

Bachelor's thesis

Information and Communications Technology

2022

Andrei Skorik

IMPLEMENTATION OF BLOCK- BASED PROGRAMMING FOR AN EDUCATIONAL DEVICE



Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2022 | 63 pages

Andrei Skorik

IMPLEMENTATION OF BLOCK-BASED PROGRAMMING FOR AN EDUCATIONAL DEVICE

One of the noticeable challenges that appear in the adaptation of a programmable device for educational purposes is the necessity to decrease the requirements for basic programming skills of an end-user. This goal can be successfully reached by the implementation of block-based programming language, which substitutes the coding process with a much simpler approach: visual blocks instead of text lines.

This Bachelor's thesis covers the process of implementing a block-based programming environment for the multipurpose portative interactive sensor platform developed by AI2AI. The main goal was achieved by means of building a MicroPython port to the Zephyr real-time operating system, adapting the BIPES integrated development environment according to the company's requirements, and creating the set of Blockly-based blocks that can utilise application programming interfaces created for the device.

The device with implemented block-based programming integrated development environment was successfully used as an educational tool during demonstration classes for pupils of different ages in Turku and Helsinki in May 2022.

Keywords:

Block-based programming, Blockly, IoT, software development, visual programming.

Content

List of abbreviations	5
1 Introduction	7
2 Device Specifications	10
2.1 Hardware	10
2.2 Zephyr RTOS	10
3 STEM and STEAM Educational Approaches	12
3.1 STEM/STEAM Activities	12
3.2 STEAM educational program influence on students' grades	13
3.3 Programmable devices' opportunities	13
4 Overview of potential competitors	14
4.1 BBC micro:bit	14
4.2 Sphero Robots	15
5 Block-based Programming Languages	18
5.1 Scratch	18
5.2 Blockly	19
6 MicroPython	22
6.1 Zephyr Port	22
6.2 APIs	23
7 BIPES	24
7.1 BIPES UI	24
7.1.1 Blocks	24
7.1.2 Console	25
7.1.3 Files	26
7.2 New Blocks	26
8 Design of New Blocks for Pall0	28
8.1 Block Definition	29

8.1.1 Input	29
8.1.2 Field	37
8.1.3 Type	38
8.1.4 Colour	41
8.1.5 Additional features	44
8.2 Generator Stub	47
8.2.1 Variables	47
8.3 XML file for BIPES toolbox directory	49
8.3.1 Shadow of another block	50
9 BIPES modification for Pall0	52
9.1 Build or update software version	52
9.2 UI Modification	52
10 Demonstration. Hot Potato Game	54
11 Conclusion	57
11.1 Results	57
11.2 Possible Improvements	59
References	60

List of abbreviations

Abbreviation	Explanation of abbreviation
API	Application Programming Interface
BIPES	Block-based Integrated Platform for Embedded Systems
CPU	Central Processor Unit
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HSV	Hue Saturation Value colour scale
IDE	Integrated Development Environment
IoT	Internet of Things
LED	Light-Emitting Diode
MPY	MicroPython
OS	Operating System
POSIX	Portable Operating System Interface
REPL	Read–Eval–Print Loop
RGB	Red, Green, and Blue primary colours
RTOS	Real-Time Operating System
SoC	System on a Chip
STEM	Science, Technology, Engineering, and Mathematics
STEAM	Science, Technology, Engineering, Arts, and Mathematics

UI

User Interface

XML

Extensible Markup Language

1 Introduction

Digital technologies change modern society and shape the modern world. Without doubts, digital technologies affect the educational system. The size of the global market for programmable robots for STEAM education was estimated at \$496.27 million in 2021, \$574.29 million in 2022, and will reach \$1,207.56 million by 2027 [1]. Teachers and students benefit from understanding how digital devices and coding affect the surrounding world. This situation creates a significant demand for different programmable devices that might be applied for educational purposes and, thus, provides great opportunities for developer companies.

AI2AI is a Finnish IT company which aim is to create a device that makes it possible to interact with the surrounding world without GUI. The latest company's prototype of such a device is Pall0 (**Figure 1**) a portative interactive sensor platform - with a focus on motivating the user [2]. It aims to help end-user to develop, test, and run their own applications.



Figure 1. Pall0

Compact size, variety of sensors, buttons, Bluetooth mesh, RGB LEDs, and Zephyr RTOS make it possible to apply Pall0 for different purposes. Pall0 was successfully tested as a well-being device, game controller, or medical

diagnostics tool. One of the potential niches for Pall0 is education, especially the STEM/STEAM educational approach.

AI2AI had a strong belief that Pall0 can be used for multiple STEM/STEAM activities at school or at home. On the other hand, the company realised that potentially the strongest competitors on the market implemented block-based programming IDE, which made their devices more user-friendly and significantly simplified acknowledgment with programming for children.

Thus, the need to implement a block-based programming environment for STEM/STEAM activities became one of the main goals for the company.

The goal of this thesis is to implement block-based IDE for the existing prototype of Pall0 that includes:

- evaluation of potential competitor devices;
- evaluation of block-based programming languages;
- implementation of required programming language on top of Zephyr RTOS;
- implementation of required IDE for the device;
- creating programming blocks for the device's APIs.

The thesis is structured as follows:

Chapter 1 introduces the goal and objectives of the thesis.

Chapter 2 provides the device's specifications.

Chapter 3 introduces brief information about STEM and STEAM educational approaches and the possibilities they provide for programmable devices.

Chapter 4 reviews the possible competitive devices.

Chapter 5 provides brief information about two block-based programming languages: Scratch and Blockly.

Chapter 6 introduces MicroPython and its implementation for Zephyr RTOS.

Chapter 7 reviews BIPES UI and BIPES requirements for new blocks.

Chapter 8 describes the process of creating new blocks for Pall0.

Chapter 9 introduces brief information about modifications of the BIPES UI that were made for Pall0.

Chapter 10 provides an example of a game that was created for Pall0 using implemented block-based programming IDE.

Chapter 11 summarises the results as well as possible improvements.

2 Device Specifications

2.1 Hardware

Pall0 is based on Nordic Semiconductor nRF5340 SoC. **Figure 2** displays Pall0's specifications.



- Bluetooth Mesh
- Accelerometer, gyroscope
- Magnetometer
- 10 Touch buttons
- 10 RGB LEDs
- Air pressure and temperature
- Ambient colour
- Stereo microphones
- Stereo audio playback
- Vibration
- Magnetic field

Figure 2. Pall0 in details [3]

2.2 Zephyr RTOS

Management of Pall0 is under the control of Zephyr RTOS created for embedded and resource-constrained devices. Zephyr is developed to meet the following list of requirements:

- Independence from CPU architecture;
- Low system requirements (10 kB is enough to run the OS);
- High level of security;

- Connectivity (support for different types of wired and wireless technologies);
- Implementation of effective development tools;
- Open-source code. [4]

Zephyr provides a wide range of essential features:

- Multithreading for cooperative, priority-based, non-preemptive, and preemptive threads;
- Interrupt services at compile time and runtime;
- Memory allocation services;
- Inter-thread synchronisation services (e.g., binary, counting, and mutex semaphores);
- Inter-thread data passing services for message queues and byte streams;
- Power management services (e.g., tickless idle and an advanced idling infrastructure);
- Support for multiple thread scheduling choices (e.g., cooperative and preemptive scheduling, earliest deadline first (EDF), multiple queuing strategies, etc.);
- Memory protection for memory-constrained devices;
- Configurability and flexibility;
- Native POSIX port, which supports running Zephyr as a Linux application. [5]

3 STEM and STEAM Educational Approaches

STEM and STEAM are two educational approaches. Both approaches use Science, Technology, Engineering, and Mathematics as access points for the educational process. In addition, the STEAM approach adds Arts as a very important and effective part of classroom and at-home learning.

3.1 STEM/STEAM Activities

The significant difference between STEM and STEAM approaches can be shown using for example the catapult activity, which is widely used by students as a part of STEM education.

Standard STEM catapult activity is creating a small catapult from rubber bands and different kinds of sticks to fling small-sized objects (e.g., marshmallows) to hit a target. In this activity students should utilise serious STEM skills:

- Science: knowledge of energy's ability to create motion;
- Technology: skills to combine different materials and to develop and test prototypes;
- Engineering: the creation of catapult design;
- Mathematics: calculations and measurements of length and mass.

To create a STEAM activity, Art elements should be added. One of the generic approaches for the catapult activity is students' acknowledgement of Jackson Pollock's arts and the methods he created. As a result, the new goal for students is not just a competition of hitting a target with their catapults, but a creation of Pollock-style art by throwing small objects covered with watercolours.

It's worth admitting, that just the addition of watercolours does not make this a STEAM activity, but the emphasis that students can create their Pollock-style arts does. The process of building a catapult turns into the creation of the instrument

children use to make their own art. In this case, it is a STEAM activity that still helps students to gain serious STEM skills. Moreover, art's elements help children to engage creatively, critically, and confidently in their learning [6].

Another example of STEAM activities is Liquid Xylophones. Children fill glasses with different liquids (e.g., syrup, ketchup, oil, or water) and get an opportunity to obtain knowledge of connections between changes in pitch and the density of a liquid creating their own music [7].

3.2 STEAM educational program influence on students' grades

Children understand lessons better if lessons utilise STEAM, and, moreover, STEAM activities can help to increase students' grades e.g., compared to students from control groups. For example, participants of the STEAM program at the Phoenix Symphony in Arizona average annual score:

- 17 points higher (science questions);
- 13 points higher (mathematics questions);
- 10 points higher (language and arts questions);
- 13 points higher (all subjects combined) [8].

3.3 Programmable devices' opportunities

Increasing STEM skills throughout the STEAM approach provides a good opportunity for programmable devices to gain their own niche in school education. Nowadays different schools make use of a number of different programmable platforms for educational purposes (e.g., Arduino, Raspberry Pi, BBC micro:bit). Sphero positions its robots as "number one" tools for STEAM learning [9].

4 Overview of potential competitors

To clarify Pall0's feasible position on the market of programmable platforms for education the evaluation of potential competitors was performed with the focus on BBC micro:bit and Sphero robots.

4.1 BBC micro:bit

Micro:bit is an open-source embedded system designed by the BBC for use primarily in education. For the moment it is one of the most popular educational programmable devices in the world since there are approximately over four million micro:bits in over 60 countries [10].

The popularity of this device is caused by its cheap price (moreover, a significant number of micro:bits were donated for schools [11]), inbuilt LEDs and sensors, and the possibility of using different languages (Scratch, MakeCode, Python) to program device.

Figure 3 features the main specification of BBC micro:bit.

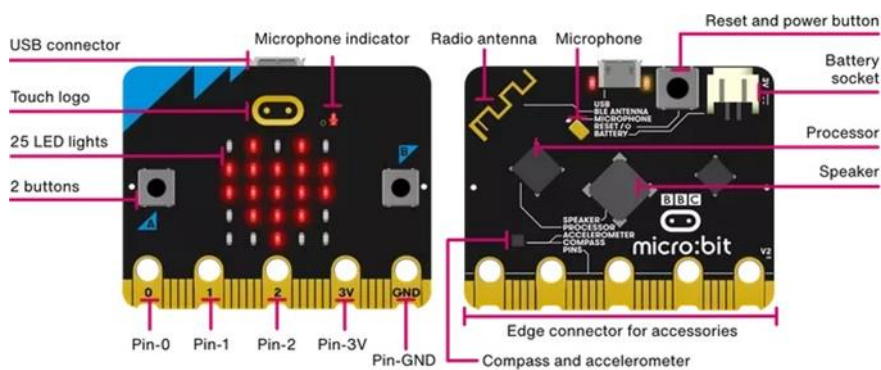


Figure 3. BBC micro:bit [12].

Inbuilt sensors provide an opportunity to program them to affect inbuilt LEDs or speaker, which might be a part of STEM/STEAM activity for students (e.g., create a program that may distort sound if the device is tilted). **Figure 4** displays basic Scratch code for such kind of activity.



Figure 4. Scratch code for BBC micro:bit [13].

4.2 Sphero Robots

Sphero is an American company that creates a line-up of spherical programmable robots. The company positions itself as the industry leader in edtech programmable robots and claims that millions of students use their robots all over the world at more than 20,000 educational institutions [14]. For the moment, the most advanced Sphero robot is BOLT. **Figure 5** depicts the main specification of Sphero BOLT.

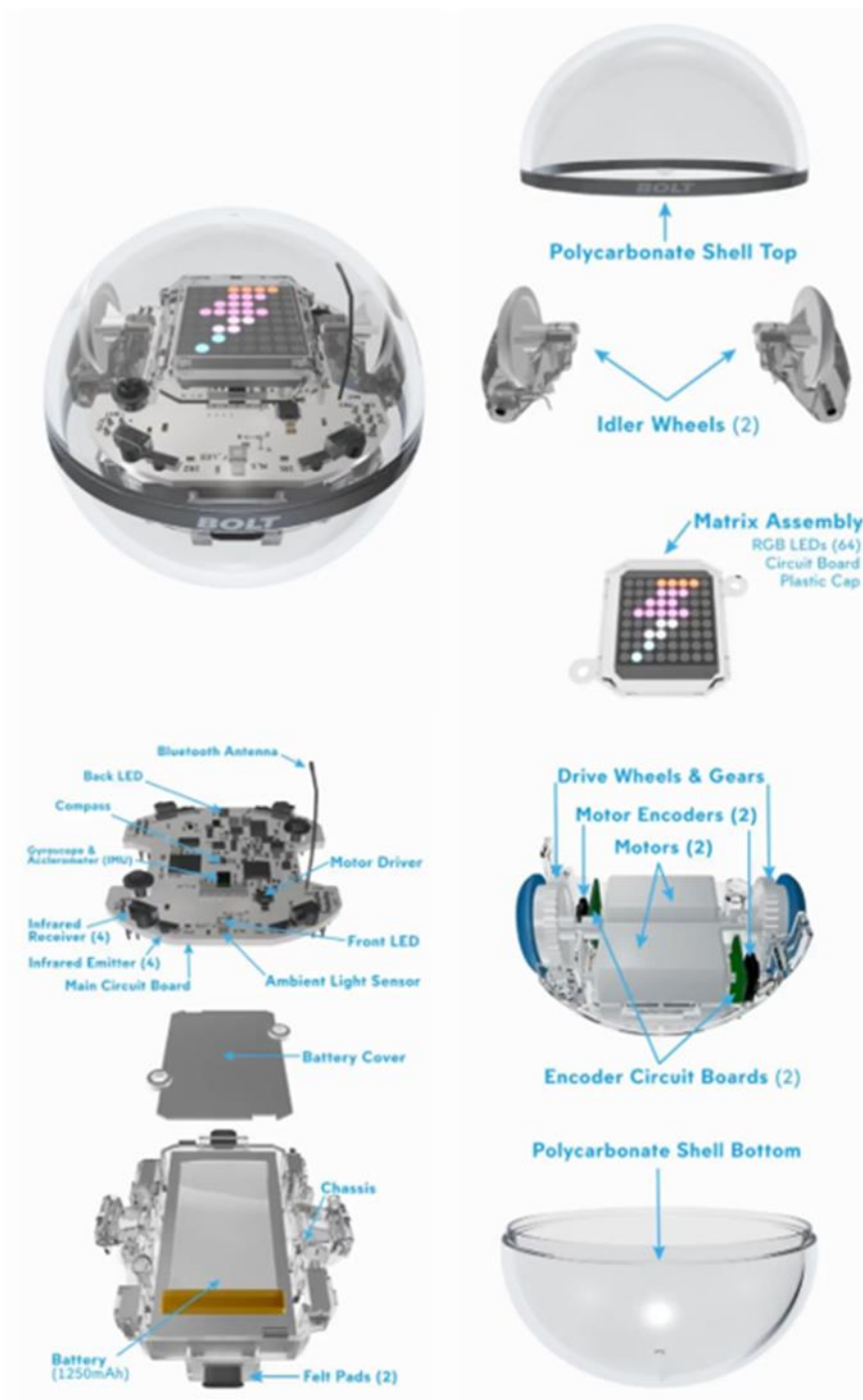


Figure 5. Sphero BOLT (composed from screenshots of the 3D model featured in the Sphero Edu App) [15].

BOLT was developed as a computer science educational tool for PK-12 students, which can be utilised as a part of the STEAM approach. It is controlled via an application installed on a smartphone or tablet. The main feature of BOLT, as well

as of all Sphero's robots, is the ability to roll around, but the LED matrix and variety of sensors significantly increase the robot's functional abilities. Students can program new functions for BOLT using Scratch language.

As an example of a STEAM activity for BOLT can be mentioned the possibility to use the robot as a substitution for a brush for painting. Covered with colour BOLT can roll around a canvas or paper under the control of a student creating paintings [16].

5 Block-based Programming Languages

Block-based coding converts text-based programming into a drag-and-drop process where lines of text code are represented as visual blocks that can be combined to create a working code. For this work two most popular programming environments that use a graphical interface of interlocking blocks were examined: Scratch and Blockly.

5.1 Scratch

Scratch is a block-based visual programming language, which was created by MIT almost 20 years ago in 2003. Creators of Scratch claim that it is “the world’s largest coding community for children” [17] with over 95 million registered users [18]. The latest third version of Scratch is based on HTML5 and JavaScript [19]. Both devices (BBC micro:bit and Sphero Bolt) that were examined as potential Pall0’s competitors implement Scratch as a programming tool.

Micro:bit uses Scratch as one of the possible programming options along with MakeCode and Python [20]. **Figure 6** shows Scratch GUI with micro:bit extension and small micro:bit project.

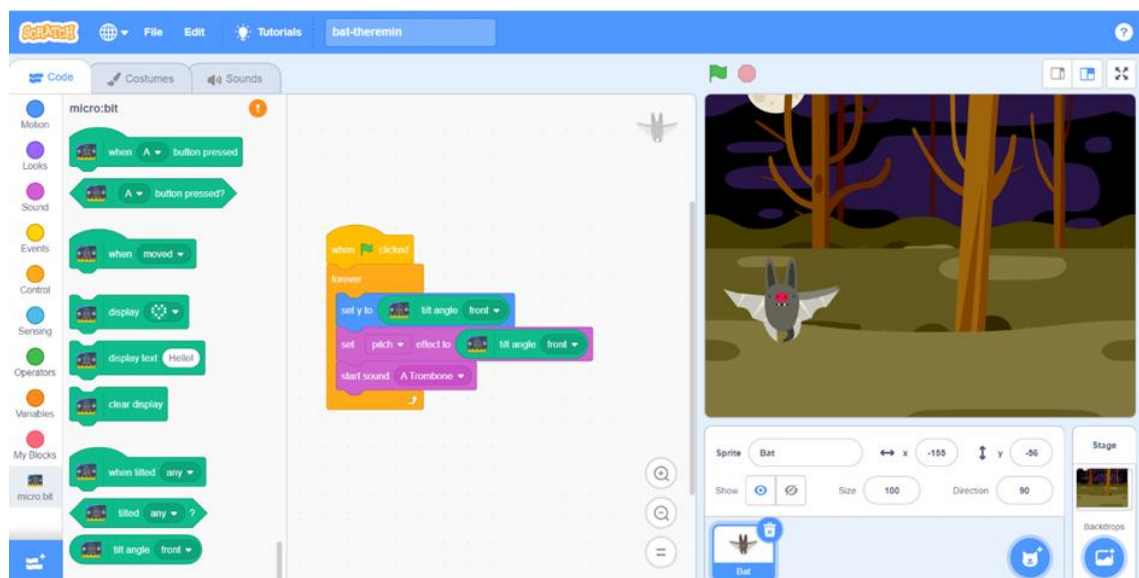


Figure 6. Scratch UI with micro:bit project.

Sphero implemented Scratch to create its own block-based IDE Sphero Edu [21]. The GUI of Sphero Edu includes the possibility to see JavaScript code that was created with blocks. Users can copy this code and edit it in a text-based editor. **Figure 7** shows one of the Sphero programs available for the Sphero community [22].

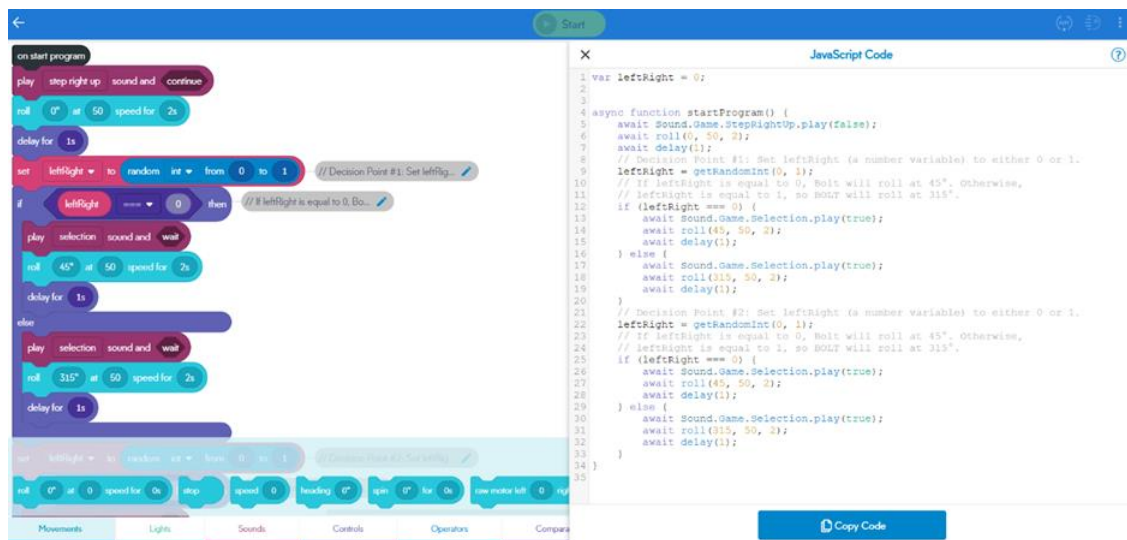


Figure 7. Sphero Edu UI with the project for the Sphero robot.

It is worth admitting that the implementation of Scratch as a programming language converts a development board into a controller connected to GUI. This approach opposes the aim to create a multipurpose programmable device that can be used without any GUI.

5.2 Blockly

Blockly is a Google project that created a visual programming language based on JavaScript [23] and makes it possible by default to implement code in JavaScript, Python, PHP, Lua, and Dart (**Figure 8**).

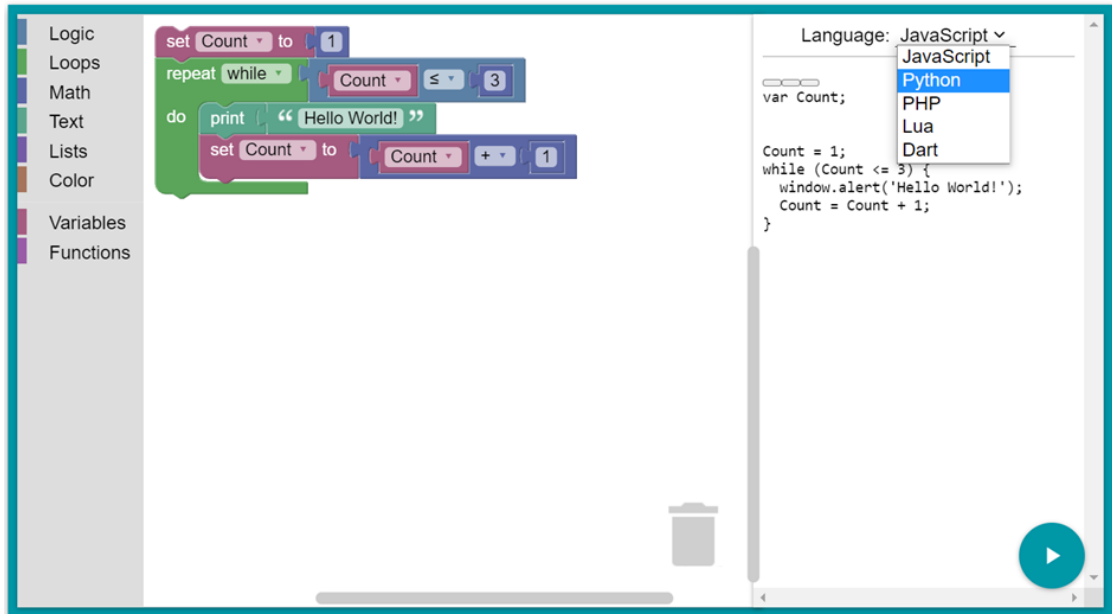


Figure 8. Blockly default UI that is available on the web [24].

Along with the possibility to generate code in multiple languages, Blockly provides GUI for building new blocks - Block Factory.

In addition to Scratch micro:bit implements Blockly as one of its programming options as Microsoft created an open-source framework MakeCode which is based on Blockly [25]. One of the most interesting features of MakeCode is an instant view of a running code on its UI (**Figure 9**).

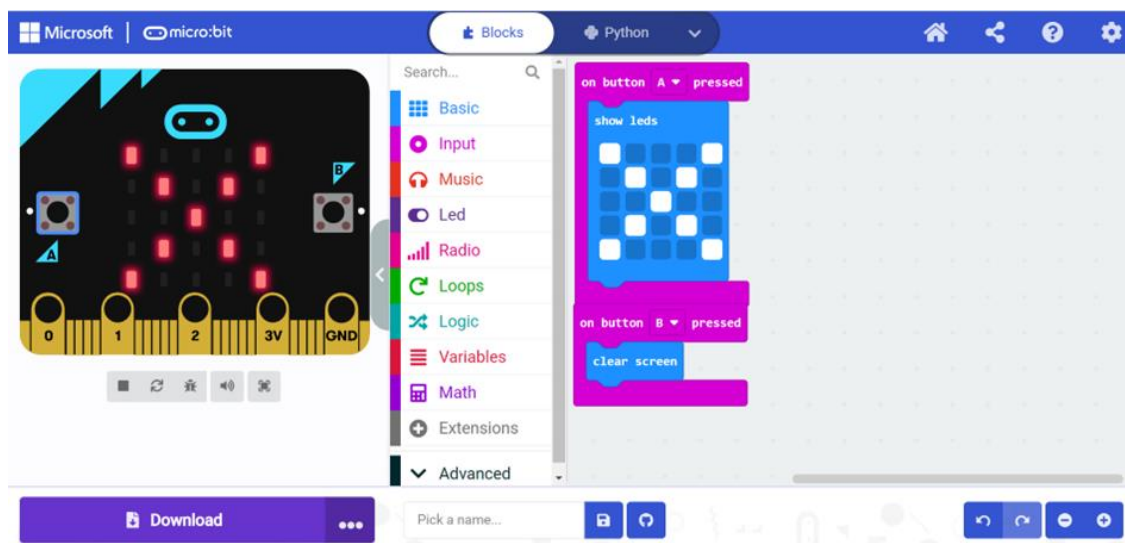


Figure 9. MakeCode UI for micro:bit available on web [26].

Other than the MakeCode implementation of Blockly is Block-based Integrated Platform for Embedded Systems or BIPES (**Figure 10**). BIPES is an open-source platform aimed for the creation of IoT applications with help of block-based code on UI and MicroPython installed on a development board [27].

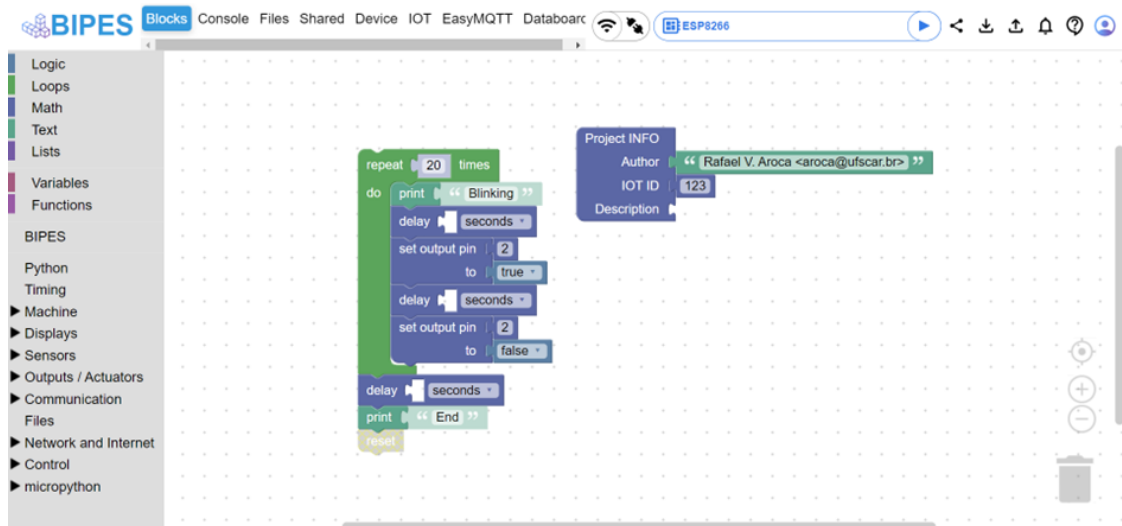


Figure 10. BIPES UI with the project for blinking a LED on ESP8266 that is available on the web [28].

One of the significant advantages of Blockly-based IDEs is that they allow flash applications to a board, then disconnect the board from the GUI of the IDE and apply the device with a standalone application [29].

6 MicroPython

MicroPython (MPY) is an open-source software implementation of a Python 3 optimised to be run directly on a microcontroller and thus may perform functions of a very minimalistic OS [30], but on the other hand, MPY can be run on top of operation system (e.g., Unix) with constrained resources [31]. One of the greatest advantages of MicroPython for AI2AI is the availability of the port for Zephyr RTOS.

6.1 Zephyr Port

Building Zephyr port of MPY is possible as a building of any other Zephyr application, utilising special Zephyr RTOS tool **west**:

```
$ west build -b name_of_the_board ~/path_to_mpy_zephyr_port
```

The same tool can be used to flash a built firmware to a board:

```
$ west flash
```

Also, it is possible to build and flash firmware using for example VS Code extension nRF Connect. Another potential way to flash the newest build in the Windows environment is to utilise the Programmer application of Nordic Semiconductor's software nRF Connect for Desktop.

After flashing a MicroPython firmware to Pall0 it is possible to establish a serial connection between Pall0 and BIPES IDE and get access to MicroPython's REPL via the BIPES console. The console displays information on a MicroPython's build and connected device info. Moreover, it is possible to check a board's response by sending commands to a board via console, for example, typing **help()** or any elementary arithmetic operation (**Figure 11**).

```
Connected using Web Serial API !
MicroPython d00f9a60-dirty on 2022-06-08; zephyr-pall0_nrf5340_rev2_cpuapp with nRF5340_CPUAPP_QKAA
Type "help()" for more information.
>>>
>>> 2+2
4
```

Figure 11. BIPES console with MicroPython's REPL response.

6.2 APIs

By default, the Zephyr port of MPY has APIs for GPIO and sensors with channels for acceleration, magnetic field, angular velocity, temperature, humidity, air pressure, illuminance, and altitude [32].

In addition to existing APIs AI2AI created APIs for LEDs, vibration, and Bluetooth mesh. These APIs were utilised to create new blocks that make it possible to program Pall0 in the BIPES environment.

7 BIPES

Blockly-based programming in combination with MicroPython made BIPES the optimal choice for Pall0 as a block-based programming IDE, that can be accessed via Google Chrome and Microsoft Edge browsers.

7.1 BIPES UI

BIPES top menu consists of several tabs and buttons which provide access to different workspaces, establish a connection to a board (serial, WiFi, Bluetooth), and allow upload/download a file with a block code (**Figure 12**).



Figure 12. BIPES UI top menu.

Although tabs of this menu provide access to different BIPES features, such as information about supported boards or EasyMQTT sessions, for this work only three main BIPES workspaces (Blocks, Console, and Files) were mainly used.

7.1.1 Blocks

Blocks workspace allows drag-and-drop blocks, combining them and creating block-based code. By default, BIPES IDE contains Python blocks that provide the possibility to run Python commands line by line even without the creation of a new block intended for a certain board. These Python blocks make it possible to construct and run the classic “blink LED” test. MicroPython has instant support of GPIO API, which allows to toggle on and off certain GPIO pin assigned to onboard LED. In this case, it is pin number 3 (**Figure 13**).

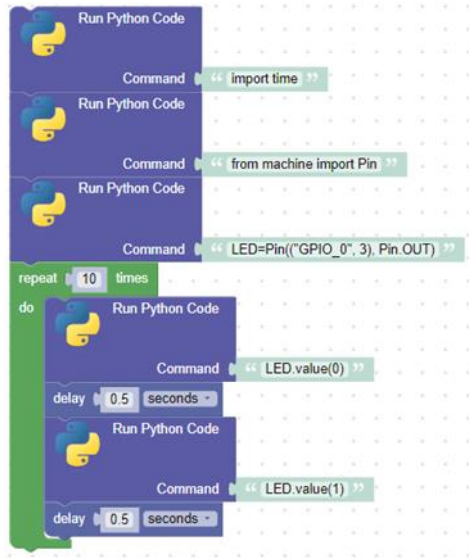


Figure 13. "Blink LED" code for Pall0 made of default Python and timing blocks.

7.1.2 Console

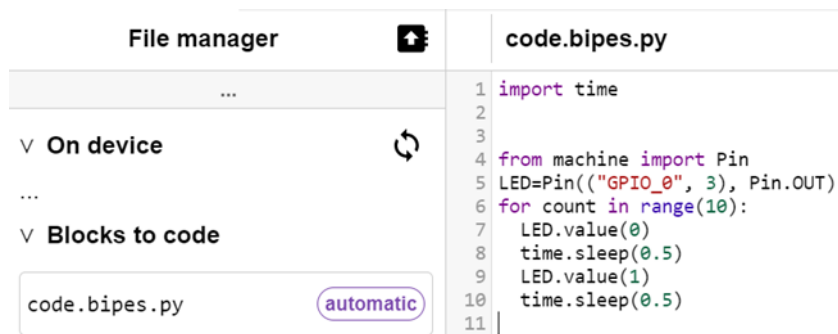
BIPES console acts as a classic terminal that provides a command prompt, shows a live view of a generated text version for a running block-based code, prints warnings, etc. **Figure 14** displays lines of Python code for the previously shown block-based "blink LED" test.

```
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
== import time
==
==
== from machine import Pin
== LED=Pin("GPIO_0", 3, Pin.OUT)
== for count in range(10):
==     LED.value(0)
==     time.sleep(0.5)
==     LED.value(1)
==     time.sleep(0.5)
==
==
==
>>> |
```

Figure 14. BIPES console with a text version of the "blink LED" code.

7.1.3 Files

This workspace provides access to Python or XML versions of running block-based code with the possibility to download these versions for further improvements or debugging. **Figure 15** shows the Python version of the “blink LED” code, which is available in the Files workspace after the block-based code has been executed.



```

File manager
...
v On device
...
v Blocks to code
code.bipes.py automatic

code.bipes.py
1 import time
2
3
4 from machine import Pin
5 LED=Pin("GPIO_0", 3), Pin.OUT
6 for count in range(10):
7     LED.value(0)
8     time.sleep(0.5)
9     LED.value(1)
10    time.sleep(0.5)
11 |

```

Figure 15. Python code for "blink LED" test available in Files.

7.2 New Blocks

Although BIPES provides an instant possibility to run a Python code, every board including Pall0 has its own APIs with the necessity to create special blocks utilising these APIs.

To prepare the device for upcoming school events, it was needed to create blocks for the accelerometer, magnetic field sensor, LEDs, vibration, and for Bluetooth mesh. In addition, default generic BIPES blocks usually don't meet the company's requirements, which establishes the next task to redesign at least several of them.

BIPES IDE involves three files to create a new working block:

- **core/block_definitions.js** - defines the appearance of each block (inputs, outputs, block type, and colour);

- **core/generator_stubs.js** - defines the actual behaviour for each block (MicroPython code for each block should be implemented here);
- **toolbox/name_of_the_device.xml** – defines which blocks are available to each device (these blocks are available to choose from the left-side tool panel) [33].

It's possible to create a new block by writing JavaScript/Python code directly to these three files, but it's easier to use Block Factory, a web-based developer tool that visualises the process of making a new block and automatically generates part of the needed JavaScript code.

8 Design of New Blocks for Pall0

This chapter explores the process of creating a custom Pall0 block for BIPES IDE by means of Block Factory UI.

Figure 16 shows Block Factory UI with an already built “Send a message” block.

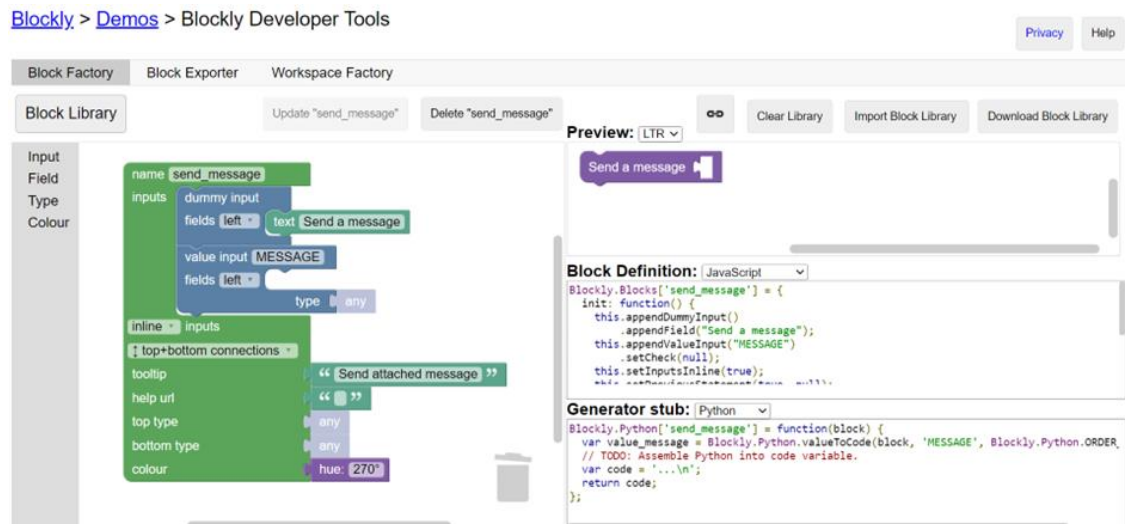


Figure 16. Block Factory UI that shows a custom-built block for BIPES.

The process of creating a new block in Block Factory resembles the process of block coding as it uses the same Blockly technology. The initial UI prompt contains only a draft for a new block, which has initial code for block definition, generation stub, and an instant block preview (with already predefined colour, type of inputs connection, etc.). All these features might be edited and completed with other blocks from the left-side menu (**Figure 17**).

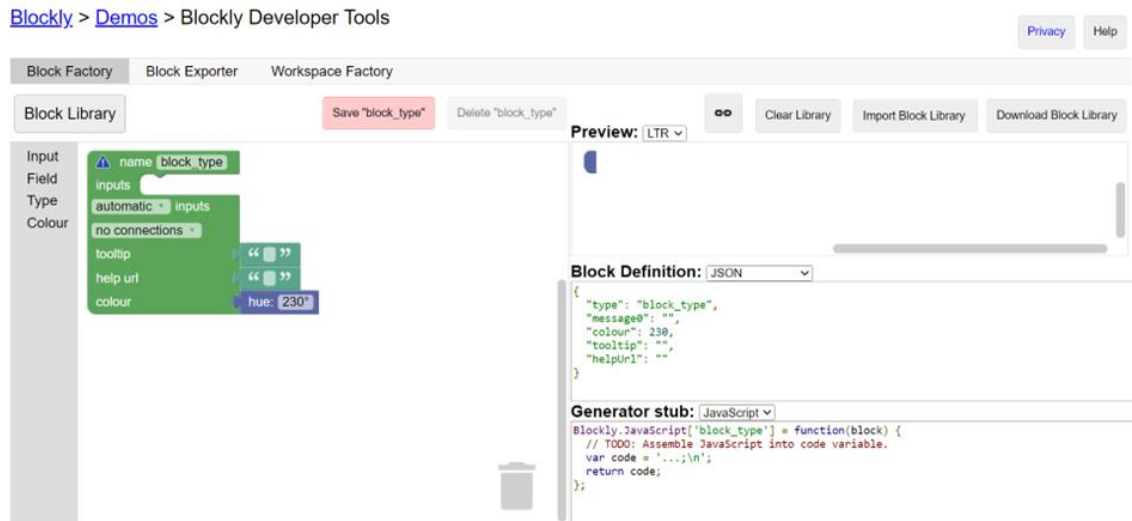


Figure 17. Block Factory. Initial GUI prompt.

8.1 Block Definition

From the left-side menu user must choose blocks for four main appearance features of a new block:

- Input (value input, statement input, or dummy input);
- Field (text input, numeric input, angle input, dropdown menu, checkbox menu, etc.);
- Type (Boolean, string, number, other (custom defined type) or any type);
- Colour of a block.

8.1.1 Input

Block Factory prompts three types of inputs: value, statement, and dummy input (**Figure 18**).

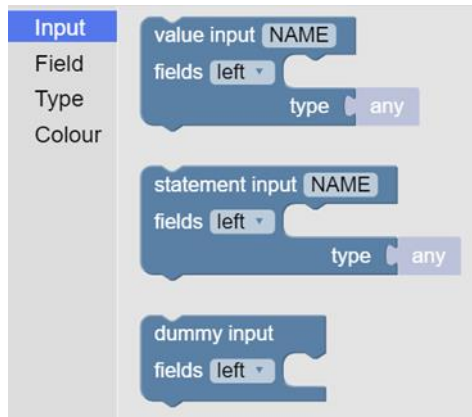


Figure 18. Block Factory input types.

Dummy input is the simplest type of input. It creates a space in a new block, which can be filled with information that helps to label a block. For this purpose, Dummy input should be completed with a text block from the Field menu (**Figure 19**), and a custom text should be added inside the text block for labelling (e.g., “Send a message” for the example block).

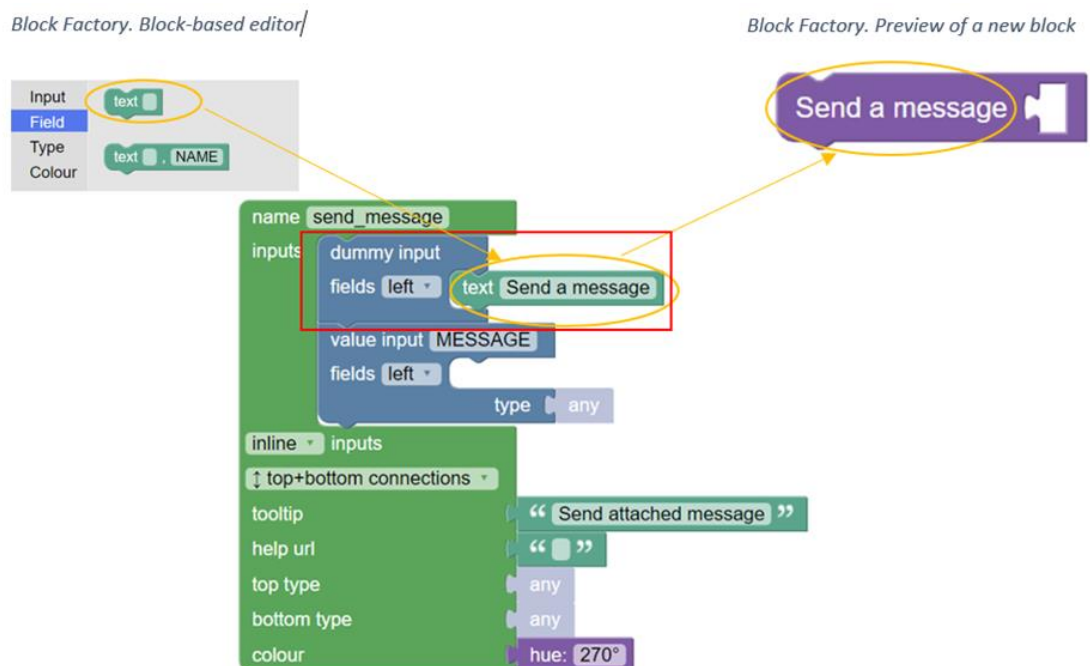


Figure 19. Block Factory. Dummy input.

Besides a text field, dummy input can be filled with other types of fields. **Figure 20** shows a dropdown menu added to a block with dummy inputs.

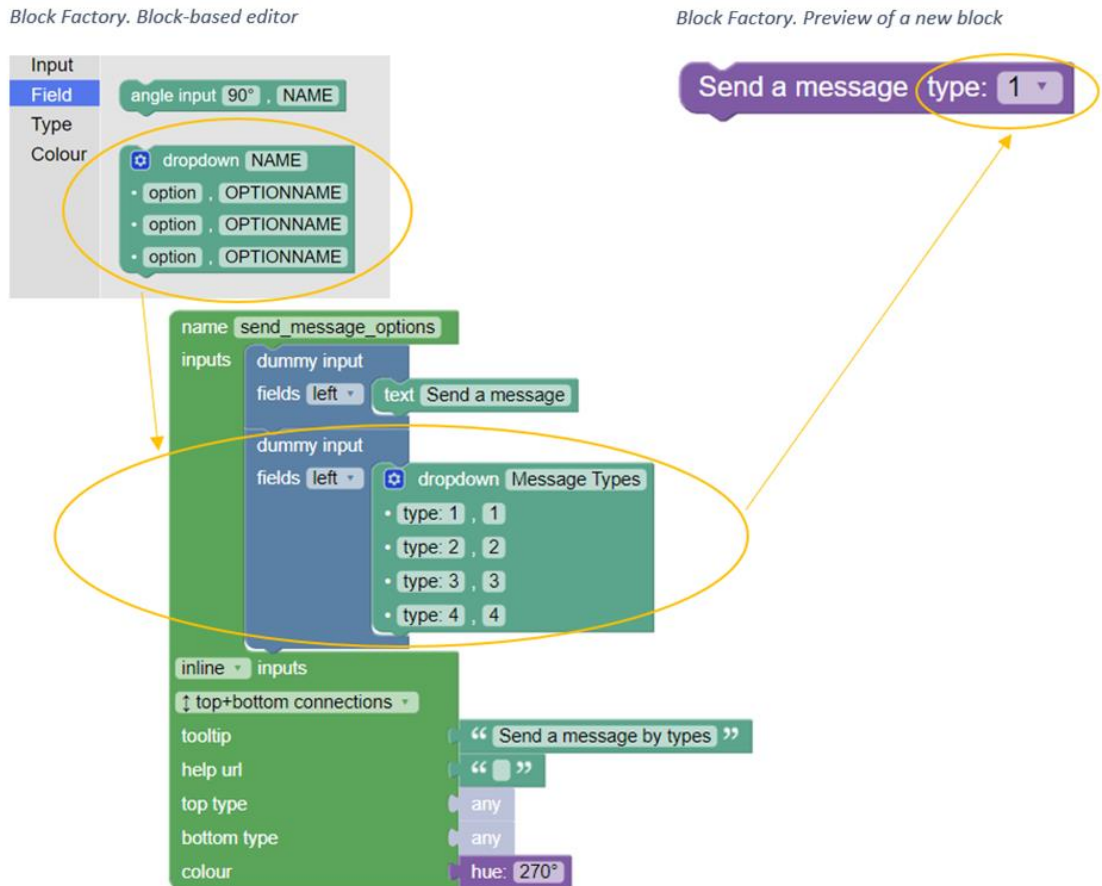


Figure 20. Block Factory. Dummy input with a dropdown menu.

It is worth admitting, that the dropdown menu field contains the icon with gear. This icon can be seen at various Blockly blocks, assigned to a construction menu, that helps to add new items to a block (e.g., **Figure 21** shows that it is possible to expand the default dropdown menu of three options with new elements).

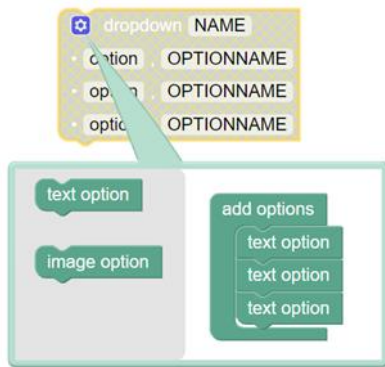


Figure 21. Gear icon.

Value input makes it possible to create a block that can connect to other blocks. Thus, it gives a possibility to create a complex block that manages values from different other blocks connected to it. **Figure 22** shows the block, which can be connected only to one block with a message for a device, but it is possible to create a block with multiple connectors for multiple blocks with value inputs (**Figure 23**).

Block Factory. Block-based editor

Block Factory. Preview of a new block

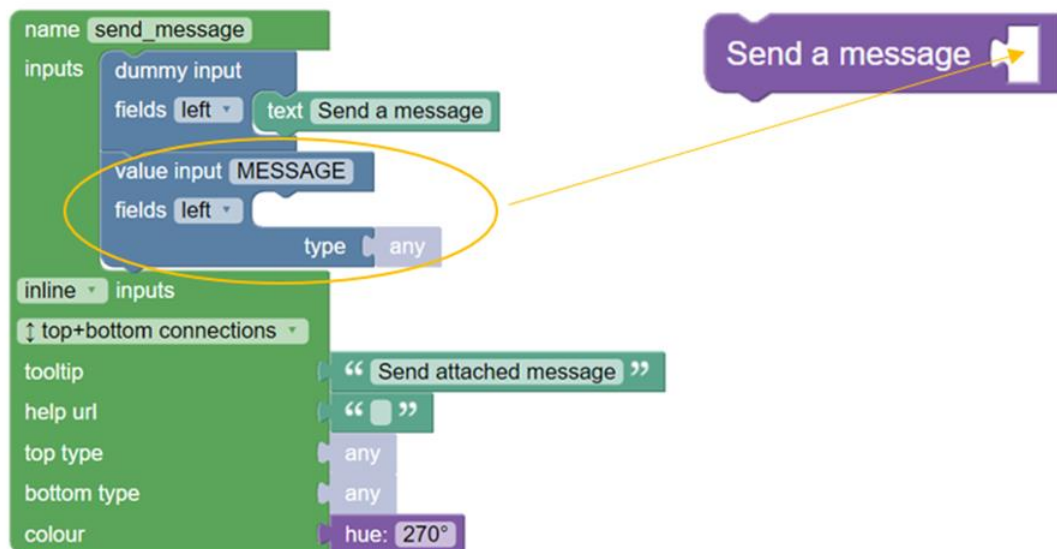


Figure 22. Value input.



Figure 23. LED control block for Pall0 with multiple value input blocks.

Block Factory constructor has an option to create blocks with different visualisation of input connections. **Figures 22,23**, and **24** display the inline type of input connection, although it is possible to choose external input (**Figure 25**).

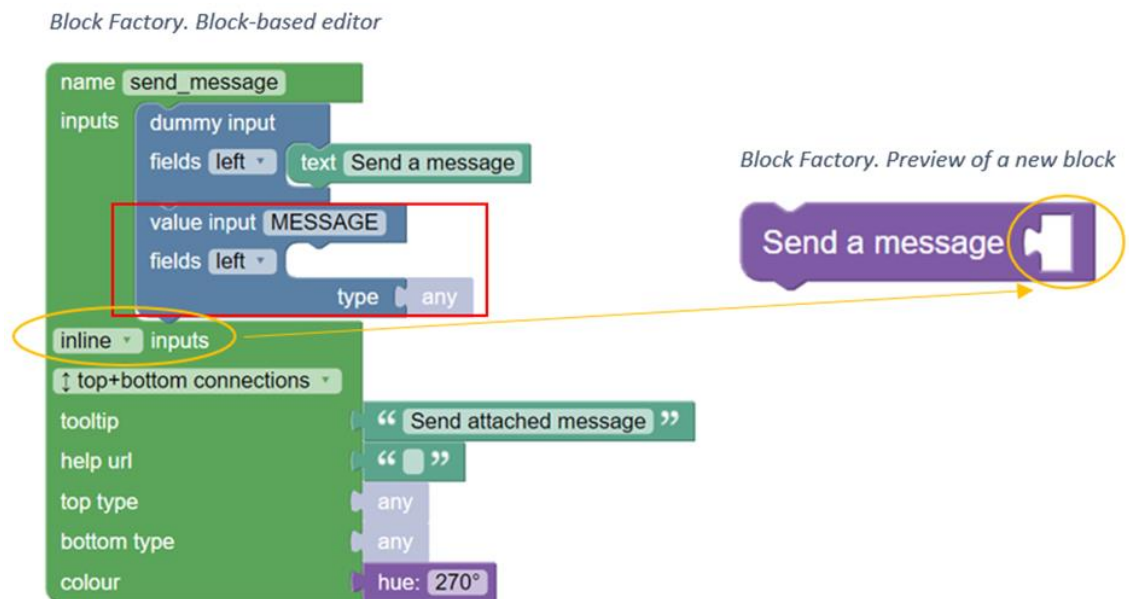


Figure 24. Block Factory. Block with an inline input.

Block Factory. Block-based editor

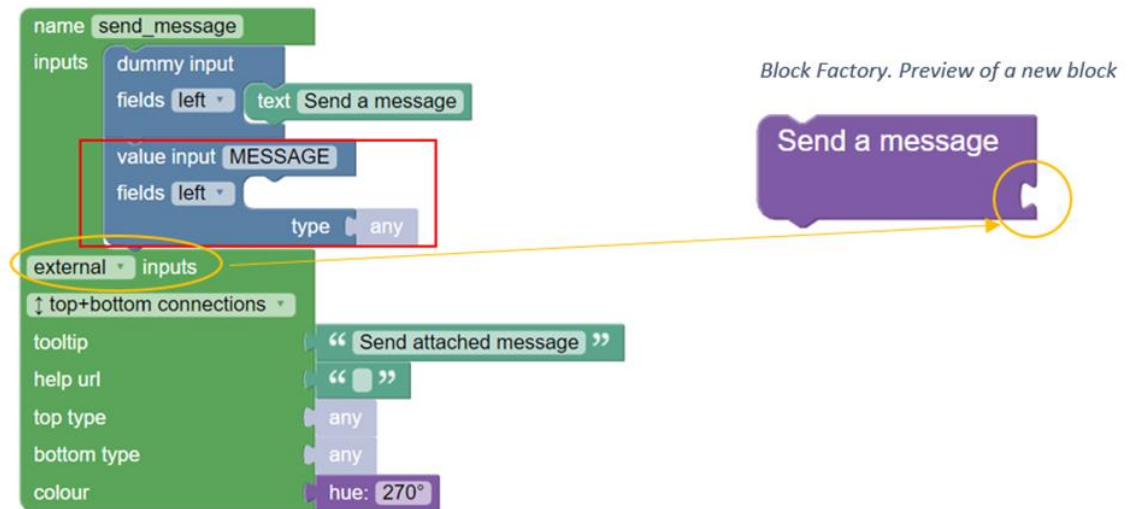


Figure 25. Block Factory. Block with external input.

External input decreases the usability of a block and readability of the whole code, which consists of blocks of such type, especially if each of these blocks has multiple external inputs. It highly increases the length of a block making it hard to see the whole code on screen without scrolling. **Figure 26** shows an example of a long block with multiple external inputs.

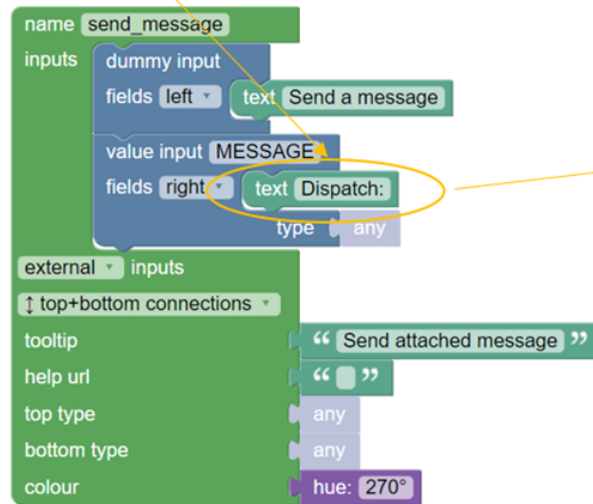
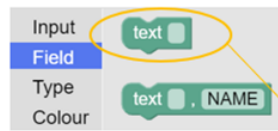


Figure 26. "Set date and time" block from original BIPES project.

Block Factory provides a possibility to add one or several fields to value input. One of the possible cases of the utilisation of this feature is a need for an additional label for a value input's connector. **Figures 26** and **27** show text fields

for value inputs. Also, it is worth noting, that besides a text field any other type of field (or several different fields) can be attached to a value input.

Block Factory. Block-based editor



Block Factory. Preview of a new block



Figure 27. Block Factory. Text field for a value input.

Statement input can be used to define a loop or IF statement. **Figure 28** shows the Pall0 timer loop, which includes the statement input.

Block Factory. Block-based editor

Block Factory. Preview of a new block

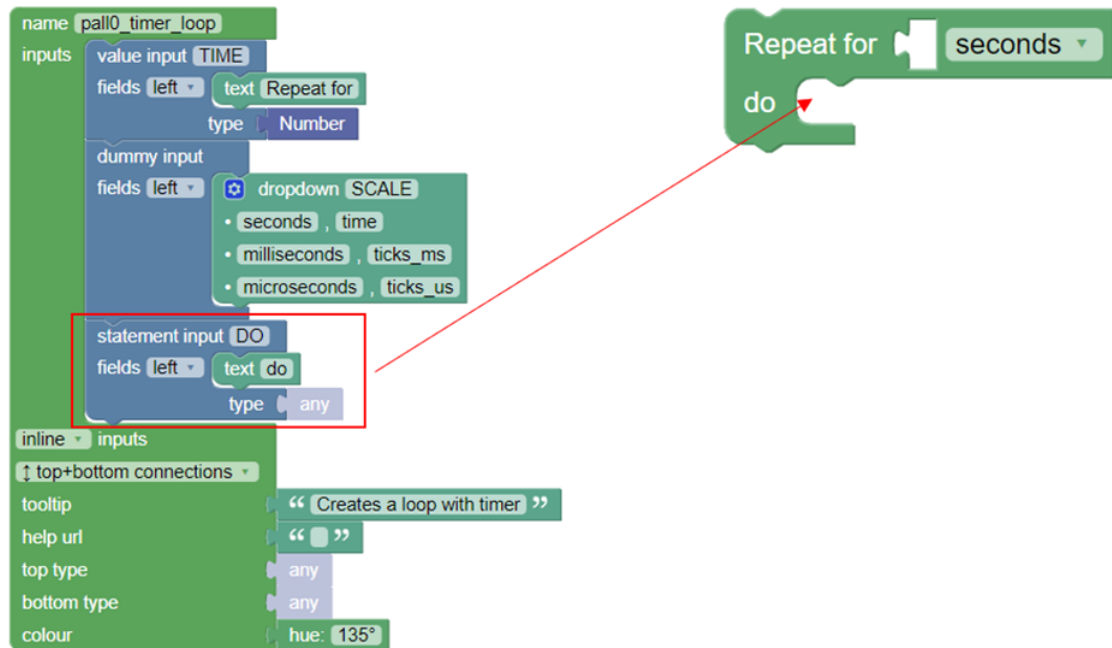
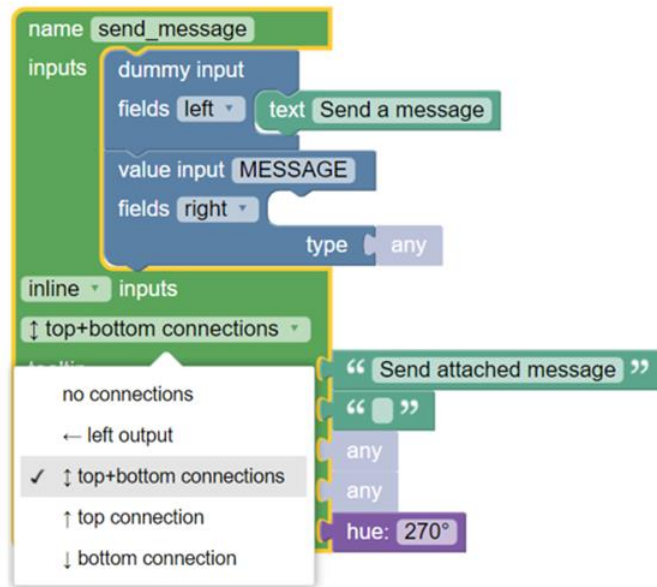


Figure 28. Block Factory. Statement input.

Regardless of the type of input, each block can be configured with different kinds of connectors, which define a method of connection to other blocks in a visual code. Top and bottom connections define the ability to form a stack of blocks, although the only bottom connection defines this block as the very first block in a stack (e.g., initialization or start block), and the only top connection defines the very last block in a stack (e.g., stop or exit block). Left output connection defines blocks, which can be connected as value inputs or parts of statements. Respectively a block without connectors is a standalone block. **Figure 29** displays Send a message block with different types of connections.

Block Factory. Block-based editor



Block Factory. Preview of a new block

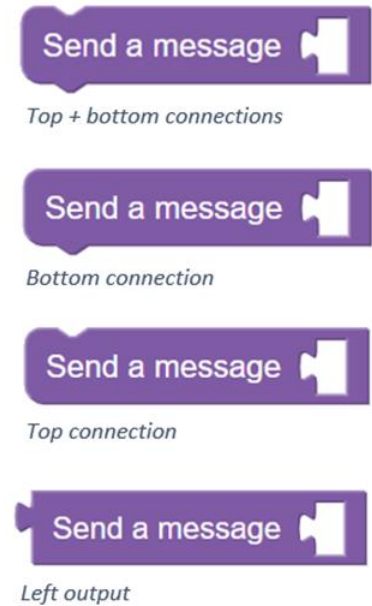


Figure 29. Block Factory. Different types of connections for blocks.

8.1.2 Field

The field menu offers blocks that elaborate any kind of input. It is possible to use several types of text fields, numeric and angle fields, fields for dropdown and checkbox menus, fields for colour selector, variable, and image. The image field by default contains the yellow star image, which can be replaced by another web or local image. **Figure 30** shows all possible variants of blocks available in the field menu of Block Factory.

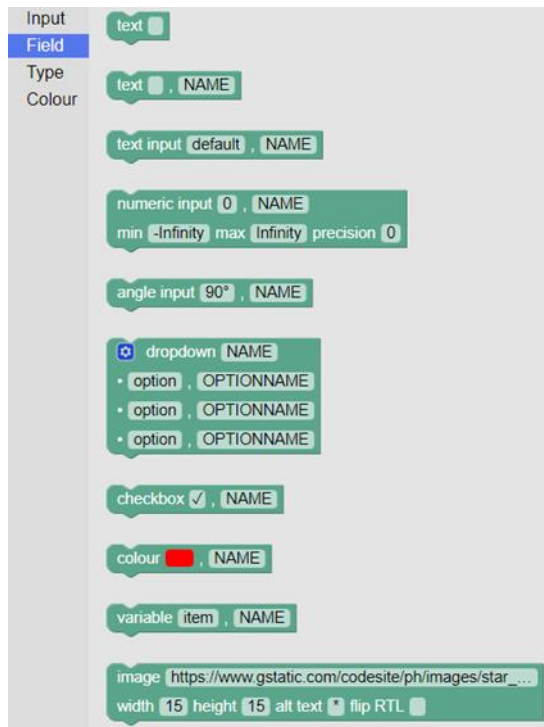


Figure 30. Block Factory. Field menu.

8.1.3 Type

Block Factory's Type menu provides settings for connectors of value and statements inputs. It is possible to make these connections compatible with any kind of block or to make a connection compatible only with certain types of blocks: Boolean, Number, String, or Array. Blockly provides a possibility to set a custom type of connection (e.g., blocks from a certain list) or to create a connection possible for several types of blocks. **Figure 31** displays Block Factory's Type menu.



Figure 31. Block Factory. Type menu.

Figure 32 shows that the value input of Pall0's "Repeat for" loop has a Number type of connector, thus only blocks with number values can be attached here, although the statement input is compatible with any type of block. The top and bottom connectors of Pall0's "Repeat for" loop are also compatible with any possible types of blocks above and below in the stack of blocks.

Block Factory. Block-based editor

Block Factory. Preview of a new block

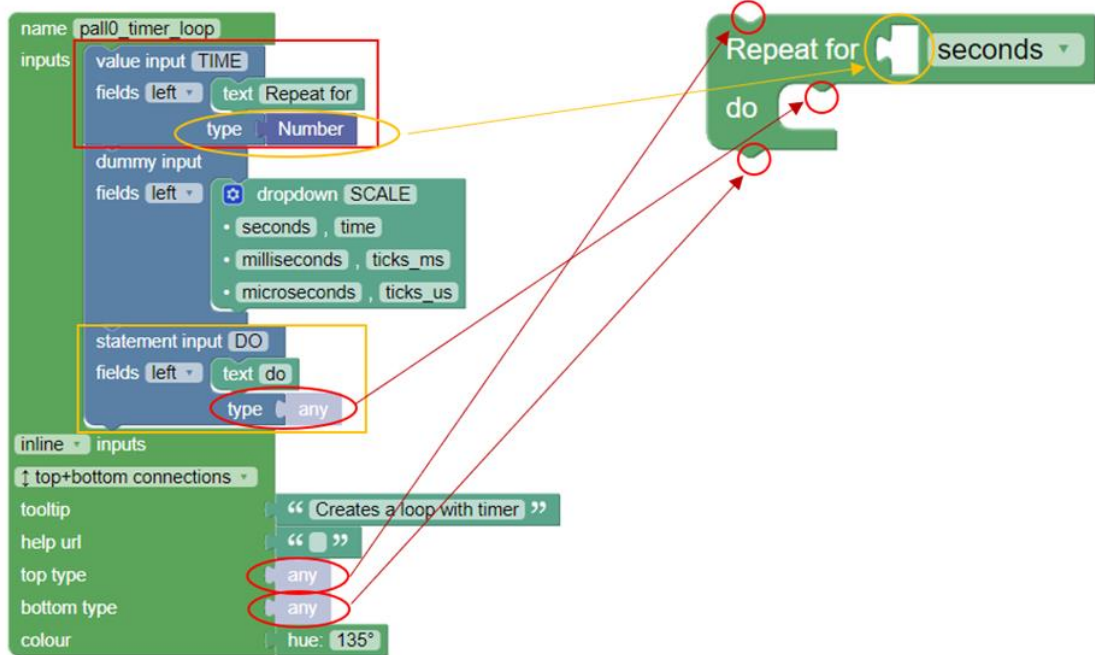


Figure 32. Block Factory. Different types of connections for Pall0's "Repeat for" loop.

Figure 33 demonstrates a custom-defined type of connection for Pall0's "LED control" block. The value input of this block has led_color type of connection, thus an end-user cannot insert into this block any other value (number, string, etc.) than the colour of the LED.

Block Factory. Block-based editor

Block Factory. Preview of a new block

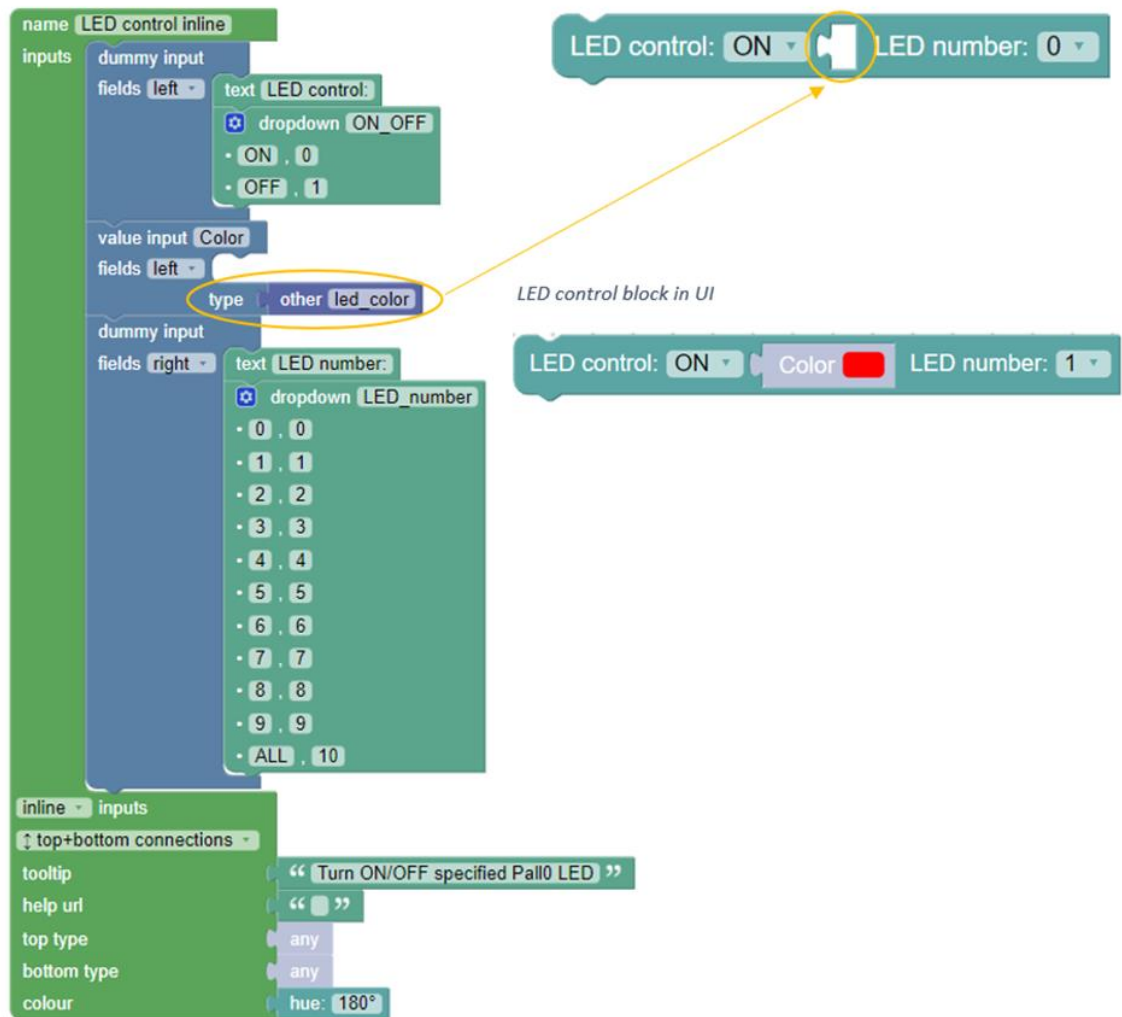


Figure 33. Block Factory. Custom-defined type of connector.

8.1.4 Colour

The last menu in Block Factory is Colour selection, which provides preselected nine colours in the Hue-Saturation-Value colour model for the block constructor (**Figure 34**).

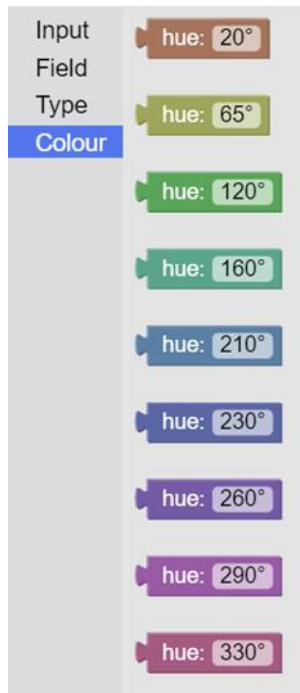
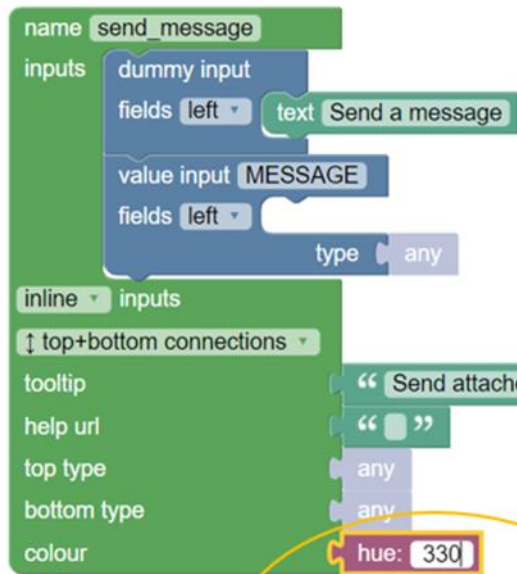


Figure 34. Block Factory. Colour menu.

These colour blocks can be attached to colour fields in a block constructor, but there are other possibilities to change a colour of a block that is under construction: a user can type the number of a hue-saturation colour directly inside the colour block or utilise a dial selector. The change of the colour of a colour block is instantly visible in the block preview (**Figure 35**).

Block Factory. Block-based editor



Block Factory. Preview of a new block

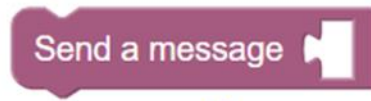


Figure 35. Block Factory. Change of the colour of Pall0's "Send a message" block.

HSV colour model shares with blocks in a block-based code a cohesive, but very constrained palette. Some types of blocks (e.g., start or stop blocks) might need to be of brighter and more perceptible colour. Blockly provides a possibility to set a colour of a block using the HEX value of RGB colour instead of the HSV number. **Figure 36** shows the code for the "Stop" block for Pall0 with HEX value for the red colour in JavaScript notation.

block_definition.js (code for STOP block in JavaScript notation)

View of the STOP block for Pall0

```
Blockly.Blocks['pall0_stop'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("STOP");
    this.setPreviousStatement(true, null);
    this.setColour("#ff0000");
    this.setTooltip("Stops Pall0");
    this.setHelpUrl("");
  }
};
```



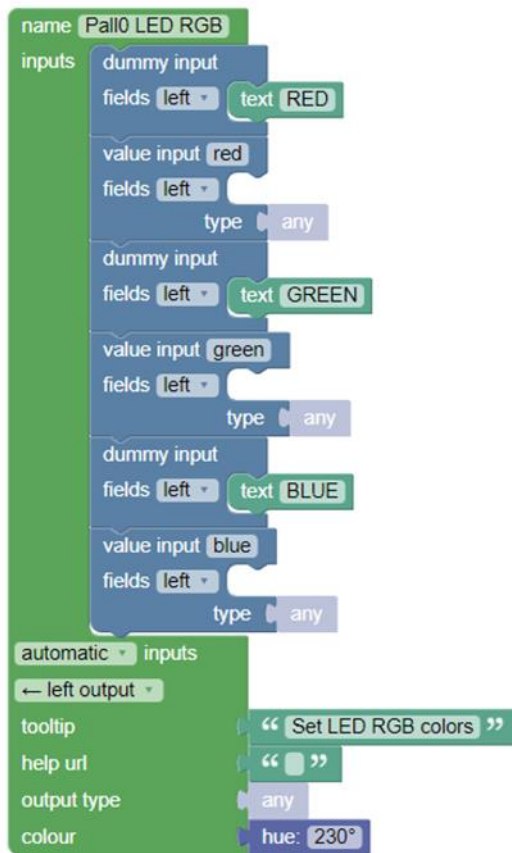
Figure 36. HEX value for colour of a block instead of HSV.

8.1.5 Additional features

It can be concluded that Block Factory provides a possibility to complete block definition code for `block_definitions.js` file only using block coding. It is possible to admit some excludes such as other than HSV colours for blocks, which cannot be set directly in Block Factory.

Moreover, it is worth admitting, that a block definition code can contain lines that affect visualisation of a block and cannot be substituted by means of Block Factory's block coding. As an example, the code for Pall0's block, which controls the colour of RGB LED, can be provided. **Figure 37** shows Block Factory's standard block code and automatically generated block definition code.

Block Factory. Block-based editor



Block Factory. Block Definition (JavaScript)

```
Blockly.Blocks['pall0_led_rgb'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("RED");
    this.appendValueInput("red")
      .setCheck(null);
    this.appendDummyInput()
      .appendField("GREEN");
    this.appendValueInput("green")
      .setCheck(null);
    this.appendDummyInput()
      .appendField("BLUE");
    this.appendValueInput("blue")
      .setCheck(null);
    this.setOutput(true, null);
    this.setColour(230);
    this.setTooltip("Set LED RGB colors");
    this.setHelpUrl("");
  }
};
```

Block Factory. Preview of a new block



Figure 37. Standard Block Factory code for Pall0's block, which controls LEDs colour.

In addition to this standard generated block definition code, additional lines of code can be manually added to the block's style code (**Figure 38**). It highly increases the usability of block coding, as an end-user gets an instant view of a colour that will be applied to the device's LED.

Edited Block Definition (JavaScript)

```
Blockly.Blocks['pall0_led_rgb'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("RED");
    this.appendValueInput("red")
      .setCheck(null);
    this.appendDummyInput()
      .appendField("GREEN");
    this.appendValueInput("green")
      .setCheck(null);
    this.appendDummyInput()
      .appendField("BLUE");
    this.appendValueInput("blue")
      .setCheck(null);
    this.setOutput(true, null);
    this.setColour(230);
    this.setTooltip("Set LED RGB colors");
    this.setHelpUrl("");
  },
  styleBlock: function(colours) {
    colours = colours.map(x => parseInt(x))
    colours = colours.includes(NaN) ? [89,102,166] : colours
    if(colours.every((e) => {return e <= 255}) && colours.every((e) => {return e >= 0})) {
      let hex_ = Tool.RGB2HEX (colours [0], colours [1], colours [2]);
      this.setColour(hex_);
    } else
      this.setColour("#FF0000");
  }
};
```

Block's behaviour after its block definition code was changed



Figure 38. Edited block definition for Pall0's block.

The completed code of a new block should be added to **core/block_definitions.js** file.

8.2 Generator Stub

While Block Factory allows almost completely to create block definition code for the **block_definitions.js** file using only block coding, it makes only a draft of a generator stub code for the **generator_stubs.js** file, that contains only JavaScript functions, which are needed for inputs defined in Block Definition. **Figure 39** shows, that the generator stub is incomplete and contains this notification line (for Python notation):

```
// TODO: Assemble Python into code variable.
```

Also, **var code** is not assigned to anything significant:

```
var code = '...\n';
```

Note, that this code already contains an assigned variable for value input “message”.

Generator stub:

```
Blockly.Python['send_message'] = function(block) {
  var value_message = Blockly.Python.valueToCode(block, 'MESSAGE', Blockly.Python.ORDER_ATOMIC);
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

Figure 39. Block Factory. Generator stub code for Pall0's “Send a message” block.

8.2.1 Variables

Variable “code” should be created manually with Python code. This variable is responsible for the actual behaviour of a block in block code. **Figure 40** displays a full variant of generator stub code for Pall0's “Send a message” block including Python code for variable “code”. Python code in its turn utilises the API for Bluetooth Mesh created by AI2AI.

```
Blockly.Python['send_message'] = function(block) {
  var value_message = Blockly.Python.valueToCode(block, 'MESSAGE', Blockly.Python.ORDER_ATOMIC);

  var code = 'm.send(' + value_message + ')\n';
  code += 'utime.sleep(1.1)\n';

  return code;
};
```

Figure 40. Full generator stub code for Pall0's "Send a message" block.

It is worth admitting, that the variable "value_message" contains a code from the generator stub of another block (e.g. Pall0's "Turn ON all LEDs" block shown in **Figure 41**).



Figure 41. Pall0's "Send a message" block with the value input that contains the "Turn ON all LEDs" block.

Thus, any value input that was defined for a block in Block Definition, creates its own variable in Generation Stub of this particular block. And as it was said before, the variable picks code from **var code** of a block, which is placed into a value input. In this case, the value input of the "Send a message" block is filled with the "Turn ON all LEDs" block. **Figure 42** shows the generator stub code for the "Turn ON all LEDs" block.

```
Blockly.Python['message_all_leds_on'] = function(block) {
  var code = "led.all_on";

  return [code, Blockly.Python.ORDER_NONE];
};
```

Figure 42. Generator stub code for the "Turn ON all LEDs" block.

Altogether these two blocks ("Send a message" and "Turn ON all LEDs") create the following consequence. The "Send a message" block utilises the specific API for Bluetooth Mesh and the "Turn ON all LEDs" block contains a string, which is needed for this API.

Python code for block's variable "code" can contain multiple lines. In this case, it is very important to use Python special characters for a new line (`\n`) and an indentation (`\t`). **Figure 43** demonstrates the generation stub's code for Pall0's "Receive a message" block, which contains a multiline Python code, that utilises APIs for Bluetooth mesh and LED control.

```
Blockly.Python['receive_message'] = function(block) {
    var code = "utime.sleep(0.1)\n";
    code += "str = m.receive()\n";
    code += "if str == 'led.all_on':\n";
    code += "\tled.all_on()\n";
    code += "elif str == 'led.all_off':\n";
    code += "\tled.all_off()\n";
    return code;
};
```

Figure 43. Generator stub's code for Pall0's "Receive a message" block with highlighted Python's special characters.

8.3 XML file for BIPES toolbox directory

BIPES toolbox folder contains **XML** files that are responsible for the display of created blocks. **Figure 44** shows Pall0's toolbox in BIPES UI and **Figure 45** displays the **XML** code for the "Start/Stop" category of blocks.



Figure 44. Pall0's toolbox in BIPES UI with opened "Start/Stop" category of blocks.

```

<category name="Start/Stop">
  <block type="pall0_start">
  </block>

  <block type="pall0_stop">
  </block>

</category>

```

Figure 45. XML code for Pall0's "Start/Stop" category of blocks.

8.3.1 Shadow of another block

To increase the usability of blocks it is worth using shadows of other blocks, which could be inserted into value inputs of the main block. **Figure 46** demonstrates that Pall0's RGB block already contains shadows of three number blocks with the pre-set value of 0 in each. Shadows of blocks have the same function as the blocks they represent. End-users need only input RGB values to set the LED colour.

XML code for Pall0 RGB block

```

<block type="pall0_led_rgb">
  <value name="red">
    <shadow type="math_number">
      <field name="NUM">0</field>
    </shadow>
  </value>
  <value name="green">
    <shadow type="math_number">
      <field name="NUM">0</field>
    </shadow>
  </value>
  <value name="blue">
    <shadow type="math_number">
      <field name="NUM">0</field>
    </shadow>
  </value>
</block>

```

View of RGB block for Pall0




Figure 46. Shadows of "Number" block inside Pall0's RGB block.

Without shadows, an end-user should first drag and drop three number blocks inside the RGB block and only then gets an opportunity to set RGB values (**Figure 47**).

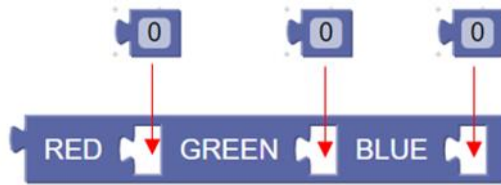


Figure 47. Pall0's RGB block without shadows of number blocks.

The shadow of a block can be replaced at any moment in BIPES GUI by any other suitable block (**Figure 48**).

"Send a message" block with the shadow of "Turn ON all LEDs" block



The same block with the original block placed over its shadow



The same block with "Turn OFF all LEDs" block placed over the shadow of other block



Figure 48. Block's shadow and its replacement.

9 BIPES modification for Pall0

The addition of any new block or alteration in BIPES UI creates a necessity to update the version of BIPES software.

9.1 Build or update software version

For this work, an offline version of modified BIPES was used. To build a new version of the software that contains all new editions of blocks or UI is needed to run the command

make offline

in the directory that contains Makefile. This command creates **index_offline.html** that runs a modified version of BIPES in a browser.

9.2 UI Modification

Originally the BIPES project was created for multiple different boards and provides for an end-user multiplicity of toolbox categories and features for boards' use. Most of those features were excessive for STEM/STEAM lessons, and one of the tasks was the modification of the BIPES UI for Pall0. To implement this task the main changes were made in two files, which are located in the UI folder of the BIPES project: **index.html** (e.g., UI buttons) and **style.css** (e.g., UI icons). **Figure 49** displays the difference between the original BIPES project UI and the modified Pall0 UI.

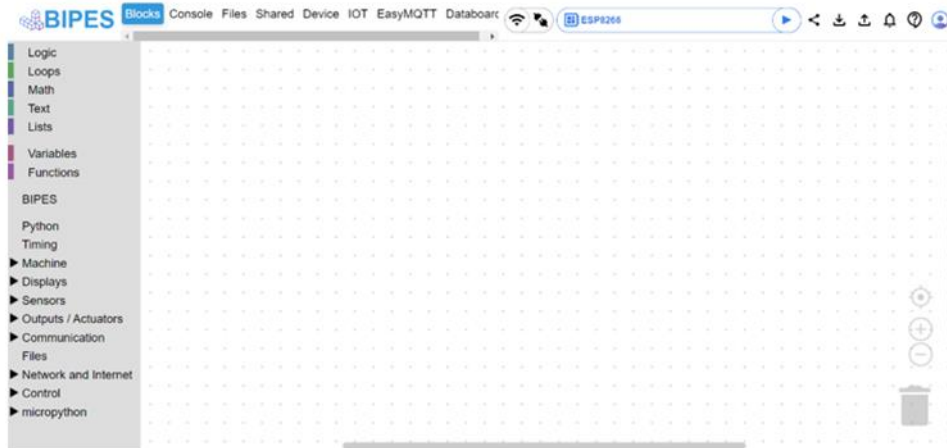
Original BIPES UI*UI modified for Pall0*

Figure 49. UI difference.

10 Demonstration. Hot Potato Game

Hot Potato was one of the games prepared for school events in May 2022 in Turku and Helsinki. The rules of this game are:

- To start the game session, the player should rotate Pall0 upside down.
- All Pall0 LEDs become green.
- Player has 30 seconds to do something with the “potato” (e.g., bring it to a specific place) before it becomes “too hot”. After 30 seconds all LEDs become red, and Pall0 vibrates. Then LEDs and vibration are turned off, which means that this game session has ended.
- Within 30 seconds of a game session the player should move “potato” as carefully as possible because any shake of “potato” makes it “hotter” and changes the LEDs’ colour gradually from green to red. When LEDs become red the game session ends.
- To start a new game session, the player should turn Pall0 upside down again.

This game utilises Pall0’s MicroPython APIs for vibration, LED, and accelerometer. **Figure 50** shows the block code for the Hot Potato game. It is worth noting, that this code was used for school events in May 2022 (**Figure 51**) and made the game process possible, but it was not optimised, because developing new blocks for Pall0 was an ongoing process, and at that time problems with “Repeat for X seconds” block were revealed. That is why it was necessary to utilise the “Break out of loop” block, although the code should work theoretically without this block. Lately, this problem was solved.

```

START
repeat while true
do
  Detect movement
  if rotated
  do
    set green to 255
    set red to 0
    LED control: ON RED red GREEN green BLUE 0 LED number: ALL
    repeat while red < 255 and green > 0
    do
      Repeat for 30 seconds
      do
        Detect movement
        if shaken level 1
        do
          change red by 10
          change green by -10
          LED control: ON RED red GREEN green BLUE 0 LED number: ALL
          if red ≥ 255
          do
            break out of loop
        break out of loop
      LED control: ON Color red LED number: ALL
      repeat 3 times
      do
        VIBRO ON/OFF ON
        delay 50 milliseconds
        VIBRO ON/OFF OFF
        delay 50 milliseconds
      STOP
      delay 0.5 seconds
  
```

Figure 50. Block code of Hot Potato game in BIPES UI.

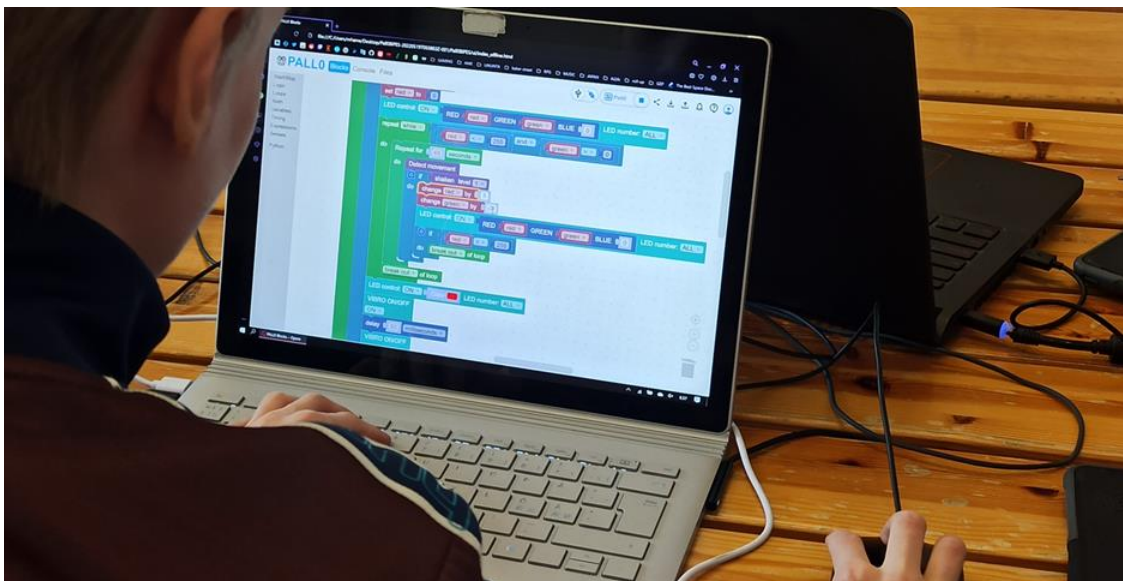


Figure 51. The pupil observes the block code of the Hot Potato game during a school event in Turku.

11 Conclusion

The process of implementation of block-based programming IDE for Pall0 was completed by May 2022, as well as the design of custom Pall0 blocks for BIPES, which can utilise APIs for vibration, LED, accelerometer, and Bluetooth Mesh.

11.1 Results

Pall0 and its block-based programming IDE were demonstrated as a potential educational device for teachers and pupils during several demonstration classes in Turku and Helsinki (**Figure 52**). It is worth admitting that those demonstrations were conducted without significant software problems on Windows OS as well as on iPadOS. Pupils could easily edit block codes that were made for Pall0, as well as could create their own block code utilising Pall0's features.

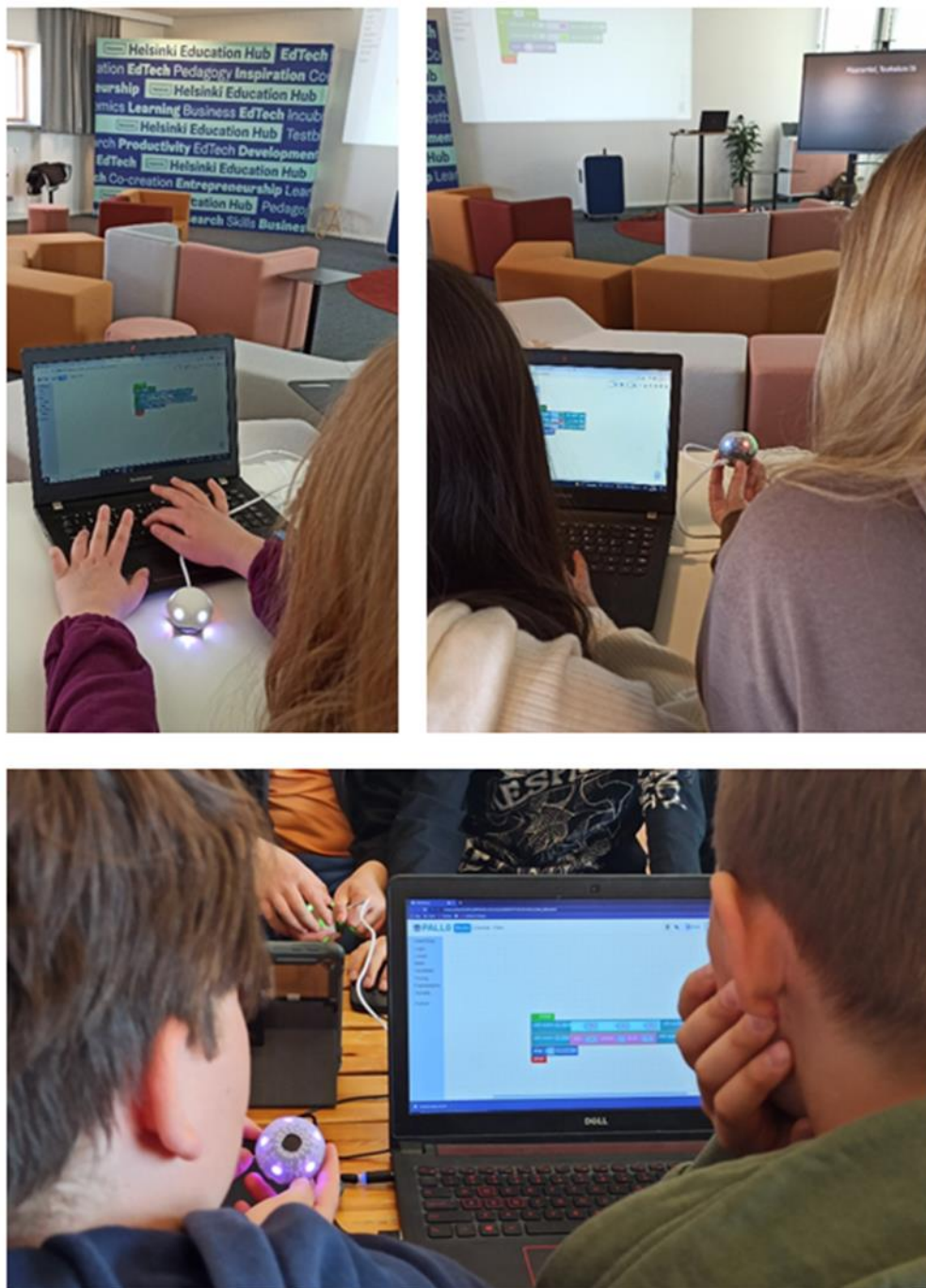


Figure 52. Events in Helsinki and Turku in May 2022.

During the school events, one of the main goals was to show pupils that a programmable device can be a game tool that helps interact with another player avoiding using a computer or smartphone with an internet connection. Moreover, it was shown that live interaction between players makes it possible to change

the rules of the game on the fly either without changing the code of the game (e.g. adding obstacles to a player's path, which makes it more difficult not to shake Pall0 while trying to bring it to a certain spot within 30 seconds), or by an edition of the game code using block-based programming (e.g. changing timer, activating random numbers in colours change, etc.).

On the other hand, during these events, it was demonstrated to pupils and teachers that block-based programming is an easy approach to start studying programming even for those who have no experience in coding.

11.2 Possible Improvements

Even though the functionality that was required was achieved, there is a list of possible improvements that can be made to increase the usability of Pall0's block-based programming IDE:

- addition of new blocks that can utilise APIs for buttons, screen, gyroscope, etc.;
- improvement of generator stub code to optimise the memory usage;
- implementation of block's design ideas from Scratch, which are not available for Blockly;
- modification of BIPES UI to add features that increase usability (e.g., display of sensors' values).

References

- [1] 360iResearch. Programmable Robots for STEAM Learning Tools Market Research Report by Type (Primary Education and Secondary Education), Component, Region (Americas, Asia-Pacific, and Europe, Middle East & Africa) - Global Forecast to 2027 - Cumulative Impact of COVID-19 [Online]. MarketResearch.com; Apr. 2022. Available from: <https://www.marketresearch.com/360iResearch-v4164/Programmable-Robots-STEAM-Learning-Tools-31205813/> [Accessed 14 September 2022].
- [2] FAQ [Online]. AI2AI; 2022. Available from: <https://www.ai2ai.fi/> [Accessed 12 September 2022].
- [3] Sensor Highlights [Image on Internet]. Developer AI2AI; 2022. Available from: <https://www.ai2ai.fi/dev> [Accessed 12 September 2022].
- [4] Lehrbaum R. Meet Linux's little brother: Zephyr, a tiny open-source IoT RTOS [Online]. LinuxGizmos; 17 Feb. 2016. Available from: <https://linuxgizmos.com/zephyr-a-tiny-open-source-iot-rtos/> [Accessed 12 September 2022].
- [5] Zephyr Project members and individual contributors. Introduction [Online]. Zephyr Project Documentation; 12 Sep. 2022. Available from: <https://docs.zephyrproject.org/latest/introduction/index.html> [Accessed 12 September 2022].
- [6] Branstetter D. Design Challenge: Understanding STEAM vs STEM [Online]. The Institute for Arts Integration and STEAM; Apr. 2018. Available from: <https://artsintegration.com/2018/04/01/stem-to-steam-action-jackson-catapults/> [Accessed 12 September 2022].
- [7] A STEM Initiative: Science, Technology, Engineering and Math [Online]. The Phoenix Symphony; 2022. Available from: <https://www.phoenixsymphony.org/education-and-community/mind-over-music> [Accessed 12 September 2022].

- [8] Mind Over Music in Practice [Online]. The Phoenix Symphony; 2022. Available from: <https://www.phoenixsymphony.org/education-and-community/mind-over-music> [Accessed 12 September 2022].
- [9] STEAM Education Kits & Educational Robots For Students PK–12 [Online]. Sphero; 2022. Available from: <https://sphero.com/collections/for-school> [Accessed 13 September 2022].
- [10] Austin J, Baker H, Ball T, Devine J, Finney J, De Halleux P, Hodges S, Moskal M, Stockdale G. The BBC micro:bit – From the U.K. to the World [Online]. Communications of the ACM, Mar. 2020, Vol. 63 No. 3, Pages 62-69. Available from: <https://cacm.acm.org/magazines/2020/3/243028-the-bbc-microbit-from-the-uk-to-the-world/fulltext> [Accessed 13 September 2022].
- [11] 57,000 BBC micro:bits to be donated to primary schools [Online]. Micro:bit Educational Foundation; 30 Mar. 2022. Available from: <https://microbit.org/news/2022-03-30/57000-bbc-microbits-to-be-donated-to-primary-schools/> [Accessed 13 September 2022].
- [12] New micro:bit with sound [Image on Internet]. Micro:bit Educational Foundation; 2022. Available from: <https://microbit.org/get-started/user-guide/overview/> [Accessed 13 September 2022].
- [13] Othermachines. Spooky Halloween micro:bit Scratch bat theremin [Video on Internet]. YouTube; 31 Oct. 2019. Available from: <https://www.youtube.com/watch?v=IOD5LDxfzgg> [Accessed 13 September 2022].
- [14] Mantz K. What is Sphero? The Who, What, and Why [Online]. Sphero; 07 Jun. 2021. Available from: <https://sphero.com/blogs/news/what-is-sphero> [Accessed 13 September 2022].
- [15] Sphero Edu App [Application on Internet]. Sphero; 2022. Available from: <https://sphero.com/pages/apps> [Accessed 13 September 2022].
- [16] Inspired Minds. Painting with Robots - Sphero [Video on Internet]. YouTube; 27 Nov. 2017. Available from:

- <https://www.youtube.com/watch?v=sroZqLzLiBU> [Accessed 13 September 2022].
- [17] About Scratch [Online]. Scratch; 2022. Available from: <https://scratch.mit.edu/about> [Accessed 13 September 2022].
- [18] Community statistics at a glance [Online]. Scratch; 2022. Available from: <https://scratch.mit.edu/statistics> [Accessed 13 September 2022].
- [19] Scratch 3.0 [Online]. Scratch Wiki; 06 Jun. 2022. Available from: https://en.scratch-wiki.info/wiki/Scratch_3.0 [Accessed 13 September 2022].
- [20] Let's code [Online]. Micro:bit Educational Foundation; 2022. Available from: <https://microbit.org/code/> [Accessed 13 September 2022].
- [21] Sphero Team. Sphero Edu Implements Scratch [Online]. Sphero; 19 Jun. 2018. Available from: <https://sphero.com/blogs/news/sphero-edu-implements-scratch> [Accessed 13 September 2022].
- [22] Sphero. Lesson 7: BOLT Plays with Probability [Online]. Sphero Edu; 2022. Available from: <https://edu.sphero.com/remixes/9564362> [Accessed 13 September 2022].
- [23] Blockly for Web [Online]. Google Developers; 2022. Available from: <https://developers.google.com/blockly> [Accessed 13 September 2022].
- [24] Try Blockly [Online]. Google Developers; 2022. Available from: <https://developers.google.com/blockly> [Accessed 13 September 2022].
- [25] About MakeCode [Online]. MakeCode (PXT) Documentation; 2022. Available from: <https://makecode.com/about> [Accessed 13 September 2022].
- [26] Editor [Online]. Micro:bit MakeCode; 2022. Available from: <https://makecode.microbit.org/#editor> [Accessed 13 September 2022].

- [27] Arouca R, Marques J, Silva CA. Tech Details. How BIPES works [Online]. BIPES:Dev; 2021. Available from: <http://bipes.net.br/docs/get-started/tech-details.html> [Accessed 13 September 2022].
- [28] Arouca R. LED test for ESP8266 [Online]. BIPES Project IDE; 2022. Available from: <http://bipes.net.br/beta2/ui/#7tmkuc> [Accessed 13 September 2022].
- [29] Arouca R, Marques J, Silva CA. Tech Details. How BIPES works [Online]. BIPES:Dev; 2021. Available from: <http://bipes.net.br/docs/get-started/tech-details.html> [Accessed 13 September 2022].
- [30] George DP. MicroPython [Online]. MicroPython; 2022. Available from: <https://micropython.org/> [Accessed 13 September 2022].
- [31] Supported platforms & architectures [Online]. GitHub; 2022. Available from: <https://github.com/micropython/micropython> [Accessed 13 September 2022].
- [32] George DP, Sokolovsky P, and contributors. zsensor — Zephyr sensor bindings [Online]. The MicroPython Documentation; Sep. 2022. Available from: <https://docs.micropython.org/en/latest/library/zephyr.zsensor.html#zsensor-sensor> [Accessed 13 September 2022].
- [33] Arouca R, Marques J, Silva CA. Creating new blocks [Online]. BIPES:Dev; 2021. Available from: <http://bipes.net.br/docs/get-started/create-block.html> [Accessed 13 September 2022].