, learner characteristics, and setting;
- **Adaptivity**: Learner engagement is facilitated by the adaptive part of the game, being this customizable by the player or personalized. This characteristic is defined by the capability of engage each learner in a way that reflects his or her specific situation;
- **Graceful Failure**: Rather than putting failure as a undesirable outcome, it is an expected and sometimes necessary step in the learning process. This is achieved by lowering the consequences on a failure situation and feedback on the wrong answer so that the player can learn from his or her mistakes.

In order to teach Computational Thinking properly it is mandatory to adapt the Learning Resources to the targeted learners. Resources must be adequate for the training of Computational Thinking so that the students develop new knowledge and the different abilities and acquisition of values that define Computational Thinking Araújo et al. (2019); Teixeira et al. (2020).

Reviewing the literature in the area, it is clear that there are some platforms to support the teaching/learning process aimed at a general audience; however, when it concerns special need kids, the available resources are very few. It is very important to realize that most of the referred resources rely upon a visual interface based program paradigm. In order to have a system that successfully helps blind kids to learn the concepts of Computational Thinking, it is mandatory that the system does not rely on a visual interface but on an audio or tactile one (Sabuncuoglu, 2020).

With this in mind, the audio based interface combined with a tactile way of input seems the best approach to teach Computational Thinking concepts through music. That musical approach has clear advantages, as listed below Guzdial (1991):

- Debugging a song to make it sound the way you want is like debugging a program in order to make it do what you want;
- Musical phrases correspond to programming commands;
- A song is a combination of musical phrases just as a program is a combination of procedures.

In this way, the purpose of this Master's work is to create a system capable of, through musical based games, teach the fundamental concepts of Computational Thinking to blind kids to help them to solve programming or general problems using those techniques.

1.2 MOTIVATION

The nonexistence of tools like those described above is the actual motivation to tackle this project. From a personal point of view, it can be said that when I saw this dissertation theme it attracted me because of its public service and inclusive component. All my life I was part of inclusive and community driven movements such as scouts and the orchestra that I am part of. This project focuses on solving a problem that is the lack of learning resources to teach Computational Thinking to visually impaired people. This subject can have a major impact on people's lives and that is what motivates me the most, having the chance to improve someone's living quality.

1.3 OBJECTIVES

This Masters Thesis can split into two main parts, a theoretical one, and a practical one. In the theoretical part the main objectives are:

- Do a comprehensive study in the blind people field, namely in their interaction with learning mechanisms, and main supportive technologies.
- Understand the role that music can play in the training of Computational Thinking.
- Understand how to help blind people using music based Learning Resources to train Computational Thinking.

In the practical part the objectives are:

- Define and plan a complete system that provides games that train Computational Thinking, implementing an intuitive interface for blind people.
- Implement and test that system.

1.4 RESEARCH APPROACH

The research work will be performed in different stages. The methodology that will be followed to achieve this master project will focus on literature revision, solution proposal, implementation and testing. This methodology is composed of the following steps:

- Do a comprehensive bibliographic research about the blind people field, including behaviour and technology (what is done, how it is done and ways on how to do it);
- Research the best practises to create a system to support blind people in the field of Computational Thinking;

- Develop a strategy and propose a solution to solve this problem;
- Design the system architecture to describe how the problem is going to be implemented;
- Implement the system in question - Melodic;
- Test and refine Melodic;
- Assess if Melodic develops Computational Thinking skills.

1.5 RESEARCH HYPOTHESIS

Resorting to music it is possible to develop methods to teach Computational Thinking breaking the barrier to this subject to children with any kind of visual impairment.

1.6 DOCUMENT STRUCTURE

In this section it will be provided a brief introduction to the content of the dissertation chapters.

Chapter 2 relies on the analysis and study of the state of the art on some particular subjects:

- **Characterization of the Visually Impaired**: This subsection focuses on characterizing and analyzing visual impaired people as well as their behaviour, learning difficulties and techniques to overcome these;
- **Computational Thinking Characterization**: This subsection focuses on the characterization of Computational Thinking as well as its main features and use cases;
- **Teaching Computational Thinking to the Visual Impaired**: Here some topics about teaching visually impaired people are addressed, like differences between teaching people with regular and impaired vision, best techniques and approaches, as well as some key topics to guide a potential new teaching activity;
- **Teaching Computational Thinking through Music**: Here are discussed some techniques to teach Computational Thinking with music as a tool. Here are also some examples of the implementation of those techniques;
- **Teaching Computational Thinking to Visual Impaired People Through music**: In this section are documented some theoretical concepts, as well as some implementation examples of activities to teach Visually impaired people Computational Thinking through musical concepts or music itself.

Chapter 3 presents the considerations about this system, designed to help visually impaired people to learn Computational Thinking concepts. This chapter talks about Melodic's architecture and used technologies. Also in this chapter are described the implementation approaches and prototype testing.

In Chapter 4 the development details and decisions are discussed. This chapter divides in the development of the hardware and software of Melodic. In the chapter all the considerations behind the creation of the tactile blocks and the mobile application that the users interact with are described. Also a description of all the algorithm types and errors that Melodic can handle.

Chapter 5 describes the process of testing if Melodic develops Computational Thinking skills. For this, firstly it is discussed the adaptation and creation of test exercises to suit this challenge. Then an analysis to the obtained results from the experiment conducted with blind students is made. Finally, with this analysis, conclusions about the effectiveness of Melodic on Computational Thinking training are drawn.

Chapter 6 summarizes all the work done until this point and the next steps that can be done with this Masters Thesis project.

2

STATE OF ART

In this chapter is presented a brief analysis of the state of the art regarding Computational Thinking teaching, visual impaired teaching and the relation between Computational Thinking and music, being this the theoretic base of this dissertation. Here are presented concepts such as Methods to teach Computational Thinking to visual impaired or normal vision people, the connection between Computational Thinking and music and the use of music in the process of teaching Computational Thinking in schools. This chapter also addresses the route aimed to teach Computational Thinking to visually impaired kids.

2.1 VISUALLY IMPAIRED CHARACTERIZATION

As far back as 1967, Haring and Schiefelbusch (1967) reported on various issues related to the education of visually impaired learners. The focus was on the importance of vision and the mode of reading, and attempted to illustrate how intelligence manifests itself in blind learners as compared to deaf. Their work showed the relevance of blindness and information processing and also illustrated the maximum utilization of available sensory data during learning activities and the translation of visual stimuli.

In visual impaired learners conceptual development and abstract thinking seem to be delayed by the absence of visual stimulation or images. With this, cognitive development occurs more slowly independently of the age group (Freeman, 1985).

A technique to teach conceptual subjects is using tactile stimuli to overcome the difficulty imposed by the lack of sight. Figure 1 illustrates how multiple tactile stimuli can compensate for the loss of sight, allowing learners to perceive size and shape of objects (Fraser and Maguvhe, 2008). However, blind people are easily overwhelmed by the complexity of very 'full or 'busy' diagrams, sketcher or models. A way to counter this feeling is to be spatially oriented when walking or reading documents.

In the learning point of view, blind and visually impaired learners require specific strategies, addressing their unique learning mediation needs. With loss or absence of vision, the amount of sensory data available to the learner is reduced and for this reason is very important the use of multi-sensory approaches, like the one in Figure 1 (Erwin et al., 2001).

To facilitate this multi-sensory interface, Braille text, talking calculators, computer voice over and many more became standard equipment to teach blind and low vision people.

In order to best suit visually impaired students it is mandatory to adapt the curriculum with strategies such as (Fraser and Maguvhe, 2008):

- Setting a substitute task of similar scope and demand;
- Replacing one impossible or unfriendly task with one of a different kind;
- Allowing the learner to undertake the task at a later date;
- Using another planned task to assess more outcomes than originally intended;
- Giving the learner concessions to complete a task;
- Using technology, aides or other special arrangements to undertake assessment tasks;
- Using an estimate based on other assessments or work completed by the learner;
- Considering the format in which the task is presented.

With this techniques it is possible to adapt conventional teaching to a visually impaired student so that ha can learn most of the subjects.

Figure 1: Combination of three-dimensional models with 'Zytec' sketches labelled in Braille as a learning strategy

2.2 COMPUTATIONAL THINKING CHARACTERIZATION

In education, Computational Thinking is characterized by being a set of problem-solving methods that involve expressing problems and their solutions in ways that a computer could also execute (Wikipedia contributors, 2020). It includes the mental skills and practises for designing programs and solve computational end general problems.

Computational Thinking is based on some key topics that are (Silva et al., 2019; CEA, 2018; Wolfram, 2016):

- **Logical Reasoning**: Use the existing knowledge to predict the behavior of a system;
- **Algorithm Design**: Create a sequence of instructions to solve a problem;
- **Decomposition**: Break down complex problems into simpler parts;
- **Pattern recognition**: Identify similarities between problems and apply the same solution to solve them;
- **Abstraction**: Remove unnecessary detail, identifying the essential information to solve a problem;
- **Evaluation**: Prove that the considered solution is suitable for solving the problem in question.

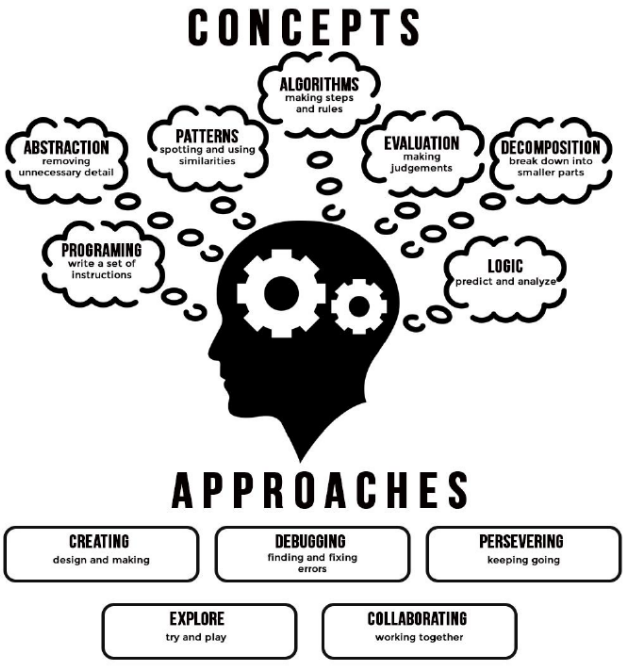


Figure 2: Computational Thinking (Azevedo et al., 2019)

2.2.1 *Teaching* Computational Thinking *to the Visual Impaired*

In a scenario where the students have normal vision, there are many methods normally used to teach Computational Thinking. Gupta (2013) describes the problem that is teaching Computational Thinking concepts to young learners. At this age it is very difficult to comprehend the idea of abstract sequence, and a way to approach this challenge is to use a familiar context like music, histories or puppeteering.

When a person sings or plays a song he/she is, involuntarily, is recreating a long sequence of notes and breaks. These types of sequences are easy to comprehend by children being that each step is in a meaningful and intuitive context. The musical context can be used for conducting educational games like the BO robot. This activity consists in programming a BO, that is a robot capable of reproducing musical notes, to play the note sequence that they want. This is done by writing actions in a tablet or laptop connected to the robot. Further the users can compose their own songs and watch the robot play them.

Puppeteering is a concept created to instill sequence and algorithmic thought into the learners. The idea behind this is to have a robot with motion sensors monitoring the first path the user goes through with it. Then the user is questioned about what steps did the robot take ex: if the path taken was three steps forward then three steps right the user will intuitively say forward, forward, forward, right, right, right. Then the robot navigates through the steps indicated by the user and he will realize that the correct answer would have been forward, forward, forward, right, forward, forward, forward. This exercise develops the learner logical reasoning, decomposition and algorithmic design.

All of these activities have great value to educate this targeted individuals but it is not adequate to teach Computational Thinking to blind or low vision students being that they all base themselves on a graphical interface.

As mentioned before, a great strategy to adapt teaching methods to make them adequate to blind people is to combine multiple senses in order to mediate the information in question without the use of a graphical interface. With this it is possible to, for instance, use a tactile and audible combination to communicate what is being transmitted via the visual interface (Fraser and Maguvhe, 2008).

For this there are some strategies involving tangible interfaces to pour Computational Thinking knowledge to visual impaired people (Sabuncuoglu, 2020). This aims to help students learning how to program without the dedicated help of a teacher, giving the student some autonomy. There are some projects like *Blocks4All* and Microsoft's *Project Torino* that base their input interface in tangible objects. There are some advantages to the use of tangible input such as keeping the users active and the fact that it promotes cooperation and develops better motor skills and spatial understanding (Sabuncuoglu, 2020). In order to

achieve a good activity design there are some aspects to take in consideration (Sabuncuoglu, 2020):

- **Avoid the excessive use of Braille** : Users tend to waste too much time learning and memorizing it;
- **Do not use too many feedback sources**: It can overwhelm students;
- **Use basic and abstract forms**: The shapes used should be easily distinguishable. In order to assign more information to a shape it can be done by using a texture or engraving.

With these techniques it is possible to successfully design an activity to teach Computational Thinking to visually impaired people.

2.2.2 *Teaching* Computational Thinking *through Music*

In order to adapt the matter of Computational Thinking to blind people, it was required a strategy that had to rely on a non graphical interface. Music is a great way to suit this need being that it is a familiar matter to most people.

With this in mind there are a lot of activities created to teach basic computational thinking concepts to learners through music. There are some key relationships between music and Computational Thinking (Bell and Bell, 2018), such as both are based on a formal language and have common key concepts like sequence and repetition.

With this, to introduce and develop Computational Thinking concepts Bell and Bell (2018) describes some games and activities to achieve that goal:

- **Binary Representation Based on Sound**: Firstly students are taught how to make binary-decimal conversion. Then the objective of the game is to assign the alphabet letters to numbers (a-1, ..., z-26). The final task is to get a sequence of notes (A, Bm, G) and convert them to binary using the letter-number relation. This activity develops concepts like algorithm design, abstraction and evaluation;
- **Harmonic Theory Through Positioning Robots**: This activities is based on programming *BeeBots* to position themselves in a musical sheet. These robots have four types of movements: forward, back, left and right and users have to create a sequence of actions to make them arrive at the desired location. This can introduce concepts like parallelism if users are programming more than one bot. A good challenge to train algorithm design is to make all bots arrive at the same time at the desired location. This activity develops algorithm design, sequential thought and decomposition;

- **Music Theory Through Programming Note Sequences**: This activity uses the platform Scratch and its musical playing capabilities. The objective is to present the student a melody or a sequence of notes and he has to recreate it using Scratch's MIDI programming options. Then students can create their own melodies even more complex than the ones presented. To introduce the concept of loop it is asked to make a sequence of notes repeat x times and with this they train algorithm design, abstraction and decomposition.

In *Algo+Ritmo* (Silva et al., 2019) is described a set of activities to train Computational Thinking using musical elements like claps and finger snaps. These games are:

- ***Decifra o Ritmo***: This game is based on the conversion of decimal to binary format. The objective is to teach the students how to make the conversion and then, given a decimal number, they have to convert it to binary and represent it as sounds, for example, zero corresponds to clap and one corresponds to a finger snap. That way they can represent with sound a decimal number;

- ***Algo + Ritmo***: This game aims to teach the concepts of algorithm. It consists in two paper sheets, one with the song_base_rithm and the other with the song_full_rithm. Firstly is presented a song and discussed the best way to learn that song. Then the students must fill the song_base_rithm paper with the steps previously discussed. To end the activity learners are asked to compares their results to the song_full_rithm paper and analyze the differences. This develops algorithm design, abstraction, decomposition and evaluation.

With this set of activities and some more, it is possible to successfully teach Computational Thinking concepts associated with music providing a more familiar and engaging environment to students.

2.3 TEACHING COMPUTATIONAL THINKING TO VISUAL IMPAIRED PEOPLE THROUGH MUSIC

In order to teach Computational Thinking though the use of music, the similarities between these two subjects must be understood.

Chong (2019) states that, like Computer Science, which has a set of sub-disciplinary domains, Music also has sub-disciplinary divisions. A good example of the application of abstraction in music is the chord concept, where every chord symbol represents a particular set of notes.

Since problem solving is also central to Computational Thinking, it can be associated to music too. If the goal is to analyze a structure of a song or melody, firstly, the task is to identify the relevant pieces of the music to examine. Given the complexity of the music in question, the task will need to be broken down in smaller tasks, which applies the decomposition concept, whereby particular aspects of the harmony can be examined. For each of these processes, the mode of representation of what is heard by the musical score will need to be decided, which trains Abstraction. In converting the visual raw data into a particular music-symbolic representation, such as reading and playing a music sheet, some form of music analysis and pattern recognition is involved. The Algorithm Design features are vast, becoming evident when the task is to think of a computer doing the work stated before. From the conversion of the raw musical input into some representation to the collation of salient features, very specific sets of instructions, an Algorithm must be crafted (Chong, 2019). Evaluation can be trained by evaluating the correctness of a harmonic progression. This activity also involves applying some decomposition to separate the different notes.

In the scope of teaching Computational Thinking with the use of Music, Sabuncuoglu (2020) describes a activity that aims just that. This activity consists in a set of tangible blocks:

- Start: Start the execution;
- ClearAll: Used to clear any previous melody that is in memory;
- Loop: Creates a Loop;
- WaveType: Changes the wave type that can be Square os Sine;
- Sound: Changes the instrument sample;
- Info: Opens the info area of the application;
- Run: Executes the code and starts the melody;
- Frequency: Changes the frequency of the following blocks;

- Beat: Changes the beat of the music;
- Note: This block has eight options, seven notes and one stop.

Users must place the blocks ordered on a grid in order to make a melody that makes sense. Then with their phone, they scan the blocks and the phone plays the music created.

2.4 SUMMARY

In this chapter the documentation of the State of the Art analysis is described. This resumes in some key aspect such as:

- **Characterization of Visually Impaired People (2.1)**: Here it is done a characterization and description of the people with some kind of visual impairment and key difficulties or facilities in learning;
- **Characterization of Computational Thinking (2.2)**: Here an analysis of the key aspects of Computational Thinking is done as well as their explanation;
- **Teaching Computational Thinking to Visual Impaired people (2.2.1)**: Here it is done a description and study of some techniques to pass information and knowledge about Computational Thinking to visually impaired people;
- **Teaching Computational Thinking through Music (2.2.2)**: Here it can be found descriptions of some key methods to pass some Computational Thinking topics through a musical interface;
- **Teaching Computational Thinking to Visually Impaired People through Music (2.3)**: Here all the topics mentioned before are combined, describing some methods to teach Computational Thinking to visually impaired people using a musical interface.

3

PROPOSED APPROACH

In this chapter is described the proposed approach and reasons behind it. Here will be discussed aspects like the System's Architecture, explaining the basic components in this project and the development decisions that were taken in the process of creating Melodic.

3.1 SYSTEM ARCHITECTURE

Melodic is a system aimed to be used with a mobile device, being this either a smartphone or tablet. To mitigate incompatibilities between *Apple* and *Android* devices, the *React Native*[1] framework was used, which provides parity between the two platforms with the same JavaScript code.

In Figure 3 it can be seen Melodic's Architecture Diagram that depicts the system components:

- **User**: The user controls the composition of the blocks to form a musical sequence that produces the desired sound after being scanned by the mobile device;
- **Melodic Blocks**: Set of ordered blocks (algorithm) composed by the user to form the desired musical sequence. Each block has a QR code, at the bottom, so that the mobile device can read its meaning;
- **Mobile Device**: Component in charge of all the logical operations behind this app containing multiple sub-domains:
  - **Camera**: The camera is in charge of reading the QR codes of the blocks;
  - **Melodic App**: It is the core of the system. This module receives, through the camera, the QR code of each block and processes it, behaving like a compiler. After processing the QR codes, the Melodic App sends the information (musical melody) to the Speaker module;
  - **Speaker**: The speaker receives the information, sent by the Melodic App, with the musical melody to play and converts it into a real sound to output to the user.

1 For more information about React Native consult: https://reactnative.dev/

It is important to notice that the mobile device also provides tactile feedback to the user, in the form of vibrations, to alert the user that the QR code of a given block has been scanned.

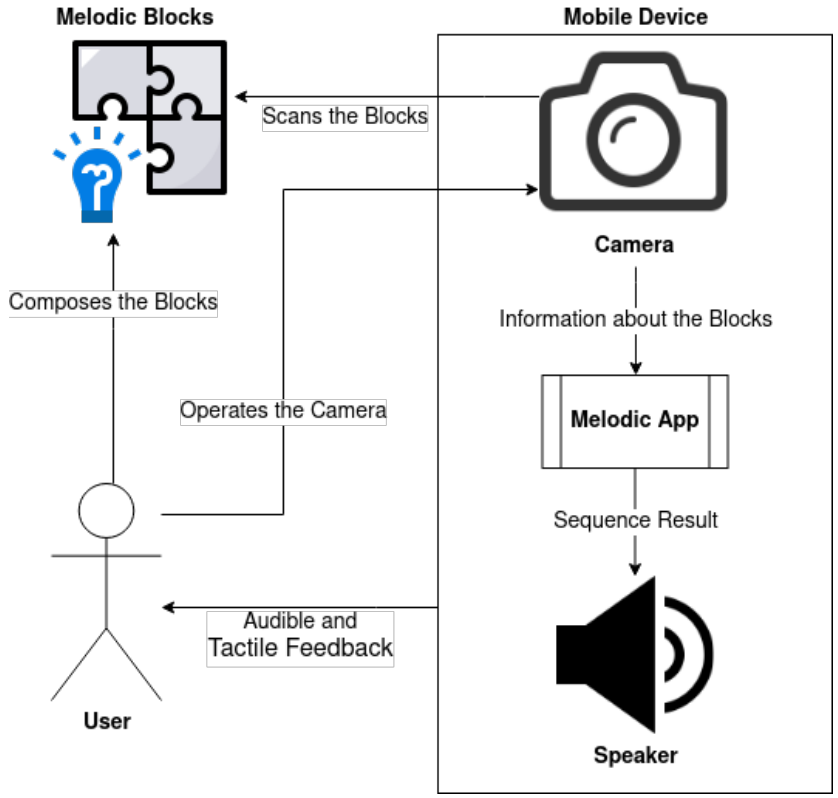


Figure 3: Melodic's System Architecture Diagram

In order to facilitate the testing of this system in a easy way, it was developed with the help of Expo[2]. Expo creates a project capable of being accessed by anyone without the need of publishing it on the App Store or Play Store. To access Melodic it is just necessary to install the app Expo.Go and then scan the QR code exhibited Figure 4. Of course this means that the mobile device must be able to read QR codes. To suit this need it is used a library called *expo-barcode-scanner* that provides the necessary functions.

After the first tests set succeeded, a new set of tests is being prepared to be experimented with members of **Íris Inclusiva**[3].

With this first System's specifications it is possible to have an initial idea of Melodic's basic structure and functionalities being this the base to the future work and application development.

2 To download Expo consult: https://expo.io/client
3 For more information about **Íris Inclusiva** consult: https://irisinclusiva.pt/

Figure 4: QR code to access Melodic

3.2 SUMMARY

In this chapter it is described the proposed System and reasons behind it.

Here were discussed aspects like the System's Architecture, explaining the basic components that in this project, the main decisions behind the Melodic creation and first tests of this System.

4

MELODIC DEVELOPMENT

In this chapter the development details and decisions will be discussed. As mentioned before, Melodic is based on two main components, the tactile blocks and the mobile application. having this in mind, the development of Melodic can be divided into two main parts, the Hardware Development and the Software Development.

4.1 HARDWARE DEVELOPMENT

To make Melodic come to life a set of tactile blocks were created (Figure 5). These had to be carefully thought through because of their fundamental part in the interaction with the application. The tactile blocks are the main part of the interaction with the system, and translate visual impaired-perceptible language to a context that the mobile application can understand.

Figure 5: All Melodic's Blocks

Before building all the blocks, the starting point was building a prototype out of Styrofoam (Figure 6). This allowed to adjust the shape and size of the blocks in order to best suit the targeted user's needs. With the model approved the development of the final blocks started.

Figure 6: Tactile Block First Prototype

The design of Melodic's tactile blocks can be decomposed in some key components:

- **Block Material**: Firstly, the block's material had to be decided. Among many options the main two were clay and wood. Clay is very easy to shape but it was found that it was too brittle for this use case. Wood, despite being harder to shape than clay, is a material that is familiar to the visual impaired and is strong enough to handle heavy use.

- **Block Shape**: In order to make the interactions with these blocks as easy and intuitive as possible it was mandatory to decide what shape they were going to have. Having this in mind the blocks were divided into three main classes:
  - **Special Block**: This block class is characterized by a smooth surface. Special blocks can be of three different types:
    - **Control**: These blocks control the beginning and the end of an algorithm or instruction. The two blocks shown at the top of Figure 7 are used to mark the beginning and the end of the sequences. The two blocks shown at the bottom of Figure 7 are used to start and stop loops;
    - **Speed**: These blocks control the time interval between musical notes. As it can be seen in Figure 8, there are three levels of speed: *slow* (represented by the symbol ">"), *medium* (represented by the symbol "> >") and *fast* (represented by the symbol "> > >"). The usage of this block is not mandatory; default value is "slow speed";
    - **Instrument**: These blocks control the instrument that shall be played. The letter `G` means *Guitar*, `P` means *Piano*, and `F` means *Flute* (Figure 9). This block can be used multiple times throughout the sequence creating a set of notes

played by different instruments. This is also a non-mandatory block being Piano the default instrument.

  - **Number Block**: This block class is characterized by a texture with lines and represents numbers from one to ten. These assign value to the number of iterations of a loop or the number of repetitions of a single note (Figure 10).
  - **Musical Note Block**: This block class is characterized by a rough texture and represents musical notes. The output will be a sound representation of these musical notes (Figure 11); The musical notes played are: *Dó*, *Ré*, *Mi*, *Fá*, *Sol*, *Lá*, *Sí*, where *Dó* is represented by the block that contains only *one point*, the *Ré* is represented by the block that contains *two points*, and so on.

- **Block Color**: Melodic aims not only blind but also low vision people. For the first group the color of the blocks is irrelevant but not for the second one. For low vision people, it was found that having contrast between the block base and the top part is fundamental to help them identify the shape of the block. As shown in figure 5, all the blocks have a dark (black) base and a light (wood color) top part, facilitating the identification process for low vision users. The colors chosen can be different from the ones presented but they must contrast from each other;

- **Block Texture**: Another way of making block identification easier is by using textures. It was discovered that, by using different textures between block classes, the user can distinguish and select the block he wants in an easier way. By applying different textures to the base of each block class the user can filter first by class and then find the block he/she wants within that class. The textures applied were:
  - Smooth to the special blocks (Figure 7, 8 and 9);
  - Lined to the number blocks (Figure 10);
  - Dotted to the musical note blocks (Figure 11).

Figure 7: Control Blocks

Figure 8: Speed Blocks

Figure 9: Instrument Blocks

Figure 10: Number Blocks

Figure 11: Musical Note Blocks

Being this a mobile application, the blocks by themselves have no meaning to the mobile device. In order to make the translation between tactile blocks and a language that the application can understand, QR codes were used. These have the advantage of being easy to make, requiring only a printer to print them, and have a great compatibility, since all mobile devices nowadays have a camera.

To make Melodic work, each tactile block must have a different QR code as shown in Figure 12.

The QR codes are glued at to bottom of the corresponding block as shown in Figure 13. QR Codes must be always in the bottom of the block in order to avoid reading errors by the application. These errors can happen when the camera can catch two or more QR codes in the same image, randomly reading one of them. Having the QR codes on the bottom of the blocks prevents this type of errors because only the only block upside-down is the one being read.

Figure 12: QR Codes and their Meanings

More QR Codes images are present in Appendix A.1.

Figure 13: Top and Bottom of a Block

One of Melodic's main principles is inclusion. This aims to make the system available to everyone and the design of the blocks took that in consideration. With this the blocks are easy to build by anyone who wants to have a set of Melodic's blocks for themselves. A block's building process consists in some key steps:

- **Choosing the Material**: In this case, as mentioned before, wood was chosen but the block can be made of other materials to the taste of the user. Then a circle for the base must be drawn with approximately eight centimeters in diameter (Figure 14);
- **Cutting the Base**: Here the circle was cut and the edges were sanded to give it a smooth finish (Figure 15);
- **Giving texture to the Base**: Then the desired texture must be applied. In Figure 16 it is possible to see the base with a dotted texture applied to it. The texture that each block class has can be changed by the person building the blocks but every block within the same class must have the same texture;
- **Paint the Base**: Then the person building the blocks must think about the color that he/she wants so give to the blocks. As mentioned before it is very important to have contrasting colors between the base and the top of the block. So the next step is to paint the base in a contrasting color. Figure 17 shows the base painted in black;
- **Glue the Top into the Base**: The next step is to glue the top shape into the base as shown in Figure 18;
- **Glue the QR Code**: Finally it is mandatory to glue the corresponding QR code to the bottom of the block as shown in Figure 19.

With this all aspects of the development and building of Melodic's tactile blocks are documented, allowing everyone to build this system for their personal or institutional use.

Figure 14: Fist Step of the Construction of a Block

Figure 15: Second Step of the Construction of a Block

Figure 16: Third Step of the Construction of a Block

Figure 17: Fourth Step of the Construction of a Block

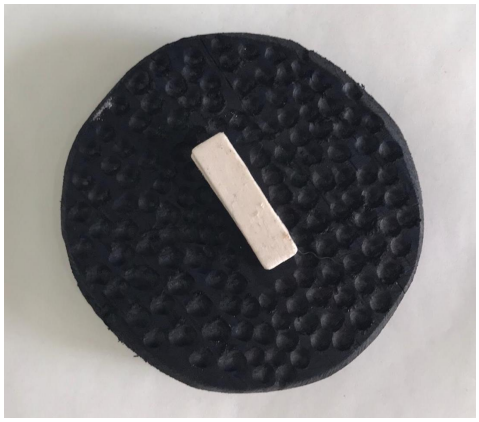


Figure 18: Fifth Step of the Construction of a Block

Figure 19: Sixth Step of the Construction of a Block

4.2 SOFTWARE DEVELOPMENT

In order to achieve Melodic's goals an app had to be developed. To best suit the aimed scope it had to be a mobile app.

The chosen technology to develop this app was React Native[1]. This gave the ability to code in JavaScript and rendering code in native platform, having a cross-platform development. With this it is possible to, with only one project, make the app work for both IOS and Android devices.

To help the development and testing, it was also used Expo[2]. It is a framework and a platform for universal React applications. It provides a set of tools and services built around React Native and native platforms that help you develop, build, deploy, and quickly iterate on iOS, Android, and web apps from the same JavaScript/TypeScript codebase.

Expo creates a project capable of being accessed by anyone without the need of publishing it on the App Store or Play Store. To access Melodic it is just necessary to install the app Expo.Go available in App Store or Expo available in Play Store and then scan the QR code exhibited in Figure 20.

Of course this means that the mobile device must be able to read QR codes. To suit this need it is used a library called *expo-barcode-scanner* that provides the necessary functions.

Figure 20: QR code to access Melodic

1 To find out more about React Native consult: https://reactnative.dev/
2 To download Expo consult: https://expo.io/client

4.2.1 *Melodic Code Logic*

Melodic Mobile app acts as a compiler and runner for the blocks language. For this as the user reads the blocks these are stored in the respective order. For this the system is prepared to handle some scenarios:

- **Start block**: When the user reads the start block a boolean variable that indicates if the sequence has started is set to true;
- **End block**: When the end block is read, the system checks if any of the error flags are set to true. If so the corresponding error message is played. If there are no errors in the sequence, it is played;
- **Start loop block**: This block sets two boolean variables to true. One indicates that the system is inside a loop and the other indicates that the next block should be a number block to define the number of iterations;
- **End loop block**: When the end block is read, the loop is formatted and its content appended to the final sequence array;
- **Speed block**: When the speed block is read the sequence speed variable is set to the chosen value. This will define the interval in milliseconds between notes when the sequence is played;
- **Instrument block**: This block sets the the musical instrument variable. The musical notes read after this definition are going to be played with the chosen instrument;
- **Number block**: When a number block is read, if a loop iteration number is needed, it defines that. If nothing the system does not need an iteration number, the block defines the number of repetitions of the next musical note read.
- **Musical note block**: This is what composes the array in the end of the sequence, a simple list of musical notes to play. When a musical note block is read, either it gets appended to the final sequence array or, if the state of the sequence is inside of a loop, it is added to that loop respecting possible repetitions and musical instrument assignments.

The final sequence is played by iterating on the final array with all the notes to play and playing them at the selected speed.

4.2.2 *Melodic Algorithms*

In order to translate the QR Code data, read by the camera, to the musical notes output, some algorithms were created to suit all Melodic's functionalities.

No matter what functions the user programmed, the output must be a sequence of notes. When running the system, according to the QR Code read, one of many algorithms will be executed:

- **Start Sequence**: When the start sequence QR code is read the control variable *started* is set to true. This variable is used to ensure that the user uses the start sequence block in the beginning of the sequence. It also increments the control variable *blockCount*;
- **Read Note Block**: When a musical note block QR code is read, if not inside a loop, the note number is appended to the array of notes to play. It is known if the note is inside a loop by the control variable *inLoop*. If the note is read inside a loop this note is appended into the global array *loopBody*. It also increments the control variable *blockCount*;
- **Read Number Block**: When a number QR code is read and it is after a musical note block, it repeats it the number of times read. To accomplish this, the last element of the notes array is consulted. Then this musical note is repeated at the final of the notes array for the number of times passed by the number block minus one. If a number block is read after a start loop block it defines the number of iterations of the loop. It also increments the control variable *blockCount*;
- **Change Speed**: When a speed block QR code is read the global variable *speed* is changed according to the value read. This will change the interval between musical notes at the time of playing the sequence. It also increments the control variable *blockCount*;
- **Change Instrument**: When an instrument QR code is read the global variable *instrument* is changed according to the value read. This changes the following notes, maintaining the previous notes with the instrument prior to this change. It also increments the control variable *blockCount*;
- **Start Loop**: When the start loop QR code is read, the global variables *inLoop* and *needIterNr* are set to true. These act as control variables for the next blocks. The *inLoop* value tells if the blocks scanned are in a loop or not and *needIterNr* tells if the loop already has a number of iterations associated with it. It also increments the control variable *blockCount*;

- **Define Loop Iterations**: After a start loop block must be a number block to define how many iterations the loop is going to have. It is known if the number block appears after a start loop block by consulting the control variable *needIterNr*. This is true if the number of iterations for the loop is missing. If this number is not defined at the end of the sequence, an error is issued;

- **End Loop**: When the end loop QR code is read, two main things happen. Firstly the control variable *inLoop* is set to false, since the execution is now out of the loop. Then the content of the loop body must be passed to the final musical notes array. For this two nested *for loops* are used to pass the content of the loop array to the final musical notes array. It also increments the control variable *blockCount*;

- **End Sequence**: When the end sequence QR code is read, some key things happen. Firstly, if it is inside the loop (variable *inLoop* is true), the variable *errorInLoop* is set to true. This means that it is trying to finish the sequence before finishing the loop body which can not be done. Then the function *playNotes* is called. This is the function in charge of playing the proper sequence generated by the blocks read. The end sequence algorithm also sets the control variable *ended* to true and after calling the *playNotes* function empties the final musical notes array. Finally the control variable *blockCount* is set to zero and terminates the execution.

Conditional value (if else statements) is also very important in the computational field. This was not implemented due to the difficulty of representing abstract conditions to the targeted users.

All these algorithms are in the function *handleBarCodeScanned* and are called every time a QR Code is read. Depending on the meaning of the QR Code read, a different algorithm is executed.

Also when a block is scanned the mobile device emits an output to confirm to the user that the QR Code was read. Besides the end sequence block all blocks scanned make the device vibrate and the end sequence block outputs a confirmation sound.

A fundamental part of this system is the sound output the it gives depending on the blocks read. To play the musical notes, the function *playNotes* was created.

This function manages the output given to the user at the end of the sequence. Firstly a check for errors in the sequence must be done. For this control variables *errorInLoop* and *started* are consulted. If one of them is true, an error is dispatched. If the sequence does not contain any errors it starts to play the musical notes. This is done by traversing the final musical notes array and playing each one of them.

To play the chosen instrument at the chosen speed, the function must consult the global variable *instrument* and *speed*. The speed is controlled by placing a sleep function between

musical notes. This sleep can have the duration of 1000ms, 1500ms or 2000ms depending on the speed chosen (fast, normal or slow, respectively).

Now that it is understood how Melodic was developed, it is mandatory to understand how to use the system and take advantage of its benefits.

4.3 MELODIC: USING THE SYSTEM

After describing Melodic's suite, components and interconnection, this section will provide some examples to illustrate how to utilize Melodic as a Learning Resource to train Computational Thinking. An incremental approach to the complexity of problems presented is advised. For this a collection of different exercises can be designed, each one increasing the complexity level:

- **Simple Algorithm**: This type of algorithm consists of only basic instructions. Figure 21 shows an example of a very simple algorithm composed of four instructions: starting block, two musical note blocks and the ending block;
- **Repetition Algorithm**: This type of algorithm is composed of blocks of basic instructions and blocks of repetition. Figure 22 shows an example of this type of algorithm. In this case, in addition to the starting and ending blocks, it has one musical note block in second position, the block with the number three, and then, another musical note block. The block with the number three is intended to play three times the musical note block preceding the repetition number.
- **Changing the Instrument and Speed Instruction**: In this case, the user can create an algorithm with attributes:
  - **Instrument Played**: With the Instrument Blocks the user can change one or more times the instrument played in the sequence (Figures 23 and 24 respectively). When the instrument block is used, the musical notes after the instrument block are played using the sound of the defined instrument;
  - **Speed between notes**: With the Speed Block the user can change the speed between notes at the time of reproduction. This changes the speed for the whole sequence (Figure 25).
- **Iterative Algorithm (Loop)**: In this case the user can program sequences that must be executed one or more times implementing the loop concept. Besides the start and end blocks for the whole algorithm, when the user wants to insert a loop, he must insert also special blocks as follows: start, number of iterations, the loop body and the end loop block. In this case, the order of the blocks is very important. Figure 26 shows an example of how to implement a loop sequence. Like all algorithms, the program starts

with the starting block. To build the loop, it is necessary to place the Start Loop block, then the number of the iteration, then the body of the loop (composed of several other blocks) and, finally, the End Loop block. The end block determines the end of the algorithm. This type of algorithms is the highest degree of complexity in this system.

The user must feel free to experiment with other block sequences at his will.

To listen the melodies resulting from the presented algorithms please access the project Web site at:
https://epl.di.uminho.pt/~gepl/Melodic.

Figure 21: Example of a Simple Algorithm

Figure 22: Example of a Repetition Algorithm

Figure 23: Example of an Algorithm with a Single Instrument Change

In order to obtain a melody, blocks must be organized in a correct sequence. Otherwise Melodic will detect and signalize an error. Regarding wrong block sequences, Melodic copes with 2 different situations, namely:

Figure 24: Example of an Algorithm with Multiple Instrument Changes

Figure 25: Example of an Algorithm with Speed Change

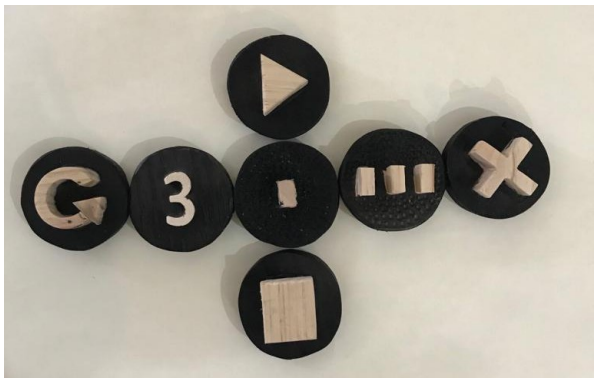


Figure 26: Example of an Iterative Algorithm (Loop)

- **Start Error**: This error occurs when the user forget to initialize the sequence of blocks with the *Start Block*. Figure 27 shows a sequence that misses a *Start Block* causing an error message to be thrown.
- **Loop Error**: In this case, the user composes the blocks to create the loop sequence in a wrong way leading to an incorrect syntax. Figure 28 shows a loop error caused by missing the number of iterations.

To listen to examples of error messages the reader can consult the projet homepage at: https://epl.di.uminho.pt/~gepl/Melodic.

Figure 27: Example of a Start Block Error

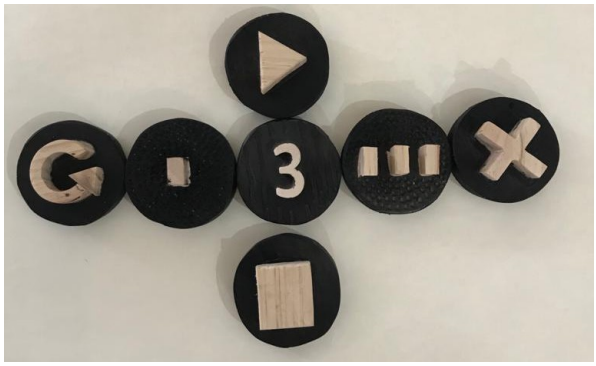


Figure 28: Example of a Loop Creation Error

4.4 MELODIC CRAMMAR

To understand how the Melodic's language works, a grammar was created:

- **Terminals**: START, STOP, REPEAT, EREP, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, SPEED1, SPEED2, SPEED3, DO, RE, MI, FA, SOL, LA, SI, GUIT, PIANO, FLUTE;
- **Grammar Rules (Productions)**:
  - **Prog**: START Inst+ STOP;
  - **Inst**: Note | NoteRep | Speed | Instrument | Cycle;
  - **NoteRep**: Note Num;

- **Cycle**: REPEAT Num CycleInst+ EREP;
- **CycleInst**: Note | NoteRep | Speed | Instrument;
- **Note**: DO | RE | MI | FA | SOL | LA | SI;
- **Num**: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10;
- **Speed**: SPEED1 | SPEED2 | SPEED3;
- **Instrument**: GUIT | PIANO | FLUTE;

4.5 SUMMARY

In this chapter Melodic's development was described. Firstly, the process of designing and construction of all the hardware was reported.

Then the software behind Melodic was described. For this a combination of QR codes and mobile software was the key to success, as presented before. Then, to conclude the chapter, an explanation of how to use the system was provided to illustrate its functionalities and usage. Melodic's grammar was also created to better explain what this system is capable of.

Finally, all the algorithm hypothesis and errors that the System can handle were described.

5

MELODIC TESTING

In this chapter all the testing and validation of Melodic will be discussed. These tests were conducted in two main phases:

- Validation of the Melodic's platform and accessibility;
- Assessment of Melodic actual impact on the acquisition of Computational Thinking skills.

5.1 VALIDATION OF MELODIC PLATFORM

To validate Melodic's usage and accessibility, Íris Inclusiva was a major contributor. In collaboration with this association it was possible to test the system in the fundamental learning scenarios. With that purpose, some experiments were conducted to test wether:

- Blind users can use Melodic;
- Low Vision users can use Melodic;
- Melodic works properly in a device with the screen reader for blind people on.

With the help of some volunteers from Íris Inclusiva, it was possible to test if blind and low vision users could use Melodic. This test was divided in two main phases. The first one was testing if the users were able to find and identify the tactile blocks. This was a fundamental test because the blocks are the main interface between the user's logic and the Melodic application. Figure 29 shows the user exploring the available blocks to make the necessary choices in order to form the desired sequence.

Having the proof that Melodic's tactile blocks were accessible for both blind and low vision people, the next step was to test wether these users could read the block's QR codes using the mobile app. Figure 30 shows the user successfully, using the Melodic App to read the created sequence. With this test it was evaluated that, within few minutes of training, the user got very proficient reading the block's QR codes. This test also confirmed that the

Figure 29: User finding and identifying Melodic's tactile blocks

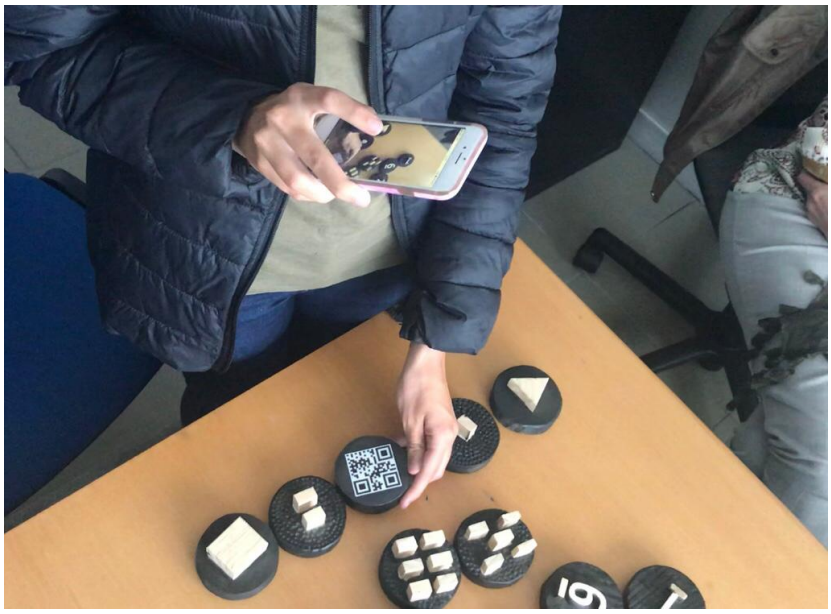


Figure 30: User reading the block's QR codes with Melodic App

user could read a sequence of blocks, receiving the tactile and audible feedback that the mobile app gives.

With this information it was possible to validate that the tactile and audible interfaces that constitute Melodic are capable of being used by blind and low vision people without any major difficulty. These tests were also performed with screen reader mode on and off and this system performed perfectly in both situations.

5.2 Evaluation of Computational Thinking skills learnt with Melodic

Now, that this system can be used seamlessly by visual impaired people, it is crucial to focus on proving this thesis. For this it is necessary to evaluate if Melodic can, in fact, teach Computational Thinking.

With that in mind, the first step is to find a clean aproach to understand how the use of Melodic actually contributes to improve the students ability to solve problems. Zapata-Cáceres et al. (2020) proposed a set of evaluation exercises designed specifically for young students, with ages between 5 to 12 years old. These exercises, although very simple, are able to help on understanding the student's reasoning and algorithm design abilities.

In this article (Zapata-Cáceres et al., 2020), two different kinds of exercises were documented. In Image 31 it is shown an example of an exercise of the first type, where the answer is a multiple choice. To complete this exercise, the user must, firstly, analyze the path on the left and then choose the option with the right steps. These steps describe the path between two points.

Figures 32 and 33 show the second type of exercise to evaluate Computational Thinking. In this case, Zapata-Cáceres et al. (2020) keeps the multiple choice kind of answer, but added a level of complexity. In order to complete the exercise, the user must analyze the given matrix and choose an answer that leads the little chick to the chicken. These exercises have multiple levels of complexity. These can be in the form of constraints of matrix display or in the answer complexity:

- **Matrix display complexity**: The matrix presented to the user can have multiple levels of complexity. Firstly, the one with only the little chick and the chicken. Here the user does not have any restrictions to choose the path to go through. Then the matrix can have a cat. In order to complete the exercise the user must find a path to go from the little chick to the chicken avoiding the cat. Finally the matrix can also contain a brown little chick. The user must find a path to go from the yellow little chick to the chicken, catching the brown little chick on the way and avoiding any cats;

- **Answer option complexity**: These exercises can also have more than one complexity in the answers presented for the user to choose. Firstly, the simple answers do not have

any repetitions. Then, in the second level of complexity, the answer can have counters to say how many times a single instruction must be repeated. Finally, in the third level of complexity, the answers can have blocks of atomic instructions to be repeated twice or more times (introducing the loop concept).

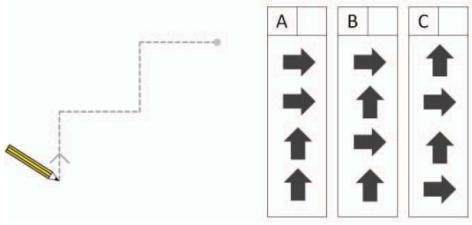


Figure 31: Zapata-Cáceres et al. (2020) fist test type

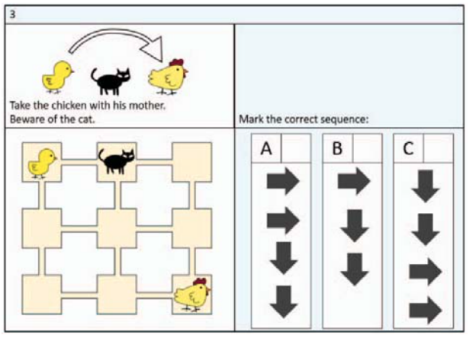


Figure 32: Zapata-Cáceres et al. (2020) second test type

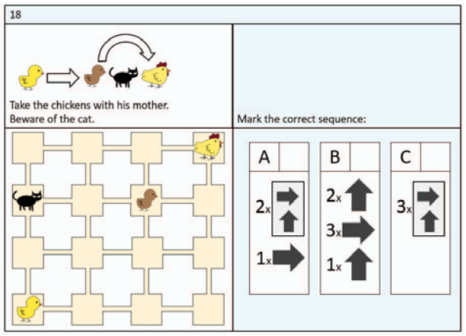


Figure 33: Zapata-Cáceres et al. (2020) second test type

Despite of aiming the same age group as is intended in this Master's work, the exercises described in Zapata-Cáceres et al. (2020) were thought to be used by people with normal vision. This implied some conversions to better suit these exercises to the visual impaired user.

5.3 EXERCISES ADAPTATION

As stated above there are two types of exercises. To convert them to work with a different audience (special needs students), it is important to understand the impact that the lack of sight imposes to a successful completion of the exercise.

With this in mind, it is obvious that any kind of visual interface had to be avoided when projecting and prototyping serious games to evaluate Computational Thinking skills. To achieve this, a wooden engraved matrix was created as shown in Figure 34. This Matrix was thought to be used in two test types, to meet the examples proposed in Zapata-Cáceres et al. (2020).

In Figure 34 it can also be seen four different shapes and connection pieces that will be used in the context of each exercise type. These four figures have specific meanings fundamental for the context of each exercise:

- **Circle**: Defines the starting position of the path;
- **Triangle**: Defines the final position of the path;
- **Square**: Defines a position where the path must not go through;
- **Cross**: Defines a position where the path must go through.

Some of these symbols are the same as used in some of Melodic's tactile blocks and have different meanings. With this the user must see each block as an abstract figure. If the context changes, the same symbol can have a different meaning.

With the apparatus created and explained above, it is possible to convert the two exercise types to adapt them to visually impaired users.

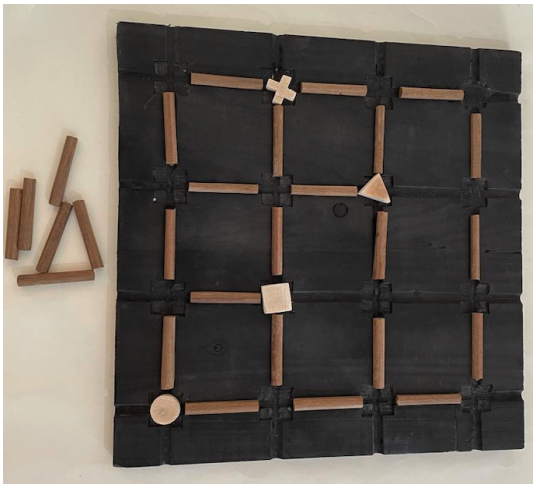


Figure 34: Wooden engraved matrix with all the game pieces

5.3.1 *First test type*

As mentioned in Zapata-Cáceres et al. (2020) and shown in Figure 31 the first exercise has lower complexity. In the context of this work, it is hard to use the multiple choice type of answer with blind people, so that approach was not used. Instead, as shown in Figure 35, a path between the start and end pieces is given to the user.

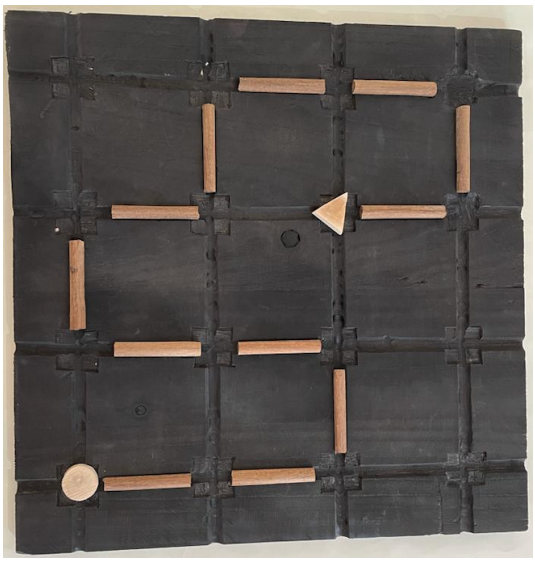


Figure 35: First exercise type example

To complete this exercise the user must create an algorithm that describes the path from start to finish. Assuming that repetitive factors can also be used in the algorithm construction, an example of an ideal answer for the exercise shown in Figure 35 would be:

1. Two times right
2. Up
3. Two times left
4. Two times (up, right)
5. Right
6. Down
7. Left

It is important to notice that the objective of this exercise is, in the shortest time possible, to describe the path presented with the lowest number of instructions possible. For this the user must group steps as shown in the example response (Two times (up, right)).

This challenge for minimal number of instructions forces the user to design algorithms with single step repetitions (ex: two times up) and loops (ex: two times up / left), which are key concepts that Melodic aims to develop.

To analyze the response the teacher/monitor must register the time that the user needed to complete the challenge as well as the complexity of the given answer.

The complexity of the answer can be measured as:

- Basic: The user did not group any steps of the path. It is important to notice that sometimes none of sequence steps are able to be grouped;
- Proficient: The user grouped single steps. For example (Two times left);
- Expert: The user was able to group multiple steps. For example (Two times (up, right)). This is the most complex answer type.

5.3.2 *Second test type*

In Zapata-Cáceres et al. (2020) a second and more complex kind of exercises was described above, as shown in Figures 32 and 33. The key difference between this exercise and the first one is that here the user must create a path rather than identifying it.

To adapt this exercise type to the visually impaired users, the matrix board shown in Figure 34 was used. In the beginning of the exercise, the user is given the board with only the shapes present in that exercise as shown in Figures 36 and 38. Then he/she must analyze the state of the board and create a path that connects the starting piece to the end piece. This path must respect the constraints that the different levels of complexity impose as seen in Figures 37 and 39.

This style of exercise can have four levels of complexity:

- Simple: This exercise only has start and end blocks for the user to connect with the path created. An example of a simple exercise can be seen in Figures 36 and 37;
- Avoid position: Besides start and end blocks, this exercise has the square block that indicates a position that the path must avoid. A simple example of this exercise can be seen in the Figures 40 and 41;
- Require position: Besides start and end blocks, this exercise has the cross block that indicates a position that the path must go through;
- Avoid and Require position: Besides start and end blocks, this exercise has both the square and cross blocks that indicates a position that the path must avoid and one that the path must go through, respectively. An example of a simple exercise can be seen in Figures 38 and 39.

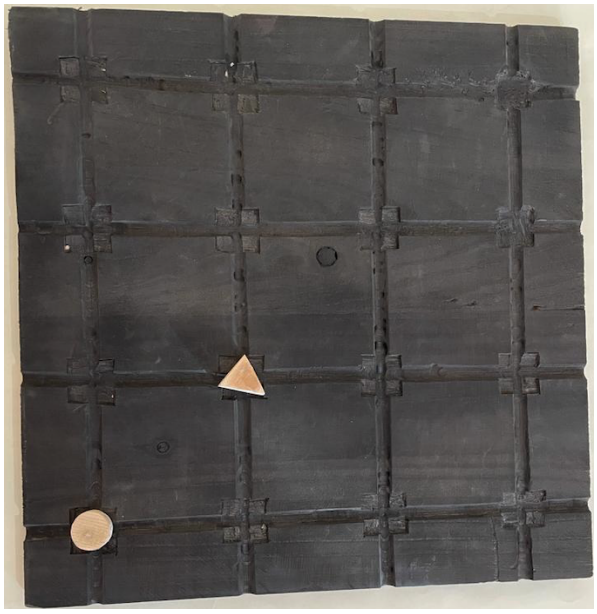


Figure 36: Simple Second exercise type in initial state

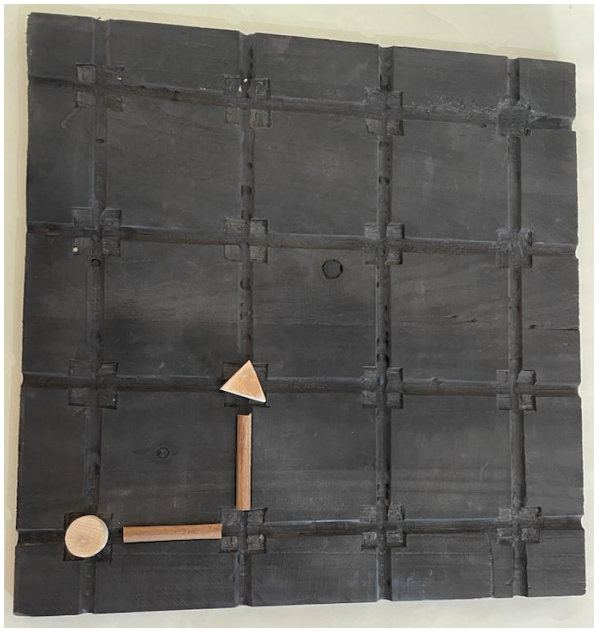


Figure 37: Simple Second exercise type in final state

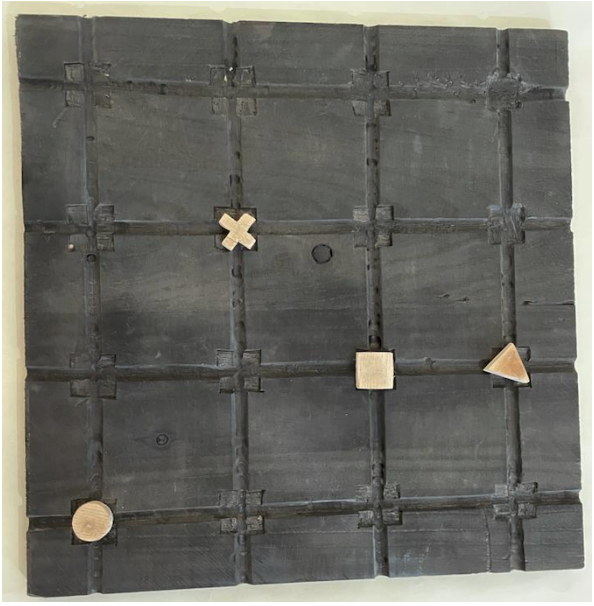


Figure 38: Complex Second exercise type in initial state

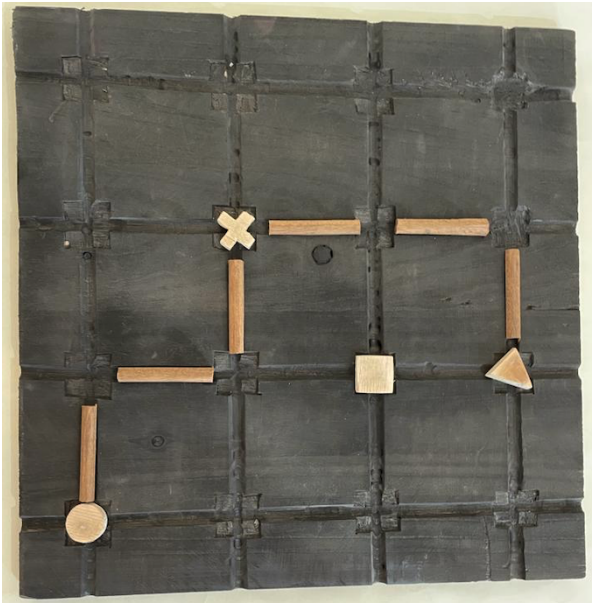


Figure 39: Complex Second exercise type in final state

Figure 40: Avoid path example exercise

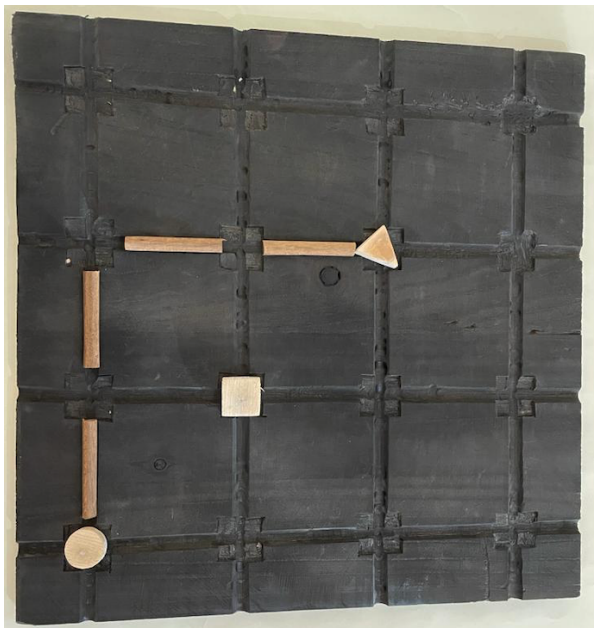


Figure 41: Complete avoid path example exercise

To evaluate the response the teacher/monitor must register the time that the user needed to complete the exercise from the moment he/she starts analyzing the board to the moment he dictates the created sequence. The answer must also be registered in order to analyze the sophistication level achieved.

Similarly to the previous exercise type, the objective of this exercise is to create the sequence with lowest possible number of instructions, in the shortest time possible. For this the user must group steps in the most effective way possible. Figure 39 shows an illustration of the ideal number of steps in the description, that is:

- two times up, right
- right
- down

The same path can be badly said by the user if he can not perceive the steps aggregation using a repetition factor. For instance a naif response for this case would be:

- up
- right
- up
- right
- right
- down

This response translates the same path but has three more instructions, which corresponds to a response with less sophistication.

These adapted exercises served as diagnostic and evaluation tests to access if Melodic actually develops Computational Thinking skills.

By registering the time and complexity of each answer before and after the use of Melodic, it is possible, as initially intended, to prove the efficacy of Melodic system.

It is very important to notice that these exercises were tested and validated with a visual impaired user (see Figure 42) in order to ensure that there were no interface limitations to the user experience.

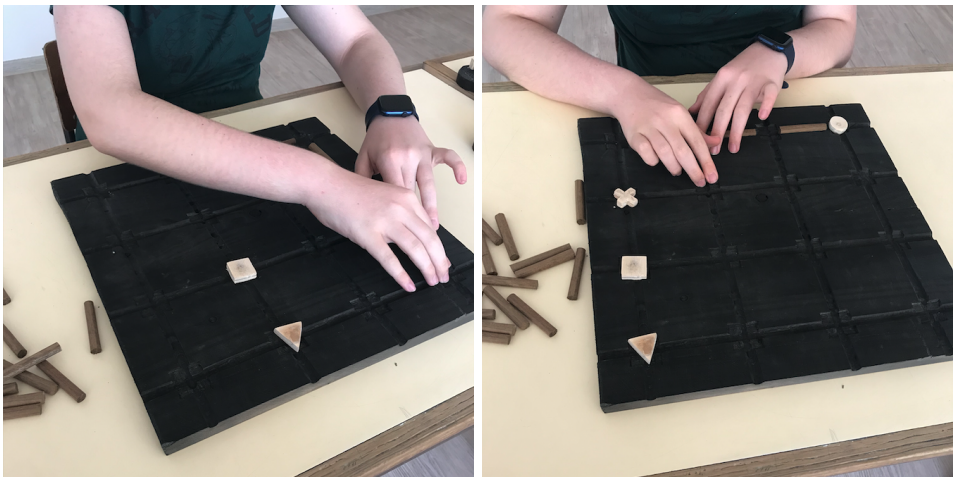


Figure 42: User testing the exercise board

5.4 TRAINING COMPUTATIONAL THINKING SKILLS WITH MELODIC

In order to test and assess the effectiveness of Melodic System, a series of exercises were created and documented to conduct a controlled and uniform training session with all the Testers. The experiment planned must be applied in the same way for every Tester in order to ensure consistency and fidelity when analyzing the various results.

These tests were designed following an incremental approach. Five exercises of each type were created, being the first exercise the most simple and the last the most complex one.

These exercises follow the logic mentioned in Section 5.3 and can be separated in two parts:

- Diagnostic exercises
- Evaluation exercises

As mentioned before, the complexity must be the same in the diagnostic and evaluation tests in order to conclude if there was any improvement in the Computational Thinking skills from the first (diagnostic) to the second (evaluation) sessions considering that in the middle the Tester trained with Melodic. To grant this similarity in complexity level, the difference between exercises is a simple inversion. According to that strategy , the exercise to be used in the Evaluation phase is obtained by inverting the diagnostic one as seen in Figures 43, 44, 45 and 46.

With this, the tests were conducted with the most fidelity and trust in the results obtained.

All the diagnostic exercises can be seen in Appendix A.2.

All the evaluation exercises can be seen in Appendix A.3.

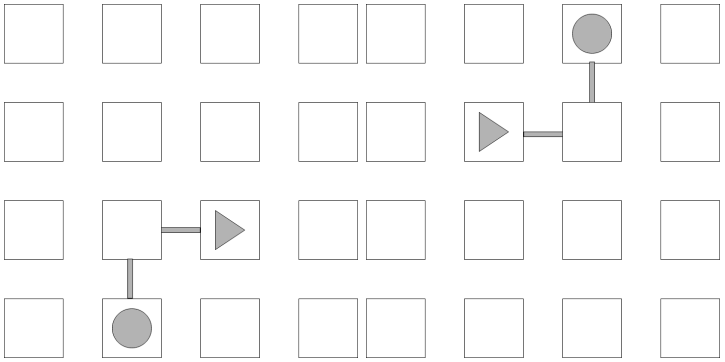


Figure 43: An exercise of the first type in 2 versions, for Diagnostic and Evaluation phases

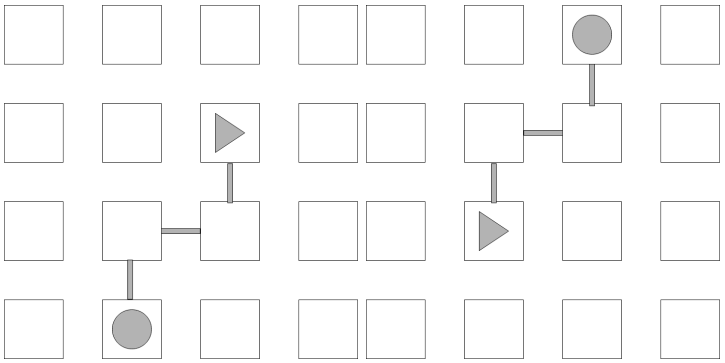


Figure 44: Difference between second exercise of the first test type

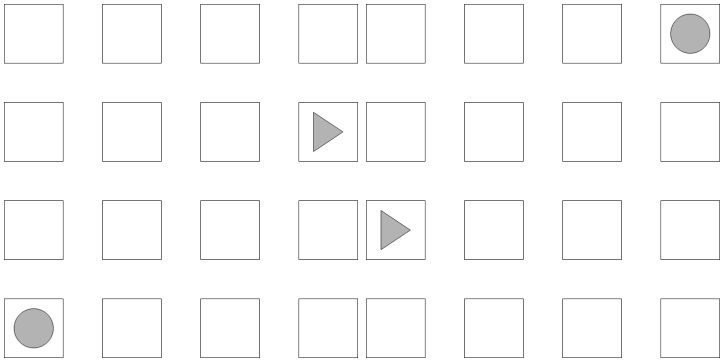


Figure 45: Difference between first exercise of the second test type

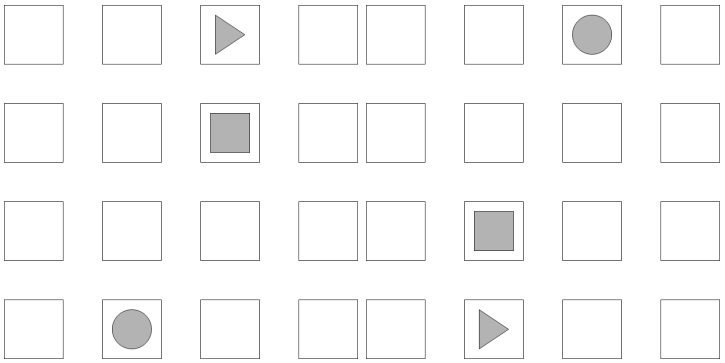


Figure 46: Difference between second exercise of the second test type

5.5 Melodic exercises to develop computational thinking skills

This section reports the design of the exercises sequence to train Computational Thinking using Melodic. For this, like before, an incremental approach was adopted.

For that purpose, the exercises were divided into the multiple complexity levels that are achievable with Melodic:

- Simple Algorithm
  1. Create a sequence of just one musical note of your choice;
  2. Create a sequence with two distinct musical notes of your choice;
  3. Create a sequence with the same musical note repeated twice;
  4. Create a sequence with only one musical note, which must be the note 5;
  5. Create a sequence with two musical notes that must be 3 and 6 in this same order;
  6. Create a sequence with 5 musical notes of which two must be repeated;
  7. Create the following sequence of notes: 1, 5, 3, 5, 4, 4, 7.
- Change Speed
  1. Create a sequence with 3 notes of your choice and set a velocity of your choice;
  2. Create a sequence with 4 notes of your choice and set the speed to fast;
  3. Create a sequence with the same musical note repeated twice with a speed of your choice;
  4. Create a sequence with 5 notes of which two should be repeated and set the speed to medium;
  5. Create the following sequence of notes 1, 5, 3, 5, 4, 4, 7 and set the speed to fast.
- Change Instrument

1. Create a sequence with 3 notes of your choice and define an instrument of your choice;
2. Create a sequence with 4 notes of your choice in which the instrument must be the flute;
3. Create a sequence with the same musical note repeated twice with an instrument of your choice;
4. Create a sequence with 5 notes of which two must be repeated and the instrument must be the guitar;
5. Create a sequence with two distinct musical notes of your choice where the first note must be played by an instrument of your choice and the second note by another instrument of your choice;
6. Create the following sequence of notes 1, 5, 3, 4, 4, 7 where the instrument playing the first two notes is the piano and the instrument playing the remaining notes is the guitar.

- Repetition Algorithm
  1. Create a sequence with a musical note repeated a number of times of your choice, using the number blocks;
  2. Create a sequence with a first musical note and then with another note repeated as many times as you like;
  3. Create a sequence with the first two musical notes different from each other, and the third note repeated 3 times;
  4. Create a sequence with two distinct musical notes repeated 2 and 4 times respectively;
  5. Create a sequence equivalent to 2, 2, 6, 6, 6, 6, 4, 4, 4, but using a repetitive factor whenever appropriate.
- Iterative Algorithm (loop)
  1. Create a sequence that contains a loop with two musical notes of your choice and as many iterations as you wish;
  2. Create a sequence with a loop with 3 iterations and with 3 musical notes of your choice;
  3. Create a sequence with a loop with 5 iterations and the musical notes in it must be 4, 3 and 5;
  4. Create a sequence with two loops in a row with the number of iterations and notes of your choice;

5. Create a sequence with two loops in a row with 2 and 3 iterations containing the musical notes 4, 5 and 7, 3 respectively.

- Mixed Algorithms
  1. Create a sequence played by a guitar that translates the following notes: 1, 2, 1, 2, 1, 2, 4, 4;
  2. Create a sequence that translates the following notes: 5, 4, 3, 5, 4, 3, 7, 7, 7, 1 where the last 4 notes must be played by a guitar. The speed of this sequence must be fast;
  3. Create a sequence that translates the following notes: 1, 4, 1, 4, 2, 5, 2, 5, 2, 5, 6, 6, 6, 3, 3. This sequence should be played at a medium speed and the last 5 notes played by a piano, with the previous notes played by a guitar.

With this set of exercises, the user can improve his Computational Thinking skills. These are a sequence of challenges incrementally more complex leading to a soft learning curve so the user does not fell overwhelmed with the difficulty.

These training session must be performed after the diagnostic session and before the evaluation session in order to conclude if Melodic actually trains Computational Thinking skills.

5.6 Experiment and Result Analysis

In this section the results collected from the experiment conducted to appraise Melodic's efficiency will be discussed. For this, the results reached by the different users in the different evaluation moments will be analyzed.

To prove this hypothesis tests were conducted with two different Testers. These are 45 years old and 15 years old. No one has experience in the computational field.

These tests were performed in three sessions, each taking roughly one hour. The first one is the Diagnostic session, where the user is challenged with a series of exercises to evaluate his/her initial Computational Thinking performance state. In the second session the user develops his Computational Thinking skills by resolving a list of exercises using melodic. The third session aims to evaluate the difference that melodic makes. This is achieved by making the user answer to a list of exercises with the same complexity of the Diagnostic session. With these results we can later evaluate if the user improved in the Computational Thinking field.

As mentioned before, to evaluate the test results, three metrics were used:

- **Execution time**: This metric applies to both exercise types. It is the time that the user took to complete the exercise.

- **Answer complexity**: This metric applies to both exercise types. It describes the level of sophistication that the Tester used to complete the exercise, being Basic the first level, Proficient the second and Expert the highest level of complexity.
- **Number of instructions**: This metric applies to only the second exercise type because in the first one the user does not have to create a set of instructions. It is the number of steps needed to connect the points to complete the exercise.

To better perceive the results, a set of plots were developed that compare the results gathered according to the different metrics. These plots are grouped by metric and exercise type. So the graphics produced represent the Execution Time, Answer Complexity and Number of Instructions differences between the Diagnostic and Evaluation phases.

5.6.1 *Execution Time*

Figures 47 and 48 show the difference in time between the diagnostic and evaluation tests exercises. The bar graphics shown represent the times consumed by each Tester to complete each of the five exercises in both phases. In those figures, it is possible to observe that generally the time used to finish an exercise decreases from the diagnostic to the evaluation sessions.

To better perceive the results, Figure 49 exhibits a comparison between the average diagnostic exercise completion time with the evaluation one. These make clear the general improvement in execution time.

Although the first user had a worse execution time in the first group of exercises, in average the completion or execution time shows a decrease of 7%. This result can be related to overthinking the answer, wasting time in this process.

In the second exercise type, both of the Testers exhibited positive results, having in average a decrease in execution time of 42,6%.

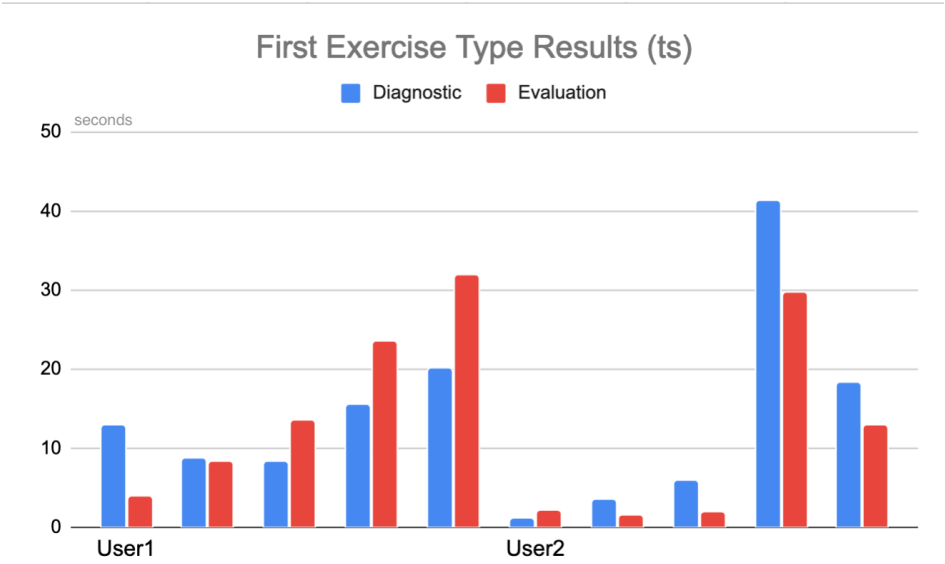


Figure 47: Comparing the Completion Time for the first type in both diagnostic and evaluation phases

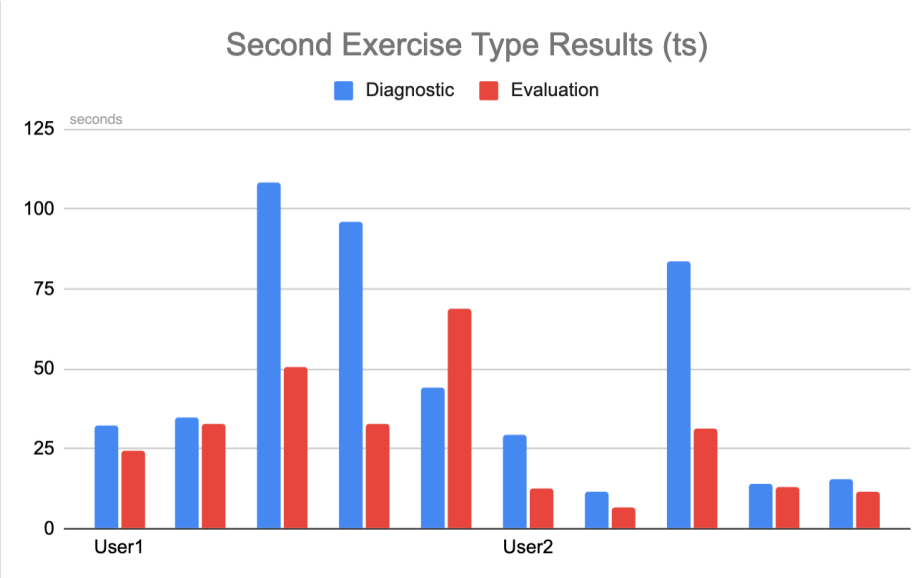


Figure 48: Comparing the Completion Time for the second type in both diagnostic and evaluation phases

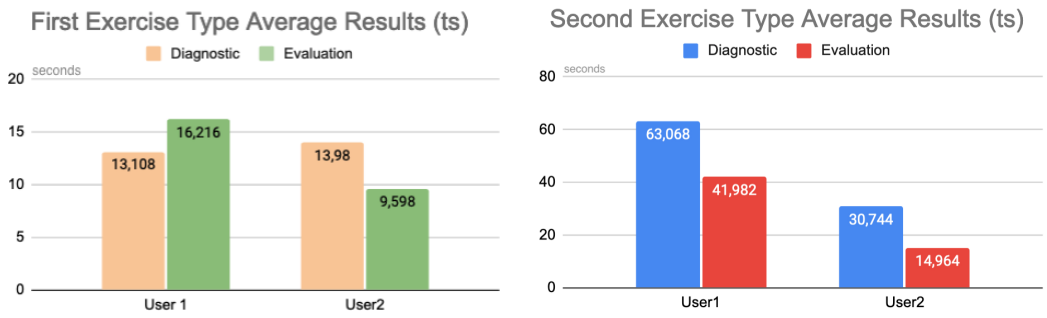


Figure 49: Average completion times for both types

5.6.2 *Answer Complexity*

Figures 50 and 51 show the difference in complexity between the diagnostic and evaluation exercise answers. Such as before, these results are separated by user representing the five exercise results. In the mentioned figures it is possible to observe that either the complexity level is the same or it increases between diagnostic and evaluation phases.

In order to better understand these results, Figure 52 shows the average complexity level of the answer in the different testing phases.

Figure 52 clearly shows that, for the fist exercise type, none of the users could perform better in the evaluation phase. This was expected given that in this exercise type the user must only dictate the path described in the tactile board, (remember Figure 61).This is, it is normal that the user describes the path in the same way.

On the other hand in the second group of exercises, both of the Testers performed better in the evaluation phase, leading to an increase of proficiency (more sophisticated answer) of 15%.

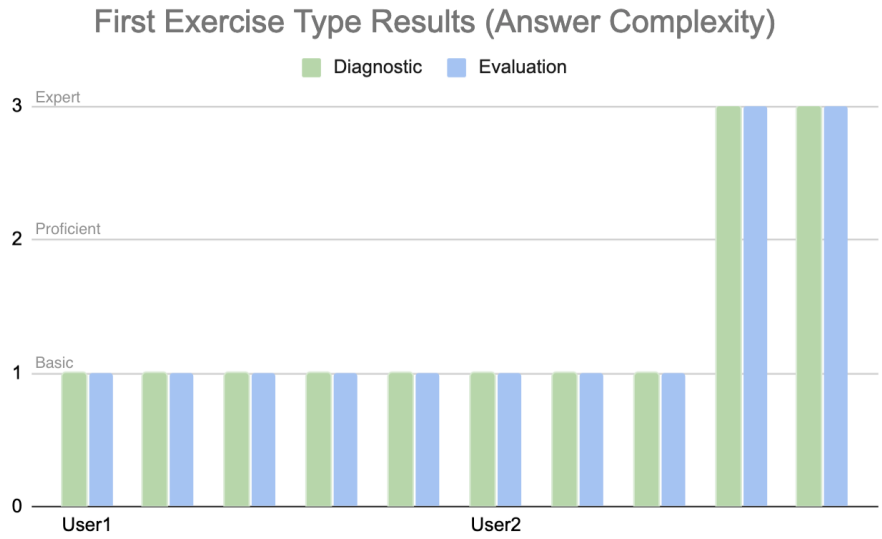


Figure 50: Comparing the Anwers Complexity for the first type in both diagnostic and evaluation phases

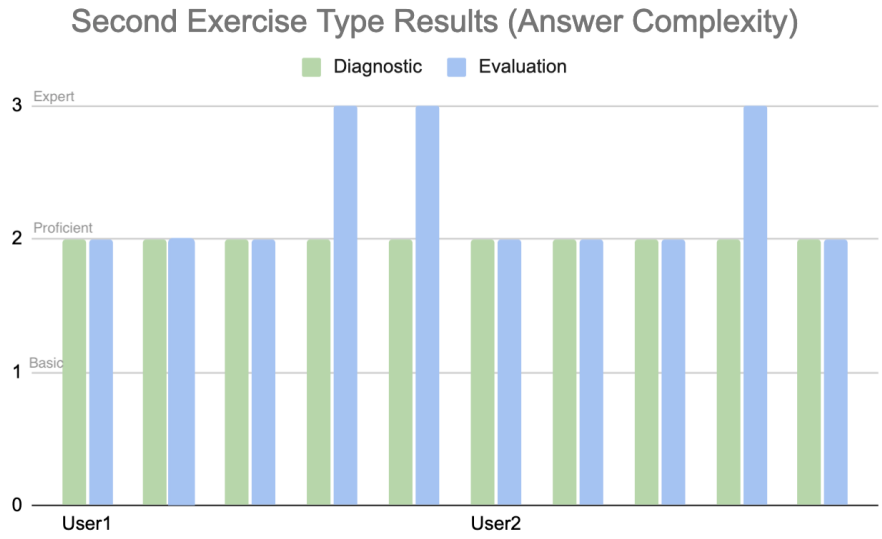


Figure 51: Comparing the Answers Complexity for the second type in both diagnostic and evaluation phases

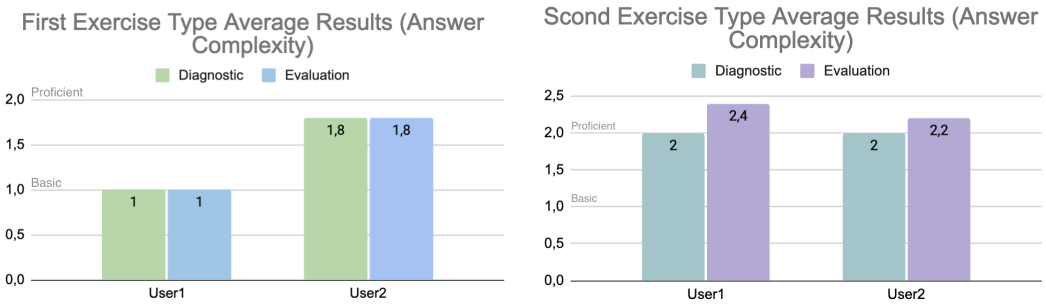


Figure 52: Average answer complexity for both types

5.6.3 *Number of Instructions*

Regarding number of instructions, Figure 53 shows the difference in number of instructions between diagnostic and evaluation phases. As mentioned before, this metric only applies to the second exercise type because, only in that group the user has to design a path, having in mind the goal of reducing to the minimum the number of instructions.

As seen in the given plot, the instructions number either decreases or stays the same between diagnostic and evaluation phases.

To better understand the results obtained, Figure 54 shows the average instruction number of both the diagnostic and evaluation phases. This plot helps to see the improvement of the answers given between the two stages of testing.

Analyzing carefully the performance of both Testers along the experiment, it is possible to conclude that both users improved their answers between diagnostic and evaluation sessions. This lead to, in average, a decrease in instructions number of 36%.

It is important to notice that the second test user is significantly older than the first one having noticed a bigger improvement in answer quality in the first test user.

With these results it is proven that Melodic can train Computational Thinking skills to visually impaired people.

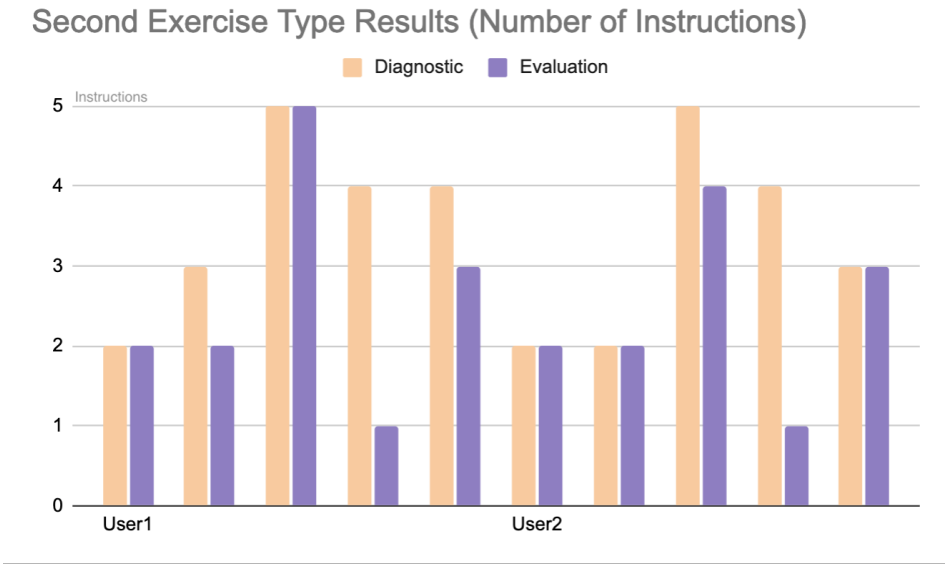


Figure 53: Comparing the number of instructions for the second exercise type in both diagnostic and evaluation phases

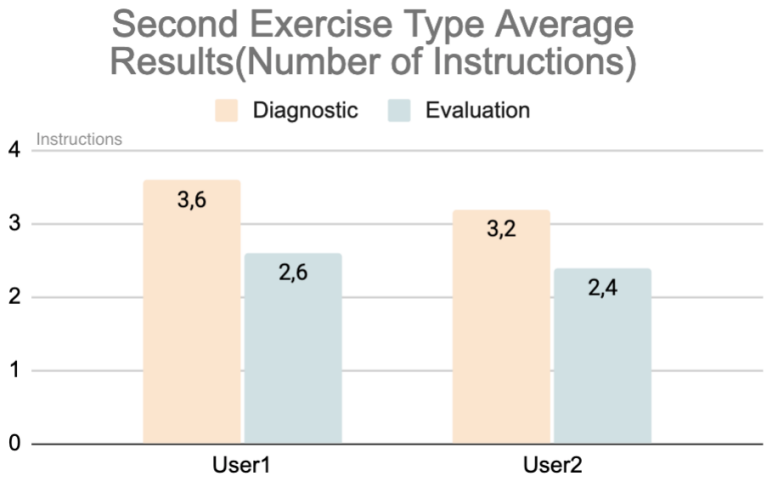


Figure 54: Average instruction number for both users

5.7 SUMMARY

In this chapter all the stages of the process to test if Melodic actually develops Computational Thinking skills are described.

To prove that Melodic develops Computational Thinking, firstly it was necessary to come up with a way to evaluate the state of the user's Computational Thinking skills before and after the training session with Melodic. For this some existing exercises were adapted to suit the

visual impaired users. Using the adapted set of exercises it was possible to perform some Diagnostic and Evaluation tests before and after the training with Melodic.

Then an analysis to the test results was performed. All the values collected from the completion of the test exercises were analyzed and documented to access if Melodic develops Computational Thinking skills. These results were separated in two main groups, Diagnostic and Evaluation, where the first is performed before the training with Melodic and the second one after.

Taking into account the positive results obtained from these two evaluation moments it is possible to state with confidence that Melodic develops Computational Thinking skills.

6

CONCLUSION

Throughout this document a main subject was addressed, teaching Computational Thinking to visually impaired children. Regarding this topic a literature review in the related fields was done and documented in order to understand the most used and best techniques to accomplish the objectives. So, Chapter 2 presents the following topics: Visual Impaired Characterization , Computational Thinking Characterization and Teaching Computational Thinking to Visual Impaired People Through Music. Concerning Computational Thinking Characterization the following aspects were discussed: Teaching Computational Thinking to the Visual Impaired and Teaching Computational Thinking Through Music. Referred Chapter 2 reflects the work scheduled from the first to third months. This is Bibliographic search in the technology for blind people field and the study of techniques to teach Computational Thinking.

Then in Chapter 3, the decisions that were taken in the creation process of Melodic were presented. In that chapter it can be seen the System Architecture, its components and main features for better understanding Melodic's structure.

After this, all the steps to develop and implement this System are documented. Chapter 4 discusses how the hardware was designed and produced, as well as the mobile application implementation. Alongside this, some examples of usage were discussed and described.

Finally, in Chapter 5, the testing of the complete (hardware and software) Melodic System is described. That chapter presents the process of creating the experiments and the final results. Although few, the results collected and analyzed led to a positive evaluation of Melodic in terms of Computational Thinking development capabilities. Here it was proven the objective of this thesis having positive results between the diagnostic and evaluation phases. This demonstrated that the user developed Computational Thinking skills with the use of Melodic and was a great outcome for the extensive work done.

From the Master's work resulted an article publised and presented at an international conference on programming education, Melodic - Teaching Computational Thinking to Visually Impaired Kids, published in Second International Computer Programming Education Conference (ICPEC 2021) (Costa et al., 2021). The presentation attracted the attention of the audience which returned a positive feedback.

6.1 FUTURE WORK

Future work would be, in a first instance, perform more tests with new test users in order to strengthen Melodic's proposal. Then, with a stronger proof that this System develops Computational Thinking skills, the objective would be to implement in schools, first in a small scale and then in a bigger one.

Another interesting investigation would be to adapt Melodic to other audience. Given the interactive and intuitive interface of the System, it is possible to believe that it could be suited for other user groups, such as older people or people with other impairment.

Lastly, an interesting feature to implement would be a way to output the current sequence. With this the user, during the reading of the sequence, would be able to perceive what blocks were scanned until that point.

A

COMPLEMENTARY MATERIAL

A.1 QR CODES

Figure 55: QR Codes and their Meanings

Figure 56: QR Codes and their Meanings

8. (7) Number 8

12. (endLoop) Loop end

9. (8) Number 9

10. (9) Number 10

11. (loop) Loop start

Figure 57: QR Codes and their Meanings

A.2 DIAGNOSTIC EXERCISES

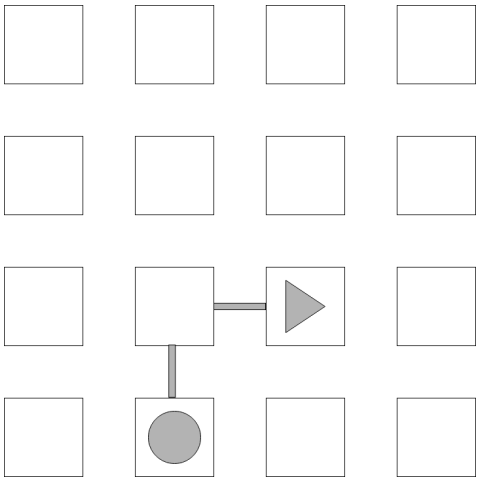


Figure 58: First diagnostic test of the first type

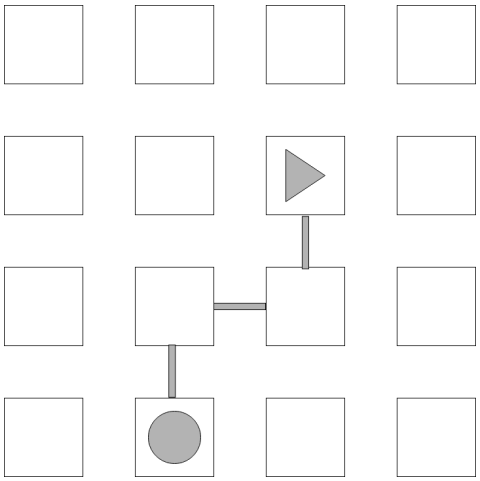


Figure 59: Second diagnostic test of the first type

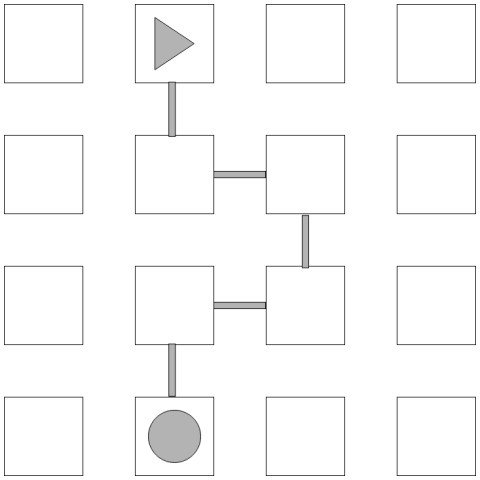


Figure 60: Third diagnostic test of the first type

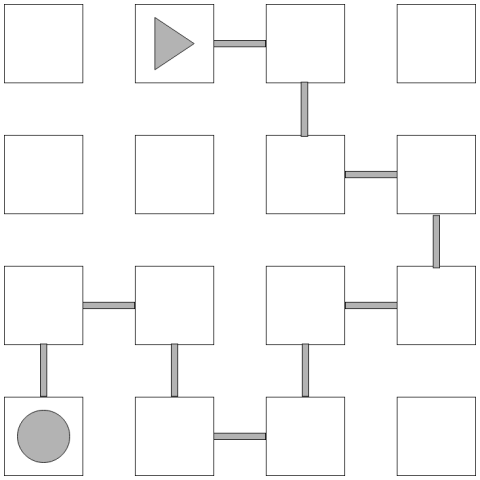


Figure 61: Fourth diagnostic test of the first type

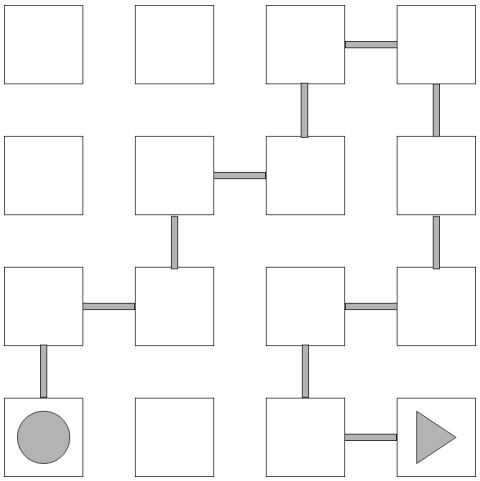


Figure 62: Fifth diagnostic test of the first type

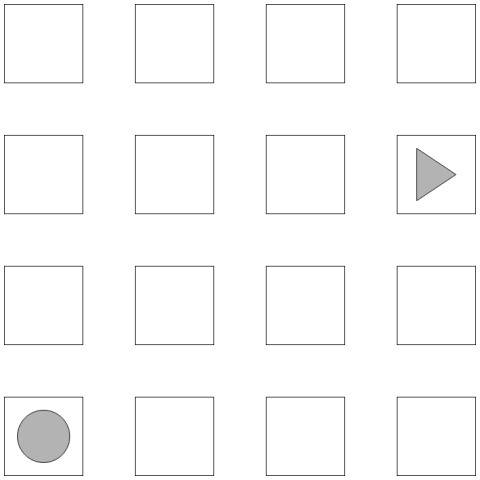


Figure 63: First diagnostic test of the second type

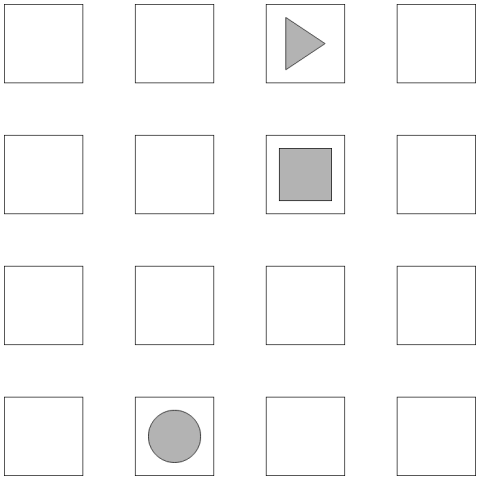


Figure 64: Second diagnostic test of the second type

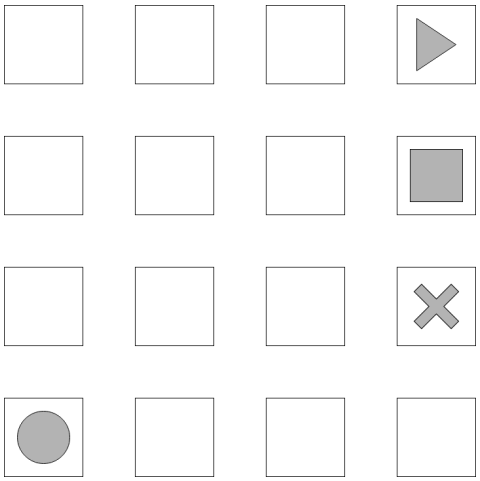


Figure 65: Third diagnostic test of the second type

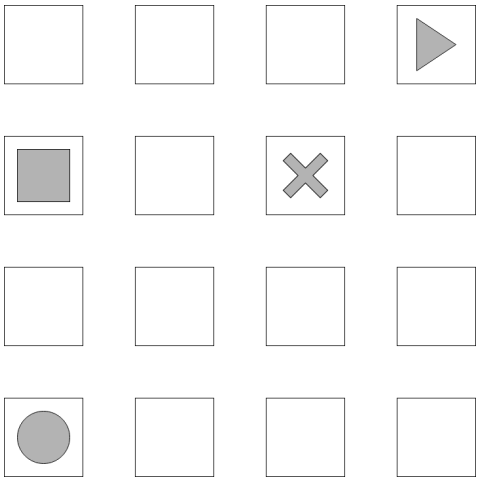


Figure 66: Fourth diagnostic test of the second type

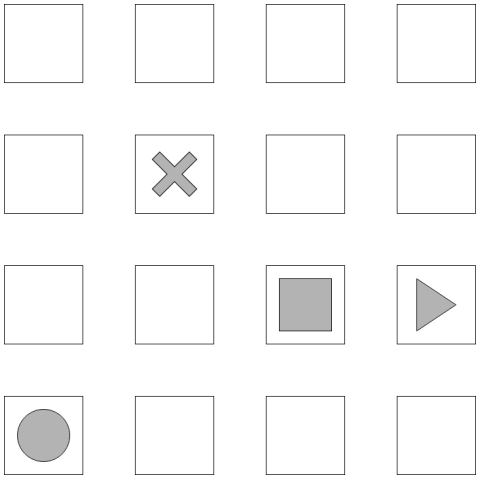


Figure 67: Fifth diagnostic test of the second type

A.3 EVALUATION EXERCISES

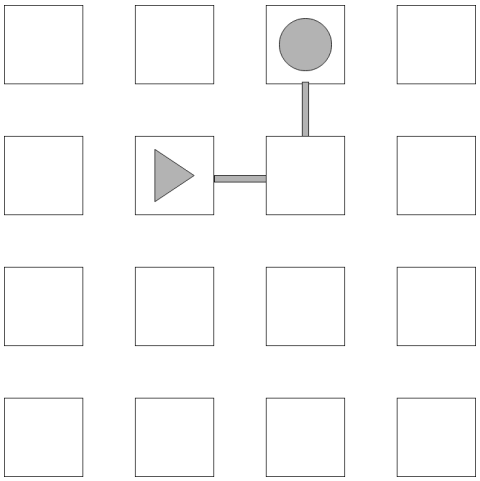


Figure 68: First evaluation test of the first type

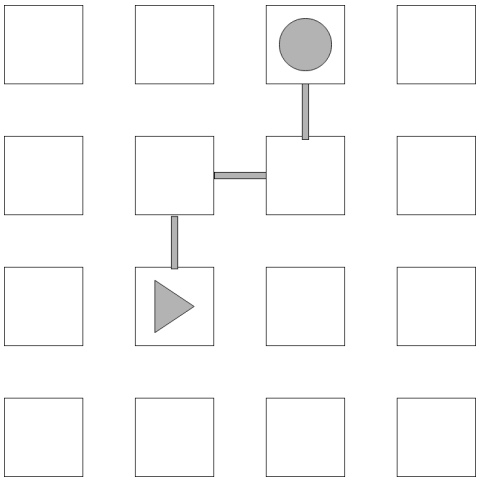


Figure 69: Second evaluation test of the first type

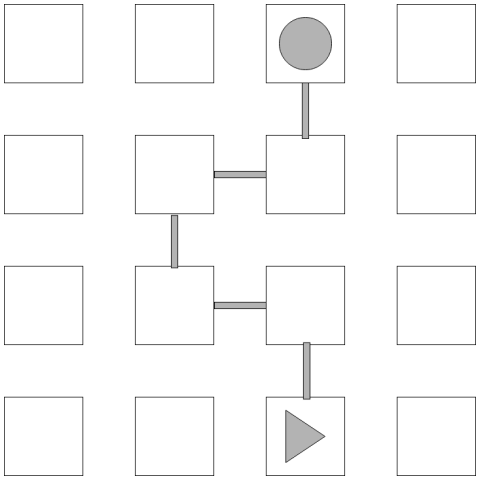


Figure 70: Third evaluation test of the first type

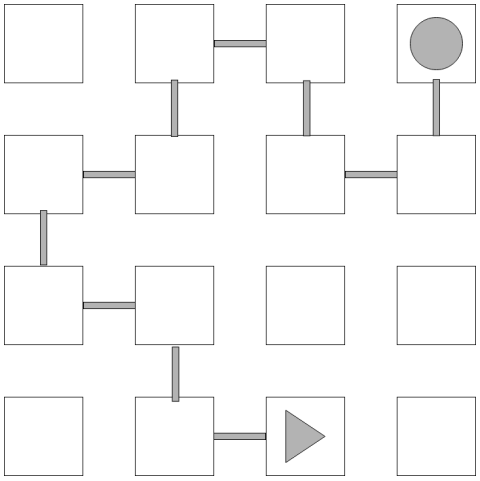


Figure 71: Fourth evaluation test of the first type

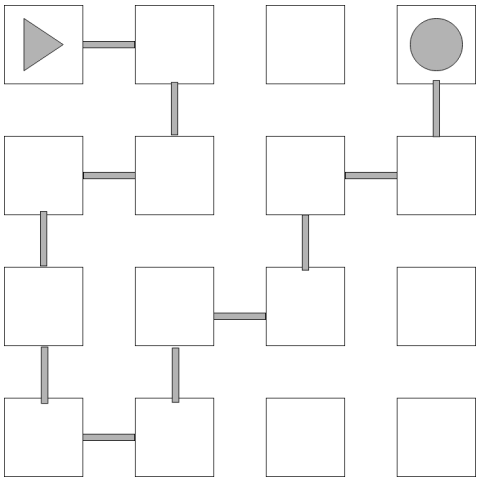


Figure 72: Fifth evaluation test of the first type

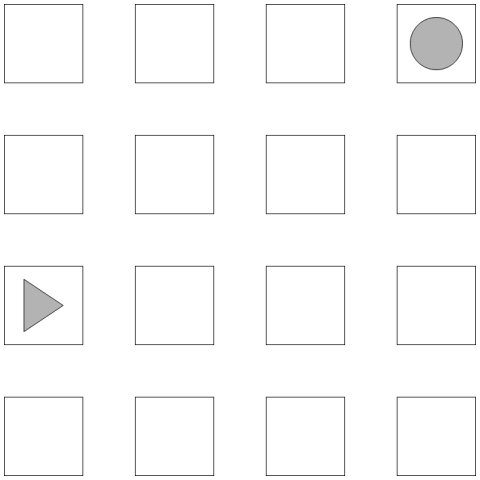


Figure 73: First evaluation test of the second type

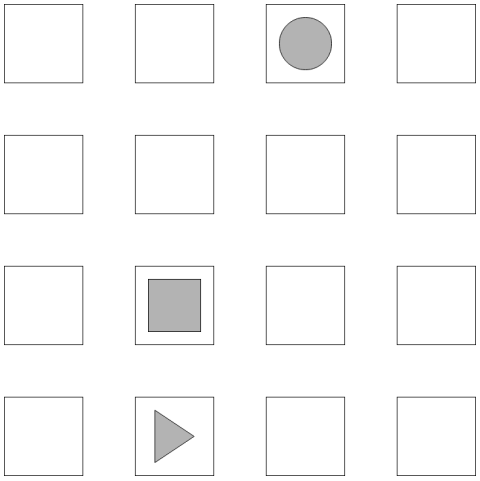


Figure 74: Second evaluation test of the second type

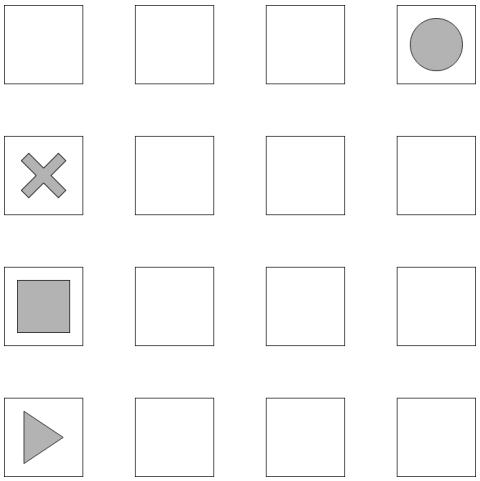


Figure 75: Third evaluation test of the second type

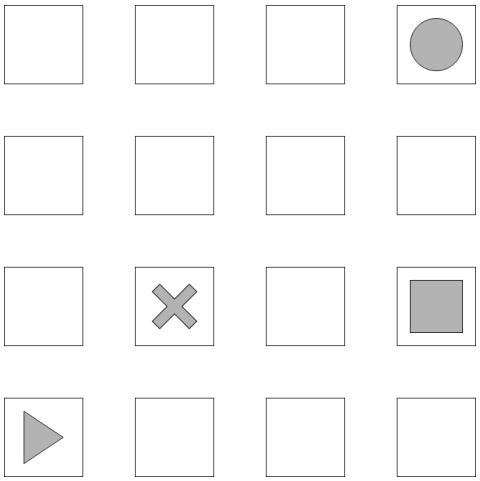


Figure 76: Fourth evaluation test of the second type

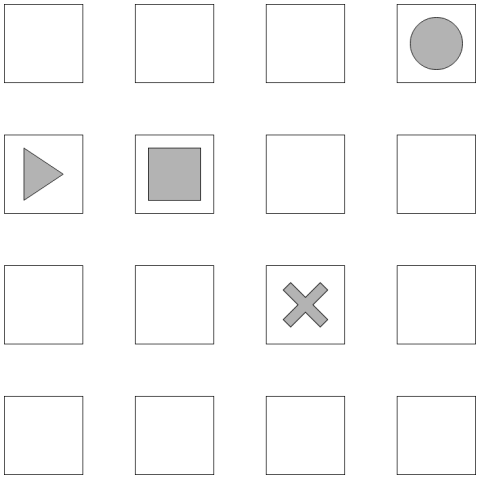


Figure 77: Fifth evaluation test of the second type

BIBLIOGRAPHY

Cristiana Araújo, Lázaro Lima, and Pedro Rangel Henriques. An Ontology based approach to teach Computational Thinking. In Célio Gonçalo Marques, Isabel Pereira, and Diana Pérez, editors, *21st International Symposium on Computers in Education (SIIE)*, pages 1–6. IEEE Xplore, Nov 2019. ISBN 978-1-7281-3182-5. doi: https://doi.org/10.1109/SIIE48397.2019.8970131.

Ana Azevedo, Cristiana Araújo, and Pedro Rangel Henriques. Micas, a Web Platform to Support Teachers of Computing at School. In A. J. Osório, M. J. Gomes, and A. L. Valente, editors, *Challenges 2019: Desafios da Inteligência Artificial, Artificial Intelligence Challenges*, pages 625–641, Braga, Portugal, May 2019. Universidade do Minho. Centro de Competência.

Judith Bell and Tim Bell. Integrating computational thinking with a music education context. *Informatics in Education*, 17:151–166, 09 2018. doi: 10.15388/infedu.2018.09.

CEA. *Council for The Curriculum Examinations and Assessment: Computing at School: Northern Ireland Curriculum Guide for Post Primary Schools. Computing at School (2018)*. 2018. ISBN 978-1-78339-521-7.

Eddy K.M. Chong. Teaching and learning music through the lens of computational thinking. In *Proceedings of the International Conference on Art and Arts Education (ICAAE 2018)*, pages 1–7. Atlantis Press, 2019. ISBN 978-94-6252-744-7. doi: https://doi.org/10.2991/icaae-18.2019.1. URL https://doi.org/10.2991/icaae-18.2019.1.

Rui Costa, Cristiana Araújo, and Pedro Rangel Henriques. Melodic - Teaching Computational Thinking to Visually Impaired Kids. In Pedro Rangel Henriques, Filipe Portela, Ricardo Queirós, and Alberto Simões, editors, *Second International Computer Programming Education Conference (ICPEC 2021)*, volume 91 of *Open Access Series in Informatics (OASIcs)*, pages 4:1–4:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-194-8. doi: 10.4230/OASIcs.ICPEC.2021.4. URL https://drops.dagstuhl.de/opus/volltexte/2021/14220.

Elizabeth Erwin, T. Perkins, Jennifer Ayala, M Fine, and E Rubin. "you don't have to be sighted to be a scientist, do you?" issues and outcomes in science education. *Journal of visual impairment & blindness*, 95:338–352, 06 2001.

William Fraser and Mbulaheni Maguvhe. Teaching life sciences to blind and visually impaired learners. *Journal of Biological Education - J BIOL EDUC*, 42:84–89, 03 2008. doi: 10.1080/00219266.2008.9656116.

J. Freeman. *The Psychology of Gifted Children*. Wiley Series in Developmental Psychology and Its Applications. Wiley, 1985. ISBN 9780471102557. URL https://books.google.pt/books?id=dtB9AAAAMAAJ.

Vikas Gupta. Teaching programming to children using stories, music, and puppeteering, 2013. URL https://joanganzcooneycenter.org/2013/11/12/teaching-programming-to-children-using-stories-music-and-puppeteering/. [Online; accessed 7-December-2020].

Mark Guzdial. Teaching programming with music: An approach to teaching young students about logo. pages 2–4, 1991. URL https://el.media.mit.edu/logo-foundation/resources/papers/pdf/teaching_progr.pdf.

N.G. Haring and R.L. Schiefelbusch. *Methods in Special Education*. McGraw-Hill series in education: Psychology and human development in education. McGraw-Hill, 1967. URL https://books.google.pt/books?id=ADQ1AAAAMAAJ.

Jan L. Plass, Bruce D. Homer, and Charles K. Kinzer. Foundations of game-based learning. *Educational Psychologist*, 50(4):258–283, 2015. doi: 10.1080/00461520.2015.1122533. URL https://www.tandfonline.com/doi/abs/10.1080/00461520.2015.1122533.

Alpay Sabuncuoglu. Tangible music programming blocks for visually impaired children. feb 2020.

Victor Silva, Heleniara Moura, Suelen Paula, and Ângelo Jesus. Algo+ritmo: Uma proposta desplugada com a música para auxiliar no desenvolvimento do pensamento computacional. In *Anais do XXV Workshop de Informática na Escola*, pages 404–413, Porto Alegre, RS, Brasil, 2019. SBC. doi: 10.5753/cbie.wie.2019.404. URL https://sol.sbc.org.br/index.php/wie/article/view/13188.

Salete Teixeira, Diana Barbosa, Cristiana Araújo, and Pedro Rangel Henriques. Improving Game-Based Learning Experience Through Game Appropriation. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 27:1–27:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-153-5. doi: 10.4230/OASIcs.ICPEC.2020.27. URL https://drops.dagstuhl.de/opus/volltexte/2020/12314.

Wikipedia contributors. Computational thinking — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Computational_thinking&oldid=983641161. [Online; accessed 6-December-2020].

Stephen Wolfram. How to teach computational thinking, 2016. URL https://writings.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/.

María Zapata-Cáceres, Estefanía Martín-Barroso, and Marcos Román-González. Computational thinking test for beginners: Design and content validation. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1905–1914, 2020. doi: 10.1109/EDUCON45650.2020.9125368.