

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



# **Robotic Simulator for the Tactode Tangible Block Programming System**

**Márcia Alves**

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Sousa

Second Supervisor: Ângela Cardoso

October 20, 2019



# Resumo

O desenvolvimento da tecnologia tem vindo a crescer. A procura por pessoas formadas nas áreas da Ciência, Tecnologia, Engenharia e Matemática, em inglês Science, Technology, Engineering, Mathematics (STEM) tem vindo a aumentar. De forma a acompanhar este crescimento procura-se uma evolução da educação para que as crianças desenvolvam hábitos, técnicas e raciocínios mais lógicos, para que possam entender e resolver problemas de carácter tecnológico. A este conjunto de conceitos chamamos Pensamento Computacional, em inglês, Computational Thinking (CT).

A programação tangível por blocos transforma a programação numa atividade acessível e direta, menos abstrata e compreendida com mais facilidade quando ligada à robótica. Usar robôs é uma forma de ensinar de maneira divertida, ajudando a desenvolver o pensamento computacional e aprendendo como funciona a tecnologia. Embora seja difícil para os professores incluírem novos temas no programa escolar o objetivo é interligar a robótica com o sistema de educação STEM. Um dos principais problemas para as escolas pode ser suportar o preço desse tipo de tecnologia para todos os estudantes.

O Tactode foi anteriormente proposto como um sistema de programação tangível, que, aliado a uma aplicação, permite programar robôs, semelhante a um jogo na sala de aula. Esta versão do Tactode, baseia-se na programação por blocos tangíveis, como peças de um puzzle que os estudantes têm de resolver. É possível criar um puzzle com peças tangíveis, tirar uma fotografia e fazer o seu upload na aplicação. Na aplicação, o puzzle é transformado em código executável para robô e é possível ver a sua execução num robô real.

O trabalho apresentado nesta dissertação adiciona um simulador ao Sistema de Programação Tangível Tactode, usando Ionic Framework para a programação da aplicação Web, fazendo grande uso da linguagem de programação TypeScript e da biblioteca Three.js. Desta forma, as crianças conseguem ver, na aplicação, uma simulação do código, num robô virtual, antes de executar no robô real. Podem também usar apenas como um jogo, brincando em qualquer lugar sem ter de levar o robô.

O Tactode pode ser usado com vários robôs, Ozobot, Cozmo, Sphero e Robobo, e plataformas não robóticas, Scratch e Python. Os movimentos no simulador estão limitados às funcionalidades de cada um. A fotografia captada é processada, colocada numa posição favorável para a sua visualização, executada no simulador ou enviada para a plataforma escolhida. Foram também realizados um conjunto de desafios, envolvendo geometria, matemática, ciclos e sensores, a serem ultrapassados pelos estudantes.

Posto isto, a sugestão é a utilização de um conjunto de peças de puzzle e um computador, Tablet ou até mesmo um Smartphone a ser partilhado pelos alunos, sendo a utilização do robô facultativa. Assim, permite-se uma redução de custos de equipamento e os estudantes podem usar o simulador para executar os seus puzzles. Desta forma, as crianças são capazes de aprender sobre robótica e ter uma lógica de programação, desde tenra idade, de uma forma divertida e também aumentar a cooperação, trabalhando em grupo para resolver um problema.



# Abstract

The development of technology has been increasing. The demand for people trained in the areas of Science, Technology, Engineering, and Mathematics (STEM) has been growing. In order to follow this development, an evolution of education is believed so that children develop more logical reasoning, habits and techniques to understand and solve technological problems.

Tangible block programming makes programming an accessible and direct activity, less abstract and more easily understood when connected to robotics. Using robots is a fun way to teach, helping students to develop computational thinking and learning how the technology works. Although it is difficult for teachers to include new things in the regular curriculum, because of the academic standards, the goal is to link robotics with the STEM education system. One of the main problems for schools may be to support the price of technology for all students.

Tactode was previously proposed as a Tangible Programming System, which, combined with an application, allows programming robots, similar to a game in the classroom. This version of Tactode is based on Tangible Block programming, like a puzzle that students need to solve. They can create a puzzle with tangible pieces, take a photo and upload it in the app. In the application, the puzzle is compiled into executable code for robot and it is possible to see its execution in a real robot.

The work presented in this dissertation adds simulation capabilities to the Tactode Tangible Programming System. The project used ionic as a programming framework for web programming and made extensive use of the TypeScript programming language and Three.js library. In this way, children can see, in the app, a virtual robot, simulating the code, or send it to the real robot. This is useful to test the program created, before execute in the real robot, or just to play anywhere without having to carry the robot.

Tactode can be used with Ozobot, Cozmo, Sphero or Robobo robots, and non-robotic platforms like Scratch and Python. The movements in the simulator are limited to the functionality of each. The captured photograph is processed, placed in a favorable position for its visualization and executed in the simulator or sent to the chosen platform. A set of challenges, involving geometry, mathematics, loops, and sensors, were created to be overcome by students.

Hereupon, the suggestion is the use of a set of puzzle pieces, a robot and a computer, Tablet or even a Smartphone to be shared by the students, so the use of the robot is optional. In this way, kids are able to learn about technology and robotics, to develop a programming logic since a young age, in a fun way. They are also able to improve team cooperation, working in a group to solve a problem.



# Agradecimentos

Primeiro, agradecer ao Professor Doutor Armando Jorge Sousa, professor da Faculdade de Engenharia da Universidade do Porto e orientador desta dissertação, pelo apoio e preocupação dado durante o meu percurso académico, incluindo este trabalho.

À Doutora Ângela Cardoso pela brilhante ideia do Tactode e por ser sempre tão prestável.

Aos meus pais por toda a paciência e por serem a voz da razão.

À Beatriz Cruz, ao Baltasar Aroso e ao João Costa por toda a amizade e apoio durante o meu percurso académico e ao longo desta dissertação.

Aos restantes amigos e colegas de curso, que me acompanharam e ajudaram ao longo do meu percurso académico.

Márcia Alves





*“Hands-on experience is the best way to learn about all the interdisciplinary aspects of robotics.”*

Rodney Brooks



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation and Goals . . . . .	2
1.3	Contributions and Publications . . . . .	2
1.4	Previous Publications and Initial State of Project . . . . .	3
1.5	Dissertation Structure . . . . .	4
<b>2</b>	<b>Fundamentals and Literature Review</b>	<b>5</b>
2.1	Block Programming . . . . .	5
2.2	Tangible Programming . . . . .	7
2.3	Three.js . . . . .	11
2.4	Programming Robotics in Elementary Schools . . . . .	13
2.4.1	Programming in Portuguese Elementary Schools . . . . .	14
2.4.2	Teachers Perception . . . . .	14
2.4.3	Logical Reasoning and Computation thinking . . . . .	15
2.4.4	Increased Interest in Computer Science . . . . .	16
2.4.5	Team Work . . . . .	16
2.4.6	Gender inequality . . . . .	16
2.5	Examples already tested . . . . .	17
2.5.1	Lego Mindstorm EV3 . . . . .	17
2.5.2	Ozobot . . . . .	17
2.5.3	Open Roberta Lab . . . . .	18
2.5.4	Blockly Games . . . . .	19
2.5.5	Scratch . . . . .	19
2.6	Conclusion . . . . .	21
<b>3</b>	<b>Problem Statement and Technological Selection</b>	<b>23</b>
3.1	Tactode 1 . . . . .	24
3.2	Tactode 2 . . . . .	25
3.3	Problem Definition . . . . .	26
3.4	Proposed Solution . . . . .	27
3.5	Development Technologies . . . . .	28
3.5.1	Web Programming . . . . .	28
3.5.2	TypeScript . . . . .	28
3.5.2.1	Promises . . . . .	29
3.5.3	Ionic Framework . . . . .	30
3.5.4	Three.js . . . . .	31
3.6	Conclusion . . . . .	31

<b>4</b>	<b>Tactode Application</b>	<b>33</b>
4.1	Requirements . . . . .	33
4.1.1	Non-Functional Requirements . . . . .	33
4.1.2	Functional Requirements . . . . .	34
4.1.2.1	Sequence Diagram . . . . .	35
4.1.2.2	Happy Path . . . . .	36
4.2	Design of Interface . . . . .	37
4.2.1	Rotating image . . . . .	37
4.2.2	Split Screen . . . . .	39
4.2.3	Creating the robot . . . . .	40
4.2.4	Robot Movements . . . . .	41
4.3	Simulator . . . . .	42
4.3.1	Abstract Syntax Tree . . . . .	43
4.3.2	Implemented Blocks . . . . .	44
4.3.2.1	Numbers and Letters . . . . .	44
4.3.2.2	Operators . . . . .	45
	Numerical Operators . . . . .	45
	Logical Operators . . . . .	46
	Comparison Operators . . . . .	47
4.3.2.3	Movements . . . . .	48
	Go Forward . . . . .	49
	Go Backward . . . . .	49
	Turn Left . . . . .	50
	Turn Right . . . . .	50
4.3.2.4	Control . . . . .	51
4.3.2.5	Variables . . . . .	52
4.3.2.6	Events . . . . .	53
4.3.2.7	Sensors . . . . .	53
4.3.2.8	Visual . . . . .	54
4.3.3	Challenges . . . . .	55
4.3.3.1	Regular polygon . . . . .	55
4.3.3.2	Obstacle Reaction . . . . .	56
4.3.3.3	Follow Line . . . . .	57
4.4	Conclusion . . . . .	58
<b>5</b>	<b>Results and Experiments</b>	<b>59</b>
5.1	Test Program 1 . . . . .	59
5.2	Test Program 2 . . . . .	62
5.3	Test Program 3 . . . . .	63
5.4	Conclusion . . . . .	66
<b>6</b>	<b>Conclusion and Future Work</b>	<b>67</b>
6.1	Future Work . . . . .	67
	<b>References</b>	<b>69</b>
<b>A</b>	<b>Tactode Pieces</b>	<b>75</b>
<b>B</b>	<b>Draft of conference paper</b>	<b>85</b>

# List of Figures

2.1	Blockly Games by Google. . . . .	6
2.2	MakeCode by Microsoft for micro:bit. . . . .	6
2.3	Scratch by MIT. . . . .	7
2.4	AlgoBlock [1]. . . . .	8
2.5	Electronic blocks [2]. . . . .	8
2.6	Cubelets [3]. . . . .	8
2.7	Code-a-Pillar [4]. . . . .	9
2.8	TagTile [5]. . . . .	9
2.9	Quetzal and Tern [6]. . . . .	9
2.10	T-Maze [7]. . . . .	10
2.11	Osmo Coding Family [8]. . . . .	10
2.12	Code Bits [9]. . . . .	10
2.13	Creating a Three.js Animation. . . . .	12
2.14	Three.js coordinate systems. . . . .	13
2.15	The Lego Mindstorms EV3 robot and its programming interface [10, 11]. . . . .	18
2.16	The Ozobot robot and OzoBlockly Editor [12]. . . . .	18
2.17	Open Roberta Lab. . . . .	19
2.18	Blockly Games. . . . .	20
2.19	Scratch. . . . .	20
3.1	Tactode 1 after uploaded the photo . . . . .	23
3.2	Tactode 2 after open Simulator . . . . .	24
3.3	Code of a polygon by using tangible pieces . . . . .	25
3.4	Code of a polygon by using Blockly Jr. . . . .	26
4.1	Use Case Diagram. . . . .	35
4.2	Sequence Diagram. . . . .	36
4.3	Happy Path. . . . .	37
4.4	Image before and after processing. . . . .	39
4.5	Canvas coordinate system. . . . .	39
4.6	Different positions of Simulator. . . . .	40
4.7	Turtle, scarab and ladybug. . . . .	40
4.8	The robot Tode. . . . .	41
4.9	The different coordinate systems used. . . . .	42
4.10	Tactode program that draws a square (AST in Figure 4.11). . . . .	43
4.11	AST of the example in Figure 4.10 . . . . .	44
4.12	Numbers. . . . .	45
4.13	Letters. . . . .	45

4.14	Numeric Operators. . . . .	46
4.15	Logical Operators. . . . .	46
4.16	Comparison Operators in Tactode. . . . .	47
4.17	Movement blocks in Tactode. . . . .	48
4.18	Stop. . . . .	51
4.19	Controls. . . . .	51
4.20	Variable number 0. . . . .	52
4.21	Create Variables and Set Values. . . . .	52
4.22	Flag. . . . .	53
4.23	Question and Answer. . . . .	54
4.24	Front Sensor. . . . .	54
4.25	Line Color, Way Forward and Line End. . . . .	54
4.26	Pen Down and Pen Up pieces. . . . .	54
4.27	Puzzle and simulation of a pentagon. . . . .	56
4.28	Puzzle and simulation of stopping in front of an obstacle. . . . .	56
4.29	Puzzle and simulation of running away of an obstacle. . . . .	57
4.30	Puzzle and simulation of a line follow. . . . .	57
5.1	Tactode Application Home Page. . . . .	59
5.2	No platform and Source selected. . . . .	60
5.3	Settings Tab. . . . .	60
5.4	Choose Platform, Source and Language. . . . .	61
5.5	Puzzle selected with no errors. . . . .	61
5.6	Downloaded file, after click button share. . . . .	61
5.7	Programs in database. . . . .	62
5.8	New settings. . . . .	62
5.9	Program with errors. . . . .	63
5.10	Correct Program. . . . .	63
5.11	Database with 2 programs. . . . .	64
5.12	Simulator opened. . . . .	64
5.13	After Flag clicked. . . . .	65
5.14	Simulation stopped. . . . .	65
5.15	End of Simulation. . . . .	66

# List of Tables

4.1	Numerical Operators Children . . . . .	46
4.2	Absolute Block Children . . . . .	46
4.3	AND, OR and NOT Block Children . . . . .	47
4.4	Greater (equal) and Less (equal) Blocks Children . . . . .	48
4.5	Different and Equal Block Children . . . . .	48
4.6	Children of Angle/Distance and Speed Blocks . . . . .	49
4.7	If and Else Condition's children . . . . .	52
4.8	While Condition's children . . . . .	52
4.9	Repeat Condition's children . . . . .	52
4.10	Value Block's Children . . . . .	53





# Abbreviations and Symbols

API	Application Programming Interface
App	Application
AST	Abstract Syntax Tree
CPR	Programming and Robotics Clubs
CT	Computational Thinking
SDK	Software Development Kit
IDE	Integrated Development Environment
PWA	Progressive Web App
STEM	Science, Technology, Engineering, and Mathematics
STEAM	Science, Technology, Engineering, Arts and Mathematics
SVG	Scalable Vector Graphics
UAC	Using Arduino in the Classroom



# Chapter 1

## Introduction

### 1.1 Context

Since the birth of the Internet, society has been changing and technology has been evolving [13]. A change in education is necessary, so that it evolves according to the needs of society and industry. According to Education at a Glance: OECD Indicators, in Portugal, [14] Science, Technology, Engineering and Mathematics (STEM) education provides workers with higher salaries and lower unemployment rates. Although one of the most popular broad fields of study among tertiary graduates in Portugal is engineering, students are not usually successful and interested in these subjects in school. The majority of the students considers math one of the hardest subjects in school.

Since early, kids are becoming accustomed to new technologies and increasingly use them, but the truth is that they do not realize what lies behind and what it takes to build it. Technology is often given as a well-stock with which people do not even think about how it came up.

The tactode effort is that children begin to realize, in primary schools, how technology works and what is the logical reasoning of an engineer in the construction of their projects, also motivating students to create, learn more and perhaps to follow their studies in these areas.

The proposal is to associate this project with a STEM-based education in study programs of Portuguese primary schools. From 2015 to 2017, emerged the pilot project "Iniciação à Programação" (*Initiation to Programming*), in the 1st cycle, involving more than seventy thousand students. As a result of this project, the initiative "Programação e Robótica" (*Programming and Robotics*) appeared in the academic year 2017/2018, extended to the 2nd and 3rd cycles of basic education, which seeks to identify and contextualize what students are expected to develop, learning to program when creating stories, animations, and games and solving daily challenges through programming and robotics, considering different scenarios of methodology and learning [15].

## 1.2 Motivation and Goals

This dissertation focuses on motivating children, attending elementary school, for engineering and programming, using a tangible programming system allied to a Web App and a Robot, so they can understand how computer science works. Children can give orders to a real robot or to a virtual robot, executing it on the simulator. This App should open in different platforms and be compatible with different robots in order to be as general as possible.

It is also intended to stimulate teamwork and mutual aid, through forming working groups and sharing equipment. In this way, there is the possibility of discussing ideas, all working towards the same objective, thus contributing to personal development, without neglecting the values of sharing and joint work. With the implementation of this project at a young age, children are expected to become more interested in engineering and mathematics. The difference in the number of people of the different genders who choose to study or work in technical and scientific subjects is still quite pronounced [16].

The purpose of this dissertation is to motivate children between the ages of six and ten to areas related to engineering and computer science, using the Tangible Block Programming System and a Simulator. Tactode Block Programming System uses pieces to construct a tangible and visual language that children can understand. Then, they can construct logical reasoning, similar to a puzzle, in order to give instructions to a real robot. In an application they can take a photograph, using a camera of the available device - smartphone, tablet or computer - the pieces are detected using ArUco markers by image processing, detect and report errors. If no errors were found, the application can finally generate the code that can be executed in real robot.

Thus, in this dissertation a Simulator for Tactode System is developed that allows virtual execution of a robot inside the application, having different functionalities with the option to choose which robot the children wants to simulate.

This process requires a connection between the mobile device (mobile phone, tablet or computer), the puzzle and the robot.

## 1.3 Contributions and Publications

The main contributions provided by this dissertation are:

- **Help children to understand technology.** The principal contribution with this work is teaching children at a young age to understand how technology works by learning to program and developing Computational Thinking.
- **Affordable teaching materials.** At this moment, the Tactode pieces are made by EVA material and paper over the EVA with the instruction and the ArUco Tag. This kind of material is cheap and easy to handle. However, it is a bit laborious when is needed to make in big quantity, so this is possible to be changed in the future. The purpose is that kids can be capable of building their pieces, for example, in handicraft classes with the supervision

of a teacher or at home with the help of the parents. In this way, not only computer and technology activities are improved, but also hands-on work.

- **Versatile material.** Tactode is compatible with several robotic and non-robotic platforms like Cozmo, Ozobot, Sphero and Robobo, Scratch and Python in which the user can share the code made by the puzzle. It is also compatible with different software like Android, iOS, macOS, Windows, and browser, making it more generic.
- **Different ways to use depending on user availability.** Tactode can also be used without a robot, so the child can use it without the obligation of buying a robot. The Tactode application has a simulator with a virtual robot that can execute orders made in the puzzle. This makes Tactode even more economic since the only need, besides the pieces, is a smartphone, tablet or computer, with camera access, to run the application.

The image captured by the user is processed, making it easy to visualize, correcting undesired rotations and highlighting the puzzle pieces. During execution, just the principal puzzle pieces are highlighting, according to which part of the puzzle is running. This way the child can see which piece is being executed at the same time the robot is simulating.

- **The Tactode Simulator also introduces some features to make it closer to reality.** For example, Ozobot has front sensors to detect obstacles and has different reactions to those situations. The Simulator has a button to add an obstacle to the simulation and the robot can stop in front of it or run away. Ozobot also has the function of following a line, so Simulator also has the ability of draw a line to be followed. In a real life it is also possible to stop the execution of the robot whenever is necessary by just getting the robot off the ground. In the Simulator, there is a stop button so it is possible to stop the simulation when it is necessary.

The Simulator can also ask questions to the user according to the content of the puzzle and receive an answer. That answer is then available to use during the program simulation.

A conference paper was accepted as a result of this dissertation: *Web Based Robotic Simulator for Tactode Tangible Block Programming System* in ROBOT'2019: Fourth Iberian Robotics Conference, ISEP, Porto, Portugal. The submitted version is attached at the end of this dissertation, in [Appendix B](#).

## 1.4 Previous Publications and Initial State of Project

This section presents some research projects which directly inspired and influenced this dissertation. The two of them talk about Tactode as programming system based on tangible tiles. Using a camera from a smartphone or tablet, a photo is taken and processed resulting in source code that can be sent to several platforms.

- A. Cardoso, A. Sousa, and H. Ferreira. Programming for young children using tangible tiles and camera-enabled handheld devices. In ICERI2018 Proceedings, 11th annual International Conference of Education, Research and Innovation, pages 6389–6394. IATED, 12-14 November, 2018 [17].
- A. Cardoso, A. Sousa, and H. Ferreira. Easy robotics with camera devices and tangible tiles. In ICERI2018 Proceedings, 11th annual International Conference of Education, Research and Innovation, pages 6400–6406. IATED, 12-14 November, 2018 [18].

## 1.5 Dissertation Structure

Besides this Introduction chapter, there are six more chapters. The content of Chapter 2 is the state of the art based on educational programming languages, robots and the impact of STEM education in young age children. In Chapter 3, a problem to solve is detected and the different solutions to achieve the Tactode goal are discussed. After choosing a solution, in Section 3.5 the different technologies and languages used to develop Tactode Application are presented. The Simulator, how the robot came about, the requirements for a good functioning, how each piece of the puzzle is implemented and the principal challenges that can be used in Tactode are explained in the Chapter 4. Chapter 5 is about the experience that a child can have when navigating through Tactode App. Finally, Chapter 6 talks about some conclusions and future work for this project.

## Chapter 2

# Fundamentals and Literature Review

Young people are starting being called as "digital natives" because they were born in the digital age, with the internet, computers and smartphones always available. They are really comfortable texting messages, playing online games, surf on social networks, etc. But can they understand technology? Are they capable of creating their own games, animations [19]? It is time to make children think about technology.

The involvement of a robot in learning from a very young age can be beneficial for the growth and development of children's skills in several aspects. The concept of 21st-century skills is associated with the need to respond to the demands of today's society and the future, where problem-solving, decision-making, teamwork, ethical sense, project management and use of digital technologies are considered core competencies [15].

In this chapter, will be discussed how these values can be achieved in schools, using different techniques, such as block-based, tangible or graphical programming languages. And what are the benefits of using this techniques with robotics to catch the attention of children and improve characteristics like teamwork, better logical reasoning, gender equality and future interest in STEM areas, such as engineering and computer science.

### 2.1 Block Programming

Educational Programming languages should be easy to get started, in order to begin teaching young children. They also need to create opportunities to develop increasingly complex projects so children with differences interests do not get bored [19].

Some of the best software companies already developed educational programming language like Blockly [20, 21, 22], Figure 2.1, created by Google, and MakeCode [23], Figure 2.2, created by Microsoft.

Nowadays, Block-based languages are probably very influential in educational languages. As the name implies, they contain a set of predefined blocks of code, which the user typically drags and drops to form a program.

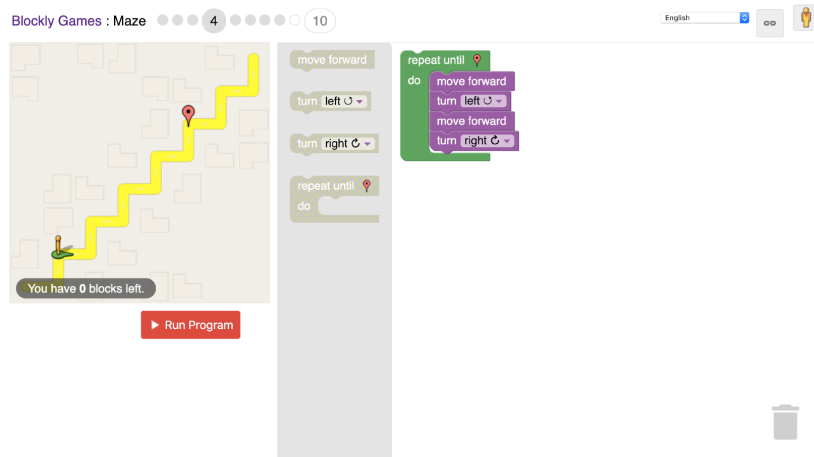


Figure 2.1: Blockly Games by Google.

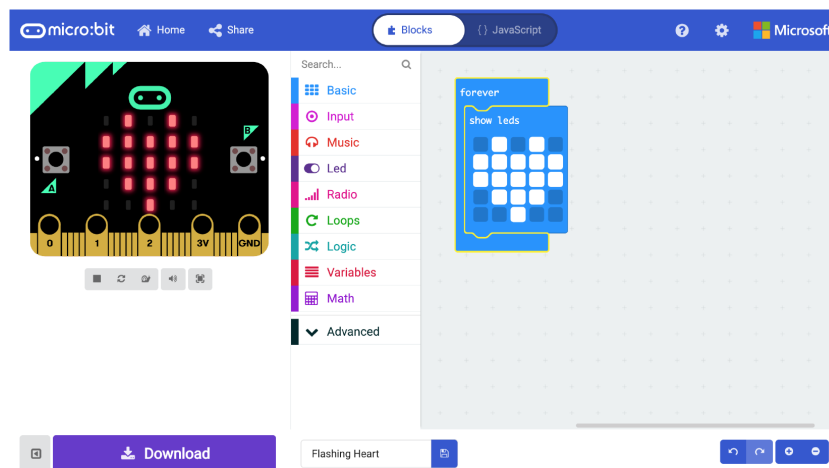


Figure 2.2: MakeCode by Microsoft for micro:bit.

The concept of block-based programming started around two decades ago, initially introduced by the MIT Media Lab following the ideas of Seymour Papert on the need to enable students to approach abstract concepts in a way that is meaningful for them. They created Scratch [19, 24], Figure 2.3, one of the best-known example of a block-based language.

This kind of programming language allows students to create digital media such as simple games and animations without typing a single line of code [25]. In this way, it makes computer science accessible to young students, expressing their creativity in different aspects.

Block-based programming environments use a puzzle pieces metaphor. Each block provides visual cues to the user on how and where the block can be used through the block's shape, color (which is associated with categories of similar blocks), and the use of a natural language label on the block, to convey its meaning.



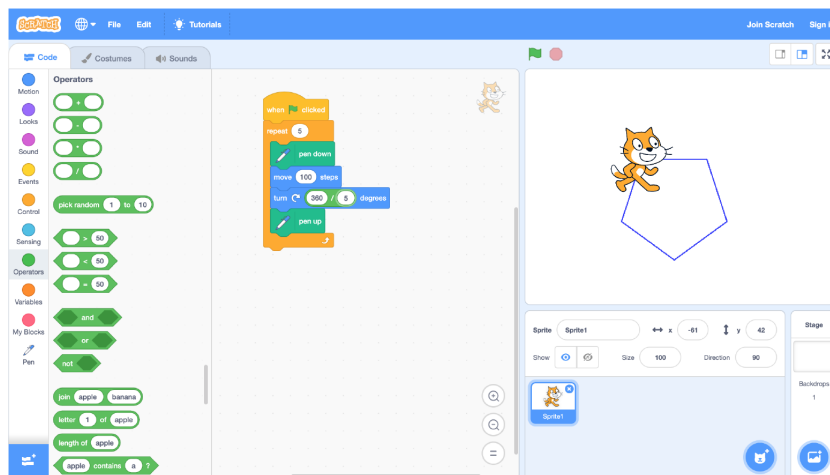


Figure 2.3: Scratch by MIT.

There are some characteristics that make these languages easy to use. According to the Wein-trop's study [26], students describe these languages as easy to read, the labels look less like a text editor and closer to English than text-based languages. They recognize the block programming similar to Java but in an English form, easier to understand. Students consider that the visual nature of the blocks and the shapes can help to see where blocks could be used and how the puzzle could be built. Another advantage is the ease to compose a program thanks to the act of drag and drop commands and being less error-prone.

## 2.2 Tangible Programming

A tangible programming language is similar to a block-based or visual programming language but instead of using a computer screen, physical objects are used to represent various programming elements, commands, and flow-of-control structures. A tangible programming language is a block-based language whose blocks can be physically grabbed and arranged by the programmer [6].

In 2015, Sapounidis et al. [27] made a study establishing many advantages of using tangible interfaces to teach programming, specially for young children. They completed the tasks faster, with higher complexity, fewer errors and more frequent teamwork. They also thought the tangible interface more attractive and easier to use, comparatively with the graphic ones. However, the material for this kind of programming language can be expensive and limited compared to block-based language. There are not many languages of this type and many are not commercially available yet.

Some examples of tangible languages are AlgoBlock [1], Figure 2.4, Electronic [2], Figure 2.5, Cubelets [3, 28, 29], Figure 2.6, the physical robot version of the Fisher Price Think and Learn Code-a-Pillar [30], Figure 2.7, TagTile [5], Figure 2.8, Quetzal and Tern [6], Figure 2.9, T-Maze [31, 7], Figure 2.10, the Osmo Coding Family [32], Figure 2.11, and CodeBits [9], Figure 2.12.



Figure 2.4: AlgoBlock [1].



Figure 2.5: Electronic blocks [2].



Figure 2.6: Cubelets [3].



Figure 2.7: Code-a-Pillar [4].



Figure 2.8: TagTile [5].



Figure 2.9: Quetzal and Tern [6].

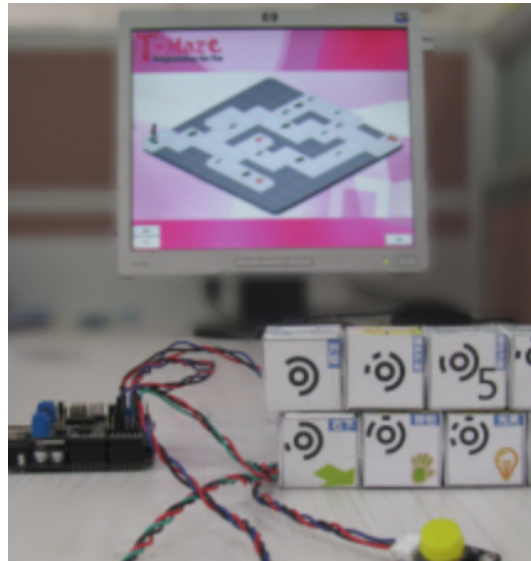


Figure 2.10: T-Maze [7].



Figure 2.11: Osmo Coding Family [8].

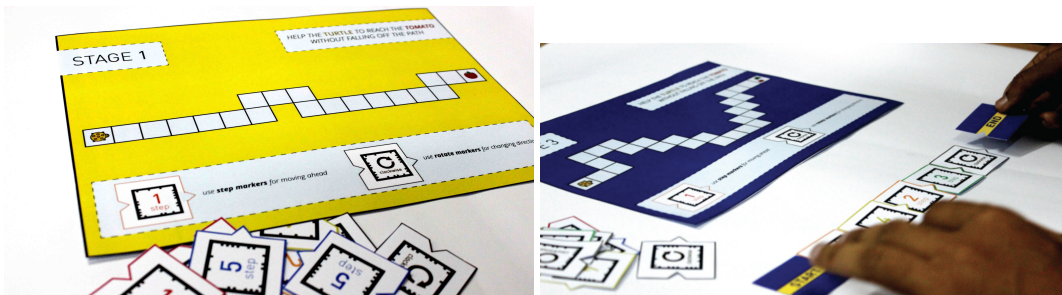


Figure 2.12: Code Bits [9].

## 2.3 Three.js

Three.js can be useful for building robot simulations, quite similar to Blockly, MakeCode and Scratch, mentioned in Section 2.1.

Three.js is a high-level JavaScript library and API used to create and display animated 3D graphics in a web browser [33]. Three.js uses the WebGL API for connecting the browser to graphics card, providing a lot more graphical processing power than is traditionally available on a website. Three.js generally uses it for displaying 3D graphics, but it is equally good at 2D graphics.

To use WebGL, a device and a browser that supports it are necessary. In these days, every modern smartphone, tablet, PC or laptop has a graphics card capable of running at least a basic scene [34]. Thus, the use of WebGL allows complex animations to be created without having to use plugins.

The Web API provides some functions built-in to the browser, used in Three.js applications. The Document Object Model (DOM) describes the way in which an HTML document is modelled as a JavaScript object, transforming a simple HTML document into nested JavaScript objects. The top-level object in the DOM is called window. Every object in JavaScript running in a web browser is attached to the window object.

Since the Three.js script is included, it creates a keyword THREE that can be used to access all the features of Three.js and it is also attached to the window object. Next level down is the document object and it represents the actual HTML document that is loaded. This is useful for example, for getting window dimensions - *window.innerWidth*, *window.innerHeight* - and accessing HTML elements - *document.body* refers to *<body>* element in HTML document.

Every Three.js app has the following basic components:

- Create Scene that holds all the objects by calling the Scene constructor to create a scene instance.

---

```
1  const scene = new THREE.Scene();
```

---

- Create Camera used to view the scene and determine how the object will be presented to the viewer. In this case, the constructor PerspectiveCamera was used.

---

```
1  const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
```

---

- Create Mesh by creating geometry and material and add it to the scene. Geometry defines the shape of our Mesh and Material defines the way that the surface of the Mesh looks.

---

```
1  const geometry = new THREE.BoxBufferGeometry(2, 2, 2);
2  const material = new THREE.MeshBasicMaterial();
3  const mesh = new THREE.Mesh(geometry, material);
4  scene.add(mesh);
```

---

- Create the renderer that takes the Camera and the Scene as inputs and renders onto the `<canvas>`. Once the page is drawn, it is necessary to fit onto the device's physical screen.

---

```

1  const renderer = new THREE.WebGLRenderer();
2  renderer.setSize(container.clientWidth, container.clientHeight);
3  renderer.setPixelRatio(window.devicePixelRatio);

```

---

- Add canvas `<canvas>` element to the page. Appends the canvas as a child of the scene container.

---

```

1  container.appendChild(render.domElement);

```

---

- Rendering the scene. Telling the renderer to take a still picture of the scene using the camera and output that picture into the canvas element.

---

```

1  renderer.render(scene, camera);

```

---

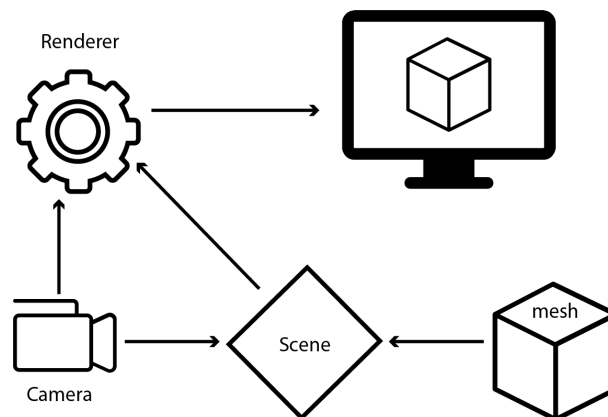


Figure 2.13: Creating a Three.js Animation.

A very important method in Three.js is `window.requestAnimationFrame`, because, when used in a periodically called way, it allows to set up an animation loop that will draw the scene every time the screen is refreshed (typically 60 times per second).

This is visible only when adding movement to the object in each frame, for example, rotating the object

---

```

1  mesh.rotation.x += 0.01;
2  mesh.rotation.y += 0.01;

```

---

Or as shown in the section before, Listing Code 3.1, with the function `moveForward()` where the robot position is incremented.

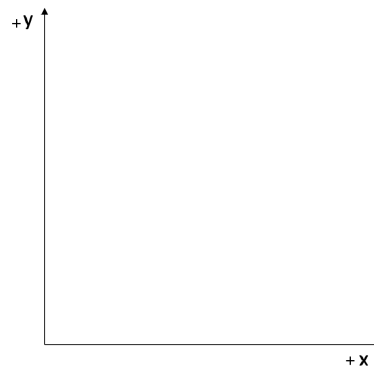


Figure 2.14: Three.js coordinate systems.

These movements are made according to the Cartesian Axis defined by the Three.js, shown in figure 2.14.

Three.js has different ways of creating an object, depending on the shape and geometry asked. In the given example, a cube is created using *BoxBufferGeometry()*, but it is possible to create a cylinder with *CylinderBufferGeometry()*, a sphere with *SphereBufferGeometry()*, etc. It is also possible to join different objects to create a group of objects. Or load external objects using loaders such as *GLTFLoader()* for binary format, *OBJLoader()* for .obj file format, *ImageLoader()* to load texture images and *SVGLoader()* to load .svg file format.

In Tactode, the loader used was *SVGLoader()*. SVG tags were used to create the robot graphics. SVG tags are used for visual shapes such as circles, rectangles, polygons, allowing embedded `<SVG>`. So, it permits to design the object externally and export it to SVG file and use it in the web. But it is not so simple when trying to do more complex objects. SVG elements have their own DOM Element.

## 2.4 Programming Robotics in Elementary Schools

Despite the growing evolution of technology, it is difficult for students to choose classes related to programming because of its complexity. For teachers not trained in this area, it also becomes complicated to teach. In addition, there is also the budgetary difficulty of schools.

Paulo Freire, in his book "Pedagogy of Oppressed" [13], explains that currently teaching expects teachers to be simple narrators, who talk about reality as if it were static and predictable. And so, students need to memorize this narrated content, were thus obliged to receive facts, memorize them and repeat them.

Robotics in primary schools improves logic rationalization in an exciting way, students learn to think deeply about technology and how it works in a fun way [13]. They are also encouraged to share equipment, work as a team and grow values such as teamwork. For these reasons, it is important to have a change in education [13].

### 2.4.1 Programming in Portuguese Elementary Schools

The transformation and innovative updating of society's technical activities should be reflected in the content of school education, and education should focus on the formation of knowledge, skills, and competencies, allowing the younger generation to be successfully integrated into modern sociotechnical systems, to maintain and develop efficiently the scientific and technological potential of society [35].

Portugal has some projects in order to involve technology in education. Starting from the beginning, since 2010 Portugal has the Scratch Day, a competition to create with Scratch, through the educational project EduScratch [36].

In the academic year 2014/2015, a National Network of Programming and Robotics Clubs (CPR) was launched and achieved a significant projection in Portuguese schools. In January 2019 an inquiry was carried out, concluding that exists 296 clubs all over the country [37].

Another technological education program is Apps for Good, launched in Portugal as a pilot project in January 2015. With an origin in London 2010, this program challenges students and teachers to develop applications for smartphones or tablets, showing them the potential of technology in transforming the world and the communities in which they operate. With a design methodology, students have the opportunity to experience the product development cycle. This initiative count with 6698 students, 571 teachers, and 232 schools [38].

From 2015 to 2017, the pilot project "Iniciação à Programação" (*Initiation to Programming*) in Portugal started in the 1st cycle, involving more than seventy thousand students. As a result of this project, the "Programação e Robótica" (*Programming and Robotics*) appears, an initiative in Basic Education - Probótica - appeared in 2017/2018, extended to the 2nd and 3rd cycles of basic education. It seeks to identify and contextualize what students are expected to develop, learning to program when creating stories, animations, and games and solving daily challenges through programming and robotics, considering different scenarios of methodology and learning [15].

A pilot project was planned for the 2018/2019 school year - UAC - Using Arduino in the Classroom. The project consists of the development of Arduino Learning Kits, to learn STEAM subjects, of disciplinary or interdisciplinary scope. The development of these KITs is carried out in an environment of continuous training of teachers, on the pedagogical applications of Arduino technologies [39].

### 2.4.2 Teachers Perception

The world and its economy are changing, but education has maintained almost the same system since the middle of the nineteenth century. Some specialists state that if teachers from nineteenth-century time travel to actual schools, they would have no problem to teach the students. However, the acceleration of the change in society is requiring a new set of skills, intellectual activities, and ways of thinking to be a successful citizen. Accordingly to [13], Grabinger and Dunlap observed teachers in conventional classrooms and concluded that they used examples and problems decontextualized leading to an inadequate understanding of acquired knowledge.



Robotics is multidisciplinary and therefore integrates very well into the STEM education system [40]. There are studies showing a STEM-based education is more efficient if it is started in childhood as soon as possible, making it easier for students to understand technology, reducing the existing stereotypes in the area of engineering [41]. In the study carried out by Khanlari [40], eleven teachers participated, revealing that robotics helps in learning related to science and technology. However, some teachers do not have the expertise to teach such knowledge to their students and therefore they are not sure whether robotics would be beneficial or not.

It is very difficult for teachers to include robotics in the regular curriculum because of the heavy focus on standardized testing and pressure to accomplish academic standards. Robotics technology has been more accessible not only for experts but also for teachers and students. However, educators need to consider innovation as a necessary focus of student learning of twenty-first-century, requiring new learning environments to promote this innovation. Accordingly to [13], some pioneer teachers like Bratzel and Kee addressed the standard education with robotics activities, helping others to bring robotics into the classroom.

### 2.4.3 Logical Reasoning and Computation thinking

In a world that is becoming complex, it is more important than ever for the young generation to be equipped with knowledge and skills to solve hard problems. If education does not change, students are learning concepts that they can not make connection between their learning and their life and they are not acquiring essential skills for effective thinking and reasoning [13].

Robotics facilitates learning in many subjects such as mathematics, physics, electronics, engineering, computing. It can be used to solve problems related to numbers such as proportions, positive and negative numbers, square roots, algebraic equations, trigonometry, geometry, and estimation. In addition to problems with numbers, robotics helps to understand concepts related to programming, logic, speed, direction, torque, acceleration, loops, sensors [40].

Besides, robotics can be very helpful in languages and arts, reading closely to determine what the text says explicitly and to make logical inferences from it, interpret words and phrases as they are used in a text.

Therefore, students develop Computational Thinking [13], a set of thinking skills, habits and approaches, that are essential to solve problems using a computer [7].

Accordingly to [13], the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) consider that CT should be integral part of education for school-age children because of its characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data.
- Representing data through abstractions such as models and simulations.
- Automating solutions through algorithmic thinking.

- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
- Generalizing and transferring this problem-solving process to a wide variety of problems.

In addition, children develop some skills like:

- Confidence in dealing with complexity.
- Persistence in working with difficult problems.
- Tolerance for ambiguity
- The ability to deal with open-ended problems.
- The ability to communicate and work with others to achieve a common goal or solution.

#### **2.4.4 Increased Interest in Computer Science**

Robotics in education is considered as a way to train engineering thinking in children, developing their interest in technical creativity, focusing on opting for the engineering professions [35].

A study demonstrates that showing young people that they can make a difference in the world through the use of engineering and technology will encourage them to the areas of mathematics, science, and programming [42]. Students learn new concepts better when they know it will be useful not only to complete their class projects [40] but also to society.

#### **2.4.5 Team Work**

When working on making robotics activities, it is recommended that students work in small groups. In this way, they obtain the skills needed for effective collaboration. They become excited and motivated when sharing ideas, learning to cooperate by decision-making, improving criticism and communication skills [13].

Formation of teams consists of members with different technical capabilities, cognitive styles and prior experiences, which implies juxtaposing multiple perspectives that children need to deal with and overcoming characteristics like egocentrism [43]. Team work also helps students with low self esteem by discussing with other students with different skills.

#### **2.4.6 Gender inequality**

Both genders should be encouraged to learn programming and robotics. From the 1980s there was a decline of female programmers, which is regrettable because they have skills that sometimes the opposite sex does not have, such as attention to detail, being meticulous and the ability to find bugs in a program [13].

Happiness and well-being have been shown to positively influence learning, however, female students tend to underestimate their abilities and eventually lose interest by taking a pragmatic approach to programming. Although males show more confidence in the programming area, they are less structured and organized. The male greater interest is often related to the games industry [13].

There are many stereotypes about people who work in engineering, technology, and science that innocently a child picks up, one of them being: a scientist usually is a Caucasian male, with his hair disheveled, wearing glasses and white coats [44].

The threat of stereotype may have negative implications for how girls doubt their abilities when evaluated in the areas of mathematics, physics, and programming, showing low performances, even if they are academically talented. Women are most interested in people and are more motivated by the opportunity to help others. This is a problem, given that professions such as engineering and computer science are not seen as professions of that caliber.

However, all these stereotypes have been changing and disappearing, and women who have chosen these areas for their future have begun to make way for other women.

A four-year study was conducted evaluating the performance of a women's course at a school where STEM-based learning was introduced. The results indicated that the scores of these students improved in the areas of science and mathematics and that more of them chose to do engineering in the future. In this study, STEM projects were found to improve the world and help the others [16].

## 2.5 Examples already tested

There are some products in the market that involve block and tangible programming and also a robot. In this section, will be presented some of these products that use applications or web browser to execute code, block-based or tangible, in some cases into real robots and, in other cases into simulated robots.

### 2.5.1 Lego Mindstorm EV3

Lego Mindstorm EV3, represented in Figure 2.15, is a set of programmable robotics construction for ages greater than 10 years. The aim is to build the own robot with Lego pieces, program and command it. Seventeen different robots can be created from humanoid robot to a shooting scorpion or a walking dinosaur. The EV3 set includes bricks, motors, and sensors to build the robot and make it walk, talk, move. It also comes with the necessary software and app that allow building, program and control the robot from computer, tablet or smartphone. There is an On-Brick Programming App where the robot can easily be programmed, with basic tasks. The communication can be done by Bluetooth, Wi-Fi or USB cable [10].

### 2.5.2 Ozobot

Ozobot, represented in Figure 2.16, is a robot and can be coded online with OzoBlockly Application or screen-free with Color Codes. The purpose is to inspire young minds to go from consuming

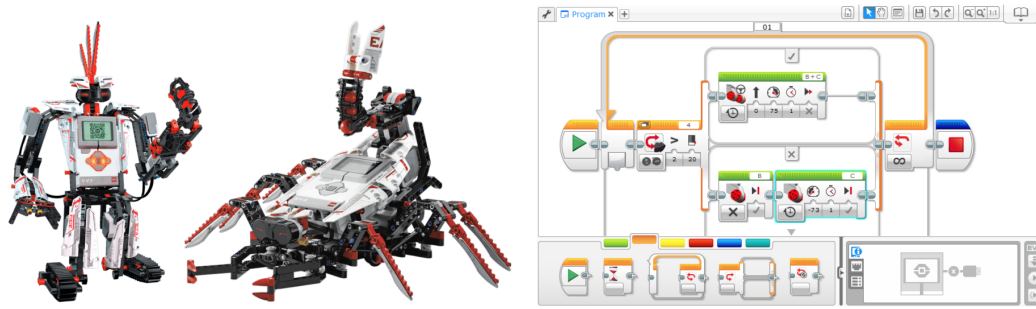


Figure 2.15: The Lego Mindstorms EV3 robot and its programming interface [10, 11].

technology to creating it [12]. OzoBlockly can be used with an application or in a Web browser. The application can be used in iOS or Android tablet, working with the Evo Robot. The browser is also compatible with Evo Robot, while used with a computer, or compatible with Bit Robot for a computer or tablet [45].

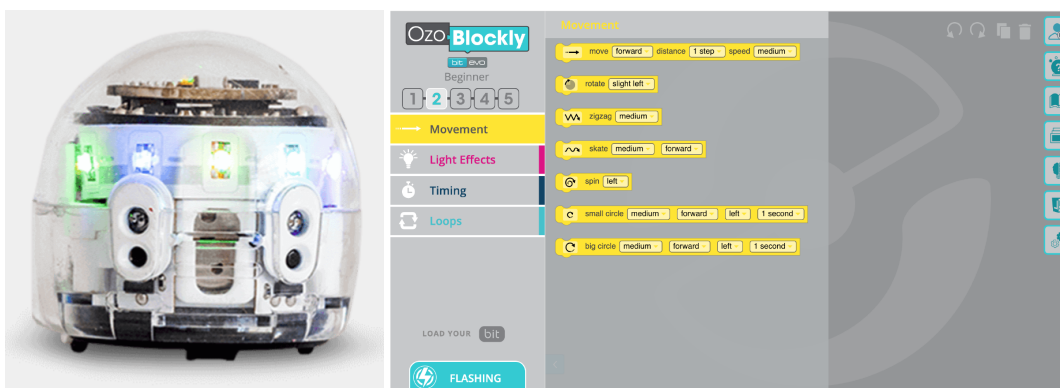


Figure 2.16: The Ozobot robot and OzoBlockly Editor [12].

### 2.5.3 Open Roberta Lab

The Open Roberta Lab, represented in Figure 2.17, is a free platform that makes programming easy to learn, from the first steps, to program robots with multiple sensors and capabilities. It can be used at any time without installation in any devices, computer or tablet, with an Internet browser. Thanks to the programming language NEPO (graphic programming language developed at Fraunhofer IAIS), simple programs can be created like puzzle pieces [46]. It is available in 14 languages and supports hardware like Lego Mindstorms EV3, Lego Mindstorms NXT and micro:bit.

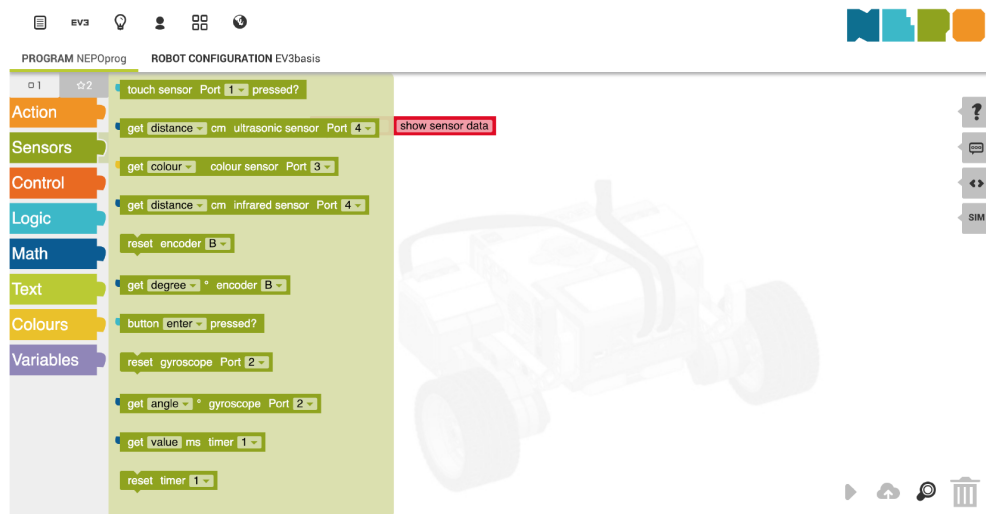


Figure 2.17: Open Roberta Lab.

### 2.5.4 Blockly Games

Blockly Games, represented in Figure 2.18, is a free Google project with a series of educational games that teach programming. There are different games, with different levels, so children who have not had prior experience are ready to use conventional text-based languages by the end of these games [22].

The different games, with different difficulties, are [47]:

- Puzzle is a quick introduction to Blockly's shapes;
- Maze is an introduction to loops and conditionals;
- Bird is a deep-dive into conditionals;
- Movie is an introduction to mathematical equations;
- Music is an introduction to functions;
- Pond Tutor introduces text-based programming;
- Pond is an open-ended contest to program the smartest duck.

### 2.5.5 Scratch

Scratch, represented in Figure 2.19, is made for children, between six and eight years old, to learn how to code and implement strategies for solving problems, designing projects, and communicating ideas [48].

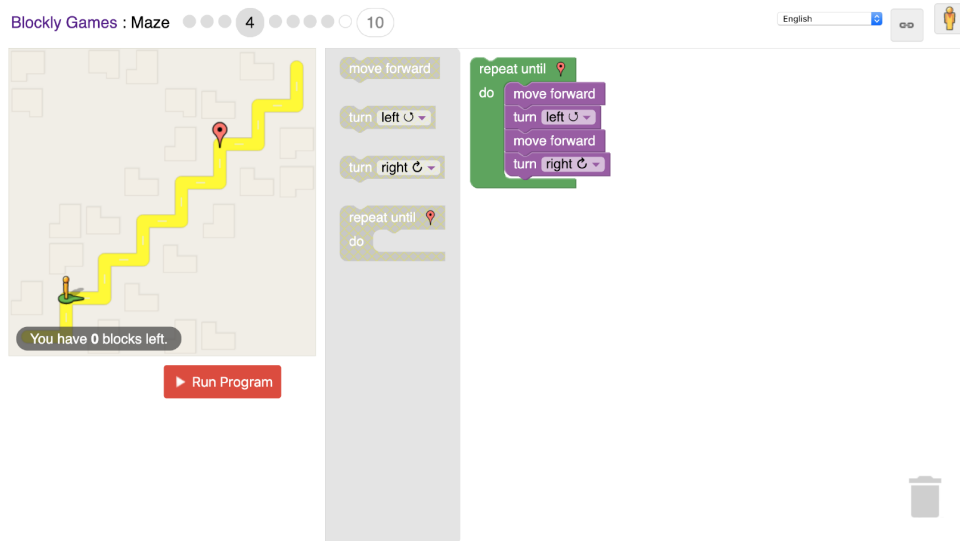


Figure 2.18: Blockly Games.

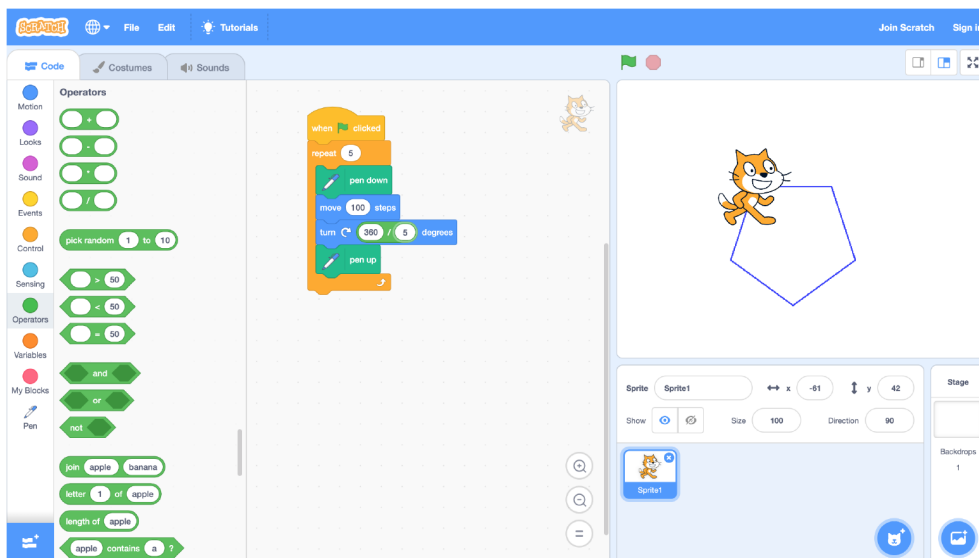


Figure 2.19: Scratch.

In Scratch programming, the activity is mixing graphics, animations, photos, music, and sound. Scratch has two design criteria: diversity, supporting different types of projects like stories, games, animations, simulations, so people are all able to work on projects they care about; and personalization, people can personalize their Scratch projects by importing photos and music clips, recording voices, and creating graphics [19]. In the end, the projects can be shared with other users.

## **2.6 Conclusion**

Taking into account what was said in this chapter, programming at a young age is really important to the learning and growth in schools and allows children to have a better Computational Thinking. After the study of some educational games in this area, it is now possible to set some goals for this dissertation and try to solve some flaws by creating a robotics-related educational programming language that is both intuitive and fun for children. Using that language for the construction of tangible code, that can be executed in a robot, but also implement a simulator to see a simulation of the robot. Therefore, in the next chapter, some proposals will be discussed, creating a more useful and efficient project.





## Chapter 3

# Problem Statement and Technological Selection

The purpose of this dissertation is to have the possibility of creating a tangible puzzle and choose between executing it virtually in the application, using a Simulator of the real robot, or a real execution, using a real robot. For this, the application must be able to read the puzzle, recognize the pieces used and compile this code into a code that various robots and platforms can execute, including a simulator. This chapter is about the different options to develop this.

At the start of this work, the Tactode Programming System<sup>1</sup> has 2 different applications, with different functionalities. The aim is to have those different functionalities in one application. The first (Tactode 1) uses tangible blocks to build a puzzle, in order to be executed by a real robot, Figure 3.1. By taking a photograph, the application recognizes the pieces in the puzzle and transpile the puzzle code into executable code for a robot. The second (Tactode 2) uses virtual blocks to build the program and simulate robot actions online, Figure 3.2.

The main advantages or disadvantages of each one need to be defined and also decide in which way they can be useful for this project.

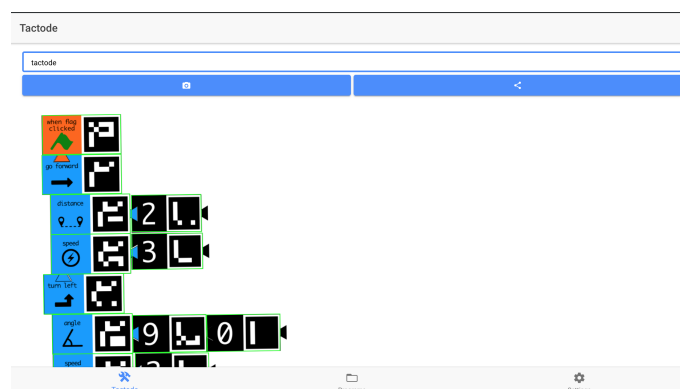


Figure 3.1: Tactode 1 after uploaded the photo

<sup>1</sup>Tactode project webpage: [fe.up.pt/asousa/tactode](http://fe.up.pt/asousa/tactode)

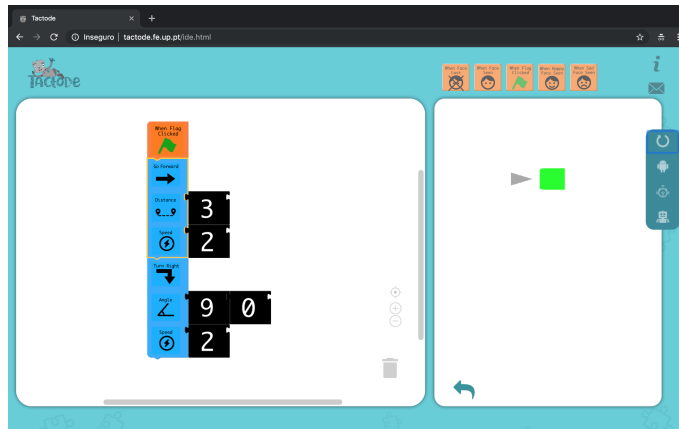


Figure 3.2: Tactode 2 after open Simulator

### 3.1 Tactode 1

The Tactode 1 application was developed in a dissertation project. It uses tangible pieces that are assembled like a puzzle. Each piece is composed by a title that summarizes the command statement, an illustrative image of the command, and an ArUco tag, which is used by the application to identify the pieces, an example is presented in Figure 3.3.

The ArUco tags represent ids, that the software can associate with programming language commands. This is possible using a camera, taking a picture and then use a library that can detect, organize and turn the ArUco markers into executable code, such as ArUco JavaScript library. Marker detection and identification has the following steps: Image segmentation, contour extraction and filtering, marker code extraction, sub-pixel corner refinement [49].

After the construction of the puzzle, the user can open the application and, using the camera of the device, capture a photograph. The App recognizes the ArUco pieces and then compiles the tangible program into executable code that the chosen platform can perceive. Currently, there are four compatible robots - Cozmo, Ozobot, Sphero and Robobo - and Scratch and Python as non-robotic platforms.

The application consists of a text box to write the project name and a button to capture a photo of the code. After capturing the photo, the application compiles the code and, if there are no errors, the user can export the code to the platform or save it in the cloud. The application also has a settings page that allows the user to choose between taking a photo or loading an existing file and also choose the platform or robot to use.

Its execution time depends on the implementation of each robot, for example, in the case of the robot Cozmo, the code can be sent directly from the application of Tactode to the application of Cozmo.

The app is compatible with Android, iOS, macOS, Windows and the browser. However, it becomes complicated to use the application on small sized screens.

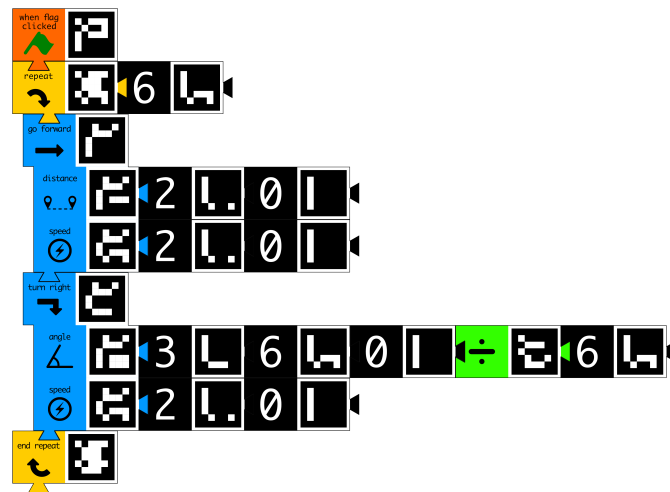


Figure 3.3: Code of a polygon by using tangible pieces

## 3.2 Tactode 2

Tactode 2 is an application developed in a computing class. It was intended to be a Progressive Web App (PWA), an application that runs in a browser, without internet connection, being accessible anywhere in any device.

PWAs are apps that are [50]:

- Progressive - increase their functionality, conform to the capabilities of the device on which they run, becoming more and more powerful as they are used.
- Web - are built using HTML, CSS, JavaScript and a new generation of JavaScript APIs.
- App - have all the most remarkable Apps features. Can be installed on mobile devices, have apps look and feel, push notifications, etc.

However, in Tactode 2 internet connection is needed when uploading a photograph. So, Tactode 2 runs in a browser with access to the internet. In its content there is a graphic editor, with access to the blocks for programming, where the puzzle can be assembled and later simulated. Kids have several options to create their puzzles, put them on the IDE and see their simulation. The user can create a puzzle in one of these two ways:

- Tactode is prepared to create the puzzle directly in the browser.
- However, in order to reduce screen time, there is the possibility of making the puzzle with tangible pieces.

And do the upload of the puzzle in these ways:

- Create the puzzle directly in the web and simulate it.

- Take a picture of the actual puzzle using the camera of the device where the application is running.
- Upload a previously taken picture by searching the image on the device.
- Load an XML file of a previously saved puzzle.

In all these options it is always possible to visualize the puzzle in the IDE. The ArUco pieces are not visible, as it is possible to see in Figure 3.4, which makes it less confusing for the children. It is also possible to highlight the piece that is being executed, so it can be seen when some error occurs.

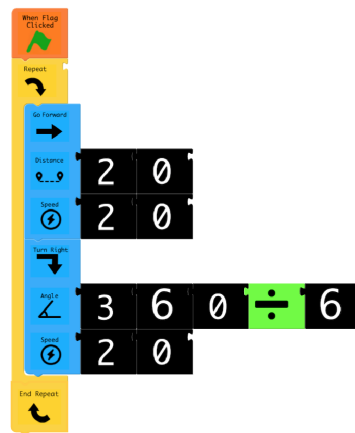


Figure 3.4: Code of a polygon by using Blockly Jr.

### 3.3 Problem Definition

Programming by tangible blocks brings a lot of advantages for children's learning. However, older children prefer graphical interface than a tangible interface. It is also proven that it is more appealing and funny for students to see in the real robot what was built by them, as it is applied to a concrete example, making the tasks more concrete.

In order to please all the public, it would be interesting to have an application that does not occupy memory in a device and that is as portable as possible to be able to use anywhere, accessing just through a tablet or smartphone, without being dependent of extra material. However, it is believed that learning is more productive when there is a purpose and it is applied to a real case, so to be able to not only simulate the robot but actually test it.

What if a merger of the two was created? It would appear a lot of advantages but also disadvantages. In case of Tactode 1, a real good advantage is the fact that it is compatible with many robots and platforms but, deploying an App available to all platforms - iOS, Android, macOS, Windows - it has a cost. For example: Apple App Store requires a payment of 99\$ per year [51];

in Google play is one-time payment of 25\$ [52]; in Microsoft is also a one-time payment of 15\$ for individuals and 99\$ for companies and organizations[53].

Besides this, if a user wants to have the application in other devices, he needs to install the App in each, and of course, this occupies memory.

Some of these problems can be solved with the Tactode 2, but some new disadvantages come. Tactode 2 just needs a browser, so it is compatible with all devices and does not occupy memory. A PWA only needs an Internet connection to open it and then it is supposed to not need any connection to the internet. However, the Tactode 2 App requires connection to the internet when the photograph is taken and sent to the server and then back to the IDE. Which does not make Tactode 2 a real PWA.

Also, the blocks used, created by Blockly Jr, are not the best ones for this work. It is possible to see in Figures 3.3 and 3.4 that the blocks of Blockly Jr, used in Tactode 2, are not similar to the tangibles ones, used in Tactode 1, and it does not have the format of a puzzle when all the pieces are connected, for example, in the use of conditions and loops. This causes that the students do not have to think so much for them because this already is done by the app and the aim is for children to learn the concepts and logic of programming. The solution here would be to create something similar to Blockly, to be as closest as possible to the tangible ones with a Drag and Drop interface.

Another problem related to Tactode 2 is that it does not have a compiler, so a correct code can not be sent to the robot. If the code has errors, the robot can react differently from what it was supposed.

On the other hand, Tactode 1 has a compiler that translate the tangible code into multiple high level languages, as many as the used platforms.

Considering the focus of this dissertation in teaching programming, there are also disadvantages in Tactode 2. The features implemented in its robot simulator are reduced. Only the basic robotic movements, like move forward or backward and turn right or left, were implemented. In order to teach programming and robotics concepts to the children, sensors, variables, flow control elements, such as while statements and if and else statements, need to be implemented.

So, the aim is to implement, in the simulator, all blocks implemented in reality, so that children can take advantage most of the knowledge of robotics and programming.

### **3.4 Proposed Solution**

The proposal is to create a Tactode 3 that can have a Simulator where can be seen the pieces of the puzzle that are being executed. It is intended to be compatible with several robotic and non-robotic platforms and the software to be the most generic as possible to any user can use at any moment and anywhere, using any device he has available. At the same time, children can't get stuck in something virtual, so it is important to program with tangible pieces.

The Proposed solution is to implement in Tactode 1 what is made in the Tactode 2 in order to create Tactode 3. So, the first goal of this dissertation is to implement a Simulator, with a virtual robot, in the Tactode 1, so that children can simulate their codes before sending it to the

real robot. The Simulator needs to implement, not only the basic movements of moving forward, backward, turn left and right, rotations, but also make use of the sensors of each robot, motion sensors, and line followers. The second goal is to highlight, in the App, which piece is being executed so children can easily make a connection from what they are seeing on the robot and what piece corresponds to the movement. It is also important to turn the App more appealing for young children in order to encourage use.

After proposing a solution, it is time to start thinking about implementation. The next section talks about some technologies needed to develop the Tactode Application and its Simulator.

## 3.5 Development Technologies

The Tactode application aims to simulate the real robot in the application. In order to be accessible for all users, it should be compatible with all platforms [54]. Nowadays, there are several options to create multi-platform applications. Ionic Framework was used. It generates applications for multiple systems from a single source code, using Angular and TypeScript [55]. Besides, Three.js was used to develop the Simulator.

### 3.5.1 Web Programming

The Web has become the main platform for the information society [56].

Web programming is the act of writing and coding involved in web development. It includes web content, client and server system and network security. It requires interdisciplinary knowledge on application area, database technology and client and server scripting.

The most common languages used in Web Programming are HTML, PHP, CSS, JavaScript and Perl 5. HTML is used to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages [57].

JavaScript was created at Netscape in the early days of the web and is a trademark licensed from Sun Microsystems (now Oracle).

It derives its syntax from Java its first-class functions from Scheme, and its prototype based inheritance from Self. It is high-level, dynamic, well-suited to object-oriented and functional programming styles. This makes it a powerful, flexible and fast programming language now used for increasingly complex web development and beyond [57, 58].

### 3.5.2 TypeScript

TypeScript has the same syntax and semantics of JavaScript. It is possible to use and incorporate JavaScript code into TypeScript. It runs on any browser that supports ECMAScript 3 (or newer) and includes JavaScript features like async functions and decorators. Types are optional but allow the developer to use practices as code refactoring and define interfaces when developing JavaScript applications [59].

As JavaScript, TypeScript is synchronous, so it is necessary to make it asynchronous when doing the Simulator since an action must finish before the next one is executed.

An asynchronous operation allows the computer to move on to other tasks while waiting for the asynchronous operation to complete. To create this, it is necessary to implement async functions and Promises.

### 3.5.2.1 Promises

Promises are objects that represent the eventual outcome of an asynchronous operation. A Promise object can be in one of three states:

- Pending: It is the initial state, the operation has not completed yet.
- Fulfilled: The operation has completed successfully and the promise now has a resolved value.
- Rejected: The operation has failed and the promise has a reason for the failure.

A promise is settled if it is no longer pending, it is either fulfilled or rejected.

Let's take an example where the user wants to move the robot, in order to draw a square. The robot needs to move forward some distance, then turn 90° and repeat this three more times. If promises are not used, the robot moves forward and turns 90° at the same time.

The promises in the Tactode Simulator were implemented using Async-Await. Async write functions that handle asynchronous operations and Await is an operator that returns the resolved value of a promise. So, Await halts (or pauses) the execution of an Async function until a given promise is resolved [60].

The code in Listing 3.1 has parts of the functions used to move forward and turn left. To move forward, the promise only returns when the move distance achieves the distance given by the user, which means that the robot reached the end of the distance established, resolving the promise of the function *simulateForward()*, and now it can turn left. The same happens when turning left, the angle is increasing until the angle established, then the promise of the function *simulateLeft()* is resolved.

---

#### Listing 3.1 Promises in Move Forward and Turn Left

---

```
1  simulateForward(forward: ForwardBlock): Promise<null> {
2      return new Promise<null>( resolve => {
3          this.forward = forward;
4          this.resolveForward = resolve;
5      })
6  }

8  simulateLeft(left: TurnLeftBlock): Promise<null> {
9      return new Promise<null>( resolve => {
10         this.turnLeft = left;
11         this.resolveLeft = resolve;
12     })
```

```

13 }

15 moveForward() {
16     var dist = this.forward.children[0].children[0].parsedNumber;
17     var speed = this.forward.children[1].children[0].parsedNumber;

19     this.robot.position.x += this.time * Math.cos(this.angle + Math.PI) *
        Math.abs(this.speedRight - this.speedLeft);
20     this.robot.position.y += this.time * Math.sin(this.angle + Math.PI) *
        Math.abs(this.speedRight - this.speedLeft);

22     this.currentDistance += this.time * Math.abs(this.speedRight -
        this.speedLeft);

24     if(this.currentDistance >= dist) {
25         this.resolveForward();
26     }
27 }

29 moveLeft() {
30     var angle = this.turnLeft.children[0].children[0].parsedNumber;
31     var speed = this.turnLeft.children[1].children[0].parsedNumber;

33     if (this.currentAngle + this.time * speed >= this.degToRad(angle)) {
34         this.turn(this.degToRad(angle) - this.currentAngle);
35         this.currentAngle = this.degToRad(angle);
36     } else {
37         this.turn(this.time * speed);
38         this.currentAngle += this.time * speed;
39     }

41     if(this.currentAngle >= this.degToRad(angle)) {
42         this.resolveLeft();
43     }
44 }

```

---

### 3.5.3 Ionic Framework

Ionic is an open-source Software Development Kit (SDK) for building hybrid mobile and desktop apps, using web technologies like HTML, CSS and Javascript. It is Cross-platform, with one code base, which means that work across multiple platforms, being compatible with a variety of mobile devices like iOS, Android, macOS, Windows and the web as a PWA. It is a Node Package Manager (npm) module and requires Node.js.

It also uses Cordova to have access to host operating systems features such as Camera, GPS, Flashlight, etc. It includes mobile components, typography, interactive paradigms, and an extensible base theme [61]. In Tactode, the principal Cordova feature used is the Camera.

Although it is possible to choose several interface frameworks such as Angular, React, Preact or Vue.js, Angular is the most commonly used. Core components have been written to work as



a standalone Web Component library, but the `@ionic/angular` package simplifies integration with the Angular ecosystem. So, Tactode is build on top of Angular [55].

### 3.5.4 Three.js

Three.js is the JavaScript library used to create the Tactode Simulator. This library allows to create and display animated 3D graphics in a browser. Its main operation was explained in Section 2.3 and basically it consists in a loop that it is always running, allowing animations (in the case, robot movements) at 60 frames per second. The number of frames per second is set by the developer (it is not predefined by Three.js).

## 3.6 Conclusion

After studying Three.js and how a scene is rendered, simple animations can be created. With knowledge in promises, more complex animations can be done, waiting for one operation to complete to start another one. In Chapter 4 it is possible to see in detail how this is implemented in the Tactode pieces and how it works. It is also shown how the Tactode simulated robot is created and some other preliminary stuff for better visualization and understanding.



## Chapter 4

# Tactode Application

This Chapter is about the development of the application, with focus on the subject of this dissertation - the Simulator. First it is possible to see the Requirements and the way user can interact with the application and, in particular, with the Simulator.

Some preliminary points are discussed such as: the rotation of the image that contains the program, to allow better visualization; the image processing and highlighting of the main pieces; the division of the screen in two parts during the simulation, allowing to visualize the simulation of the robot while seeing the blocks of code that are to be executed and the design of the virtual robot and its movements.

Then the implementation of the Simulator in Three.js and the internal structure of the puzzle are explained, detailing the implemented blocks and giving examples of some challenges that users can overcome.

### 4.1 Requirements

In this section, the Non-Functional and Functional Requirements will be discussed, designed for the proper functioning of this application and in particular the Simulator, taking into account pleasant navigation and the needs of the users.

#### 4.1.1 Non-Functional Requirements

**Performance** is a concern. A user does not want to spend most of the time waiting for something to load. The Tactode Simulator needs to be fast and execute tasks quickly. Besides the simulation, Tactode needs to process the image taken, and sometimes there are some large programs that require more processing. So, it is somewhat acceptable that the user has to wait a few seconds for this process.

On the other hand, **security** is not a big worry since sensitive and personal data are not used. It just requires camera access and image files and only stores images with no program errors.

It must be **intuitive**. Users should understand how to work with the application without the need for an instruction manual. In this way, the Tactode Application also must ensure **reliability**,

since the young age of users can lead them to confusion if something does not run as expected or an error occurs. It should have visual elements that guide the user of the application, giving feedback about what is happening.

And last but not least, the user wants to use the application at any device, in this way, Tactode App should be responsive to any device and screen dimensions.

#### 4.1.2 Functional Requirements

To better describe the functional requirements for this project, a set of Use Cases will be enumerated, followed by an explanation and a Use Case Diagram.

A group of basic Use Cases is needed for the simple operation of Tactode Application, without thinking about the Simulator. The very first requirement is the need to build a Program with the tangible puzzle pieces, open the application, name the program, open the camera and take a picture of it or upload an existing one, see the highlighted code and the errors if any. If no error exists, the program can be exported and/or shared to the target platform. The user can also see, delete or export previous programs. This is shortly stated in Use Case (UC) 1.

**UC 1.** *The user can Take/Upload a photograph of the program, wait until image processing and export or share to the target platform.*

The UC 2 follows the previous Use Case. The user can change the settings of the application and choose different target platforms between Ozobot, Cozmo, Sphero, Robobo, Scratch, and Python, select a photograph already taken, saved in the device, or take a new one at the moment. The user can also pick a comfortable language for him.

**UC 2.** *The user wants to configure the application, so he can select image source, target platform and also an adequate idiom.*

After the image processed, with no errors and set to the desired platform, the simulation can finally be executed. The user wants to see the Simulator and the robot. He can click on the Flag button, used to start simulation, (if required by the chosen target platform) and start the simulation of the created program. This is stated in UC 3.

**UC 3.** *The user wants to open Simulator (click the Flag button, depending on the target platform) and see the robot starting the simulation.*

Besides, the user wants to see what blocks are being executed during the simulation (UC 4). To do that, when the robot is executing one task of the program, that piece of the puzzle is highlighted and when moving to the next task, the corresponding one is highlighted and the first one is no longer evidenced.

**UC 4.** *The user wants to see what blocks are being executed during the simulation.*

If the program created has a Forever in the code, or, for some reason, the user needs to stop the simulation, a stop button is required so that the simulation stops. This is UC 5.

**UC 5.** *The user wants to click the Stop button and interrupt the simulation when he wants.*

In order to take advantage of all functionalities of all platforms, depending on what the user built in the puzzle, the robot can draw a line, run away from obstacles and follow a line. So, the user can use the puzzle piece *PenDown* to draw the robot path. This is very useful when constructing, for example, a polygon, in this way, the movement of the robot is more visual (UC 6). When the robot sensors are turned on, the robot can detect an obstacle, so the users can click on obstacle button and move the obstacle wherever they want. The robot can stop in front of the obstacle or run away from it (UC 7). Another sensor used in the Simulator is the follow line sensor. When used in the puzzle, the user can add a line to the Simulator and the robot can detect and follow that line (UC 8).

**UC 6.** *The user wants to see the robot's path so that he can confirm visually its movement.*

**UC 7.** *The user wants to add an obstacle to the Simulator and see the robot's reactions to it.*

**UC 8.** *The user wants to add a line to the Simulator and see the robot following that line.*

In Figure 4.1 it is possible to see the Use Case diagram summarizing the use cases listed above.

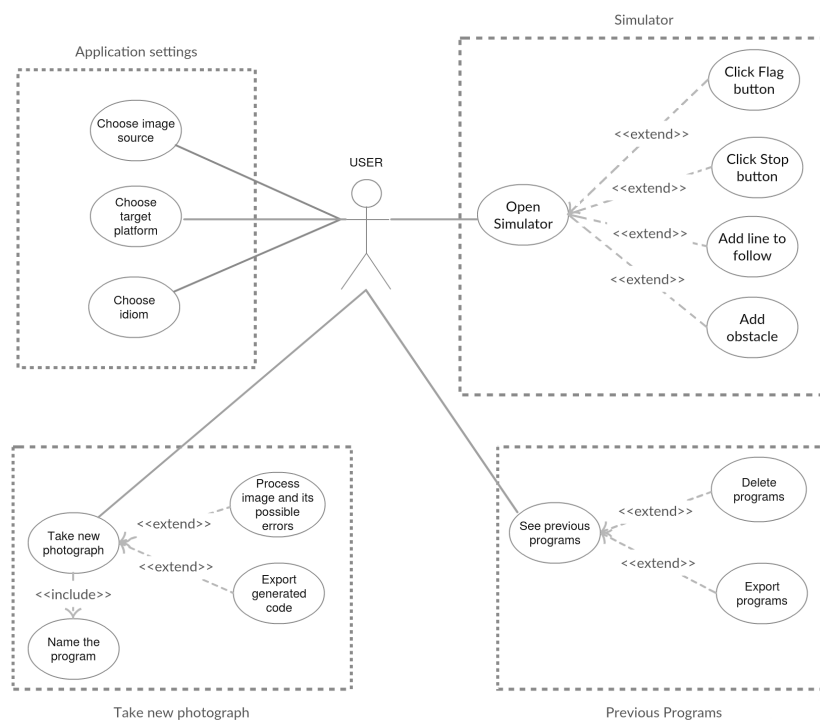


Figure 4.1: Use Case Diagram.

#### 4.1.2.1 Sequence Diagram

The following sequence diagram is based on the architecture of the Simulator. According to what was explained in Section 3.5.2.1 about JavaScript Promises, the user creates the puzzle and uploads

it to the Tactode application. The Simulator can only be opened if the puzzle has no errors. After the Simulator reads the first instruction and executes it, the next instruction is waiting until the first is finished. So, each Block in the Simulator that has visual simulation (like Move Forward, Backward, Turn Left, Right, Question and Answer, etc) has a promise that only is resolved when that action is finished. When there is no instruction to simulate or when the user clicks the Stop button, the simulation stops. After this, the user can upload a new program.

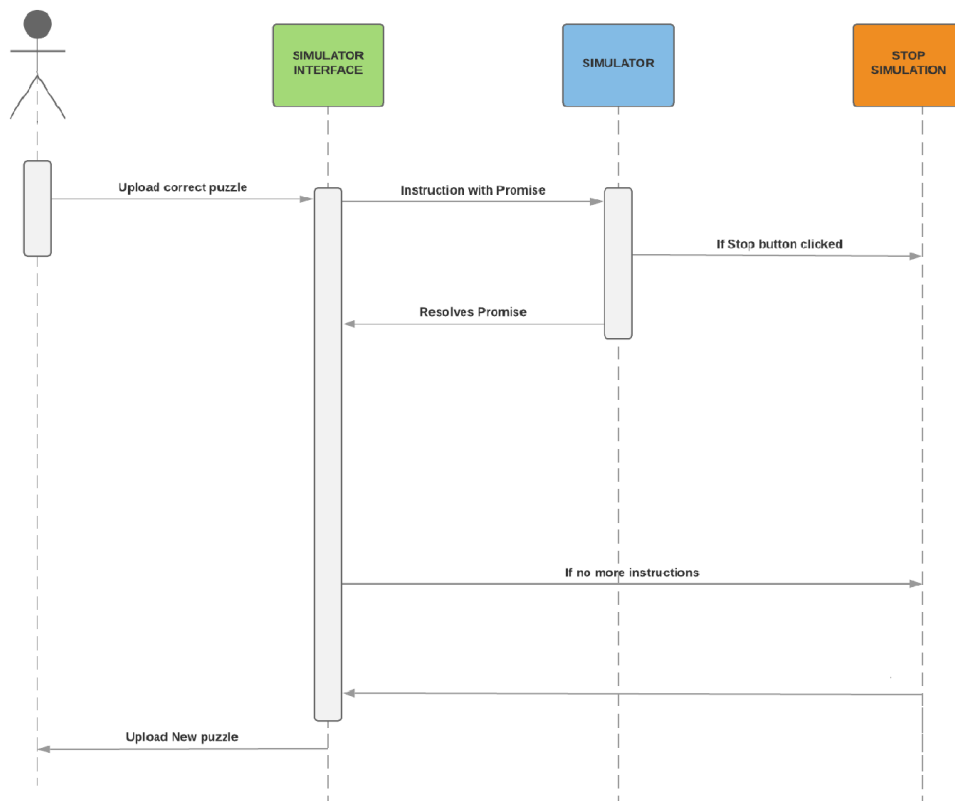


Figure 4.2: Sequence Diagram.

#### 4.1.2.2 Happy Path

Happy Path is a default scenario with no exceptional or error conditions, where the input is known and the output is expected.

In Tactode, the users can build a puzzle with the tangible tiles, open the application, go to the Settings Tab and select the desired Target Platform, the source of the image and the language they are most comfortable with. After that, a previously taken photograph of the program can be uploaded or a new one can be taken at the moment. The user can also go to the Previous Programs Tab and see or export previous programs. Because a happy path considers that there are no errors and the program created has correct syntax, the user can also export the actual program. If the

intention of the users is only to export programs, they can finish the action now. If not, the user can continue by opening the Simulator and starting the simulation of the puzzle. When there are no tasks to execute, the simulation stops and the action is finished or the users can stop the simulation whenever they want and the action also arrives at the end. This path is visually represented in Figure 4.3, where the green circle is the beginning and the red is the end.

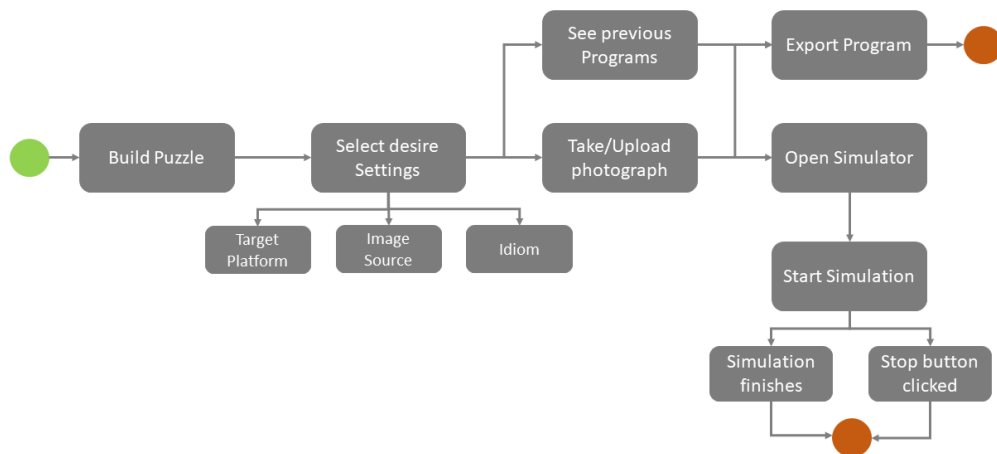


Figure 4.3: Happy Path.

## 4.2 Design of Interface

This section is about the primary tasks to do before starting to think about the Simulator. Although this dissertation focuses on creating a Simulator for a robot, some adjustments were needed to be made before starting this work. The first problem to solve was placing the image in the correct orientation and position on the screen, to be easier for children to see the puzzle, and highlight the pieces so they can understand what was done. All of this, working with different coordinate systems and mathematical equations. The second task was dividing the screen into two responsive sides so that the Simulator could be placed on the right side. And then, design a robot to use in the simulation.

### 4.2.1 Rotating image

The first problem was rotating the image to the correct orientation, for a better visualization of the pieces. If the child, after constructing the puzzle, takes the picture that is not aligned with the puzzle, the result will be a rotated image.

In Figure 4.4, it is possible to see a picture taken with no image processing and another already in the Tactode IDE after processing. The rotation of the image was done through canvas library

functions. The main function is `context.rotate(angle)`. This function rotates the image but the rotation center point is the canvas origin which is not the center of the image. This has generated a problem, where the image becomes displaced from the canvas. In order to solve it, it was necessary to calculate the correct position of the image and draw it there.

The solution was a translation to the upper left corner of the canvas. This corner was calculated by the point of the most left marker and the most top marker, multiplied by the ratio. The ArUco code recognizes the position and corners of all puzzle pieces (markers). After that, using a loop cycle, it is possible to find the left top corner  $(x, y) = (leftX, topY)$ .

However, one more problem appeared. Some images exceed the width limits available by the canvas area and some markers disappeared because of that. The solution was quite similar to the first one. It was also necessary to find the bottom right corner  $(x, y) = (rightX, bottomY)$ , limiting width and height according to the left top corner and bottom right corner.

---

#### Listing 4.1 Width and height according to left top and bottom right corners

---

```
1 width = rightX - leftX;
2 height = bottomY - topY
```

---

The ratio is calculated by the input width of the real device screen and the width calculated before. In the end, the image is placed where it should be. After all this calculated, it is possible to draw the image, using the canvas function.

---

#### Listing 4.2 Function used to draw the image

---

```
1 context.drawImage(image, dx, dy, dWidth, dHeight)
```

---

Where  $dx$  and  $dy$  are the destination point  $(x, y)$  where the image should be placed.

According to the canvas coordinate system shown in Figure 4.5, and since coordinate system is placed on the center of canvas,  $leftX$  and  $topY$  should be negative points.

---

#### Listing 4.3 drawImage applied to Tactode context

---

```
1 context.drawImage(image, -leftX * ratio, -topY * ratio, width * ratio, height *
   ratio);
```

---





Figure 4.4: Image before and after processing.

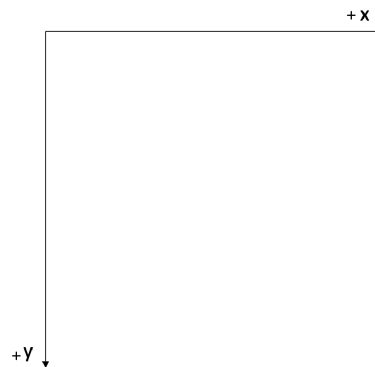


Figure 4.5: Canvas coordinate system.

### 4.2.2 Split Screen

The best way to place the Simulator is next to the puzzle, once the main tiles of the puzzle are highlighting during the execution of the robot in the Simulator, because it is more intuitive for children to understand what is happening and also know when something went wrong with their reasoning. However, it is not always possible to split the screen in two horizontally and clearly see both sides. Because of this, when the height is greater than the width, instead of dividing the screen in two side by side halves, the Simulator is placed below the puzzle. In other words, the goal is to make the Tactode application responsive to any device with any dimensions. Figure 4.6 shows Tactode Application with the Simulator on the right of the puzzle and also with the Simulator below the puzzle.

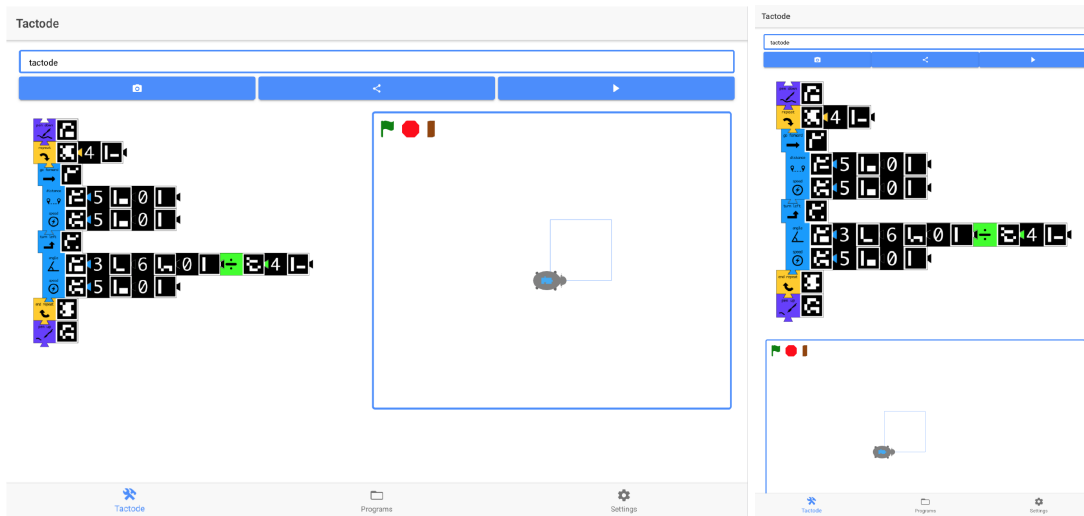


Figure 4.6: Different positions of Simulator.

The Simulator only appears when the button play is clicked. This positioning was achieved by using properties of HTML5 and CSS. In the HTML code of the main page, defined the workspace as a d-flex row and divide that row into two columns. The first one for the puzzle and the second one for the Simulator. Then, in CSS, using media queries, change the width of the canvas left and right according to the device size and its orientation.

### 4.2.3 Creating the robot

Since the purpose is to simulate a scene of the execution of a robot, it is necessary to create one. When people see a real robot moving, normally it is on the ground and the person is looking from above to the robot. Given this and after researching about simulations scene and robots, it was concluded that the more intuitive way to represent the Simulator is a 2D scene which is viewed from the top.

In this case, the robots compatible with Tactode, when viewed from above, are mostly circular and, because users need to know the orientation of the robot, the idea was to create a robot with spherical structure but with something that indicates in what direction it is moving. There are many animals in nature with these characteristics like a turtle, a scarab or a ladybug, Figure 4.7.

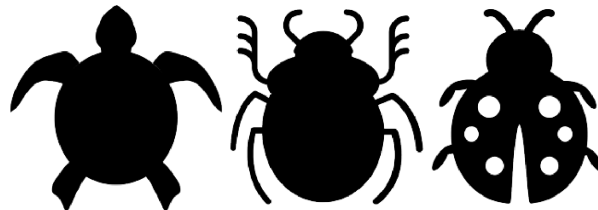


Figure 4.7: Turtle, scarab and ladybug.

The idea was to create something with the same meaning but related to robotics and Tactode as a tangible block programming system. With more straight lines and hinges, typical color of a robot and something related to puzzles.

The format used to create and implement the robot in Three.js was SVG and essentially it is composed by: a large ellipse to draw the robot body and 4 small ellipses for the legs; two straight line segments grouped to draw paws; a rectangle and a semicircle, drawn by a path, to build the head; a three-sided polygon (triangle) plus a small rectangle and a small ellipse for the ears. The SVG Tags implemented were:

---

**Listing 4.4** SVG Tags used to build Tode.

---

```
1 <svg></svg> <!-- Open and close SVG file Tag -->
2 <ellipse/> <!-- Ellipse Tag -->
3 <path/> <!-- Path Tag -->
4 <rect/> <!-- Rectangle Tag -->
5 <polygon/> <!-- Polygon Tag -->
6 <g></g> <!-- Group Tag -->
```

---

This is how Tode, represented in Figure 4.8, is born. Tode, the simulated robot for Tactode application.

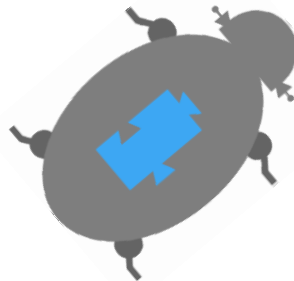


Figure 4.8: The robot Tode.

#### 4.2.4 Robot Movements

The robot also has axes. This is important to understand the movements direction. When constructing the robot movements, it was considered a robot with two wheels.

Figure 4.9 shows the two different coordinate systems. One defined by Three.js and the other is the coordinated system of the robot based on its center. The movement equations are defined by:

$$x_{r_t} = x_{r_{t-1}} + \cos(\theta_t) \times v \times \Delta t_t$$

$$y_{r_t} = y_{r_{t-1}} + \sin(\theta_t) \times v \times \Delta t_t$$

$$\theta_{r_t} = \theta_{r_{t-1}} + \omega \times \Delta t_t$$

Where:

- $v$  is linear velocity.
- $\omega$  is angular velocity.

$$v = \frac{v_1 + v_2}{2}$$

$$\omega = \frac{v_1 - v_2}{d_r}$$

And:

- $v_1$  and  $v_2$  are wheel speed of the robot.
- $d_r$  robot's diameter.

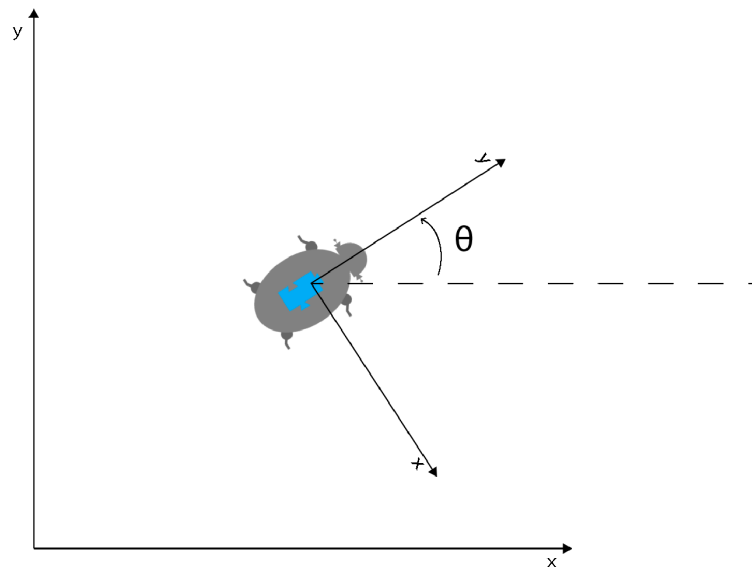


Figure 4.9: The different coordinate systems used.

### 4.3 Simulator

This section is about the design and implementation of the Simulator. In order to better understand how the Simulator is structured, it is necessary to know more about the architecture of the application. Then it will be explained which blocks of the Tactode tangible block programming

system were implemented in the Simulator and the challenges created with those blocks in order to children solve them.

### 4.3.1 Abstract Syntax Tree

The Tactode programming language has many pieces (defined as blocks), but not all are compatible with all platforms and robots. The tables shown in Appendix A. show all the pieces Tactode has and their compatibility with the platforms and robots available. Figure 4.10 shows an example of a square - a loop, running four times, that repeats move forward, with a fixed distance, and turn right/left 90°. The corresponding AST is shown in Figure 4.11.

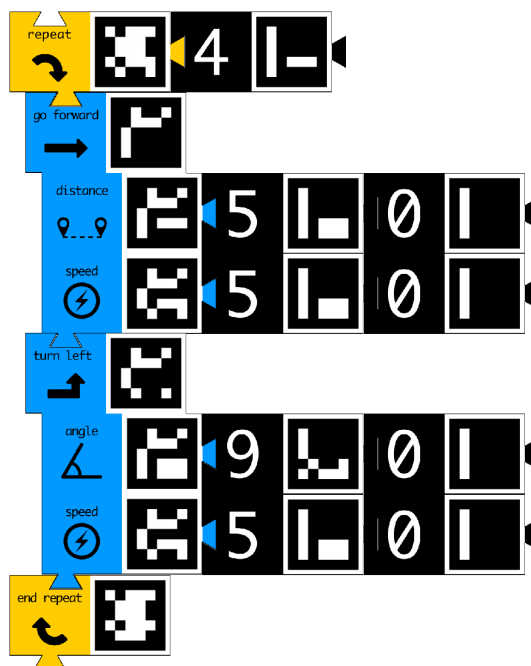


Figure 4.10: Tactode program that draws a square (AST in Figure 4.11).

Every uploaded puzzle is processed as an Abstract Syntax Tree (AST). Each original piece has a corresponding Block and each of them has an array of other Block objects (children) and also a parent Block object. In this way, each object knows its parent and its children. However, there are some extra elements in the AST that do not have a piece in Tactode, such as:

- **RootBlock:** special Block with no parent that serves as the root of the AST. Its children are the command blocks that are not inside of any control flow block, which means that have no indentation in the tangible language.
- **BodyBlock:** child of control flow blocks RepeatBlock, ForeverBlock, IfBlock, ElseBlock or WhileBlock.
- **ConditionBlock:** child of RepeatBlock, IfBlock or WhileBlock and, as the name suggests, it contains the condition to be verified by these control flow blocks.

In case of Figure 4.11, Root Block has one child - Repeat Block - and this block has three more children:

- Condition Block that usually has as child the number of repeats of the cycle.
- Body Block where the main instructions are introduced. It has two children: Forward Block and Turn Left Block. Forward Block has two more children - Distance and Speed - each with Number block as children. Turn Left Block is similar but instead of distance, it has an Angle Block.
- End Repeat Block only ends the repeat, as the name indicates.

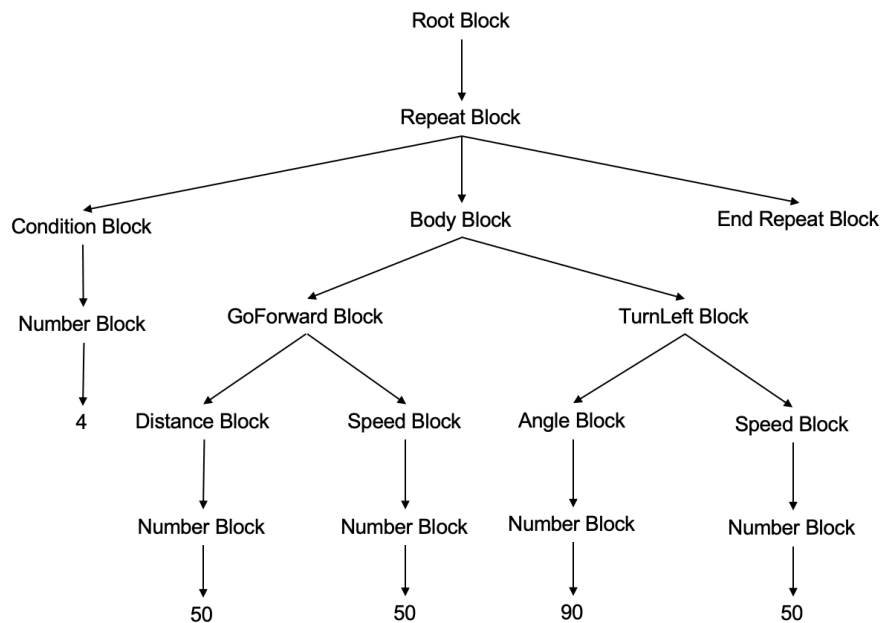


Figure 4.11: AST of the example in Figure 4.10

Now that it is known how to read the puzzle program, it is time to implement the blocks in the Simulator. As mentioned in Section 3.5.2.1, asynchronous functions and promises were used to make one action wait for another.

## 4.3.2 Implemented Blocks

This section is organized by type of pieces and will show the blocks used in the Tactode Simulator, describing how they are implemented and their compatibility with the different robots and platforms.

### 4.3.2.1 Numbers and Letters

The AST concatenates digits as numbers into NumberBlock and letters as words into LetterBlock. Tactode has a distinct piece for each digit and each letter and uses those pieces to form numbers

and strings, respectively. The pieces used for the numbers are represented in Figure 4.12 and for the letters in Figure 4.13. The AST has a single NumberBlock object for each sequence of numbers in the tangible language, independently of the number of digits, where the sequence of concatenated digits is stored. The same for strings, instead of storing individual letters, the AST has a single LetterBlock object where the string of concatenated letters is stored.

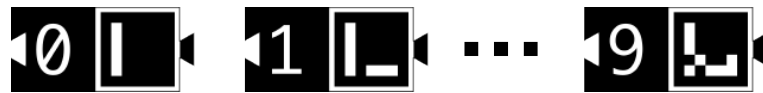


Figure 4.12: Numbers.

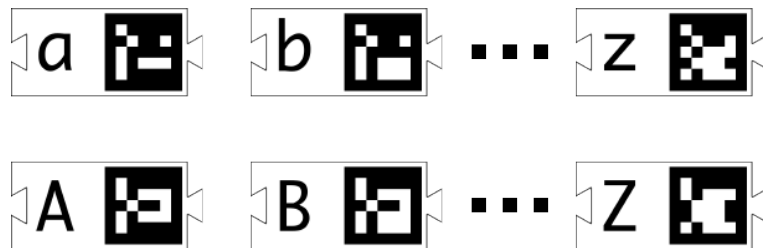


Figure 4.13: Letters.

#### 4.3.2.2 Operators

There are three different types of results for operators: Numerical, Logical and Comparison. Operators such as addition, subtraction, division, multiplication, remainder and absolute return a number and operators like and, different, equal, greater, greater equal, less, less equal, not and or return true or false if the condition is or not verified.

However, the children of the operations are not always numbers or expressions true/false. Numerical Operators children are usually numbers. Greater (equal) and Less (equal) Blocks have the same children of Numerical Operators. The children of operators && (AND), || (OR) and ! (NOT) are Boolean expressions and Different and Equal Block can have both numerical and Boolean children. For better understanding, let's divide operators in three groups.

##### Numerical Operators

This group is about operators that return numbers as result. These operators are represented in Figure 4.14.

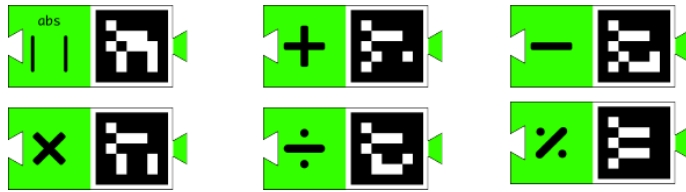


Figure 4.14: Numeric Operators.

Every operation (absolute, addition, subtraction, multiplication, division or remainder) has its own function, returning the result of that operation. In this way, when an operation occurs, the Simulator will verify what is the type of operation, using a switch and call the function of that operation, returning its result.

However, operators' children are not always numbers. They can be, for example, the content of a variable or an answer. Operations as addition, subtraction, division, multiplication and remainder have 2 children but absolute only has one. Tables 4.1 and 4.2 show the children that each operator can have.

Child 0	Child 1
Answer	Answer
Answer	Number
Number	Number
Number	Variable
Number	Operation
Number	Answer
Operation	Number
Operation	Operation
Variable	Number
Variable	Variable

Child
Answer
Number
Operation
Variable

Table 4.2: Absolute Block Children

Table 4.1: Numerical Operators Children

### Logical Operators

These operators are represented in Figure 4.15.



Figure 4.15: Logical Operators.



This group is about operators used to compare Boolean expressions and also returns a Boolean value. Logical Operators are defined in the Simulator the same way of numerical operators, but the returned value of each function is of type Boolean.

It returns true if the condition, represented by the operator, is verified or false if it is not.

The Table 4.3 shows the children that each operator can have.

Children 0	Children 1
Boolean Operator	Boolean Operator

Table 4.3: AND, OR and NOT Block Children

### Comparison Operators

These operators are represented in Figure 4.16.

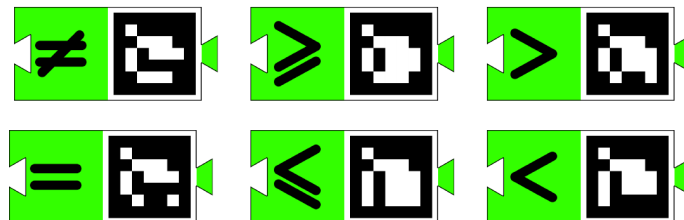


Figure 4.16: Comparison Operators in Tactode.

This group is about operators used to compare numerical and also Boolean expressions. The result of these operations is a Boolean value. Comparison Operators are defined in the Simulator the same way of Logical Operators. It returns true if the condition is verified or false if it is not. A simple example is checking if 4 is greater than three. If it is verified, return true, if not, return false.

Tables 4.4, 4.5 show the children that each operator can have, where Boolean Operator is an operation where the result is of type Boolean. Some of them are represented below in Comparison Operators.

Child 0	Child 1
Answer	Answer
Answer	Number
Number	Number
Number	Variable
Number	Operation
Number	Answer
Operation	Number
Operation	Operation
Variable	Number
Variable	Variable

Child 0	Child 1
Answer	Answer
Answer	Number
Number	Number
Number	Variable
Number	Operation
Number	Answer
Operation	Number
Operation	Operation
Variable	Number
Variable	Variable
Boolean Operator	Boolean Operator

Table 4.4: Greater (equal) and Less (equal) Blocks Children

Table 4.5: Different and Equal Block Children

### 4.3.2.3 Movements

After defining numbers, letters and operations, it is time to create movements. The movements in the Tactode Simulator are: move forward or backward and turn left or right. It is necessary to be acquainted with robotics concepts to create movements in the Simulator (Section 4.2.4). Figure 4.17 display the movement Blocks. Three.js allow movement from objects by increasing (or decreasing) its position x (or y) and rotation.

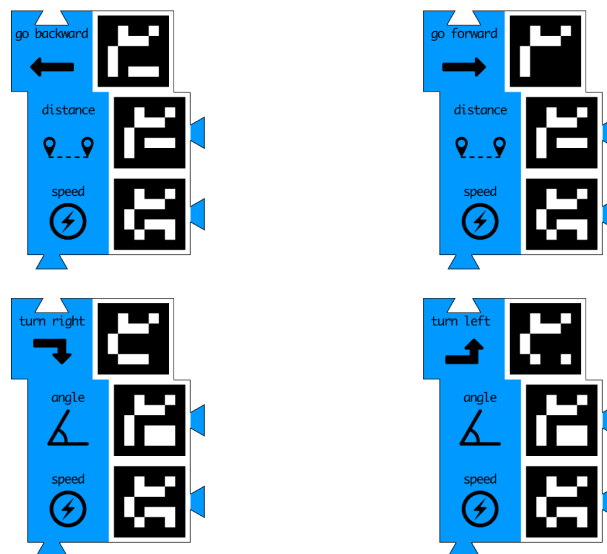


Figure 4.17: Movement blocks in Tactode.

The Go Forward and Go Backward Blocks has as children the distance and speed and Turn Left and Right has as children the angle and speed. However, each child can be defined with other Blocks such as Operations, variables, etc. This is defined in the next Table.

Angle/Distance	Speed
Answer	Answer
Number	Number
Operation	Operation
Variable	Variable

Table 4.6: Children of Angle/Distance and Speed Blocks

### Go Forward

To Go Forward it is necessary to increment robot position, considering the current position and angle. For example, when constructing a Pentagon the robot will have different angles and, consequently, different wheel speeds. So, taking into account the equations in Section 4.2.4, if the robot has different wheel speeds, that means that the robot is rotating and the position is incremented, using the time, angle and speed of the right and left wheels, as shown in Listing 4.5.

---

#### Listing 4.5 Simple code to Move Forward

---

```

1 this.robot.position.x += this.time * Math.cos(this.angle + Math.PI) * Math.abs (
    this.speedRight - this.speedLeft);
2 this.robot.position.y += this.time * Math.sin(this.angle + Math.PI) * Math.abs (
    this.speedRight - this.speedLeft);
3 this.currentDistance += this.time * Math.abs(this.speedRight - this.speedLeft);

```

---

In case of simulation, as the actual diameter of the robot is not known, this value is ignored. So,  $x$  position is given by the previous position plus the time multiplied by the cosine of the angle and by the difference of the speeds of the wheels. The same for  $y$  position but instead of a cosine, the sine is used. In the end, it is necessary to update the robot's current distance to know when the user-defined distance is reached. If the wheel speed is equal, the expression is the same but it is no longer necessary to subtract the value of the two speeds, using only one speed.

### Go Backward

The backward movement is quite similar to forward. But instead of increasing the position, it is decremented, Listing 4.6.

---

#### Listing 4.6 Simple code to Move Backward

---

```

1 this.robot.position.x -= this.time * Math.cos(this.angle + Math.PI) * .abs (
    this.speedRight - this.speedLeft);

```

---

```

2 this.robot.position.y -= this.time * Math.sin(this.angle + Math.PI) * Math.abs(
    this.speedRight - this.speedLeft);
3 this.currentDistance -= this.time * Math.abs(this.speedRight - this.speedLeft);

```

---

### Turn Left

To Turn Left it is necessary to increment robot rotation, considering the current angle and the angle necessary to turn. The angle is in Radians. In order to do this, the rotation is incremented according to the speed given by the user and the constant time defined as number of frames per second. In this way, it is possible to visualize the movement in small increments until it reaches the user-defined angle, instead of just turning to the requested angle.

Considering the previous Pentagon example, the robot will have a constant value of angle that needs to turn. If the current angle plus the next bit of angle ( $\text{time} \times \text{speed}$ ) is equal or greater than the angle defined by the user, it means that this angle has already been reached and then just need to turn that difference between the fixed angle and the current angle. If not, the current angle needs to continue to be increasing and turning a bit more angle. The corresponding code is represented in Listing 4.7.

---

#### Listing 4.7 Simple code to Turn Left.

---

```

1 if(this.currentAngle + this.time * speed >= this.degToRad(angle)) {
2     this.turn(this.degToRad(angle) - this.currentAngle);
3     this.currentAngle = this.degToRad(angle);
4 } else {
5     this.turn(this.time * speed);
6     this.currentAngle += this.time * speed;
7 }

```

---

Turn function in Listing 4.8 is where rotation is incremented:

---

#### Listing 4.8 turn function.

---

```

1 private turn(angleToTurn: number) {
2     this.robot.rotation.z += angleToTurn;
3 }

```

---

### Turn Right

The Turn Right is quite similar to Left, but instead of a turning with a positive values of angle, it rotates with negative. This is because of the coordinate system of Three.js. Three.js uses positive numbers for counterclockwise rotation and negative numbers for clockwise.

---

#### Listing 4.9 Simple code to Turn Right.

---

```

1 if(this.currentAngle - this.time * speed <= this.degToRad(angle)) {

```

```

2   this.turn(this.degToRad(angle) - this.currentAngle);
3   this.currentAngle = this.degToRad(angle);
4 } else {
5   this.turn(this.time * speed);
6   this.currentAngle -= this.time * speed;
7 }

```

---

Besides movements, the Stop Tag was also implemented, represented in Figure 4.18, to stop any movement of the robot.

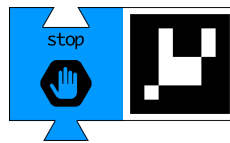


Figure 4.18: Stop.

#### 4.3.2.4 Control

The control blocks implemented in Tactode are: Repeat, Forever, While, If and Else, represented in Figure 4.19. These blocks need to verify a condition. If that condition is verified then the body can execute its children. In case of loops like repeat, forever and while it is necessary to always check the condition. The execution stops when the condition is no longer verified, with the exception of Forever block, as the name suggest, the cycle never stops, so Tactode has a stop button to stop the simulation when desired. The same happens with If and Else statement, but instead of a condition verified in a loop, it only checks once and its children only execute once.

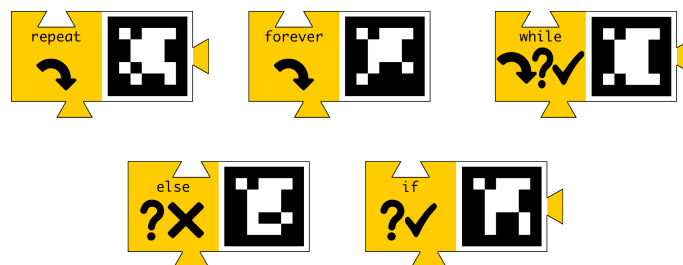


Figure 4.19: Controls.

So, the conditions of the controls Blocks can have different children. Repeat Block only needs a Number of Repetitions, Table 4.9. Forever Block does not need a condition and If/Else and While need conditions that return true or false, Tables 4.7 and 4.8. For example,  $4 < 5$  is a true condition.

Condition
Logical Operators
Comparison Operators

Condition
Logical Operators
Comparison Operators

Table 4.7: If and Else Condition's children

Table 4.8: While Condition's children

Condition
Answer
Number
Variable
Numerical Operators

Table 4.9: Repeat Condition's children

### 4.3.2.5 Variables

Variables in Tactode are created using TypeScript dictionary. When the user wants to create a variable in the puzzle, the Simulator program creates that variable internally, having the capability of also updating its value. In Figure 4.21 it is possible to see the Create Block, where a name can be set to a variable, and the Set Block, where a value can be set to a previously created variable. There are 395 available variables in Tactode, Figure 4.20 shows the variable number 0.



Figure 4.20: Variable number 0.

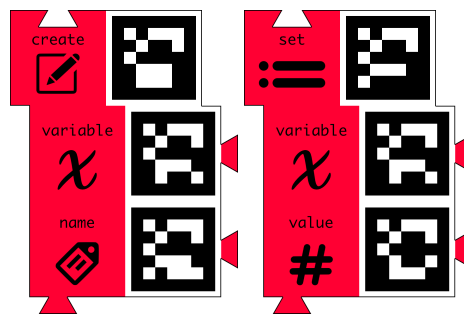


Figure 4.21: Create Variables and Set Values.

In this way, the Variable Create Block has two children, the first one is the Variable Block and the second one is its name, so it can be a group of letters. Variable Set Block also has two children, the first one is the Variable Block and the second one is the value set by the user. That value can be a number, a Numerical Operator or defined by another variable, as shown in the next Table 4.10.

Value
Answer
Number
Variable
Numerical Operators

Table 4.10: Value Block's Children

#### 4.3.2.6 Events

The event implemented is Flag Event. Flag Event is defined for the platforms: Cozmo, Sphero, Robobo and Scratch. When the Flag button is clicked, the simulation can start. This is useful, for example, when it is needed to add an obstacle to the simulation. In this way, it is possible to open the Simulator, introduce the obstacle anywhere and click Flag button to start simulation. The Flag Block is represented in Figure 4.22 and doesn't have any children. In order to implement Flag is necessary to make a promise, so that the simulation waits until the button Flag is clicked. After that, the promise is resolved.

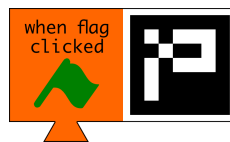


Figure 4.22: Flag.

#### 4.3.2.7 Sensors

There are different types of sensors. Proximity sensors, Line following sensors and Question-Answer sensors.

The Question-Answer sensors, represented in Figure 4.23, are implemented like a variable. When the Simulator detects the Question piece, a question followed by an input box appears, so that, the user can write the answer. That answer is internally saved and can be used in other blocks of the puzzle. In Figure 4.27, shown in the Regular Polygon Challenge section, it is possible to see the question box followed by an input box. The question appears to the user in the Simulator side and the simulation only continues after the user responds to the question. In the figure, the number of sides introduced by the user was five and the robot drew a polygon with five sides.

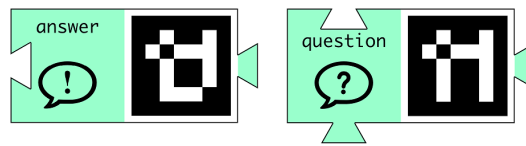


Figure 4.23: Question and Answer.

In Proximity Sensors there are several types. The proximity sensor implemented in the Simulator is the Front Sensor, represented in Figure 4.24. When an obstacle is detected, an action chosen by the user and programmed in the puzzle is activated. It can be, for example, stop in front of the obstacle or run away from it. Section 4.3.3.2 explains some of these, represented in Figures 4.28 and 4.29.

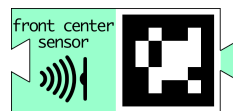


Figure 4.24: Front Sensor.

Line Following was also implemented. When the piece Line Color is detected, a line is drawn and the robot can follow the line with the piece Way Forward and finish when the line is over. Line Color, Way Forward and Line End are represented in the Figure 4.25.

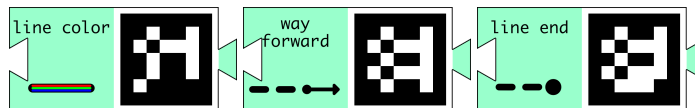


Figure 4.25: Line Color, Way Forward and Line End.

#### 4.3.2.8 Visual

The only visual piece used in the Simulator was PenDown. PenDown is used when the user wants to see the path made by the robot.

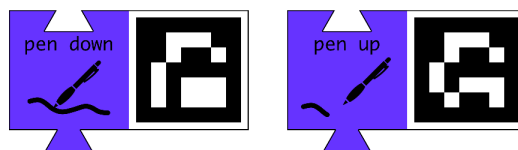


Figure 4.26: Pen Down and Pen Up pieces.



When Simulator detects this piece, the function `drawLine()` is called. This function draws a line in the same position of the robot to look like a track. This is really useful when drawing polygons because it is possible to truly visualize the shape.

---

**Listing 4.10** Draw line function.

---

```

1 private drawLine() {
2   let geometryLine = new THREE.Geometry();
3   geometryLine.vertices.push(new THREE.Vector3(this.robot.position.x,
4     this.robot.position.y, 0));
5   geometryLine.vertices.push(new THREE.Vector3(this.robot.position.x +
6     this.time * Math.cos(this.angle) * this.speedRight,
7     this.robot.position.y + this.time * Math.sin(this.angle) *
8     this.speedLeft, 0));
9   this.line = new THREE.Line(geometryLine, this.materialLine);
10  this._SCENE.add(this.line);
11 }

```

---

When PenDown is no longer needed, the piece PenUp can be used and the robot stops drawing the line.

### 4.3.3 Challenges

For each target platform, a set of challenges was designed for experiments. These challenges are detailed in this section. There are many possibilities to program each target. Challenges were designed to improve educational value. Kids will improve math concepts by using operators such as addition, subtraction, division, multiplication, and tact to move using velocity and sensors. They can also program flow control, such as repeat, forever, while, if, and else. There are four main challenges: regular polygon construction, two types of obstacle reaction and line follow.

#### 4.3.3.1 Regular polygon

This challenge is intuitive for children understand. They need to know some concepts about regular polygons. Regular Polygons have all sides with the same length and their internal and external angles have the same amplitude. There are  $n$  regular polygons with  $n$  sides where the amplitude that the robot needs to turn is  $360^\circ/n$ .

In Figure 4.27 it is possible to see a program for building a pentagon and its simulation. Note that, the Tag Pen Down is only available for platform scratch, so this program only works when the platform scratch is selected. Another exception is that Ozobot does not implement events like the Tag Flag. So, this program can be built without pen down tag in Cozmo, Shero, Robobo, and Ozobot (without flag). In Scratch and Robobo, it is also feasible to ask the user a question like how many sides the user wants. After the question, the program waits to the answer and uses it to build the polygon, like Figure 4.27 shows.

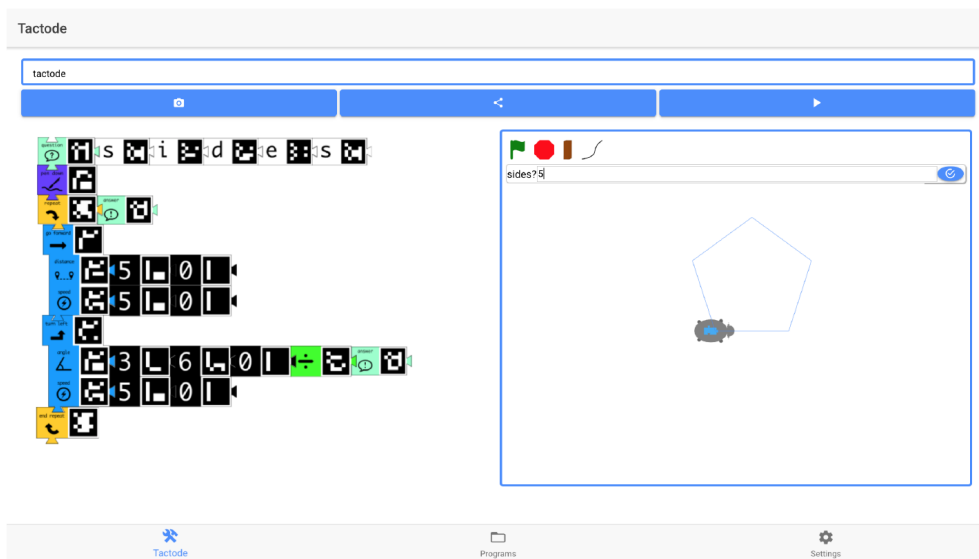


Figure 4.27: Puzzle and simulation of a pentagon.

#### 4.3.3.2 Obstacle Reaction

The obstacle reaction and sensors are for Ozobot. In these examples, the robot can stop or run away when facing an obstacle. The obstacle can be placed anywhere and moved around the scene. In Figures 4.28 and 4.29 different robot reactions to that situations can be seen. In both cases, the robot is given instructions to move in order to draw a square. The robot tries to complete the orders but in Figure 4.28 it stops when the obstacle is in front of it and, in Figure 4.29, the robot pick back direction, running away from the obstacle.

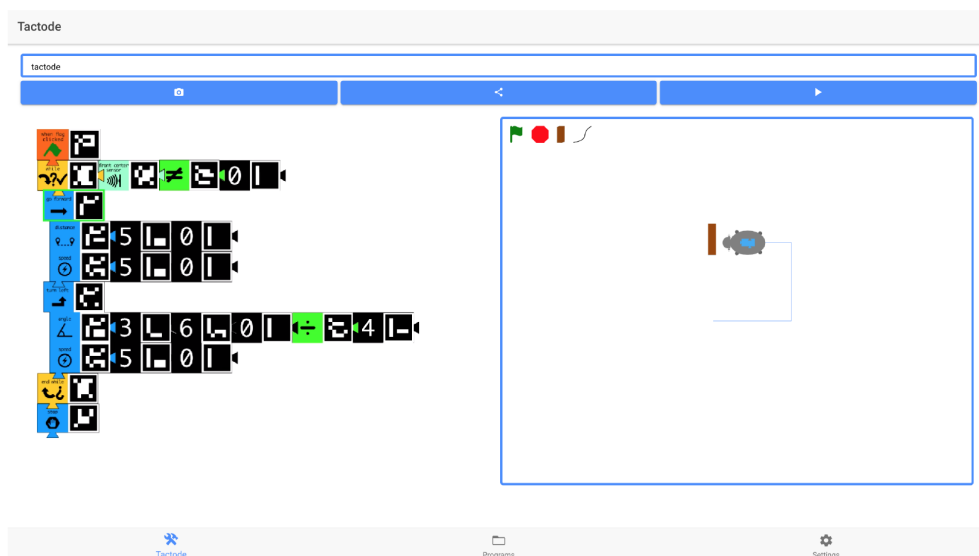


Figure 4.28: Puzzle and simulation of stopping in front of an obstacle.

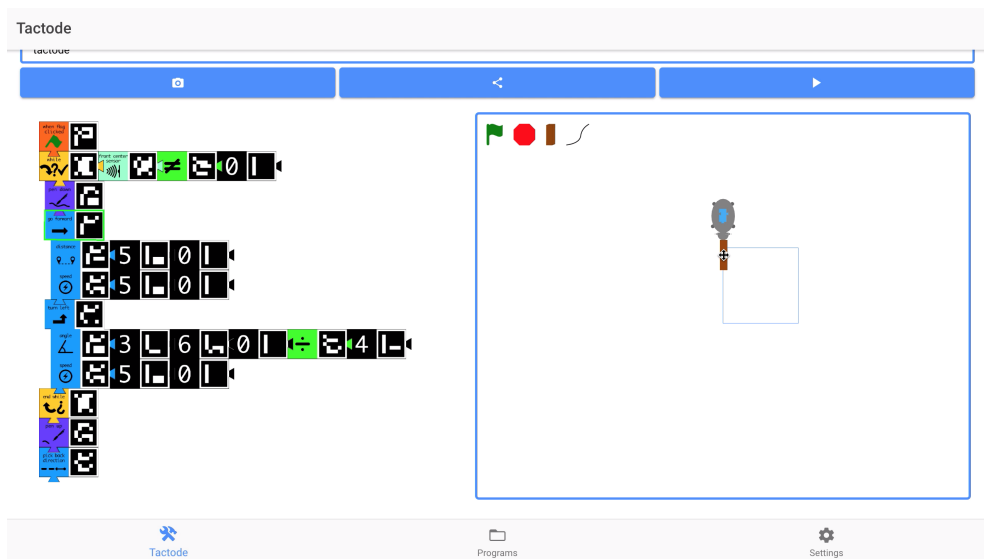


Figure 4.29: Puzzle and simulation of running away of an obstacle.

### 4.3.3.3 Follow Line

This functionality only works in Ozobot, as it is the only one with line detection capabilities. The line is generated randomly when the button line is clicked. The figure 4.30 shows that while the robot sees a black line, it follows it. When the line ends, the while loop ends, stopping the movement.

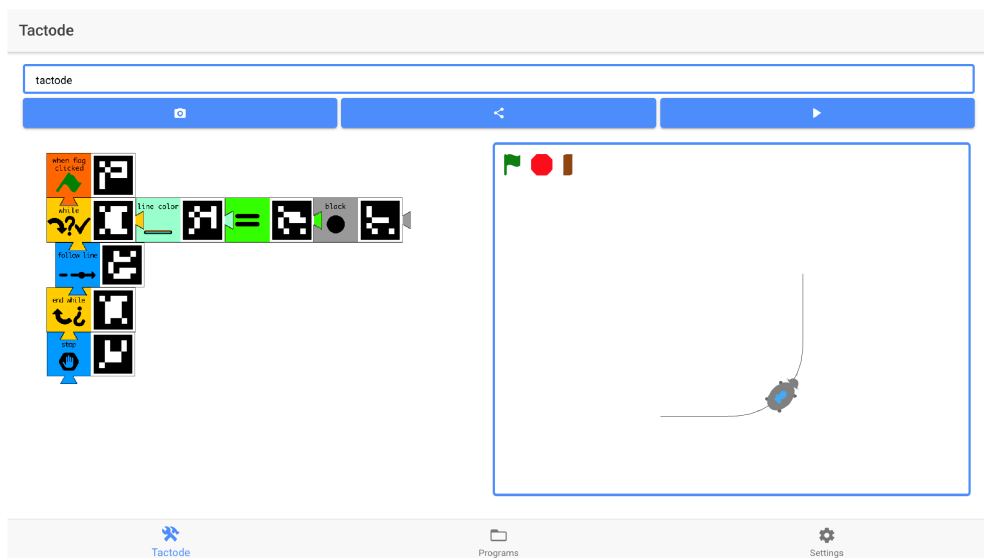


Figure 4.30: Puzzle and simulation of a line follow.

## **4.4 Conclusion**

The application together with the Simulator have an important role in the Tactode tangible language. The children are not limited to the usage of a robot to see what they created in the puzzle, they can execute the program in the Simulator. It allows freedom of movement. The users do not need to carry the tangible pieces if they already have the photographs of the programs, they just need to upload the photograph in the application and see its simulation. A set of Tactode pieces were implemented in the Simulator and a collection of challenges were presented as suggestions to be used, for example, in classrooms. However, the Simulator implementation can be improved. More pieces can be implemented such as more sensors, colors and sound pieces.

## Chapter 5

# Results and Experiments

In this chapter, the main functionalities will be demonstrated, according to the requirements defined in the Section 4.1. It will be divided in three tests with different programs. Test Program 1 is focused on the general functionality, Test Program 2 on an incorrect puzzle and Test Program 3 on the Simulator.

### 5.1 Test Program 1

Starting from the beginning, Figure 5.1 shows the home page.

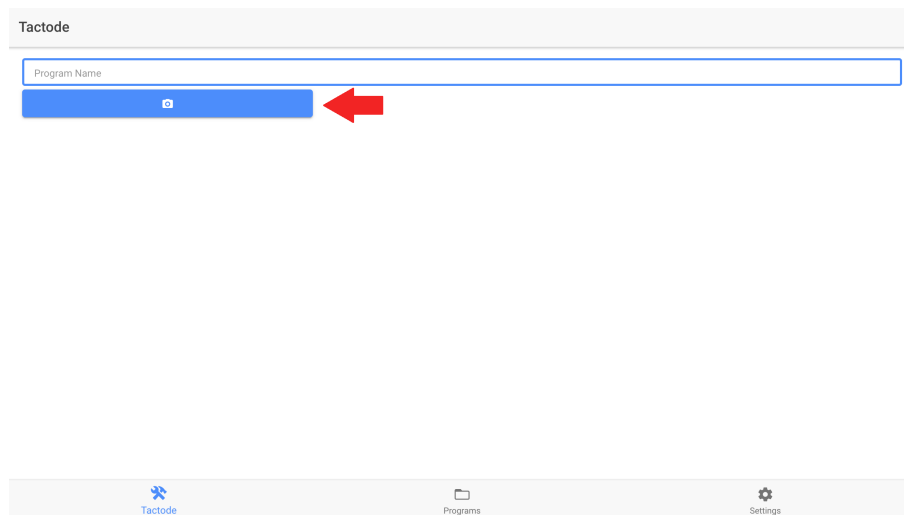


Figure 5.1: Tactode Application Home Page.

When the application is opened for the first time and the user tries to click the camera button, ticked as red in Figure 5.1, it shows a message that no platform and source was selected.

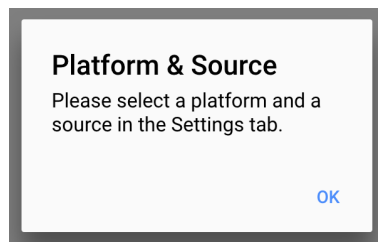


Figure 5.2: No platform and Source selected.

When clicked button "OK", a new PopUp window will appear with all platforms available to choose, Figure 5.4, letter B, and, followed by another PopUp window to select the source of the image, Figure 5.4, letter C. If the user goes to Settings Tab, by clicking in button A, in Figure 5.3, without selecting Platform and Source, they will see empty Platform and Source. The language is set, by default, to English but it can be changed to Portuguese.

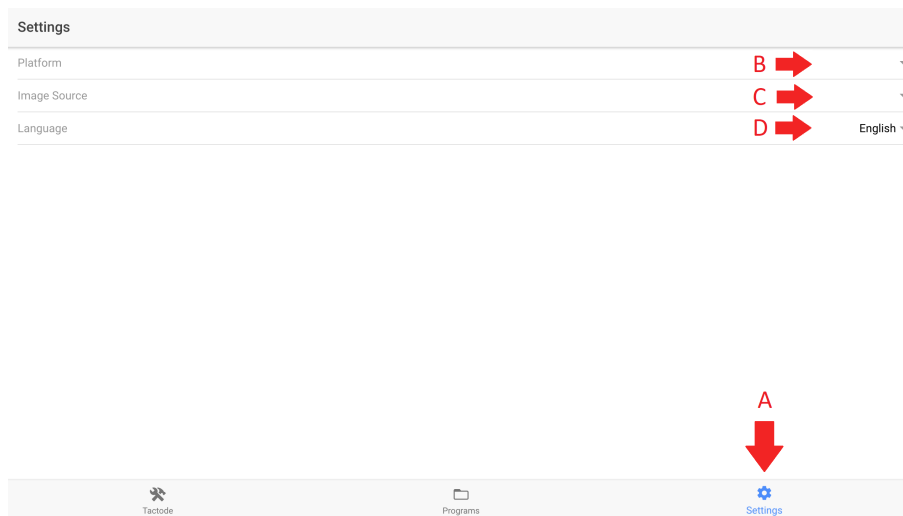


Figure 5.3: Settings Tab.

When clicked Platform, the PopUp Platform window mentioned before will come up and the user can select one from Cozmo, Ozobot, Robobo, Scratch, Sphero and Python. This is presented in Figure 5.4 by letter B.

The user also needs to select the image source, Figure 5.4, letter C, where option "File" is a photograph that already exists in the device and "Camera" is to take a new picture.

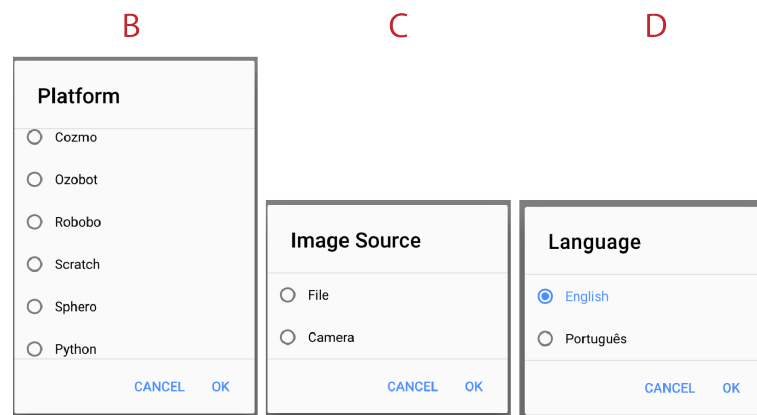


Figure 5.4: Choose Platform, Source and Language.

The settings selected were Cozmo as Platform, File as Image Source and the Language is English. Now, the user is able to proceed, he can write a Name to their program. Because "File" was selected as source, the user can choose a puzzle created before. Figure 5.5 shows Application after file uploaded.

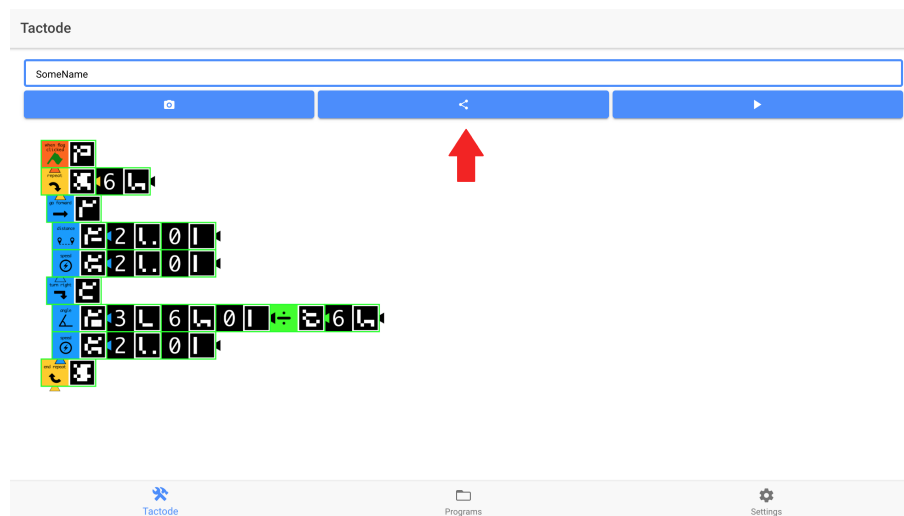


Figure 5.5: Puzzle selected with no errors.

Because this is a correct puzzle, the user is able to share the program to the platform selected or simulate. So when they click the button marked with the red arrow, in Figure 5.5, a file will be downloaded. This file has the transpiled code to the platform language selected, Figure 5.6.

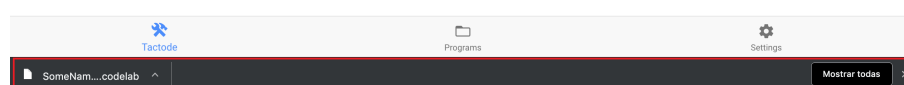


Figure 5.6: Downloaded file, after click button share.

In the Programs Tab, this correct puzzle is now available, Figure 5.7.

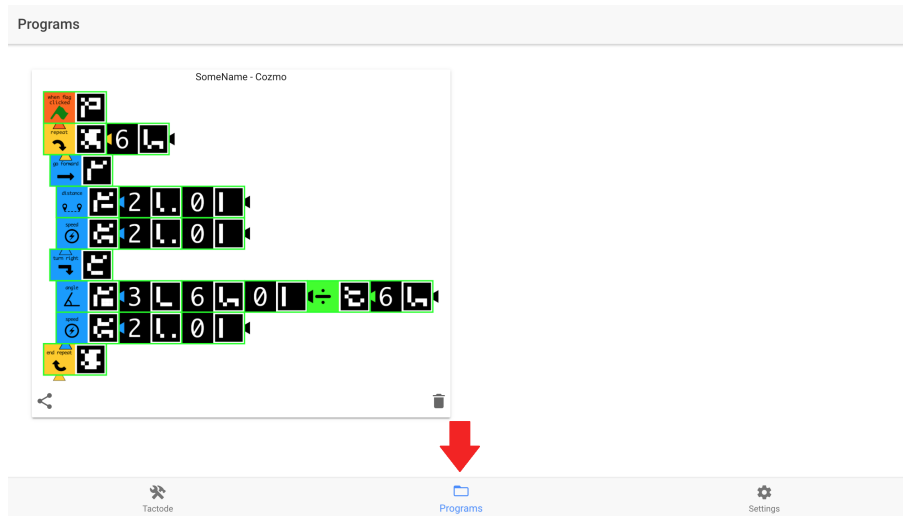


Figure 5.7: Programs in database.

## 5.2 Test Program 2

If the user try to upload a program that has errors or that some pieces that do not exist for that platform, errors will be shown and the share and simulate buttons are no longer available, as is the case of Figure 5.9. In this case, the Platform property was changed for what it is in the Figure 5.8.

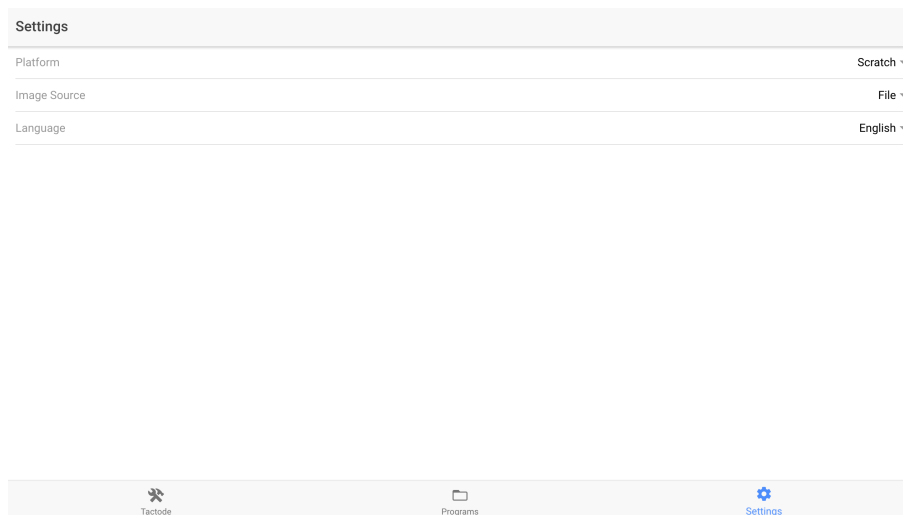


Figure 5.8: New settings.

As Scratch platform does not support the Line Color Tag, the first error is about the platform type. Since this tag is not acceptable, it is not possible to use a Tag Color. Follow line piece is also



not defined to Scratch. The space marked with red boxes, in Figure 5.9, is the space of share and simulate button.

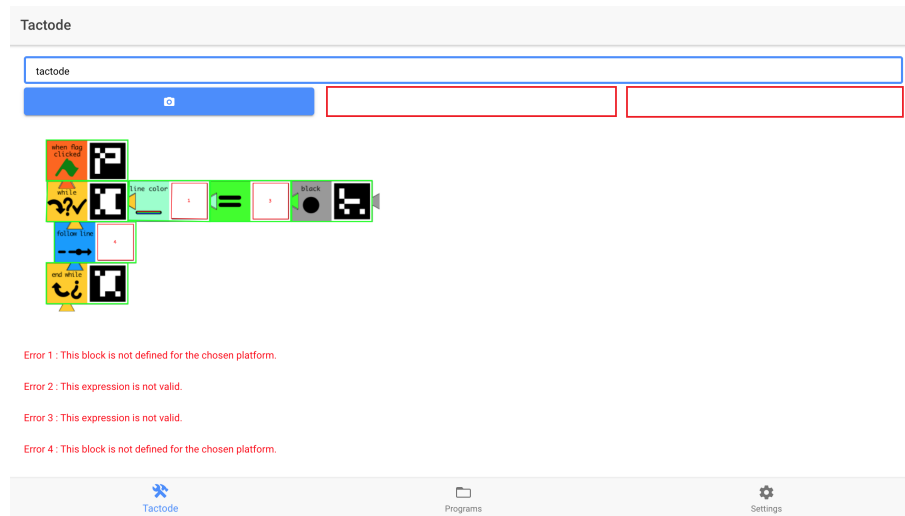


Figure 5.9: Program with errors.

### 5.3 Test Program 3

Lets see what happen if the user tries to upload a new program, now with no errors and compatible with the target platform, Figure 5.10.

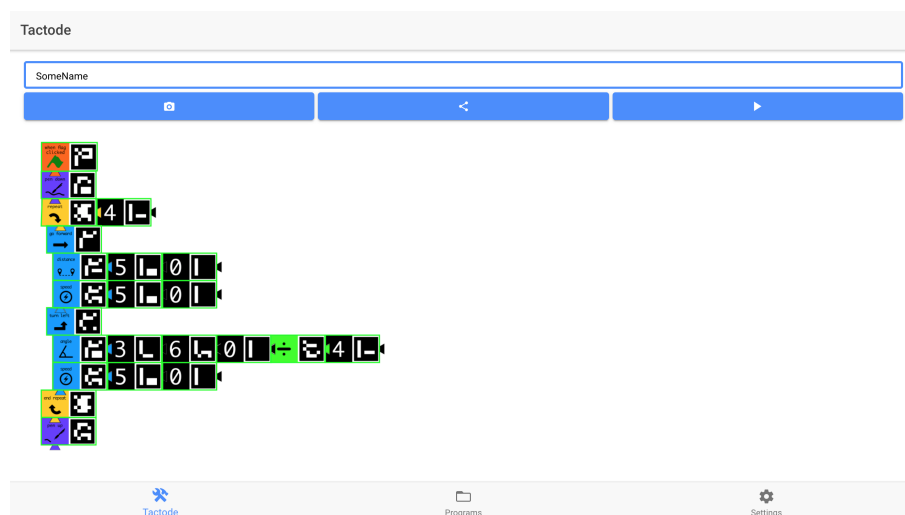


Figure 5.10: Correct Program.

Verifying the content of the programs database, it is possible to check that now exists 2 programs with different target platforms, Figure 5.11.

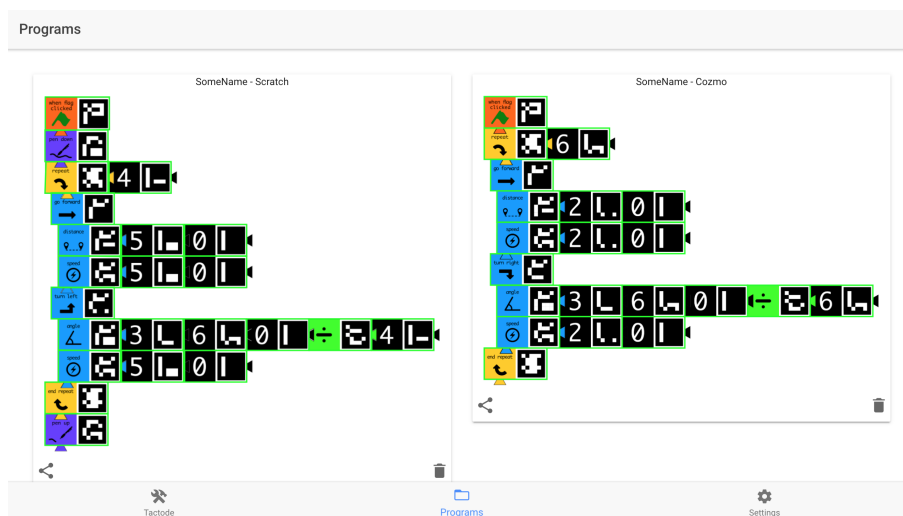


Figure 5.11: Database with 2 programs.

This time, instead of clicking in Share button, the user click on the Simulator button to open it. The Figure 5.12 shows the Simulator scenario.

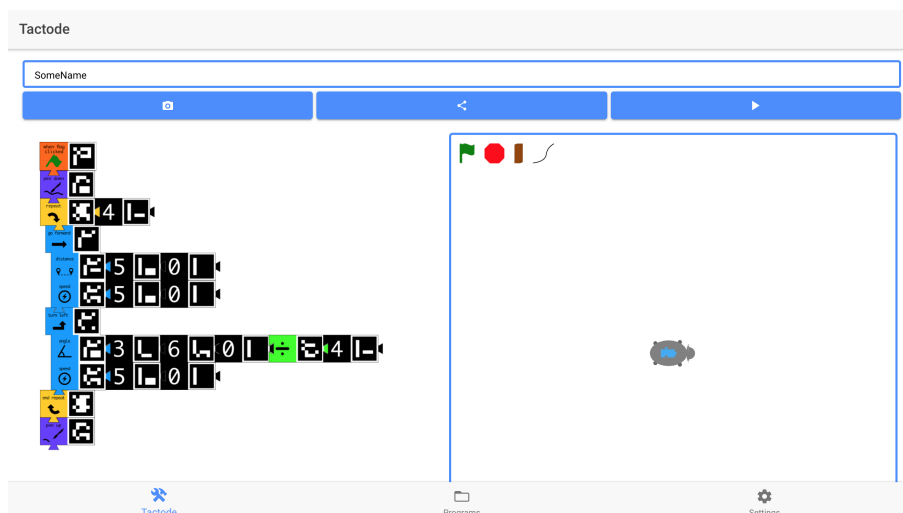


Figure 5.12: Simulator opened.

To better understand what is happening next, it is important to explain the content of the puzzle uploaded. This puzzle only starts when button Flag is clicked, marked with a red circle and arrow (Figure 5.13). The goal is to build a square. For that, a repeat cycle that will repeat four times (one per each side of the square) is used. The sides are made by moving forwarding and turning left ninety ( $360 / 4 = 90$ ) degrees. When Figure 5.13 was captured, the robot was halfway through the second repetition, moving Forward as pointed with the red arrow.

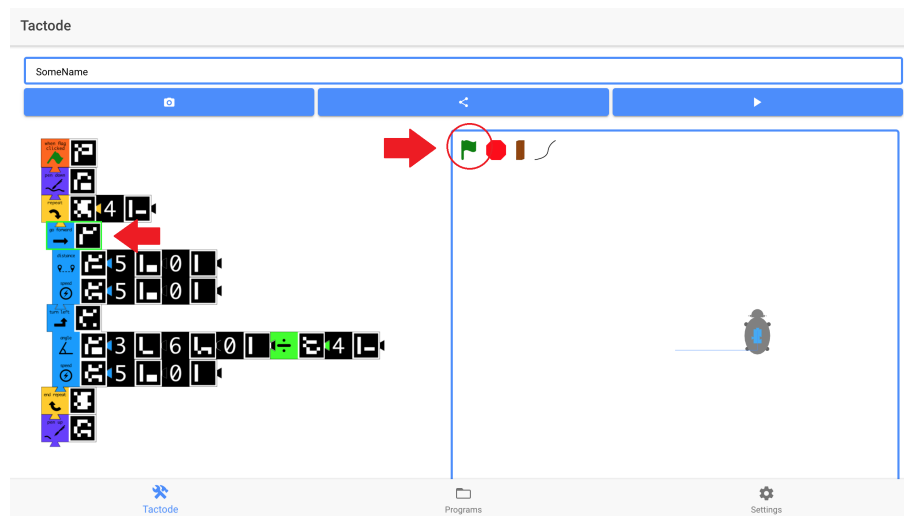


Figure 5.13: After Flag clicked.

Now, the user can stop the simulation, by clicking in the Stop button, marked with a red circle and arrow, in Figure 5.14, or simply wait until the end of the simulation, Figure 5.15. When the stop button is clicked, the tile, where the robot stops, continues highlighted, to the user know in which action stopped. When the simulation ends, no tiles are highlighted.

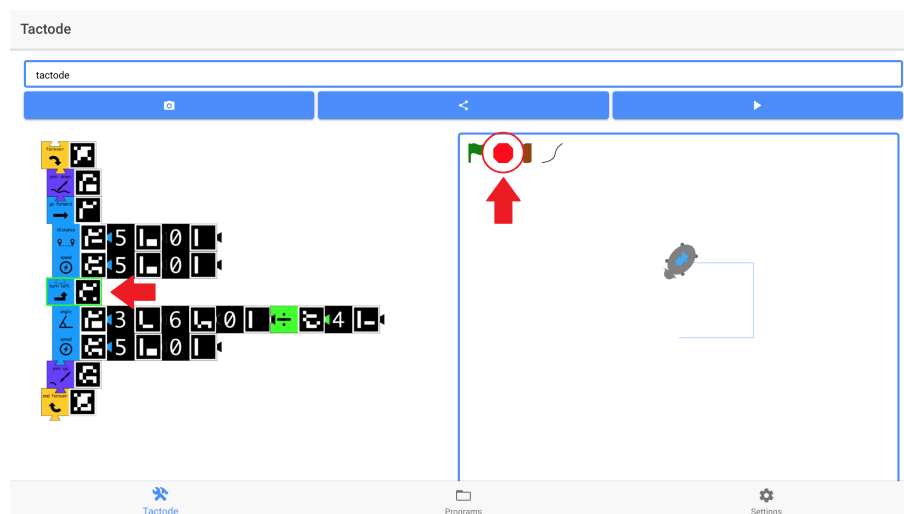


Figure 5.14: Simulation stopped.

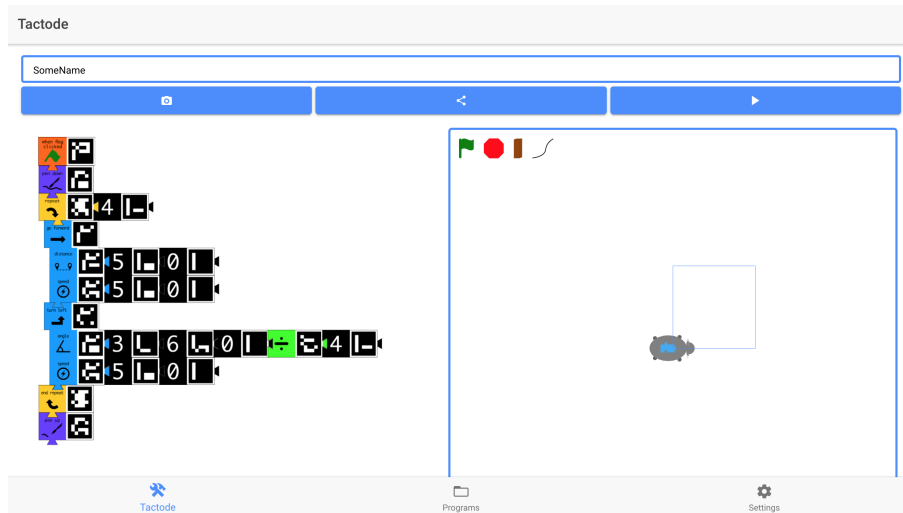


Figure 5.15: End of Simulation.

## 5.4 Conclusion

In this Chapter it was possible to see how Tactode Application works and its functionalities. The simulator should allow a pleasant to the users and easy to understand. In the future, several groups of students, with different ages, should test the Simulator, in order to figure out what is missing and the existing problems.

## Chapter 6

# Conclusion and Future Work

The Tactode Tangible Programming System is focused on motivating children, attending elementary education, to understand how programming and how computer science work. Using robotics and a Simulator to encourage an evolution in the children's Computational Thinking, so they can understand and solve technological problems.

The Tactode System after this work allows robot programming and now users can give orders to a virtual robot, run it on the Simulator, that also can execute orders made in the puzzle, and possibly later test on a real robot. This makes Tactode independent of a real robot, so the users do not need to buy any robot to use Tactode, only tangible pieces.

The Simulator is easy to understand, since the pieces that are being executed, by the virtual robot, are highlight during the execution.

The Simulator also introduces some features to make it closer to reality, like front sensors to detect obstacles and has different reactions to those situations, follow a line, question and answer interaction with the users and a stop button to stop simulation whenever is necessary.

The simulation was built with web programming tools and tested with several devices, being responsive at any screen.

### 6.1 Future Work

To be accessible to more people, it would be interesting if the applications could be a Progressive Web App (PWA), which runs on any browser with Internet access. The PWA only needs internet to open the application and then it can run offline. Another attractive improvement would be the possibility of drag and drop Tactode pieces, similar to Scratch, in order to build the puzzle directly into the App. This implies creating a set of online puzzle pieces, similar to Blockly.

In this way, Tactode can be more general and accessible to society anytime, anywhere. More challenges could be created, a labyrinth, for example, and the current ones could be improved. An interesting improvement would be if it were possible to paint the line to be followed, with different colors. Another could be to choose what kind of reactions the robot might have when facing an obstacle, such as approaching the obstacle instead of running away and being available on all sides

(/sensors) of the robot (front, left frontal, right frontal and back). In terms of results, it would be interesting if tested with more groups of students with different ages.

# References

- [1] H. Suzuki and H. Kato. Algoblock: a tangible programming language, a tool for collaborative learning. In *Proceedings of the 4th European logo conference*, pages 297–393, 1993. URL: [https://mafiadoc.com/queue/algoblock-a-tangible-programming-language\\_59d157681723dd2a0293242f.html](https://mafiadoc.com/queue/algoblock-a-tangible-programming-language_59d157681723dd2a0293242f.html).
- [2] P. Wyeth and H. Purchase. Designing technology for children: moving from the computer into the physical world with electronic blocks. *Information Technology in Childhood Education Annual*, 2002(1):219–244, 2002. URL: <http://eprints.gla.ac.uk/14107/>.
- [3] Global STEAM solutions. Cubelets. Available at <https://www.globalsteamsolutions.com.br/cubelets/>, 2019.
- [4] Fisher Price. Think & learn code-a-pillar application. Available at <https://www.walmart.ca/en/ip/fisher-price-think-learn-code-a-pillar-learning-toy/6000196194493>.
- [5] KUBO Robotics. KUBO. Available at <https://kubo.education/bundles/>, 2017.
- [6] M. S. Horn and R. J. K. Jacob. Designing tangible programming languages for classroom use. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction, TEI '07*, pages 159–162, New York, NY, USA, 2007. ACM. URL: <http://doi.acm.org/10.1145/1226969.1227003>, doi:10.1145/1226969.1227003.
- [7] D. Wang, T. Wang, and Z. Liu. A tangible programming tool for children to cultivate computational thinking. *The Scientific World Journal*, 2014, 2014. doi:10.1155/2014/428080.
- [8] The Windy Side. Osmo adds steam to kids’ coding with new music creator system. Available at <https://thewindyside.com/2017/10/04/osmo-adds-steam-to-kids-coding-with-new-music-creator-system/>, 2019.
- [9] S. Goyal, R. S. Vijay, C. Monga, and P. Kalita. Code bits: An inexpensive tangible computational thinking toolkit for k-12 curriculum. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction, TEI '16*, pages 441–447, New York, NY, USA, 2016. ACM. URL: <http://doi.acm.org/10.1145/2839462.2856541>, doi:10.1145/2839462.2856541.
- [10] LEGO group. Support - mindstorms. Available at <https://www.lego.com/en-us/mindstorms/support>, 2018.

- [11] DoPlay! Robótica. avanzado. lego mindstorms ev3. Available at <https://www.doplay.es/product/robotica-nivel-avanzado/>, 2019.
- [12] Ozobot. About us. Available at <https://ozobot.com/about-us>.
- [13] Amy Eguchi. *Bringing Robotics in Classrooms*, pages 3–31. Springer International Publishing, Cham, 2017. URL: [https://doi.org/10.1007/978-3-319-57786-9\\_1](https://doi.org/10.1007/978-3-319-57786-9_1), doi:10.1007/978-3-319-57786-9\_1.
- [14] OECD (2019). Education at a glance 2019: Oecd indicators. 2019. URL: <https://dx.doi.org/10.1787/f8d7880d-en>.
- [15] Ana Pedro, João Filipe Matos, João Piedade, Nuno Dorotea. Probótica - Programação e Robótica no Ensino Básico. URL: [http://www.erte.dge.mec.pt/sites/default/files/probotica\\_protect\\_discretionary{\char\hyphenchar\font}{}{}\\_linhas\\_orientadoras\\_2017.pdf](http://www.erte.dge.mec.pt/sites/default/files/probotica_protect_discretionary{\char\hyphenchar\font}{}{}_linhas_orientadoras_2017.pdf), 2017.
- [16] Robert M. Capraro & Mary Margaret Capraro Peter Boedeker, Sandra Nite. Women in STEM: The Impact of STEM PBL Implementation on Performance, Attrition, and Course Choice of Women. *2015 IEEE Frontiers in Education Conference (FIE)*, 2015.
- [17] A. Cardoso, A. Sousa, and H. Ferreira. Programming for young children using tangible tiles and camera-enable handheld devices. pages 6389–6394, 11 2018. doi:10.21125/iceri.2018.2504.
- [18] A. Cardoso, A. Sousa, and H. Ferreira. Easy robotics with camera devices and tangible tiles. pages 6400–6406, 11 2018. doi:10.21125/iceri.2018.2506.
- [19] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, November 2009. URL: <http://doi.acm.org/10.1145/1592761.1592779>, doi:10.1145/1592761.1592779.
- [20] N. Fraser. Ten things we’ve learned from blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 49–50, October 2015. doi:10.1109/BLOCKS.2015.7369000.
- [21] E. Pasternak, R. Fenichel, and A. N. Marshall. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B B)*, pages 21–24, October 2017. doi:10.1109/BLOCKS.2017.8120404.
- [22] Google for Education. Blockly. Available at <https://developers.google.com/blockly/>.
- [23] Microsoft. Makecode. Available at <https://makecode.com>.
- [24] Lifelong Kindergarten Group at the MIT Media Lab. Scratch. Available at <https://scratch.mit.edu>, 2005.
- [25] Stavros Christodoulakis Charalampos Kyfonidis, Nektarios Moumoutzis. Block-c: A block-based programming teaching tool to facilitate introductory c programming courses. *IEEE Global Engineering Education Conference*, 2017.



- [26] Uri Wilensky David Weintrop. To block or not to block, that is the question: Students' perceptions of blocks-based programming. *14th International Conference on Interaction Design and Children, IDC 2015 - Boston, United States*, 2015.
- [27] T. Sapounidis, S. Demetriadis, and I. Stamelos. Evaluating children performance with graphical and tangible robot programming tools. *Personal and Ubiquitous Computing*, 19(1):225–237, January 2015. URL: <https://doi.org/10.1007/s00779-014-0774-3>, doi: [10.1007/s00779-014-0774-3](https://doi.org/10.1007/s00779-014-0774-3).
- [28] Nikolaus Correll, Chris Wailes, and Scott Slaby. A one-hour curriculum to engage middle school students in robotics and computer science using cubelets. In M. Ani Hsieh and Gregory Chirikjian, editors, *Distributed Autonomous Robotic Systems*, pages 165–176, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [29] B. Wohl, B. Porter, and S. Clinch. Teaching computer science to 5-7 year-olds: An initial study with scratch, cubelets and unplugged computing. In *Proceedings of the Workshop in Primary and Secondary Computing Education, WiPSCE '15*, pages 55–60, New York, NY, USA, 2015. ACM. URL: <http://doi.acm.org/10.1145/2818314.2818340>, doi: [10.1145/2818314.2818340](https://doi.org/10.1145/2818314.2818340).
- [30] Fisher Price. Think & learn code-a-pillar. Available at <https://fisher-price.mattel.com/shop/en-us/fp/think-learn/think-learn-code-a-pillar-dkt39>.
- [31] D. Wang, C. Zhang, and H. Wang. T-maze: A tangible programming tool for children. In *Proceedings of the 10th International Conference on Interaction Design and Children, IDC '11*, pages 127–135, New York, NY, USA, 2011. ACM. URL: <http://doi.acm.org/10.1145/1999030.1999045>, doi: [10.1145/1999030.1999045](https://doi.org/10.1145/1999030.1999045).
- [32] Osmo. Osmo coding family. Available at <https://www.playosmo.com/en/coding-family/>.
- [33] Threejs. Three.js. Available at <https://threejs.org/>, 2019.
- [34] Lewy Blue. DISCOVER three.js. URL: <https://discoverthreejs.com/>, 2019.
- [35] Ospennikova E., M. Ershov, and I. Iljin. Educational robotics as an inovative educational technology. *Procedia - Social and Behavioral Sciences*, 214:18 – 26, 2015. World-wide trends in the development of education and academic research, Sofia, Bulgaria, 15-18 June,2015. URL: <http://www.sciencedirect.com/science/article/pii/S1877042815059431>, doi: <https://doi.org/10.1016/j.sbspro.2015.11.588>.
- [36] Scratch - Divulgar, Formar, Partilhar. UAC - Using Arduíno in the Classroom. URL: <http://eduscratch.dge.mec.pt/>.
- [37] ERTE - Equipa de Recursos e Tecnologias Educativas - Ministério da Educação. Clubes de Programação e Robótica. URL: <http://www.erte.dge.mec.pt/clubes-de-programacao-e-robotica>.
- [38] CDi - Portugal. Apps for Good. URL: <http://cdi.org.pt/apps-for-good/>.

- [39] ERTE - Equipa de Recursos e Tecnologias Educativas - Ministério da Educação. UAC - Using Arduino in the Classroom. URL: <http://erte.dge.mec.pt/uac-using-arduino-classroom/>.
- [40] Fatemeh Mansour Kiaie Ahmad Khanlari. Using Robotics for STEM Education in Primary/Elementary Schools: Teachers' Perceptions. *The 10th International Conference on Computer Science & Education (ICCSE 2015)*, 2015.
- [41] M. Mattis R. Burke. Developing career commitment in STEM-related fields: myth versus reality. *Women and minorities in science, technology, engineering and mathematics: Upping the numbers*, 2007.
- [42] National Academy of Engineering. How to Attract Young People to Engineering: Make a Difference' Message is Key. Available at <http://www.sciencedaily.com/releases/2008/06/080624145221.htm>, 2008.
- [43] Christina Chalmers and Rod Nason. *Systems Thinking Approach to Robotics Curriculum in Schools*, pages 33–57. Springer International Publishing, Cham, 2017. URL: [https://doi.org/10.1007/978-3-319-57786-9\\_1](https://doi.org/10.1007/978-3-319-57786-9_1), doi:10.1007/978-3-319-57786-9\_1.
- [44] Linda Rae Markert. Gender Related to Success in Science and Technology. *The Journal of Technology Studies*, pages 21–29, 1996.
- [45] Ozobot. Getting started guide. Available at <https://files.ozobot.com/stem-education/ozoblockly-getting-started.pdf>.
- [46] Roberta. Learning to program intuitively in the open roberta lab. Available at <https://www.roberta-home.de/en/lab/>.
- [47] Blockly games : About. Available at <https://blockly-games.appspot.com/about?lang=en>.
- [48] Scratch. About scratch. Available at <https://scratch.mit.edu/about>.
- [49] Francisco Romero Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 2018.
- [50] Upplication. Progressive web app - the new way of understanding apps. [https://www.upplication.com/images/pwa/upplication\\_PWA\\_2018\\_en-EN.pdf](https://www.upplication.com/images/pwa/upplication_PWA_2018_en-EN.pdf), 2018.
- [51] Apple. Developer - Support. URL: <https://developer.apple.com/support/compare-memberships/>, 2019.
- [52] Google. How to use the Play Console. URL: <https://support.google.com/googleplay/android-developer/answer/6112435?hl=en>, 2019.
- [53] Microsoft. Account types, locations, and fees. URL: <https://docs.microsoft.com/pt-pt/windows/uwp/publish/account-types-locations-and-fees>, 2018.
- [54] Ionic. Browser Support. URL: <https://ionicframework.com/docs/intro/browser-support/>, 2019.
- [55] Ionic. What is Ionic Framework? URL: <https://ionicframework.com/docs/intro>, 2019.

- [56] N. Ahmadi, F. Lelli, and M. Jazayeri. Supporting domain-specific programming in web 2.0: A case study of smart devices. In *2010 21st Australian Software Engineering Conference*, pages 215–223, April 2010. doi:10.1109/ASWEC.2010.36.
- [57] David Flanagan. *JavaScript: The Definitive Guide*. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2011. URL: <http://www.wowebook.pw/book/javascript-the-definitive-guide-6th-edition>, doi:10.17226/9824.
- [58] Codecademy. Intruduction to JavaScript. URL: [https://www.codecademy.com/courses/introduction-to-javascript/lessons/introduction-to-javascript/exercises/intro?action=resume\\_content\\_item](https://www.codecademy.com/courses/introduction-to-javascript/lessons/introduction-to-javascript/exercises/intro?action=resume_content_item), 2019.
- [59] TypeScript. TypeScript. Available at <https://www.typescriptlang.org/>, 2019.
- [60] Codecademy. JAVASCRIPT PROMISES. URL: [https://www.codecademy.com/courses/introduction-to-javascript/lessons/promises/exercises/introduction?action=resume\\_content\\_item](https://www.codecademy.com/courses/introduction-to-javascript/lessons/promises/exercises/introduction?action=resume_content_item), 2019.
- [61] Ionic creator. Custom Code Editing. URL: <https://docs.usecreator.com/docs/custom-code-editing>, 2019.



## **Appendix A**

### **Tactode Pieces**

## 1 Numbers

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	0	number 0	✓	✓	✓	✓	✓	✓
	1	number 1	✓	✓	✓	✓	✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	9	number 9	✓	✓	✓	✓	✓	✓

Table 1: Tactode number pieces

## 2 Colors

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	50	color black	✓	✓	✓	✓	✓	✗
	51	color blue	✓	✓	✓	✓	✓	✗
	52	color cyan	✓	✓	✓	✗	✓	✗
	53	color green	✓	✓	✓	✓	✓	✗
	54	color magenta	✓	✓	✓	✗	✓	✗
	55	color orange	✓	✓	✓	✗	✓	✗
	56	color red	✓	✓	✓	✓	✓	✗
	57	color white	✓	✓	✓	✗	✓	✗
	58	color yellow	✓	✓	✓	✗	✓	✗

Table 2: Tactode color pieces

### 3 Letters







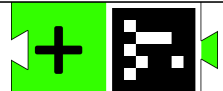


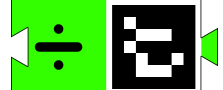
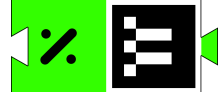
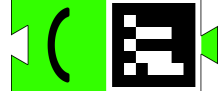

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	100	letter A	✓	✓	✓	✓	✓	✓
	101	letter B	✓	✓	✓	✓	✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	125	letter Z	✓	✓	✓	✓	✓	✓
	132	letter a	✓	✓	✓	✓	✓	✓
	132	letter b	✓	✓	✓	✓	✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	157	letter z	✓	✓	✓	✓	✓	✓

Table 3: Tactode letter pieces

### 4 Operators

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	200	addition operator	✓	✓	✓	✓	✓	✓
	201	subtraction operator	✓	✓	✓	✓	✓	✓
	202	multiplication op.	✓	✓	✓	✓	✓	✓
	203	division operator	✓	✓	✓	✓	✓	✓
	204	remainder operator	✓	✓	✓	✓	✓	✓
	205	left parenthesis	✓	✓	✓	✓	✓	✓
	206	right parenthesis	✓	✓	✓	✓	✓	✓

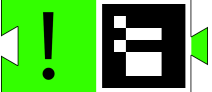












Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	207	negation operator	✓	✓	✗	✓	✓	✓
	208	disjunction operator	✓	✓	✓	✓	✓	✓
	209	conjunction operator	✓	✓	✓	✓	✓	✓
	210	equality operator	✓	✓	✓	✓	✓	✓
	211	inequality operator	✓	✗	✓	✗	✗	✓
	212	less than operator	✓	✓	✓	✓	✓	✓
	213	less than or eq. op.	✓	✗	✓	✗	✗	✓
	214	greater than operator	✓	✓	✓	✓	✓	✓
	215	greater than or eq. op.	✓	✗	✓	✗	✗	✓
	218	absolute value op.	✓	✓	✓	✓	✓	✓

Table 4: Tactode operator pieces

## 5 Variables

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	300	variable 300	✓	✓	✓	✓	✓	✓
	301	variable 301	✓	✓	✓	✓	✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	394	variable 394	✓	✓	✓	✓	✓	✓









Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	399	create variable						
	398	a variable, 300-394	✓	✓	✓	✓	✓	✓
	397	variable name, string						
	396	set variable value						
	398	a variable, 300-394	✓	✓	✓	✓	✓	✓
	395	value, numerical exp.						

Table 5: Tactode variable pieces

## 6 Events





Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	400	when flag clicked event	✗	✓	✓	✓	✓	✗

Table 6: Tactode event pieces















## 7 Control

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	502	repeat a number of times	✓	✓	✓	✓	✓	✓
	503	end of repeat	✓	✓	✓	✓	✓	✓
	504	repeat forever	✓	✓	✓	✓	✓	✓

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
 	505	end of forever	✓	✓	✓	✓	✓	✓
 	506	execute if cond. true	✓	✓	✓	✓	✓	✓
 	507	execute if cond. false	✓	✓	✓	✓	✓	✓
 	508	end of if	✓	✓	✓	✓	✓	✓
 	509	repeat while cond. true	✓	✓	✓	✓	✓	✓
 	510	end of while	✓	✓	✓	✓	✓	✓
 	511	break out of loop	✓	✗	✗	✗	✗	✓

Table 7: Tactode control pieces

## 8 Sensors

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
 	600	front left prox. sensor	✓	✗	✗	✓	✗	✗
 	601	front right prox. sensor	✓	✗	✗	✓	✗	✗
 	602	back left prox. sensor	✓	✗	✗	✓	✗	✗
 	603	back right prox. sensor	✓	✗	✗	✓	✗	✗
 	604	front far left prox.	✗	✗	✗	✓	✗	✗
 	605	front far right prox.	✗	✗	✗	✓	✗	✗
 	606	front center prox.	✗	✗	✗	✓	✗	✗






















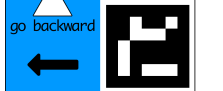
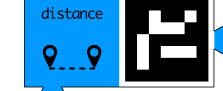
Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
back center sensor 	607	back center prox.	✗	✗	✗	✓	✗	✗
way left 	612	is there line on left	✓	✗	✗	✗	✗	✗
way right 	613	is there line on right	✓	✗	✗	✗	✗	✗
way forward 	614	is there line forward	✓	✗	✗	✗	✗	✗
line end 	615	is this the line end	✓	✗	✗	✗	✗	✗
line color 	616	get color of line	✓	✗	✗	✗	✗	✗
question 	618	ask a question, string	✗	✗	✗	✓	✓	✓
answer 	619	user answer to quest.	✗	✗	✗	✓	✓	✓

Table 8: Tactode sensor pieces

## 9 Movement

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
stop 	700	stop wheels	✓	✓	✓	✓	✗	✗
wheel speed 	701	set wheel speed						
left 	702	left wheel, num. exp.	✓	✓	✓	✓	✗	✗
right 	703	right wheel, num. exp.						

Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
  	704	move forward						
	708	distance, num. exp.	✓	✓	✓	✓	✗	✗
	710	speed, num. exp.						
  	705	move backward						
	708	distance, num. exp.	✓	✓	✓	✓	✗	✗
	710	speed, num. exp.						
  	706	turn left						
	709	angle, num. exp.	✓	✓	✓	✓	✗	✗
	710	speed, num. exp.						
  	707	turn right						
	709	angle, num. exp.	✓	✓	✓	✓	✗	✗
	710	speed, num. exp.						
 	704	move forward						
	708	distance, num. exp.	✗	✗	✗	✗	✓	✗
 	705	move backward						
	708	distance, num. exp.	✗	✗	✗	✗	✓	✗



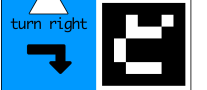






Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
 	706	turn left						
	709	angle, num. exp.	×	×	×	×	✓	×
 	707	turn right						
	709	angle, num. exp.	×	×	×	×	✓	×
	711	follow line	✓	×	×	×	×	×
	712	pick left direction	✓	×	×	×	×	×
	713	pick right direction	✓	×	×	×	×	×
	714	pick forward direction	✓	×	×	×	×	×
	715	pick back direction	✓	×	×	×	×	×

Table 9: Tactode movement pieces

## 10 Sound



Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	800	say a phrase, string	✓	✓	✓	✓	✓	✓
	801	think a phrase, string	×	×	×	×	✓	×

Table 10: Tactode sound pieces

## 11 Visual



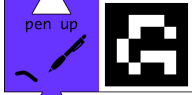
Piece	Tag	Function	Ozobot	Cozmo	Sphero	Robobo	Scratch	Python
	900	erase screen content	×	×	×	×	✓	×
	901	put pen down to write	×	×	×	×	✓	×
	902	lift pen to stop writing	×	×	×	×	✓	×

Table 11: Tactode visual pieces

## **Appendix B**

### **Draft of conference paper**

# Web Based Robotic simulator for Tactode Tangible Block Programming System

Márcia Alves<sup>1</sup>, Armando Sousa<sup>2</sup>, and Ângela Cardoso<sup>3</sup>

<sup>1</sup> FEUP

<sup>2</sup> INESC-TEC & FEUP

<sup>3</sup> INEGI

**Abstract.** Nowadays, with the increase of technology, it is important to adapt children and their education to this development. This article proposes programming blocks for young students to learn concepts related to math and technology in an easy and funny way, using a Web Application and a robot.

The students can build a puzzle, with tangible tiles, giving instructions for the robot execute. Then, it is possible to take a photograph of the puzzle and upload it on the application. This photograph is processed and converted in executable code for the robot that can be simulated in the app by the virtual robot or performed in the real robot.

**Keywords:** Education, Programming, Technology for education, Tangible system, Web Application, ArUco, Simulator.

## 1 Introduction

Since the birth of the internet, the development of technology has been increasing. With this growth, education also has to change and evolve [1]. Computational thinking (CT) is a set of thinking skills, habits, and approaches that are essential to solving problems using a computer [2] and has to be more present in education. However, students are reluctant to choose computer programming as a subject due to its perceived difficulty. But, it is well known that children that are introduced to computer programming will finish graduates in computer science in the future [1].

Tangible programming makes programming an activity that is accessible to the hands and minds by making it more direct, less abstract [2] and easy to understand when connected to robotics. Robotics apply and content knowledge in a meaningful and exciting way [1], so students are able to improve CT and think deeply, therefore they learn how the technology works.

Although it is difficult for teachers to include new things in the regular curriculum, because of the academic standards, the aim is to connect robotics with STEM (Science, Technology, Engineering, Mathematics) standards. Besides, it is also difficult for schools, support the price of that kind of technology for all students. [3]



Howsoever, the propose is Tactode as a Web Application, to program robots like a game in the classroom, based on programming blocks with tiles, similar to a puzzle that children need to solve. It is possible to create the puzzle, with tangible tiles, take a photograph and upload it in the application. In this way, children can see, in the app, a virtual robot simulating the code or send it to the real robot. This is useful to test the code created before execute in the real robot, or just to play in anywhere without having to carry the robot.

Hereupon, the suggestion is just a set of puzzle pieces and one computer, tablet or even a smartphone per group of students. In this way, the kids are able to learn about robotics and have a programming logic since a young age in a fun way and also improve team cooperation, working in a group to solve a problem [4]. Even for the teachers is easier to manage a small group of equipment than equipment for each student.

## 2 State Of the Art

### 2.1 Programming Education

When teaching programming to children and young adolescents there are clear advantages in starting with an educational language [5], because it should have a simpler syntax, lower entry-level requirements and sometimes already provide activities aimed at grabbing the attention of the young. Also, it is getting easier everyday to have access to an educational language for every situation, particularly with development of educational programming languages by giant software companies such as Apple (Swift Playgrounds [6]), Google (Blockly [7–9]) and Microsoft (MakeCode [10]).

Block based languages are probably the most influential educational languages of today. As the name implies, they contain a set of predefined blocks of code, which the user typically drags and drops to form a program. Examples of this kind of languages are Scratch [5][11], Snap! [12], Stencyl [13], Blockly, MakeCode, Alice [14–16] and Etoys [17]. While Scratch drew inspiration from the precursors Alice and Etoys, it has since influenced most of the others.

### 2.2 Tangible Blocks Programming Language

A programming language is considered visual when it is mostly text independent, relying on icons and images to represent its elements. This is not, however, the consensus, as all the languages above use text in their blocks and yet they are typically considered visual. Examples of visual languages are ScratchJr [18], Kodu [19], the Lego block based visual programming language designed to program their EV3 [20] robot, Lightbot [21], the Fisher Price Think & Learn Code-a-Pillar [22] in its tablet application format, Coding Safari [23], SpriteBox [24] and codeSpark [25]. All these languages allow younger children, even before they can read, to learn programming concepts while playing.

A tangible programming language is a block-based language whose blocks can be physically grabbed and arranged by the programmer. In their 2015 study,

Sapounidis et al. [26] established many advantages of using tangible interfaces to teach programming, particularly for children up to ten years old. The children completed the tasks faster, with fewer errors, more debugging of the errors they made, better collaboration, using a wider set of different blocks and, in some cases, achieving higher complexity. [27][3] Also, they considered the tangible interface more attractive, more enjoyable and, for the younger ones, easier to use.

As for disadvantages, tangible languages are more expensive and less portable, due to their physical aspect. However, these issues can be mitigated by using less expensive materials and make it possible for schools to manufacturing the tangible blocks themselves.

Examples of tangible languages are AlgoBlock [28], Electronic [29], Cubelets [30–32], the physical robot version of the Fisher Price Think & Learn Code-a-Pillar [33], TagTile [34], Quetzal and Tern [35], T-Maze [36, 2], the Osmo Coding Family [37], and CodeBits [38].

The more recent tangible languages are replacing electronic components with image processing to execute their programs, unless they need the electronics because the language is simultaneously a robot, which is what happens in Cubelets and Code-a-Pillar.

### 2.3 Similar Projects

In order to understand the similar projects mentioned before, this section explains briefly some of them.

**Lego Mindstorm EV3** is a set of programmable robotics construction for ages greater than 10 years. The aim is to build the own robot with Lego pieces, program and command it. The EV3 set includes bricks, motors, and sensors to build the robot and make it walk, talk, move. It also comes with the necessary software and App, where the robot can be easily programmed and controlled, with basic tasks, from PC, Mac, tablet or smartphone. [39]

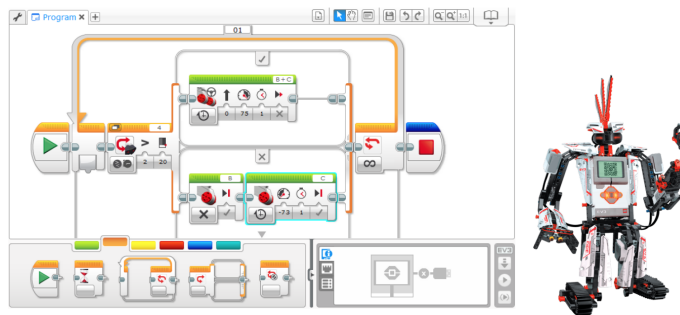
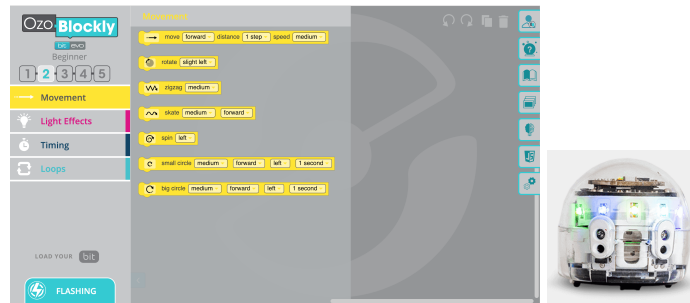


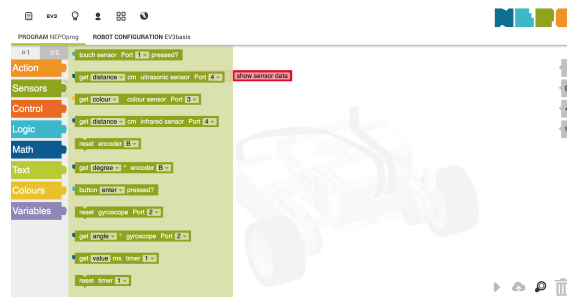
Fig. 1. The Lego Mindstorms EV3 programming interface and robot.

**Ozobot** has two ways to code their robots. They can be coded online with OzoBlockly or screen-free with Color Codes. The purpose is to inspire young minds to go from consuming technology to creating it. [40] OzoBlockly can be used with an application or in a Web browser. The application can be used in iOS or Android tablet, working with the Evo robot. The browser is also compatible with Evo robot, while used with a computer, or compatible with bit robot for a computer or tablet. [41]



**Fig. 2.** The OzoBlockly Editor and Ozobot robot.

**Open Roberta Lab** is a free platform that makes learning programming easy from the first steps to program robots with multiple sensors and capabilities. It can be used at any time without installation by any devices, PC, Mac or tablet, with an Internet browser. Thanks to the programming language NEPO (a graphical programming language developed at Fraunhofer IAIS), simple programs can be created like puzzle pieces [42].



**Fig. 3.** Open Roberta Lab.

**Blockly Games** is a free Google project with a series of educational games that teach programming. There are different games, with different levels, so children

who have not had prior experience are ready to use conventional text-based languages by the end of these games. [43]

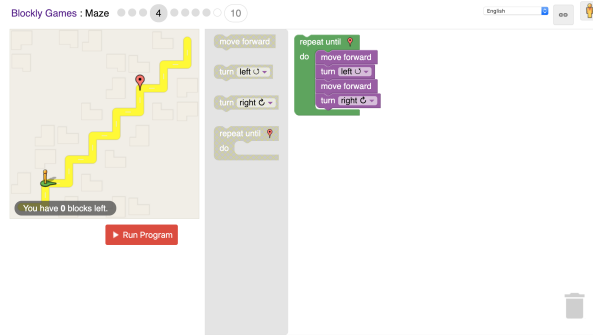


Fig. 4. Blockly Games.

**Scratch** is made for children between six and eight years old learn how to code and important strategies for solving problems, designing projects, and communicating ideas. [44] The activity is mixing graphics, animations, photos, music, and sound, supporting different types of projects like stories, games, animations, simulations, so people are all able to work on projects they care about. [5]

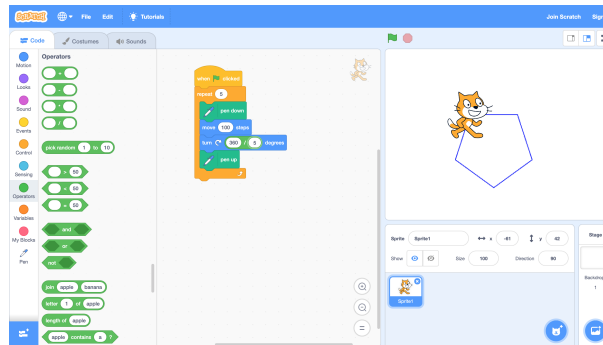


Fig. 5. Scratch.

## 2.4 Development tools

The Tactode application aims to simulate the real robot into the application. In order to be accessible for all users, the Tactode application should be compatible with all platforms. [45] Nowadays, exist several options to create that. It was used Ionic Framework, that generates applications for multiple systems from a

single source code, using AngularJs and TypeScript. [46] Besides, it was used Threejs to develop the simulator. Ionic uses Cordova to have access to host operating systems features such as Camera, GPS, Flashlight, etc. It includes mobile components, typography, interactive paradigms, and an extensible base theme. [47] In Tactode, the principal feature used is Camera. Threejs is a high-level JavaScript library and Application Programming Interface used to create and display animated 3D graphics in web browser. The use of WebGL allows complex animations to be created without having to use plugins. [48]

### 3 Tactode programming system

Tactode Programming System is made for elementary school children with the aim of teaching them how to program in a fun and interactive way, by building puzzles that represent a chain of commands which a robot will follow, so they can see how it reacts to the different puzzle compositions. The mobile device will capture a photograph of the tangible puzzle, process it, create the commands for the robot and show the results in a simulator. The focus is the simulation of the robot and, in this section, will be addressed the requirements for this simulator, its architecture and possible challenges that children can create.

#### 3.1 Requirements

The purpose of Tactode Programming System is to be easy to understand for the users, so they can immediately see how Tactode works and easy to use and install. It has a Web App prepared to process the tangible puzzle directly in the browser.

There are two ways to upload a puzzle on the app:

- Upload a previous photo by searching an image on the device.
- Take a photo directly, using a hand-held camera device where the application is running.

After uploading the photo, children have two options to see the execution.

- Simulating in the application, seeing a virtual robot execution.
- Real execution, using a real robot.

#### 3.2 Architecture

This section will explain how Tactode pieces are defined and built in the simulator. Every uploaded puzzle is processed as Abstract Syntax Tree Structure (AST). Each original piece has a corresponding Block and each of them has an array of other Block objects (children) and also a parent Block object. In this way, each object knows the parent and the children. However, there are some extra elements in the AST that do not have a piece in Tactode, such as:

- RootBlock: special Block with no parent that serves as the root of the AST. Its children are the command blocks that are not inside of any control flow block, which means that have no indentation in the tangible language.
- BodyBlock: child of control flow blocks RepeatBlock, ForeverBlock, IfBlock, ElseBlock or WhileBlock.
- ConditionBlock: child of RepeatBlock, IfBlock or WhileBlock and as the name suggests, it contains the condition to be verified by these control flow blocks;

Figure 6 shows an example of a square - a loop, running four times, that inside move forward, with a fixed distance, and turn right/left 90°.

In this case, Root Block has one child - Repeat Block - and this block has three more children:

- Condition Block that usually has a child the number of repeats of the cycle.
- Body Block where are introduced the main instructions. It has two children: Forward Block and Turn Left Block. Forward Block has two more children - Distance and Speed - each with Number block as children. Turn Left Block is similar but instead of distance, it has Angle Block.
- End Repeat Block only ends the repeat, as the name indicates.

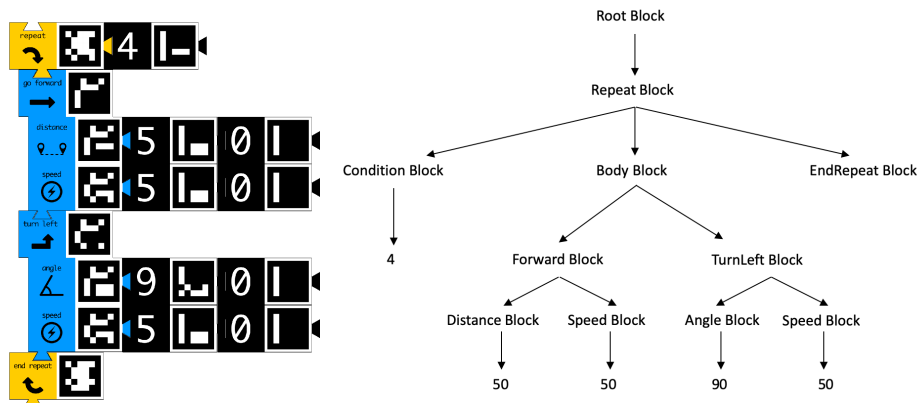


Fig. 6. AST of a square.

## 4 Challenges

For each target platform, a set of challenges is designed for experiments. These challenges are detailed in this section. There are many possibilities to program each target. Challenges were designed to improve educational value. Kids will improve math concepts by using operators such as addition, subtraction, division, multiplication, and tact to move using velocity and sensors. They can also

program flow control, such as repeat, forever, while, if, and else. There are four main challenges: regular polygon construction, two types of obstacle reaction and line follow.

#### 4.1 Regular polygon

This challenge is intuitive for children understand. They need to know some concepts about regular polygons. Regular Polygons have all sides with the same length and their internal and external angles have the same amplitude. There are  $n$  regular polygons with  $n$  sides where the amplitude of each angle the robot needs to turn is  $360^\circ/n$ .

In Figure 7 it is possible to see a program for building a pentagon and its simulation. Note that, the tag pen down is only available for platform scratch, so this program only works when the platform scratch is selected. Another exception is that Ozobot does not implement events like the tag Flag. So, this program can be built without pen down tag in Cozmo, Shero, Robobo, and Ozobot (without flag). In Scratch and Robobo, it is also feasible to ask the user a question like how many sides the user wants. After the question, the program waits to the answer and uses that to build the polygon, like Figure 7 shows.

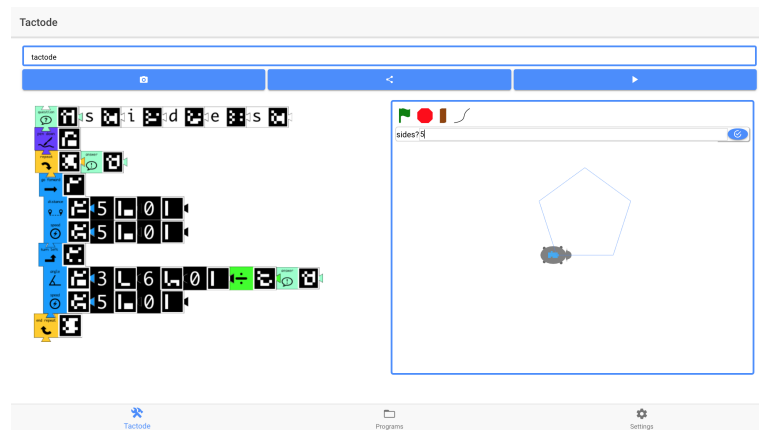


Fig. 7. Puzzle and simulation of a pentagon.

#### 4.2 Obstacle Reaction

The obstacle reaction and sensors are for Ozobot. In these examples, the robot can stop or running away when facing an obstacle. The obstacle can be placed anywhere and moved around the scene. In Figure 8 and 9 can be seen different robot reactions to that situations. In both cases, it is told to the robot to build a square. The robot tries to complete the orders but in Figure 8 it stops when the



Fig. 8. Puzzle and simulation of stopping in front of an obstacle.

obstacle is in front of it and, in Figure 9, the robot pick back direction, running away from the obstacle.



Fig. 9. Puzzle and simulation of running away of an obstacle.

### 4.3 Follow Line

This functionality only works in Ozobot because it is the only one with line detection capabilities. The line is generated randomly when the button line is clicked. The figure 10 shows that while the robot sees a black line, it follows it. When the line ends, the robot no longer sees the line and the while loop ends, stopping the movement.



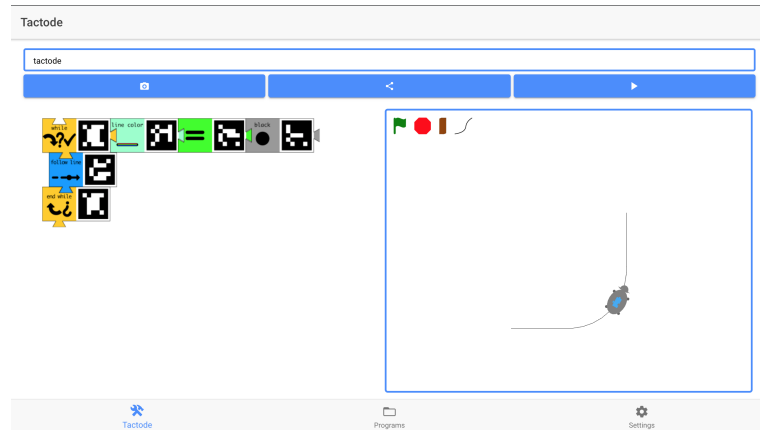


Fig. 10. Puzzle and simulation of a line follow.

## 5 Conclusion

This project focuses on motivating children, attending elementary education, engineering, and programming, using an application and robotics, so they can understand how computer science works. Children can give orders to a virtual robot, run it on the simulator or a real robot. To be accessible to more people, it would be interesting if the applications could be a Progressive Web App (PWA), which runs on any browser with Internet access. The PWA only needs internet to open the application and then it can run offline. In this way, Tactode can be more general and accessible to society anytime, anywhere. More challenges could be created, a labyrinth, for example, and the current ones could be improved. An interesting improvement would be if it were possible to paint to be followed, with different colors. Another could be to choose what kind of reactions the robot might have when facing an obstacle, such as approaching the obstacle instead of running away and being available on all sides (/ sensors) of the robot (front, left frontal, right frontal and back).

## References

1. Eguchi A.: Bringing Robotics in Classrooms. In: Khine M. (eds) Robotics in STEM Education. pp 3?31. Springer International Publishing, Cham (2017). doi:10.1007/978-3-319-57786-9\1
2. Danli Wang, Tingting Wang, and Zhen Liu: A Tangible Programming Tool for Children to Cultivate Computational Thinking. In: The Scientific World Journal. vol. 2014 (2014). doi:10.1155/2014/428080
3. Cardoso, A. and Sousa, A. and Ferreira, H.: Programming for young children using tangible tiles and camera-enable handheld devices. In: 11th annual International Conference of Education, Research and Innovation. pp 6389?6394 (2018). doi: 10.21125/iceri.2018.2504

4. Chetty, J.: Combatting the War Against Machines: An Innovative Hands-on Approach to Coding. In: Khine M. (eds) *Robotics in STEM Education: Redesigning the Learning Experience*. pp 59?83. Springer International Publishing, Cham (2017). doi:10.1007/978-3-319-57786-9\\_3
5. Resnick, M. and Maloney, J. and Monroy-Hernández, A. and Rusk, N. and Eastmond, E. and Brennan, K. and Millner, A. and Rosenbaum, E. and Silver, J. and Silverman, B. and Kafai, Y.: Scratch: Programming for All. In: *Communications of the ACM*. pp 60?67 (2009). doi:10.1145/1592761.1592779
6. Apple: Swift Playgrounds. <https://www.apple.com/swift/playgrounds>
7. Fraser, N.: Ten things we've learned from Blockly. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. pp 49?50 (2015). doi:10.1109/BLOCKS.2015.7369000
8. Pasternak, E. and Fenichel, R. and Marshall, A. N.: Tips for creating a block language with Blockly. In: *2017 IEEE Blocks and Beyond Workshop (B B)*. pp 21?24 (2017). doi:10.1109/BLOCKS.2017.8120404
9. Google for Education: Blockly. <https://developers.google.com/blockly>
10. Microsoft: MakeCode. <https://makecode.com>
11. Lifelong Kindergarten Group at the MIT Media Lab: Scratch (2005). <https://scratch.mit.edu>
12. Mönig, J.: Snap!. <http://snap.berkeley.edu/about.html>
13. Chung, J.: Stencyl. <http://stencyl.com>
14. Pausch, R. and Burnette, T. and Capehart, A. C. and Conway, M. and Cosgrove, D. and DeLine, R. and Durbin, J. and Gossweiler, R. and Koga, S. and White, J.: Alice: Rapid Prototyping for Virtual Reality. In: *IEEE Computer Graphics and Applications*. vol. 15, pp 8?11. IEEE Computer Society Press, Los Alamitos, CA, USA (1995). doi:10.1109/38.376600
15. Cooper, Stephen and Dann, Wanda and Pausch, Randy: Alice: A 3-D Tool for Introductory Programming Concepts. In: *Journal of Computing Sciences in Colleges*. vol. 15, pp 107?116. Consortium for Computing Sciences in Colleges, USA (2000). <http://dl.acm.org/citation.cfm?id=364133.364161>
16. Carnegie Mellon University: Alice. <https://www.alice.org>
17. Alan Kay et. al.: Squeakland. <http://www.squeakland.org>
18. Lifelong Kindergarten Group at the MIT Media Lab: ScratchJr. <https://www.scratchjr.org>
19. Microsoft Research: Koduv (2009) <https://www.kodugamelab.com>
20. Lego: Mindstorms: Learn To Program (2013). <https://www.lego.com/en-us/mindstorms/learn-to-program>
21. Yaroslavski, D.: LightBot (2017). <http://lightbot.com>
22. Fisher Price: Think & Learn Code-a-Pillar Application. [https://www.fisher-price.com/en\\_US/brands/think-and-learn/learning-apps/index.html](https://www.fisher-price.com/en_US/brands/think-and-learn/learning-apps/index.html)
23. Hopster: Coding Safari. <https://www.hopster.tv/coding-safari/>
24. SpriteBox LLC: SpriteBox. <http://spritebox.com/hour.html>
25. codeSpark: codeSpark Academy: Kids Coding. <https://codespark.com>
26. Sapounidis, T. and Demetriadis, S. and Stamelos, I.: Evaluating children performance with graphical and tangible robot programming tools. In: *Personal and Ubiquitous Computing*. vol. 19, pp. 225?237 (2015). doi:10.1007/s00779-014-0774-3
27. Cardoso, A. and Sousa, A. and Ferreira, H.: Easy Robotics with Camera Devices and Tangible Tiles. In: *11th annual International Conference of Education, Research and Innovation*. pp. 6400?6406 (2018). doi:10.21125/iceri.2018.2506

28. Suzuki, H. and Kato, H.: AlgoBlock: a tangible programming language, a tool for collaborative learning. In: Proceedings of the 4th European logo conference. pp. 297?393 (1993)
29. Wyeth, P. and Purchase, H.: Designing technology for children: moving from the computer into the physical world with electronic blocks. In: Information Technology in Childhood Education Annual. vol. 2002, pp. 219?244 (2002). <http://eprints.gla.ac.uk/14107/>
30. Modular Robotics: Cubelets (2012). <https://www.modrobotics.com/cubelets/>
31. Correll, Nikolaus and Wailes, Chris and Slaby, Scott: A One-Hour Curriculum to Engage Middle School Students in Robotics and Computer Science Using Cubelets. In: Ani Hsieh, M. and Chirikjian, Gregory (eds) Distributed Autonomous Robotic Systems. pp. 165?176. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). doi: 10.1007/978-3-642-55146-8
32. Wohl, B. and Porter, B. and Clinch, S.: Teaching Computer Science to 5-7 Year-olds: An Initial Study with Scratch, Cubelets and Unplugged Computing. In: Proceedings of the Workshop in Primary and Secondary Computing Education. pp. 55?60. ACM, New York, NY, USA. doi:10.1145/2818314.2818340
33. Fisher Price: Think & Learn Code-a-Pillar. <https://fisher-price.mattel.com/shop/en-us/fp/think-learn/think-learn-code-a-pillar-dkt39>
34. KUBO Robotics: KUBO (2017). <https://kubo-robot.com>
35. Horn, M. S. and Jacob, R. J. K.: Designing Tangible Programming Languages for Classroom Use. In: Proceedings of the 1st International Conference on Tangible and Embedded Interaction. pp. 159?162. ACM, New York, NY, USA (2007). doi: 10.1145/1226969.1227003
36. Wang, D. and Zhang, C. and Wang, H.: T-Maze: A Tangible Programming Tool for Children. In: Proceedings of the 10th International Conference on Interaction Design and Children. pp. 127?135. ACM, New York, NY, USA (2011). doi:10.1145/1999030.1999045
37. Osmo: Osmo Coding Family. <https://www.playosmo.com/en/coding-family/>
38. Goyal, S. and Vijay, R. S. and Monga, C. and Kalita, P.: Code Bits: An Inexpensive Tangible Computational Thinking Toolkit For K-12 Curriculum. In: Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction. pp. 441?447. ACM, New York, NY, USA (2016). doi:10.1145/2839462.2856541
39. LEGO group: Support - Mindstorms (2018). <https://www.lego.com/en-us/mindstorms/support>
40. Ozobot: About us. <https://ozobot.com/about-us>
41. Ozobot: Getting Started Guide <https://files.ozobot.com/stem-education/ozoblockly-getting-started.pdf>
42. Roberta: Learning to program intuitively in the Open Roberta Lab <https://www.roberta-home.de/en/lab/>
43. Google for Education: Blockly Games : About. <https://blockly-games.appspot.com/about?lang=en>
44. Scratch: About Scratch. <https://scratch.mit.edu/about>
45. Ionic: Browser Support (2019). <https://ionicframework.com/docs/intro/browser-support/>
46. Ionic: What is Ionic Framework? (2019). <https://ionicframework.com/docs/intro>
47. Ionic creator: Custom Code Editing (2019) <https://docs.usecreator.com/docs/custom-code-editing>
48. Threejs: Three.js (2019) <https://threejs.org/>