CodeBuddy: Teaching Visually Impaired Children How to Code

João Tiago Abreu
Faculdade de Ciências
Universidade de Lisboa
fc48858@alunos.fc.ul.pt

Catarina Silva
Faculdade de Ciências
Universidade de Lisboa
fc41677@alunos.fc.ul.pt

Daniel São Pedro
Faculdade de Ciências
Universidade de Lisboa
fc37704@alunos.fc.ul.pt

ABSTRACT

This paper proposes an approach to teaching the fundamental concepts and reasoning of programming to visually impaired children: CodeBuddy. By incorporating learning into something children to solve, with the help of CODI, our virtual assistant. Using the TopCode computer vision library, speech recognition and sound/movement feedback through a robot, our solution works particularly for visually impaired children: with only touch and hearing, children can easily understand what the games goals are and how to play them (what pieces to use, what they are for), with the fun part of having a robot allowing them to physically perceive the results of their programming.

In this paper we go from a brief description of our research work, our solution definition and architecture, to demonstrate some of the results of the tests we performed on the system. We also mention some of the intended future work to achieve a more complete and flexible solution.

Keywords

Learning; Programming; Visual Impaired; Speech Recognition; Object Recognition; Robot API; Speech Synthesis

1. INTRODUCTION

Learning programming is becoming an important part in children's education. The logical thinking required to perform even simple operations should help them in their further education, not only in programming but also in other areas that require problem solving. However, because of their short attention span, traditional approaches to teaching programming are often inappropriate for children. New methods are emerging, allowing them to play games while they learn, since it has been proven that children are better able to learn while having fun. [1]

There are a few software and toys that teach children how to learn programming key concepts and thinking, including giving simple instructions in specific order, conditional statements or functions with arguments. However, some of these tasks can be difficult for visually impaired children. Without reliable access to clear visual information, these children must rely on additional modes of learning, such as learning through touch and sound. The idea of our educational project is to use the already explored concepts of programming learning for children, implementing them through touch and sound, instead of in a visual way. Tactile representations must make sense to a child in order to convey meaning [2], so we tried to have that in mind with the definition of the game boards, or the blocks they need to present their solution to solve the game in question. The audible feedback and the speech recognition components might improve the way the child perceives the game, as well as her/his interaction with the system. Which, in the end, can increase her/his motivation and engagement with the game.

The solution presented in this paper intends to be an open source project, so that teachers/educators feel free to explore new ways of teaching and playing the game, as well as trying different methods for the physical modelling (for example, using different materials, more adapted to the needs of each child).

2. RESEARCH WORK

2.1. Related Projects

Code Jumper (Microsoft)

A kit that uses differently shaped physical blocks that can be attached in patterns. Each block represents a line of code. When the blocks are attached together and buttons are adjusted, the series creates an audible output, like a song or a joke. Children can touch and manipulate the elements of the programming environment, while they develop programming skills, regardless of their level of vision.

Swift Playgrounds (Apple)

An app that require no coding knowledge. The goal is to solve puzzles to learn the basics of Swift (a programming language created by Apple). The app comes with a set of lessons using real code to guide a character through a 3D world. Children have the possibility to move on to more advanced concepts. It involves the learning of more complex commands, such as functions, loops, or even the notion of types.

Osmo

A collection of 5 smart games using physical action (as tangram, games with words and numbers,...). Osmo has additionally an iPad base that captures and reacts to every real-live move.

Strawbies!

Developed by TIDAL Lab, Strawbies! is also an iPad game designed for children. Players use wooden coding blocks and an Osmo to guide the character through an expanding world. The game logic supports simple and complex problem solving.

Tern

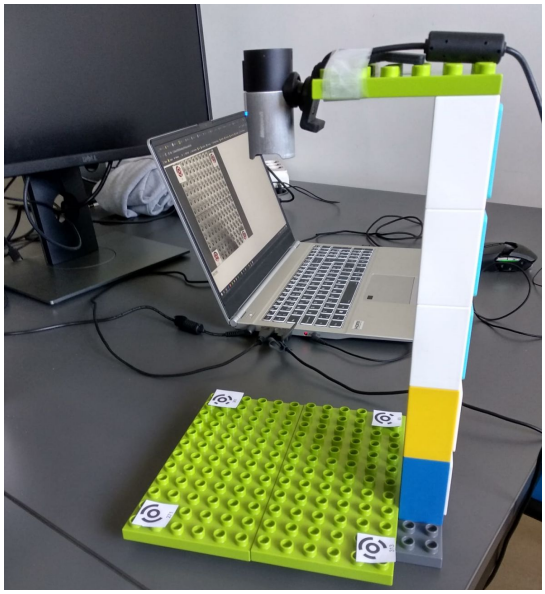
Tern (by TIDAL Lab) is a tangible computer language also designed to introduce children to computer programming in educational settings. Tern uses a standard webcam connected to a

desktop or laptop computer to take a picture of the built program (with wooden blocks representing the instructions), which is then converted into digital code using TopCodes library.

2.2. Object Detection

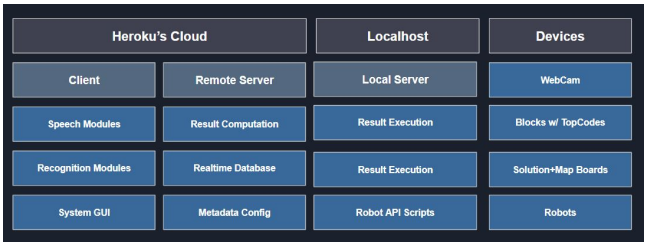
TopCodes

The TopCode [4] computer vision library is designed to identify and track tangible objects on a flat surface. If a physical object is tagged with a TopCode, the system returns an id number, the location, angular orientation and diameter of the tag. The TopCode library will identify 90 unique codes. In this project, this library is used to identify all the different blocks in the game, so that the system can establish what are the limits of the maze/game board as well as all the other elements in it. It is also used to recognize the set of different commands children can use to solve the problem.

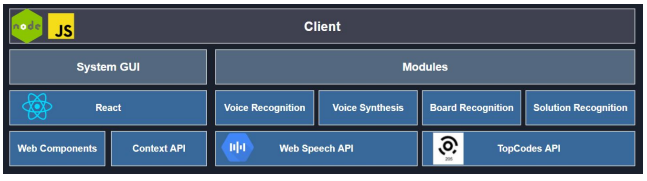


3. SOLUTION DEFINITION

3.1. Architecture Overview



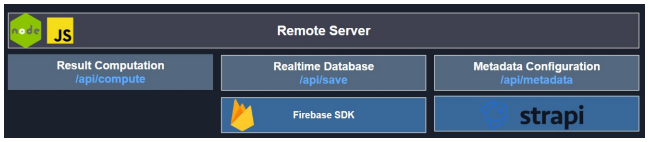
3.1.1. Client / GUI



For this project to work we needed to have some sort of User Interface to guide the teacher and eventually include Speech Recognition and Synthesis Modules to communicate with the Visually Impaired student and include the Map and Commands Recognition Modules to identify both the board and the commands issued, we call this our **client**:

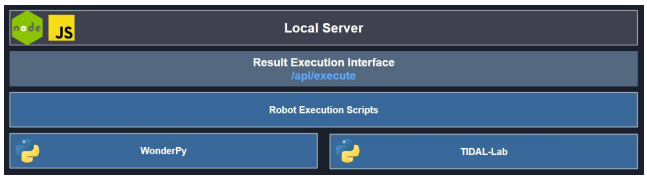
We built our client in React using Web Components and leveraging the easy to manipulate React Context API to manage application state. We included modules for Speech Recognition based on the Web Speech API [3] that enables web apps to handle voice data, currently only implemented by Google Chrome and Chrome for Android. For the object recognition part we leverage TopCodes API [4], particularly Michael Horn's implementation of the library in JavaScript [5].

3.1.2. Remote Server && Meta



We then needed a remote server that could receive requests from CodeBuddy users that would perform result computation when receives a board configuration, game type and a list of commands to execute as well as realtime database communication (later to be used in analytics, etc), we choose Firebase [6] for this since it was the most performant and accessible option. It should also include some metadata configuration enabling the possibility to have pre-configured game modes, maps, list of current compatible robots and their available block commands, we choose strapi.ai [7]. We call this our **remote** (server).

3.1.3. Local Server



Lastly we needed to setup a simple API able to run on a VM or Container enabling us to leverage some Robot API's, in this case WonderWorks DASH non-official API's were only available on MAC or Linux, this API would allow us to receive a simple execution call with the list of actions to execute, enriched with feedback post result processing, we call this our **local** (server).

Most of the implementation of our system is written in JavaScript, while the Robot Interface is written in Python due to the available libraries. Overall our vision is to have a system as much decoupled as possible and able to scale with the possibility of each user to have their own configurations and access its analytics. Each module of the system can be replaced with a more advanced technology if required, for example, we could add ChatBot capabilities to our application, including Natural Language Processing for a more natural feel and advanced interaction with CODI.

The entire solution is available on the project's GitHub link [6] and its licensed under Apache 2.0 so anyone can contribute to further evolve it and create an accessible system for children to learn programming.

4. HOW TO PLAY

4.1. Setting Up

The main proposed game is a maze where the child must give a set of instructions in order to find the way to reach (one of) the finish cell(s).

We chose to model the physical part of the game with LEGO Double blocks. However, the idea is to give some freedom of choice regarding all the materials used in the modelling.

A webcam must be set to detect the board, along with the 4 blocks representing the corners. These 4 blocks need to be set first in the board, so that our system can recognize and calculate the distance between each of them, to virtually determine "a cell", given the boards size (eg, 6x6). Additionally, the start and finish blocks also need to be set on the board, otherwise the solution will not be computed. Our board game used for testing has a dimension of 6x6 cells (each occupying cell is represented by 2x2 LEGO pins). Then, the maze is built according to the complexity we want to see in the game. The image captured from the camera as well as the configuration of the game and all the changes that happen can be observed in our client interface. The various blocks are marked with different TopCodes, to provide different meaning, according to the presented list (found in Attachment A).

4.2. Detecting the Map

Visually impaired children can feel the map structure/obstacles using touch, while a digital representation is built in our system, thanks to the continuous detection built in our system. If a block is changed, the system will change the board representation, without the need to re-start the detection or any additional commands.

4.3. Detecting commands

Using the instruction blocks, players can choose any sequence of instructions to be validated by the system. To do that, the webcam must be rotated 180º, and the player must give the voice instructions "recognize commands", or click on the corresponding button on the screen of our interface (which could also be available through a keyboard shortcut). Then, she/he should dispose the sequence of blocks in the area captured by the camera: the system reads the instructions from left to right.

We did several iterations through this concept, including having a block representing START and another representing END but we felt it could ruin the user experience and simply having an additional command ordering the system to compute result/execute the outcome on the robot would be a lot better.

4.3. Saving Board to Database

We added a block called "SAVE" which when detected automatically saves the map configuration on the database, this is an additional way of interacting with the system, which also saves configuration to the database through voice commands "save map to database".

The system also deletes unknown commands/codes or invalid actions (such as malformed loops).

4.4. Compute solution and add feedback

(this has not yet been implemented) Since the system identifies the direction for our starting point and previously excluded invalid actions, it will be able to then compute the outcome for each action, essentially adding a "success" or "failure" flag after each step. Depending on the robot sent together with the request for computing solution, different feedback codes might be added for including robot noises (sound feedback), lights/movements to simulate a lively interaction of success or failure.

4.5. Robots programming and interface

(this has not yet been fully implemented) A simple Interface, created in Python, through a switch executes directly on the console the script for each different Robot, regardless of the language it was implemented in, simply providing it the final array of actions to execute and initial position. Additionally a second endpoint would override execution and stop it moving the robot back to the initial position.

5. TEST RESULTS

We ran a few preliminary tests focused on the interaction with the system, in particular Voice Recognition attained a success rate of 16 accurate recognitions out of 22 tests performed (72%+).

After tweaking our RegEx Patterns further alongside applying specific grammar to Google SpeechRecognition service, we increased our success rate to 19/22 (86%+).

As for Map/Commands Recognition optimal light conditions are required to ensure that the recognition system doesn't recognize a new board when in fact it's the same one previously identified, this of course can be worked around programmatically. It's also recommended to avoid having tape on top of the TopCodes to avoid glare effect.

6. FUTURE WORK

Due to time and resource restrictions, there is quite a lot of work we would like to implement, in particular:

1. Result computation module;
2. Robot Execution Interface;
3. Robot Execution Script for WonderWorks DASH;
4. ChatBot Integration to replace RegEx Patterns, we started prototyping with Google's DialogFlow [9]
5. Further integration with other Robots, in particular LEGO EV3
6. Further enhances on the Web GUI
7. User Accounts and Analytics with Gamification included with the main goal to provide insight on a user progress on his/her programming skills
8. New game modes and more sophisticated blocks, including representation of functions
9. 3D Modelling of CodeBuddy's specific blocks with TopCodes embedded and braille
10. CodeBuddy's Command Line Interface for an easy project bootstrap
11. Improve communication between system and robot using ZeroMQ [10]

7. ACKNOWLEDGMENTS

We would like to thank:

Luis Carriço, professor of Advanced Interaction Techniques 2018-2019 at University of Lisbon, for giving us the opportunity and exposure to this kind of research topics.

Tiago Guerreiro, professor at the University of Lisbon for guiding us through the project execution.

8. REFERENCES

[1] Zaharija, G., Mladenovic, S., Boljat, I., Introducing basic programming concepts to elementary school children, 2013, University of Split, Faculty of Science.

[2] Chen, D., Downing, J., Rodriguez-Gil, G.. 2007. Tactile Learning Strategies for Children who are Deaf-Blind: Concerns and Considerations from Project SALUTE. California State University, Northridge

[3] MDN web docs mozilla, Web Speech API, https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

[4] TopCodes, Tangible Object Placement Codes, http://users.eecs.northwestern.edu/~mhorn/topcodes/

[5] TIDAL-Lab TopCodes' JavaScript Implementation, https://github.com/TIDAL-Lab/TopCodes/tree/master/javascript

[6] Google's Firebase: https://firebase.google.com

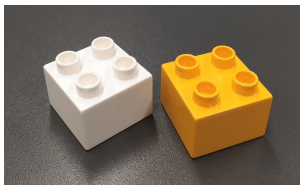
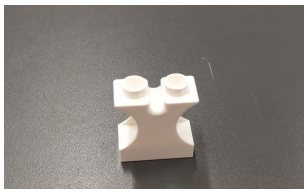
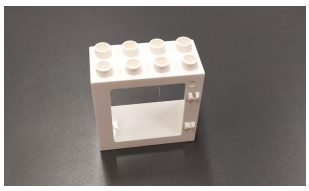
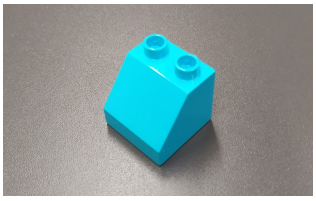
[7] Strapi.io - Content Management System: https://strapi.io/

[8] CodeBuddy's GitHub Repository: https://github.com/jtiagodev/codebuddy/blob/master/README.md

[9] Google's Dialogflow: https://dialogflow.com/

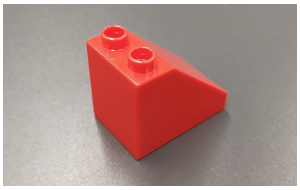
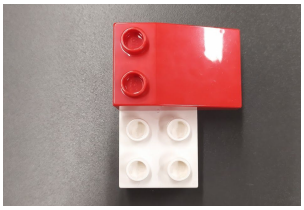
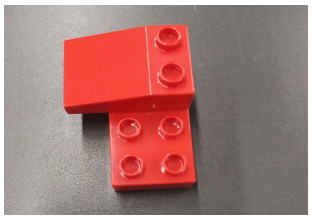
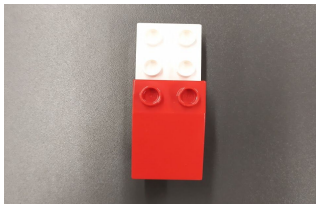
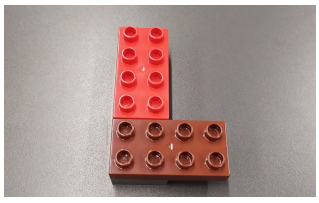
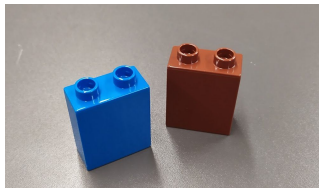
[10] ZeroMQ - Distributed Messaging: http://zeromq.org/

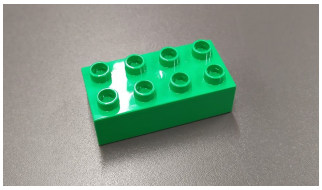
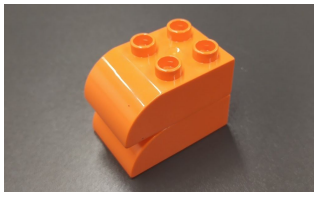
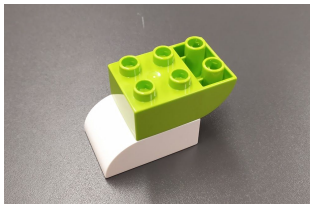
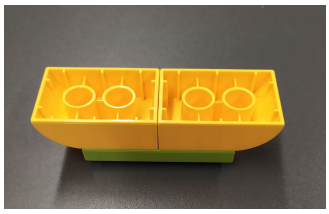
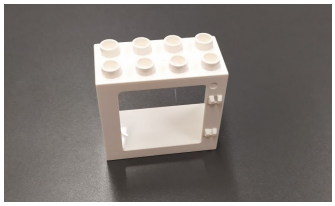
ATTACHMENT A

Board game modeling		
Block/set of blocks	**Representation**	**TopCode**
	Walls: Game board boundaries/delimiters Corners of the game board (T,L),(T,R),(B,L),(B,R) Finish cell	61 **31,47,55,59** *(1) (2)* **93** *(1)*
	Starting position Represents the robot's initial position	**91** *(1)*
	Points: The player can choose a path containing these blocks in order to obtain a better score in the game	103
	Door: Used as a shortcut in the maze (2 cells in the board)	79,79
	Water: Obstacle in the game, the player must find a way to move through it (eg, swim)	87

(1) required to be present on the board so it's considered valid
(2) shouldn't have any duplicates to avoid bugs

ATTACHMENT B

Blocks of instructions		
Block/set of blocks	**Representation**	**TopCode**
	Moving forward: Moves the robot/player one cell forward	109
	Rotate right (90º): Rotates the robot/player to the right, in the same position	143
	Rotate left (90º): Rotates the robot/player to the left, in the same position	121
	Rotate backwards (180º): Rotates the robot/player backwards, in the same position	151
	Initiate loop: Used to repeat the next instruction in the sequence	155
	Stop loop: Placed immediately after the last instruction “inside” the loop, indicates the player wants to stop the repetition of the sub-sequence	157
	Number of iterations	

	<u>If:</u> The beginning of an if clause. It needs to be followed by a block of the board game (ex: if -> wall)	171
	<u>Then:</u> Specifies what instruction to do. Placed after the “if” and object block	173
	<u>Else:</u> Placed after the instruction next to the “else” block	179
	<u>Boat / Swim:</u> Used to cross one cell of water	115
	<u>Open door:</u> Gives the instruction to open the door through the shortcut, without the need of a “move forward” block	117