



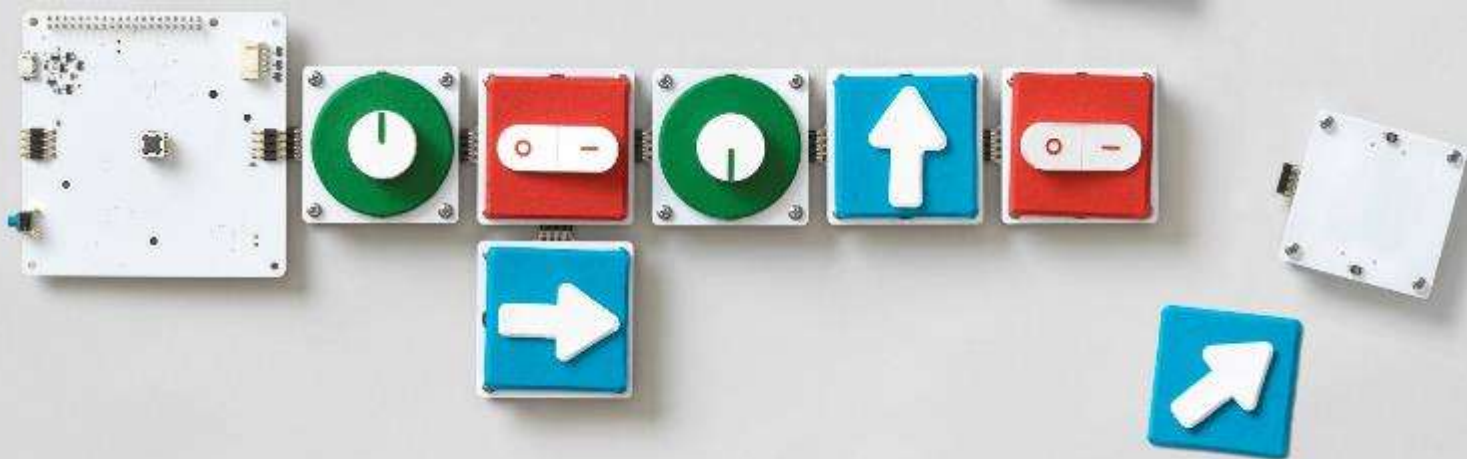
Tanglin

Tangible Programming & Inclusion

TEACHERS HANDBOOK

A GUIDE FOR UNDERSTANDING TANGIBLE PROGRAMMING AND THE IMPLEMENTATION OF THE TANGIN TOOLBOX OF RESOURCES

December 2018



www.tangin.eu

Consortium:



INOVA⁺



This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein. Project N°: 2017-1-PT01-KA201-035975

DOCUMENT SUMMARY INFORMATION

Authors and contributors

Initials	Author	Organisation	Role
PC	Pedro Costa	INOVA+	Trainer
DG	Daniel Gonçalves	C&A	Researcher

Revision history

Rev.	Who	Date	Comment

Quality Control

	Who	Date
Checked and Reviewed by IO Leader		
Approved by Coordinator		

TABLE OF CONTENTS

1	What is tangible programming	1
2	TangIn toolbox.....	3
3	Resources	7
3.1	Summary.....	7
3.2	Kits.....	10
3.3	Instructions	16
3.4	Troubleshooting.....	21
3.5	Tips & Examples.....	24
4	Annexes.....	Error! Bookmark not defined. 7



1 What is tangible programming

There are different definitions for digital thinking and digital programming but, there is the frequent image association to describe it “as solving a puzzle”, in the way of recognizing patterns and breaking problems in smaller and simpler parts and organizing it in a logical and sequential way (table 1). Tangible programming is then a way to use physical objects to make programming an activity that is appealing and accessible to the hands and minds of young children by making it more direct and less abstract.

Table 1 – Digital thinking categories and principles.

Digital Thinking Categories and Principles	
Digital thinking skills	Definition
Abstraction	Process of representing/converting a subject/object (tangible or not) in a more understandable form by eliminating unnecessary detail. Prioritizing and choosing the most relevant descriptors by sorting according to the degree of information (100% meaning that one characteristic is enough to describe the subject)
Decomposition	Separating coherently the logical component parts of a subject/object and keep deconstruct them in simpler units/axioms until they can be understood, solved and evaluated separately but without losing crucial information on the original object/subject
Sequencing	Arrange the different parts of a problem in an order allowing to create steps towards a solution
Automation	By recognizing patterns find shortcuts and create repetitive tasks and loops in a way to save labour and time and improve the flow of information
Debugging	Systematic approach to predict and verify outcomes
Generalization	Strategy of exploring and exploiting previous solutions to similar problems by finding connections and similarities

Why tangible programming?

Training pattern recognition, abstraction and spatial orientation, where we must transpose an object reality to oneself (robots right or left might be different from ours) are important skills that can be used not only as introductory concepts to programming/coding, but also as a method of thinking and problem-solving to be applied virtually in any subject and level.

To foster inclusion: do to the interactive and physical nature of tangible programming, where groups of children come together to solve problems, there is a great opportunity to shorten differences in terms of previous exposure and motivation between different backgrounds and groups of children. Teamwork and group discussions can take place and, differently from a classical computer interface, more than one student can be in control of the input which foster social negotiation and collaborative behaviours.

In summary, tangible programming offers several advantages over graphic alternatives, such as:

- Facilitates collaborative peer-to-peer programming;
- Simplifies debugging processes;
- Helps to narrow gender differences of interest in computing;
- Promotes physical involvement and empathy. With more sensory input (touch, sight, hearing) it is easier to form a mental picture (abstraction) and the embodied experiences can lead to even more efficient ways of learning.

Complementarily with the three key-study areas (Mathematics, Sciences and Technology) explored in this project, and in order to strengthen the STEM approach by promoting inclusion and promotion of holistic student knowledge, some of the United Nations "Sustainable Development Goals"¹ (Fig 1.) that are very related to the Tangible Programming dynamics were purposively included in the framework of the development of the project resources (e.g.: lesson plan – Recycling; exploring the 3 R's: Reduce, Reuse, Recycle).

Ensure inclusive and quality education for all and promote lifelong learning



Achieve gender equality and empower all women and girls



Reduce inequality within and among countries



Sustainably manage forests, combat desertification, halt and reverse land degradation, halt biodiversity loss



Figure 1 – Sustainable development goals that are most applicable to the TangIn project

¹ Full list of United Nations Sustainable Goals is available at: <https://www.un.org/sustainabledevelopment/sustainable-development-goals/>

2 TanglIn toolbox

The TanglIn toolbox consists on a set of lesson plans (Table 2) that cover several subjects such as Mathematics, Biology, Geography, etc. integrated with tangible programming resources.

Each lesson plan serves as a guideline for the teachers to explore a given subject using tangible programming. The plans should not be considered as immutable and air tight as teachers must have the freedom to adapt to the dynamics of the class and explore what they consider more relevant.

The cover of each lesson plan states the name of the session, the age range for which the lesson plan is more suitable, and the main subjects covered.



Figure 2 – Main page of each lesson plan

The structure of each lesson plan is as follows:

- **Summary** – A short summary gives an overview of the subjects that are covered by the lesson plan
- **Learning Outcomes** – States what the students are expected to achieve
- **Links with curriculum topics** – Relates the lesson contents plan with curricular topics
- **Notes for teachers** – Gives some remarks and advices for teachers
- **Lesson Plan** – Provides a step by step description of each stage of the session.
- **Resources List & Support Material** – Lists the resources needed to implement the session as well as printing material (when applicable).

Table 2 – TangIn lesson plans

Lesson plans in the TangIn Toolbox		
Lesson Plan	Summary	Subjects
01_Intro Programming	Introduction to Computational Thinking, Programming and Robotics by using commands and role play dynamics. Simulate inputs and outputs and predict outcomes. Give examples of programming and algorithms in everyday life. Age group: all .	<ul style="list-style-type: none"> • Computational Thinking • Robotics & Algorithms
02_Introducing MIGO	MI-GO is a tool that embodies the tangible programming concepts. This session is aimed to the introduction of the story of MI-GO and the explanation on how it is programmed. The first part of the session is dedicated to the customization of MI-GO where students use their imagination and art skills to build a character. Students will also learn that the robot is programmed through the use of blocks and the function of each block. At the end of the session they will be able to execute simple instructions. Age group: all .	<ul style="list-style-type: none"> • Robotics & Algorithms • Arts
03_Animal characteristics	Animal cards scattered around a grid. Using the BOT to travel to them according to specific characteristics Age group: 6-10 years old .	<ul style="list-style-type: none"> • Biology
04_Magic Square	Using the BOT to go to all squares without repeating in a chess board using only the knight movement while doing it building magic squares. Age group: 9-12 years old .	<ul style="list-style-type: none"> • Maths • Operations
05_Maps and Traffic signs	Learn that maps are a representation of a reality on a different scale. Use coordinates to find the correct correspondence between the small and big scale. Identify traffic signs and what they mean. Age group: 6-10 years old .	<ul style="list-style-type: none"> • Geography • Math
06_Minecraft	Using the BOT to mine minerals in the correct sequence according to their properties. Age group: 8-12 years old .	<ul style="list-style-type: none"> • Natural Science • Geology/minerals
07_Multiplication	Multiplication Tables are not only about memory there is logic behind it, and fun also. With the BOT's help we will build them by counting squares (areas). Age group: 7-9 years old .	<ul style="list-style-type: none"> • Maths

Table 2 – TangIn lesson plans

Lesson plans in the TangIn Toolbox		
Lesson Plan	Summary	Subjects
08_Recycling	Learn about the different types of waste and where they should be placed. Plastic, metal, glass and organic waste are valuable resources that must be reused and recycled. With the help of the robot, students sort different types of waste and put it in the correct recycling bin. Age group: 6-10 years old.	<ul style="list-style-type: none"> • Environment • Society
09_Symmetry	Finding the symmetry plan of geometrical figures drawn by MI-GO and divide in halves. Age group: 7-10 years old	<ul style="list-style-type: none"> • Geometry
10_Triangles	Locate and group different triangles in a geometrical figure drawn by MI-GO. Age group: 9-12 years old.	<ul style="list-style-type: none"> • Geometry
11_Patchwork	Areas puzzle. Rotating Tetris like pieces to see where they fit. Same puzzle requires team work of different groups to solve it. Age group: 9-12 years old.	<ul style="list-style-type: none"> • Mathematics • Areas
12_Circulatory System	Drawing the circulatory map, connecting organs while separating venous and arterial blood flows Age group: 7-9 years old.	<ul style="list-style-type: none"> • Biology • Human Body
13_Angles	First introduction to angles, and distinguish between a right angle, acute and obtuse. Age group: 6-10 years old.	<ul style="list-style-type: none"> • Geometry
14_Connecting Dots	Same puzzle to train/introduce loops that increases gradually the level of complexity. Ideal to assess current level of proficiency. Age group: 8-12+ years old.	<ul style="list-style-type: none"> • Programming/loops
15_Space Train	Activity to create circuits and time-tables using the solar system as theme. Age group: 6-10 years old.	<ul style="list-style-type: none"> • Solar system • Logistics (time schedules)
16_Water Cycle	Competitive game with the water cycle as theme. Age group: 6-12 years old.	<ul style="list-style-type: none"> • Environment
17_Words	Scrabble like game using MI-GO to create words for a subject at choice. Age group: 6-12 years old.	<ul style="list-style-type: none"> • Multiple subjects

Table 2 – TangIn lesson plans

Lesson plans in the TangIn Toolbox		
Lesson Plan	Summary	Subjects
18_Constellations	Drawing constellations with MI-GO and using scaled cards/maps to measure angles and lengths and convert to the <i>Set</i> dimensions. Age group: 9-12 years old.	<ul style="list-style-type: none"> • Space • Geometry
19_Calculations	Matching cards with basic arithmetic operations and numbers on the <i>Set</i> as possible solutions Age group: 8-10 years old.	<ul style="list-style-type: none"> • Mathematics • Operations
20_Countries and Flags	Matching flag cards with countries and capitals. Age group: 6-10 years old.	<ul style="list-style-type: none"> • Society • Geography
21_Measuring Units	Using MI-GO to measure the classroom and use the data to create maps at scale. 8-10 years old.	<ul style="list-style-type: none"> • Physics

Tips (first lessons)

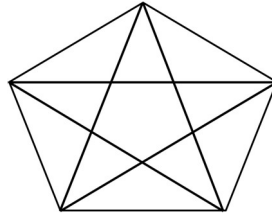
- Start without the *Bot* (see lesson plan 01_Intro Programming). Rationalize in the “others” perspective.
- Introduce *Bot* by storytelling and character building (see lesson plan 02_Introducing MIGO), and let the students to get familiar with it.
- Choose an order of lesson plans but be able to adapt according to results

Tips (for using all the lesson plans)

- Make sure the groups are heterogenous and everyone is active.
- Include other elements of dynamics by assigning different roles to students and switching between them (inside each group).
- Progressive evolution: start with only direction functions, if suited then introduce loops, and finally angles

Dynamics examples

- Using angles does not mean you need to know them
- Example (see lesson plan n° 09/10 – Triangles/Symmetry): while using the *Bot* to draw the figure below, the code is given, the challenge will then be to find as many triangles as possible. The *Bot* here would only be a drawing tool, but even without understanding the code, the concept of angle being different of normal rotation will start to be embedded in them.



Solving different parts of the same puzzle

- Example: (see lesson plan n° 04/06/11 - Magic square/Patchwork/Minecraft) The puzzle is divided into different pieces or tasks and each group will work to solve his part and then come together to share and put the whole puzzle together. Each group has a different task/assignment and in the end, they come together to discuss their findings

Adapt to your own reality

- Example (see lesson plan n° 08 - Recycling): different countries have different colour codes for recycling. Make sure you adapt it to your reality.

Create your own puzzle and switch roles

- Example (see lesson plan n° 05 - Maps and traffic signs): recreate your own map/way to your school. Some might be putting signs other roads other building the code... and make them switch.

Competition

- Examples (most of the lesson plans): try to give points or some kind of reward system for their missions. Does not need to be win or lose but keep them motivated.

3 Resources

3.1 Summary

MI-GO robot kit provided by C&A was chosen as the main resource. It is a 100% tangible platform able to reach younger and older kids with the same tool kit (5-12 years old). Thought not only for young minds but also for teachers, it is extremely versatile and simple to use with only one button and unique features such as the LED feedback system, and loops, decimal and angle programming blocks.

It also presents the advantage of the programming blocks being completely independent of the robots allowing them to be shared, mixed and passed around between different groups and robots as needed, thus leveraging resources and stimulating interaction and group dynamics and hence inclusion.

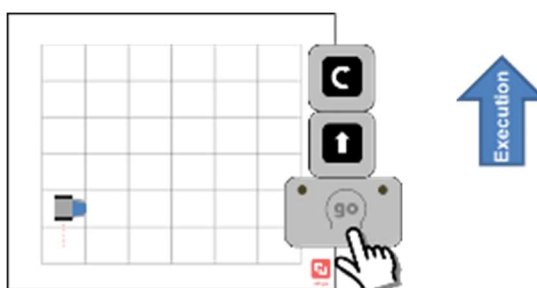
As a tangible programming tool MI-GO kit has two different types of inputs and outputs:

Table 3 – MI-GO inputs and outputs

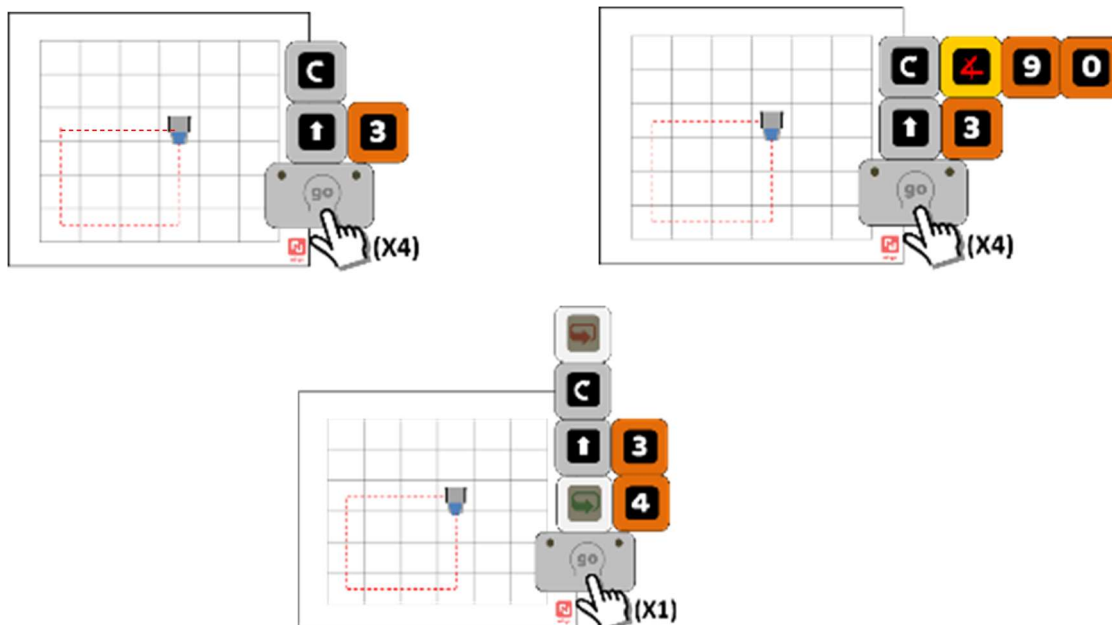
inputs		outputs	
Blocks		Steps (grid)	
Functions (directions & loop)	Numbers & angle (0-9)	Normal (10cm)	Decimal (1 cm)
Colors (sensor)		Drawing (markers)	
Red (STOP)		Permanent (paper)	Temporary (cleanable set)

Inputs

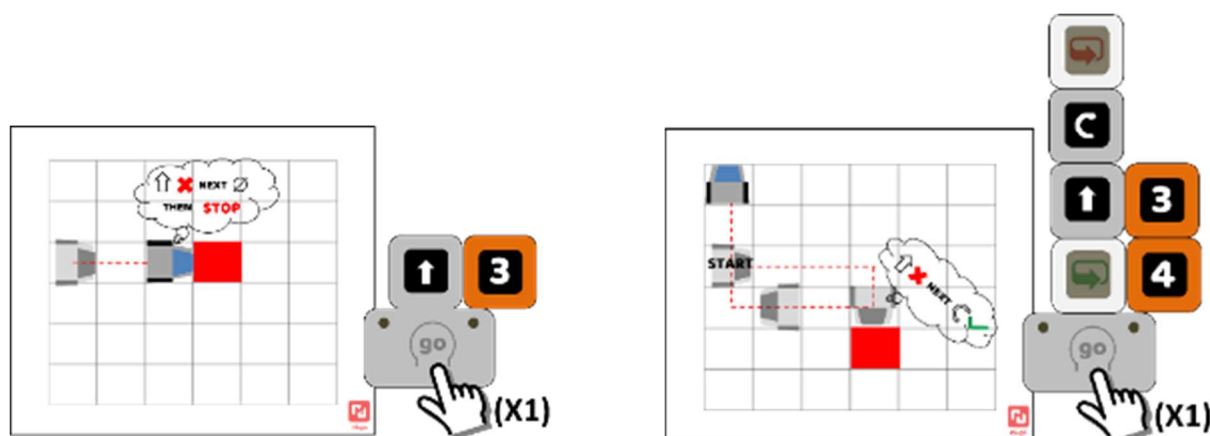
- **Programming blocks:** divided by functions and numbers, those are the basic units to build with as a code of instructions to the robot:
 - Functions (see section 3.2 Kits - Blocks) are logic commands such as Move Forward and Rotate Right and connected starting at the Main (see section 3.2 Kits - Main) from first to last instruction in a sequential way in what will be the main body of the code. Fifteen is the maximum number of functions possible in a single code.



- **Numbers from 0 to 9** (see section 3.2 Kits - Blocks) can be added to give values to the functions and connected to the right-hand side of the intended function, as a single or multiple (max 4) blocks of numbers. In this category it is also included the *Angle* block which can be connected to a Rotate *Right* or *Left* function to change the rotation from the default 90° to an angle of choice by adding number blocks to its right.



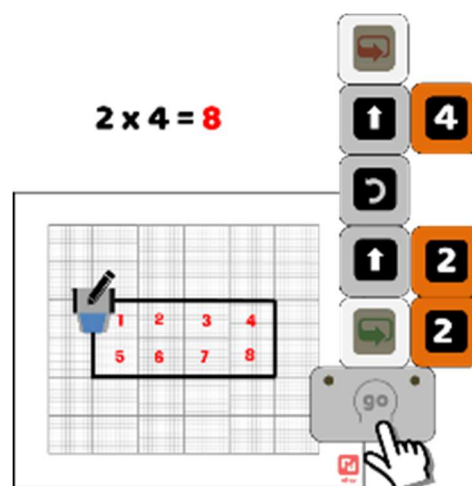
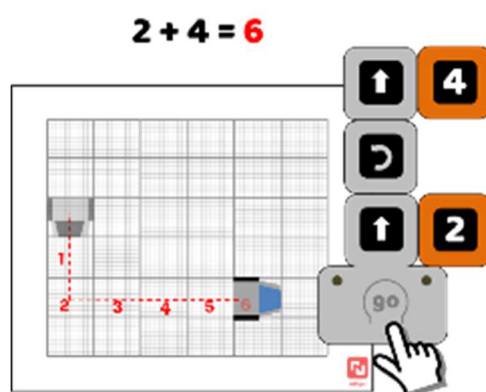
- **Colours** (optional): The *Bot* has an inbuilt colour sensor that can be activated (see section 3.3 Instructions – Colour sensor). This option allows new dynamics with colours acting as *IF -> Then* functions. For the moment only red can be detected as a *STOP* function, meaning when the *Bot* finds red in his way (moving forward) he will skip to the next function until being able to get rid of the “obstacle” or else remain in place. The colours can be either drawn directly in the *Set* or placed as a card on top or underneath.



Outputs

- **Steps:** counting the number and direction of the *Bot* steps is a quantifiable output and with the help of a grid can be used to set a specific trajectory/point as goals, and solving mathematical/logical puzzles with the number of squares covered or contained. It can also be used to measure distances and areas directly on the floor.
- Normal: Steps of 10 cm (larger squares) and its multiples using *Forward* blocks.

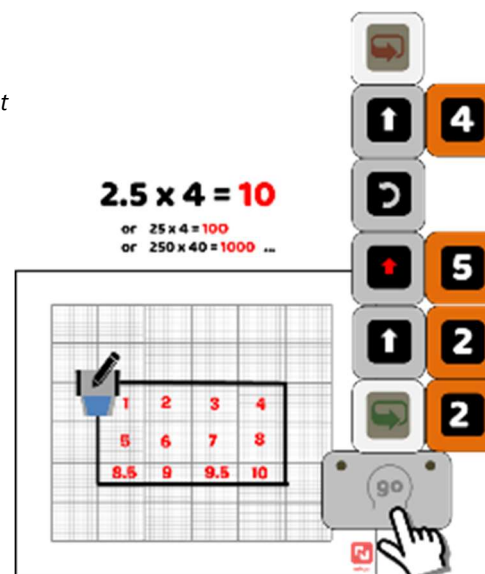
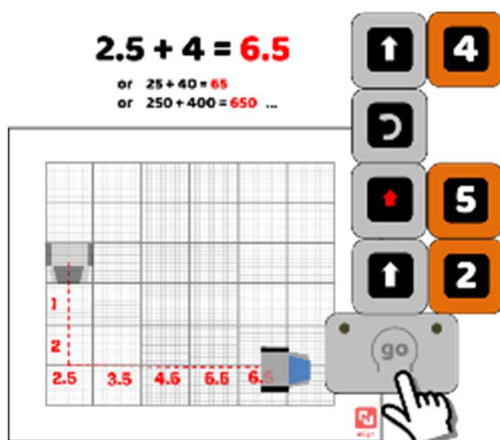
Addition operations by counting squares **covered** by the Bot



Multiplication operations by counting squares **contained** in a perimeter drawn by the Bot (area)

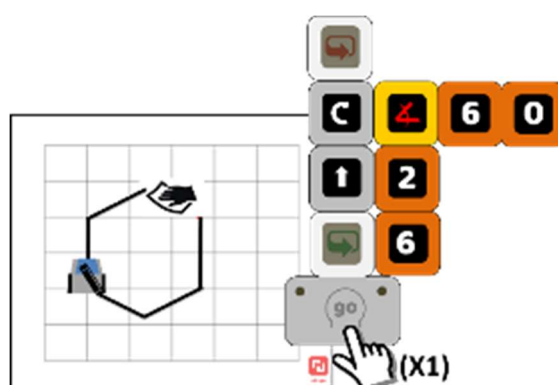
- Decimal: Steps of 1 cm (smaller squares). With the *Decimal* block one is not limited to lengths with whole numbers. But it does not necessarily have to be decimal, it represents a tenth of the normal step and thus can be used as decimal vs unit, unit vs ten, ten vs hundred, etc.

Addition operations by counting squares **covered** by the Bot

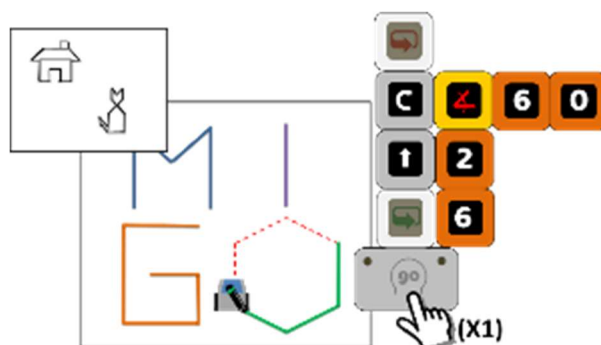


Multiplication operations by counting squares **contained** in a perimeter drawn by the Bot (area)

- **Drawing:** The *Bot* has a cavity that cuts through its axis that can be used to fix the most common commercial markers (15 mm). This capability allows to trace the *Bot*'s route and drawing geometrical or any figure composed of straight lines.
- Temporary: Using the provided *Set* made of a translucent vinyl material, it is possible to remove/erase any trace or drawing made by markers (with permanent ones use alcohol, (see section 3.2 Kits - *Set*). This allows the constant reuse of the *Set*, not only with drawings made by the *Bot* but also by hand to create different activities and games.



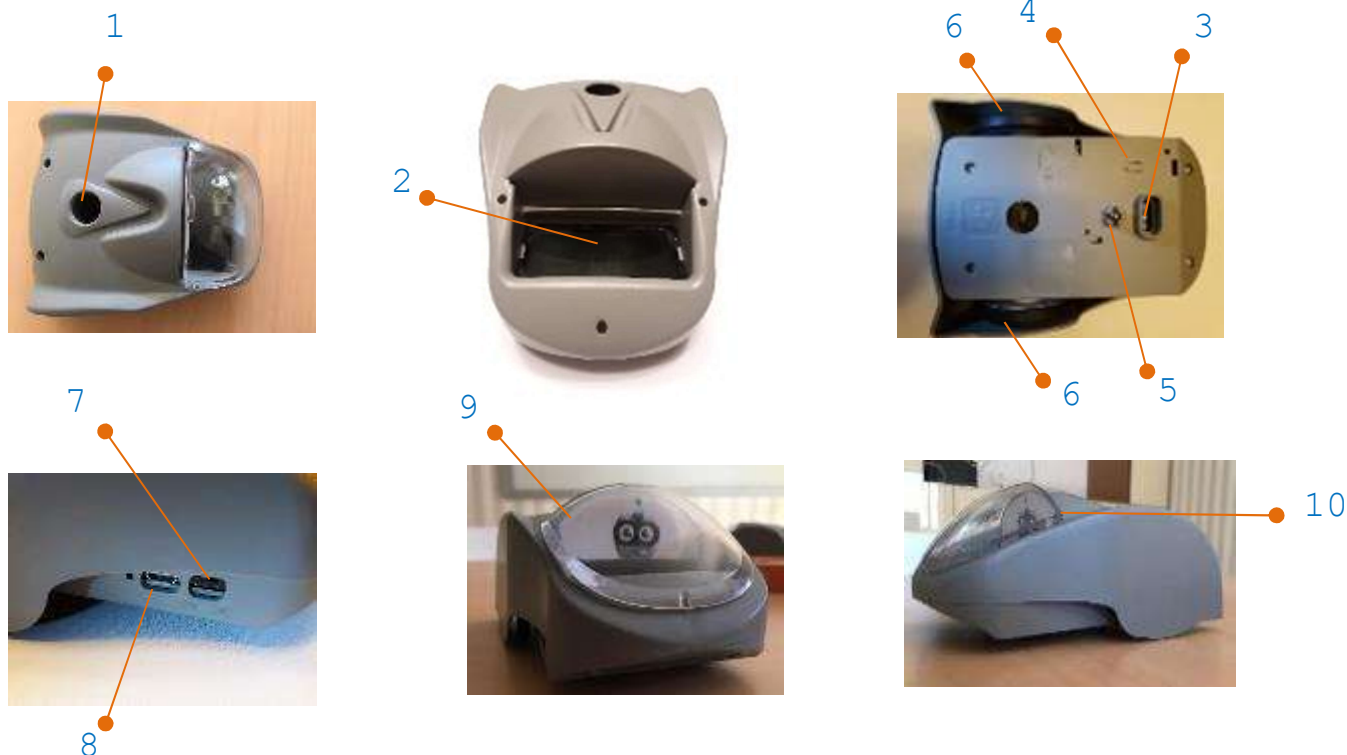
- Permanent: the action is not limited to the *Set*, with color markers and big sheets of paper, one can use MI-GO to create real pieces of art. Creativity is the limit!



3.2 Kits

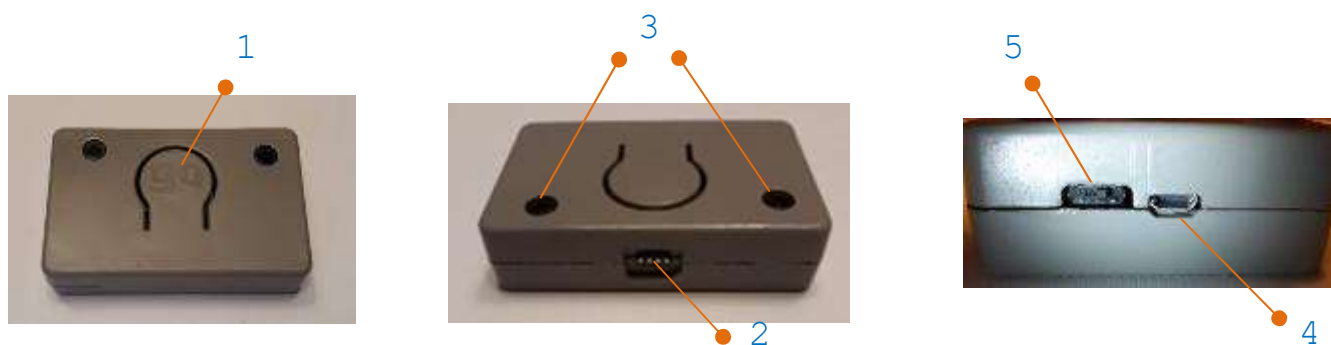
Bot

The *Bot* executes precisely (<1° precision) the instruction from the Blocks (via Main – see below). Components: hole to fix a marker (1); translucent lens allowing PCB view and LEDs feedback (2); colour sensor & white LEDs (3); pair button (4); precision sphere for rotations (5); wheels (6); ON/OFF switch (7); a power mini-USB entry (8); removable cupula (9); cavity to fix robot face card (10).



Main

The main is responsible for blocks reading and power supply (own batteries) and communicating (wireless) instructions and receive feedback from the *Bot* (~5-7 metres range). Components: multi-function GO button (1), one female connector (2) to connect the first block (Function) of the code; Two Lens for LEDs feedback (3); a power mini-USB entry (4) and a ON/OFF switch (5).



Main's top perspective

Main's front perspective

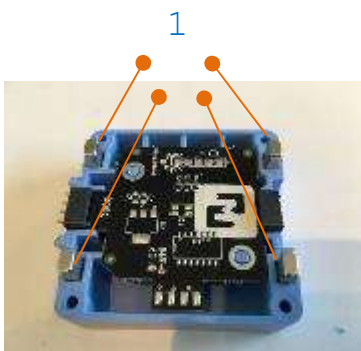
Main's lateral perspective

Blocks

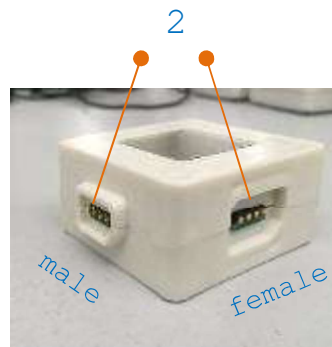
The blocks are single units with specific commands to be combined for the desired code of instructions. Components: 4 magnets (1) to make it easier to connect the blocks in the correct position (and repulse when it's wrong); connectors (2): functions have 3 (2 females & 1 male) with the exception of *End Loop* (1 male and 1 female, in the vertical axis), numbers have 2 in the horizontal axis (1 female and 1 male); Translucid Lens (3) for feedback with LEDs (4 colours).



All blocks except decimal



Block's inside



Block's connectors



Top view lens

Forward and Decimal: both will instruct the *Bot* to move forward one step. If there is a number(s) connected on the right side it will instead instruct that exact number of steps forward. If the number is 0 then it will skip it. The *Forward* step is 10 cm, the *Decimal* step is 1 cm. Connectors: (1) Male, connects with the *Main* or another *Function*; (2) Female, connects with another *function*; (3) Female, connects with a *Number* block (not angle).



Forward block



Decimal block connected to Main

Rotate Right & Left: will instruct the *Bot* to rotate in its axis 90° (by default). If a number is connected to its right (see center image) it will rotate 90° that number of times (if 0 nothing happens), if instead it is an angle block following by number blocks it will instruct the Bot to rotate exactly those exact (numbers) degrees. Connectors: **(1)** Male, connects with the *Main* or another Function; **(2)** Female, connects with another function; **(3)** Female, connects with a Number or angle block.



Rotate left block

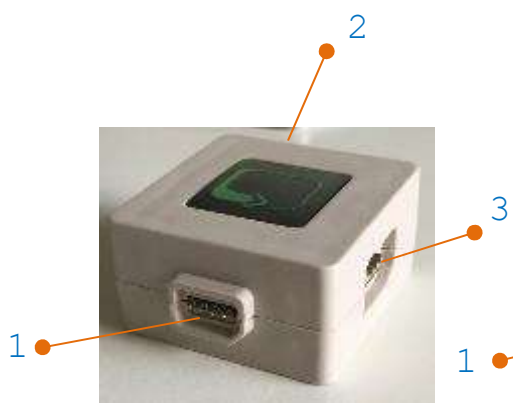


Code with rotate left 2 times 90°

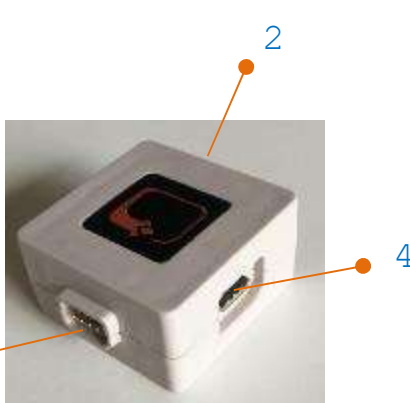


Code with rotate left 95°

Start & End Loop: will instruct the *Bot* to repeat as many times the code between *Start Loop* and *End Loop* as the number connected to *Start Loop* : If it is 1 then It will simple do the code one time (no repetition, if it is 2 then it will do the code 2 times and so on... if there is no number it will also do the code one time, and the same for number is 0 (it will be updated to skip the entire code between the loop blocks for a matter of consistency). Connectors: **(1)** Male, connects with the *Main* or another Function; **(2)** Female, connects with another function; **(3)** Female, connects with a Number block (not angle). **(4)** Female, in this case (*End Loop*) there is supposed to be no block connected (connector is there for a question of mass production economy), if one block is connected the code will not be executed (see error "Syntax" in Table 6)



Start Loop



End Loop



Code with four loops

Numbers: have to be associated/connected to functions, giving a value to them . Any combination of 3 digit numbers in a single function is possible, more than that and the *Bot* will not execute and the *Main* will give an error feedback (see error "Syntax" Table 6). Connectors: **(1)** Male, connects with a Function or another Number (including *Angle*); **(2)** Female, connects with another Number (excluding *Angle*).



Number Two block

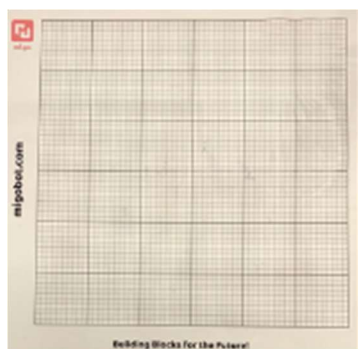
Angle: has to be associated/connected to a *Rotate* (right/left) function on its left and a number on its right or else the *Bot* will not execute and the *Main* will give an error feedback (see error "Syntax" Table 6). It will change the rotation angle of *Rotate* blocks from the default 90° to the number/s connected to it. Connectors: **(1)** Male, connects with a *Rotate* (left/right); **(2)** Female, connects with a Number.



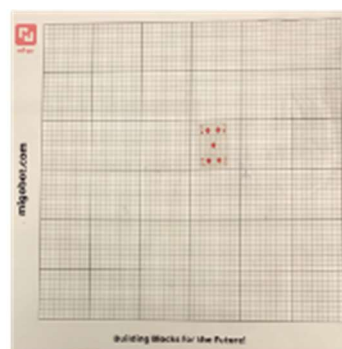
Angle block

Set: 6x6 vinyl grid set. Features: Big squares 100 cm² **(1)**; small squares 1cm² **(2)**; translucent **(3)**; foldable (roller) and reusable*

***Recommended** board markers, to clean use dry cloth/tissue/cotton to wipe. If using permanent markers, you need to soak the tissue/cloth/cotton with alcohol. DO NOT use solvent glues (UHU, tube...). You can use duct tape on the corners but try to avoid it because it will accumulate dirt and erode along the time. To straighten it faster use a hair blower.



Grid Set on a white table



Grid Set with a card underneath

Kits

Because of different frequency of use and simultaneous need (blocks can be shared by different groups) the kits will have relatively more functions than numbers. To start the pilot each partner will have a 3 *Bot* kit (Table 4, green column).

Table 4 – the different kits options and the number of elements in each.

		Kits					
<i>Bot</i>		1	2	3	4	5	6
<i>Main</i>		1	2	3	4	5	6
Blocks	<i>Functions</i>	7	13	22	28	35	41
	<i>Forward</i>	2	4	6	8	10	12
	<i>Decimal</i>	1	1	2	2	3	3
	<i>Right</i>	1	2	4	5	6	7
	<i>Left</i>	1	2	4	5	6	7
	<i>Start Loop</i>	1	2	3	4	5	6
	<i>End Loop</i>	1	2	3	4	5	6
	<i>Numbers</i>	11	15	26	32	43	49
	<i>Angle</i>	1	2	3	4	5	6
	<i>0</i>	1	1	2	2	3	3
	<i>1</i>	1	1	2	2	3	3
	<i>2</i>	1	2	3	4	5	6
	<i>3</i>	1	2	3	4	5	6
	<i>4</i>	1	2	3	4	5	6
	<i>5</i>	1	1	2	3	4	5
<i>6</i>	1	1	2	3	4	5	
<i>7</i>	1	1	2	2	3	3	
<i>8</i>	1	1	2	2	3	3	
<i>9</i>	1	1	2	2	3	3	
<i>Set</i>		1	2	3	4	5	6

3.3 Instructions


Pair

1. Make sure both the *Bot* and the *Main* are turned ON. You will see a white led blinking and then a blue one, on the Bot; and a green Led coming out of the GO button on the *Main*.
2. Press the "go" button on the *Main* for about 3-5 seconds (A). A blue/cyan LED will appear on the *Main* (B) and in all *Bots* (C) that are "listening". Stop pressing the "go" button.
3. You have then 5 seconds to press a small button (D) underneath the *Bot* you want to pair. If it is successful, both the *Bot* and the *Main* will turn a green LED at the same time (E), if not, the *Main* will show a pink LED and go back to previous state (connected to another *Bot* or not paired)
4. You only have to do it once (unless you want to pair another *Bot*), it will remain paired even after switched OFF.

1.



2.

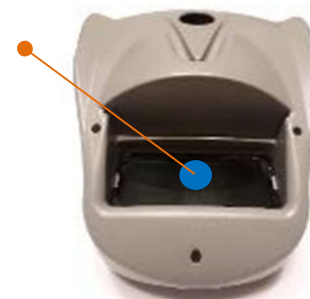
A  (3-5 seconds)



B



C



3.

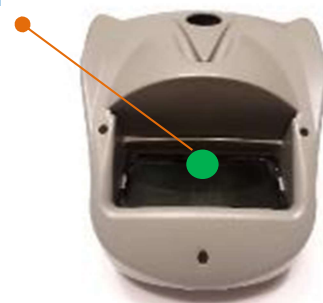
D



E



E



Code assembly, initiate and execution & feedback

1. Assembly the code starting from the *Main* facing GO to yourself. Connect the first function to be executed and progress from there in sequence of consecutive steps.
2. Press GO one time
3. a) the *Main* will read the blocks (each turn blue at a time) and then execute while giving feedback of the step by turning the current function green
- b) the *Main* will read the blocks (each turn blue at a time) and find a syntax error and blink red four times while the blocks until the error also blink.

1.



2.



3a)

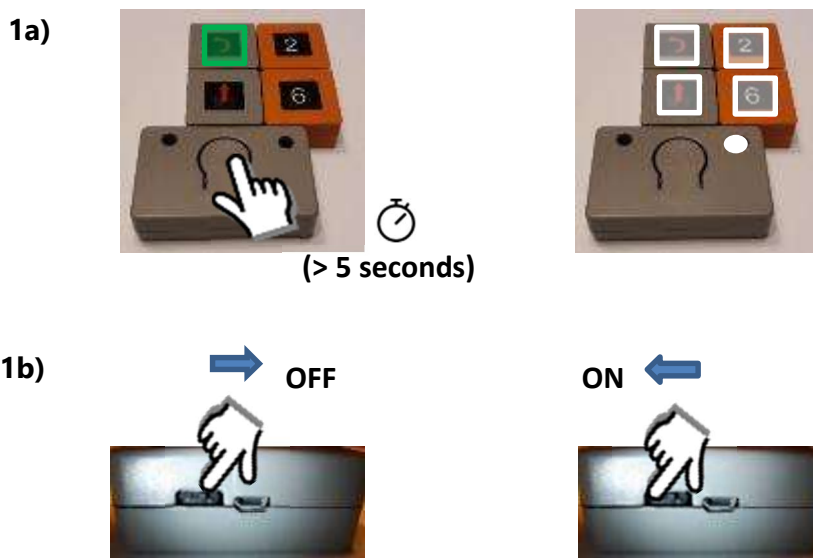


3b)



Reset code (Main): Rarely needed, used to erase the memory of the previous code in case it gets stuck (see error "" in the Table 5). *Bot* and *Main* will still remain paired after it.

1. **a)** Press the GO button for more than 5 seconds until a white LED blinks on the *Main* and blocks.
- b)** Switch OFF (slide with your finger) the *Main* and turn ON again (slide with your finger).



Charging

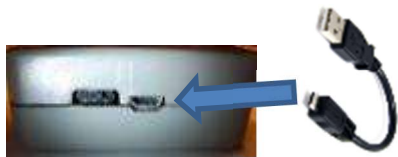
1. **a)** When the *Main* starts missing lights during the feedback it is probably lacking charge. Usually the autonomy is in the range of days (2-4) with constant use. Plug a micro-USB charger and a red LED will turn on (ON or OFF does not matter).

b) When the battery is full it will turn off the red LED, it usually takes less than 2 hours.

2. **a)** When the *Bot* starts accumulating errors like skipping functions and making discontinuous steps (e.g. 1+1 instead of 2), it is most likely in need of charging. Plug a micro-USB charger and a red LED will turn on (ON or OFF does not matter).


b) When it is full it will turn off the LED, it usually takes between 1-3 hours. However, the *Bot* can become unstable (see troubleshooting and error "Too much charge" in Table 6) with too much charge so we recommend leaving it only 45 minutes which is sufficient for a 1-2 days of constant use.

1. a)

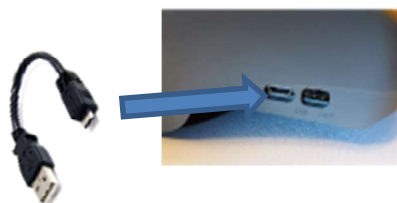


b)





(< 2 hours)

2. a)



b)




(45minutes – 1 hour)

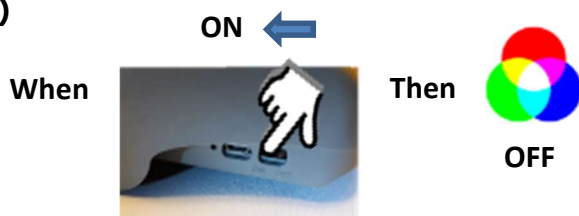
Colour Sensor (Turn OFF/ON is the same procedure)

1. a) When *Bot* switches ON the colour sensor is OFF by default.

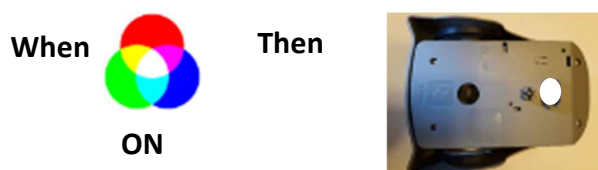
b) When the colour sensor is ON, the *Bot* will have white LEDs on its bottom (near the sensor).

2. To change the status of the sensor (ON/OFF), press and hold the PAIR button for one second, a yellow LED will appear in the *Bot* and then change to blue to confirm the change of status.

1. a)



b)



2.



Feedback System

As described, there is a Feedback System of Colours (with LEDs) for all instructions (Table 5). It is a great feature for the kids (and not only) to do code debugging (watching the action of the *Bot* and matching with the Blocks) but also helpful to see if the components are performing as expected and which mode/s are currently ON or OFF

Table 5 – Feedback map for instructions.

Feedback System - Instructions Map								
Goal	Action	Duration	LEDs					
			Main		Bot			Blocks
			Left	Right	Bottom	Top	Side	
Turn ON Bot & Main	ON switch (side)	-	-	-	-	-	-	-
Pairing	press & hold GO button (Main)	3-5 seconds	-	-	-	-	-	-
	press PAIR button (Bot bottom)	before 5 seconds	-	-	-	-	-	-
Initiate code	press GO (Main)	one time	-	-	-	-	-	-
Following execution (debugging)	-	-	-	-	-	-	-	-
Reset	press & hold GO button (Main)	> 5 seconds	-	-	-	-	-	-
Charging Bot & Main	micro usb cable on power entry (side)	15 minutes	-	-	-	-	-	-
Turn ON/OFF color sensor	press & hold PAIR button (Bot)	1 second	-	-	-	-	-	-
Detecting color	-	-	-	-	-	-	-	-

3.4 Troubleshooting

Table 6 – Feedback map for errors.

Feedback System - Troubleshooting Map							
Error	LEDs						Action
	Main		Bot			Blocks	
	Left	Right	Bottom	Top	Side		
Syntax	-	-	-	-	-	-	Code debugging
Too much charge (LED disappears)	-	-	-	-	X	-	If many errors, Turn OFF/ON Bot and let it roll for a while
Feedback but Bot does not move	-	-	-	Stop	-	-	Turn OFF/ON Bot
Feedback and Bot frozen	-	-	-	Stop	-	Stop	Turn OFF/ON Main
Skipping function	-	-	-	-	-	-	Turn OFF/ON Bot or charge
GO with no Pair or OFF Bot	-	-	-	-	-	-	Pair with Bot or check if ON
Bot unstable & or discontinuous steps	-	-	-	shaky	-	-	Charge Bot
No Main feedback	-	-	-	-	-	-	Charge Main

Q & A

If the batteries are low, what happens?

Main: starts missing LEDs during the feedback. It is possible to freeze in a function during execution.

Bot: keeps accumulating errors like skipping functions and making discontinuous steps (e.g. 1+1 instead of 2)

How much time should I charge them?

One hour each is more than enough. Too much charge may cause the *Bot* to be unstable in the first hour. If so, let it go for a few spins, by making it rotate for 120 times and leave it ON to rest a little bit.

How long does the Bot and Main last without charge?

The autonomy is in the range of days, depending on how many executions, if very busy (> 50 execs) it will last 2-3 days before losing performance.

How do I know the Main is turned ON?

Besides the ON/OFF switch position (see image *Reset 1b* in the *Instructions* section), you will see a green light coming under the GO button.

Can I use any block and Main for any Bot?

Yes! If you pair the *Main* and the *Bot* you want, all blocks available are up for grabs.

After switching OFF Main &/OR the Bot, will they still be paired after switched back ON?

Always. Unless you find another Pair for it. Like relationships...

If I do not remember which Bot is connected to which Main what should I do?

No worries it happens to everyone. Stay calm and press GO (and see which one is your friend).

What is the maximum number of blocks I can execute in a single code?

The number of functions is limited to 15, and then you can add as many numbers you want. Well not really, you are limited to a maximum of 3 numbers digits, and assuming all the functions can have numbers, the answer would then be 45. Well not really, if you would be willing to build a code only with *Rotate* functions, then you could also have an angle block + 3 numbers, so the final number would be 60, and extremely fun to watch (please send video).

How do I know one block is defective/malfunctioning?

When you first connect and also while the Main is reading the code, you will see all the blocks responding with a light (see image *3a-b Assembly Code and initiate...* in *Instructions* section). This means energy and information is passing through it. If so, try to connect again.

How many combinations of loops is it possible to make?

If we are talking about the number of repetitions one *loop* function can make then the limitation is only the maximum number allowed in general (3 digits, so the precise answer is 999). If we are talking about multiple *Loop* functions, one after the other, the limitation is then the number of *Loop* blocks you have in your arsenal. If we are talking about loops inside/within loops then the maximum is 2!

I press again the GO button before the code is finished, what happens?

Nothing really.... both *Main* and *Bot* will ignore you 😞

If I take a block from the code after it started, what happens?

If it is still reading (green light on the *Main*) then it will give a red/purple error light and will reset. If after executing that step, then nothing happens. If it is before execution... then it depends on who's faster to get to it, you or the *Bot*. An old fashion duel, good luck Lucky Luck!

I pressed GO with a code that I do not want to finish, what should I do?

Two options: you hold the *Bot* and switch OFF, disadvantage – you must chase and reach it; you turn OFF the *Main*, disadvantage – it will only stop when the current function (already read) is finished. If in a live-or-die decision (like going to fall from a table), go for the first option...

I pressed GO but there is no Bot paired or it is turned OFF...

The *Main* will wait a couple of seconds for an answer, if none, will give a red-light error (see Table 6) and reset the code. Nobody likes to be ignored.

Some blocks were not well connected, will the Bot explode?

Calm down soldier, it will probably retrieve a red signal error on the *Main* and nothing else happens. Make it right and try it again.

If I press GO simultaneously in different Main's what happens?

Excellent question Mr Random Person! And also, a tricky one... Even if every *Main* generates a different channel for its messages, the chances of something going wrong increases because of interference. One *Bot* might be not able to receive the message while other skips a function during the execution. Interference in general might happen occasionally, and the best strategy to avoid it is to keep some distance from *Main* to *Main*.

How do I know the Colour Sensor is turned ON?

You have not been paying attention to the rest of the document (see Colour Sensor, Instructions section) have you Mr. Random? No problem, always better to ask. The default state of the sensor is OFF, meaning every time you switch OFF your *Bot*, he goes back to the default state and so the sensor will be OFF when you turn the *Bot* ON again. One way to be sure is to check if the white LEDs on the bottom of the *Bot* are turned ON. Pay attention, if it is turned ON and you are not aware, the *Bot* might start to show some strange behaviours.

If the Blocks are giving feedback during execution but the Bot is not moving, what should I do?

It is definitely an error! Most likely might be either too much or too less battery, that causes the *Bot* to be unstable. Like computers, turn OFF turn ON (the *Bot*) is always the answer. Unless you have the colour sensor unintentionally ON...

If the Bot stops and the feedback freezes in one block?

Something wrong happened, and you must reset the *Main*. Turn OFF turn ON (*Main*) is still the answer.

If the Bot skips one function during execution, what should I do?

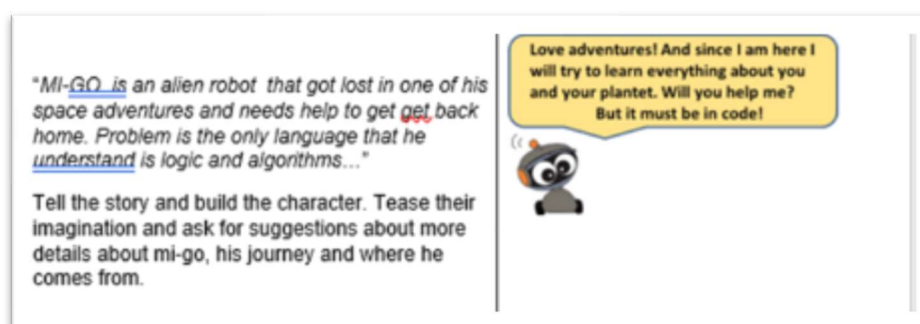
Not much, besides trying again. If persists, try turn the *Bot* OFF and ON. If it does not get better than most likely it is either *Main's* or *Bot's* time to refill.

Are there safety concerns?

No. Voltage & radio frequency are tested and certified and all the materials are children-safe. Batteries and electronics are concealed (connectors are the exception but are well contained and not harmful) and resistant to shock. No small pieces to swallow. And the blocks are not heavy enough to make substantial damage (but avoid throwing it). For durability try to avoid too much elements exposure (rain, sun...) and water in general.

3.5 Useful tips & Code examples and geometries**Tips**

- 1) **Storytelling** and **character building** are important for the children to relate and empathize with the Bot, and in consequence motivate them to help it complete its mission.



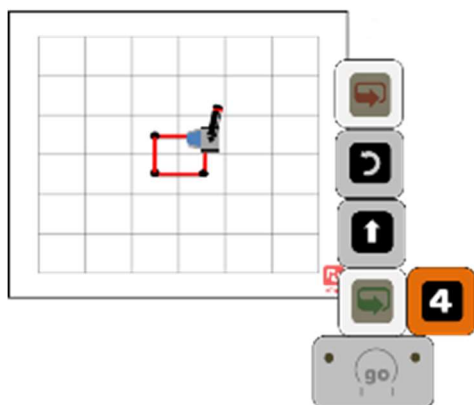
- 2) Promote **sharing** of blocks and solutions between groups. Tangible Programming and MI-GO in particular is all about social skills (negotiation, compromise, team-work, problem solving...)
- 3) *Bot* executes the functions **continuously** (e.g.: *forward: two times* at once instead of 1+1). If you want to do it **discretely** use loops (e.g.: *loop two times: forward*)
- 4) Explore the **decimal** block it gives you unlimited potential for any unit of irregular shape
- 5) Try to **evolve** the groups along time, start with basic functions and then introduce loops and eventually angles according to the age group.
- 6) Less is more! Find **solutions** for your **limited blocks** problems. E.g. *Rotate Right = 3x Rotate Left; Forward = Decimal x 10*
- 7) If not, **enough Bots** adapt activities and share tasks
- 8) **Customize** and adapt the **Set**. Draw, use cards (on top and underneath), build obstacles, imagination is the limit!

- Use any marker you want but, non-permanent board marks are more convenient to clean. The size is commercially standard but if using *eddying* markers you will have to cut a little salient (**A**) close to the tip to let it fit the hole.

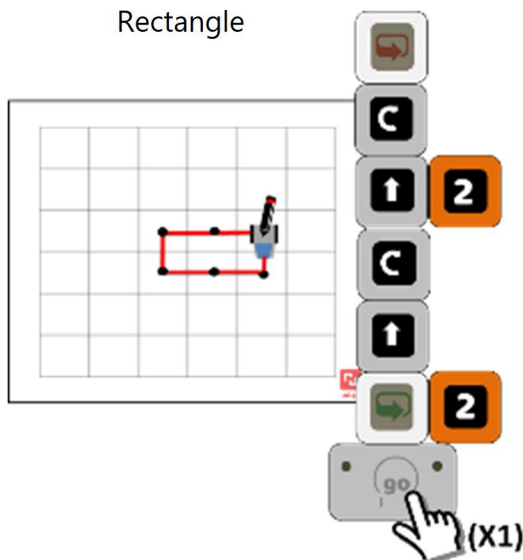


Examples

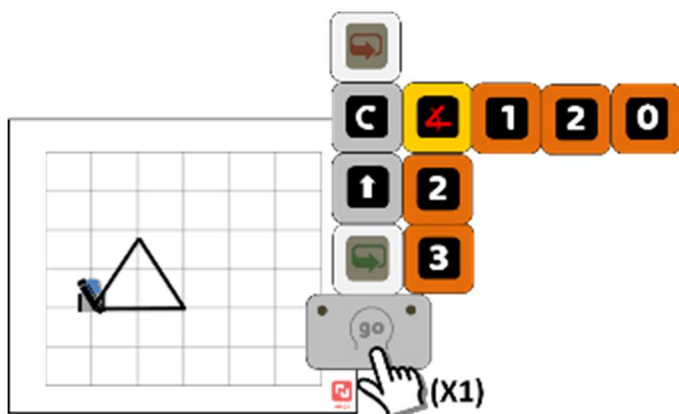
Square



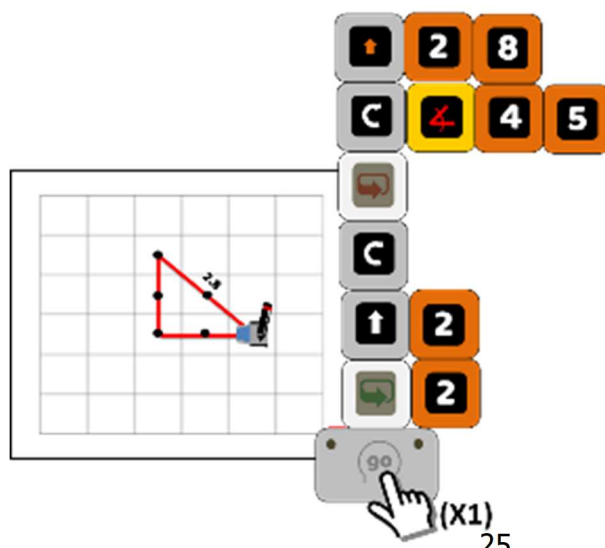
Rectangle



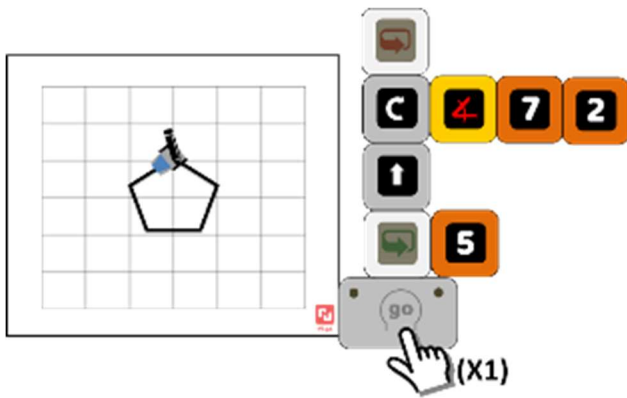
Equilateral Triangle



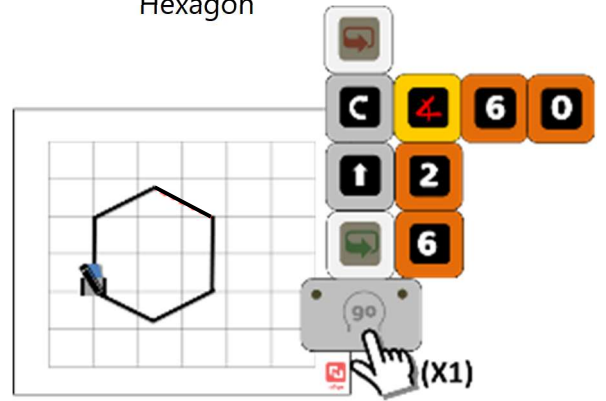
Right Triangle



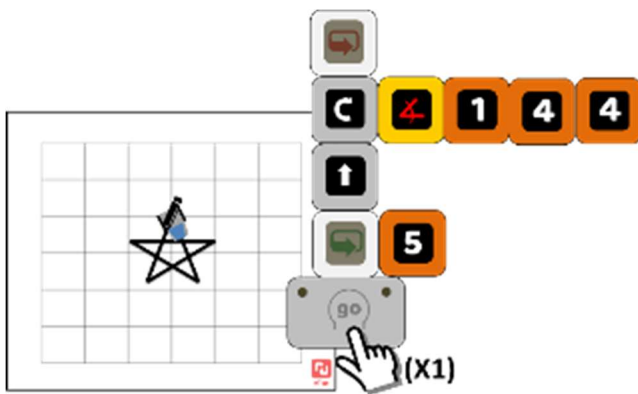
Pentagon



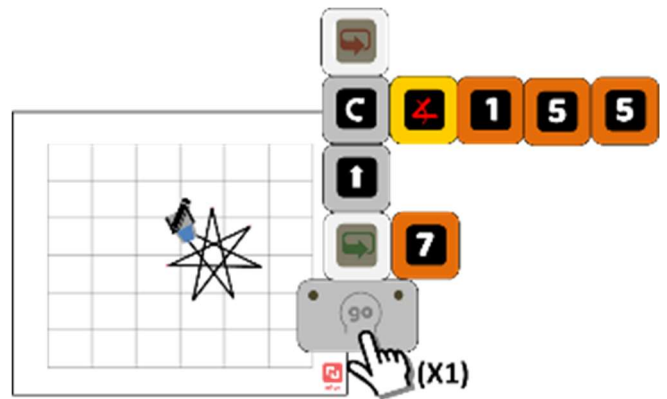
Hexagon



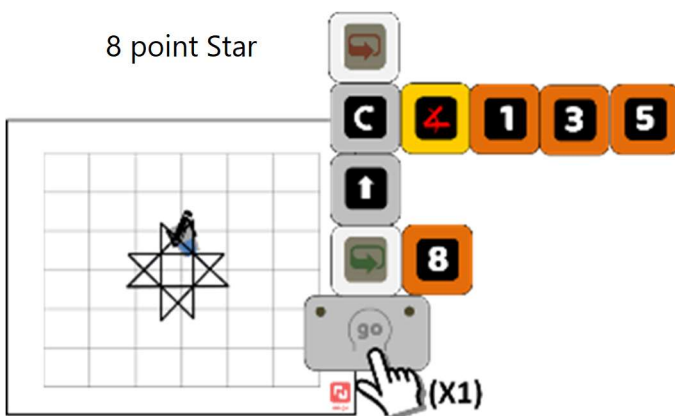
5 point Star



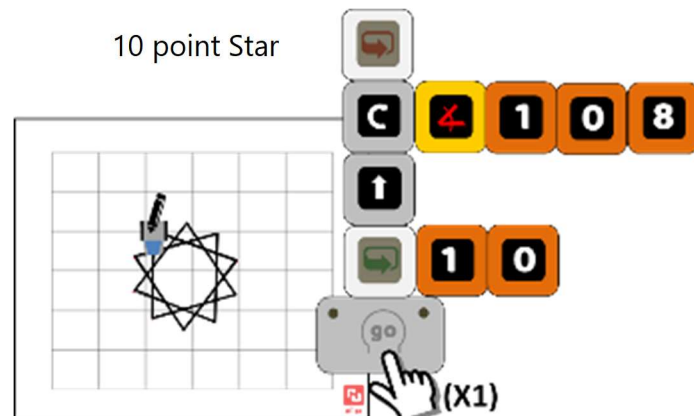
7 point Star



8 point Star



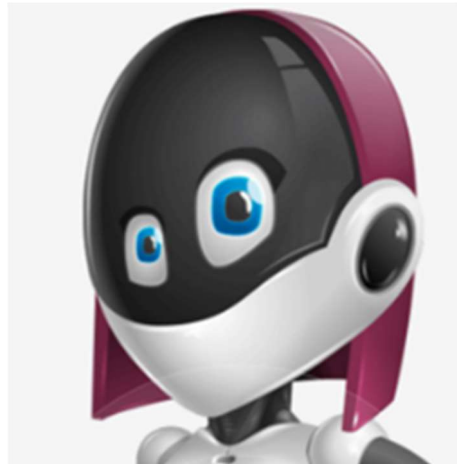
10 point Star



4 Annexes

4.1 Robot faces





TangIn

Tangible Programming & Inclusion



www.tangin.eu



This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein. Project N°.: 2017-1-PT01-KA201-035975